Learning to See the Physical World

by

Jiajun Wu

B.Eng., B.Ec., Tsinghua University (2014)
S.M., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2020

© Massachusetts Institute of Technology 2020. All rights reserved.

Signature redacted

Author . . . .
Department of Electrical Engineering and Computer Science
September 30, 2019

Signature redacted

Certified by . . . .
William T. Freeman
Thomas and Gerd Perkins Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Certified by . . . .
Joshua B. Tenenbaum
Professor of Computational Cognitive Science
Thesis Supervisor

Signature redacted

Accepted by . . . .
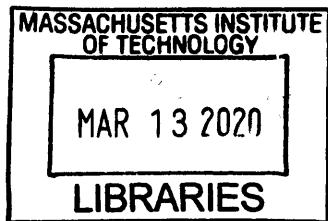Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Learning to See the Physical World

by

Jiajun Wu

Submitted to the Department of Electrical Engineering and Computer Science
on September 30, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Human intelligence is beyond pattern recognition. From a single image, we are able to explain what we see, reconstruct the scene in 3D, predict what's going to happen, and plan our actions accordingly. Artificial intelligence, in particular deep learning, still falls short in some preeminent aspects when compared with human intelligence, despite its phenomenal development in the past decade: they in general tackle specific problems, require large amounts of training data, and easily break when generalizing to new tasks or environments.

In this dissertation, we study the problem of physical scene understanding—building versatile, data-efficient, and generalizable machines that learn to see, reason about, and interact with the physical world. The core idea is to exploit the generic, causal structure behind the world, including knowledge from computer graphics, physics, and language, in the form of approximate simulation engines, and to integrate them with deep learning. Here, learning plays a multifaceted role: models may learn to invert simulation engines for efficient inference; they may also learn to approximate or augment simulation engines for more powerful forward simulation.

This dissertation consists of three parts, where we investigate the use of such a hybrid model for perception, dynamics modeling, and cognitive reasoning, respectively. In Part I, we use learning in conjunction with graphics engines to build an object-centered scene representation for object shape, pose, and texture. In Part II, in addition to graphics engines, we pair learning with physics engines to simultaneously infer physical object properties. We also explore learning approximate simulation engines for better flexibility and expressiveness. In Part III, we leverage and extend the models introduced in Parts I and II for concept discovery and cognitive reasoning by looping in a program execution engine. The enhanced models discover program-like structures in objects and scenes and, in turn, exploit them for downstream tasks such as visual question answering and scene manipulation.

Thesis Supervisor: William T. Freeman
Title: Thomas and Gerd Perkins Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Joshua B. Tenenbaum
Title: Professor of Computational Cognitive Science

Acknowledgments

My first and deepest thanks go to my advisors, Bill Freeman and Josh Tenenbaum. Bill has always influenced me with his curiosity, passion, and persistence for science. But what I have learned from Bill is much beyond science: most importantly, a unified, consistent, powerful, and effective set of principles that guide you through all the hills and valleys of life. Josh is the best advisor I can ever imagine. One can think of many factors that make a good advisor: amazing depth and breadth of knowledge, original and creative research ideas, encouraging, constructive, and insightful conversations, extensive connections within and outside academia, strong support in job search, etc. Magically, Josh has all those, to the best extent possible. Bill and Josh are the best role models for me in both research and life. I wish, after I start advising students myself, one day I would become half as good through decades of learning and practice.

I appreciate Leslie Kaelbling and Fei-Fei Li for serving on my thesis committee. I have wanted to work with Leslie since I was her teaching assistant during my second year of graduate school. I was glad that this later came true. Fei-Fei has long inspired me through her contributions to computer vision, AI, and beyond. It is my honor to have her on my committee, and I hope we will work more together at Stanford.

Many great minds have guided me and led me through this journey. Zhuowen Tu introduced me into the world of AI and vision when I was a rising sophomore, and has been my mentor and friend since then. Antonio Torralba mentored me throughout my PhD. I have always been amazed by his insightful and creative thoughts. I was fortunate enough to work with Josh McDermott, from whom I learned all I know about human and machine auditory perception. In Summer 2016, I interned with Pushmeet Kohli at Microsoft Research; this turned out to be the beginning of our multi-year, fruitful collaborations. I also thank Liz Spelke, whom I got to know at the beginning of my graduate study and eventually worked together when I am finishing up. I wish I had started working with Liz earlier and I hope we will collaborate more in the future.

I studied under Andrew Chi-Chih Yao and Jian Li at Tsinghua University, Yan Xu and Eric Chang at Microsoft Research Asia, Kai Yu and Yinan Yu at Baidu Research, and Yuandong Tian at Facebook AI Research. Polina Golland and Randall Davis served on my research qualifying exam committee. I thank them for their patience, kindness, and help.

Many colleagues have also played a mentorship role throughout my study. My appreciation to them cannot be understated. Tianfan Xue demonstrated to me the beauty of low-level vision as well as the importance of communication skills in teamwork. In my first year, Joseph Lim connected me with my senior collaborators and, since then, has been a strong supporter of my research career. Ilker Yildirim

THIS PAGE INTENTIONALLY LEFT BLANK

To Lujun

Contents

III   Reasoning: Learning with Program Executors   159

8   Learning to Discover Concepts from Images and Language   161

9   Learning to Organize Concepts into 3D Shape Programs   183

List of Figures

List of Tables

Chapter 1

Introduction

Human intelligence is rich and flexible. From a quick glance, we effortlessly recognize the 3D geometry and texture of objects within the scene, reason about their relation, and when they move, track and predict their trajectories (Figure 1-1A). Stacking blocks, picking up fruits—we also plan and interact with scenes and objects in many ways.

The goal of the research presented in this dissertation is to build machines that see, interact with, and reason about the physical world just like humans. This problem, which we call **physical scene understanding**, involves three key topics that bridge research in computer science, artificial intelligence (AI), robotics, cognitive science, and neuroscience:

- **Perception** (Figure 1-1B): How can structured, physical object and scene representations arise from raw, multi-modal sensory input (e.g., videos, sound, tactile signals)?

- **Physics** (Figure 1-1C): How can we build dynamics models that quickly adapt to complex, stochastic real-world scenarios, and how can they contribute to planning and motor control? Modeling physical interactions helps robots to build bridges from a single image and to play challenging games such as Jenga.

- **Reasoning** (Figure 1-1D): How can physical models integrate structured, often symbolic, priors such as symmetry and repetition, and use them for commonsense reasoning?

Physical scene understanding is challenging, because it requires a holistic interpretation of scenes and objects, including their 3D geometry, physics, functionality, and modes of interactions, beyond the scope of a single discipline such as computer vision. Structured priors and representations of the physical world are essential: we need proper representations and learning paradigms to build data-efficient, flexible, and generalizable intelligent systems that understand physical scenes.

1
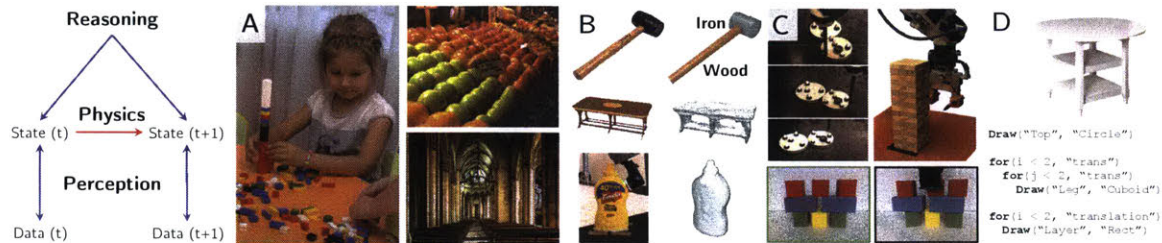
Figure 1-1: **Physical scene understanding** involves **perception**, building physical object representations from multi-modal data, **(II) physical interaction**, capturing scene dynamics for planning and control, and **(III) commonsense reasoning**, understanding high-level structured priors in objects and scenes.

Despite great advances in computer vision, deep learning, and other AI tools in the past decade, areas where they are most successful, such as image classification, are still about pattern recognition with abundant training data. We are still far from having machines with human-level scene understanding: machines that can understand the world as richly, as quickly, as flexibly, and as robustly as humans do.

In this dissertation, we integrate approaches to representation, inference, learning, and generation that have not been combined before. We adopt new AI paradigms that combine the power of deep networks for pattern recognition, simulation engines for structure and generality, symbolic languages for knowledge representation and abstraction, and causal and counterfactual reasoning in generative models for explainability, imagination, and planning. Building upon it, we develop novel approaches for key problems in perception, dynamics modeling, and cognitive reasoning.

1.1 Approach: Integrating Learning with Simulation Engines

My approach to constructing representations of the physical world is to integrate bottom-up recognition models, deep networks, and efficient inference algorithms, with top-down, structured graphical models, simulation engines, and probabilistic programs. In my research, I develop and extend techniques in these areas (e.g., proposing new deep networks and physical simulators); I further explore innovative ways to combine them, building upon studies across vision, learning, graphics, and robotics. I believe that only by exploiting knowledge from all these areas, may we build machines that have human-like, physical understanding of complex, real-world scenes.

The approach presented in this dissertation is also highly interdisciplinary: we build computational models with inspiration from human cognition, developmental psychology, neuroscience, robotics, and computational linguistics; we also explore how these models can in turn assist in solving tasks in these fields.

Figure 1-2: **Learning to see shapes, texture, and physics. A.** Reconstructing 3D shapes from a single color image via 2.5D sketches [Wu et al., 2017c, 2018b, Zhang et al., 2018b]. **B.** Generative modeling of 3D shapes and 2D images via a disentangled representation for object geometry, viewpoint, and texture [Wu et al., 2016c, Zhu et al., 2018b]. **C.** 3D-aware representations for objects and scenes [Wu et al., 2017b, Yao et al., 2018]. **D.** Part-based object representations for its geometry and physics [Liu et al., 2018b, Wu et al., 2016a, 2017a, 2015a].

1.1.1 Perception: Learning with Graphics Engines

Motivated by human perception—rich, complex, generalizable, learning much from little—my research on perception has been centered on building structured, object-based models to characterize the appearance and physics of daily objects. These models integrate bottom-up deep recognition models with top-down simulation engines; they learn by perceiving and explaining the physical world just like humans.

Seeing shapes and texture. Drawing inspiration from human perception and computer graphics, I have built object appearance model that learns to perceive object shape and texture from raw visual observations, and to synthesize new shapes and images. The core object representation builds upon a coherent understanding of its intrinsic properties such as shape, surface, and texture, and its extrinsic properties such as pose.

My research covers various components of the appearance model. On bottom-up recognition, I have developed a general pipeline for 3D shape reconstruction from a single color image [Wu et al., 2017c, 2018a] via modeling *intrinsic images*—depth, surface normals, and reflectance maps [Janner et al., 2017] (Figure 1-2A). My research is inspired by the classic research on multi-stage human visual perception [Marr, 1982], and has been extended to integrating learned priors of 3D shapes (i.e., 'what shapes look like?') for more realistic 3D reconstructions [Wu et al., 2018b], and to tackling cases where the object in the image is not from the training categories [Zhang et al., 2018b].

Complementary to these bottom-up recognition models, I have also explored learning top-down graphics engines directly. I proposed 3D generative adversarial networks, first applying generative-adversarial learning to 3D shapes for unconditional shape synthesis [Wu et al., 2016c]. We have later extended the model as visual object networks [Zhu et al., 2018b], which synthesize object shape and texture simultaneously, enforcing various consistencies with a distributed representation for object shape, 2.5D sketches, viewpoint, and texture (Figure 1-2B). We have generalized our models to scenes [Wu et al., 2017b, Yao et al., 2018], recovering structured scene represen-

tations that not only capture object shape and texture, but enable 3D-aware scene manipulations (Figure 1-2C).

Seeing physics. Beyond object appearance, intuition of object physics assists humans in scene understanding [Battaglia et al., 2013]. I have developed computational models that learn to infer object physics directly from visual observations [Wu et al., 2016a, 2015a]. My research on visual intuitive physics is the first in the computer vision community, and has since then inspired many researchers to follow up on the methods [Fragkiadaki et al., 2016, Mottaghi et al., 2016a].

The Galileo model [Wu et al., 2015a] marries a physics engine with deep recognition nets to infer physical object properties (e.g., mass, friction). With an embedded physical simulator, the Galileo model discovers physical properties simply by watching objects move in unlabeled videos; it also predicts how they interact based on the inferred physical properties. The model was tested on a real-world video dataset, Physics 101 [Wu et al., 2016a], of 101 objects interacting in various physical events.

I have also worked on integrating geometry and physics perception (Figure 1-2D), with two primary results as "physical primitive decomposition" (PPD) [Liu et al., 2018b] and "visual de-animation" (VDA) [Wu et al., 2017a]. In PPD, we decompose an object into parts with distinct geometry and physics, by learning to explain both the object's appearance and its behaviors in physical events; in VDA, our model learns to jointly infer physical world states and to simulate scene dynamics, integrating both a physics engine and a graphics engine. Our recent work has extended these models to complex indoor scenes, exploiting stability for more accurate 3D scene parsing [Du et al., 2018].

Multi-modal perception. Humans see, hear, and feel, perceiving the world through fusing multi-sensory signals. These signals play complementary roles: we see object shape and texture through vision, hear their material through sound, and feel their surface details through touch. In computer science, however, most recognition models and simulation engines have primarily focused on visual data. Based on techniques from the graphics community, I have been building generative audio-visual engines and using them for cross-modal perception [Zhang et al., 2017b,c]: how much do we know about objects from videos, and how much from audio? Beyond auditory signals, our recent work has also explored integrating tactile signals with vision for better shape perception and reconstruction [Wang et al., 2018b].

1.1.2 Dynamics: Learning with Physics Engines

Beyond learning object-centric models from raw observations by inverting simulation engines, my research also includes learning to approximate simulation engines (forward models) themselves. I have explored building physical models in various

4
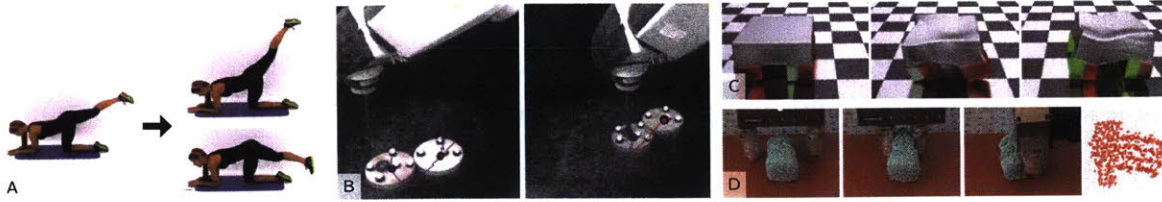
Figure 1-3: **Physical models for future prediction and control. A.** Modeling visual dynamics allows us to generate multiple possible future frames from a single image [Xu et al., 2019, Xue et al., 2016]. **B.** We have developed a hybrid model that captures object-based dynamics by integrating analytical models and neural nets. It assists the robot in accomplishing a highly underactuated task: pushing the right disk to the target (green) by only interacting with the left disk [Ajay et al., 2019, 2018]. **C. D.** Particle-based dynamics models support controlling soft robots [Hu et al., 2019] and manipulating deformable objects and liquids [Li et al., 2019b,c].

forms—image-based, object-based, and particle-based; analytical, neural, and hybrid—and have demonstrated their power in challenging, highly underactuated control tasks (Figure 1-3).

Compared with off-the-shelf simulators, a learned dynamics simulator flexibly adapts to novel environments and captures the stochasticity in scene dynamics. Our visual dynamics model demonstrates this in the pixel domain, where it learns to synthesize multiple possible future frames from a single color image by automatically discovering independent movable parts and their motion distributions [Xue et al., 2016] (Figure 1-3A). We have later extended the model to additionally capture the hierarchical structure among object parts [Xu et al., 2019].

Modeling dynamics directly in the pixel space is universal but challenging due to the intricate interplay between physics and graphics; an alternative is to separate perception from dynamics modeling, and learn dynamics from object states. Our recent work along this line has shown that a model that learns to approximate object dynamics can be useful for planning [Janner et al., 2019], generalize to scenarios where only partial observations are available [Li et al., 2019c], and discover physical object properties without supervision [Zheng et al., 2018]. We have further extended our model to particle-based representations, so that it can characterize the dynamics of soft robots [Hu et al., 2019] (Figure 1-3C) and of scenes with complex interactions among rigid bodies, deformable shapes, and fluids [Li et al., 2019b] (Figure 1-3D).

We have also explored the idea of learning a hybrid dynamics model, augmenting analytical physics engines with neural dynamics models [Ajay et al., 2018] (Figure 1-3B). Such a hybrid system achieves the best of both worlds: it performs better, captures uncertainty in data, learns efficiently from limited annotations, and generalizes to novel shapes and materials.

These dynamics models can be used in various control tasks: they help to solve highly underactuated control problems (pushing disk A, which in turn pushes disk

B to the target position) [Ajay et al., 2019], to control and co-design soft robots [Hu et al., 2019], to manipulate fluids and rigid bodies on a Kuka robot [Li et al., 2019b], and to interact and play games such as Jenga that involve complex frictional micro-interactions [Fazeli et al., 2019].

1.1.3 Reasoning: Learning with Program Executors

The physical world is rich but structured: natural objects and scenes are compositional (scene are made of objects which, in turn, are made of parts); they often have program-like structure (objects are symmetric and made of evenly spaced repetitive parts). I have been exploring ways to bridge structured, often symbolic, priors into powerful deep recognition models. Beyond perception models that invert simulation engines, and physical dynamics models that approximate simulation engines themselves, we move one step further to learn the representation priors these simulation engines have—why they represent the world in the way they currently are.

A test of these neuro-symbolic representations is how well they support solving various reasoning tasks such as analogy making and question answering. Our recent work demonstrated that, when combined with deep visual perception modules, a symbolic reasoning system achieves impressive performance on visual reasoning benchmarks [Yi et al., 2018], outperforming end-to-end trained neural models. We have also extended it to jointly learn visual concepts (e.g., colors, shapes) and their correspondence with words from natural supervision (question-answer pairs) via curriculum learning [Mao et al., 2019a], without human annotations.

Beyond static images, we've integrated neuro-symbolic representations with learned object-based dynamics models for temporal and casual reasoning on videos. On our newly proposed video reasoning benchmark, our model performs significantly better in answering all four types of questions: descriptive (e.g., 'what color'), explanatory ('what's responsible for'), predictive ('what will happen next'), and counterfactual ('what if').

Symbolic structure learning is closely coupled with program synthesis. In particular, our recent work has made progress on the problem of inferring programs as a novel representation for shapes [Tian et al., 2019] and scenes [Liu et al., 2019]. This marks the start of our exploration in wiring highly structured, hierarchical priors into learning representations for physical scene understanding.

1.2 Dissertation Structure

This dissertation consists of three parts, exploring the integration of machine learning with simulation engines in three distinctive but deeply connected domains: perception, dynamics modeling, and cognitive reasoning.

Part I is about perception—how we build vision systems by connecting learning with graphics engines. We demonstrate the generality and flexibility of our approach by working with multiple graphics engines of distinctive features.

In Chapter 2, we start with a simplified graphics engine that projects sparse 3D keypoints into 2D, and present a learning system that infers 3D object skeleton from a single image by inverting the engine. This chapter was previously published as Wu et al. [2016b, 2018a].

In Chapter 3, we extend the model to work with dense 3D-to-2D projection, so that the model learns to infer dense 3D object shape instead. This chapter is primarily based on Zhang et al. [2018b] and also includes materials from Wu et al. [2016c, 2017c, 2018b], Sun et al. [2018b].

In Chapter 4, beyond single objects, we further extend the model to work with graphics engines that handle multi-object scenes. This chapter is primarily based on Wu et al. [2017b] and also includes materials from Yao et al. [2018].

Part II presents results on dynamics modeling—after seeing objects, the computational model should also be able to predict how they will move. The main approach is to integrate learning with another type of simulators—physics engines.

In Chapter 5, we demonstrate how a learning system can infer physical object properties, such as mass and friction, by having a physics engine in the loop. This chapter is primarily based on Wu et al. [2015a] and also includes materials from Wu et al. [2016a].

In Chapter 6, we revisit graphics engines—the subject of Part I, and present a novel model that learns to see objects' visual and physical properties by working with both physics and graphics engines. This chapter is primarily based on Wu et al. [2017a] and also includes materials from Liu et al. [2018b], Janner et al. [2019].

In Chapter 7, we present a model that learns not only to use simulators, but to approximate graphics and physics engines themselves. Compared with off-the-shelf simulators, learned models can bring additional features such as stochasticity. This chapter is primarily based on Xue et al. [2019, 2016] and also includes materials from Xu et al. [2019].

Part III includes contributions to high-level cognitive reasoning, where we integrate learning with program execution engines. We also draw connections to various models introduced in Part I and Part II.

In Chapter 8, we introduce a neuro-symbolic reasoning model that simultaneously learns visual concepts and words from minimally-annotated images and language. Here, instead of learning to infer the attributes given by simulators, such as shape or mass, we aim to discover attributes themselves. This chapter is primarily based on Mao et al. [2019a] and also includes materials from Yi et al. [2018].

In Chapter 9, building upon entry-level visual concepts (e.g., a red cylinder), we learn to compose them into complex object shapes. Our approach is based on what we introduced in Part I and Part II: we use learning to simultaneously invert and approximate a simulation engine, which, this time, is the shape program executor. This chapter was previously published as Tian et al. [2019].

In Chapter 10, we further extend the model so that it learns not only to compose primitives into object shapes, but to automatically discover program-like regularities in natural images, and to exploit them for image manipulation. This chapter is primarily based on Mao et al. [2019b] and also includes materials from Liu et al. [2019].

We conclude and discuss future directions in Chapter 11.

Part I

Perception: Learning with Graphics Engines

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Learning with a Graphics Engine for Sparse Keypoints

We begin our journey by looking into the problem of 3D perception. Understanding 3D object structure from a single image is an important but challenging task in computer vision, mostly due to the lack of 3D object annotations to real images. Previous research tackled this problem by either searching for a 3D shape that best explains 2D annotations, or training purely on synthetic data with ground truth 3D information.

In this chapter, we propose 3D INterpreter Networks (3D-INN), an end-to-end trainable framework that sequentially estimates 2D keypoint heatmaps and 3D object skeletons and poses. Our system learns from both 2D-annotated real images and synthetic 3D data. This is made possible mainly by two technical innovations. First, heatmaps of 2D keypoints serve as an intermediate representation to connect real and synthetic data. 3D-INN is trained on real images to estimate 2D keypoint heatmaps from an input image; it then predicts 3D object structure from heatmaps using knowledge learned from synthetic 3D shapes. By doing so, 3D-INN benefits from the variation and abundance of synthetic 3D objects, without suffering from the domain difference between real and synthesized images, often due to imperfect rendering. Second, we incorporate a simplified graphics engine, a differentiable, 3D-to-2D Projection Layer, mapping estimated 3D structure back to 2D. During training, it ensures 3D-INN to predict 3D structure whose projection is consistent with the 2D annotations to real images.

Experiments show that the proposed system performs well on both 2D keypoint estimation and 3D structure recovery. We also demonstrate that the recovered 3D information has wide vision applications, such as image retrieval.

This chapter was previously published as Wu et al. [2016b, 2018a]. Tianfan Xue, in particular, contributed significantly to the materials presented in this chapter.

11

Figure 2-1: **Problem setup: 3D skeleton recovery.** Given an image of a chair, we are interested in its intrinsic properties such as its height, leg length, and seat width, and extrinsic properties such as its pose.

2.1 Introduction

Deep networks have achieved impressive performance on image recognition [Russakovsky et al., 2015]. Nonetheless, for any visual system to parse objects in the real world, it needs not only to assign category labels to objects, but also to interpret their intra-class variation. Figure 2-1 shows an example: for a chair, we are interested in its intrinsic properties such as its *style*, *height*, leg *length*, and seat *width*, and extrinsic properties such as its *pose*.

Our goal is to recover these object properties from a single image by jointly estimating the object's 3D wireframe and the viewpoint. We choose to use a single image as input primarily for three reasons. First, this problem has great scientific value: humans can easily recognize 3D structure from a single image, and we want to build machines that replicate such an ability. Second, by starting with a single image, instead of multi-view images or videos, our model can be directly applied to images taken in the wild, e.g., from the web, and can cluster them based on their structure or pose. This offers extensive practical usages: social media and e-commerce companies can better understand and analyze user-uploaded data, and household robots can efficiently interact with objects after modeling them in 3D. Third, using a single image as input enables online inference: for moving objects like cars, the system can reconstruct their geometry and viewpoint on the fly. This is crucial for real-time applications such as autonomous driving.

We represent an object via a 3D skeleton [Torresani et al., 2003] (Figure 2-2c), instead of a 3D mesh or a depth map [Dosovitskiy et al., 2015, Aubry et al., 2014, Prasad et al., 2010, Kar et al., 2015, Su et al., 2014, Vicente et al., 2014, Huang et al., 2015], because skeletons are simpler and preserve the structural properties that we are interested in. We refer readers to Section 2.6 for detailed discussions on our design choices. We assume that a 3D skeleton consists of keypoints and the connections
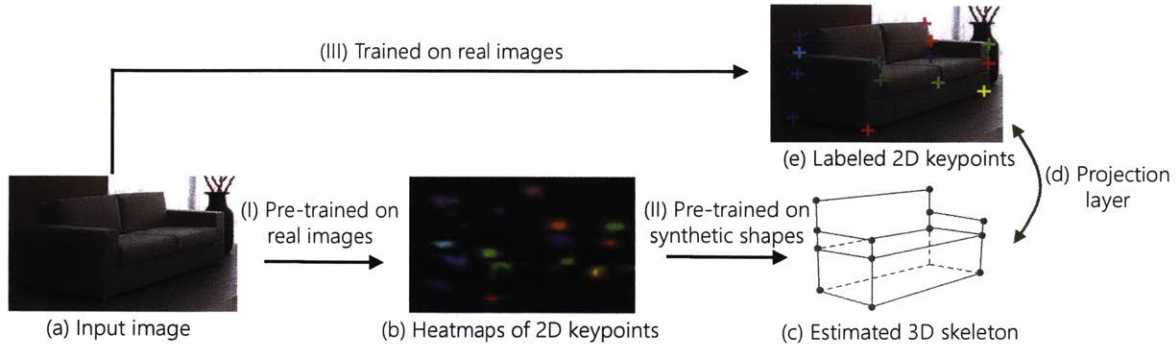
Figure 2-2: **Overview of 3D INterpreter Networks (3D-INN)**. For an image (a) with a category-level label (sofa), the system first estimates its 2D keypoint heatmaps (b), and then recovers the 3D skeleton of the object (c). During training, through the projection layer (d), it also enforces the consistency between annotated 2D keypoint locations (e) and projected 2D locations of estimated 3D keypoints.

between them, and manually pre-define the skeleton model for each object category (e.g., chair, sofa, and car). Then, our task is to estimate 3D keypoint locations of an object from a single RGB image.

The main challenge of single image 3D estimation is the difficulty in obtaining training images with ground truth 3D structure, as manually annotating 3D object structure in real images is labor-intensive and often inaccurate. Previous methods tackled this problem mostly in two ways. One is to directly solve for a 3D skeleton from estimated 2D keypoint locations by minimizing its reprojection error [Zhou et al., 2015], without any 3D annotations. Most algorithms in this category do not encode 3D shape priors within the model, and thus they are not robust to inaccurate keypoint estimation, as shown in experiments in Section 2.4. The other is to train on synthetically rendered images of 3D objects [Li et al., 2015, Su et al., 2015]; in synthetic data, complete 3D structure is available. But the statistics of synthesized images are often different from those of real images, due to changes in lighting, occlusion, and shape details. This makes it hard for models trained only on synthetic data to generalize well to real images.

In this chapter, we propose 3D INterpreter Networks (3D-INN), an end-to-end trainable framework for 3D skeleton and viewpoint estimation. In contrast to prior art, our model learns from both 2D-labeled real images and synthetic 3D objects. This is made possible by two major innovations. First, we use heatmaps of 2D keypoints as an intermediate representation to connect real and synthetic data. 3D-INN is trained on real images to estimate 2D keypoint heatmaps from an input image (Figure 2-2-I); it then learns from synthetic data to estimate 3D structure from heatmaps (Figure 2-2-II). By doing so, 3D-INN benefits from the variations in abundant synthetic 3D objects, without suffering from the domain difference between real and synthesized data.

Second, we introduce a *Projection Layer*, a rendering function that calculates

13

projected 2D keypoint positions given a 3D skeleton and camera parameters. We attach it at the end of the framework (Figure 2-2d). This enables 3D-INN to predict 3D structural parameters that minimize errors in 2D on real images with keypoint labels, without requiring 3D object annotations. Our training paradigm therefore consists of three steps: we first train the keypoint estimation component (Figure 2-2-I) on 2D-annotated real images; we then train the 3D interpreter (Figure 2-2-II) on synthetic 3D data; we finally fine-tune the entire framework end-to-end with the projection layer (Figure 2-2-III).

Both innovations are essential for the system to exploit the complementary richness of real 2D data and synthetic 3D data. Without the heatmap representation and synthetic 3D data, the system can still be trained on real images with 2D annotations, using the projection layer. But it does not perform well: because of intrinsic ambiguities in 2D-to-3D mapping, the algorithm recovers unnatural 3D geometries, though their projections may perfectly align with 2D annotations, as explored in Lowe [1987]. Synthetic 3D shapes help the network resolve this ambiguity by learning prior knowledge of "plausible shapes". Without the projection layer, the system becomes two separately trained networks: one trained on real images to estimate 2D keypoint heatmaps, and the other trained on synthetic data to estimate 3D structure. As shown in Section 2.4, the 3D predictions in this case are not as accurate due to the domain adaptation issue.

Several experiments demonstrate the effectiveness of 3D-INN. First, the proposed network achieves good performance on various keypoint localization datasets, including FLIC [Sapp and Taskar, 2013] for human bodies, CUB-200-2011 [Wah et al., 2011] for birds, and our new dataset, Keypoint-5, for furniture. We then evaluate our network on IKEA [Lim et al., 2013], a dataset with ground truth 3D object structure and viewpoints. We augmented the original IKEA dataset with additional 2D keypoint labels. On 3D structure estimation, 3D-INN shows its advantage over an optimization-based method [Zhou et al., 2015] when keypoint estimation is imperfect. On 3D viewpoint estimation, it also performs better than the state of the art [Su et al., 2015]. We further evaluate 3D-INN, in combination with an object detection framework, R-CNN [Girshick et al., 2014], on the PASCAL 3D+ benchmark [Xiang et al., 2014] for joint detection and viewpoint estimation. 3D-INN also achieves results comparable to the state of the art [Su et al., 2015, Tulsiani and Malik, 2015]. At last, we show that 3D-INN has wide vision applications including 3D object retrieval.

Our contributions in this chapter are three-fold. First, we introduce 3D INterpreter Networks (3D-INN); by incorporating 2D keypoint heatmaps to connect real and synthetic worlds, we strengthen the generalization ability of the network. Second, we propose a projection layer, so that 3D-INN can be trained to predict 3D structural parameters using only 2D-annotated images. Third, our model achieves state-of-the-art

performance on the estimation of 2D keypoints, 3D structure, and viewpoint.

2.2 Related work

Single image 3D reconstruction. Previous 3D reconstruction methods mainly modeled objects using either dense representations such as depth or meshes, or sparse representations such as skeletons or pictorial structure. Depth-/mesh-based models can recover detailed 3D object structure from a single image, either by adapting existing 3D models from a database [Aubry et al., 2014, Satkin et al., 2012, Su et al., 2014, Huang et al., 2015, Zeng et al., 2017, Wu et al., 2016c, Hu and Zhu, 2015, Bansal and Russell, 2016, Shrivastava and Gupta, 2013, Choy et al., 2016], or by inferring from its detected 2D silhouette [Kar et al., 2015, Soltani et al., 2017, Vicente et al., 2014, Prasad et al., 2010, Wu et al., 2017c].

We choose to use a skeleton-based representation, exploiting the power of abstraction. The skeleton model can capture geometric changes of articulated objects [Torresani et al., 2003, Yasin et al., 2016, Akhter and Black, 2015], like a human body or the base of a swivel chair. Typically, researchers recovered a 3D skeleton from a single image by minimizing its projection error on the 2D image plane [Lowe, 1987, Leclerc and Fischler, 1992, Hejrati and Ramanan, 2014, Xue et al., 2012, Ramakrishna et al., 2012, Zia et al., 2013]. Recent work in this line [Akhter and Black, 2015, Zhou et al., 2015] demonstrated state-of-the-art performance. In contrast to them, we propose to use neural networks to predict a 3D object skeleton from its 2D keypoints, which is more robust to imperfect detection results and can be jointly learned with keypoint estimators.

Our work also connects to the traditional field of vision as inverse graphics [Hinton and Ghahramani, 1997, Kulkarni et al., 2015b] and analysis by synthesis [Yuille and Kersten, 2006, Kulkarni et al., 2015a, Bever and Poeppel, 2010, Wu et al., 2015a], as we use neural nets to decode latent 3D structure from images, and use a projection layer for rendering. Their approaches often required supervision for the inferred representations or made over-simplified assumptions of background and occlusion in images. Our 3D-INN learns 3D representation without using 3D supervision, and generalizes to real images well.

2D keypoint estimation. Another line of related work is 2D keypoint estimation. During the past decade, researchers have made significant progress in estimating keypoints on humans [Sapp and Taskar, 2013, Yang and Ramanan, 2011] and other objects [Wah et al., 2011, Shih et al., 2015]. Recently, there have been several attempts to apply convolutional neural networks to human keypoint estimation [Toshev and Szegedy, 2014, Tompson et al., 2015, Carreira et al., 2016, Newell et al., 2016], which all achieved significant improvement. 3D-INN uses 2D keypoints as an intermediate

15

representation, and aims to recover a 3D skeleton from them.

3D viewpoint estimation. 3D viewpoint estimation seeks to estimate the 3D orientation of an object from a single image [Xiang et al., 2014]. Some previous methods formulated it as a classification or regression problem, and aimed to directly estimate the viewpoint from an image [Fidler et al., 2012, Su et al., 2015]. Others proposed to estimate 3D viewpoint from detected 2D keypoints or edges in the image [Zia et al., 2013, Lim et al., 2014, Tulsiani and Malik, 2015]. While the main focus of our work is to estimate 3D object structure, our method can also predict the corresponding 3D viewpoint.

Training with synthetic data. Synthetic data are often used to augment the training set [Su et al., 2014, Shakhnarovich et al., 2003], especially when ground truth labels of real images are hard to obtain. This technique has found wide applications in computer vision. To name a few, Sun and Saenko [2014] and Zhou et al. [2016a] combined real and synthetic data for object detection and matching, respectively. Huang et al. [2015] analyzed the invariance of convolutional neural networks using synthetic images. Dosovitskiy et al. [2015] trained a neural network for image synthesis using synthetic images. McCormac et al. [2017] rendered images for indoor scene understanding. Su et al. [2014] attempted to train a 3D viewpoint estimator on both real and synthetic images.

We combine real 2D-annotated images and synthetic 3D data for training 3D-INN to recover a 3D skeleton. We use heatmaps of 2D keypoints, instead of (often imperfectly) rendered images, from synthetic 3D data, so that our algorithm has better generalization ability as the effects of imperfect rendering are minimized. Yasin et al. [2016] also proposed to use both 2D and 3D data for training, but they used keypoint locations, instead of heatmaps, as the intermediate representation that connects 2D and 3D. While their focus is on estimating human poses, we study the problem of recovering the 3D structure of furniture and cars.

2.3 Method

We design a deep convolutional network to recover 3D object structure. The input to the network is a single image with the object of interest at its center, which can be obtained by state-of-the-art object detectors. The output of the network is a 3D object skeleton, including its 2D keypoint locations, 3D structural parameters, and 3D pose (see Figure 2-4). In the following sections, we will describe our 3D skeleton representation and camera model (Section 2.3.1), network design (Section 2.3.2), and training strategy (Section 2.3.3).

(a) Base shapes for chairs. B_1 is the mean shape of chairs, and the others characterize possible variations of the structure.

(b) Base shapes for cars
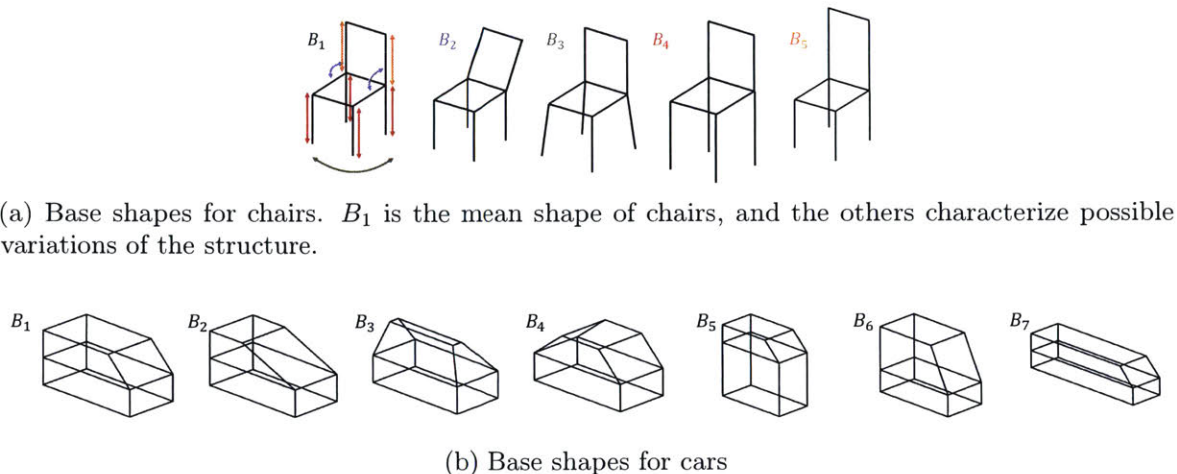
Figure 2-3: **Our skeleton model and base shapes** for chairs (a) and cars (b)

2.3.1 3D Skeleton Representation

We use skeletons as our 3D object representation. A skeleton consists of a set of keypoints as well as their connections. For each object category, we manually design a 3D skeleton characterizing its abstract 3D geometry.

There exist intrinsic ambiguities in recovering 3D keypoint locations from a single 2D image. We resolve this issue by assuming that objects can only have constrained deformations [Torresani et al., 2003]. For example, chairs may have various leg lengths, but for a single chair, its four legs are typically of equal length. We model these constraints by formulating 3D keypoint locations as a weighted sum of a set of base shapes [Kar et al., 2015]. The first base shape is the mean shape of all objects within the category, and the rest define possible deformations and intra-class variations. Figure 2-3a shows our skeleton representation for chairs: the first is the mean shape of chairs, the second controls how the back bends, and the last two are for legs. Figure 2-3b shows base shapes for cars. The weight for each base shape determines how strong the deformation is, and we denote these weights as the *structural parameters* of an object.

Formally, let $\mathbf{Y} \in \mathbb{R}^{3 \times N}$ be a matrix of 3D coordinates of all N keypoints. Our assumption is that the 3D keypoint locations are a weighted sum of base shapes $B_k \in \mathbb{R}^{3 \times N}$, or

$$\mathbf{Y} = \sum_{k=1}^{K} \alpha_k B_k, \tag{2.1}$$

where $\{\alpha_k\}$ is the set of structural parameters of this object, and K is the number of base shapes.

Further, let $\mathbf{X} \in \mathbb{R}^{2 \times N}$ be the corresponding 2D coordinates. Then the relationship

between the observed 2D coordinates $\mathbf{X}$ and the structural parameters $\{\alpha_k\}$ is

$$\mathbf{X} = P(R\mathbf{Y} + T) = P(R\sum_{k=1}^{K} \alpha_k B_k + T), \qquad (2.2)$$

where $R \in \mathbb{R}^{3\times3}$ (rotation) and $T \in \mathbb{R}^3$ (translation) are the external parameters of the camera, and $P \in \mathbb{R}^{3\times4}$ is the camera projection matrix which we will discuss soon.

Therefore, to recover the 3D structural information of an object in a 2D image, we only need to estimate its structural parameters ($\{\alpha_k\}$) and the external viewpoint parameters (R, T, and f). We supply the detailed camera model below. In Section 2.3.2 and Section 2.3.3, we discuss how we design a neural network for this task, and how it can be jointly trained with real 2D images and synthetic 3D objects.

Camera model. We use perspective projection in order to model the perspective distortion in 2D images. We assume that the principal point is at the origin, all pixels are square, and there is no axis skew. In this way, we only need to estimate the focal length in the camera projection matrix P.

For the ease of inference in neural networks, we rewrite the normal perspective projection as follows. Let $x_i \in \mathbb{R}^2$ be a column vector of the 2D coordinates of the i-th keypoint and y_i be the corresponding 3D coordinates to be recovered. We assume that the camera center is at $(0, 0, f)$ instead of the origin. The perspective projection is written as (using projective coordinates):

$$\begin{pmatrix} x_i^1 \\ x_i^2 \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} y_i^1 \\ y_i^2 \\ y_i^3 + f \\ 1 \end{pmatrix}, \qquad (2.3)$$

where f is the focal length, x_i^1 and x_i^2 are the x- and y-components of x_i, and y_i^1, y_i^2, and y_i^3 are x-, y-, and z-components of y_i.

When $f^{-1} \to 0$, Equation 2.3 converges to the formulation of parallel projection. To see that, based on Equation 2.3, we get the Euclidean coordinates of the 2D projection as (we abuse the notation of x_i^1 and x_i^2 for both Euclidean coordinates and projective coordinates)

$$\begin{cases} x_i^1 = \dfrac{fy_i^1}{y^3 + f} = \dfrac{y_i^1}{f^{-1}y^3 + 1}, \\ x_i^2 = \dfrac{fy_i^2}{y^3 + f} = \dfrac{y_i^2}{f^{-1}y^3 + 1}. \end{cases} \qquad (2.4)$$

18
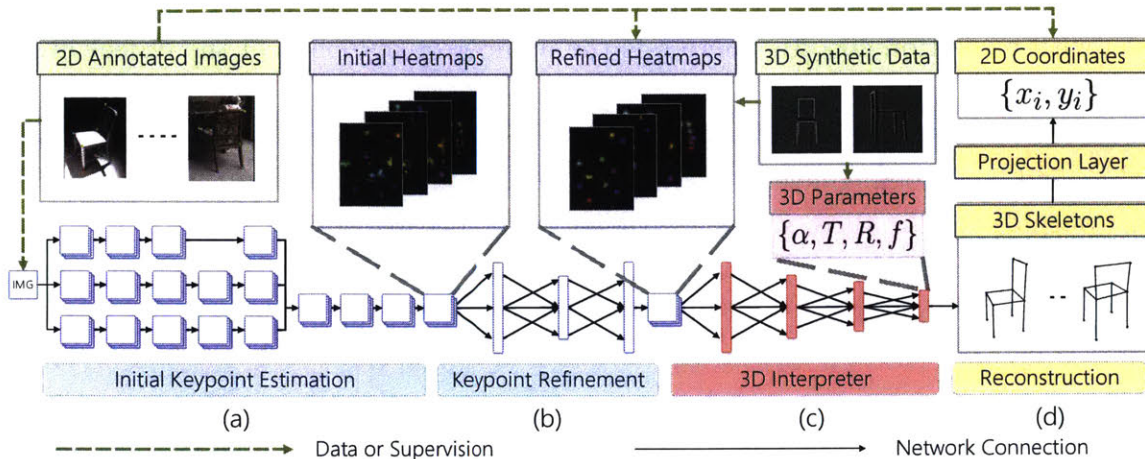
Figure 2-4: **Architecture of 3D-INN**. 3D-INN takes a single image as input and reconstructs the detailed 3D structure of the object in the image (e.g., human, chair, etc.). The network is trained independently for each category, and here we use chairs as an example. **(a)** Estimating 2D keypoint heatmaps with a multi-scale CNN. **(b)** Refining keypoint locations by considering the structural constraints between keypoints. This is implicitly enforced with an information bottleneck and yields cleaner heatmaps. **(c)** Recovered 3D structural and camera parameters $\{\alpha, T, R, f\}$. **(d)** The projection layer maps reconstructed 3D skeletons back to 2D keypoint coordinates.

Then when $f \to \infty$, we have

$$
\begin{cases}
x_i^1 = y_i^1, \\
x_i^2 = y_i^2,
\end{cases}
\tag{2.5}
$$

which is the formulation of parallel projection. Therefore, Equation 2.3 models the perspective projection when $f^{-1} \neq 0$ and models the parallel projection when $f^{-1} = 0$.

2.3.2 3D Interpreter Networks

Our network consists of three components: first, a keypoint estimator, which localizes 2D keypoints of objects from 2D images by regressing their heatmaps (Figure 2-4a and b, in blue); second, a 3D interpreter, which infers internal 3D structural and viewpoint parameters from the heatmaps (Figure 2-4c, in red); third, a projection layer, mapping 3D skeletons to 2D keypoint locations so that real 2D-annotated images can be used as supervision (Figure 2-4d, in yellow).

Keypoint estimation. The keypoint estimation stage consists of two steps: initial estimation (Figure 2-4a) and keypoint refinement (Figure 2-4b).

The network architecture for initial keypoint estimation is inspired by the pipeline proposed by Tompson et al. [2014, 2015]. The network takes multi-scaled images as input and estimates keypoint heatmaps. Specifically, we apply Local Contrast Normalization (LCN) on each image, and then scale it to 320×240, 160×120, and

80×60 as input to three separate scales of the network. The output is k heatmaps, each with resolution 40×30, where k is the number of keypoints of the object in the image.

At each scale, the network has three sets of 5×5 convolutional (with zero padding), ReLU, and 2×2 pooling layers, followed by a 9×9 convolutional and ReLU layer. The final outputs for the three scales are therefore images with resolution 40×30, 20×15, and 10×7, respectively. We then upsample the outputs of the last two scales to ensure they have the same resolution (40×30). The outputs from the three scales are later summed up and sent to a Batch Normalization layer and three 1×1 convolution layers, whose goal is to regress target heatmaps. We found that Batch Normalization is critical for convergence, while Spatial Dropout, proposed in Tompson et al. [2015], does not affect performance.

The second step of keypoint estimation is keypoint refinement, whose goal is to implicitly learn category-level structural constraints on keypoint locations after the initial keypoint localization. The motivation is to exploit the contextual and structural knowledge among keypoints (e.g., arms cannot be too far from the torso). We design a mini-network which, like an autoencoder, has information bottleneck layers, enforcing it to implicitly model the relationship among keypoints. Some previous papers have also explored this idea to achieve better performance in object detection [Ren et al., 2015] and face recognition [Taigman et al., 2015].

In the keypoint refinement network, we use three fully connected layers with widths 8,192, 4,096, and 8,192, respectively. After refinement, the heatmaps of keypoints are much cleaner, as shown in Section 2.4.

3D interpreter. The goal of our 3D interpreter is to infer 3D structure and viewpoint parameters, using estimated 2D heatmaps from earlier layers. While there are many different ways of solving Equation 2.2, our deep learning approach has clear advantages. First, traditional methods [Hejrati and Ramanan, 2012, Torresani et al., 2003] that minimize the reprojection error consider only one keypoint hypothesis, and are therefore not robust to noisy keypoint detection. In contrast, our framework uses soft heatmaps of keypoint locations as input, as shown in Figure 2-4c, which is more robust when some keypoints are invisible or incorrectly located. Second, our algorithm only requires a single forward propagation during testing, making it more efficient than the most previous optimization-base methods.

As discussed in Section 2.3.1, the set of 3D parameters we estimate is of $S = \{\alpha_i, R, T, f\}$, with which we are able to recover the 3D object structure using Equation 2.2. In our implementation, the network predicts f^{-1} instead of f for better numerical stability. As shown in Figure 2-4c, we use four fully connected layers as our 3D interpreter, with widths 2,048, 512, 128, and $|S|$, respectively. Spatial Transformer Networks [Jaderberg et al., 2015] also explored the idea of learning rotation parameters

R with neural nets, but our network can also recover structural parameters $\{\alpha_i\}$.

Projection layer. The last component of the network is a projection layer (Figure 2-4d). The projection layer takes estimated 3D parameters as input, and computes projected 2D keypoint coordinates $\{x_i, y_i\}$ using Equation 2.2. As all operations are differentiable, the projection layer enables us to use 2D-annotated images as ground truth, and to run back-propagation to update the entire network.

2.3.3 Training Strategy

A straightforward training strategy is to use real 2D images as input, and their 2D keypoint locations as supervision for the output of the projection layer. Unfortunately, experiments show that the network can hardly converge using this training scheme, due to the high-dimensional search space and the ambiguity in the 3D to 2D projection.

We therefore adopt an alternative three-step training strategy: first, training the keypoint estimator (Figure 2-4a and 2-4b) using real images with 2D keypoint heatmaps as supervision; second, training the 3D interpreter (Figure 2-4c) using synthetic 3D data as there are no ground truth 3D annotations available for real images; and third, training the whole network using real 2D images with supervision on the output of the projection layer at the end.

To generate synthetic 3D objects, for each object category, we first randomly sample structural parameters $\{\alpha_i\}$ and viewpoint parameters P, R and T. We calculate 3D keypoint coordinates using Equation 2.2. To model deformations that cannot be captured by base shapes, we add Gaussian perturbation to 3D keypoint locations of each synthetic 3D object, whose variance is 1% of its diagonal length. Examples of synthetic 3D shapes are shown in Figure 2-4c. In experiments, we do not render synthesized shapes; we use heatmaps of keypoints, rather than rendered images, as training input.

2.4 Evaluation

We evaluate our entire framework, 3D-INN, as well as each component within. In this section, we present both qualitative and quantitative results on 2D keypoint estimation (Section 2.4.1) and 3D structure and viewpoint recovery (Section 2.4.2).

2.4.1 2D Keypoint Estimation

Data. For 2D keypoint estimation, we evaluate our algorithm on three image datasets: FLIC [Sapp and Taskar, 2013] for human bodies, CUB-200-2011 [Wah et al., 2011] for birds, and a new dataset Keypoint-5 for furniture. Specifically, FLIC is a challenging dataset containing 3,987 training images and 1,016 test images, each labeled with 10 keypoints of human bodies. The CUB-200-2011 dataset was originally proposed for

| (a) PCK curves on the FLIC dataset | (b) A zoomed view of the dashed rectangle in (a) |
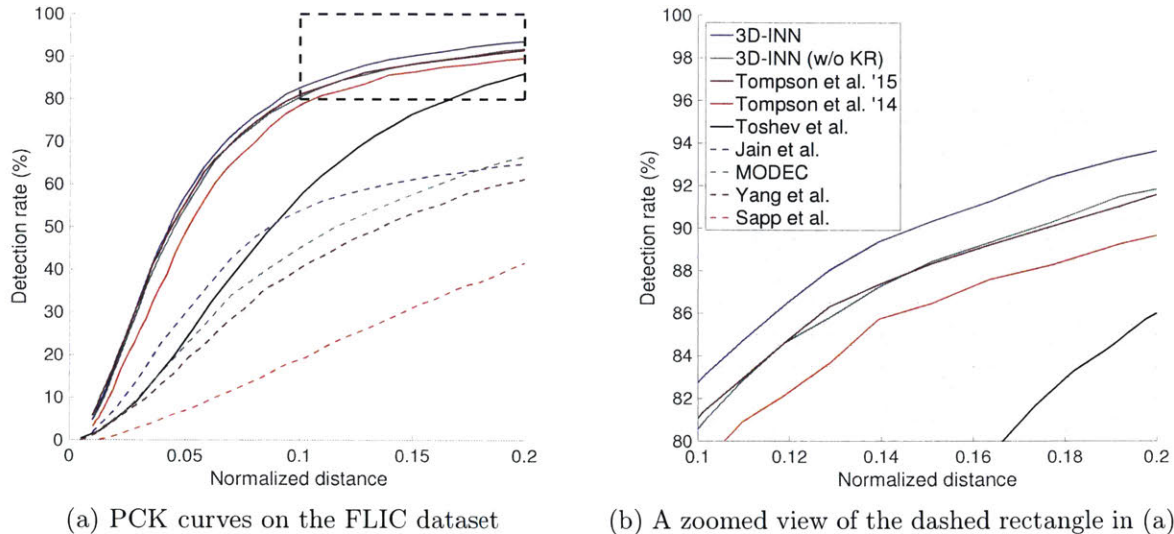
Figure 2-5: **Results on human keypoint estimation.** (a) PCK curves on the FLIC dataset [Sapp and Taskar, 2013]. 3D-INN performs consistently better than other methods. Without keypoint refinement, it is comparable to Tompson et al. [2015]. (b) A zoomed view of the dashed rectangle in (a).

fine-grained bird classification, but with labeled keypoints of bird parts. It has 5,994 images for training and 5,794 images for testing, each coming with up to 15 keypoints.

We also introduce a new dataset, Keypoint-5, which contains five categories: bed, chair, sofa, swivel chair, and table. There are 1,000 to 2,000 images in each category, where 80% are for training and 20% for testing. For each image, we asked three workers on Amazon Mechanical Turk to label locations of a pre-defined category-specific set of keypoints; we then, for each keypoint, used the median of the three labeled locations as ground truth.

Metrics. To quantitatively evaluate the accuracy of estimated keypoints on FLIC (human body), we use the standard Percentage of Correct Keypoints (PCK) measure [Sapp and Taskar, 2013] to be consistent with previous methods [Sapp and Taskar, 2013, Tompson et al., 2014, 2015]. We use the evaluation toolkit and results of competing methods released by Tompson et al. [2015]. On CUB-200-2011 (bird) and the new Keypoint-5 (furniture) dataset, following the convention [Liu and Belhumeur, 2013, Shih et al., 2015], we evaluate results in Percentage of Correct Parts (PCP) and Average Error (AE). PCP is defined as the percentage of keypoints localized within 1.5 times of the standard deviation of annotations. We use the evaluation code from [Liu and Belhumeur, 2013] to ensure consistency. Average error is computed as the mean of the distance, bounded by 5, between a predicted keypoint location and ground truth.

Results. For 2D keypoint detection, we only train the keypoint estimator in our 3D-INN (Figure 2-4a and 2-4b) using the training images in each dataset. Figure 2-5 shows the accuracy of keypoint estimation on the FLIC dataset. On this dataset,

Figure 2-6: **Qualitative results on 2D keypoint estimation** from a single image, where each color corresponds to a keypoint. The keypoint refinement step cleans up false positives and produces more regulated predictions.

Method	PCP (%)	Average Error
Poselets [Bourdev, 2011]	27.47	2.89
Consensus [Belhumeur et al., 2013]	48.70	2.13
Exemplar [Liu and Belhumeur, 2013]	59.74	1.80
Mdshift [Shih et al., 2015]	**69.1**	1.39
3D-INN (ours)	66.7	**1.36**
Human	84.72	1.00

Table 2.1: **Keypoint estimation results on the CUB-200-2011 dataset**, measured in PCP (%) and AE. Our method is comparable to Mdshift [Shih et al., 2015] (better in AE but worse in PCP), and better than all other algorithms.

we employ a fine-level network for post-processing, as suggested by Tompson et al. [2015]. Our method performs better than all previous methods [Sapp and Taskar, 2013, Tompson et al., 2014, 2015, Yang and Ramanan, 2011, Toshev and Szegedy, 2014] at all precisions. Moreover, the keypoint refinement step (Figure 2-4c) improves results significantly (about 2% for a normalized distance ≥ 0.15), without which our framework has similar performance with Tompson et al. [2015]. Such improvement is also demonstrated in Figure 2-6, where the heatmaps after refinement are far less noisy.

The accuracy of keypoint estimation on CUB-200-201 dataset is listed in Table 2.1. Our method is better than Liu and Belhumeur [2013] in both metrics, and is comparable to the state-of-the-art [Shih et al., 2015]. Specifically, compared with Shih et al. [2015], our model more precisely estimates the keypoint locations for correctly detected parts (a lower AE), but misses more parts in the detection (a lower PCP). On our

	Method	Bed	Chair	Sofa	Swivel Chair
PCP	3D-INN (ours)	**77.4**	**87.7**	**77.4**	**78.5**
	Tompson et al. [2015]	76.2	85.3	76.9	69.2
AE	3D-INN (ours)	**1.16**	**0.92**	**1.14**	**1.19**
	Tompson et al. [2015]	1.20	1.02	1.19	1.54

Table 2.2: **Keypoint estimation results of 3D-INN and Tompson et al. [2015] on Keypoint-5**, measured in PCP (%) and AE. 3D-INN is consistently better in both measures. We retrained the network in Tompson et al. [2015] on Keypoint-5.

Keypoint-5 dataset, our model achieves higher PCPs and lower AEs compared to the state-of-the-art [Tompson et al., 2015] for all categories, as shown in Table 2.2. These experiments in general demonstrate the effectiveness of our model on keypoint detection.

2.4.2 3D Structure and Viewpoint Estimation

For 3D structural parameter estimation, we evaluate 3D-INN from three different perspectives. First, we evaluate our 3D interpreter (Figure 2-4c alone) against the optimization-based method [Zhou et al., 2015]. Second, we test our full pipeline on the IKEA dataset [Lim et al., 2013], where ground truth 3D labels are available, comparing with both baselines and the state-of-the-art. Third, we show qualitative results on four datasets: Keypoint-5, IKEA, the SUN database [Xiao et al., 2010], and PASCAL 3D+ [Xiang et al., 2014].

Comparing with an optimization-based method. First, we compare our 3D interpreter (Figure 2-4c) with the state-of-the-art optimization-based method that directly minimizes re-projection error (Equation 2.2) on the synthetic data. As most optimization based methods only consider the parallel projection, while we model perspective projection for real images, we extend the one by Zhou et al. [2015] as follows: we first use their algorithm to get an initial guess of internal parameters and viewpoints, and then apply a simple gradient descent method to refine it considering perspective distortion.

We generate synthetic data for this experiment, using the scheme described in Section 2.3.3. Each data point contains the 2D keypoint heatmaps of an object, and its corresponding 3D keypoint locations and viewpoint, which we would like to estimate. We also add different levels of salt-and-pepper noise to heatmaps to evaluate the robustness of both methods. We generated 30,000 training and 1,000 testing cases. Because the analytic solution only takes keypoint coordinates as input, we convert heatmaps to coordinates using an argmax function.

For both methods, we evaluate their performance on both 3D structure recovery and 3D viewpoint estimation. To evaluate the estimated 3D structure, we compare

(a) Structure estimation   (b) Viewpoint estimation
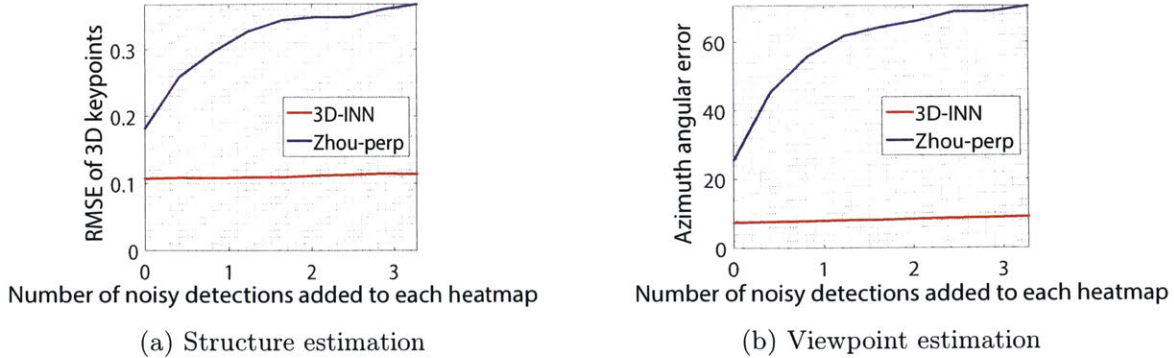
Figure 2-7: **Plots comparing our method against an analytical solution on synthetic heatmaps.** (a) The accuracy of 3D structure estimation; (b) The accuracy of 3D viewpoint estimation.

their accuracies on 3D keypoint estimation ($\mathbf{Y}$ in Section 2.3.1); for 3D viewpoint estimation, we compute errors in azimuth angle, following previous work [Su et al., 2015]. As the original algorithm by Zhou et al. [2015] was mainly designed for the parallel projection and comparatively clean heatmaps, our 3D interpreter outperforms it in the presence of noise and perspective distortion, as shown in Figure 2-7. Our algorithm is also efficient, taking less than 50 milliseconds for each test image.

Evaluating the full pipeline. We now evaluate 3D-INN on estimating 3D structure and 3D viewpoint. We use the IKEA dataset [Lim et al., 2013] for evaluation, as it provides ground truth 3D mesh models and the associated viewpoints for testing images. We manually label ground truth 3D keypoint locations on provided 3D meshes, and calculate the root-mean-square error (RMSE) between estimated and ground truth 3D keypoint locations.

As IKEA only has no more than 200 images per category, we instead train 3D-INN on our Keypoint-5, as well as one million synthetic data points, using the strategy described in Section 2.3.3. Note that, first, we are only using no more than 2,000 real images per category for training and, second, we are testing the trained model on different datasets, avoiding possible dataset biases [Torralba and Efros, 2011].

We first compare with a baseline method to evaluate our training paradigm: we show quantitative comparisons between 3D-INN trained using our paradigm proposed in Section 2.3.3, and the same network but only end-to-end trained with real images, without having the two pre-training stages. We called it the *scratch* model.

As shown in the RMSE-Recall curve in Figure 2-8, 3D-INN performs much better than *scratch* on both 3D structure and viewpoint estimation. The average recall of 3D-INN is about 20% higher than *scratch* in 3D structure estimation, and about 40% higher in 3D pose estimation. This shows the effectiveness of the proposed training paradigm.

(a) Structure estimation   (b) Viewpoint estimation
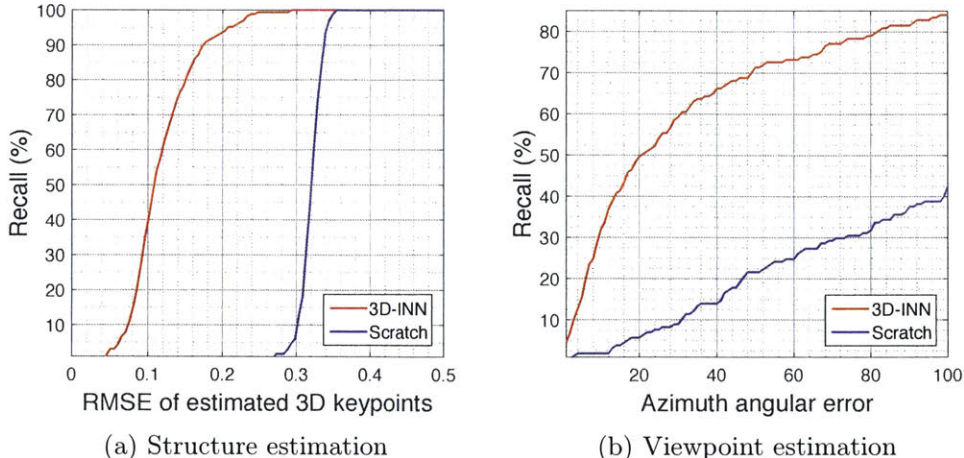
Figure 2-8: **Evaluation the training paradigm on chairs from the IKEA dataset** [Lim et al., 2013]. The network trained with our paradigm (3D-INN) is significantly better than the one trained from *scratch* on both 3D structure (a) and viewpoint estimation (b).

We then compare our full model with the state-of-the-art methods. The left half of Figure 2-9 shows RMSE-Recall curve of both our algorithm and the optimization-based method described above (Zhou et al. [2015]-perp). The y-axis shows the recall — the percentage of testing samples under a certain RMSE threshold. We test two versions of our algorithm: with fine-tuning (3D-INN) and without fine-tuning (3D-INN w/o FT). Both significantly outperform the optimization-based method [Zhou et al., 2015]. This is because the method from Zhou et al. [2015] was not designed to handle noisy keypoint estimation and perspective distortions, while our 3D-INN can deal with them. Also, fine-tuning improves the accuracy of keypoint estimation by about 5% under the RMSE threshold 0.15.

Though we focus on recovering 3D object structure, as an extension, we also evaluate 3D-INN on 3D viewpoint estimation. We compare it with the state-of-the-art viewpoint estimation algorithm by Su et al. [2015]. The right half of Figure 2-9 shows the results (recall) in azimuth angle. As shown in the table, 3D-INN outperforms Su et al. [2015] by about 40% (relative), measured in average recall. This is mainly because it is not straightforward for Su et al. [2015], mostly trained on (cropped) synthesized images, to deal with the large number of heavily occluded objects in the IKEA dataset.

Although our algorithm assumes a centered object in an input image, we can apply it, in combination with an object detection algorithm, on images where object locations are unknown. We evaluate the results of joint object detection and viewpoint estimation on PASCAL 3D+ dataset [Xiang et al., 2014]. PASCAL 3D+ and Keypoint-5 has two overlapping categories: chair and sofa, and we evaluate on both. We also study an additional object category, car, for which 3D-INN is trained on 1,000 car

RMSE of estimated 3D keypoints

Azimuth angular error

	Average recall (%)			
Method	Bed	Sofa	Chair	Avg.
3D-INN	**88.64**	**88.03**	**87.84**	**88.03**
3D-INN w/o FT	87.19	87.10	87.08	87.10
Zhou et al. [2015]-perp	52.31	58.02	60.76	58.46

(a) Structure estimation

	Average recall (%)			
Method	Table	Sofa	Chair	Avg.
3D-INN	**55.02**	64.65	**63.46**	**60.30**
3D-INN w/o FT	52.33	**65.45**	62.01	58.90
Su et al. [2015]	52.73	35.65	37.69	43.34

(b) Viewpoint estimation

Figure 2-9: **Evaluation on the IKEA dataset** [Lim et al., 2013]. (a) The accuracy of structure estimation. RMSE-Recall curved is shown in the first row, and zoomed-views of the dashed rectangular regions are shown on the right. The third row shows the average recall on all thresholds. (b) The accuracy of viewpoint estimation.

Category	VDPM	DPM-VOC+VP	RenderForCNN	V&K	3D-INN (ours)
Chair	6.8	6.1	15.7	**25.1**	23.1
Sofa	5.1	11.8	18.6	43.8	**45.8**
Car	20.2	36.9	41.8	**55.2**	52.2

Table 2.3: **Joint object detection and viewpoint estimation on the PASCAL 3D+ dataset [Xiang et al., 2014].** Following previous work, we use Average Viewpoint Precision (AVP) as our measure, which extends AP so that a true positive should have both a correct bounding box and a correct viewpoint (here we use a 4-view quantization). Both 3D-INN and V&K [Tulsiani and Malik, 2015] use R-CNN [Girshick et al., 2014] for object detection, precluding the influence of object detectors. The others use their own detection algorithm. VDPM [Xiang et al., 2014] and DPM-VOC+VP [Pepik et al., 2012] are trained on PASCAL VOC 2012, V&K [Tulsiani and Malik, 2015] is trained on PASCAL 3D+, RenderForCNN [Su et al., 2015] is trained on PASCAL VOC 2012, together with synthetic 3D CAD models, and 3D-INN is trained on Keypoint-5.

images from ImageNet [Russakovsky et al., 2015] with 2D keypoint annotations. Following Tulsiani and Malik [2015], we use non-occluded and non-truncated objects for testing. We use the standard R-CNN [Girshick et al., 2014] for object detection, and our 3D-INN for viewpoint estimation.

Table 2.3 shows that 3D-INN is comparable with Viewpoints and Keypoints (V&K by Tulsiani and Malik [2015]), and outperforms other algorithms with a significant margin. Both 3D-INN and V&K use R-CNN [Girshick et al., 2014] for object detection (we use the R-CNN detection results provided by Tulsiani and Malik [2015]); this rules out the influence of object detectors. Further, while all the other algorithms are trained on either PASCAL VOC or PASCAL 3D+, ours is trained on Keypoint-5 or ImageNet. This indicates our learned model transfers well across datasets, without

Figure 2-10: **Qualitative results on Keypoint-5, IKEA, and SUN databases**. For each example, the first one is the input image, the second one is the reconstruct 3D skeleton using the network before fine-tuning, and third one is using the network after fine-tuning. The last column shows failure cases.

suffering much from the domain adaptation issue.

Qualitative results on benchmarks. We show qualitative results on Keypoint-5, IKEA, the SUN database [Xiao et al., 2010], and the PASCAL 3D+ dataset [Xiang et al., 2014] in Figure 2-10. When the image is clean and objects are not occluded,
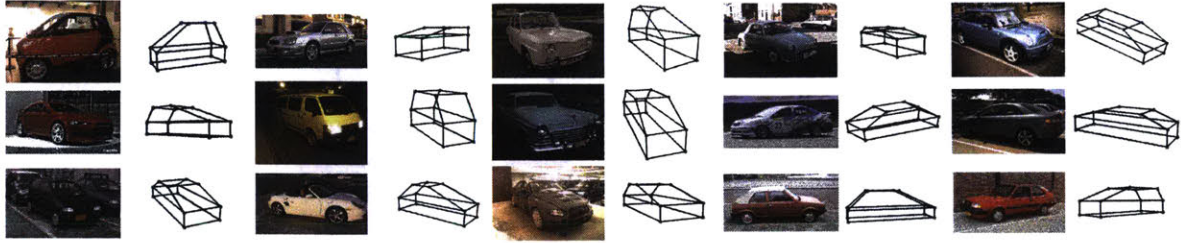
Figure 2-11: **Car structure estimation** on images from the PASCAL 3D+ dataset [Xiang et al., 2014]

(a) Training: beds, Test: chairs　　　　(b) Training: sofas, Test: chairs

Figure 2-12: **Qualitative results on chairs using networks trained on sofas or beds**. In most cases models provide reasonable output. Mistakes are often due to the difference between the training and test sets, e.g., in the third example, the model trained on beds fails to estimate chairs facing backward.

our algorithm can recover 3D object structure and viewpoint with high accuracy. Fine-tuning further helps to improve the results (see chairs at row 1 column 1, and row 4 column 1). Our algorithm is also robust to partial occlusion, demonstrated by the IKEA bed at row 5 column 1. We show failure cases in the last column: one major failure case is when the object is heavily cropped in the input image (the last column, row 4 to 7), as the 3D object skeleton becomes hard to infer. Figure 2-11 shows more results on car structure recovery.

When 3D-INN is used in combination with detection models, it needs to deal with imperfect detection results. Here, we also evaluate 3D-INN on noisy input, specifically, on images with an object from a different but similar category. Figure 2-12 shows the recovered 3D structures of chairs using a model trained either on sofas or beds. In most cases 3D-INN still provides reasonable output, and the mistakes are mostly due to the difference between training and test sets, e.g., the model trained on beds does not perform well on chairs facing backward, because there are almost no beds with a similar viewpoint in the training set.

Figure 2-13 and Figure 2-14 include more results on chair and sofa images randomly sampled from the test set of Keypoint-5.

2.5　Applications

The inferred latent parameters, as a compact and informative representation of objects in images, have wide applications. In this section, we demonstrate representative

Figure 2-13: **Estimated 3D skeletons on more Keypoint-5 chair images**. Images are randomly sampled from the test set.

Figure 2-14: **Estimated 3D skeletons on more Keypoint-5 sofa images**. Images are randomly sampled from the test set.

ones including 3D object rendering, image retrieval, and object graph visualization.
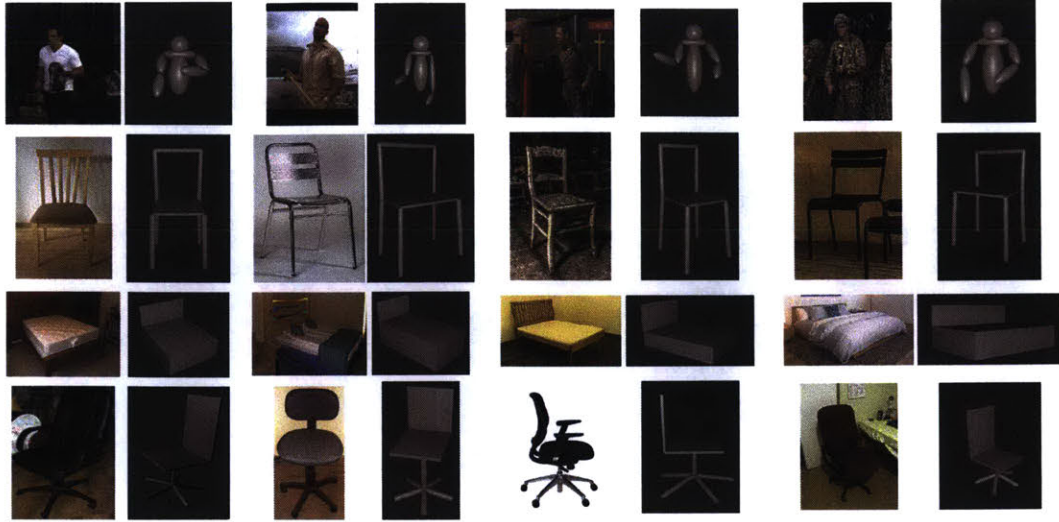
Figure 2-15: **Visualization of 3D reconstruction results**. We render objects using *Blender*.

(a) Retrieval results of a sofa image

(b) Retrieval results of a chair image
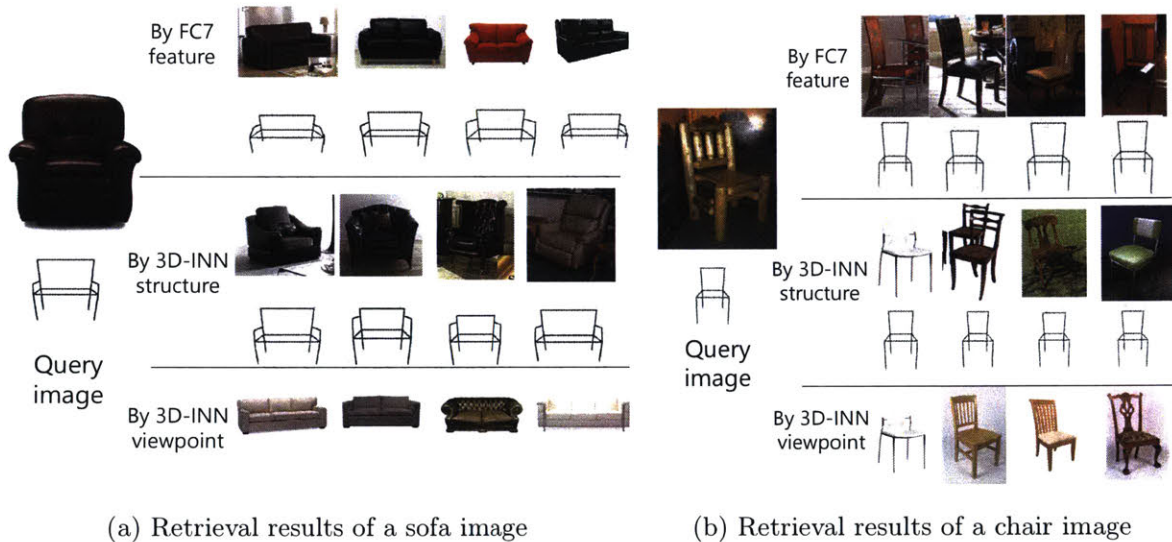
Figure 2-16: **Retrieval results for a sofa (b) and a chair (b)** in different feature spaces. 3D-INN helps to retrieve objects with similar 3D structure or pictured in a similar viewpoint.

3D object rendering. Given an estimated 3D object structure, we can render it in a 3D graphics engine like Blender, as shown in Figure 2-15.

Image retrieval. Using estimated 3D structural and viewpoint information, we can retrieve images based on their 3D configurations. Figure 2-16 shows image retrieval results using FC7 features from AlexNet [Krizhevsky et al., 2012] and using the 3D structure and viewpoint learned by 3D-INN. Our retrieval database includes all testing images of chairs and sofas in Keypoint-5. In each row, we sort the best matches of the query image, measured by Euclidean distance in a specific feature space. We retrieve images in two ways: *by structure* uses estimated internal structural parameters ($\{\alpha_i\}$

31
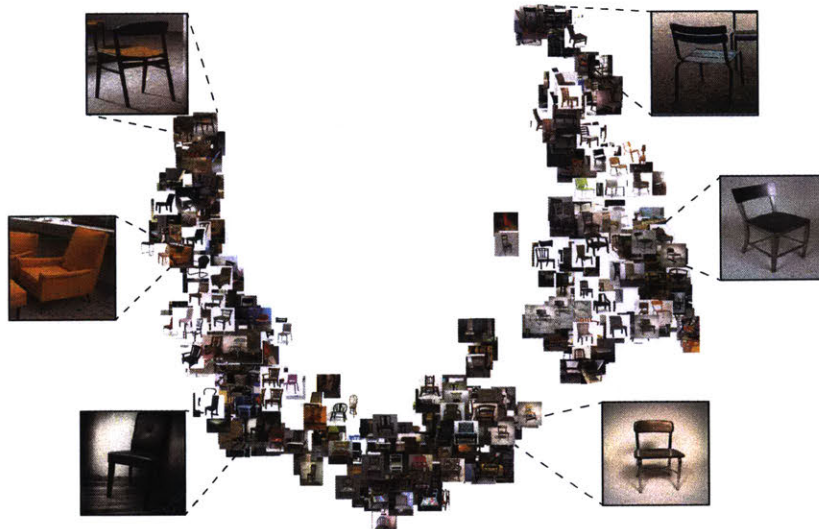
Figure 2-17: **Object graph visualization based on learned object representations**: we visualize images using t-SNE [Van der Maaten and Hinton, 2008] on 3D viewpoints predicted by 3D-INN.

in Equation 2.2), and *by viewpoint* uses estimated external viewpoint parameters (R in Equation 2.2).

Object graph. Similar to the retrieval task, we visualize all test images for chairs in Keypoint-5 in Figure 2-17, using t-SNE [Van der Maaten and Hinton, 2008] on estimated 3D viewpoints. Note the smooth transition from the chairs facing left to those facing right.

2.6 Discussion

We have introduced 3D INterpreter Networks (3D-INN). From a single image, our model recovers the 2D keypoints and 3D structure of a (possibly deformable) object, as well as camera parameters. To achieve this goal, we used 3D skeletons as an abstract 3D representation, incorporated a projection layer to the network for learning 3D parameters from 2D labels, and employed keypoint heatmaps to connect real and synthetic data. Empirically, we showed that 3D-INN performs well on both 2D keypoint estimation and 3D structure and viewpoint recovery, comparable to or better than the state of the art. Further, various applications demonstrated the potential of the skeleton representation learned by 3D-INN.

We choose to model objects via 3D skeletons and the corresponding 2D keypoints, as opposed to other dense 3D representations such as voxels, meshes, and point clouds, because skeletons offer unique advantages. First, given an RGB image, its sparse 2D annotations like keypoints are easier and more affordable to acquire, and can be used as 2D supervision for 3D skeleton and viewpoint estimation; in comparison, it is

prohibitively challenging to obtain dense annotations like a depth map to constrain 3D reconstructions in voxels or meshes. Second, the employed base shapes carry rich category-specific shape priors, with which 3D-INN can encode an object skeleton with a few parameters. This feature is particularly useful on platforms with severe memory and computational constraints, such as on autonomous cars and on mobile phones.

That being said, skeletons have their own limitations. The most significant is on its generalization power: there are many real-world objects whose keypoints are hard to define, such as trees, flowers, and deformable shapes like ropes; in those cases, there lacks a straightforward way to apply 3D-INN to model these objects. Recent research on 3D reconstruction via richer, generic intermediate representations like intrinsic images [Barrow and Tenenbaum, 1978] suggests a potential solution to the problem, though as discussed above it is much harder to obtain annotated intrinsic images, compared to keypoints [Wu et al., 2017c].

We focus on single-view 3D reconstruction. As discussed in Section 2.1, requiring only a single image as input has unique practical advantages, in addition to its scientific value. First, our algorithm can be directly applied to cases where only in-the-wild images are available, not multi-view images or videos. Second, taking a single image as input enables online inference and therefore fits real-time applications; in contrast, most multi-view reconstruction algorithms are offline. It is also possible to our 3D-INN to use multi-view data when they are available [Kar et al., 2017], and more generally, to integrate viewer-centered and object-centered representations in a principled manner [Hinton, 1981].

3D-INN estimates the 3D skeleton and pose of an object from an RGB image, and can therefore be applied to the enormous existing RGB data. But we are also aware that depth sensors have recently become affordable to end users [Newcombe et al., 2011], and large-scale RGB-D datasets are being built [Song et al., 2017, McCormac et al., 2017]. Depth data help to resolve the ambiguity in the projection from 3D shapes to 2D images, allow object structure prediction in metric scale, and enable wide applications [Chen et al., 2012]. Hence, a promising future research topic would be to extend the current framework to handle depth data, while enforcing the 2D-3D differentiable consistencies in various forms [Tulsiani et al., 2017b, Wu et al., 2017c]. This will be our main focus in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Learning with a Graphics Engine
for Dense Object Shapes

In this chapter, we extend our model to produce dense 3D object shapes from a single image, beyond 3D skeletons. From a single image, humans are able to perceive the full 3D shape of an object by exploiting learned shape priors from everyday life, even if we've never seen the object before. Contemporary single-image 3D reconstruction algorithms aim to solve this task in a similar fashion, but often end up with priors that are highly biased by training classes. Our goal is to build a shape reconstruction model that generalizes.

Our main contribution in this chapter is an algorithm, *Generalizable Reconstruction (GenRe)*, designed to capture more generic, class-agnostic shape priors. Similar to the 3D interpreter networks introduced in Chapter 2, we achieve this with two major technical innovations. First, analogous to heatmaps, in GenRe we exploit multiple intermediate representations, including 2.5D visible surfaces (depth and silhouette) and spherical shape representations of both visible and non-visible surfaces. Second, these representations, together with the final 3D voxel-based representations, are combined in a principled manner that exploits the causal structure of how 3D shapes give rise to 2D images, namely the 3D-to-2D projection. Experiments demonstrate that GenRe performs well on single-view shape reconstruction, and generalizes to diverse novel objects from categories not seen during training.

This chapter includes materials previously published as Wu et al. [2016c, 2017c, 2018b], Sun et al. [2018b], Zhang et al. [2018b]. Many collaborators, in particular Zhoutong Zhang, Xiuming Zhang, Chengkai Zhang, Tianfan Xue, Yifan Wang, and Xingyuan Sun, contributed significantly to the materials presented in this chapter.

3.1 Introduction

Humans can imagine an object's full 3D shape from just a single image, showing only a fraction of the object's surface. This applies to common objects such as chairs,

but also to novel objects that we have never seen before. Vision researchers have long argued that the key to this ability may be a sophisticated hierarchy of representations, extending from images through surfaces to volumetric shape, which process different aspects of shape in different representational formats [Marr, 1982]. Here we explore how these ideas can be integrated into state-of-the-art computer vision systems for 3D shape reconstruction.

Recently, computer vision and machine learning researchers have made impressive progress on single-image 3D reconstruction by learning a parametric function $f_{\text{2D}\rightarrow\text{3D}}$, implemented as deep neural networks, that maps a 2D image to its corresponding 3D shape. Essentially, $f_{\text{2D}\rightarrow\text{3D}}$ encodes shape priors ("what realistic shapes look like"), often learned from large shape repositories such as ShapeNet [Chang et al., 2015]. Because the problem is well-known to be ill-posed—there exist many 3D explanations for any 2D visual observation—modern systems have explored looping in various structures into this learning process. For example, MarrNet [Wu et al., 2017c] uses intrinsic images or 2.5D sketches [Marr, 1982] as an intermediate representation, and concatenates two learned mappings for shape reconstruction: $f_{\text{2D}\rightarrow\text{3D}} = f_{\text{2.5D}\rightarrow\text{3D}} \circ f_{\text{2D}\rightarrow\text{2.5D}}$.

Many existing methods, however, ignore the fact that mapping a 2D image or a 2.5D sketch to a 3D shape involves complex, but deterministic geometric projections. Simply using a neural network to approximate this projection, instead of modeling this mapping explicitly, leads to inference models that are overparametrized (and hence subject to overfitting training classes). It also misses valuable inductive biases that can be wired in through such projections. Both factors contribute to poor generalization to unseen classes.

Here we propose to disentangle geometric projections from shape reconstruction to better generalize to unseen shape categories. Building upon the MarrNet framework [Wu et al., 2017c], we further decompose $f_{\text{2.5D}\rightarrow\text{3D}}$ into a deterministic geometric projection p from 2.5D to a partial 3D model, and a learnable completion c of the 3D model. A straightforward version of this idea would be to perform shape completion in the 3D voxel grid: $f_{\text{2.5D}\rightarrow\text{3D}} = c_{\text{3D}\rightarrow\text{3D}} \circ p_{\text{2.5D}\rightarrow\text{3D}}$. However, shape completion in 3D is challenging, as the manifold of plausible shapes is sparser in 3D than in 2D, and empirically this fails to reconstruct shapes well.

Instead we perform completion based on spherical maps. Spherical maps are surface representations defined on the UV coordinates of a unit sphere, where the value at each coordinate is calculated as the minimal distance travelled from this point to the 3D object surface along the sphere's radius. Such a representation combines appealing features of 2D and 3D: spherical maps are a form of 2D images, on which neural inpainting models work well; but they have a semantics that allows them to be projected into 3D to recover full shape geometry. They essentially allow us to complete non-visible object surfaces from visible ones, as a further intermediate step

to full 3D reconstruction. We now have $f_{2.5D\to3D} = p_{S\to3D} \circ c_{S\to S} \circ p_{2.5D\to S}$, where S stands for spherical maps.

Our full model, named *Generalizable Reconstruction (GenRe)*, thus comprises three cascaded, learnable modules connected by fixed geometric projections. First, a single-view depth estimator predicts depth from a 2D image ($f_{2D\to2.5D}$); the depth map is then projected into a spherical map ($p_{2.5D\to S}$). Second, a spherical map inpainting network inpaints the partial spherical map ($c_{S\to S}$); the inpainted spherical map is then projected into 3D voxels ($p_{2.5D\to3D}$). Finally, we introduce an additional voxel refinement network to refine the estimated 3D shape in voxel space. Our neural modules only have to model object geometry for reconstruction, without having to learn geometric projections. This enhances generalizability, along with several other factors: during training, our modularized design forces each module of the network to use features from the previous module, instead of directly memorizing shapes from the training classes; also, each module only predicts outputs that are in the same domain as its inputs (image-based or voxel-based), which leads to more regular mappings.

Our GenRe model achieves state-of-the-art performance on reconstructing shapes both within and outside training classes. Figure 3-1 shows examples of our model reconstructing a table and a bed from single images, after training only on cars, chairs, and airplanes. We also present detailed analyses of how each component contributes to the final prediction.

Our model can also be complemented with learned 3D shape priors. We show that deep networks are also effective at capturing the prior distribution of possible 3D shapes and, further, the learned 3D shape priors can be integrated with recognition models for better performance on single-image 3D reconstruction.

In this chapter, we present four contributions. First, we emphasize the task of generalizable single-image 3D shape reconstruction. Second, we propose to disentangle geometric projections from shape reconstruction, and include spherical maps with differentiable, deterministic projections in an integrated neural model. Third, we demonstrate that it is possible to learn 3D shape priors with deep networks and, further, the learned priors can assist in shape reconstruction; last, we show that the resulting model achieves state-of-the-art performance on single-image 3D shape reconstruction for objects within and outside training classes.

3.2   Related Work

Single-image 3D reconstruction.   The problem of recovering the object shape from a single image is challenging, as it requires both powerful recognition systems and prior knowledge of plausible 3D shapes. Large CAD model repositories [Chang et al., 2015] and deep networks have contributed to the significant progress in recent years,

| Input (Novel Class) | Our Reconstruction | Input (Novel Class) | Our Reconstruction |
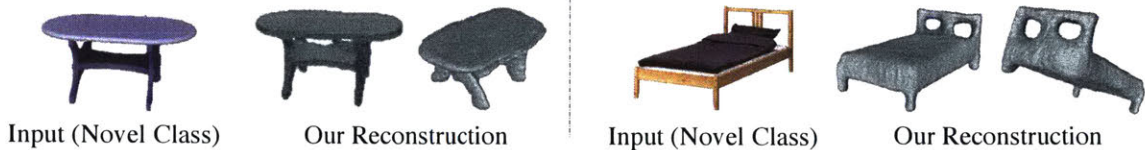
Figure 3-1: **We study the task of generalizable single-image 3D reconstruction**, aiming to reconstruct the 3D shape of an object outside training classes. Here we show a table and a bed reconstructed from single RGB images by our model trained on cars, chairs, and airplanes. Our model learns to reconstruct objects outside the training classes.

mostly with voxel representations [Choy et al., 2016, Girdhar et al., 2016, Häne et al., 2017, Kar et al., 2015, Novotny et al., 2017, Rezende et al., 2016a, Tatarchenko et al., 2016, Tulsiani et al., 2017b, Wu et al., 2016c, 2017c, 2018b, Zhu et al., 2018b, Yan et al., 2016b]. Apart from voxels, some researchers have also studied reconstructing objects in point clouds [Fan et al., 2017] or octave trees [Riegler et al., 2017, Tatarchenko et al., 2017]. The shape priors learned in these approaches, however, are in general only applicable to their training classes, with very limited generalization power for reconstructing shapes from unseen categories. In contrast, our system exploits 2.5D sketches and spherical representations for better generalization to objects outside training classes.

Spherical projections. Spherical projections have been shown effective in 3D shape retrieval [Esteves et al., 2018], classification [Cao et al., 2017], and finding possible rotational as well as reflective symmetries [Kazhdan et al., 2004, 2002]. Recent papers [Cohen et al., 2018, 2017] have studied differentiable, spherical convolution on spherical projections, aiming to preserve rotational equivariance within a neural network. These designs, however, perform convolution in the spectral domain with limited frequency bands, causing aliasing and loss of high-frequency information. In particular, convolution in the spectral domain is not suitable for shape reconstruction, since the reconstruction quality highly depends on the high-frequency components. In addition, the ringing effects caused by aliasing would introduce undesired artifacts.

2.5D sketch recovery. The origin of intrinsic image estimation dates back to the early years of computer vision [Barrow and Tenenbaum, 1978]. Through the years, researchers have explored recovering 2.5D sketches from texture, shading, or color images [Barron and Malik, 2015, Bell et al., 2014, Horn and Brooks, 1989, Tappen et al., 2003, Weiss, 2001, Zhang et al., 1999]. As handy depth sensors get mature [Izadi et al., 2011], and larger-scale RGB-D datasets become available [McCormac et al., 2017, Silberman et al., 2012, Song et al., 2017], many papers start to estimate depth [Chen et al., 2016b, Eigen and Fergus, 2015], surface normals [Bansal and Russell, 2016, Wang et al., 2015], and other intrinsic images [Janner et al., 2017, Shi et al., 2017] with deep networks. Our method employs 2.5D estimation as a component, but focuses on

38

reconstructing shapes from unseen categories.

Zero- and few-shot recognition. In computer vision, abundant attempts have been made to tackle the problem of few-shot recognition. We refer readers to the review article [Xian et al., 2017] for a comprehensive list. A number of earlier papers have explored sharing features across categories to recognize new objects from a few examples [Bart and Ullman, 2005, Farhadi et al., 2009, Lampert et al., 2009, Torralba et al., 2007]. More recently, many researchers have begun to study zero- or few-shot recognition with deep networks [Akata et al., 2016, Antol et al., 2014, Hariharan and Girshick, 2017, Wang et al., 2017, Wang and Hebert, 2016]. Especially, Peng et al. [2015] explored the idea of learning to recognize novel 3D models via domain adaptation.

While these proposed methods are for recognizing and categorizing images or shapes, in this chapter we explore reconstructing the 3D shape of an object from unseen classes. This problem has received little attention in the past, possibly due to its considerable difficulty. A few imaging systems have attempted to recover 3D shape from a single shot by making use of special cameras [Proesmans et al., 1996, Sagawa et al., 2011]. Unlike them, we study 3D reconstruction from a single RGB image. Very recently, researchers have begun to look at the generalization power of 3D reconstruction algorithms [Shin et al., 2018, Jayaraman et al., 2018, Rock et al., 2015, Funk and Liu, 2017]. Here we present a novel approach that makes use of spherical representations for better generalization.

3.3 Dataset: Pix3D

A major challenge to building better reconstruction algorithms is the lack of high-quality data. Existing datasets have various limitations for the task of single-image 3D shape modeling: ShapeNet [Chang et al., 2015] is a large dataset for 3D models, but does not come with real images; Pascal 3D+ [Xiang et al., 2014] and ObjectNet3D [Xiang et al., 2016] have real images, but the image-shape alignment is rough because the 3D models do not match the objects in images; IKEA [Lim et al., 2013] has high-quality image-3D alignment, but it only contains 90 3D models and 759 images.

We desire a dataset that has all three merits—a large-scale dataset of real images and ground-truth shapes with precise 2D-3D alignment. In this section, we introduce our dataset, Pix3D, including 395 3D shapes of nine object categories and 10,169 real images, capturing the exact object in diverse environments. Further, all image-shape pairs have precise 3D annotations, giving pixel-level alignment between shapes and their silhouettes in the images.

Figure 3-2 summarizes how we build Pix3D. We collect raw images from web
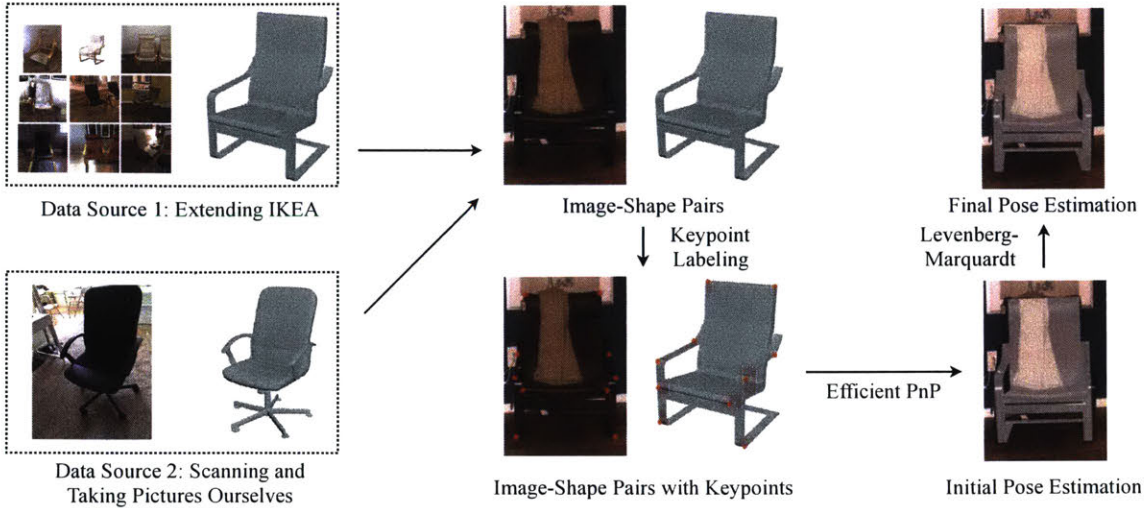
Figure 3-2: **The construction of Pix3D.** We build the dataset in two steps. First, we collect image-shape pairs by crawling web images of IKEA furniture as well as scanning objects and taking pictures ourselves. Second, we align the shapes with their 2D silhouettes by minimizing the 2D coordinates of the keypoints and their projected positions from 3D, using the Efficient PnP and the Levenberg-Marquardt algorithm.

search engines and shapes from 3D repositories; we also take pictures and scan shapes ourselves. Finally, we use labeled keypoints on both 2D images and 3D shapes to align them.

3.3.1 Collecting Image-Shape Pairs

We obtain raw image-shape pairs in two ways. One is to crawl images of IKEA furniture from the web and align them with CAD models provided in the IKEA dataset [Lim et al., 2013]. The other is to directly scan 3D shapes and take pictures.

Extending IKEA. The IKEA dataset contains 219 high-quality 3D models of IKEA furniture, but has only 759 images for 90 shapes. Therefore, we choose to keep the 3D shapes from IKEA dataset, but expand the set of 2D images using online image search engines and crowdsourcing.

For each 3D shape, we first search for its corresponding 2D images through Google, Bing, and Baidu, using its IKEA model name as the keyword. We obtain 104,220 images for the 219 shapes. We then use Amazon Mechanical Turk (AMT) to remove irrelevant ones. For each image, we ask three AMT workers to label whether this image matches the 3D shape or not. For images whose three responses differ, we ask three additional workers and decide whether to keep them based on majority voting. We end up with 14,600 images for the 219 IKEA shapes.

3D scan. We scan non-IKEA objects with a Structure Sensor* mounted on an iPad. We choose to use the Structure Sensor because its mobility enables us to capture a wide range of shapes.

The iPad RGB camera is synchronized with the depth sensor at 30 Hz, and calibrated by the Scanner App provided by Occipital, Inc.† The resolution of RGB frames is 2592×1936, and the resolution of depth frames is 320×240. For each object, we take a short video and fuse the depth data to get its 3D mesh by using fusion algorithm provided by Occipital, Inc. We also take 10–20 images for each scanned object in front of various backgrounds from different viewpoints, making sure the object is neither cropped nor occluded. In total, we have scanned 209 objects and taken 2,313 images. Combining these with the IKEA shapes and images, we have 418 shapes and 16,913 images altogether.

3.3.2 Image-Shape Alignment

To align a 3D CAD model with its projection in a 2D image, we need to solve for its 3D pose (translation and rotation), and the camera parameters used to capture the image.

We use a keypoint-based method inspired by Lim et al. [2013]. Denote the keypoints' 2D coordinates as $X_{2D} = \{x_1, x_2, \cdots, x_n\}$ and their corresponding 3D coordinates as $X_{3D} = \{X_1, X_2, \cdots, X_n\}$. We solve for camera parameters and 3D poses that minimize the reprojection error of the keypoints. Specifically, we want to find the projection matrix P that minimizes

$$\mathcal{L}(P; X_{3D}, X_{2D}) = \sum_i \|\text{Proj}_P(X_i) - x_i\|_2^2, \tag{3.1}$$

where $\text{Proj}_P(\cdot)$ is the projection function.

Under the central projection assumption (zero-skew, square pixel, and the optical center is at the center of the frame), we have $P = K[R|T]$, where K is the camera intrinsic matrix; $R \in \mathbb{R}^{3\times3}$ and $T \in \mathbb{R}^3$ represent the object's 3D rotation and 3D translation, respectively. We know

$$K = \begin{bmatrix} f & 0 & w/2 \\ 0 & f & h/2 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.2}$$

where f is the focal length, and w and h are the width and height of the image. Therefore, there are altogether seven parameters to be estimated: rotations θ, ϕ, ψ,

---

*https://structure.io
†https://occipital.com

translations x, y, z, and focal length f (Rotation matrix R is determined by θ, ϕ, and ψ).

To solve Equation 3.1, we first calculate a rough 3D pose using the Efficient PnP algorithm [Lepetit et al., 2009] and then refine it using the Levenberg-Marquardt algorithm [Levenberg, 1944, Marquardt, 1963], as shown in Figure 3-2. Details of each step are described below.

Efficient PnP. Perspective-n-Point (PnP) is the problem of estimating the pose of a calibrated camera given paired 3D points and 2D projections. The Efficient PnP (EPnP) algorithm solves the problem using virtual control points [Levenberg, 1944]. Because EPnP does not estimate the focal length, we enumerate the focal length f from 300 to 2,000 with a step size of 10, solve for the 3D pose with each f, and choose the one with the minimum projection error.

The Levenberg-Marquardt algorithm (LMA). We take the output of EPnP with 50 random disturbances as the initial states, and run LMA on each of them. Finally, we choose the solution with the minimum projection error.

Implementation details. For each 3D shape, we manually label its 3D keypoints. The number of keypoints ranges from 8 to 24. For each image, we ask three AMT workers to label if each keypoint is visible on the image, and if so, where it is. We only consider visible keypoints during the optimization.

The 2D keypoint annotations are noisy, which severely hurts the performance of the optimization algorithm. We try two methods to increase its robustness. The first is to use RANSAC. The second is to use only a subset of 2D keypoint annotations. For each image, denote $C = \{c_1, c_2, c_3\}$ as its three sets of human annotations. We then enumerate the seven nonempty subsets $C_k \subseteq C$; for each keypoint, we compute the median of its 2D coordinates in C_k. We apply our optimization algorithm on every subset C_k, and keep the output with the minimum projection error. After that, we let three AMT workers choose, for each image, which of the two methods offers better alignment, or neither performs well. At the same time, we also collect attributes (i.e., truncation, occlusion) for each image. Finally, we fine-tune the annotations ourselves using the GUI offered in ObjectNet3D [Xiang et al., 2016]. Altogether there are 395 3D shapes and 10,069 images. Sample 2D-3D pairs are shown in Figure 3-3.

3.4 Inverting a Graphics Engine by Modeling Surfaces

In this section we introduce MarrNet, a single-image 3D shape reconstruction algorithm using depth maps as an intermediate representation. As mentioned in Section 3.1, MarrNet is the basis of GenRe, the main algorithm of the chapter.

MarrNet contains three parts: first, a 2.5D sketch estimator, which predicts the depth, surface normal, and silhouette images of the object (Figure 3-4a); second,

| 3D Shape | Image | Alignment | 3D Shape | Image | Alignment | 3D Shape | Image | Alignment |

Figure 3-3: **Sample images and shapes in Pix3D.** From left to right: 3D shapes, 2D images, and 2D-3D alignment. Rows 1–2 show some chairs we scanned, rows 3–4 show a few IKEA objects, and rows 5–6 show some objects of other categories we scanned.

a 3D shape estimator, which infers 3D object shape using a voxel representation (Figure 3-4b); third, a reprojection consistency function, enforcing the alignment between the estimated 3D structure and inferred 2.5D sketches (Figure 3-4c).

3.4.1 2.5D Sketch Estimation

The first component of MarrNet (Figure 3-4a) takes a 2D RGB image as input, and predicts its 2.5D sketch: surface normal, depth, and silhouette. The goal of the 2.5D sketch estimation step is to distill intrinsic object properties from input images, while discarding properties that are non-essential for the task of 3D reconstruction, such as object texture and lighting.

We use an encoder-decoder network architecture for 2.5D sketch estimation. Our encoder is a ResNet-18 [He et al., 2016], encoding a 256×256 RGB image into 512 feature maps of size 8×8. The decoder contains four sets of 5×5 transposed convolutional and ReLU layers, followed by four sets of 1×1 convolutional and ReLU

43
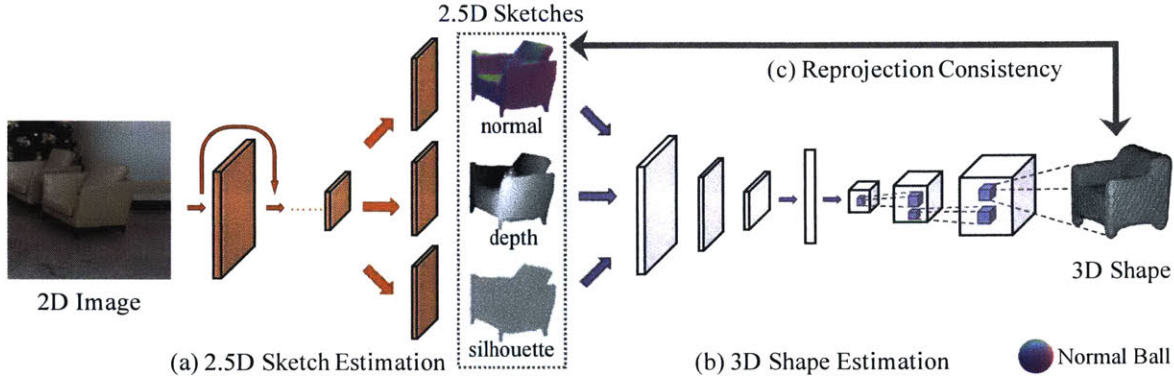
Figure 3-4: **Our model (MarrNet) has three major components**: (a) 2.5D sketch estimation, (b) 3D shape estimation, and (c) a loss function for reprojection consistency. MarrNet first recovers object normal, depth, and silhouette images from an RGB image. It then regresses the 3D shape from the 2.5D sketches. In both steps, it uses an encoding-decoding network. It finally employs a reprojection consistency loss to ensure the estimated 3D shape aligns with the 2.5D sketches. The entire framework can be trained end-to-end.

layers. It outputs the corresponding depth, surface normal, and silhouette images, also at the resolution of 256×256.

3.4.2 3D Shape Estimation

The second part of our framework (Figure 3-4b) infers 3D object shape from estimated 2.5D sketches. Here, the network focuses on learning the shape prior that explains input well. As it takes only surface normal and depth images as input, it can be trained on synthetic data, without suffering from the domain adaption problem: it is straightforward to render nearly perfect 2.5D sketches, but much harder to render realistic images.

The network architecture is inspired by the TL network [Girdhar et al., 2016], and the 3D-VAE-GAN [Wu et al., 2016c], again with an encoding-decoding style. It takes a normal image and a depth image as input (both masked by the estimated silhouette), maps them to a 200-dim vector via five sets of convolutional, ReLU, and pooling layers, followed by two fully connected layers. The detailed encoder structure can be found in Girdhar et al. [2016]. The vector then goes through a decoder, which consists of five transposed convolutional and ReLU layers to output a 128×128×128 voxel-based reconstruction of the input. The detailed decoder structure can be found in Wu et al. [2016c].

3.4.3 Reprojection Consistency

There have been some attempts to enforce the consistency between estimated 3D shape and 2D representations in a neural network [Yan et al., 2016b, Rezende et al., 2016a, Wu et al., 2016c, Tulsiani et al., 2017b]. Here, we explore novel ways to include

44
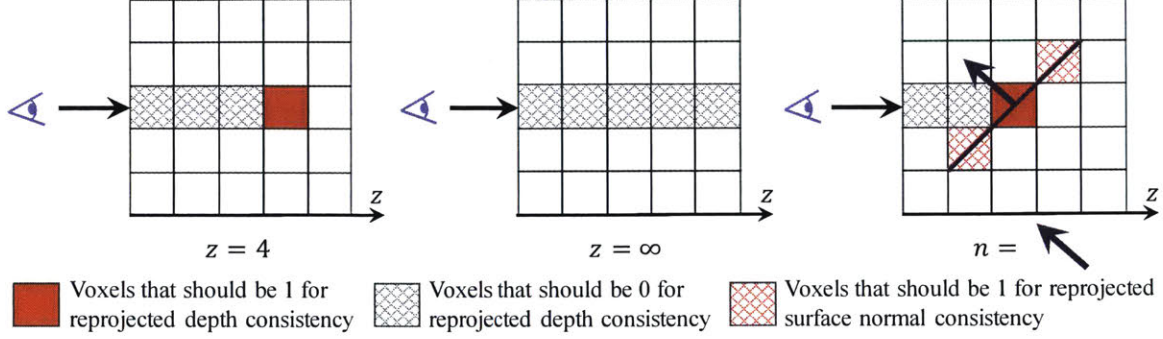
Figure 3-5: **Reprojection consistency between 2.5D sketches and 3D shape.** Left and middle: the criteria for depths and silhouettes; right: the criterion for surface normals. See Section 3.4.3 for details.

a reprojection consistency loss between the predicted 3D shape and the estimated 2.5D sketch, consisting of a depth reprojection loss and a surface normal reprojection loss.

We use $v_{x,y,z}$ to represent the value at position (x, y, z) in a 3D voxel grid, assuming that $v_{x,y,z} \in [0, 1]$, $\forall x, y, z$. We use $d_{x,y}$ to denote the estimated depth at position (x, y), and $n_{x,y} = (n_a, n_b, n_c)$ to denote the estimated surface normal. We assume orthographic projection in this work.

Depths. The projected depth loss tries to guarantee that the voxel with depth $v_{x,y,d_{x,y}}$ should be 1, and all voxels in front of it should be 0. This ensures that the estimated 3D shape matches the estimated depth values.

As illustrated in Figure 3-5a, we define projected depth loss as follows:

$$
L_{\text{depth}}(x, y, z) = \begin{cases} v_{x,y,z}^2, & z < d_{x,y} \\ (1 - v_{x,y,z})^2, & z = d_{x,y} \\ 0, & z > d_{x,y} \end{cases} \tag{3.3}
$$

The gradients are

$$
\frac{\partial L_{\text{depth}}(x, y, z)}{\partial v_{x,y,z}} = \begin{cases} 2v_{x,y,z}, & z < d_{x,y} \\ 2(v_{x,y,z} - 1), & z = d_{x,y} \\ 0, & z > d_{x,y} \end{cases} \tag{3.4}
$$

When $d_{x,y} = \infty$, our depth criterion reduces to a special case — the silhouette criterion. As shown in Figure 3-5b, for a line that has no intersection with the shape, all voxels in it should be 0.

Surface normals. As vectors $n_x = (0, -n_c, n_b)$ and $n_y = (-n_c, 0, n_a)$ are orthogonal to the normal vector $n_{x,y} = (n_a, n_b, n_c)$, we can normalize them to obtain two vectors,

45

$n'_x = (0, -1, n_b/n_c)$ and $n'_y = (-1, 0, n_a/n_c)$, both on the estimated surface plane at (x, y, z). The projected surface normal loss tries to guarantee that the voxels at $(x, y, z) \pm n'_x$ and $(x, y, z) \pm n_{+y}$ should be 1 to match the estimated surface normals. These constraints only apply when the target voxels are inside the estimated silhouette.

As shown in Figure 3-5c, let $z = d_{x,y}$, the projected surface normal loss is defined as

$$L_{\text{normal}}(x, y, z) = \left(1 - v_{x,y-1,z+\frac{n_b}{n_c}}\right)^2 + \left(1 - v_{x,y+1,z-\frac{n_b}{n_c}}\right)^2 +$$
$$\left(1 - v_{x-1,y,z+\frac{n_a}{n_c}}\right)^2 + \left(1 - v_{x+1,y,z-\frac{n_a}{n_c}}\right)^2 . \tag{3.5}$$

Then the gradients along the x direction are

$$\frac{\partial L_{\text{normal}}(x, y, z)}{\partial v_{x-1,y,z+\frac{n_a}{n_c}}} = 2 \left(v_{x-1,y,z+\frac{n_a}{n_c}} - 1\right) \text{ and } \frac{\partial L_{\text{normal}}(x, y, z)}{\partial v_{x+1,y,z-\frac{n_a}{n_c}}} = 2 \left(v_{x+1,y,z-\frac{n_a}{n_c}} - 1\right).$$
$$\tag{3.6}$$

The gradients along the y direction are similar.

3.5 Integrating 3D Shape Priors

In this section, we introduce algorithms that learn 3D shape priors—the underlying distribution of 3D shapes. The learned shape priors can be used to synthesize new shapes; they can also be integrated into models such as MarrNet for better results on shape reconstruction.

3.5.1 Learning Priors with 3D Generative Adversarial Networks

As proposed in Goodfellow et al. [2014], the Generative Adversarial Network (GAN) consists of a generator and a discriminator, where the discriminator tries to classify real objects and objects synthesized by the generator, and the generator attempts to confuse the discriminator. Here, we introduce 3D Generative Adversarial Networks (3D-GAN), leveraging GANs for 3D shape generation. The generator G in 3D-GAN maps a 200-dimensional latent vector z, randomly sampled from a probabilistic latent space, to a $64 \times 64 \times 64$ cube, representing an object $G(z)$ in 3D voxel space. The discriminator D outputs a confidence value $D(x)$ of whether a 3D object input x is real or synthetic.

Following Goodfellow et al. [2014], we use binary cross entropy as the classification loss, and present our overall adversarial loss function as

$$L_{\text{3D-GAN}} = \log D(x) + \log(1 - D(G(z))), \tag{3.7}$$

where x is a real object in a $64 \times 64 \times 64$ space, and z is a randomly sampled noise
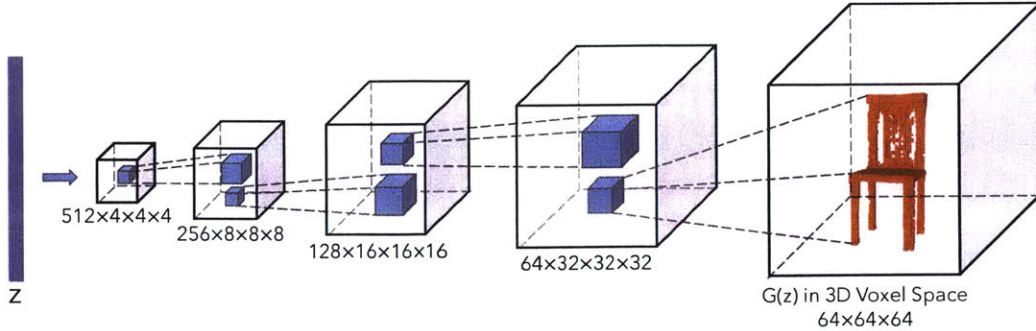
Figure 3-6: **The generator in 3D-GAN.** The discriminator mostly mirrors the generator.

vector from a distribution $p(z)$. In this work, each dimension of z is an i.i.d. uniform distribution over $[0, 1]$.

Network structure. Inspired by Radford et al. [2016], we design an all-convolutional neural network to generate 3D objects. As shown in Figure 3-6, the generator consists of five volumetric transposed convolutional layers with kernel size $4 \times 4 \times 4$ and stride 2, with batch normalization and ReLU layers added in between and a sigmoid layer at the end. The discriminator basically mirrors the generator, except that it uses Leaky ReLU [Maas et al., 2013] instead of ReLU layers. There are no pooling or linear layers in our network.

Training details. A straightforward training procedure is to update both the generator and the discriminator in every batch. However, the discriminator usually learns much faster than the generator, possibly because generating objects in a 3D voxel space is more difficult than differentiating between real and synthetic objects [Goodfellow et al., 2014, Radford et al., 2016]. It then becomes hard for the generator to extract signals for improvement from a discriminator that is way ahead, as all examples it generated would be correctly identified as synthetic with high confidence. Therefore, to keep the training of both networks in pace, we employ an adaptive training strategy: for each batch, the discriminator only gets updated if its accuracy in the last batch is not higher than 80%. We observe this helps to stabilize the training and to produce better results. We set the learning rate of G to 0.0025, D to 10^{-5}, and use a batch size of 100. We use ADAM [Kingma and Ba, 2015] for optimization, with $\beta = 0.5$.

Results on 3D shape generation. Figure 3-7 shows 3D objects generated by our 3D-GAN. For this experiment, we train one 3D-GAN for each object category. For generation, we sample 200-dimensional vectors following an i.i.d. uniform distribution over $[0, 1]$, and render the largest connected component of each generated object. We compare 3D-GAN with Wu et al. [2015b], the state-of-the-art in 3D object synthesis from a probabilistic space, and with a volumetric autoencoder, whose variants have been employed by multiple recent methods [Girdhar et al., 2016, Sharma et al., 2016].
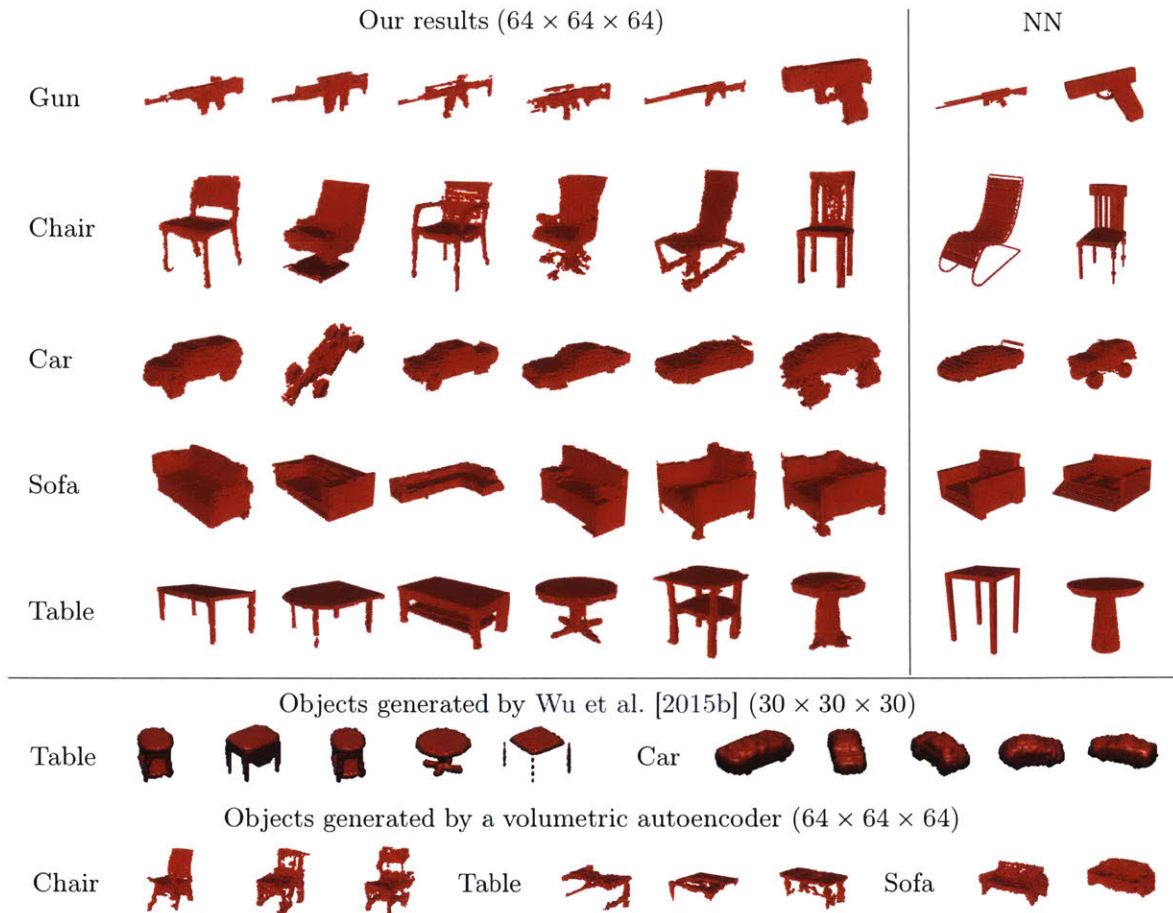
Figure 3-7: **Shapes generated by 3D-GAN from vectors**, without a reference image/shape. We show, for the last two shapes in each row, the nearest neighbor retrieved from the training set. We see that the generated shapes are similar, but not identical, to examples in the training set. For comparison, we show shapes generated by the previous state-of-the-art [Wu et al., 2015b] (results supplied by the authors). We also show shapes generated by autoencoders trained on a single category, with latent vectors sampled from empirical distribution. See text for details.

Because an autoencoder does not restrict the distribution of its latent representation, we compute the empirical distribution $p_0(z)$ of the latent vector z of all training examples, fit a Gaussian distribution g_0 to p_0, and sample from g_0. Our algorithm produces 3D objects with much higher quality and more fine-grained details.

Compared with previous methods, our 3D-GAN can synthesize high-resolution 3D objects with detailed geometries. Figure 3-8 shows both high-res voxels and down-sampled low-res voxels for comparison. Note that it is relatively easy to synthesize a low-res object, but is much harder to obtain a high-res one due to the rapid growth of 3D space. However, object details are only revealed in high resolution.

A natural concern to our generative model is whether it is simply memorizing objects from training data. To demonstrate that the network can generalize beyond the training set, we compare synthesized objects with their nearest neighbor in the

High-res   Low-res   High-res   Low-res   High-res   Low-res   High-res   Low-res
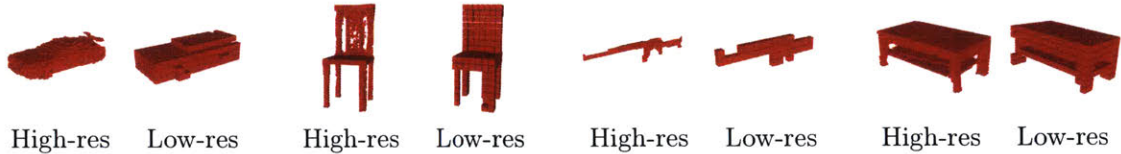
Figure 3-8: **High-resolution shapes.** We present each shape at high resolution ($64 \times 64 \times 64$) on the left and at low resolution (down-sampled to $16 \times 16 \times 16$) on the right. While humans can perceive object structure at a relatively low resolution, fine details and variations only appear in high-res objects.
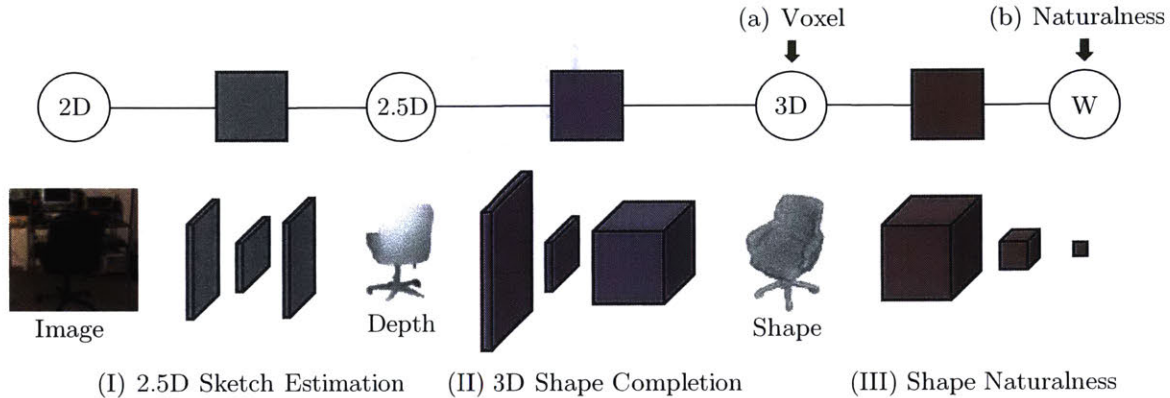
Figure 3-9: **ShapeHD contains three components**: (I) a 2.5D sketch estimator that predicts depth, surface normal and silhouette images from a single image; (II) a 3D shape completion module that regresses 3D shapes from silhouette-masked depth and surface normal images; (III) an adversarially pretrained convolutional net that serves as the naturalness loss function. While fine-tuning the 3D shape completion net, we use two losses: a supervised loss on the output shape, and a naturalness loss offered by the pretrained discriminator.

training set. Since the retrieval objects based on ℓ^2 distance in the voxel space are visually very different from the queries, we use the output of the last convolutional layer in our discriminator (with a 2x pooling) as features for retrieval instead. Figure 3-7 shows that generated objects are similar, but not identical, to the nearest examples in the training set.

3.5.2   Integrating Priors into Reconstruction Models

We now present ShapeHD, a single-image 3D reconstruction model that extends MarrNet by incorporating shape priors learned by 3D-GAN.

ShapeHD consists of three components: a 2.5D sketch estimator and a 3D shape estimator that jointly predict a 3D shape from an RGB image via 2.5D sketches (Figure 3-9-I and Figure 3-9-II, inspired by MarrNet Wu et al. [2017c]), and a deep naturalness model that penalizes the shape estimator if the predicted shape is unnatural (Figure 3-9-III). Models trained with a supervised reconstruction loss alone often generate blurry mean shapes. Our learned naturalness model helps to avoid this issue.
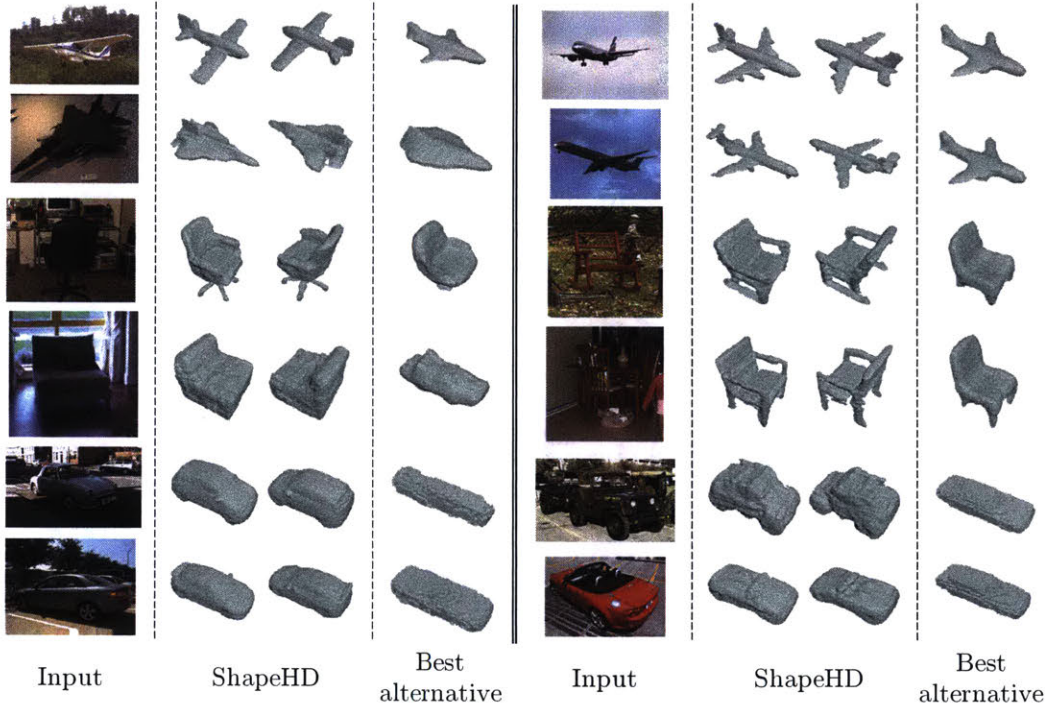
Figure 3-10: **Reconstruction results of ShapeHD on PASCAL 3D+** [Xiang et al., 2014]. From left to right: input, two views of reconstructions from ShapeHD, and reconstructions by the best alternative in Table 3.1. Assisted by the learned naturalness losses, ShapeHD recovers accurate 3D shapes with fine details.

We pre-train a 3D generative adversarial network [Wu et al., 2016c] to determine whether a shape is realistic. Its generator synthesizes a 3D shape from a randomly sampled vector, and its discriminator distinguishes generated shapes from real ones. Therefore, the discriminator has the ability to model the real shape distribution and can be used as a naturalness loss for the shape completion network. The generator is not involved in our later training process. Following 3D-GAN, we use 5 transposed convolutional layers with batch normalization and ReLU for the generator, and 5 convolutional layers with leaky ReLU for the discriminator.

Due to the high dimensionality of 3D shapes ($128 \times 128 \times 128$), training a GAN becomes highly unstable. To deal with this issue, we follow Gulrajani et al. [2017] and use the Wasserstein GAN loss with a gradient penalty to train our adversarial generative network. Specifically,

$$L_{\mathrm{WGAN}} = \mathop{\mathbb{E}}_{\tilde{x} \sim P_g}[D(\tilde{x})] - \mathop{\mathbb{E}}_{x \sim P_r}[D(x)] + \lambda \mathop{\mathbb{E}}_{\hat{x} \sim P_x}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2], \qquad (3.8)$$

where D is the discriminator, P_g and P_r are distributions of generated shapes and real shapes, respectively. The last term is the gradient penalty from Gulrajani et al. [2017]. During training, the discriminator attempts to minimize the overall loss L_{WGAN}

(a) Input   (b) AtlasNet   (c) DRC (3D)   (d) ShapeHD   (e) GT
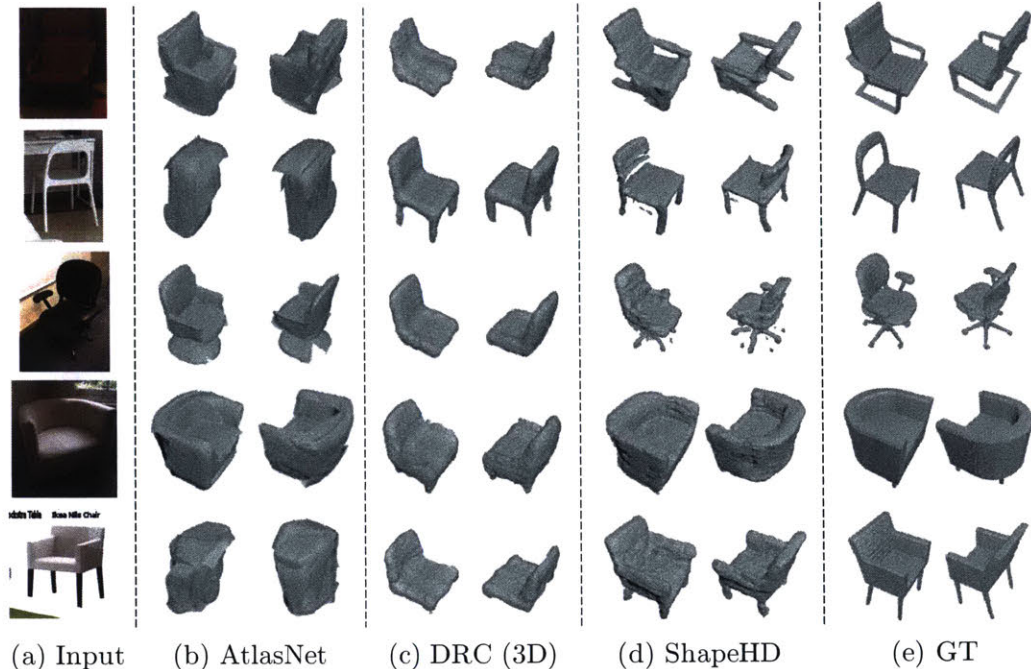
Figure 3-11: **Reconstruction results of ShapeHD on Pix3D** [Sun et al., 2018b]. For each input image, we show reconstructions by AtlasNet, DRC, our ShapeHD, and ground truth. Our ShapeHD reconstructs complete 3D shapes with fine details that resemble the ground truth.

while the generator attempts to maximize the loss via the first term in Equation 3.8, so we can define our naturalness loss as $L_{\text{natural}} = - \mathbb{E}_{\tilde{x} \sim P_c} [D(\tilde{x})]$, where P_c are the reconstructed shapes from our completion network.

Results on shape reconstruction. We evaluate on two real datasets, PASCAL 3D+ [Xiang et al., 2014] and Pix3D [Sun et al., 2018b]. Here, we train our model on synthetic ShapeNet renderings and use the pre-trained models released by the authors as baselines. All methods take ground truth 3D shapes as supervision during training. As shown in Figures 3-10 and 3-11, ShapeHD works well, inferring a reasonable shape even in the presence of strong self-occlusions. In particular, in Figure 3-10, we compare our reconstructions with the best-performing alternatives (DRC on chairs and airplanes, and AtlasNet on cars). In addition to preserving details, our model captures the shape variations of the objects, while the competitors produce similar reconstructions across instances.

Quantitatively, Tables 3.1 and 3.2 suggest that ShapeHD performs significantly better than the other methods in almost all metrics. The only exception is the CD on PASCAL 3D+ cars, where OGN performs the best. However, as PASCAL 3D+ only has around 10 CAD models for each object category as ground truth 3D shapes, the ground truth labels and the scores can be inaccurate, failing to reflect human perception [Tulsiani et al., 2017b].

51

Methods	CD			
	chair	car	plane	avg
3D-R2N2 [Choy et al., 2016]	0.238	0.305	0.305	0.284
DRC (3D) [Tulsiani et al., 2017b]	0.158	0.099	0.112	0.122
OGN [Tatarchenko et al., 2017]	-	**0.087**	-	-
ShapeHD (ours)	**0.137**	0.129	**0.094**	**0.119**

(a) CDs on PASCAL 3D+ [Xiang et al., 2014]
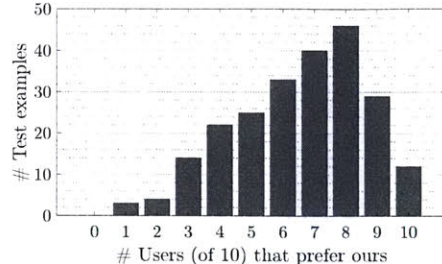
(b) Human Study results

Table 3.1: **Reconstruction errors of ShapeHD on the PASCAL 3D+ dataset** [Xiang et al., 2014]. (a) We compare our ShapeHD with 3D-R2N2, DRC, and OGN. PSGN and AtlasNet are not evaluated, because they require object masks as additional input, but PASCAL 3D+ has only inaccurate masks. (b) In the behavioral study, most users prefer our constructions on most images. Overall, our reconstructions are preferred 64.5% of the time to OGN's.

	3D-R2N2	DRC (3D)	PSGN*	AtlasNet*	ShapeHD
IoU (32^3)	0.136	0.265	-	-	**0.284**
IoU (128^3)	0.089	0.185	-	-	**0.205**
CD	0.239	0.160	0.199	0.126	**0.123**

Table 3.2: **Reconstruction errors of ShapeHD on the Pix3D dataset** [Sun et al., 2018b]. All methods were trained with full 3D supervision on rendered images of ShapeNet objects. *While 3D-R2N2, DRC, and ShapeHD take a single image as input, PSGN and AtlasNet additionally require the ground truth mask as input. Also, PSGN and AtlasNet generate surface point clouds without guaranteeing watertight meshes and therefore cannot be evaluated in IoU.

We therefore conduct an additional user study, where we show an input image and its two reconstructions (from ShapeHD and from OGN, each in two views) to users on Amazon Mechanical Turk, and ask them to choose the shape that looks closer to the object in the image. For each image, we collect 10 responses from "Masters" (workers who have demonstrated excellence across a wide range of HITs). Table 3.1b suggests that on most images, most users prefer our reconstruction to OGN's. In general, our reconstructions are preferred 64.5% of the time.

3.6 Generalizable Reconstruction

In this section, we present our main contribution of this chapter, Generalizable Reconstruction (GenRe), a single-image 3D reconstruction algorithm that generalizes to unseen classes. As discussed earlier, single-image reconstruction algorithms learn a parametric function $f_{2D \to 3D}$ that maps a 2D image to a 3D shape. We tackle the problem of generalization by regularizing $f_{2D \to 3D}$. The key regularization we impose is to factorize $f_{2D \to 3D}$ into geometric projections and learnable reconstruction modules.

Our GenRe model consists of three learnable modules, connected by geometric
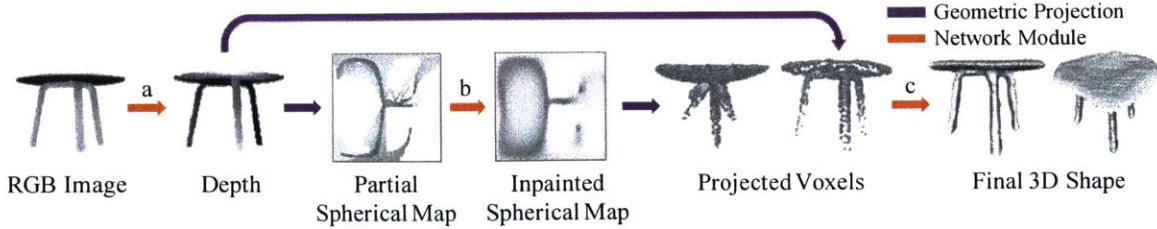
Figure 3-12: **Our model for generalizable single-image 3D reconstruction (GenRe)** has three components: (a) a depth estimator that predicts depth in the original view from a single RGB image, (b) a spherical inpainting network that inpaints a partial, single-view spherical map, and (c) a voxel refinement network that integrates two backprojected 3D shapes (from the inpainted spherical map and from depth) to produce the final output.

projections as shown in Figure 3-12. As in MarrNet and ShapeHD, the first module is a single-view depth estimator $f_{2D \rightarrow 2.5D}$ (Figure 3-12a), taking a color image as input and estimates its depth map. As the depth map can be interpreted as the visible surface of the object, the reconstruction problem becomes predicting the object's complete surface given this partial estimate.

As 3D surfaces are hard to parametrize efficiently, we use spherical maps as a surrogate representation. A geometric projection module ($p_{2.5D \rightarrow S}$) converts the estimated depth map into a spherical map, referred to as the partial spherical map. It is then passed to the spherical map inpainting network ($c_{S \rightarrow S}$, Figure 3-12b) to predict an inpainted spherical map, representing the object's complete surface. Another projection module ($p_{S \rightarrow 3D}$) projects the inpainted spherical map back to the voxel space.

As spherical maps only capture the outermost surface toward the sphere, they cannot handle self-occlusion along the sphere's radius. We use a voxel refinement module (Figure 3-12c) to tackle this problem. It takes two 3D shapes as input, one projected from the inpainted spherical map and the other from the estimated depth map, and outputs a final 3D shape.

3.6.1 Single-View Depth Estimator

The first component of our network predicts a depth map from an image with a clean background. Using depth as an intermediate representation facilitates the reconstruction process by distilling essential geometric information from the input image [Wu et al., 2017c].

Further, depth estimation is a class-agnostic task: shapes from different classes often share common geometric structure, despite distinct visual appearances. Take beds and cabinets as examples. Although they are of different anatomy in general, both have perpendicular planes and hence similar patches in their depth images. We demonstrate this both qualitatively and quantitatively in Section 3.7.4.

53
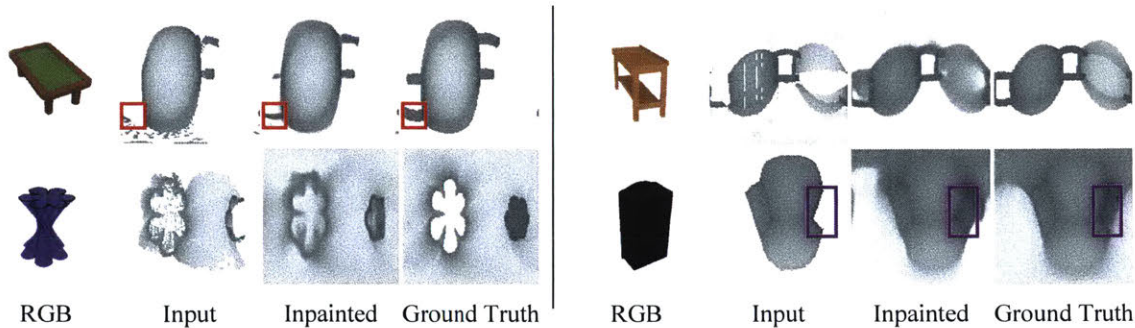
Figure 3-13: **Examples of our spherical inpainting module generalizing to new classes.** Trained on chairs, cars, and planes, the module completes the partially visible leg of the table (red boxes) and the unseen cabinet bottom (purple boxes) from partial spherical maps projected from ground-truth depth.

3.6.2   Spherical Map Inpainting Network

With spherical maps, we cast the problem of 3D surface completion into 2D spherical map inpainting. Empirically we observe that networks trained to inpaint spherical maps generalize well to new shape classes (Figure 3-13). Also, compared with voxels, spherical maps are more efficient to process, as 3D surfaces are sparse in nature; quantitatively, as we demonstrate in Section 3.7.5 and Section 3.7.6, using spherical maps results in better performance.

As spherical maps are signals on the unit sphere, it is tempting to use network architectures based on spherical convolution [Cohen et al., 2018]. They are however not suitable for our task of shape reconstruction. This is because spherical convolution is conducted in the spectral domain. Every conversion to and from the spectral domain requires capping the maximum frequency, causing extra aliasing and information loss. For tasks such as recognition, the information loss may be negligible compared with the advantage of rotational invariance offered by spherical convolution. But for reconstruction, the loss leads to blurred output with only low-frequency components. We empirically find that standard convolution works much better than spherical convolution under our setup.

3.6.3   Voxel Refinement Network

Although an inpainted spherical map provides a projection of an object's surface onto the unit sphere, the surface information is lost when self-occlusion occurs. We use a refinement network that operates in the voxel space to recover the lost information. This module takes two voxelized shapes as input, one projected from the estimated depth map and the other from the inpainted spherical map, and predicts the final shape. As the occluded regions can be recovered from local neighboring regions, this network only needs to capture local shape priors and is therefore class-agnostic. As shown in the experiments, when provided with ground-truth depth and spherical maps,

this module performs consistently well across training and unseen classes.

3.6.4 Technical Details

Single-view depth estimator. Following Wu et al. [2017c], we use an encoder-decoder network for depth estimation. Our encoder is a ResNet-18 [He et al., 2016], encoding a 256×256 RGB image into 512 feature maps of size 1×1. The decoder is a mirrored version of the encoder, replacing all convolution layers with transposed convolution layers. In addition, we adopt the U-Net structure [Ronneberger et al., 2015] and feed the intermediate outputs of each block of the encoder to the corresponding block of the decoder. The decoder outputs the depth map in the original view at the resolution of 256×256. We use an ℓ_2 loss between predicted and target images.

Spherical map inpainting network. The spherical map inpainting network has a similar architecture as the single-view depth estimator. To reduce the gap between standard and spherical convolutions, we use periodic padding to both inputs and training targets in the longitude dimension, making the network aware of the periodic nature of spherical maps.

Voxel refinement network. Our voxel refinement network takes as input voxels projected from the estimated, original-view depth and from the inpainted spherical map, and recovers the final shape in voxel space. Specifically, the encoder takes as input a two-channel 128×128×128 voxel grid (one for coarse shape estimation and the other for surface estimation), and outputs a 320-D latent vector. In decoding, each layer takes an extra input directly from the corresponding level of the encoder.

Geometric projections. We make use of three geometric projections: a depth to spherical map projection, a depth map to voxel projection, and a spherical map to voxel projection. For the depth to spherical map projection, we first convert depth into 3D point clouds using camera parameters, and then turn them into surfaces with the marching cubes algorithm [Lewiner et al., 2003]. Then, the spherical representation is generated by casting rays from each UV coordinate on the unit sphere to the sphere's center. This process is not differentiable. To project depth or spherical maps into voxels, we first convert them into 3D point clouds. Then, a grid of voxels is initialized, where the value of each voxel is determined by the average distance between all the points inside it to its center. Then, for all the voxels that contain points, we negate its value and add it by 1. This projection process is fully differentiable.

Training. We train our network with viewer-centered 3D supervision, where the 3D shape is rotated to match the object's pose in the input image. This is in contrast to object-centered approaches, where the 3D supervision is always in a predefined pose regardless of the object's pose in the input image. Object-centered approaches are

less suitable for reconstructing shapes from new categories, as predefined poses are unlikely to generalize across categories.

We first train the 2.5D sketch estimator with RGB images and their corresponding depth images, all rendered with ShapeNet [Chang et al., 2015] objects (see Section 3.7.2 and Appendix A for details). We then train the spherical map inpainting network with single-view (partial) spherical maps and the ground-truth full spherical maps as supervision. Finally, we train the voxel refinement network on coarse shapes predicted by the inpainting network as well as 3D surfaces backprojected from the estimated 2.5D sketches, with the corresponding ground-truth shapes as supervision. We then jointly fine-tune the spherical inpainting module and the voxel refinement module with both 3D shape and 2D spherical map supervision.

3.7 Experiments

3.7.1 Baselines

We organize baselines based on the shape representation they use.

Voxels. Voxels are arguably the most common representation for 3D shapes in the deep learning era due to their amenability to 3D convolution. For this representation, we consider DRC [Tulsiani et al., 2017b] and MarrNet [Wu et al., 2017c] as baselines. Our model uses 128^3 voxels of $[0, 1]$ occupancy.

Mesh and point clouds. Considering the cubic complexity of the voxel representation, recent papers have explored meshes [Groueix et al., 2018, Yao et al., 2018] and point clouds [Fan et al., 2017] in the context of neural networks. In this work, we consider AtlasNet [Groueix et al., 2018] as a baseline.

Multi-view maps. Another way of representing 3D shapes is to use a set of multi-view depth images [Soltani et al., 2017, Shin et al., 2018, Jayaraman et al., 2018]. We compare with the model from Shin et al. [2018] in this regime.

Spherical maps. As introduced in Section 3.1, one can also represent 3D shapes as spherical maps. We include two baselines with spherical maps: first, a one-step baseline that predicts final spherical maps directly from RGB images (GenRe-1step); second, a two-step baseline that first predicts single-view spherical maps from RGB images and then inpaints them (GenRe-2step). Both baselines use the aforementioned U-ResNet image-to-image network architecture.

To provide justification for using spherical maps, we provide a baseline (3D Completion) that directly performs 3D shape completion in voxel space. This baseline first predicts depth from an input image; it then projects the depth map into the voxel space. A completion module takes the projected voxels as input and predicts the final result.

To provide a performance upper bound for our spherical inpainting and voxel refinement networks (b and c in Figure 3-12), we also include the results when our model has access to ground-truth depth in the original view (GenRe-Oracle) and to ground-truth full spherical maps (GenRe-SphOracle).

3.7.2 Data

We use ShapeNet [Chang et al., 2015] renderings for network training and testing. Specifically, we render each object in 20 random views. In addition to RGB images, we also render their corresponding ground-truth depth maps. We use Mitsuba [Jakob, 2010], a physically-based rendering engine, for all our renderings. Please see Appendix A for details on data generation and augmentation.

For all models, we train them on the three largest ShapeNet classes (cars, chairs, and airplanes), and test them on the next 10 largest classes: bench, vessel, rifle, sofa, table, phone, cabinet, speaker, lamp, and display. Besides ShapeNet renderings, we also test these models, trained only on synthetic data, on real images from Pix3D [Sun et al., 2018b], a dataset of real images and the ground-truth shape of every pictured object. In Section 3.8, we also test our model on non-rigid shapes such as humans and horses [Bronstein et al., 2008] and on highly regular shape primitives.

3.7.3 Metrics

Because neither depth maps nor spherical maps provide information inside shapes, our model predicts only surface voxels that are not guaranteed watertight. Consequently, intersection over union (IoU) cannot be used as an evaluation metric. We hence evaluate reconstruction quality using Chamfer distance (CD) [Barrow et al., 1977], defined as

$$\mathrm{CD}(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2, \tag{3.9}$$

where S_1 and S_2 are sets of points sampled from surfaces of the 3D shape pair. For models that output voxels, including DRC and our GenRe model, we sweep voxel thresholds from 0.3 to 0.7 with a step size of 0.05 for isosurfaces, compute CD with 1,024 points sampled from all isosurfaces, and report the best average CD for each object class.

Shin et al. [2018] reported that object-centered supervision produces better reconstructions for objects from the training classes, whereas viewer-centered supervision is advantaged in generalizing to novel classes. Therefore, for DRC and AtlasNet, we train each network with both types of supervision. Note that AtlasNet, when trained with viewer-centered supervision, tends to produce unstable predictions that render CD meaningless. Hence, we only present CD for the object-centered AtlasNet.
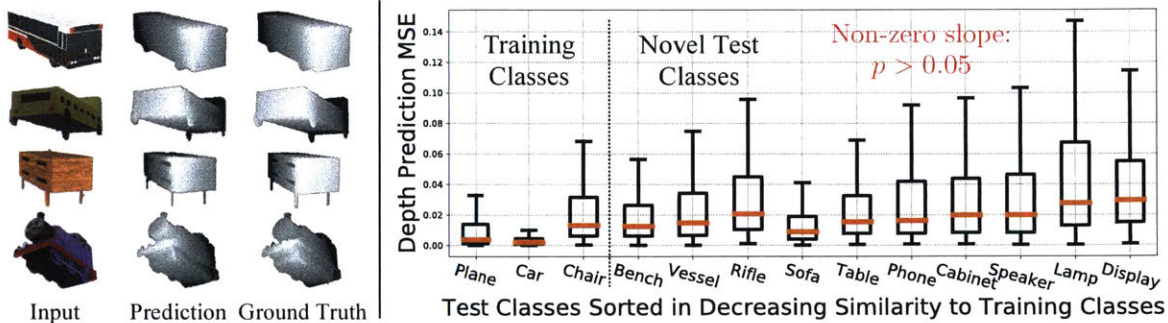
Figure 3-14: **Results on depth prediction.** Left: Our single-view depth estimator, trained on cars, chairs, and airplanes, generalizes to novel classes: buses, trains, and tables. Right: As the novel test class gets increasingly dissimilar to the training classes (left to right), depth prediction does not show statistically significant degradation ($p > 0.05$).

	Models	Seen	Unseen										
			Bch	Vsl	Rfl	Sfa	Tbl	Phn	Cbn	Spk	Lmp	Dsp	Avg
Object-Centered	DRC	.072	.112	.100	.104	.108	.133	.199	.168	.164	.145	.188	.142
	AtlasNet	**.059**	.102	**.092**	**.088**	.098	.130	.146	.149	.158	.131	.173	.127
Viewer-Centered	DRC	.092	.120	.109	.121	.107	.129	.132	.142	.141	.131	.156	.129
	MarrNet	.070	.107	.094	.125	.090	.122	.117	.125	.123	.144	.149	.120
	Multi-View	.065	.092	**.092**	.102	.085	.105	.110	.119	.117	.142	.142	.111
	3D Completion	.076	.102	.099	.121	.095	.109	.122	.131	.126	.138	.141	.118
	GenRe-1step	.063	.104	.093	.114	.084	.108	.121	.128	.124	.126	.151	.115
	GenRe-2step	.061	.098	.094	.117	.084	.102	.115	.125	.125	**.118**	**.118**	.110
	GenRe (Ours)	.064	**.089**	**.092**	.112	**.082**	**.096**	**.107**	**.116**	**.115**	.124	.130	**.106**
	GenRe-Oracle	.045	.050	.048	.031	.059	.057	.054	.076	.077	.060	.060	.057
	GenRe-SphOracle	.034	.032	.030	.021	.044	.038	.037	.044	.045	.031	.040	.036

Table 3.3: **Reconstruction errors (in CD) of the training classes and 10 novel classes from ShapeNet**, ordered from the most to the least similar to the training classes. We compare with DRC [Tulsiani et al., 2017b], AtlasNet [Groueix et al., 2018], MarrNet [Wu et al., 2017c], and Multi-View representations [Shin et al., 2018]. Our model is viewer-centered by design, but achieves performance on par with the object-centered state of the art (AtlasNet) in reconstructing the seen classes. As for generalization to novel classes, our model outperforms the state of the art across 9 out of the 10 classes.

3.7.4 Results on Depth Estimation

We show qualitative and quantitative results on depth estimation quality across categories. As shown in Figure 3-14, our depth estimator learns effectively the concept of near and far, generalizes well to unseen categories, and does not show statistically significant deterioration as the novel test class gets increasingly dissimilar to the training classes, laying the foundation for the generalization power of our approach. Formally, the dissimilarity from test class C_{test} to training classes C_{train} is defined as $(1/|C_{\text{test}}|) \sum_{x \in C_{\text{test}}} \min_{y \in C_{\text{train}}} \text{CD}(x, y)$.
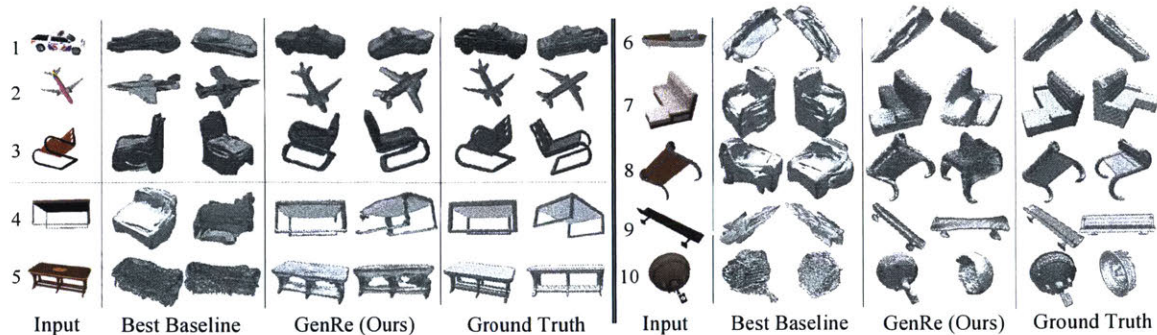
Figure 3-15: **Single-image 3D reconstructions of objects within and beyond training classes.** Each row from left to right: the input image, two views from the best-performing baseline for each testing object (1-4, 6-9: AtlasNet; 5, 10: Shin et al. [2018]), two views of our GenRe predictions, and the ground truth. All models are trained on the same dataset of cars, chairs, and airplanes.

3.7.5 Reconstructing Novel Objects from Training Classes

We present results on generalizing to novel objects from the training classes. All models are trained on cars, chairs, and airplanes, and tested on unseen objects from the same three categories.

As shown in Table 3.3, our GenRe model is the best-performing viewer-centered model. It also outperforms most object-centered models except AtlasNet. GenRe's preformance is impressive given that object-centered models tend to perform much better on objects from seen classes [Shin et al., 2018]. This is because object-centered models, by exploiting the concept of canonical views, actually solve an easier problem. The performance drop from object-centered DRC to viewer-centered DRC supports this empirically. However, for objects from unseen classes, the concept of canonical views is no longer well-defined. As we will see in Section 3.7.6, this hurts the generalization power of object-centered methods.

3.7.6 Reconstructing Objects from Unseen Classes

We study how our approach enables generalization to novel shape classes that were not seen during training.

Synthetic renderings. We use the 10 largest ShapeNet classes other than chairs, cars, and airplanes as our test set. Table 3.3 shows that our model consistently outperforms the state of the art, except for the class of rifles, in which AtlasNet performs the best. Qualitatively, our model produces reconstructions that are much more consistent with input images, as shown in Figure 3-15. In particular, on unseen classes, our results still attain good consistency with the input images, while the competitors either lack structural details present in the input (e.g., 5) or retrieve shapes from the training classes (e.g., 4, 6, 7, 8, 9).

59

	AtlasNet	Shin et al.	GenRe
Chair	**.080**	.089	.093
Bed	.114	**.106**	.113
Bookcase	.140	.109	**.101**
Desk	.126	.121	**.109**
Sofa	.095	.088	**.083**
Table	.134	.124	**.116**
Wardrobe	.121	.116	**.109**

Table 3.4: **Reconstruction errors (in CD) for seen (chairs) and unseen classes (the rest) on real Pix3D images**. GenRe outperforms baselines across all unseen classes except beds. For chairs, object-centered AtlasNet performs the best by leveraging the canonical view.
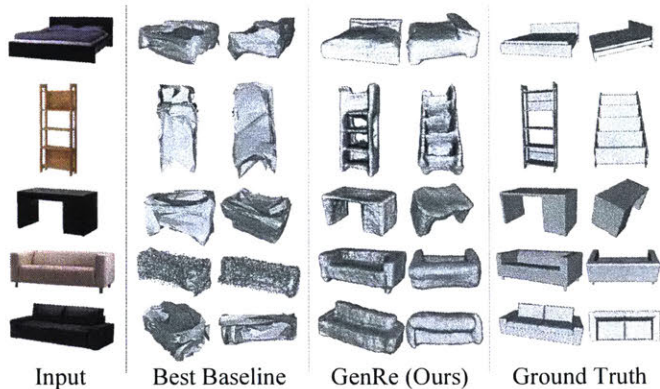
Figure 3-16: **Reconstructions on real images from Pix3D** by GenRe and AtlasNet or Shin et al. [2018]. All models are trained on cars, chairs, and airplanes.

Comparing our model with its variants, we find that the two-step approaches (GenRe-2step and GenRe) outperform the one-step approach across all novel categories. This empirically supports the advantage of our two-step modeling strategy that disentangles geometric projections from shape reconstruction.

Real images. We further compare how our model, AtlasNet, and Shin et al. [2018] perform on real images from Pix3D. Here, all models are trained on ShapeNet cars, chairs, and airplanes, and tested on real images of beds, bookcases, desks, sofas, tables, and wardrobes.

Quantitatively, Table 3.4 shows that our model outperforms the two competitors across all novel classes except beds, for which the model by Shin et al. [2018] performs the best. For chairs, one of the training classes, the object-centered AtlasNet leverages the canonical view and outperforms the two viewer-centered approaches. Qualitatively, our reconstructions preserve the details present in the input (e.g., the hollow structures in the second row of Figure 3-16).

3.8 Analyses

3.8.1 The Effect of Viewpoints on Generalization

The generic viewpoint assumption states that the observer is not in a special position relative to the object [Freeman, 1994]. This makes us wonder if the "accidentalness" of the viewpoint affects the quality of reconstructions.

As a quantitative analysis, we test our model trained on ShapeNet chairs, cars, and airplanes on 100 randomly sampled ShapeNet tables, each rendered in 200 different views sampled uniformly on a sphere. We then compute, for each of the 200 views, the median CD of the 100 reconstructions. Finally, in Figure 3-17, we visualize these
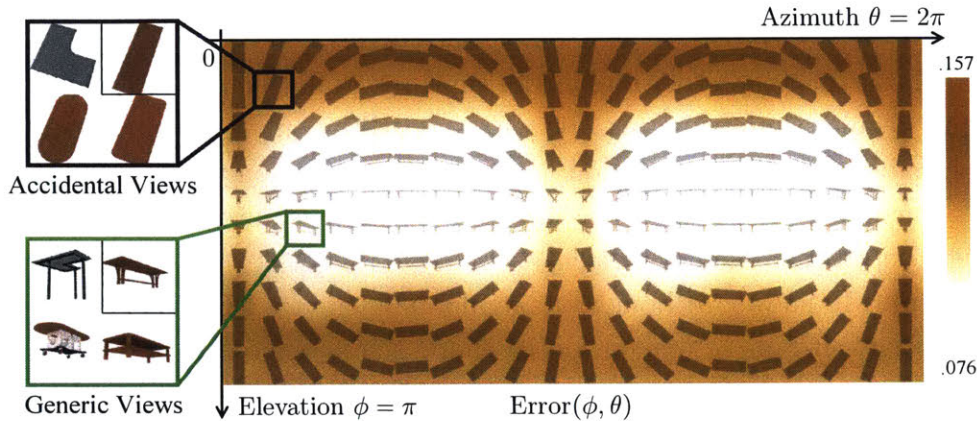
Figure 3-17: **Reconstruction errors (CD) for different input viewpoints.** The vertical (horizontal) axis represents elevation (azimuth). Accidental views (blue box) lead to large errors, while generic views (green box) result in smaller errors. Errors are computed for 100 tables; these particular tables are for visualization purposes only.
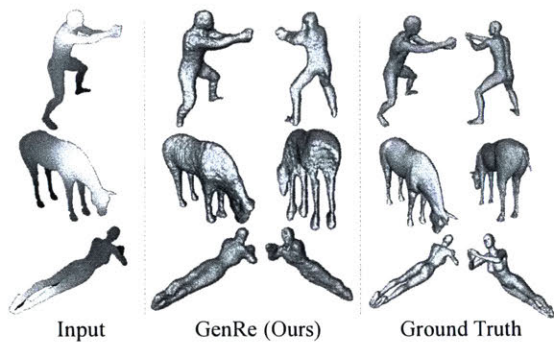
Figure 3-18: **Single-view completion of non-rigid shapes from depth maps** by our model trained on cars, chairs, and airplanes.
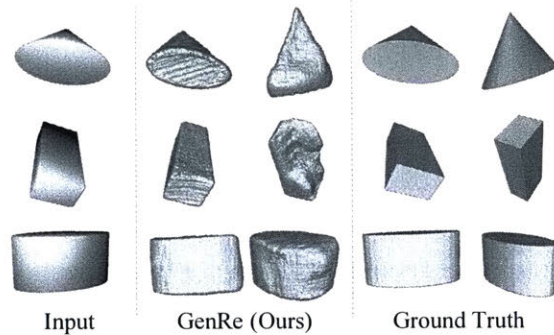
Figure 3-19: **Single-view completion of primitives from depth maps** by our model trained on cars, chairs, and airplanes.

median CDs as a heatmap over an elevation-azimuth view grid. As the heatmap shows, our model makes better predictions when the input view is generic than when it is accidental, consistent with our intuition.

3.8.2 Reconstructing Non-Rigid Shapes

We probe the generalization limit of our model by testing it with unseen non-rigid shapes, such as horses and humans. As the focus is mainly on the spherical map inpainting network (Figure 3-12b) and the voxel refinement network (Figure 3-12c), we assume our model has access to the ground-truth single-view depth (i.e., GenRe-Oracle) in this experiment. As demonstrated in Figure 3-18, our model not only retains the visible details in the original view, but also completes the unseen surfaces using the generic shape priors learned from rigid objects (cars, chairs, and airplanes).

3.8.3 Reconstructing Highly Regular Shapes

We further explore whether our model captures global shape attributes by testing it on highly regular shapes that can be parametrized by only a few attributes (such as cones and cubes). Similar to Section 3.8.2, the model has only seen cars, chairs, and airplanes during training, and we assume our model has access to the ground-truth single-view depth (i.e., GenRe-Oracle).

As Figure 3-19 shows, although our model hallucinates the unseen parts of these shape primitives, it fails to exploit global shape symmetry to produce correct predictions. This is not surprising given that our network design does not explicitly model such regularity. A possible future direction is to incorporate priors that facilitate learning high-level concepts such as symmetry.

3.9 Discussion

In this chapter, we have studied the problem of generalizable single-image 3D reconstruction. We exploit various image and shape representations, including 2.5D sketches, spherical maps, and voxels. We have proposed GenRe, a novel viewer-centered model that integrates these representations for generalizable, high-quality 3D shape reconstruction, as well as its precursors such as MarrNet, 3D-GAN, and ShapeHD. Experiments demonstrate that GenRe achieves state-of-the-art performance on shape reconstruction for both seen and unseen classes. We hope our system will inspire future research along this challenging but rewarding research direction.

Chapter 4

Learning with a Graphics Engine
for Multi-Object Scenes

In Chapter 2 and Chapter 3, we have explored how learning can be integrated with graphics engines for reconstructing sparse and dense 3D object representations. Those methods, however, still focus on images of a single object and work with simplified graphics engines. In this chapter, we study the problem of holistic scene understanding. We would like to obtain a compact, expressive, and interpretable representation of scenes that encodes information such as the number of objects and their categories, poses, positions, etc. Such a representation would allow us to reason about and even reconstruct or manipulate elements of the scene. While, prior work has used encoder-decoder based neural architectures to learn image representations, representations obtained in this way are typically uninterpretable, or only explain a single object in the scene.

Our main contribution in this chapter is a new approach to learn an interpretable, distributed representation of scenes. Our approach combines deep learning with a general, deterministic rendering function as the decoder, mapping a naturally structured and disentangled scene description, which we named scene XML, to an image. By doing so, the learned encoder is forced to perform the inverse of the rendering operation (a.k.a. de-rendering) to transform an input image to the structured scene XML that the decoder used to produce the image. We use a object proposal–based encoder that is trained by minimizing both the supervised prediction and the unsupervised reconstruction errors. We further develop an extension of the model to build 3D-aware representations for natural scenes. Experiments demonstrate that our approach works well on scene de-rendering, and our learned representation can be easily adapted for a wide range of applications like image editing, inpainting, visual analogy-making, and image captioning.

This chapter includes materials previously published as Wu et al. [2017b], Yao et al. [2018]. Shunyu Yao and Harry Hsu contributed significantly to the materials presented in this chapter.

4.1   Introduction

What properties are desirable in an image representation for visual understanding? We argue that the representation needs to be compact, expressive, and interpretable. Compactness makes it possible to store and exploit large amounts of data. Expressiveness allows it to capture the variations in the number, category, appearance, and pose of objects in an image. Lastly, an interpretable and disentangled representation enables us to reason about and even reconstruct or manipulate elements of an image.

Image representations learned by neural networks are often compact and expressive, but are hard to interpret. Recently, researchers studied how to obtain interpretable representations [Chen et al., 2016c, Kulkarni et al., 2015b, Yang et al., 2015]. They mostly employed an encoding-decoding framework, using neural nets for both inference and approximate rendering. However, these methods typically assume each input image contains only a single, centered object in front of a clean background. Consequently, they are not robust and powerful enough for practical applications, where we often see images with an indefinite number of objects, heavy occlusions, and a cluttered background.

In contrast to neural decoders like the ones used in Denton et al. [2015], Kulkarni et al. [2015b], the deterministic rendering functions used in graphics engines naturally take a structured and disentangled input to generate images. From this perspective, if we assume a given image is rendered by a generic graphics engine, we can aim to recover the structured representation required by renderer to reconstruct the exact image (a.k.a. de-rendering). By learning an image representation this way, we achieve interpretability for free, and we will also be able to use the representation in a range of applications like image editing.

This image de-rendering problem, as shown in Figure 4-1, is however very challenging for multiple reasons. First, as we are no longer assuming a localized object, and the number of objects in an image is unknown, our representation should be extensible to an arbitrary number of objects in different positions. This cannot be achieved in a straightforward way with traditional convolutional networks that learn image representations of a fixed dimension. Previous approaches explored the use of recurrent networks like LSTM [Hochreiter and Schmidhuber, 1997] in these cases. However, for a scene with many objects, it is counterintuitive and often ambiguous to manually define a sequential ordering over them. In this work, we instead draw inspiration from research in bottom-up visual recognition and propose a framework based on object proposals.

Second, we want the encoded representation to be generalizable to various graphics engines, though they may require very different input. We therefore design a unified structured language, named *scene XML*, which can be easily translated to inputs

64
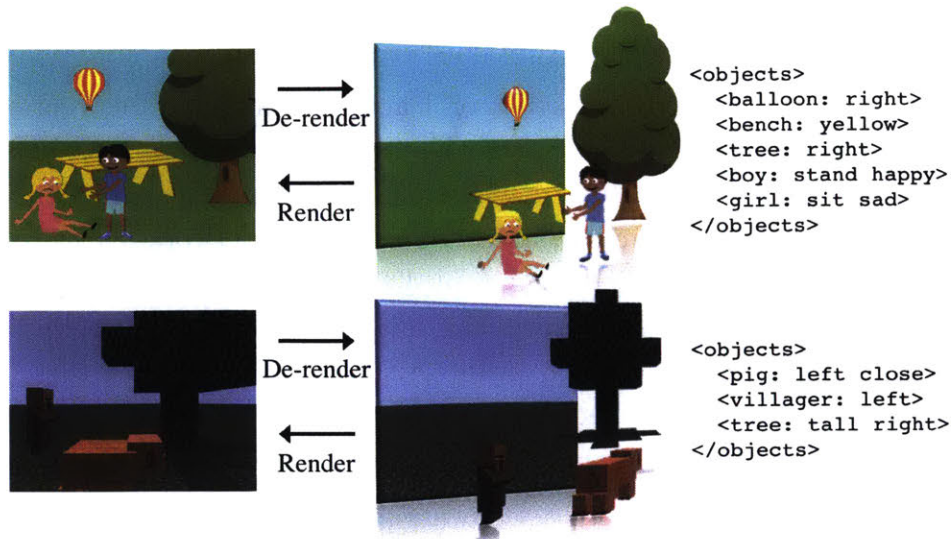
Figure 4-1: **Our goal is to interpret an image in a holistic way.** Assuming an image is rendered by a graphics engine on an indefinite length input, we aim to recover the input so that the the exact image can be reconstructed and manipulated. Here we show a simplified version of the XML we use.

that renderers can take. We evaluate our framework on two datasets with different rendering engines: the virtual Kitti dataset [Gaidon et al., 2016] and the Cityscapes dataset [Cordts et al., 2016].

Third, the space of encoded representations and the space of images do not share the same metric: a pair of close latent representations may correspond to images with significantly different visual appearance, and vice versa. Thus, learning a direct mapping from images to labeled representations does not guarantee good performance in reconstruction. In this chapter, we explore the possibility of having loss functions in both spaces within an end-to-end neural net framework. This is technically nontrivial because graphics engines are often not differentiable, with few exceptions [Loper and Black, 2014]. To overcome this problem, we use the multi-sample REINFORCE algorithm [Williams, 1992] for optimization.

Our contributions in this chapter are three-fold: first, we propose a new problem formulation, scene de-rendering, aiming to interpret a scene and the objects inside holistically by incorporating a graphics engine and a structured representation; second, we design a novel end-to-end framework for scene de-rendering, which involves optimization in both the latent representation space and the image space; third, we demonstrate the effectiveness of our framework by showing how it performs on synthetic and real scenes, and how it enables multiple applications.

4.2 Related Work

Our work is closely related to research on learning an interpretable representation with a neural network [Hinton et al., 1995, Kulkarni et al., 2015b, Yang et al., 2015, Chen et al., 2016c, Wu et al., 2016b]. Kulkarni et al. [2015b] proposed a convolutional inverse graphics network. Taking an image of a face, the network learns to infer its properties like pose and lighting. Yang et al. [2015] explored learning disentangled representations of pose and content from chair images. Chen et al. [2016c] proposed to learn disentangled representation without direct supervision. While all these methods dealt with images of a single object (chair, face, or digit), we study the problem of general scene de-rendering with an indefinite number of objects and possibly heavy occlusions.

Another line of related research is on sequential generative models for image recognition or synthesis [Huang and Murphy, 2015, Gregor et al., 2015, Eslami et al., 2016, Rezende et al., 2016b, Ba et al., 2015], which typically involve recurrent networks like LSTM [Hochreiter and Schmidhuber, 1997]. Many of them also trained a network as an approximate renderer simultaneously. In contrast, we explicitly model a graphics engine in the framework, and let neural nets focus on inverse graphics. The use of a real renderer provides us with an interpretable representation for free, and also generates images of higher quality.

Our framework also relates to the field of generative models with data-driven proposals [Yuille and Kersten, 2006, Zhu and Mumford, 2007, Tu and Zhu, 2002, Kulkarni et al., 2015a, Wu et al., 2015a, Jampani et al., 2015], as we are incorporating a graphics engine as a black-box synthesizer. However, our focus is still on using a feedforward model for bottom-up recognition and inference. Please see Bever and Poeppel [2010] for a nice review of analysis-by-synthesis methods.

4.3 Neural Scene De-rendering

We now present our analysis and approach to the scene de-rendering problem. We begin with a high-level abstraction of our method as a generalized encoding-decoding structure; we then discuss optimization and implementation details.

4.3.1 Generalized Encoding-Decoding Structure

Autoencoder. Traditionally autoencoder have neural networks as both the encoder and the decoder, as shown in Figure 4-2a. The goal of the network is to encode input into a compact representation (the bottleneck layer) and then to reconstruct the input. The latent vector learned this way can be viewed as an informative representation of the input.

(a) A standard autoencoder   (b) A generalized autoencoder
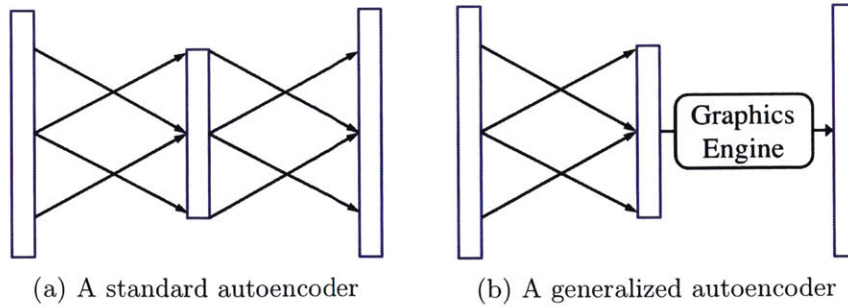
Figure 4-2: **Generalized encoding-decoding structure.** Different from a standard autoencoder (a), our generalized structure (b) uses a graphics engine as the decoder, which by nature takes an interpretable and disentangled representation as input, and renders a high quality image.


```
<object>
        <category>triangle</category>
        <size>1.5</size>
        <color>blue</color>
        <position>1.5,2,1</position>
        <yaw>0</yaw>
        ...
</object>
<object>
        ...
```

Figure 4-3: **An image and part of its scene XML,** encoding the background and the category, appearance, position, and pose of objects in the image.

Rendering engine as a generalized decoder. The latent representation of a standard autoencoder is neither disentangled nor interpretable, making it hard to generalize to other tasks. Here, we propose a generalized encoding-decoding structure, where we use a graphics engine as our decoder, as shown in Figure 4-2b. Unlike a neural decoder, a graphics engine in its nature requires a structured and interpretable image representation as input for rendering. In this way, the generalized autoencoder naturally learns to encode the image into an interpretable image representation.

The generalized structure needs to achieve two goals: first, minimizing the supervised prediction error on the inverted representations of input images; and second, minimizing the unsupervised reconstruction error on the rendered images. In Section 4.3.2, we explore how to integrate and balance both goals for better performance.

Scene XML. We want our framework to be independent of the graphics engine involved. To be specific, we hope to connect our encoder to a meta-renderer that translates learned representations to input that a specific graphics engine could take. To do this, we design a cross-platform structured image representation, named *Scene XML*, as the output of the encoder. Our goal is to design scene XML in a way that requires minimal effort to connect it to various graphics engines.

Our current design is in essence an object-centered representation. It starts with
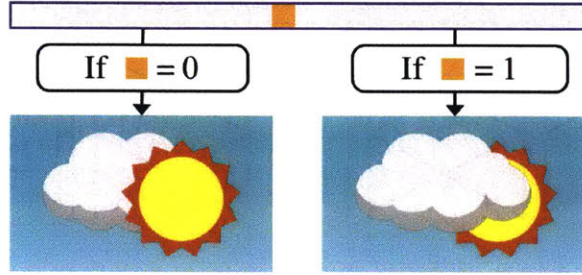
Figure 4-4: **The role of image-space loss:** a small change in the latent space (e.g., the depth of cloud) may lead to significant difference in rendered images. It is hence important to consider losses in both spaces.

some brief description of background, similar to the `<head>` tag in HTML. Then for each object, we track its category, appearance (size and color), position in 3D space ($\{x, y, z\}$), and pose (yaw, pitch, roll). In the future, we plan to also include its physical properties, and to model its actual 3D shape instead of using categories with fixed geometry as an abstraction. Figure 4-3 shows a sample image and part of its corresponding scene XML.

For each input image, our framework learns to interpret it in scene XML, and then translates the XML to the structured input that a graphics engine could take.

4.3.2 Black-Box Optimization via REINFORCE

As discussed in Section 4.1, visually similar images might have very different latent representations; also, two similar points in the representation space could lead to, after rendering, images with drastically different appearance. We show an example in Figure 4-4. With a small change in the value of a single dimension in the representation, here the depth of the cloud, the rendered images look totally different. Therefore, during training, we would like to minimize both the prediction error after the inference/encoding step, and the reconstruction error after the synthesis/rendering step.

This is, however, not practically straightforward as graphics engines are typically not differentiable, making it hard to back-propagate the gradients. Inspired by Rezende et al. [2016a], Ba et al. [2015], Jayaraman and Grauman [2016], we formulate this as a reinforcement learning problem, and adopt a multi-sample REINFORCE paradigm [Mnih and Rezende, 2016, Williams, 1992] to address this issue.

Specifically, instead of having a deterministic prediction, we have a stochastic layer at the end of our encoder, where our final prediction can be sampled from certain distributions (e.g., Gaussian for position and pose, multinomial for category). We obtain multiple samples from an input, and for each sample, we compute its reconstruction error after rendering. We use the negative log error as reward r of the sample, with its variance reduced by a baseline computed from the other samples.

The REINFORCE algorithm then allows us to calculate gradients on these stochastic layers and to back-propagate them to all layers before, via

$$\Delta w = \alpha(r - b)e, \qquad (4.1)$$

where w are the parameters of the distributions we are sampling from, α is the learning rate, b is the reinforcement baseline computed from other samples, and e is the distribution-dependent characteristic eligibility. Please refer to Mnih and Rezende [2016], Williams [1992] for more details.

REINFORCE as weight balancing. The mapping from latent representations to images is highly discontinuous. For each dimension in the latent representation, its impact on the rendered image changes as we move over the manifold. It is intractable to model the exact correlation; however, from a different perspective, the use of a graphics engine and the reinforcement learning (RL) framework implicitly guides the recognition network to balance the weights of each dimension under different circumstances.

Semi-supervised curriculum learning. The RL formulation also opens up the possibility for unsupervised learning: we can attempt to minimize the reconstruction error directly, and hopefully the network learns the disentangled representation required by the graphics engine automatically. We unfortunately observe that this is infeasible in practice. One reason for this failure is the large search space arising from the parametrization of the encoder. To address this, we employ a curriculum based approach where we initialize the training by using both reconstruction error and the label prediction loss on a small number of labeled images. Thereafter, we fine-tune the model with only unlabeled data, relying on the reconstruction error. We observe that the reinforcement learning framework can help to reduce the supervision required for training the encoder through curriculum learning [Bengio et al., 2009]. This semi-supervised learning setting could be useful in practice, where labeled data are often scarce.

4.3.3 Network Structure

Based on the generalized encoding-decoding structure, our framework has a neural encoder and a graphics engine as a generalized decoder. We show an overview of our model in Figure 4-5. We now describe our encoder in detail.

Our encoder has two components: a proposal generator for producing proposals that potentially contain objects, and an object interpreter for discriminating whether there is an object in each proposal, and if so, what its attributes are.

Our proposal generator (Figure 4-5-I) produces segment proposals instead of bounding boxes. This is because heavily occluded objects cannot be correctly interpreted
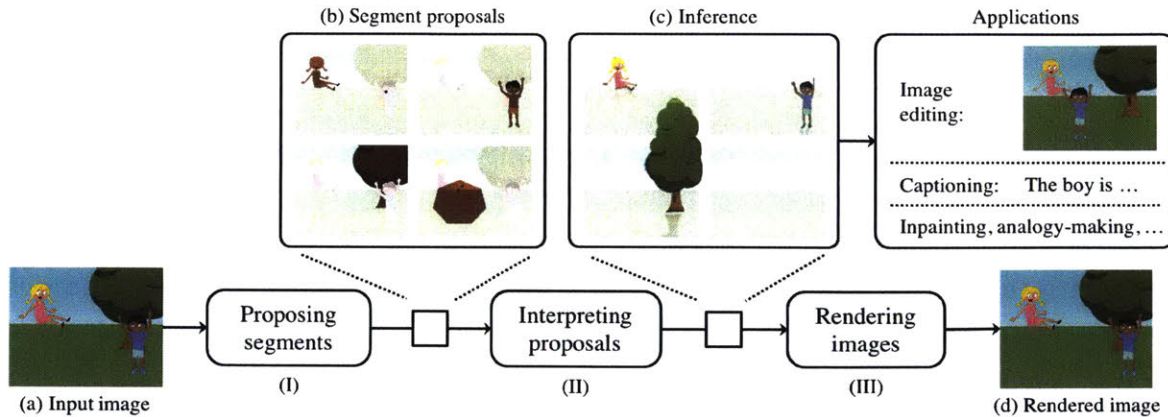
Figure 4-5: **Our neural scene de-rendering framework** consists of three component. Given an input image, it first generates a number of segment proposals (Stage I). It then tries to interpret if there is an object in the each proposal, and if so what its properties are (Stage II). Eventually, these inference results are integrated and sent to a graphics engine for rendering, so that the original image can be reconstructed (Stage III). We have supervision on both the latent representation space and the image space. Also note that the latent representations have wide applications including image editing, captioning, etc.

from box proposals. Also, during reconstruction, it would also be preferable for the model to incorrectly interpret the box proposal of the sun to be cloud, only because the cloud occupies a larger area in the box. In contrast, segment proposals do not suffer from this issue.

For the proposal generator, we use the network structure from an instance segmentation method, MNC [Dai et al., 2016]. It is a cascaded model where the network first learns both feature maps and coordinates of box instances (regions of interests, or RoI), and sends them through a RoI pooling layer to extract features of boxes. It then predicts masks of candidate objects within each box.

The object interpreter (Figure 4-5-II) takes a segment proposal (masked image) as input, and predicts whether there is an object in the segment. If the network believes an object exists, it also predicts its properties required by our scene XML. For each segment, we consider objects in the image that have an IoU over 0.3 with the segment, and select the one with the maximum IoU as ground truth for training the object interpreter. At the end, we apply non-maximal suppression (NMS) over the interpretations of all segments, and send it to the decoder (a graphics engine) for rendering (Figure 4-5-III).

4.4 Extension to Natural Scenes

The idea of integrating an off-the-shelf graphics engine with neural networks is powerful, but also limits its application to natural scenes, where the scene representation can be prohibitively high-dimensional. In this section, we present an extension

70
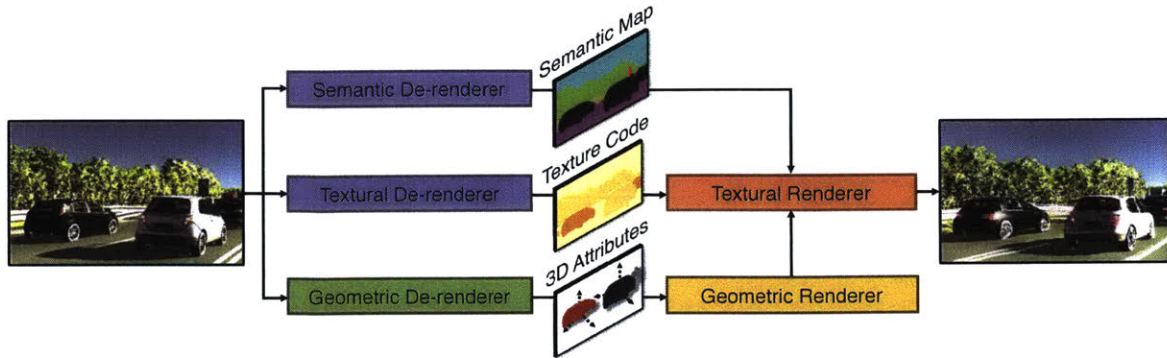
Figure 4-6: **Overview of 3D-SDN.** The de-renderer (encoder) consists of a semantic-, a textural- and a geometric branch. The textural renderer and geometric renderer then learn to reconstruct the original image from the representations obtained by the encoder modules.

to the original model that allows it to generalize to natural scenes via a combination of classic and neural graphics engines. We name our model 3D Scene De-rendering Networks (3D-SDN).

4.4.1 3D Scene De-rendering Networks

3D-SDN also has an encoder-decoder framework. As shown in Figure 4-6, it first de-renders (encodes) an image into disentangled representations for semantic, textural, and geometric information. Then, a renderer (decoder) reconstructs the image from the representation.

The semantic de-renderer learns to produce the semantic segmentation (e.g. trees, sky, road) of the input image. The 3D geometric de-renderer detects and segments objects (cars and vans) from image, and infers the geometry and 3D pose for each object with a differentiable shape renderer. After inference, the geometric renderer computes an instance map, a pose map, and normal maps for objects in the scene for the textural branch. The textural de-renderer first fuses the semantic map generated by the semantic branch and the instance map generated by the geometric branch into an instance-level semantic label map, and learns to encode the color and texture of each instance (object or background semantic class) into a texture code. Finally, the textural renderer combines the instance-wise label map (from the textural de-renderer), textural codes (from the textural de-renderer), and 3D information (instance, normal, and pose maps from the geometric branch) to reconstruct the input image.

3D geometric inference. Figure 4-7 shows the 3D geometric inference module for the 3D-SDN. We first segment object instances with Mask-RCNN [He et al., 2017]. For each object, we infer its 3D mesh model and other attributes from its masked image patch and bounding box.

We describe a 3D object with a mesh M, its scale $s \in \mathbb{R}^3$, rotation $q \in \mathbb{R}^4$ as

71
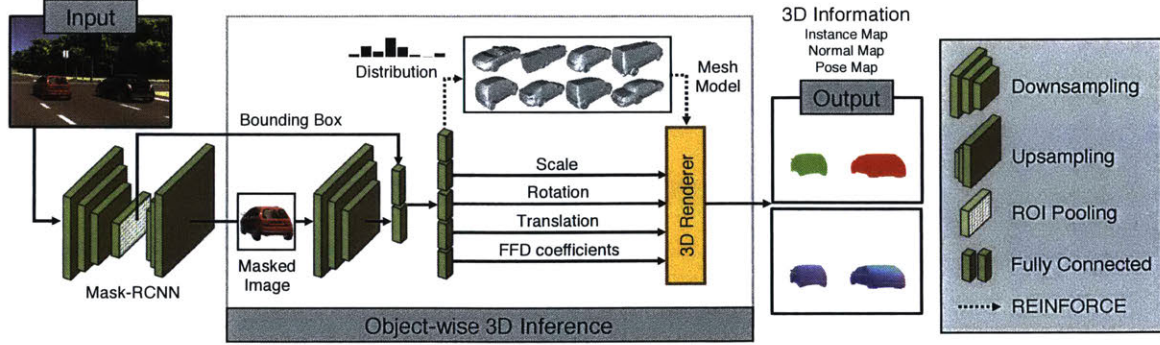
Figure 4-7: **3D geometric inference in 3D-SDN.** Given a masked object image and its bounding box, the geometric branch of the 3D-SDN predicts the object's mesh model, scale, rotation, translation, and the free-form deformation (FFD) coefficients. We then compute 3D information (instance map, normal maps, and pose map) using a differentiable renderer [Kato et al., 2018].

an unit quaternion, and translation $t \in \mathbb{R}^3$. For most real-world scenarios such as road scenes, objects often lie on the ground. Therefore, the quaternion has only one rotational degree of freedom: i.e., $q \in \mathbb{R}$.

As shown in Figure 4-7, given an object's masked image and estimated bounding box, the geometric de-renderer learns to predict the mesh M by first selecting a mesh from eight candidate shapes, and then applying a Free-Form Deformation (FFD) [Sederberg and Parry, 1986] with inferred grid point coordinates ϕ. It also predicts the scale, rotation, and translation of the 3D object. Below we describe the training objective for the network.

The geometric de-renderer directly predicts the values of scale s and rotation q. For translation t, it instead predicts the object's distance to the camera t and the image-plane 2D coordinates of the object's 3D center, denoted as $[x_{3D}, y_{3D}]$. Given the intrinsic camera matrix, we can calculate t from t and $[x_{3D}, y_{3D}]$. We parametrize t in the log-space [Eigen et al., 2014]. As determining t from the image patch of the object is under-constrained, our model predicts a normalized distance $\tau = t\sqrt{wh}$, where $[w, h]$ is the width and height of the bounding box. This reparameterization improves results as shown in later experiments. For $[x_{3D}, y_{3D}]$, we follow the prior work [Ren et al., 2015] and predict the offset $e = [(x_{3D} - x_{2D})/w, (y_{3D} - y_{2D})/h]$ relative to the estimated bounding box center $[x_{2D}, y_{2D}]$. The 3D attribute prediction loss for scale, rotation, and translation can be calculated as

$$\mathcal{L}_{\text{pred}} = \|\log \tilde{s} - \log s\|_2^2 + \left(1 - (\tilde{q} \cdot q)^2\right) + \|\tilde{e} - e\|_2^2 + (\log \tilde{\tau} - \log \tau)^2, \qquad (4.2)$$

where $\tilde{\cdot}$ denotes the predicted attributes.

We also use a reprojection loss to ensure the 2D rendering of the predicted shape fits its silhouette **S** [Yan et al., 2016b, Rezende et al., 2016a, Wu et al., 2016b, 2017c]. Figure 4-8a and Figure 4-8b show an example. Note that for mesh selection and

| (a) w/o re-projection | (b) w/ re-projection | (c) Single CAD w/o FFD | (d) Multiple CADs w/ FFD |
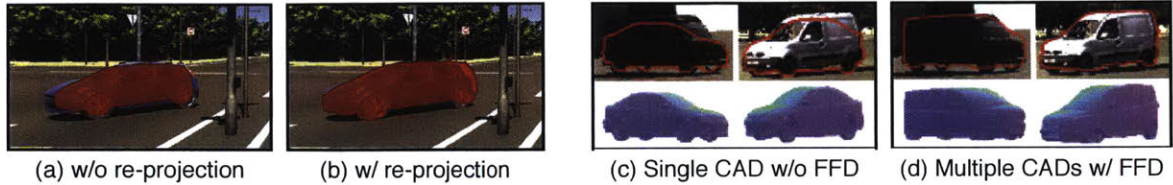
Figure 4-8: **(a)(b) Re-projection consistency loss**: Object silhouettes rendered without and with re-projection consistency loss. **(c)(d) Multiple CAD models and free form deformation (FFD)**: In (c), a generic car model without FFD fails to represent the input vans. In (d), our model learns to choose the best-fitting mesh from eight candidate meshes and allows FFD. As a result, we can reconstruct the silhouettes more precisely.

deformation, the reprojection loss is the only training signal, as we do not have a ground truth mesh model.

We use a differentiable renderer [Kato et al., 2018] to render the 2D silhouette of a 3D mesh M, according to the FFD coefficients ϕ and the object's scale, rotation and translation $\tilde{\pi} = \{\tilde{s}, \tilde{q}, \tilde{t}\}$: $\tilde{\mathsf{S}} = \mathsf{RenderSilhouette}(\mathsf{FFD}_\phi(M), \tilde{\pi})$. We then calculate the reprojection loss as $\mathcal{L}_{\mathrm{reproj}} = \left\| \tilde{\mathsf{S}} - \mathsf{S} \right\|$. We ignore the region occluded by other objects. The full loss function for the geometric branch is thus $\mathcal{L}_{\mathrm{pred}} + \lambda_{\mathrm{reproj}} \mathcal{L}_{\mathrm{reproj}}$, where λ controls the relative importance of two terms.

We choose the mesh M from a set of eight meshes to minimize the reprojection loss. As the model selection process is non-differentiable, we formulate the model selection as a reinforcement learning problem and adopt a multi-sample REINFORCE paradigm [Williams, 1992] to address the issue. The network predicts a multinomial distribution over the mesh models. We use the negative reprojection loss as the reward. We experimented with a single mesh without FFD in Figure 4-8c. Figure 4-8d shows a significant improvement when the geometric branch learns to select from multiple candidate meshes and allows flexible deformation.

Semantic and textural inference. The semantic branch of the 3D-SDN uses a semantic segmentation model DRN [Yu et al., 2017, Zhou et al., 2017b] to obtain an semantic map of the input image. The textural branch of the 3D-SDN first obtains an instance-wise semantic label map L by combining the semantic map generated by the semantic branch and the instance map generated by the geometric branch, resolving any conflict in favor of the instance map [Kirillov et al., 2019]. Built on recent work on multimodal image-to-image translation [Zhu et al., 2017a, Wang et al., 2018c], our textural branch encodes the texture of each instance into a low dimensional latent code, so that the textural renderer can later reconstruct the appearance of the original instance from the code. By 'instance' we mean a background semantic class (e.g., road, sky) or a foreground object (e.g., car, van). Later, we combine the object textural code with the estimated 3D information to better reconstruct objects.

Formally speaking, given an image I and its instance label map L, we want to

obtain a feature embedding $\mathbf{z}$ such that $(\mathbf{L}, \mathbf{z})$ can later reconstruct $\mathbf{I}$. We formulate the textural branch of the 3D-SDN under a conditional adversarial learning framework with three networks (G, D, E): a textural de-renderer $E : (\mathbf{L}, \mathbf{I}) \rightarrow \mathbf{z}$, a texture renderer $G : (\mathbf{L}, \mathbf{z}) \rightarrow \mathbf{I}$ and a discriminator $D : (\mathbf{L}, \mathbf{I}) \rightarrow [0, 1]$ are trained jointly with the following objectives.

To increase the photorealism of generated images, we use a standard conditional GAN loss [Goodfellow et al., 2014, Mirza and Osindero, 2014, Isola et al., 2017] as

$$\mathcal{L}_{\text{GAN}}(G, D, E) = \mathbb{E}_{\mathbf{L}, \mathbf{I}} \Big[\log \left(D(\mathbf{L}, \mathbf{I}) \right) + \log \left(1 - D\left(\mathbf{L}, \tilde{\mathbf{I}} \right) \right) \Big], \qquad (4.3)$$

where $\tilde{\mathbf{I}} = G(\mathbf{L}, E(\mathbf{L}, \mathbf{I}))$ is the reconstructed image, and we denote $\mathbb{E}_{\mathbf{L}, \mathbf{I}} \triangleq \mathbb{E}_{(\mathbf{L}, \mathbf{I}) \sim p_{\text{data}(\mathbf{L}, \mathbf{I})}}$ for simplicity. To stabilize the training, we follow the prior work [Wang et al., 2018c] and use both discriminator feature matching loss [Wang et al., 2018c, Larsen et al., 2016] and perceptual loss [Dosovitskiy and Brox, 2016, Johnson et al., 2016a], both of which aim to match the statistics of intermediate features between generated and real images:

$$\mathcal{L}_{\text{FM}}(G, D, E) = \mathbb{E}_{\mathbf{L}, \mathbf{I}} \left[\sum_{i=1}^{T_F} \frac{1}{N_i} \left\| F^{(i)}(\mathbf{I}) - F^{(i)}\left(\tilde{\mathbf{I}} \right) \right\|_1 + \sum_{i=1}^{T_D} \frac{1}{M_i} \left\| D^{(i)}(\mathbf{I}) - D^{(i)}\left(\tilde{\mathbf{I}} \right) \right\|_1 \right],$$
$$(4.4)$$

where $F^{(i)}$ denotes the i-th layer of a pre-trained VGG network [Simonyan and Zisserman, 2015] with N_i elements. Similarly, for our our discriminator D, $D^{(i)}$ denotes the i-th layer with M_i elements. T_F and T_D denote the number of layers in network F and D. We fix the network F during our training. Finally, we use a pixel-wise image reconstruction loss as:

$$\mathcal{L}_{\text{Recon}}(G, E) = \mathbb{E}_{\mathbf{L}, \mathbf{I}} \left[\left\| \mathbf{I} - \tilde{\mathbf{I}} \right\|_1 \right]. \qquad (4.5)$$

The final training objective is a minimax game between (G, E) and D:

$$\min_{(G, E)} \left(\max_D \left(\mathcal{L}_{\text{GAN}}(G, D, E) \right) + \lambda_{\text{FM}} \mathcal{L}_{\text{FM}}(G, D, E) + \lambda_{\text{Recon}} \mathcal{L}_{\text{Recon}}(G, E) \right), \qquad (4.6)$$

where λ_{FM} and λ_{Recon} control the relative importance of each term.

We observe that the textural de-renderer often learns not only texture but also object poses. To further decouple these two factors, we concatenate the inferred 3D information (i.e., pose map and normal map) from the geometric branch to the texture code map $\mathbf{z}$ and feed both of them to the textural renderer G. Also, we reduce the dimension of the texture code so that the code can focus on texture as the 3D geometry and pose are already provided. These two modifications help encode textural features

that are independent of the object geometry. It also resolves ambiguity in object poses: e.g., cars share similar silhouettes when facing forward or backward. Therefore, our renderer can synthesize an object under different 3D poses. (See Figure 4-9b and Figure 4-11b for example).

4.4.2 Implementation Details

Semantic branch. Our semantic branch adopts Dilated Residual Networks (DRN) for semantic segmentation [Yu et al., 2017, Zhou et al., 2017b]. We train the network for 25 epochs.

Geometric branch. We use Mask-RCNN for object proposal generation [He et al., 2017]. For object meshes, we choose eight CAD models from ShapeNet [Chang et al., 2015] including cars, vans, and buses. Given an object proposal, we predict its scale, rotation, translation, 4^3 FFD grid point coefficients, and an 8-dimensional distribution across candidate meshes with a ResNet-18 network [He et al., 2016]. The translation t can be recovered using the estimated offset e, the normalized distance $\log \tau$, and the ground truth focal length of the image. They are then fed to a differentiable renderer [Kato et al., 2018] to render the instance map and normal map.

We empirically set $\lambda_{\text{reproj}} = 0.1$. We first train the network with $\mathcal{L}_{\text{pred}}$ using Adam [Kingma and Ba, 2015] with a learning rate of 10^{-3} for 256 epochs and then fine-tune the model with $\mathcal{L}_{\text{pred}} + \lambda_{\text{reproj}}\mathcal{L}_{\text{reproj}}$ and REINFORCE with a learning rate of 10^{-4} for another 64 epochs.

Textural branch. We first train the semantic branch and the geometric branch separately and then train the textural branch using the input from the above two branches. We use the same architecture as in Wang et al. [2018c]. We use two discriminators of different scales and one generator. We use the VGG network [Simonyan and Zisserman, 2015] as the feature extractor F for loss λ_{FM} (Eqn. 4.4). We set the dimension of the texture code as 5. We quantize the object's rotation into 24 bins with one-hot encoding and fill each rendered silhouette of the object with its rotation encoding, yielding a pose map of the input image. Then we concatenate the pose map, the predicted object normal map, the texture code map z, the semantic label map, and the instance boundary map together, and feed them to the neural textural renderer to reconstruct the input image. We set $\lambda_{\text{FM}} = 5$ and $\lambda_{\text{Recon}} = 10$, and train the textural branch for 60 epochs on Virtual KITTI and 100 epochs on Cityscapes.

4.5 Experiments

We report our results in two parts. First, we present how the 3D-SDN enables 3D-aware image editing. For quantitative comparison, we compile a Virtual KITTI image editing benchmark to contrast 3D-SDNs and baselines without 3D knowledge.

Second, we analyze our design choices and evaluate the accuracy of representations obtained by different variants.

We conduct experiments on two street scene datasets: Virtual KITTI [Gaidon et al., 2016] and Cityscapes [Cordts et al., 2016]. Virtual KITTI serves as a proxy to the KITTI dataset [Geiger et al., 2012]. The dataset contains five virtual worlds, each rendered under ten different conditions, leading to a sum of 21,260 images. For each world, we use either the first or the last 80% consecutive frames for training and the rest for testing. For object-wise evaluations, we use objects with more than 256 visible pixels, a < 70% occlusion ratio, and a < 70% truncation ratio, following the ratios defined in Gaidon et al. [2016]. In our experiments, we downscale Virtual KITTI images to 624×192 and Cityscapes images to 512×256.

We have also built the *Virtual KITTI Image Editing Benchmark*, allowing us to evaluate image editing algorithms systematically. The benchmark contains 92 pairs of images in the test set with the camera either stationary or almost still. For each pair, we formulate the edit with object-wise operations. Each operation is parametrized by a starting position $(x_{3D}^{src}, y_{3D}^{src})$, an ending position $\left(x_{3D}^{tgt}, y_{3D}^{tgt}\right)$ (both are object's 3D center in image plane), a zoom-in factor ρ, and a rotation Δr_y with respect to the y-axis of the camera coordinate system.

The Cityscapes dataset contains 2,975 training images with pixel-level semantic segmentation and instance segmentation ground truth, but with no 3D annotations, making the geometric inference more challenging. Therefore, given each image, we first predict 3D attributes with our geometric branch pre-trained on Virtual KITTI dataset; we then optimize both attributes and mesh parameters π and ϕ by minimizing the reprojection loss $\mathcal{L}_{reproj}$. We use the Adam solver [Kingma and Ba, 2015] with a learning rate of 0.03 for 16 iterations.

4.5.1 3D-Aware Image Editing

The semantic, geometric, and textural disentanglement provides an expressive 3D image manipulation scheme. We can modify the 3D attributes of an object to translate, scale, or rotate it in the 3D world, while keeping the consistent visual appearance. We can also change the appearance of the object or the background by modifying the texture code alone.

Methods. We compare our 3D-SDNs with the following two baselines:

- 2D: Given the source and target positions, the naïve 2D baseline only applies the 2D translation and scaling, discarding the Δr_y rotation.

- 2D+: The 2D+ baseline includes the 2D operations above and rotates the 2D silhouette (instead of the 3D shape) along the y-axis according to the rotation Δr_y in the benchmark.
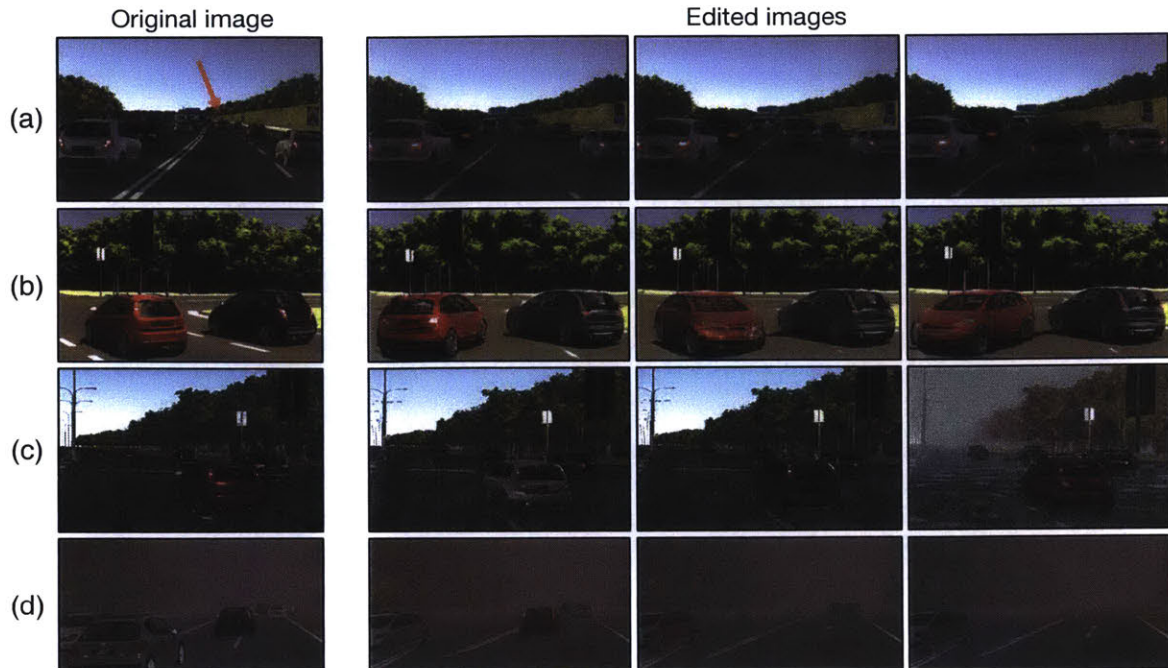
Figure 4-9: **Example user editing results on Virtual KITTI.** (a) We move a car closer to the camera, keeping the same texture. (b) We can synthesize the same car with different 3D poses. The same texture code is used for different poses. (c) We modify the appearance of the input red car using new texture codes. Note that its geometry and pose stay the same. We can also change the environment by editing the background texture codes. (d) We can inpaint occluded regions and remove objects.

Metrics. The pixel-level distance might not be a meaningful similarity metric, as two visually similar images may have a large L1/L2 distance [Isola et al., 2017]. Instead, we adopt the Learned Perceptual Image Patch Similarity (LPIPS) metric [Zhang et al., 2018a], which is designed to match the human perception. LPIPS ranges from 0 to 1, with 0 being the most similar. We apply LPIPS on (1) the full image, (2) all edited objects, and (3) the largest edited object.

Besides, we conduct a human study, where we show the target image as well as the edited results from two different methods: 3D-SDN vs. 2D and 3D-SDN vs. 2D+. We ask 120 human subjects on Amazon Mechanical Turk which edited result looks closer to the target. For better visualization, we highlight the largest edited object in red. We then compute, between a pair of methods, how often one method is preferred, across all test images.

Results. Figure 4-9 and Figure 4-10 show qualitative results on Virtual KITTI and Cityscapes, respectively. By modifying semantic, geometric, and texture codes, our editing interface enables a wide range of scene manipulation applications. Figure 4-11 shows a direct comparison to a state-of-the-art 2D manipulation method pix2pixHD [Wang et al., 2018c]. Quantitatively, Table 4.1a shows that our 3D-SDN

Figure 4-10: **Example user editing results on Cityscapes.** (a) We move two cars closer to the camera. (b) We rotate the car with different angles. (c) We recover a tiny and occluded car and move it closer. Our model can synthesize the occluded region as well as view the occluded car from the side. (d) We move a small car closer and then change its locations.

	3D-SDN (ours)	2D	2D+
LPIPS (whole)	**0.1280**	0.1316	0.1317
LPIPS (all)	**0.1444**	0.1782	0.1799
LPIPS (largest)	**0.1461**	0.1795	0.1813

(a) Perception similarity scores

	2D	2D+
3D-SDN (ours)	76.88%	74.28%

(b) Human study results

Table 4.1: **Results on Virtual KITTI editing benchmark.** (a) We evaluate the perceptual similarity [Zhang et al., 2018a] on the whole image (whole), all edited regions (all) of the image, and the largest edited region (largest) of the image, respectively. Lower scores are better. (b) Human subjects compare our method against two baselines. The percentage shows how often they prefer 3D-SDNs to the baselines. Our method outperforms previous 2D approaches consistently.

outperforms both baselines by a large margin regarding LPIPS. Table 4.1b shows that a majority of the human subjects perfer our results to 2D baselines.

4.5.2 Evaluation on the geometric representation.

Methods. We adopt multiple strategies to improve the estimation of 3D attributes. As an ablation study, we compare the full 3D-SDN, which is first trained using $\mathcal{L}_{\mathrm{pred}}$ then fine-tuned using $\mathcal{L}_{\mathrm{pred}} + \lambda_{\mathrm{reproj}}\mathcal{L}_{\mathrm{reproj}}$, with its four variants:

- w/o $\mathcal{L}_{\mathrm{reproj}}$: we only use the 3D attribute prediction loss $\mathcal{L}_{\mathrm{pred}}$.

- w/o quaternion constraint: we use the full rotation space characterized by a unit quaternion $\boldsymbol{q} \in \mathbb{R}^4$, instead of limiting to $\mathbb{R}$.
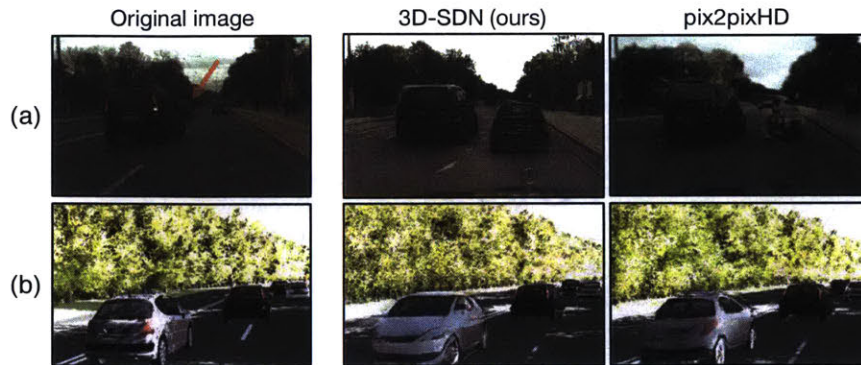
Figure 4-11: **Comparing 3D-SDN (ours) and pix2pixHD** [Wang et al., 2018c]. (a) We successfully recover the mask of an occluded car and move it closer to the camera while pix2pixHD fails. (b) We rotate the car from back to front. With the texture code encoded from the back view and a frontal pose, our model can remove the tail lights, while pix2pixHD cannot, given the same instance map.

	Orient. sim.	Distance ($\times 10^{-2}$)	Scale	Reproj. error ($\times 10^{-3}$)
Mousavian et al. [2017]	0.976	4.41	0.391	9.80
w/o $\mathcal{L}_{\text{reproj}}$	0.980	3.76	**0.372**	9.54
w/o quaternion constraint	0.970	4.59	0.403	7.58
w/o normalized distance τ	0.979	4.27	0.420	6.42
w/o MultiCAD and FFD	0.984	**3.37**	0.464	4.60
3D-SDN (ours)	**0.987**	3.87	0.382	**3.37**

Table 4.2: **Performance of 3D attributes prediction on Virtual KITTI**. We compare our full model with its four variants. Our full model performs the best regarding most metrics. Our model obtains much lower reprojection error. Refer to the text for details about our metrics.

- w/o normalized distance τ: we predict the original distance t in log space rather than the normalized distance τ.

- w/o MultiCAD and FFD: we use a single CAD model without free-form deformation (FFD).

We also compare with a 3D bounding box estimation method [Mousavian et al., 2017], which first infers the object's 2D bounding box and pose from input and then searches for its 3D bounding box.

Metrics. We use different metrics for different quantities. For rotation, we compute the orientation similarity $(1 + \cos \theta)/2$ [Geiger et al., 2012], where θ is the geodesic distance between the predicted and the ground truth rotations; for distance, we adopt an absolute logarithm error $|\log t - \log \tilde{t}|$; and for scale, we adopt the Euclidean distance $\|s - \tilde{s}\|_2$. In addition, we compute the per-pixel reprojection error between projected 2D silhouettes and ground truth segmentation masks.

Raw (LSTM)	jenny gets in the sandbox
NSD (LSTM)	jenny and mike both are both playing while the football sits in the sandbox
Raw (NN)	mike and jenny tired of playing frisbee decide to fly Jenny's new kite instead.
NSD (NN)	jenny and mike are having fun in the sandbox unaware of the storm that's coming their way
Raw (LSTM)	a picnic table while a snake and mike on it
NSD (LSTM)	jenny is angry by a snake while she and mike are running towards the park
Raw (NN)	a sad mike is hitting a baseball to an angry Jenny.
NSD (NN)	jenny is scared of a snake at their campsite but mike wants to go catch it.

Figure 4-12: **Results on image captioning.** Both LSTM and the nearest neighbor method work better using the de-rendered representations, compared to using raw pixels.

Results. Table 4.2 shows that our full model has significantly smaller 2D reprojection error than other variants. All of the proposed components contribute to the performance.

4.6   Applications

Our learned representation has extensive applications due to its expressiveness and interpretability. Here, we present qualitative results of image captioning and visual analogy-making on the Abstract Scene dataset [Zitnick and Parikh, 2013] and the Minecraft dataset [Wu et al., 2017b]. In Chapter 8 and Chapter 10, we will see more advanced development of our model for these tasks.

Image captioning. We explore to describe images from our inferred latent representation instead of end-to-end learning. First, as the representation carries full knowledge of the original image, we obtain some basic descriptions for free, e.g., there is *a happy boy* at *bottom-right*, facing to the *left*.

For captions involving high-level semantic understanding, we can build another model to map latent representations to captions. We consider two pilot studies. First, we train a seq2seq model [Sutskever et al., 2014] that reads an image representation, and directly generates a caption. Its core is a 256-dim LSTM. We compare with a CNN+LSTM model that reads a raw image and generates a caption. We train both models on the Abstract Scene dataset, sampling 90% captions and using the corresponding images for training, and the rest for testing.

Alternatively, for a test image, we may find the training image which has the

|Reference A|Reference A'|Query B|Prediction B'|
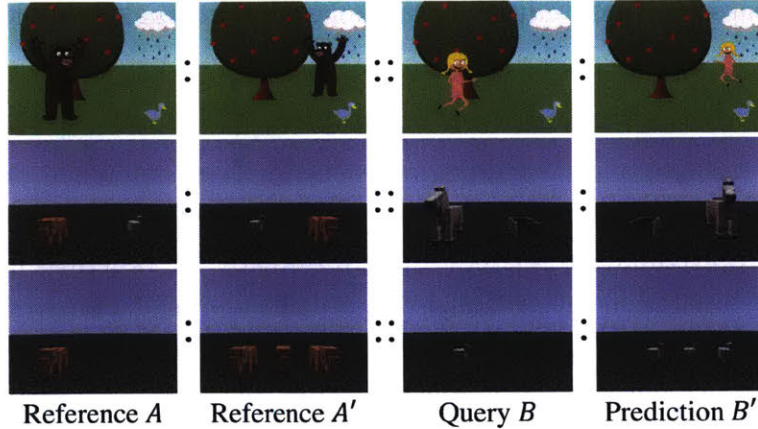
Figure 4-13: **Results on visual analogy-making.** Given a pair of reference images and a query, our framework can make analogies based on the position and pose of an object (top), and on the number of objects (bottom). See text for details.

minimum distance in the representation space, and transfer its caption. We compare with caption transfer from the nearest neighbor in pixel space.

Figure 4-12 shows qualitative results, where both LSTM and nearest neighbor perform better on our distributed representations, compared with on raw pixels. In Chapter 8, we will see more advanced development of our scene understanding model on visual question answering and concept learning.

Visual analogy-making. Visual analogy-making [Reed et al., 2015], or visalogy [Sadeghi et al., 2015], is an emerging research topic in AI and vision. A typical setting is to give a system a pair of images A and A' and an additional source image B, and ask for an analogy B'. While previous methods looked into learning analogies between objects, we study the problem of making scene analogies involving multiple objects.

We consider a principled formulation for this seemingly ambiguous problem. Given two image representations Z_A and $Z_{A'}$, we consider their *minimum edit distance* — the minimum number of operations required to derive $Z_{A'}$ from Z_A. We then apply these operations on Z_B to get an analogy $Z_{B'}$. The operations we consider are changing the pose, position, and category of an object, duplicating or removing an object, and swapping two objects.

Learning an expressive, interpretable, and disentangled representation could be a well-fit solution to this problem. We show results of depth first search (depth capped at 3) on top of the representations reconstructed by our scene de-rendering framework in Figure 4-13. It successfully makes analogies with respect not only to the position and pose of an object, but also to the number of objects in the image. In Chapter 10, we will see extensions of our model that captures the regularity and program-like

structure in scenes for image editing, extrapolation, and analogy-making.

4.7 Discussion

It has been popular to use neural networks for both inference and synthesis in image understanding. Research in this direction is fruitful and inspiring; however, current neural approximate renderers are still unready for practical use. In contrast, graphics engines have been rather mature, especially for virtual environments [Gaidon et al., 2016, Zhu et al., 2017b]. We feel it could be a promising direction to incorporate a black-box graphics engine into a generalized encoding-decoding structure. Based on this observation, in this chapter we have proposed a neural scene de-rendering framework for image representation learning and reconstruction, extending the shape reconstruction models introduced in the first two chapters. Results proved that our method performed well, and the learned representation has wide applications in a diverse set of vision tasks.

Part II

Dynamics: Learning with Physics Engines

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Learning with a Physics Engine

Humans demonstrate remarkable abilities to predict physical events in dynamic scenes, and to infer the physical properties of objects from static images. In Part II of this dissertation, we describe our efforts on building computational models that see 'physics' from visual input via a combination of deep learning with graphics and physics engines.

In this chapter, we propose a model for perceiving physical object properties from videos and images. At the core of our generative model is a 3D physics engine, operating on an object-based representation of physical properties, including mass, position, 3D shape, and friction. We can infer these latent properties using relatively brief runs of MCMC, which drive simulations in the physics engine to fit key features of visual observations. We further explore directly mapping visual inputs to physical properties, inverting a part of the generative process using deep learning.

We name our model Galileo, and evaluate it on a video dataset named Physics 101 of simple yet physically rich scenarios. Results show that Galileo is able to infer the physical properties of objects and predict the outcome of a variety of physical events, with an accuracy comparable to human subjects.

This chapter includes materials previously published as Wu et al. [2015a, 2016a]. Ilker Yildirim and Joseph Lim contributed significantly to the materials presented in this chapter.

5.1   Introduction

Our visual system is designed to perceive a physical world that is full of dynamic content. Consider yourself watching a Rube Goldberg machine unfold: as the kinetic energy moves through the machine, you may see objects sliding down ramps, colliding with each other, rolling, entering other objects, falling—many kinds of physical interactions between objects of different masses, materials and other physical properties. How does our visual system recover so much content from the dynamic physical world? What is the role of experience in interpreting a novel dynamical scene?

Recent behavioral and computational studies of human physical scene understanding push forward an account that people's judgments are best explained as probabilistic simulations of a realistic, but mental, physics engine [Battaglia et al., 2013, Sanborn et al., 2013]. Specifically, these studies suggest that the brain carries detailed but noisy knowledge of the physical attributes of objects and the laws of physical interactions between objects (i.e., Newtonian mechanics). To understand a physical scene, and more crucially, to predict the future dynamical evolution of a scene, the brain relies on simulations from this mental physics engine.

Even though the probabilistic simulation account is very appealing, there are missing practical and conceptual leaps. First, as a practical matter, the probabilistic simulation approach is shown to work only with synthetically generated stimuli: either in 2D worlds, or in 3D worlds but each object is constrained to be a block and the joint inference of the mass and friction coefficient is not handled [Battaglia et al., 2013]. Second, as a conceptual matter, previous research rarely clarifies how a mental physics engine could take advantage of previous experience of the agent [Ullman et al., 2017]. It is the case that humans have a life long experience with dynamical scenes, and a fuller account of human physical scene understanding should address it.

Here, we build on the idea that humans utilize a realistic physics engine as part of a generative model to interpret real-world physical scenes. The first component of our generative model is the physical object representations, where each object is a rigid body and represented not only by its 3D geometric shape (or volume) and its position in space, but also by its mass and its friction. All of these object attributes are treated as latent variables in the model, and are approximated or estimated on the basis of the visual input.

The second part is a fully-fledged realistic physics engine—in this chapter, specifically the Bullet physics engine [Coumans, 2010]. The physics engine takes a scene setup as input (e.g., specification of each of the physical objects in the scene, which constitutes a hypothesis in our generative model), and physically simulates it forward in time, generating simulated velocity profiles and positions for each object.

The third part is the likelihood function. We evaluate the observed real-world videos with respect to the model's hypotheses using the velocity vectors of objects in the scene. We use a standard tracking algorithm to map the videos to the velocity space.

Now, given a video as observation to the model, physical scene understanding in the model corresponds to inverting the generative model by probabilistic inference to recover the underlying physical object properties in the scene. We name our model Galileo in honor of the scientist as well as his well-known ramp experiment, which we also explore in this chapter.

For evaluation, we build a video dataset named Physics 101 to evaluate our model

and humans on real-world data. From Physics 101, we sample 150 videos of different objects with a range of materials and masses over a simple yet physically rich scenario: an object sliding down an inclined surface, and potentially collide with another object on the ground. Note that in the fields of computer vision and robotics, there have been studies on predicting physical interactions or inferring 3D properties of objects for various purposes including 3D reasoning [Jia et al., 2015, Zheng et al., 2015] and tracking [Schulman et al., 2013]. However, none of them focused on learning physical properties directly, and nor they have incorporated a physics engine with representation learning.

Based on the estimates we derived from visual input with a physics engine, a natural extension is to generate or synthesize training data for any automatic learning systems by bootstrapping from the videos already collected, and labeling them with estimates of Galileo. This is a self-supervised learning algorithm for inferring generic physical properties, and relates to the wake/sleep phases in Helmholtz machines [Dayan et al., 1995], and to the cognitive development of infants. Extensive studies suggest that infants either are born with or can learn quickly physical knowledge about objects when they are very young, even before they acquire more advanced high-level knowledge like semantic categories of objects [Carey, 2009, Baillargeon, 2004]. Young babies are sensitive to physics of objects mainly from the motion of foreground objects from background [Baillargeon, 2004]; in other words, they learn by watching *videos* of moving objects. But later in life, and clearly in adulthood, we can perceive physical attributes in just static scenes without any motion.

Here, building upon the idea of Helmholtz machines [Dayan et al., 1995], our approach suggests one potential computational path to the development of the ability to perceive physical content in static scenes. Following recent work [Yildirim et al., 2018a], we train a recognition model (i.e., sleep cycle) that is in the form of a deep convolutional network, where the training data is generated in a self-supervised manner by the generative model itself (i.e., wake cycle: real-world videos observed by our model and the resulting physical inferences).

We make four contributions in this chapter. First, we build Physics 101, a dataset of diverse physical events for evaluating physical scene understanding models. Second, we propose Galileo, a novel model for estimating physical properties of objects from visual inputs by incorporating the feedback of a physics engine in the loop. We demonstrate that it achieves encouraging performance on a real-world video dataset. Third, we train a deep learning based recognition model that leads to efficient inference in the generative model, and enables the generative model to predict future dynamical evolution of static scenes (e.g., how would that scene unfold in time). Last, we test our model and compare it to humans on a variety of physical judgment tasks. Our results indicate that humans are quite successful in these tasks, and our model closely matches

Figure 5-1: **Sample videos from the Physics 101 dataset.** Our data are taken by four sensors (3 RGB and 1 depth camera).

humans in performance, but also consistently makes similar errors as humans do, providing further evidence in favor of the probabilistic simulation account of human physical scene understanding.

5.2   The Physics 101 Dataset

The AI and vision community has made much progress through its datasets, and there are datasets of objects, attributes, materials, and scene categories. Here, we introduce a new type of dataset—one that captures physical interactions of objects. The dataset consists of four different scenarios, for each of which plenty of intriguing questions may be asked. For example, in the ramp scenario, will the object on the ramp move, and if so and two objects collide, which of them will move next and how far?

5.2.1   Scenarios

We seek to learn physical properties of objects by observing videos. To this end, we build a dataset by recording videos of moving objects. We pick an introductory setup with four different scenarios, which are illustrated in Figures 5-1 and 5-2. We then introduce each scenario in detail.

Ramp.   We put an object on an inclined surface, and the object may either slide down or keep static, due to gravity and friction. This seemingly straightforward scenario already involves understanding many physical object properties including material, coefficient of friction, mass, and velocity. Figure 5-2a analyzes the physics

88

I. Initial setup   II. Sliding   III. Colliding   IV. Final outcome

(a) The ramp scenario. Several physical properties will determine if object A will move, if it will reach to object B, and how far each object will move. Here, N, R, and G indicate a normal force, a friction force, and a gravity force, respectively.

I. Initial   II. Final   I. Initial   II. Colliding   III. Bouncing   I. Floating   II. Sunk

(b) The spring scenario   (c) The fall scenario   (d) The liquid scenario

Figure 5-2: **Illustrations of the scenarios in the Physics 101 dataset**

behind our setup.

In this scenario, the observable descriptive physical properties are the velocities of the objects, and the distances both objects traveled. The latent properties directly involved are coefficient of friction and mass.

Spring. We hang objects on a spring, and gravity on the object will stretch the spring. Here the observable descriptive physical property is length that the spring gets stretched, and the latent properties are the mass of the object and the elasticity of the spring.

Fall. We drop objects in the air, and they freely fall onto various surfaces. Here the observable descriptive physical properties are the the bounce heights of the object, and the latent properties are the coefficient of restitution of the object and the surface.

Liquid. We drop objects into some liquid, and they may float or sink at various speeds. In this scenario, the observable descriptive physical property is the velocity of the sinking object (0 if it floats), and the latent properties are the densities of the object and the liquid.

5.2.2 Building Physics 101

The outcomes of various physical events depend on multiple factors of objects, such as materials (density and friction coefficient), sizes and shapes (volume), and slopes of ramps (gravity), elasticities of springs, etc. We collect our dataset while varying all these conditions. Figure 5-3 shows the entire collection of our 101 objects, and the following are more details about our variations:

89

Figure 5-3: **The set of objects used in the Physics 101 dataset.** We vary object material, color, shape, and size, together with external conditions such as the slope of a surface or the stiffness of a string. Videos recording the motions of these objects interacting with target objects will be used to train our algorithm.

Material. Our 101 objects are made of 15 different materials: cardboard, dough, foam, hollow rubber, hollow wood, metal coin, pole, and block, plastic doll, ring, and toy, porcelain, rubber, wooden block, and wooden pole.

Appearance. For each material, we have 4 to 12 objects of different sizes and colors.

Slope (ramp). We also vary the angle α between the inclined surface and the ground (to vary the gravity force). We set $\alpha = 10^o$ and 20° for each object.

Target (ramp). We have two different target objects—a cardboard and a foam box. They are made of different materials, thus having different friction coefficients and densities.

Spring. We use two springs with different stiffness.

Surface (fall). We drop objects onto five different surfaces: foam, glass, metal, wooden table, and woolen rug. These materials have different coefficients of restitution.

We also measure the physical properties of these objects. We record the mass and volume of each object. For each setup, we record their actions for 3–10 trials. We measure multiple times because some external factors, e.g., orientations of objects and rough planes, may lead to different outcomes. Having more than one trial per condition increases the diversity of our dataset by making it cover more possible outcomes.

Finally, we record each trial from three different viewpoints: side, top-down, and upper-top. For the first two, we take data with DSLR cameras, and for the upper-top view, we use a Kinect V2 to record both RGB and depth maps. We have 4,352 trials in total. Given we captured videos in three RGB maps and one depth map, there are 17,408 video clips altogether. These video clips constitute the Physics 101 dataset.

In this chapter, we focus on the ramp scenario. The observed outcome of these scenarios are physical values which help to describe the scenario, such as the velocity

(a) Snapshots of the Physics 101 dataset   (b) Overview of the Galileo model
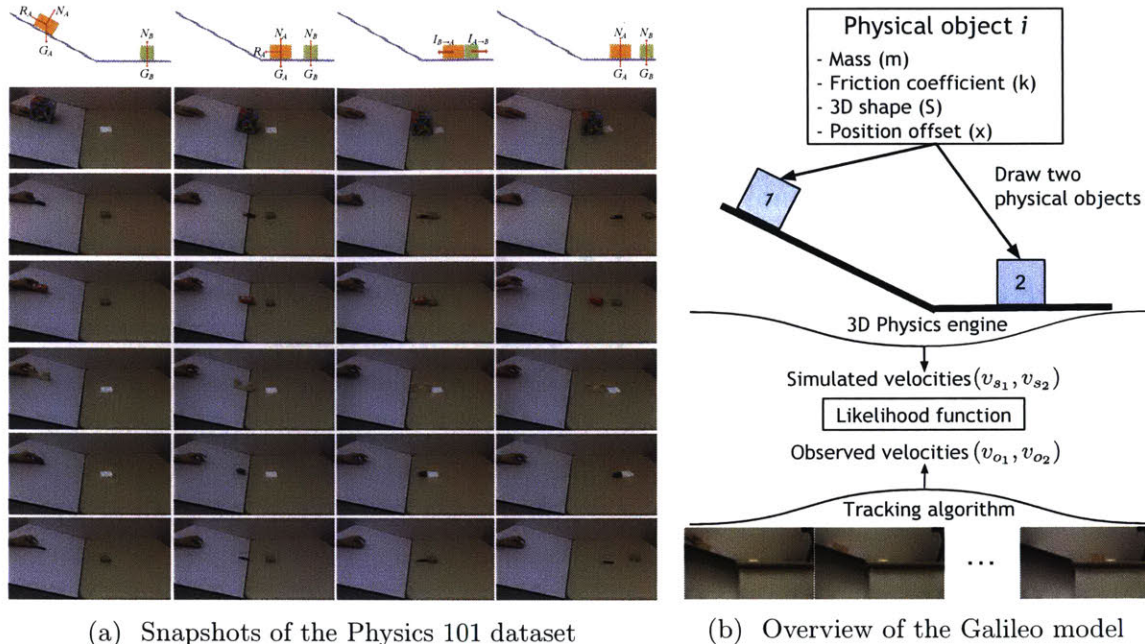
Figure 5-4: **The Galileo model and the Physics 101 dataset.** Our model formalizes a hypothesis space of physical object representations, where each object is defined by its mass, friction coefficient, 3D shape, and a positional offset w.r.t. an origin. To model videos, we draw exactly two objects from that hypothesis space into the physics engine. The simulations from the physics engine are compared to observations in the velocity space, a much "nicer" space than pixels.

and moving distance of objects. Causally underlying these observations are the latent physical properties of objects such as the material, density, mass and friction coefficient. As shown in Section 5.3, our Galileo model intends to model the causal generative relationship between these observed and unobserved variables.

5.3 Galileo: A Physical Object Model

The gist of our model (Figure 5-4b) can be summarized as probabilistically inverting a physics engine in order to recover unobserved physical properties of objects. We collectively refer to the unobserved latent variables of an object as its *physical representation T*. For each object i, T_i consists of its mass m_i, friction coefficient k_i, 3D shape V_i, and position offset p_i w.r.t. an origin in 3D space.

We place uniform priors over the mass and the friction coefficient for each object: $m_i \sim \text{Uniform}(0.001, 1)$ and $k_i \sim \text{Uniform}(0, 1)$, respectively. For 3D shape V_i, we have four variables: a shape type t_i, and the scaling factors for three dimensions x_i, y_i, z_i. We simplify the possible shape space in our model by constraining each shape type t_i to be one of the three with equal probability: a box, a cylinder, and a torus. Note that applying scaling differently on each dimension to these three basic

shapes results in a large space of shapes*. The scaling factors are chosen to be uniform over the range of values to capture the extent of different shapes in the dataset.

Remember that our scenario consists of an object on the ramp and another on the ground (Figure 5-4a). The position offset, p_i, for each object is uniform over the set $\{0, \pm1, \pm2, \cdots, \pm5\}$. This indicates that for the object on the ramp, its position can be perturbed along the ramp (i.e., in 2D) at most 5 units upwards or downwards from its starting position, which is 30 units upwards on the ramp from the ground.

The next component of our generative model is a fully-fledged realistic physics engine that we denote as ρ. Specifically we use the Bullet physics engine [Coumans, 2010] following the earlier related work. The physics engine takes a specification of each of the physical objects in the scene within the basic ramp setting as input, and simulates it forward in time, generating simulated velocity vectors for each object in the scene, v_{s_1} and v_{s_2} respectively—among other physical properties such as position, rendered image of each simulation step, etc.

In light of initial qualitative analysis, we use velocity vectors as our feature representation in evaluating the hypothesis generated by the model against data. We employ a standard tracking algorithm (the KLT point tracker [Tomasi and Kanade, 1991]) to "lift" the visual observations to the velocity space. That is, for each video, we first run the tracking algorithm, and we obtain velocities by simply using the center locations of each of the tracked moving objects between frames. This gives us the velocity vectors for the object on the ramp and the object on the ground, v_{o_1} and v_{o_2}, respectively. Note that we could replace the KLT tracker with state-of-the-art tracking algorithms for more complicated scenarios.

Given a pair of observed velocity vectors, v_{o_1} and v_{o_2}, the recovery of the physical object representations T_1 and T_2 for the two objects via physics-based simulation can be formalized as:

$$P(T_1, T_2 | v_{o_1}, v_{o_2}, \rho(\cdot)) \propto P(v_{o_1}, v_{o_2} | v_{s_1}, v_{s_2}) \cdot P(v_{s_1}, v_{s_2} | T_1, T_2, \rho(\cdot)) \cdot P(T_1, T_2). \quad (5.1)$$

where we define the likelihood function as $P(v_{o_1}, v_{o_2} | v_{s_1}, v_{s_2}) = N(v_o | v_s, \Sigma)$, where v_o is the concatenated vector of v_{o_1}, v_{o_2}, and v_s is the concatenated vector of v_{s_1}, v_{s_2}. The dimensionality of v_o and v_s are kept the same for a video by adjusting the number of simulation steps we use to obtain v_o according to the length of the video. But from video to video, the length of these vectors may vary. In all of our simulations, we fix Σ to 0.05, which is the only free parameter in our model. Experiments show that the value of Σ does not change our results significantly.

---

*For shape type box, x_i, y_i, and z_i could all be different values; for shape type torus, we constrained the scaling factors such that $x_i = z_i$; and for shape type cylinder, we constrained the scaling factors such that $y_i = z_i$.
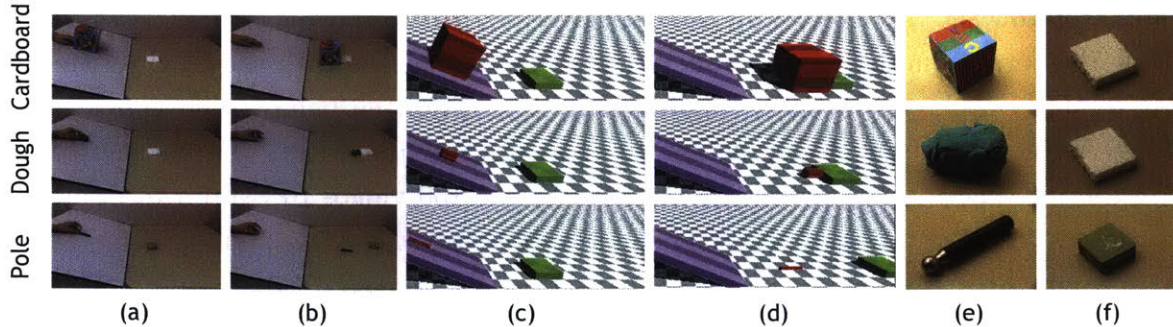
Figure 5-5: **Simulation results**. Each row represents one video in the data: (a) the first frame of the video, (b) the last frame of the video, (c) the first frame of the simulated scene generated by Bullet, (d) the last frame of the simulated scene, (e) the estimated object with larger mass, (f) the estimated object with larger friction coefficient.

Tracking as recognition. The posterior distribution in Equation 5.1 is intractable. In order to alleviate the burden of posterior inference, we use the output of our recognition model to predict and fix some of the latent variables in the model.

Specifically, we determine the V_i, or $\{t_i, x_i, y_i, z_i\}$, using the output of the tracking algorithm, and fix these variables without further sampling them. Furthermore, we fix values of p_is also on the basis of the output of the tracking algorithm.

Inference. Once we initialize and fix the latent variables using the tracking algorithm as our recognition model, we then perform single-site Metropolis Hasting updates on the remaining four latent variables, m_1, m_2, k_1 and k_2. At each MCMC sweep, we propose a new value for one of these random variables, where the proposal distribution is Uniform$(-0.05, 0.05)$. In order to help with mixing, we also use a broader proposal distribution, Uniform$(-0.5, 0.5)$ at every 20 MCMC sweeps.

5.4   Simulations

For each video, as mentioned earlier, we use the tracking algorithm to initialize and fix the shapes of the objects, S_1 and S_2, and the position offsets, p_1 and p_2. We also obtain the velocity vector for each object using the tracking algorithm. We determine the length of the physics engine simulation by the length of the observed video—that is, the simulation runs until it outputs a velocity vector for each object that is as long as the input velocity vector from the tracking algorithm.

We sample 150 videos from the Physics 101 dataset, uniformly distributed across different object categories. We perform 16 MCMC simulations for a single video, each of which was 75 MCMC sweeps long. We report the results with the highest log-likelihood score across the 16 chains (i.e., the MAP estimate).

In Figure 5-5, we illustrate the results for three individual videos. Every two frame

of the top row shows the first and the last frame of a video, and the bottom row images show the corresponding frames from our model's simulations with the MAP estimate. We quantify different aspects of our model in the following behavioral experiments, where we compare our model against human subjects' judgments. Furthermore, we use the inferences made by our model here on the 150 videos to train a recognition model to arrive at physical object perception in static scenes with the model.

Importantly, note that our model can generalize across a broad range of tasks beyond the ramp scenario. For example, once we infer the coefficient friction of an object, we can make a prediction on whether it will slide down a ramp with a different slope by doing simulation. We test some of the generalizations in Section 5.6.

5.5   Bootstrapping as Efficient Perception in Static Scenes

Based on the estimates we derived from the visual input with a physics engine, we bootstrap from the videos already collected, by labeling them with estimates of Galileo. This is a self-supervised learning algorithm for inferring generic physical properties. As discussed in Section 5.1, this formulation is also related to the wake/sleep phases in Helmholtz machines, and to the cognitive development of infants.

Here we focus on two physical properties: mass and friction coefficient. To do this, we first estimate these physical properties using the method described in earlier sections. Then, we train LeNet [LeCun et al., 1998], a widely used deep neural network for small-scale datasets, using image patches cropped from videos based on the output of the tracker as data, and estimated physical properties as labels. The trained model can then be used to predict these physical properties of objects based on purely visual cues, even though they might have never appeared in the training set.

We also measure masses of all objects in the dataset, which makes it possible for us to quantitatively evaluate the predictions of the deep network. We choose one object per material as our test cases, use all data of those objects as test data, and the others as training data. We compare our model with a baseline, which always outputs a uniform estimate calculated by averaging the masses of all objects in the test data, and with an oracle algorithm, which is a LeNet trained using the same training data, but has access to the ground truth masses of training objects as labels. Apparently, the performance of the oracle model can be viewed as an upper bound of our Galileo system.

Table 5.1 compares the performance of Galileo, the oracle algorithm, and the baseline. We can observe that Galileo is much better than baseline, although there is still some space for improvement.

Because we trained LeNet using static images to predict physical object properties such as friction and mass ratios, we can use it to *recognize* those attributes in a quick

94

Methods	Mass	
	MSE	Correlation
Oracle	0.042	0.71
Galileo	0.052	0.44
Uniform	0.081	0

Table 5.1: **Mean squared errors of oracle estimation, our estimation, and uniform estimations of mass** on a log-normalized scale, and the correlations between estimations and ground truths
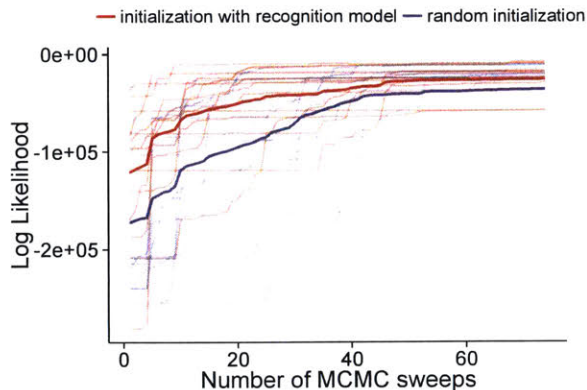
Figure 5-6: **The log-likelihood traces** of several chains with and without recognition-model (LeNet) based initializations

bottom-up pass at the very first frame of the video. To the extent that the trained LeNet is accurate, if we initialize the MCMC chains with these bottom-up predictions, we expect to see an overall boost in our log-likelihood traces. We test by running several chains with and without LeNet-based initializations. Results can be seen in Figure 5-6. Despite the fact that LeNet is not achieving perfect performance by itself, we indeed get a boost in speed and quality in the inference.

5.6 Experiments

In this section, we conduct experiments from multiple perspectives to evaluate our model. Specifically, we use the model to predict how far objects will move after the collision; whether the object will remain stable in a different scene; and which of the two objects is heavier based on observations of collisions. For every experiment, we also conduct behavioral experiments on Amazon Mechanical Turk so that we may compare the performance of human and machine on these tasks.

5.6.1 Outcome Prediction

In the outcome prediction experiment, our goal is to measure and compare how well human and machines can predict the moving distance of an object if only part of the video can be observed. Specifically, for behavioral experiments on Amazon Mechanical Turk, we first provide users four full videos of objects made of a certain material, which contain complete collisions. In this way, users may infer the physical properties associated with that material in their mind. We select a different object, but made of the same material, show users a video of the object, but only to the moment of collision. We finally ask users to label where they believe the target object
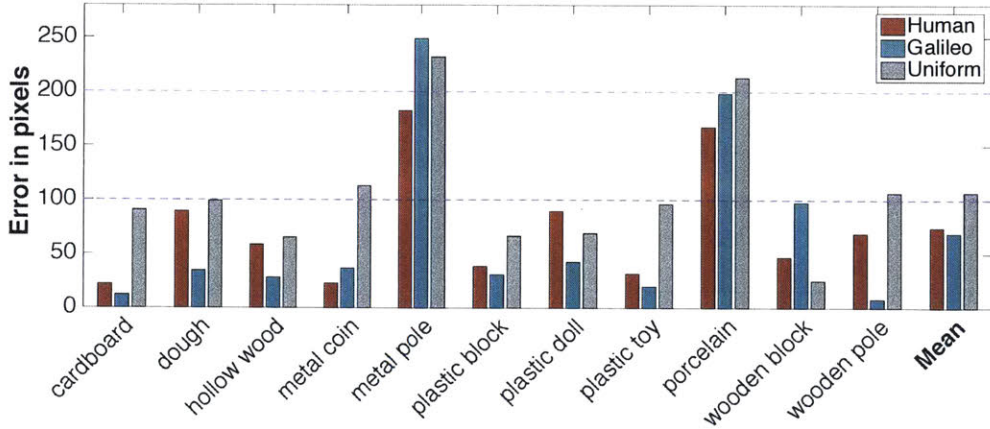
Figure 5-7: **Mean errors in the number of pixels** of human predictions, Galileo outputs, and a uniform estimate calculated by averaging ground truth ending points over all test cases. As the error patterns are similar for both target objects (foam and cardboard), the errors here are averaged across target objects for each material.
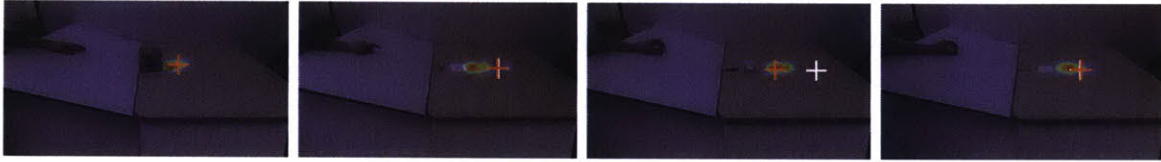
Figure 5-8: **Heat maps** of user predictions, Galileo outputs (orange crosses), and ground truths (white crosses)

(either cardboard or foam) will be after the collision, i.e., how far the target will move. We tested 30 users per case.

Given a partial video, for Galileo to generate predicted destinations, we first run it to fit the part of the video to derive our estimate of its friction coefficient. We then estimate its density by averaging the density values we derived from other objects with that material by observing collisions that they are involved. We further estimate the density (mass) and friction coefficient of the target object by averaging our estimates from other collisions. We now have all required information for the model to predict the ending point of the target after the collision. Note that the information available to Galileo is exactly the same as that available to humans.

We compare three kinds of predictions: human feedback, Galileo output, and, as a baseline, a uniform estimate calculated by averaging ground truth ending points over all test cases. Figure 5-7 shows the Euclidean distance in pixels between each of them and the ground truth. We can see that human predictions are much better than the uniform estimate, but still far from perfect. Galileo performs similar to human in the average on this task. Figure 5-8 shows, for some test cases, heat maps of user predictions, Galileo outputs (orange crosses), and ground truths (white crosses). The

Mass	Spearman's Coefficient
Human vs Galileo	0.51
Human vs Truth	0.68
Galileo vs Truth	0.52

"Will it move"	Pearson's Coefficient
Human vs Galileo	0.56
Human vs Truth	0.42
Galileo vs Truth	0.20
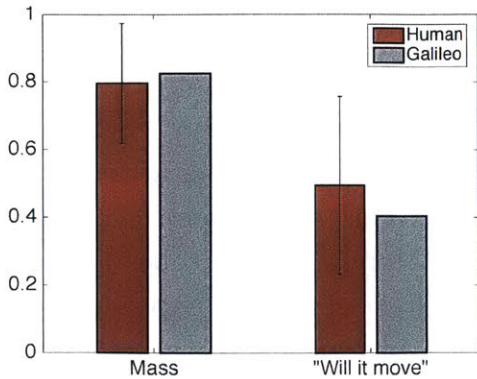
Figure 5-9: **Average accuracy of human predictions and Galileo outputs on the tasks of mass prediction and "will it move" prediction**. Error bars indicate standard deviations of human accuracies.

Table 5.2: **Correlations between pairs of outputs** in the mass prediction experiment (in Spearman's coefficient) and in the "will it move" prediction experiment (in Pearson's coefficient)

error correlation between human and POM is 0.70. The correlation analysis for the uniform model is not useful because the correlation is a constant independent of the uniform prediction value.

5.6.2   Mass Prediction

The second experiment is to predict which of two objects is heavier, after observing a video of a collision of them. For this task, we also randomly choose 50 objects, we test each of them on 50 users. For Galileo, we can directly obtain its guess based on the estimates of the masses of the objects.

Figure 5-9 demonstrates that human and our model achieve about the same accuracy on this task. We also calculate correlations between different outputs. Here for correlation analysis, we use the ratio of the masses of the two objects estimated by Galileo as its predictor. Human responses are aggregated for each trial to get the proportion of people making each decision. As the relation is highly nonlinear, we calculate Spearman's coefficients. From Table 5.2, we notice that human responses, machine outputs, and ground truths are all positively correlated.

5.6.3   "Will It Move" Prediction

Our third experiment is to predict whether a certain object will move in a different scene, after observing one of its collisions. On Amazon Mechanical Turk, we show users a video containing a collision of two objects. In this video, the angle between the inclined surface and the ground is 20 degrees. We then show users the first frame of a 10-degree video of the same object, and ask them to predict whether the object will slide down the surface in this case. We randomly choose 50 objects for the experiment,

and divide them into lists of 10 objects per user, and get each of the item tested on 50 users overall.

For Galileo, it is straightforward to predict the stability of an object in the 10-degree case using estimates from the 20-degree video. Interestingly, both humans and the model are at chance on this task (Figure 5-9), and their responses are reasonably correlated (Table 5.2). Again, here we aggregate human responses for each trial to get the proportion of people making each decision. Moreover, both subjects and the model show a bias toward saying "it will move." Future controlled experimentation and simulations will investigate what underlies this correspondence.

5.7 Discussion

The Galileo model, together with the Physics 101 dataset, accomplishes three goals: first, it shows that a generative vision system with physical object representations and a realistic 3D physics engine at its core can efficiently deal with constrained real-world data when proper recognition models and feature spaces are used. Second, it shows that humans' intuitions about physical outcomes are often accurate, and our model largely captures these intuitions—but crucially, humans and the model make similar errors. Lastly, the experience of the model, that is, the inferences it makes on the basis of dynamical visual scenes, can be used to train a deep learning model, which leads to more efficient inference and to the ability to see physical properties in the static images. Our study points toward an account of human vision with generative physical knowledge at its core, and various recognition models as helpers to induce efficient inference.

Chapter 6

Learning with an Integrated Physics + Graphics Engine

In this chapter, building upon the models introduced in previous chapters, we present a paradigm for understanding physical scenes, including both object appearance and physics, all without human annotations. At the core of our system is a physical world representation that is first recovered by a learned perception module, and then utilized by physics and graphics engines. During training, the perception module and the simulation engines learn by *visual de-animation*—interpreting and reconstructing the visual information stream. During testing, the system first recovers the physical world state, and then uses the simulators for reasoning and future prediction.

Even more so than forward simulation, inverting a physics or graphics engine is a computationally hard problem; we overcome this challenge by using a convolutional inversion network. Our system quickly recognizes the physical world state from appearance and motion cues, and has the flexibility to incorporate both differentiable and non-differentiable physics and graphics engines. We evaluate our system on both synthetic and real datasets involving multiple physical scenes, and demonstrate that our system performs well on both physical state estimation and reasoning problems.

We further show that our model can be extended to multi-part objects, where each part has distinct physical properties. Finally, we use the learned model in robot planning and manipulation, so that the system not only understands scenes, but also interact with them.

This chapter includes materials previously published as Wu et al. [2017a], Liu et al. [2018b], Janner et al. [2019]. Erika Lu, Zhijian Liu, and Michael Janner contributed significantly to the materials presented in this chapter.

6.1 Introduction

Inspired by human abilities, we wish to develop machine systems that understand scenes. Scene understanding has multiple defining characteristics which break down

broadly into two features. First, human scene understanding is *rich*. Scene understanding is physical, predictive, and causal: rather than simply knowing what is where, one can also predict what may happen next, or what actions one can take, based on the physics afforded by the objects, their properties, and relations. These predictions, hypotheticals, and counterfactuals are probabilistic, integrating uncertainty as to what is more or less likely to occur. Second, human scene understanding is *fast*. Most of the computation has to happen in a single, feedforward, bottom-up pass.

There have been many systems proposed recently to tackle these challenges, but existing systems have architectural features that allow them to address one of these features but not the other. Typical approaches based on inverting graphics engines and physics simulators [Kulkarni et al., 2015b] achieve richness at the expense of speed. Conversely, neural networks such as PhysNet [Lerer et al., 2016] are fast, but their ability to generalize to rich physical predictions is limited.

We propose a new approach to combine the best of both. Our overall framework for representation is based on graphics and physics engines, where graphics is run in reverse to build the initial physical scene representation, and physics is then run forward to imagine what will happen next or what can be done. Graphics can also be run in the forward direction to visualize the outputs of the physics simulation as images of what we expect to see in the future, or under different viewing conditions. Rather than use traditional, often slow inverse graphics methods [Kulkarni et al., 2015b], we learn to invert the graphics engine efficiently using convolutional nets. Specifically, we use deep learning to train recognition models on the objects in our world for object detection, structure and viewpoint estimation, and physical property estimation. Bootstrapping from these predictions, we then infer the remaining scene properties through inference via forward simulation of the physics engine.

Without human supervision, our system learns by *visual de-animation*: interpreting and reconstructing visual input. We show the problem formulation in Figure 6-1. The simulation and rendering engines in the framework force the perception module to extract physical world states that best explain the data. As the physical world states are inputs to physics and graphics engines, we simultaneously obtain an interpretable, disentangled, and compact physical scene representation.

Our framework is flexible and adaptable to a number of graphics and physics engines. We present model variants that use neural, differentiable physics engines [Chang et al., 2017], and variants that use traditional physics engines, which are more mature but non-differentiable [Coumans, 2010]. We also explore various graphics engines operating at different levels, ranging from mid-level cues such as object velocity, to pixel-level rendering of images.

We demonstrate our system on real and synthetic datasets across multiple domains: synthetic billiard videos [Fragkiadaki et al., 2016], in which balls have varied physical

100
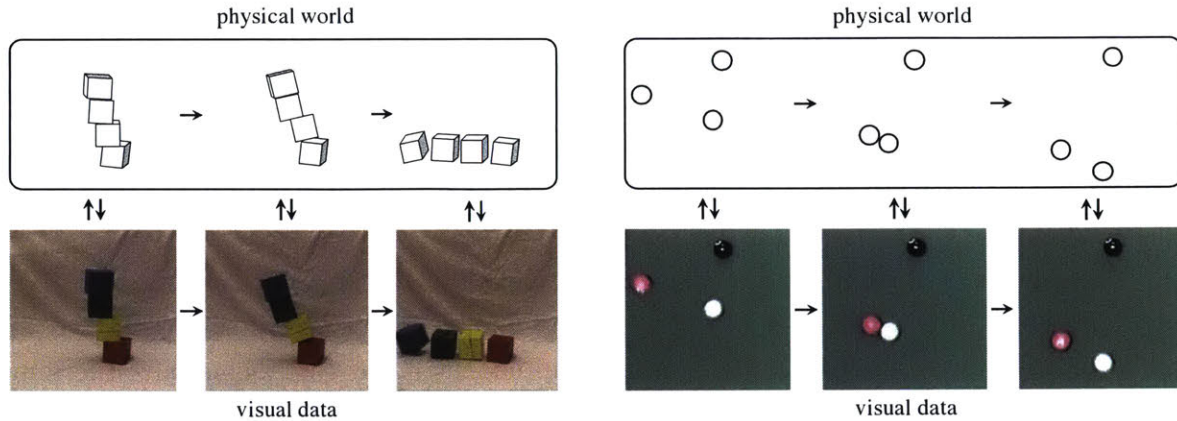
Figure 6-1: **Visual de-animation**—we would like to recover the physical world representation behind the visual input, and combine it with generative physics simulation and rendering engines.

properties, real billiard videos from the web, and real images of block towers from Facebook AI Research [Lerer et al., 2016].

Later in the chapter, we also present extensions to our model that handle multi-part objects, where each part has distinct physical properties. Finally, we present applications of our model in robot planning and manipulation.

Our contributions in this chapter are four-fold. First, we introduce the problem of visual de-animation—learning rich scene representations without supervision by interpreting and reconstructing visual input. Second, we propose a novel generative pipeline for physical scene understanding, and demonstrate its flexibility by incorporating various graphics and physics engines. Third, we show that our system performs well across multiple scenarios on both synthetic and constrained real videos, including those involving heterogeneous objects. Last, we present extensions of our model that can be deployed on real robots for planning and manipulation.

6.2 Related Work

Physical scene understanding has attracted increasing attention in recent years [Gupta et al., 2010, Jia et al., 2015, Lerer et al., 2016, Zheng et al., 2015, Battaglia et al., 2013, Mottaghi et al., 2016b, Fragkiadaki et al., 2016, Battaglia et al., 2016, Mottaghi et al., 2016a, Chang et al., 2017, Agrawal et al., 2016, Pinto et al., 2016, Finn et al., 2016, Hamrick et al., 2017, Ehrhardt et al., 2019, Shao et al., 2014, Zhang et al., 2016]. Researchers have attempted to go beyond the traditional goals of high-level computer vision, inferring "what is where", to capture the physics needed to predict the immediate future of dynamic scenes, and to infer the actions an agent should take to achieve a goal. Most of these efforts do not attempt to learn physical object representations from raw observations. Some systems emphasize learning from pixels

but without an explicitly object-based representation [Lerer et al., 2016, Fragkiadaki et al., 2016, Agrawal et al., 2016, Pinto et al., 2016, Li et al., 2017b], which makes generalization challenging. Others learn a flexible model of the dynamics of object interactions, but assume a decomposition of the scene into physical objects and their properties rather than learning directly from images [Chang et al., 2017, Battaglia et al., 2016].

There have been some models that aim to estimate physical object properties [Wu et al., 2016a, 2015a, Denil et al., 2017]. The Galileo model we just introduced in Chapter 5 explored an analysis-by-synthesis approach that is easily generalizable, but less efficient. Their framework also lacked a perception module. Denil et al. [2017] instead proposed a reinforcement learning approach. These approaches, however, assumed strong priors of the scene and approximated object shapes with primitives. Wu et al. [2016a] used a feed-forward network for physical property estimation without assuming prior knowledge of the environment, but the constrained setup did not allow interactions between multiple objects. By incorporating physics and graphics engines, our approach can jointly learn the perception module and physical model, optionally in a Helmholtz machine style [Hinton et al., 1995], and recover an explicit physical object representation in a range of scenarios.

Another line of related work is on future state prediction in either image pixels [Xue et al., 2016, Mathieu et al., 2016] or object trajectories [Kitani et al., 2017, Walker et al., 2015]. Our model builds upon and extends these ideas by jointly modeling an approximate physics engine and a perceptual module, with wide applications including, but not limited to, future prediction.

Our model also connects to the area of "vision as inverse graphics" [Zhu and Mumford, 2007, Yuille and Kersten, 2006, Bai et al., 2012]. Unlike traditional analysis-by-synthesis approaches, recent methods explored using deep neural networks to efficiently explain an object [Kulkarni et al., 2015a, Rezende et al., 2016a], or a scene with multiple objects [Ba et al., 2015, Huang and Murphy, 2015, Eslami et al., 2016]. In particular, in Chapter 4 we have proposed "scene de-rendering", building an object-based, structured representation from a static image. In this chapter, we develop a model that incorporates inverse graphics with simulation engines for physical scene understanding and scene dynamics modeling.

6.3   Visual De-animation

Our visual de-animation (VDA) model consists of an efficient inverse graphics component to build the initial physical world representation from visual input, a physics engine for physical reasoning of the scene, and a graphics engine for rendering videos. We show the framework in Figure 6-2. In this section, we first present an
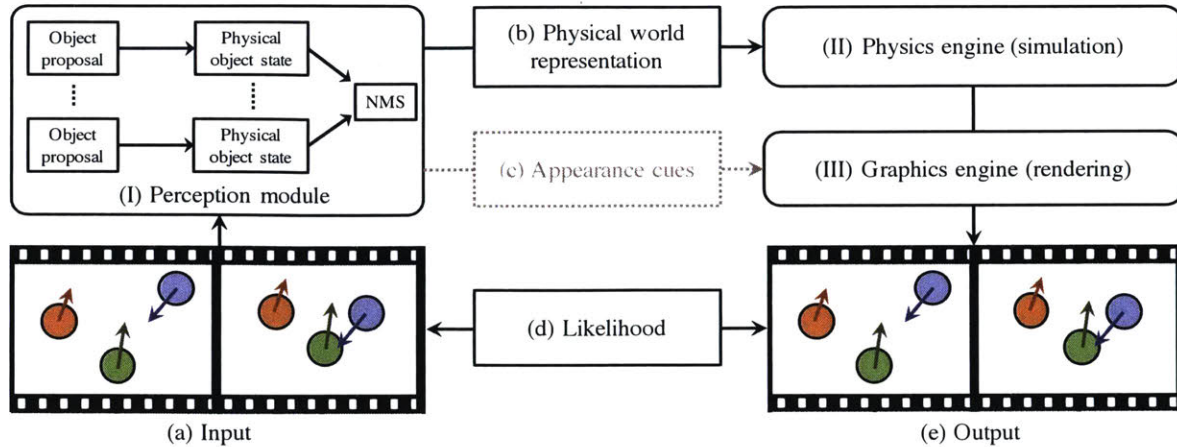
Figure 6-2: **Our visual de-animation (VDA) model contains three major components**: a convolutional perception module (I), a physics engine (II), and a graphics engine (III). The perception module efficiently inverts the graphics engine by inferring the physical object state for each segment proposal in input (a), and combines them to obtain a physical world representation (b). The generative physics and graphics engines then run forward to reconstruct the visual data (e). See Section 6.3 for details.

overview of the system, and then describe each component in detail.

6.3.1 Overview

The first component of our system is an approximate inverse graphics module for physical object and scene understanding, as shown in Figure 6-2-I. Specifically, the system sequentially computes object proposals, recognizes objects and estimates their physical state, and recovers the scene layout.

The second component of our system is a physics engine, which uses the physical scene representation recovered by the inverse graphics module to simulate future dynamics of the environment (Figure 6-2-II). Our system adapts to both neural, differentiable simulators, which can be jointly trained with the perception module, and rigid-body, non-differentiable simulators, which can be incorporated using methods such as REINFORCE [Williams, 1992].

The third component of our framework is a graphics engine (Figure 6-2-III), which takes the scene representations from the physics engine and re-renders the video at various levels (e.g. optical flow, raw pixel). The graphics engine may need additional appearance cues such as object shape or color (Figure 6-2c). Here, we approximate them using simple heuristics, as they are not a focus of our model. There is a tradeoff between various rendering levels: while pixel-level reconstruction captures details of the scene, rendering at a more abstract level (e.g. silhouettes) may better generalize. We then use a likelihood function (Figure 6-2d) to evaluate the difference between synthesized and observed signals, and compute gradients or rewards for differentiable and non-differentiable systems, respectively.

Our model combines efficient and powerful deep networks for recognition with rich simulation engines for forward prediction. This provides us two major advantages over existing methods: first, simulation engines take an interpretable representation of the physical world, and can thus easily generalize and supply rich physical predictions; second, the model learns by explaining the observations—it can be trained in a self-supervised manner without requiring human annotations.

6.3.2 Physical Object and Scene Modeling

We now discuss each component in detail, starting with the perception module.

Object proposal generation. Given one or a few frames (Figure 6-2a), we first generate a number of object proposals. The masked images are then used as input to the following stages of the pipeline.

Physical object state estimation. For each segment proposal, we use a convolutional network to recognize the physical state of the object, which consists of intrinsic properties such as shape, mass, and friction, as well as extrinsic properties such as 3D position and pose. The input to the network is the masked image of the proposal, and the output is an interpretable vector for its physical state.

Physical world reconstruction. Given objects' physical states, we first apply nonmaximum suppression to remove object duplicates, and then reconstruct the physical world according to object states. The physical world representation (Figure 6-2b) will be employed by the physics and graphics engines for simulation and rendering.

6.3.3 Physical Simulation and Prediction

The two types of physics engines we explore in this chapter include a neural, differentiable physics engine and a standard rigid-body simulation engine.

Neural physics engines. The neural physics engine is an extension of the recent work from Chang et al. [2017], which simulates scene dynamics by taking object mass, position, and velocity. We extend their framework to model object friction in our experiments on billiard table videos. Though basic, the neural physics engine is differentiable, and thus can be end-to-end trained with our perception module to explain videos. Please refer to Chang et al. [2017] for details of the neural physics engine.

Rigid body simulation engines. There exist rather mature, rigid-body physics simulation engines, e.g. Bullet [Coumans, 2010]. Such physics engines are much more powerful, but non-differentiable. In our experiments on block towers, we used a non-differentiable simulator with multi-sample REINFORCE [Rezende et al., 2016a, Mnih and Rezende, 2016] for joint training.

104

(a) shared appearance,
shared physics

(b) varied appearance,
varied physics

(c) shared appearance,
varied physics
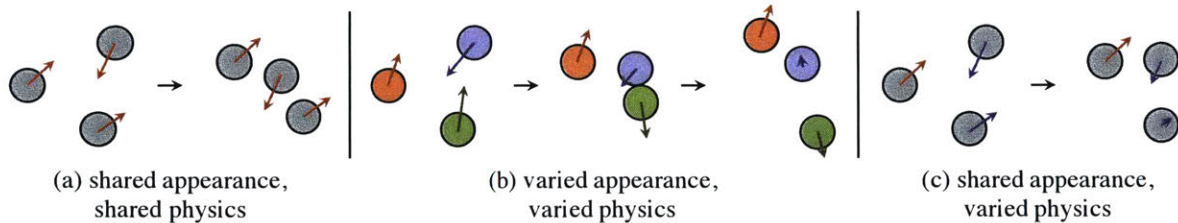
Figure 6-3: **The three settings of our synthetic billiard videos**: (a) balls have the same appearance and physical properties, where the system learns to discover them and simulate the dynamics; (b) balls have the same appearance but different physics, and the system learns their physics from motion; (c) balls have varied appearance and physics, and the system learns to associate appearance cues with underlying object states, even from a single image.

6.3.4 Re-rendering with a Graphics Engine

In this work, we consider two graphics engines operating at different levels: for the billiard table scenario, we use a renderer that takes the output of a physics engine and generates pixel-level rendering; for block towers, we use one that computes only object silhouettes.

6.4 Evaluation

We evaluate variants of our frameworks on videos of billiard tables and block towers. We also test how models trained on synthetic data generalize to real cases.

6.4.1 Billiard Tables: A Motivating Example

We begin with synthetic billiard videos to explore end-to-end learning of the perceptual module along with differentiable simulation engines. We explore how our framework learns the physical object state (position, velocity, mass, and friction) from its appearance and/or motion.

Data. For the billiard table scenario, we generate data using the released code from Fragkiadaki et al. [2016]. We updated the code to allow balls of different mass and friction. We used the billiard table scenario as an initial exploration of whether our models can learn to associate visual object appearance and motion with physical properties. As shown in Figure 6-3, we generated three subsets, in which balls may have shared or differing appearance (color), and physical properties. For each case, we generated 9,000 videos for training and 200 for testing.

(I) Shared appearance and physics (Figure 6-3a): balls all have the same appearance and the same physical properties. This basic setup evaluates whether we can jointly learn an object (ball) discoverer and a physics engine for scene dynamics.

(II) Varied appearance and physics (Figure 6-3b): balls can be of three different masses (light, medium, heavy), and two different friction coefficients. Each of the six
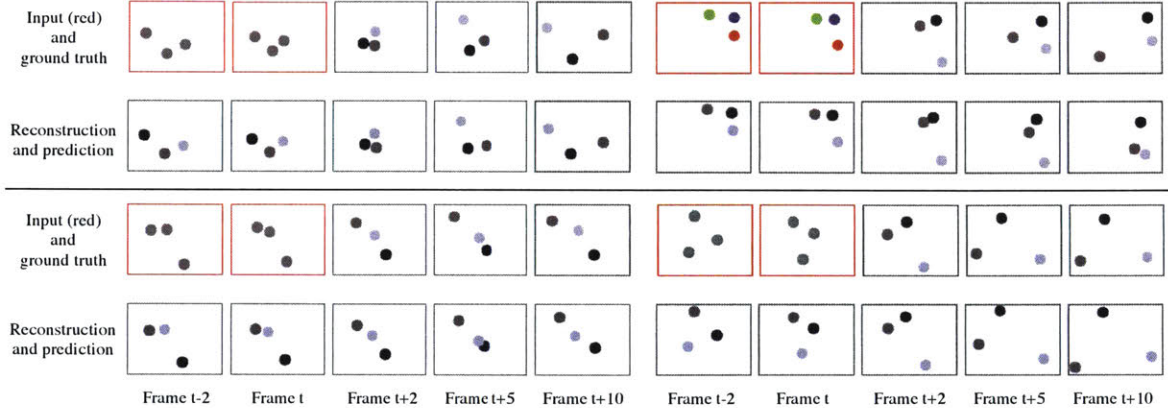
Figure 6-4: **Qualitative results on the billiard videos**, comparing ground truth videos with our predictions. We show two of three input frames (in red) due to space constraints. Left: balls share appearance and physics (I), where our framework learns to discover objects and simulate scene dynamics. Top right: balls have different appearance and physics (II), where our model learns to associate appearance with physics and simulate collisions. It learns that the green ball should move further than the heavier blue ball after the collision. Bottom right: balls share appearance but have different frictions (III), where our model learns to associate motion with friction. It realizes from three input frames that the right-most ball in the first frame has a large friction coefficient and will stop before the other balls.

possible combinations is associated with a unique color (appearance). In this setup, the scene de-rendering component should be able to associate object appearance with its physical properties, even from a single image.

(III) Shared appearance, varied physics (Figure 6-3c): balls have the same appearance, but have one of two different friction coefficients. Here, the perceptual component should be able to associate object motion with its corresponding friction coefficients, from just a few input images.

Setup. For this task, the physical state of an object is its intrinsic properties, including mass m and friction f, and its extrinsic properties, including 2D position $\{x, y\}$ and velocity v. Our system takes three 256×256 RGB frames I_1, I_2, I_3 as input. It first obtains flow fields from I_1 to I_2 and from I_2 to I_3 by a pre-trained spatial pyramid network (SPyNet) [Ranjan and Black, 2017]. It then generates object proposals by applying color filters on input images.

Our perceptual model is a ResNet-18 [He et al., 2016], which takes as input three masked RGB frames and two masked flow images of each object proposal, and recovers the object's physical state. We use a differentiable, neural physics engine with object intrinsic properties as parameters; at each step, it predicts objects' extrinsic properties (position $\{x, y\}$ and velocity v) in the next frame, based on their current estimates. We employ a graphics engine that renders original images from the predicted positions, where the color of the balls is set as the mean color of the input object proposal. The likelihood function compares, at a pixel level, these rendered images and observations.
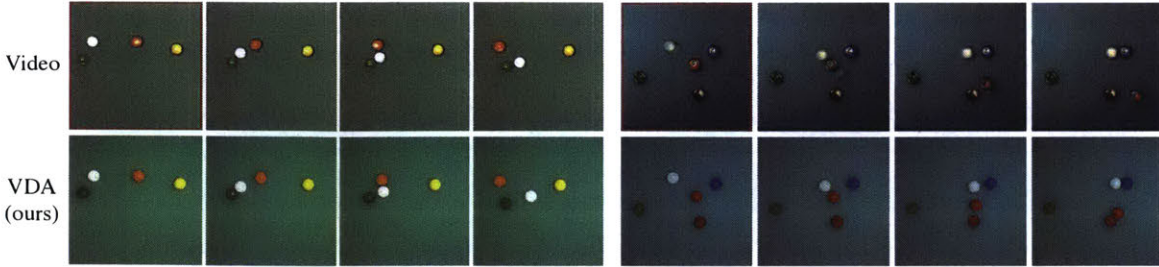
Figure 6-5: **Sample results on web videos of real billiard games and computer games with realistic rendering**. Left: our method correctly estimates the trajectories of multiple objects. Right: our framework correctly predicts the two collisions (white vs. red, white vs. blue), despite the motion blur in the input, though it underestimates the velocity of the red ball after the collision. Note that the billiard table is a chaotic system, and highly accurate long-term prediction is intractable.

It is straightforward to compute the gradient of object position from rendered RGB images and ground truth. Thus, this simple graphics engine is also differentiable, making our system end-to-end trainable.

Our training paradigm consists of two steps. First, we pre-train the perception module and the neural physics engine separately on synthetic data, where ground truth is available. The second step is end-to-end fine-tuning without annotations. We observe that the framework does not converge well without pre-training, possibly due to the multiple hypotheses that can explain a scene (e.g., we can only observe relative, not absolute masses from collisions). We train our framework using SGD, with a learning rate of 0.001 and a momentum of 0.9. We implement our framework in Torch7 [Collobert et al., 2011]. During testing, the perception module is run in reverse to recover object physical states, and the learned physics engine is then run in forward for future prediction.

Results. Our formulation recovers a rich representation of the scene. With the generative models, we show results in scene reconstruction and future prediction. On scene reconstruction, given input frames, our model can reconstruct the images based on inferred physical states. We show qualitative results in Figure 6-4. On future prediction, with the learned neural simulation engine, our system is able to predict future events based on physical world states. We show qualitative results in Figure 6-4. Our model achieves good performance in reconstructing the scene, understanding object physics, and predicting scene dynamics.

6.4.2 Billiard Tables: Transferring to Real Videos

Data. We also collected videos from YouTube, segmenting them into two-second clips. Some videos are from real billiard competitions, and the others are from computer games with realistic rendering. We use it as an out-of-sample test set for evaluating the model's generalization ability.

Setup and results. Our setup is the same as that in Section 6.4.1, except that we now re-train the perceptual model on the synthetic data of varied physics, but with flow images as input instead of RGB images. Flow images abstract away appearance changes (color, lighting, etc.), allowing the model to generalize better to real data. We show qualitative results of reconstruction and future prediction in Figure 6-5 by rendering our inferred representation using the graphics software, Blender.

6.4.3 The Blocks World

We now look into a different scenario—block towers. In this experiment, we demonstrate the applicability of our model to explain and reason from a static image, instead of a video. We focus on the reasoning of object states in the 3D world, instead of physical properties such as mass. We also explore how our framework performs with non-differentiable simulation engines, and how physics signals (e.g., stability) could help in physical reasoning, even when given only a static image.

Data. Lerer et al. [2016] built a dataset of 492 images of real block towers, with ground truth stability values. Each image may contain 2, 3, or 4 blocks of red, blue, yellow, or green color. Though the blocks are the same size, their sizes in each 2D image differ due to 3D-to-2D perspective transformation. Objects are made of the same material and thus have identical mass and friction.

Setup. Here, the physical state of an object (block) consists of its 3D position $\{x, y, z\}$ and 3D rotation (roll, pitch, yaw, each quantized into 20 bins). Our perceptual model is again a ResNet-18 [He et al., 2016], which takes block silhouettes generated by simple color filters as input, and recovers the object's physical state. For this task, we implement an efficient, non-differentiable, rigid body simulator, to predict whether the blocks are stable. We also implement a graphics engine to render object silhouettes for reconstructing the input. Our likelihood function consists of two terms: MSE between rendered silhouettes and observations, and the binary cross-entropy between the predicted stability and the ground truth stability.

Our training paradigm resembles the classic wake-sleep algorithm [Hinton et al., 1995]: first, generate 10,000 training images using the simulation engines; second, train the perception module on synthetic data with ground truth physical states; third, end-to-end fine-tuning of the perceptual module by explaining an additional 100,000 synthetic images without annotations of physical states, but with binary annotations of stability. We use multi-sample REINFORCE [Rezende et al., 2016a, Mnih and Rezende, 2016] with 16 samples per input, assuming each position parameter is from a Gaussian distribution and each rotation parameter is from a multinomial distribution (quantized into 20 bins). We observe that the training paradigm helps the framework converge. The other setting is the same as that in Section 6.4.1.
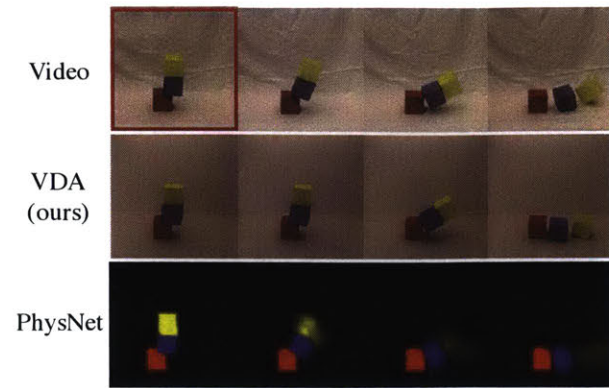
(a) Reconstruction and prediction given a single frame (marked in red). From top to bottom: ground truth, our results, results from Lerer et al. [2016].

Methods	# Blocks			Mean
	2	3	4	
Chance	50	50	50	50
Humans	67	62	62	64
PhysNet	66	66	73	68
GoogLeNet	70	70	70	70
VDA (init)	73	74	72	73
VDA (joint)	75	76	73	75
VDA (full)	76	76	74	75

(b) Accuracy (%) of stability prediction on the blocks dataset

Methods	2	3	4	Mean
PhysNet	56	68	70	65
GoogLeNet	70	67	71	69
VDA (init)	74	74	67	72
VDA (joint)	75	77	70	74
VDA (full)	76	76	72	75

(c) Accuracy (%) of stability prediction when trained on synthetic towers of 2 and 4 blocks, and tested on all block tower sizes.
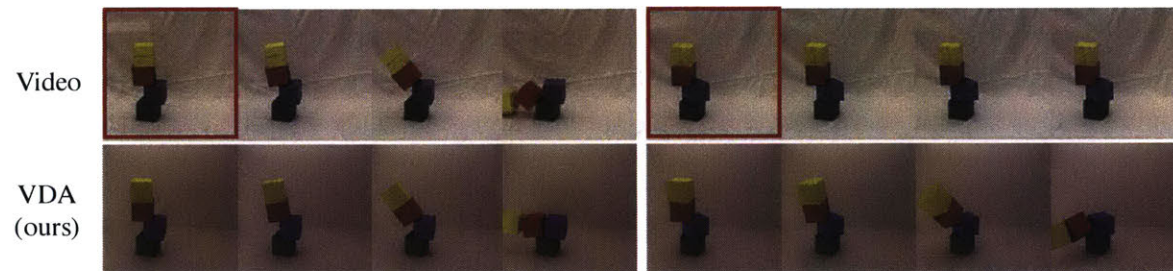
(d) Our reconstruction and prediction results given a single frame (marked in red)

Figure 6-6: **Results on the blocks dataset** [Lerer et al., 2016]. For quantitative results (b), we compare three variants of our visual de-animation (VDA) model: perceptual module trained without fine-tuning (init), joint fine-tuning with REINFORCE (joint), and full model considering stability constraint (full). We also compare with PhysNet [Lerer et al., 2016] and GoogLeNet [Szegedy et al., 2015].

Results. We show results on two tasks: scene reconstruction and stability prediction. For each task, we compare three variants of our algorithm: the initial system has its perception module trained without fine-tuning; an intermediate system has joint end-to-end fine-tuning, but without considering the physics constraint; and the full system considers both reconstruction and physical stability during fine-tuning.
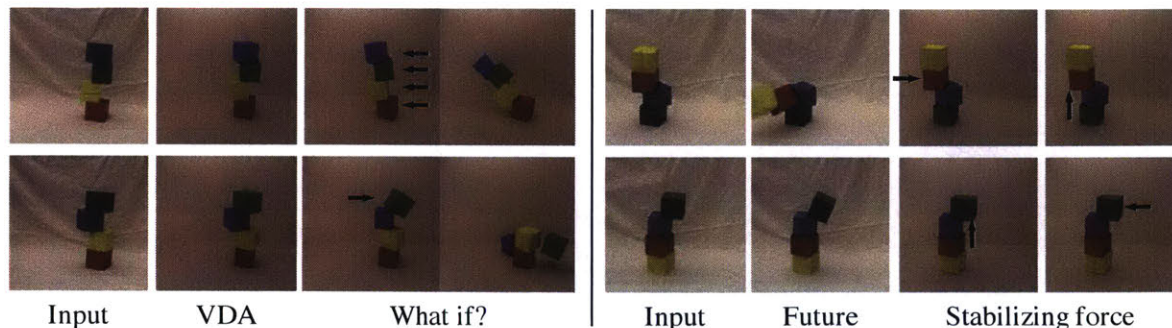
Figure 6-7: **Predicting hypothetical scenarios and actively engaging with the scene**. Left: predictions of the outcome of forces applied to two stable towers. Right: multiple ways to stabilize two unstable towers.

We show qualitative results on scene reconstruction in Figures 6-6a and 6-6d, where we also demonstrate future prediction results by exporting our inferred physical states into Blender. We show quantitative results on stability prediction in Table 6-6b, where we compare our models with PhysNet [Lerer et al., 2016] and GoogleNet [Szegedy et al., 2015]. All given a static image as test input, our algorithms achieve higher prediction accuracy (75% vs. 70%) efficiently (<10 milliseconds per image).

Our framework also generalizes well. We test out-of-sample generalization ability, where we train our model on 2- and 4-block towers, but test it on all tower sizes. We show results in Table 6-6c. Further, in Figure 6-7, we show examples where our physical scene representation combined with a physics engine can easily make conditional predictions, answering "What happens if..."-type questions. Specifically, we show frame prediction of external forces on stable block towers, as well as ways that an agent can stabilize currently unstable towers, with the help of rich simulation engines.

6.5 Extension to Heterogeneous Objects

Many daily objects such as hammers consist of multiple parts, each with distinct physical properties. In this section, we extend our model to further decompose objects into parts. We call this problem Physical Primitive Decomposition (PPD).

6.5.1 Problem Statement

Both primitive decomposition and physical primitive decomposition attempt to approximate an object with primitives. We highlight their difference in Figure 6-8.

Primitive decomposition. As formulated in Tulsiani et al. [2017a] and Zou et al. [2017], primitive decomposition aims to decompose an object O into a set of simple transformed primitives $x = \{x_k\}$ so that these primitives can accurately approximate

110

(a) Primitive decomposition   (b) Physical primitive decomposition
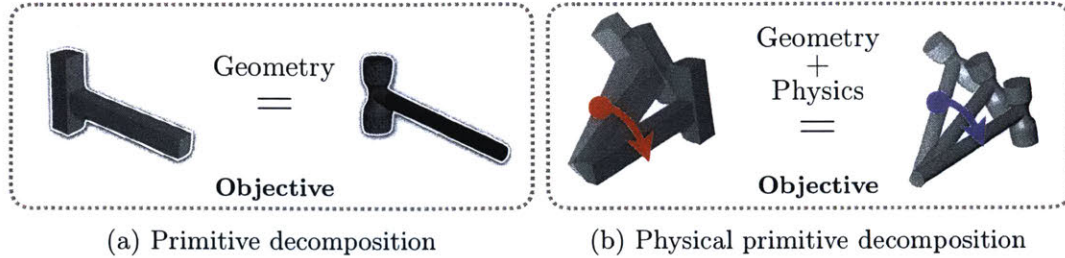
Figure 6-8: **Primitive decomposition (a) and physical primitive decomposition (b)**. Both tasks attempt to convert an object into a set of primitives yet with different purposes: the former problem targets at shape reconstruction, while the latter one aims to recover both geometric and physical properties.

its geometry shape. This task can be seen as to minimize

$$\mathcal{L}_G(x) = \mathcal{D}_S\big(\mathcal{S}(\underset{k}{\cup} x_k), \mathcal{S}(O)\big), \tag{6.1}$$

where $\mathcal{S}(\cdot)$ denotes the geometry shape (i.e. point cloud), and $\mathcal{D}_S(\cdot, \cdot)$ denotes the distance metric between shapes.

Physical primitive decomposition. In order to understand the functionality of object parts, we require the decomposed primitives $x = \{x_k\}$ to also approximate the physical behavior of object O. To this end, we extend the previous objective function with an additional physics term:

$$\mathcal{L}_P(x) = \sum_{p \in \mathcal{P}} \mathcal{D}_T\big(\mathcal{T}_p(\underset{k}{\cup} x_k), \mathcal{T}_p(O)\big), \tag{6.2}$$

where $\mathcal{T}_p(\cdot)$ denotes the trajectory after physics interaction p, $\mathcal{D}_T(\cdot, \cdot)$ denotes the distance metric between trajectories (i.e. mean squared error), and $\mathcal{P}$ denotes a predefined set of physics interactions. Therefore, the task of physical primitive decomposition is to minimize an overall objective function constraining both geometry and physics: $\mathcal{L}(x) = \mathcal{L}_G(x) + w \cdot \mathcal{L}_P(x)$, where w is a weighting factor.

Primitive-based representation. We design a structured primitive-based object representation, which describes an object by listing all of its primitives with different attributes. For each primitive x_k, we record its size $x_k^S = (s_x, s_y, s_z)$, position in 3D space $x_k^T = (p_x, p_y, p_z)$, rotation in quaternion form $x_k^R = (q_w, q_x, q_y, q_z)$. Apart from these geometry information, we also track its physical properties: density x_k^D.

In our object representation, the shape parameters, x_k^S, x_k^T and x_k^R, are vectors of continuous real values, whereas the density parameter x_k^D is a discrete value. We discretize the density values into $N_D = 100$ slots, so that estimating density becomes a N_D-way classification. Discretization helps to deal with multi-modal density values. Figure 6-9a shows that two parts with similar visual appearance may have very

(a) **Above**: Aluminum and Wood; **Below**: Iron and Wood.

(b) **Above**: Wood and Iron; **Below**: Two Coppers.
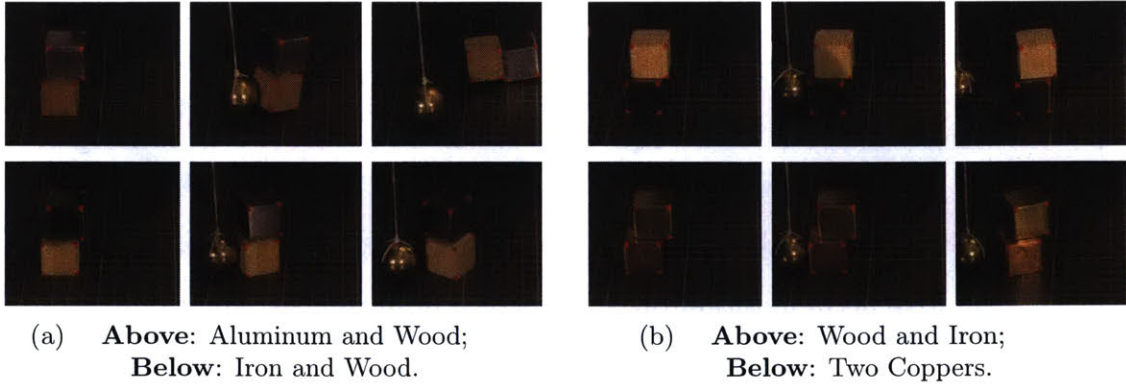
Figure 6-9: **Challenges of inferring physical parameters from visual and physical observations**: objects with different physical parameters might have **(a)** similar visual appearance or **(b)** similar physics trajectory.

different physical parameters. In such cases, regression with an $\mathcal{L}_2$ loss will encourage the model to predict the *mean* value of possible densities; in contrast, discretization allows it to give high probabilities to every possible density. We then figure out which candidate value is optimal from the trajectories.

6.5.2 Approach

Inferring physical parameters from solely visual or physical observation is highly challenging. This is because two objects with different physical parameters might have similar visual appearance (Figure 6-9a) or have similar physics trajectories (Figure 6-9b). Therefore, our model takes both types of observations as input:

1. **Visual Observation**. We take a voxelized shape and an image as our input because they can provide us with valuable visual information. Voxels help us recover object geometry, and images contain texture information of object materials. Note that, even with voxels as input, it is still highly nontrivial to infer geometric parameters: the model needs to learn to segment 3D parts within the object—an unsolved problem by itself [Tulsiani et al., 2017a].

2. **Physics Observation**. In order to explain the physical behavior of an object, we also need to observe its response after some physics interactions. In this work, we choose to use 3D object trajectories rather than RGB (or RGB-D) videos. Its abstractness enables the model to transfer better from synthetic to real data, because synthetic and real videos can be starkly different; in contrast, itâĂŹs easy to generate synthetic 3D trajectories that look realistic.

Specifically, as shown in Figure 6-10 our network takes a voxel V, an image I, and N_{T} object trajectories $\boldsymbol{T} = \{T_k\}$ as input. V is a 3D binary voxelized grid, I is a single RGB image, and $\boldsymbol{T}$ consists of several object trajectories T_k, each of which

112
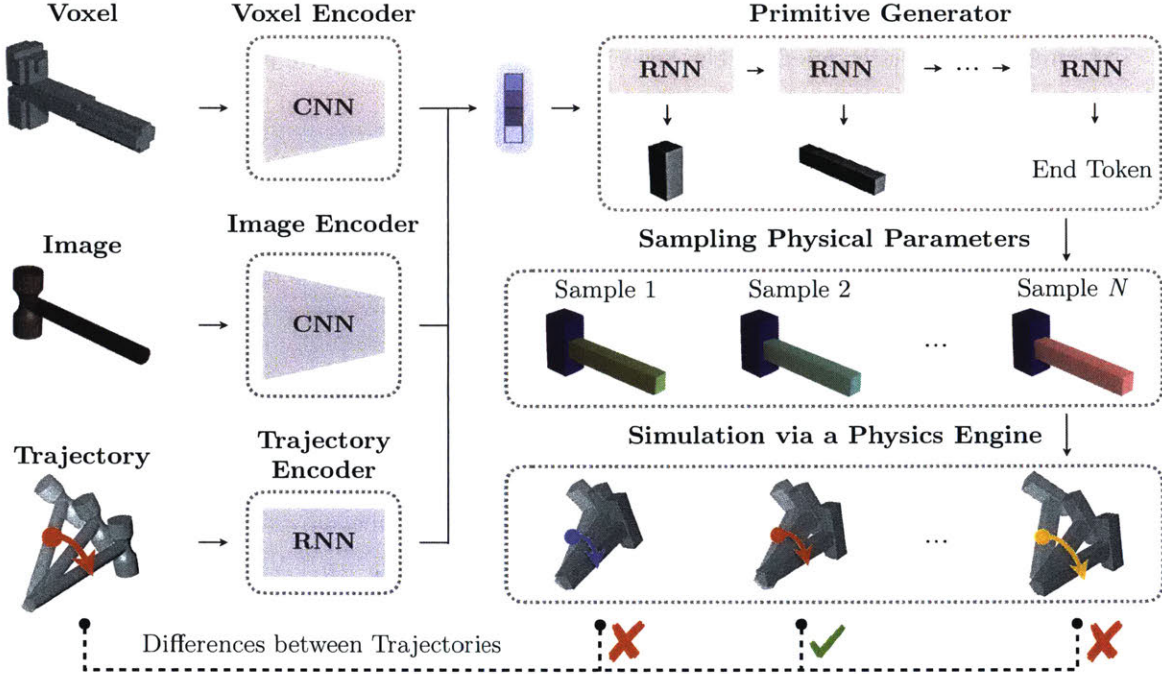
Figure 6-10: **Overview of our Physical Primitive Decomposition (PPD) model**

records the response to one specific physics interaction. Trajectory T_k is a sequence of 3D object pose $(p_x, p_y, p_z, q_w, q_x, q_y, q_z)$, where (p_x, p_y, p_z) denotes the object's center position and quaternion (q_w, q_x, q_y, q_z) denotes its rotation at each time step.

After receiving the inputs, our network encodes voxel, image and trajectory with separate encoders, and sequentially predicts primitives using a recurrent primitive generator. For each primitive, the network predicts its geometry shape (i.e., scale, translation and rotation) and physical property (i.e., density).

For input voxel V, we employ a 3D volumetric convolutional network to encode the 3D shape information into a voxel feature f_V.

For input image I, we pass it into the ResNet-18 [He et al., 2016] encoder to obtain an image feature f_I. We refer the readers to He et al. [2016] for details.

For input trajectories $\boldsymbol{T}$, we encode each trajectory T_k into a low-dimensional feature vector h_k with a separate bi-directional recurrent neural network. Specifically, we feed the trajectory sequence, T_k, and also the same trajectory sequence in reverse order, T_k^{reverse}, into two encoding RNNs, to obtain two final hidden states: $\overrightarrow{h_k} = \overrightarrow{\text{encode}}_k(T_k)$ and $\overleftarrow{h_k} = \overleftarrow{\text{encode}}_k(T_k^{\text{reverse}})$. We take $[\overrightarrow{h_k}; \overleftarrow{h_k}]$ as the feature vector h_k. Finally, we concatenate the features of each trajectory, $\{h_k \mid k = 1, 2, \ldots, N_{\text{T}}\}$, and project it into a low-dimensional trajectory feature f_{T} with a fully-connected layer.

We concatenate the voxel feature f_V, image feature f_I and trajectory feature f_{T} together as $\hat{f} = [f_V; f_I; f_{\text{T}}]$, and map it to a low-dimensional feature f using a fully-connected layer. We predict the set of physical primitives $\{x_k\}$ sequentially by a

113

recurrent generator.

At each time step k, we feed the previous generated primitive x_{k-1} and the feature vector f in as input, and we receive one hidden vector h_k as output. Then, we compute the new primitive $x_k = (x_k^{\mathrm{D}}, x_k^{\mathrm{S}}, x_k^{\mathrm{T}}, x_k^{\mathrm{R}})$ as

$$
\begin{aligned}
x_k^{\mathrm{D}} &= \mathrm{softmax}(W_{\mathrm{D}} \times h_k + b_{\mathrm{D}}), \quad x_k^{\mathrm{S}} = \mathrm{sigmoid}(W_{\mathrm{S}} \times h_k + b_{\mathrm{S}}) \times C_{\mathrm{S}}, \\
x_k^{\mathrm{T}} &= \tanh(W_{\mathrm{T}} \times h_k + b_{\mathrm{T}}) \times C_{\mathrm{T}}, \quad x_k^{\mathrm{R}} = \frac{W_{\mathrm{R}} \times h_k + b_{\mathrm{R}}}{\max(\|W_{\mathrm{R}} \times h_k + b_{\mathrm{R}}\|_2, \epsilon)},
\end{aligned} \tag{6.3}
$$

where C_{S} and C_{T} are scaling factors, and $\epsilon = 10^{-12}$ is a small constant for numerical stability. Equation 6.3 guarantees that x_k^{S} is in the range of $[0, C_{\mathrm{S}}]$, x_k^{T} is in the range of $[-C_{\mathrm{T}}, C_{\mathrm{T}}]$, and $\|x_k^{\mathrm{R}}\|_2$ is 1 (if ignoring ϵ), which ensures that x_k will always be a valid primitive. In our experiments, we set $C_{\mathrm{S}} = C_{\mathrm{T}} = 0.5$, since we normalize all objects so that they can fit in unit cubes. Also note that, x_k^{D} is an $(N_{\mathrm{D}} + 2)$-dimensional vector, where the first N_{D} dimensions indicate different density values and the last two indicate the "start token" and "end token".

Sampling and simulating with a physics engine. During testing time, we treat the predicted x_k^{D} as a multinomial distribution, and we sample multiple possible predictions from it. For each sample, we use its physical parameters to simulate the trajectory with a physics engine. Finally, we select the one whose simulated trajectory is closest to the observed trajectory.

An alternative way to incorporate physics engine is to directly optimize our model over it. As most physics engines are not differentiable, we employ REIN-FORCE [Williams, 1992] for optimization. Empirically, we observe that this reinforcement learning–based method performs worse than sampling-based methods, possibly due to the large variance of the approximate gradient signals.

Simulating with a physics engine requires we know the force during testing. Such an assumption is essential to ensure the problem is well-posed: without knowing the force, we can only infer the relative part density, but not the actual values. Note that in many real-world applications such as robot manipulation, the external force is indeed available.

Loss functions. Let $x = (x_1, x_2, \ldots, x_n)$ and $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m)$ be the predicted and ground-truth physical primitives, respectively. Our loss function consists of two terms, geometry loss $\mathcal{L}_{\mathrm{G}}$ and physics loss $\mathcal{L}_{\mathrm{D}}$:

$$
\mathcal{L}_{\mathrm{G}}(x, \hat{x}) = \sum_k \left(\omega_{\mathrm{S}} \cdot \|x_k^{\mathrm{S}} - \hat{x}_k^{\mathrm{S}}\|_1 + \omega_{\mathrm{T}} \cdot \|x_k^{\mathrm{T}} - \hat{x}_k^{\mathrm{T}}\|_1 + \omega_{\mathrm{R}} \cdot \|x_k^{\mathrm{R}} - \hat{x}_k^{\mathrm{R}}\|_1 \right), \tag{6.4}
$$

$$
\mathcal{L}_{\mathrm{P}}(x, \hat{x}) = -\sum_k \sum_i \hat{x}_k^{\mathrm{D}}(i) \cdot \log x_k^{\mathrm{D}}(i), \tag{6.5}
$$

114
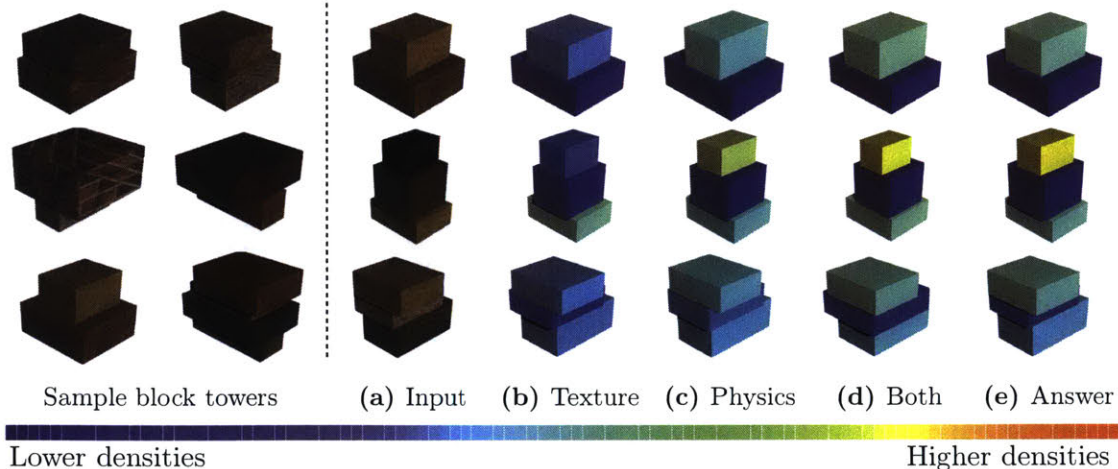
Figure 6-11: **Results on the block dataset.** Left: sample objects in our block towers dataset. Right: qualitative results of our model with different combinations of observations as input.

Material	Wood	Brick	Stone	Ceramic	Metal
Density	$[1, 10]$	$[11, 20]$	$[21, 30]$	$[31, 60]$	$[21, 35] \cup [71, 100]$

Table 6.1: **Materials and their real-world density values** (unit: $\times 10^2 \cdot \mathrm{kg/m^3}$). Objects made of similar materials (different types of metals) may have different physical properties, while different materials (i.e., stone and metal) may have same physical properties.

where ω_S, ω_T and ω_R are weighting factors, which are set to 1's because x^S, x^T and x^R are of the same magnitude (10^{-1}) in our datasets. Integrating Equation 6.4 and Equation 6.5, we define the overall loss function as $\mathcal{L}(x, \hat{x}) = \mathcal{L}_\mathrm{G}(x, \hat{x}) + w \cdot \mathcal{L}_\mathrm{P}(x, \hat{x})$, where w is set to ensure that $\mathcal{L}_G$ and $\mathcal{L}_P$ are of the same magnitude.

6.5.3 Decomposing Block Towers

We build the block towers by stacking variable number of blocks (2-5 in our experiments) together. We first sample the size of each block and then compute the center position of blocks from bottom to top. For the kth block, we denote the size as (w_k, h_k, d_k), and its center (x_k, y_k, z_k) is sampled and computed by $x_k \sim \mathcal{N}(x_{k-1}, w_{k-1}/4)$, $y_k \sim \mathcal{N}(y_{k-1}, h_{k-1}/4)$, and $z_k = z_{k-1} + (d_{k-1} + d_k)/2$, where $\mathcal{N}(\mu, \sigma)$ is a normal distribution with mean μ and standard deviation σ. We illustrate some constructed block towers in Figure 6-11. We perform the exact voxelization with grid size of $32 \times 32 \times 32$ by binvox, a 3D mesh voxelizer [Nooruddin and Turk, 2003].

Materials. In our experiments, we use five different materials, and follow their real-world densities with minor modifications. The materials and the ranges of their densities are listed in Table 6.1. For each block in the block towers, we first assign it to one of the five materials, and then uniformly sample its density from possible values of its material. We generate 8 configurations for each block tower.

| Methods | Observations | | Density | | | | Trajectory |
| | Texture | Physics | Accuracy | | | RMSE | MAE |
			Top 1	Top 5	Top 10		
Frequent	−	−	2.0	9.7	13.4	25.4	74.4
Nearest	−	+	1.9	7.9	12.4	41.1	91.0
Oracle	+	−	6.9	35.7	72.0	18.5	51.3
PPD (no trajectory)	+	−	7.2	35.2	69.5	19.0	51.7
PPD (no image)	−	+	7.1	31.0	50.8	16.7	36.4
PPD (no voxels)	+	+	15.9	56.3	82.4	10.3	29.9
PPD (RGB-D)	+	+	11.6	50.5	79.5	12.8	30.2
PPD (full)	+	+	16.1	56.4	82.5	9.9	21.0
PPD (full)+Sample	+	+	**18.2**	**59.7**	**84.0**	**8.8**	**13.9**

Table 6.2: **Quantitative results of physical parameter estimation on block towers.** Combining appearance with physics does help our model to achieve better estimation on physical parameters, and our model performs significantly better than all other baselines.

Textures. We obtain the textures for materials by cropping the center portion of images from the MINC dataset [Bell et al., 2015]. We show sample images rendered with material textures in Figure 6-11. Since we render the textures only with respect to the material, the images rendered do not provide any information about density.

Physics interactions. We place the block towers at the origin and perform four physics interactions to obtain the object trajectories ($N_T = 4$). In detail, we exert a force with the magnitude of 10^5 on the block tower from four pre-defined positions $\{(\pm 1, -1, \pm 1)\}$. We simulate each physics interaction for 256 time steps using the Bullet Physics Engine [Coumans, 2010]. To ensure simulation accuracy, we set the time step for simulation to 1/300s.

Metrics. We evaluate the performance of shape reconstruction by the F_1 score between the prediction and ground truth: each primitive in prediction is labeled as a true positive if its intersection over union (IoU) with a ground-truth primitive is greater than 0.5. For physics estimation, we employ two types of metrics, i) density measures: top-k accuracy ($k \in \{1, 5, 10\}$) and root-mean-square error (RMSE) and ii) trajectory measure: mean-absolute error (MAE) between simulated trajectory (using predicted the physical parameters) and ground-truth trajectory.

Methods. We evaluate our model with different combinations of observations as input: i) texture only (i.e., no trajectory, by setting $f_T = 0$), ii) physics only (i.e., no image, by setting $f_I = 0$), iii) both texture and physics but without the voxelized shape, iv) both texture and physics but with replacing the 3D trajectory with a raw depth video, v) full data in our original setup (image, voxels, and trajectory). We also compare our model with several baselines: i) predicting the most frequent density

116

in the training set (*Frequent*), ii) nearest neighbor retrieval from the training set (*Nearest*), and iii) knowing the ground-truth material and guessing within its density value range (*Oracle*). While all these baselines assume perfect shape reconstruction, our model learns to decompose the shape.

Results. For the shape reconstruction, our model achieves 97.5 in terms of F1 score. For the physics estimation, we present quantitative results of our model with different observations as input in Table 6.2. We compare our model with an oracle that infers material properties from appearance while assuming ground-truth reconstruction. It gives *upper-bound* performance of methods that rely on only appearance cues. Experiments suggest that appearance alone is not sufficient for density estimation. From Table 6.2, we observe that combining appearance with physics performs well on physical parameter estimation, because the object trajectories can provide crucial additional information about the density distribution (i.e., moment of inertia). Also, all input modalities and sampling contribute to the model's final performance.

6.6 Object-Oriented Prediction and Planning

In this section, we explore how our visual de-animation framework can be deployed in robot planning and manipulation. We describe an extension of our model, named Object-Oriented Prediction and Planning (O2P2), for learning object-based representations suitable for planning in physical reasoning tasks.

As opposed to much prior work on object-factorized scene representations, including the de-animation models described in previous sections, we no longer supervise the content of the object representations directly by way of labeled attributes (such as position, velocity, or orientation). Instead, we assume access only to segments or region proposals for individual video frames. Since we do not have labels for the object representations, we must have a means for converting back and forth between images and object representations for training. O2P2 consists of three components, which are trained jointly:

- A **perception** module that maps from an image to an object encoding. The perception module is applied to each object segment independently.

- A **physics** module to predict the time evolution of a set of objects. We formulate the engine as a sum of binary object interactions plus a unary transition function.

- A **rendering** engine that produces an image prediction from a variable number of objects. We first predict an image and single-channel heatmap for each object. We then combine all of the object images according to the weights in their heatmaps at every pixel location to produce a single composite image.

117

a) No object factorization

b) Object property supervision

c) O2P2: Object factorization without object property supervision
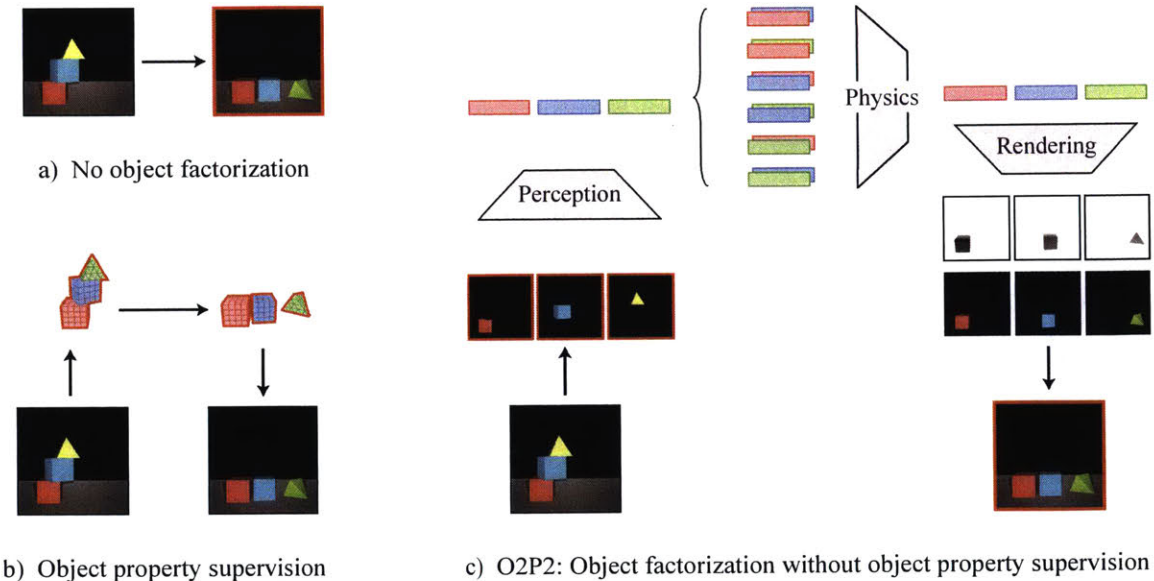
Figure 6-12: **We divide physical understanding tasks into three distinct paradigms. (a)** The first approach makes the fewest assumptions, posing prediction tasks as an instance of image-to-image translation. **(b)** The second uses ground-truth labels of object properties to supervise a learning algorithm that can map to the space of a traditional or learned physics engine. **(c)** O2P2, like (b), employs an object factorization and the functional structure of a physics engine, but like (a), does not assume access to supervision of object properties. Without object-level supervision, we must jointly learn a perception function to map from images to objects, a physics engine to simulate a collection of objects, and a rendering engine to map a set of objects back to a single composite image prediction. In all three approaches, we highlight the key supervision in orange.

6.6.1 Model

A high-level overview of the model is shown in Figure 6-12c. Below, we give details for the design of each component and their subsequent use in a model-based planning setting.

Perception module. The perception module is a four-layer convolutional encoder that maps an image observation to object representation vectors $\mathbf{O} = \{o_k\}_{k=1...N}$. We assume access to a segmentation of the input image $\mathbf{S} = \{s_k\}_{k=1...N}$ and apply the encoder individually to each segment. The perception module is not supervised directly to predict semantically meaningful properties such as position or orientation; instead, its outputs are used by the physics and rendering modules to make image predictions. In this way, the perception module must be trained jointly with the other modules.

Physics module. The physics module predicts the effects of simulating a collection of object representations $\mathbf{O}$ forward in time. As in Chang et al. [2017], Watters et al. [2017], we consider the interactions of all pairs of object vectors. The physics engine contains two learned subcomponents: a unary transition function f_{trans} applied to each

object representation independently, and a binary interaction function f_{interact} applied to all pairs of object representations. Letting $\bar{\mathbf{O}} = \{\bar{o}_k\}_{k=1...N}$ denote the output of the physics predictor, the kth object is given by $\bar{o}_k = f_{\text{trans}}(o_k) + \sum_{j \neq k} f_{\text{interact}}(o_k, o_j) + o_k$, where both f_{trans} and f_{interact} are instantiated as two-layer MLPs.

Much prior work has focused on learning to model physical interactions as an end goal. In contrast, we rely on physics predictions only insofar as they affect action planning. To that end, it is more important to know the resultant effects of an action than to make predictions at a fixed time interval. We therefore only need to make a single prediction, $\bar{\mathbf{O}} = f_{\text{physics}}(\mathbf{O})$, to estimate the steady-state configuration of objects as a result of simulating physics indefinitely. This simplification avoids the complications of long-horizon sequential prediction while retaining the information relevant to planning under physical laws and constraints.

Rendering engine. Because our only supervision occurs at the pixel level, to train our model we learn to map all object-vector predictions back to images. A challenge here lies in designing a function which constructs a single image from an entire collection of objects. The learned renderer consists of two networks, both instantiated as convolutional decoders. The first network predicts an image independently for each input object vector. Composing these images into a single reconstruction amounts to selecting which object is visible at every pixel location. In a traditional graphics engine, this would be accomplished by calculating a depth pass at each location and rendering the nearest object.

To incorporate this structure into our learned renderer, we use the second decoder network to produce a single-channel heatmap for each object. The composite scene image is a weighted average of all of the object-specific renderings, where the weights come from the negative of the predicted heatmaps. In effect, objects with lower heatmap predictions at a given pixel location will be more visible than objects with higher heatmap values. This encourages lower heatmap values for nearer objects. Although this structure is reminiscent of a depth pass in a traditional renderer, the comparison should not be taken literally; the model is only supervised by composite images and no true depth maps are provided during training.

Learning object representations. We train the perception, physics, and rendering modules jointly on an image reconstruction and prediction task. Our training data consists of image pairs (I_0, I_1) depicting a collection of objects on a platform before and after a new object has been dropped. (I_0 shows one object mid-air, as if being held in place before being released.) We assume access to a segmentation $\mathbf{S}_0$ for the initial image I_0.

Given the observed segmented image $\mathbf{S}_0$, we predict object representations using the perception module $\mathbf{O} = f_{\text{percept}}(\mathbf{S}_0)$ and their time-evolution using the physics

module $\bar{\mathbf{O}} = f_{\text{physics}}(\mathbf{O})$. The rendering engine then predicts an image from each of the object representations: $\hat{I}_0 = f_{\text{render}}(\mathbf{O}), \hat{I}_1 = f_{\text{render}}(\bar{\mathbf{O}})$.

We compare each image prediction $\hat{I}_t$ to its ground-truth counterpart using both $\mathcal{L}_2$ distance and a perceptual loss $\mathcal{L}_{\text{VGG}}$. As in Johnson et al. [2016a], we use $\mathcal{L}_2$ distance in the feature space of a pre-trained VGG network [Simonyan and Zisserman, 2015] as a perceptual loss function. The perception module is supervised by the reconstruction of I_0, the physics engine is supervised by the reconstruction of I_1, and the rendering engine is supervised by the reconstruction of both images. Specifically, $\mathcal{L}_{\text{percept}}(\cdot) = \mathcal{L}_2(\hat{I}_0, I_0) + \mathcal{L}_{\text{VGG}}(\hat{I}_0, I_0), \mathcal{L}_{\text{physics}}(\cdot) = \mathcal{L}_2(\hat{I}_1, I_1) + \mathcal{L}_{\text{VGG}}(\hat{I}_1, I_1)$, and $\mathcal{L}_{\text{render}}(\cdot) = \mathcal{L}_{\text{percept}}(\cdot) + \mathcal{L}_{\text{physics}}(\cdot)$.

Planning with learned models. We now describe the use of our perception, physics, and rendering modules, with the goal of building a block tower to match an observed image. Here, matching a tower does not refer simply to producing an image from the rendering engine that looks like the observation. Instead, we consider the scenario where the model must output a sequence of actions to construct the configuration.

This setting is much more challenging because there is an implicit sequential ordering to building such a tower. For example, the bottom cubes must be placed before the topmost triangle. O2P2 was trained solely on a pixel-prediction task, in which it was never shown such valid action orderings (or any actions at all). However, these orderings are essentially constraints on the physical stability of intermediate towers, and should be derivable from a model with sufficient understanding of physical interactions.

Although we train a rendering function as part of our model, we guide the planning procedure for constructing towers solely through errors in the learned object representation space. The planning procedure can be described at a high level in four components:

1. The perception module **encodes** the segmented goal image into a set of object representations $\mathbf{O}^{\text{goal}}$.

2. We **sample** actions of the form *(shape, position, orientation, color)*, where *shape* is categorical and describes the type of block, and the remainder of the action space is continuous and describes the block's appearance and where it should be dropped.

3. We **evaluate** the samples by likewise encoding them as object vectors and comparing them with $\mathbf{O}^{\text{goal}}$. We view action sample a_m as an image segment s_m (analogous to observing a block held in place before dropping it) and use the perception module to produce object vectors $\mathbf{O}^m$. Because the actions selected

should produce a stable tower, we run these object representations through the physics engine to yield $\bar{\mathbf{O}}^m$ before comparing with $\mathbf{O}^{goal}$. The cost is the $\mathcal{L}_2$ distance between the object $\bar{o} \in \bar{\mathbf{O}}^m$ corresponding to the most recent action and the goal object in $\mathbf{O}^{goal}$ that minimizes this distance.

4. Using the action sampler and evaluation metric, we **select** the sampled action that minimizes $\mathcal{L}_2$ distance. We then **execute** that action in MuJoCo [Todorov et al., 2012]. We continue this procedure, iteratively re-planning and executing actions, until there are as many actions in the executed sequence as there are objects in the goal image. In the simplest case, the distribution from which actions are sampled may be uniform. Alternatively, the cross-entropy method (CEM) [Rubinstein and Kroese, 2004] may be used, repeating the sampling loop multiple times and fitting a Gaussian distribution to the lowest-cost samples. In practice, we used CEM starting from a uniform distribution with five iterations, 1000 samples per iteration, and used the top 10% of samples to fit the subsequent iteration's sampling distribution.

6.6.2 Building Towers

After training O2P2 on the random configurations of blocks, we fixed its parameters and employed the planning procedure to build tower configurations observed in images. We also evaluated the following models as comparisons:

- **No physics** is an ablation of our model that does not run the learned physics engine, but instead simply sets $\bar{\mathbf{O}} = \mathbf{O}$.

- Stochastic adversarial video prediction (**SAVP**), a block-box video prediction model which does not employ an object factorization [Lee et al., 2018]. The cost function of samples is evaluated directly on pixels. The sampling-based planning routine is otherwise the same as in ours.

- **Oracle (pixels)** uses the MuJoCo simulator to evaluate samples instead of our learned physics and graphics engines. The cost of a block configuration is evaluated directly in pixel space using $\mathcal{L}_2$ distance.

- **Oracle (objects)** also uses MuJoCo, but has access to segmentation masks on input images while evaluating the cost of proposals. Constraining proposed actions to account for only a single object in the observation resolves some of the inherent difficulties of using pixel-wise loss functions.

Qualitative results of all models are shown in Figure 6-13 and a quantitative evaluation is shown in Table 6.3. We evaluated tower stacking success by greedily matching the built configuration to the ground-truth state of the goal tower, and

121
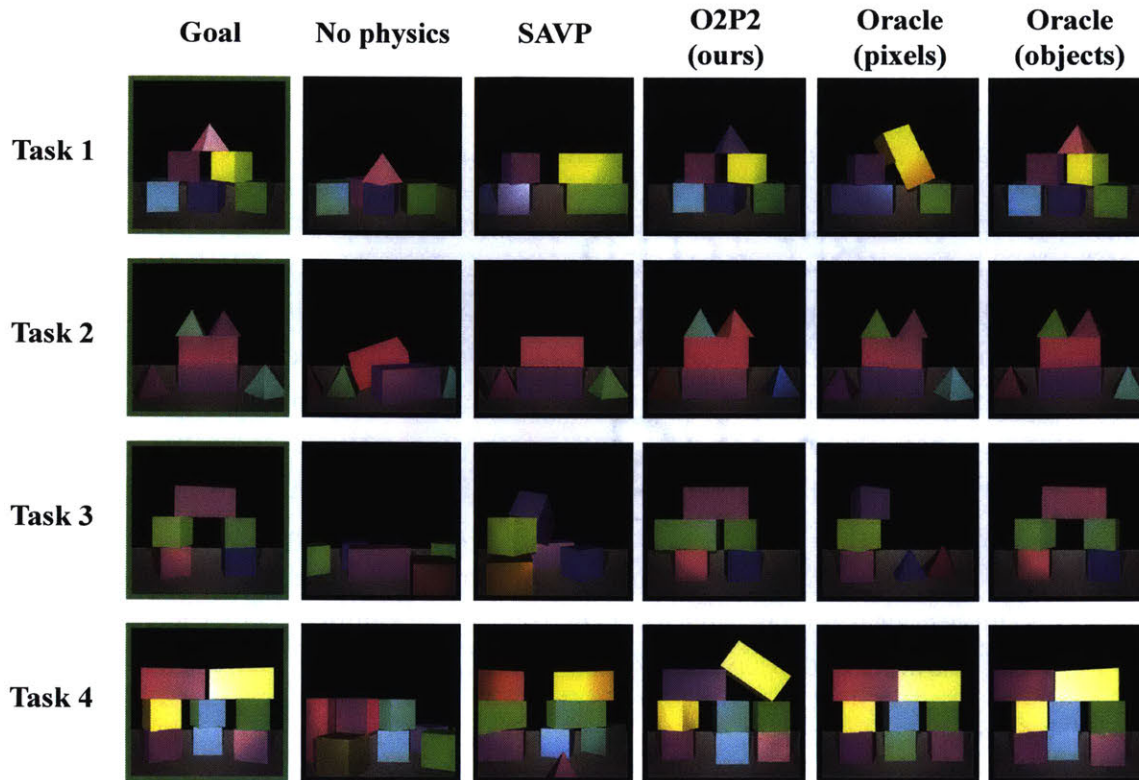
Figure 6-13: **Qualitative results on building towers using planning.** Given an image of the goal tower, we can use the learned object representations and predictive model in O2P2 for guiding a planner to place blocks in the world and recreate the configuration. We compare with an ablation, an object-agnostic video prediction model, and two 'oracles' with access to the ground-truth simulator.

comparing the maximum object error (defined on its position, identity, and color) to a predetermined threshold. Although the threshold is arbitrary in the sense that it can be chosen low enough such that all builds are incorrect, the relative ordering of the models is robust to changes in this value. All objects must be of the correct shape for a built tower to be considered correct, meaning that our third row prediction in Figure 6-13 was incorrect because a green cube was mistaken for a green rectangular cuboid.

While SAVP made accurate predictions on the training data, it did not generalize well to these more complicated configurations with more objects per frame. As such, its stacking success was low. Physics simulation was crucial to our model, as our No-physics ablation failed to stack any towers correctly. We explored the role of physics simulation in the stacking task in Section 6.6.2. The 'oracle' model with access to the ground-truth physics simulator was hampered when making comparisons in pixel space. A common failure mode of this model was to drop a single large block on the first step to cover the visual area of multiple smaller blocks in the goal image. This scenario was depicted by the blue rectangular cuboid in the first row of Figure 6-13 in

No physics	SAVP	Ours	Oracle (pixels)	Oracle (objects)
0	24	76	71	92

Table 6.3: **Accuracy (%) of block tower builds by our approach and the four comparison models.** Our model outperforms Oracle (pixels) despite not having the ground-truth simulator by virtue of a more appropriate object-factorized objective to guide the planning procedure.
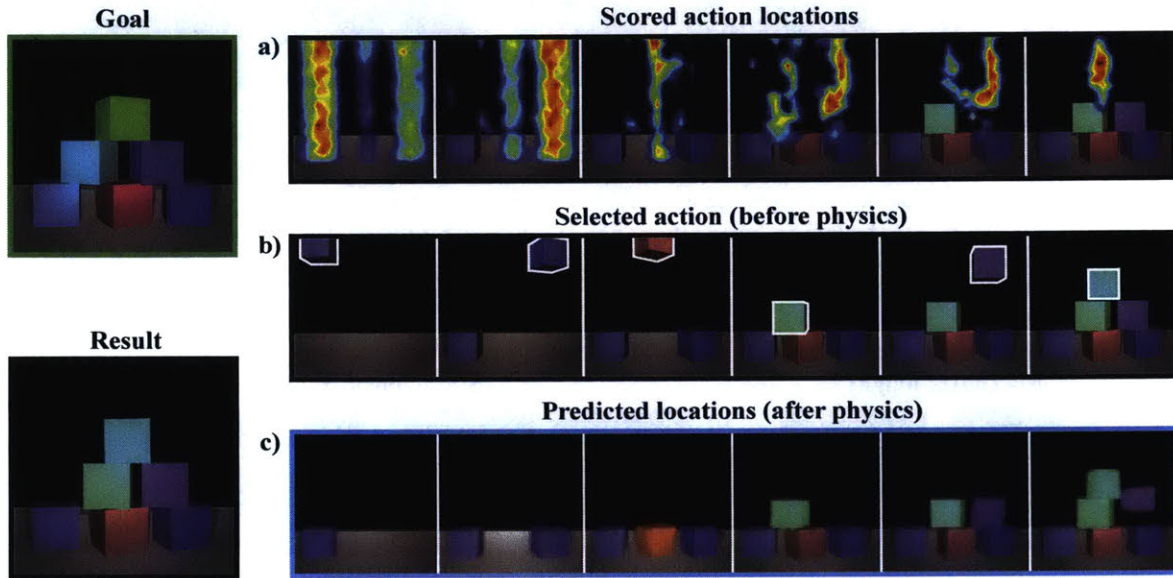
Figure 6-14: **Visualization of proposals and planning.** (a) Visualization of scored locations for dropping an object at each timestep. Because O2P2 simulates physics before selecting an action, it is able to plan a sequence of stable actions. (b) The selected block and drop position from the scored samples, outlined in white. (c) The prediction from our physics model of the result of running physics on the selected block.

the Oracle (pixels) column.

The importance of understanding physics. Figure 6-14 depicts the entire planning and execution procedure for O2P2 on a pyramid of six blocks. At each step, we visualize the process by which our model selects an action by showing a heatmap of scores (negative MSE) for each action sample according to the sample's (x, y) position (Figure 6-14a). Although the model is never trained to produce valid action decisions, the planning procedure selects a physically stable sequence of actions. For example, at the first timestep, the model scores three x-locations highly, corresponding to the three blocks at the bottom of the pyramid. It correctly determines that the height at which it releases a block at any of these locations does not particularly matter, since the block will drop to the correct height after running the physics engine. Figure 6-14b shows the selected action at each step, and Figure 6-14c shows the model's predictions about the configuration after releasing the sampled block.
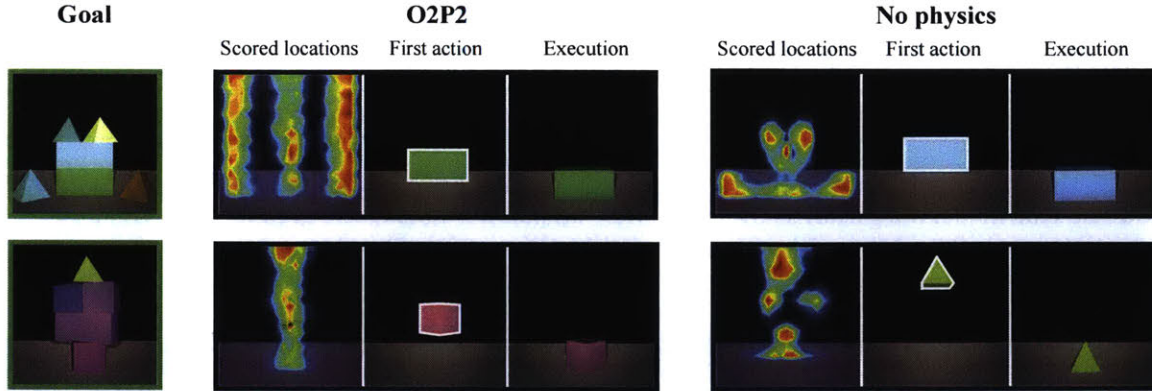
Figure 6-15: **Heatmaps showing sampled action scores for the initial action given a goal block tower.** O2P2's scores reflect that the objects resting directly on the platform must be dropped first, and that they may be dropped from any height because they will fall to the ground. The No-physics ablation, on the other hand, does not implicitly represent that the blocks need to be dropped in a stable sequence of actions because it does not predict the blocks moving after being released.
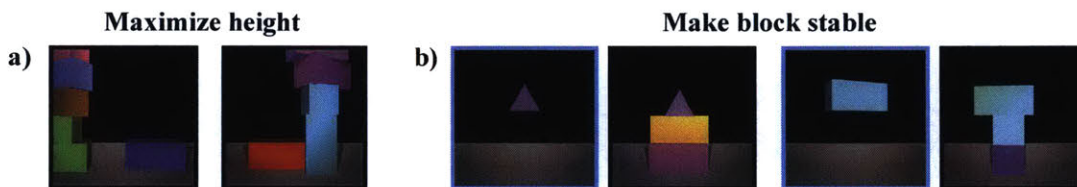
Figure 6-16: **O2P2 being used to plan for the alternate goals** of (a) maximizing the height of a tower and (b) making an observed block stable by use of any other blocks.

Similar heatmaps of scored samples are shown for the No-physics ablation of our model in Figure 6-15. Because this ablation does not simulate the effect of dropping a block, its highly-scored action samples correspond almost exactly to the actual locations of the objects in the goal image. Further, without physics simulation it does not implicitly select for stable action sequences; there is nothing to prevent the model from selecting the topmost block of the tower as the first action.

Planning for alternate goals. By implicitly learning the underlying physics of a domain, our model can be used for various tasks besides matching towers. In Figure 6-16a, we show our model's representations being used to plan a sequence of actions to maximize the height of a tower. There is no observation for this task, and the action scores are calculated based on the highest non-zero pixels after rendering samples with the learned renderer. In Figure 6-16b, we consider a similar sampling procedure as in the tower-matching experiments, except here only a single *unstable* block is shown. Matching a free-floating block requires planning with O2P2 for multiple steps at once.

6.7 Discussion

In this chapter, we have proposed to combine efficient, bottom-up, neural perception modules with rich, generalizable simulation engines for physical scene understanding. Our framework is flexible and can incorporate various graphics and physics engines. It performs well across multiple synthetic and real scenarios, reconstructing the scene and making future predictions accurately and efficiently.

We have also discussed how our model can be extended to handle multi-part objects with a non-uniform density distribution, and explored how our model can be adapted for robot planning and manipulation. We expect our framework to have wider applications in the future, due to the rapid development of scene description languages, 3D reconstruction methods, simulation engines, and virtual environments.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Learning Physics and Graphics Engines Themselves

In previous chapters, we have seen how the integration of deep learning and simulation engines enables efficient inference of object appearance and physics. In this chapter, we explore a novel use of learning systems: we build deep networks to approximate simulation engines themselves. These learned, data-augmented simulators come with features that are typically missing in their traditional counterparts, such as stochasticity and differentiability.

The problem we look into is synthesizing multiple plausible future frames from a single image. Such a multi-modal prediction problem is easy for humans, but traditional methods have mostly tackled this problem in a deterministic or non-parametric way. In this chapter, we propose to use learning to model future frames in a probabilistic manner. Our probabilistic model enables synthesizing many possible future frames from a single input image.

To synthesize realistic movement of objects, we propose a novel network structure, *Cross Convolutional Networks*, encoding image and motion information as feature maps and convolutional kernels, respectively. In experiments, our model performs well on synthetic data, such as 2D shapes and animated game sprites, and on real-world video frames. We present analyses of the learned network representations, showing it is implicitly learning a compact encoding of object appearance and motion. We also demonstrate a few of its applications, including visual analogy-making and video extrapolation. Finally, we present an extension of the model that not only discovers a compact, layered representation of objects, but the hierarchical structure among them.

This chapter includes materials previously published as Xue et al. [2019, 2016], Xu et al. [2019]. Tianfan Xue, Zhijian Liu, and Zhenjia Xu contributed significantly to the materials presented in this chapter.

7.1 Introduction

From just a single snapshot, humans are often able to imagine multiple possible ways that the scene can change over time. For instance, due to the pose of the girl in Figure 7-1, most would predict that her arms are stationary but her leg is moving. However, the exact motion is often unpredictable due to an intrinsic ambiguity. Is the girl's leg moving up or down? How large is the movement?

In this chapter, we aim to depict the conditional distribution of future frames given a single observed image. We name the problem *visual dynamics*, as it involves understanding how visual content relates to dynamic motions. We propose to tackle this problem using a probabilistic, content-aware motion prediction model that learns this distribution without using annotations. Sampling from this model allows us to visualize the many possible ways that an input image is likely to change over time.

The visual dynamics problem is in contrast to two traditional ways to model motion. The first is to assume object motion is deterministic, and to learn the direct mapping from an image's visual appearance to its motion [Mathieu et al., 2016, Walker et al., 2014]. These methods are more likely to produce accurate results when there is little ambiguity in object motion (e.g., when long-range videos are available). For single image future prediction, however, the results are unlikely to align with reality. The second way to model motion is to derive its prior distribution, which is invariant to image content [Fleet et al., 2000, Weiss and Adelson, 1998]. Understanding motion priors, like understanding image priors [Roth and Black, 2005], is a problem of fundamental scientific interest. However, as the motion we observe in real life is strongly correlated with visual content, methods ignoring visual signals do not work well on future frame prediction.

Modeling content-aware motion distributions is highly challenging mostly for two reasons. First, it involves modeling the correlation between motion and image content; to predict plausible future motion, the model must be able to relate visual content to semantic knowledge (e.g., object parts) in order to generate reasonable motions. Second, natural images lie on a high dimensional manifold that is difficult to describe precisely. Despite recent progresses on applying deep learning methods for image synthesis [Radford et al., 2016], building a generative model for real images is far from being solved.

We tackle the first problem using a novel convolutional neural network. During training, the network observes a set of consecutive image pairs from videos, and automatically infers the relationship between them without any supervision. During testing, the network then predicts the conditional distribution, $P(J|I)$, of future RGB images J (Figure 7-1b) given an RGB input image I that was not in the training set (Figure 7-1a). Using this distribution, the network is able to synthesize multiple

A Conditional Distribution of Future Frames

(a) An Input Image   (b) A Probabilistic Model   (c) Desired Output Samples
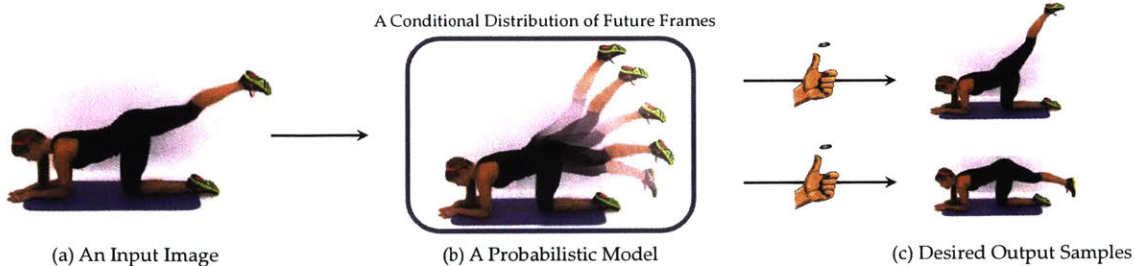
Figure 7-1: **Predicting the movement of an object from a single snapshot is often ambiguous.** For instance, is the girl's leg in (a) moving up or down? We propose a probabilistic, content-aware motion prediction model (b) that learns the conditional distribution of future frames, and produces a probable set of future frames (c). This schematic illustrates the idea behind our method, but does not show actual results produced by our model.

different image samples corresponding to possible future frames for the input image (Figure 7-1c).

We use a conditional variational autoencoder to model the complex conditional distribution of future frames [Kingma and Welling, 2014, Yan et al., 2016a]. This allows us to approximate a sample, J, from the distribution of future images by using a trainable function $J = f(I, z)$. The argument z is a sample from a simple (e.g., Gaussian) distribution, which introduces randomness into the sampling of J. This formulation makes the problem of learning the distribution more tractable than explicitly modeling the distribution.

To synthesize complex movement of objects, we proposed a novel layer-based synthesis network. The network splits an image into multiple segments and then uses a layered model to predict how each segment moves. This is a much easier task than modeling the motion of an entire image. Note that here we call each layer of an image as a segment to avoid confusion with convolutional layers. This layered prediction model synthesizes motion using a novel cross-convolutional layer. Unlike in standard convolutional layers, the values of the kernels are image-dependent, as different images may have different motions. Our model has several advantages. First, avoiding blurry outputs, the model does not directly synthesize the output image, but instead transforms pixels in the input frame based on sampled motion parameters. Second, the model only samples the movement of each layer, instead of all the pixels or a dense flow field. Because the motion of each layer lies on a lower-dimension manifold, its distribution is easier to model and the network can sample more diverse and realistic motions.

We test the proposed model on four datasets. Given an RGB input image, the algorithm can correctly model the distribution of possible future frames, and generate different samples that cover a variety of realistic motions. Our system significantly outperforms baselines in quantitative evaluation, and our results are in general preferred

by humans in user studies.

We present analyses to reveal the knowledge captured by our model: the cross convolutional network is able to discover semantically meaningful parts that have coherent motion patterns in an unsupervised fashion; and the latent representation z in the variational autoencoder is in essence a compact, interpretable encoding of object motion, as visualized in Section 7.6. Our model has wide applications: we demonstrate that it can be applied to visual analogy-making and video extrapolation straightforwardly, with good qualitative and quantitative performance. We also show that our model can be extended to discover not only object parts, but the hierarchical structure among them.

7.2 Related Work

Motion priors. Research studying the human visual system and motion priors provides evidence for low-level statistics of object motion. The pioneering work by Weiss and Adelson [1998] found that the human visual system prefers slow and smooth motion fields. Later, Lu and Yuille [2006] found that humans make similar motion predictions as a Bayesian ideal observer. Roth and Black [2005] analyzed the response of spatial filters applied to optical flow fields. Fleet et al. [2000] also found that a local motion field can be represented by a linear combination of a small number of bases. All these methods focused on the distribution of a motion field itself without considering any image information. On the contrary, our context-aware model captures the relationship between an observed image and its motion field.

These prior methods focused on modeling the distribution of an image's motion field using low-level statistics without any additional information. In real life, however, the distribution of motion fields is not independent of image content. For example, given an image of a car in front of a building, many would predict that the car is moving and the building is fixed. Thus, rather than modeling a motion prior as a context-free distribution, we propose to model the *conditional* motion distribution of future frames given an input image by incorporating a series of low- and high-level visual cues.

Motion or future prediction. Given an observed image or a short video sequence, models have been proposed to predict a future motion field [Liu et al., 2011, Pintea et al., 2014, Xue et al., 2014, Walker et al., 2015, 2016], future trajectories of objects [Walker et al., 2014, Wu et al., 2015a, 2017a, Zheng et al., 2018, Wu et al., 2016a], or a future visual representation Vondrick et al. [2016a]. However, unlike in our proposed model, most of these methods use a deterministic prediction model [Pintea et al., 2014, Vondrick et al., 2016a], which cannot model the uncertainty of the future.

Concurrently, Walker et al. [2016] identified the intrinsic ambiguity in deterministic

prediction, and has proposed a probabilistic prediction framework. Our model is also probabilistic, but it directly predicts the pixel values rather than motion fields or image features.

Parametric image synthesis. Early work in parametric image synthesis mostly focused on texture synthesis using hand-crafted features [Portilla and Simoncelli, 2000]. More recently, image synthesis algorithms have begun to produce impressive results by training variants of neural network structures to produce novel images [Gregor et al., 2015, Xie et al., 2017, 2016, Zhou et al., 2016b]. Generative adversarial networks [Goodfellow et al., 2014, Denton et al., 2015, Radford et al., 2016] and variational autoencoders [Kingma and Welling, 2014, Yan et al., 2016a] have been used to model and sample from natural image distributions. Our proposed algorithm is also based on the variational autoencoder, but unlike previous methods, we also model the temporal consistency between frames.

Video synthesis. Techniques that exploit the periodic structure of motion in videos have also been successful at generating novel frames from an input sequence. Schödl et al. [2000] proposed to shuffle frames from an existing video to generate a temporally consistent, looping image sequence. This idea was later used in video inpainting [Wexler et al., 2004], and was extended to generate cinemagraphs [Joshi et al., 2012] and seamlessly looping videos containing a variety of objects with different motion patterns [Agarwala et al., 2005, Liao et al., 2013]. While these techniques are able to generate high-resolution and realistic-looking videos, they are often applicable only to videos with periodic motions, and they require a reference video as input. In contrast, we build an image generation model that does not require a reference video during testing.

Recently, several neural network architectures have been proposed to synthesize a new frame from observed frames. They infer the future motion either from multiple previous frames [Srivastava et al., 2015, Mathieu et al., 2016], user-supplied action labels [Oh et al., 2015, Finn et al., 2016], or directly model the joint distribution of all frames without conditioning on the input [Vondrick et al., 2016b]. In contrast to these approaches, our network takes a single frame as input and learns the conditional distribution of future frames without any supervision.

7.3 Formulation

In this section, we first present a rigorous definition for the visual dynamics problem. Using a toy example, we then discuss three approaches to this problem, and show how the approach we take in our proposed model is more suitable for the task than the other two. We further present how our approach could be realized with a conditional variational autoencoder.

(a) Ground truth motion distribution

(b) Deterministic motion prediction

(c) Motion priors

(d) Probabilistic frame prediction
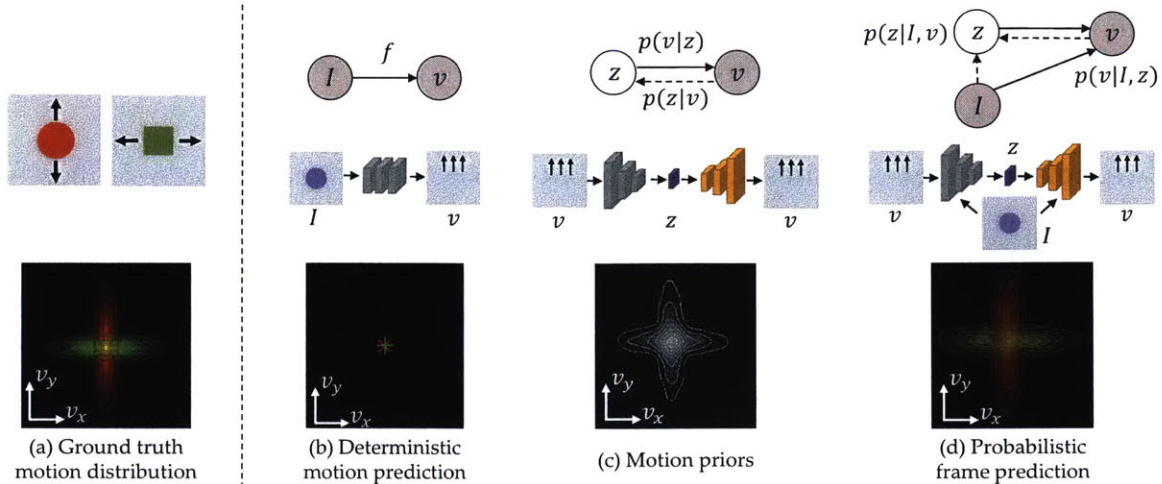
Figure 7-2: **An illustrative example on future prediction**. Imagine a world composed of circles that move mostly vertically and squares that move mostly horizontally (a). We consider three different models (b-d) to learn the mapping from an image to a motion field. The first row shows graphical models, the second row shows corresponding network structures, and the third row shows estimated motion distributions. The deterministic motion prediction structure shown in (b) attempts to learn a one-to-one mapping from appearance to motion, but is unable to model multiple possible motions of an object, and tends to predict a mean motion for each object (the third row of (b)). The content-agnostic motion prior structure shown in (c) is able to capture a low-dimensional representation of motion, but is unable to leverage cues from image appearance for motion prediction. Therefore, it can only recovers the joint distribution of all objects (third row of (c)). The content-aware probabilistic motion predictor (d) brings together the advantages of models of (b) and (c) and uses appearance cues along with motion modeling to predict a motion field from a single input image. Therefore, the estimated motion distribution is very close to the ground truth (compare the last row and (a) and (d)).

7.3.1 Problem Definition

In this section, we describe how to sample future frames from a current observation image. Here we focus on next frame synthesis; given an RGB image I observed at time t, our goal is to model the conditional distribution of possible frames observed at time $t + 1$.

Formally, let $\{(I^{(1)}, J^{(1)}), \ldots, (I^{(n)}, J^{(n)})\}$ be the set of image pairs in the training set, where $I^{(i)}$ and $J^{(i)}$ are images observed at two consecutive time steps. Using this data, our task is to model the distribution $p_\theta(J|I)$ of all possible next frames J for a new, previously unseen test image I, and then to sample new images from this distribution (θ is the set of model parameters). In practice, we choose not to directly predict the next frame, but instead to predict the difference image $v = J - I$ between the observed frame I and the future frame J (also known as the Eulerian motion). The task is then to learn the conditional distribution $p_\theta(v|I)$ from a set of training pairs $\{(I^{(1)}, v^{(1)}), \ldots, (I^{(n)}, v^{(n)})\}$.

132

7.3.2 An Illustrative Example

To understand how to design a model to best characterize the conditional distribution of object motion, consider a simple toy world that only consists of circles and squares. Circles mostly move vertically, while squares mostly move horizontally. As shown in Figure 7-2a, the ground truth distribution of a circle is $(v_x, v_y) \sim N((0,0), (0.2, 1))$ and the distribution of a square is $N((0,0), (1, 0.2))$, where $N(\vec{\mu}, \vec{\sigma})$ is a Gaussian distribution with mean equals to $\vec{\mu}$ and diagonal variation equal to $\vec{\sigma}$. Using this toy model, we discuss how each of the three models shown in Figure 7-2b-d is able to infer the underlying motion.

Approach I: Deterministic motion prediction. In this structure, the model tries to find a deterministic relationship between the input image I and object motion v (Figure 7-2b). In our toy world, $I \in \{\text{circle}, \text{square}\}$ is simply the binary label of each possible object and v is a 2D motion vector*.

In order to evaluate this model, we generate a toy training set which consists of 160,000 samples as follows. For each sample, we first randomly generate the object label I with equal probabilities of being circles or squares, and then sample the 2D motion vector of the object based on its label. The model is trained by minimizing the reconstruction error $\sum_i ||v^{(i)} - f(I^{(i)})||$ on this toy training set. The two other models we will soon introduce are also trained on this toy dataset.[†]

One drawback of this deterministic model is that it cannot capture the multiple possible motions that a shape can have. Essentially, the model can only learn the average motion of each object, I. The third row of Figure 7-2b shows the estimated motion of both circles (the red cross) and squares (the green cross). Since the both circles and squares have zero-mean, symmetric motion distributions, this method predicts a nearly static motion field for each input image.

Approach II: Motion priors. A simple way to model the multiple possible motions of future frames is to use a variational autoencoder [Kingma and Welling, 2014], as shown in Figure 7-2c. This model contains a latent representation, z that encodes the intrinsic dimensionality of the motion fields. The network that learns this intrinsic representation z consists of two parts: an encoder network f that maps the motion field v to an intrinsic representation z (the gray network in Figure 7-2c, which corresponds to $p(z|v)$), and a decoder network g that maps the intrinsic representation z to the motion field v (the yellow network, which corresponds to $p(v|z)$). During training, the network learns the latent representation z by minimizing the reconstruction error on the training set $\sum_i ||v^{(i)} - g(f(v^{(i)}))||$.

---

*Although in practice we choose v to be the RGB intensity difference between consecutive frames ($v = I - J$), for this toy example we define v as the 2D motion vector.

[†]The last row of Figure 7-2 shows actual predictions by our model trained on this dataset.

A shortcoming of this model is that the network does not see the input image when predicting the motion field. Therefore, it can only learn a joint distribution of both objects, as illustrated the third row of Figure 7-2c. Thus, during test time, the model is not be able to disambiguate between the specific motion distribution of circles and squares.

Approach III: Probabilistic frame prediction. We combine the deterministic motion prediction structure (approach I) with a motion prior (approach II), to model the uncertainty in a motion field and the correlation between motion and image content. We extend the decoder in (2) to take two inputs, the intrinsic motion representation z and an image I (see the yellow network in Figure 7-2d, which corresponds to $p(v|I, z)$). Therefore, instead of solely being able to model a joint distribution of motion v, it is now able to learn a conditional distribution of motion given the input image I.

In this toy example, since squares and circles move primarily in one (although different) direction, the intrinsic motion representation z only records the magnitude of motion along their major and minor directions. Combining the intrinsic motion representation with the direction of motion inferred from the image content, the model can correctly model the distribution of motion. Figure 7-2d shows that the inferred motion distribution of each object is quite similar to the ground truth distribution.

7.3.3 Conditional Variational Autoencoder

In this section, we will formally derive the training objective of our model, following the similar derivations [Kingma and Welling, 2014, Kingma et al., 2014, Yan et al., 2016a]. Consider the following generative process that samples a future frame conditioned on an observed image, I. First, the algorithm samples the hidden variable z from a prior distribution $p_z(z)$; we assume $p_z(z)$ is a multivariate Gaussian distribution where each dimension is i.i.d. with zero-mean and unit-variance. Then, given a value of z, the algorithm samples the intensity difference image v from the conditional distribution $p_\theta(v|I, z)$. The final image, $J = I + v$, is then returned as output.

A variational upper-bound. In the training stage, the algorithm attempts to maximize the log-likelihood of the conditional marginal distribution $\sum_i \log p(v^{(i)}|I^{(i)})$. Assuming I and z are independent, the marginal distribution is expanded as

$$\sum_i \log \int_z p(v^{(i)}|I^{(i)}, z)p_z(z)dz. \tag{7.1}$$

Directly maximizing this marginal distribution is hard, thus we instead maximize its variational upper bound [Kingma and Welling, 2014]. Each term in the marginal

distribution is upper-bounded by

$$\mathcal{L}(\theta, \phi, v^{(i)}|I^{(i)}) \approx - D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z))$$

$$+ \frac{1}{L} \sum_{l=1}^{L} \left[\log p_\theta(v^{(i)}|z^{(i,l)}, I^{(i)})\right], \tag{7.2}$$

where D_{KL} is the KL-divergence, $q_\phi(z|v^{(i)}, I^{(i)})$ is the variational distribution that approximates the posterior $p(z|v^{(i)}, I^{(i)})$, and $z^{(i,l)}$ are samples from the variational distribution. Recall that z and $I^{(i)}$ are independent, so that $p_z(z|I^{(i)}) = p_z(z)$. Please see the end of the section for detailed derivation of Equation 7.2. For simplicity, we refer to the conditional data distribution, $p_\theta(\cdot)$, as the *generative model*, and the variational distribution, $q_\phi(\cdot)$, as the *recognition model*.

In practice, we always choose $L = 1$. Therefore, the upper bound of the KL-divergence can be simplified as

$$-D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z)) + \log p_\theta(v^{(i)}|z^i, I^{(i)}). \tag{7.3}$$

If the assumption that I and z are independent does not hold, we can convert a latent variable that depends on I to one that does not, without affecting the expressiveness of our generative model, as suggested by Kingma et al. [2014], Sohn et al. [2015].

Distribution reparametrization. We assume Gaussian distributions for both the generative model and recognition model[‡], where the mean and variance of the distributions are functions specified by neural networks, that is[§]

$$p_\theta(v^{(i)}|z^{(i,l)}, I^{(i)}) = \mathcal{N}(v^{(i)}; f_{\mathrm{mean}}(z^{(i,l)}, I^{(i)}), \sigma^2 \mathbf{I}), \tag{7.4}$$

$$q_\phi(z^{(i,l)}|v^{(i)}, I^{(i)}) = \mathcal{N}(z^{(i,l)}; g_{\mathrm{mean}}(v^{(i)}, I^{(i)}), g_{\mathrm{var}}(v^{(i)}, I^{(i)})), \tag{7.5}$$

where $\mathcal{N}(\cdot; a, b)$ is a Gaussian distribution with mean a and variance b. f_{mean} is a function that predicts the mean of the generative model, defined by the generative network (the yellow network in Figure 7-2d). g_{mean} and g_{var} are functions that predict the mean and variance of the recognition model, respectively, defined by the recognition network (the gray network in Figure 7-2d). Here we assume that all dimensions of the generative model have the same variance σ^2, where σ is a hand-tuned hyperparameter.

[‡]A complex distribution can be approximated as a function of a simple distribution, such as a Gaussian. This is referred to as the reparameterization trick [Kingma and Welling, 2014].

[§]Here the bold $\mathbf{I}$ denotes an identity matrix, whereas the normal-font I denotes the observed image.

The objective function. Plugging Equation 7.4 and Equation 7.5 in to Equation 7.2, for $L = 1$ (only use one sample for each training iteration) we obtain the objective function that we minimize for each sample:

$$D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})\|p_z(z)) + \lambda\|v^{(i)} - f_{\mathrm{mean}}(z^{(i)}, I^{(i)})\|, \qquad (7.6)$$

where $z^{(i)}$ is sampled from the distribution defined by Equation 7.5, and λ is a constant. During training, we use stochastic gradient descent to minimize the variational lower bound defined in Equation 7.6.

Verification of Equation 7.2 and Equation 7.6 We now formally derive how we obtain the training objective function in Equation 7.2, following similar derivations in [Kingma and Welling, 2014, Kingma et al., 2014, Yan et al., 2016a]. As mentioned in Section 7.3.3, the generative process that samples a difference image v from a θ-parametrized model, conditioned on an observed image I, consists of two steps. First, the algorithm samples the hidden variable z from a prior distribution $p_z(z)$. Then, given a value of z, the algorithm samples the intensity difference image v from the conditional distribution $p_\theta(v|I, z)$. This process is also described in the graphical model in Figure 7-2d.

Given a set of training pairs $\{I^{(i)}, v^{(i)}\}$, the algorithm maximizes the log-likelihood of the conditional marginal distribution during training

$$\sum_i \log p(v^{(i)}|I^{(i)}). \qquad (7.7)$$

Recall that I and z are independent as shown in the graphical model in Figure 7-2. Therefore, based on the Bayes' theorem, we have

$$p(v^{(i)}|I^{(i)}) = \frac{p_z(z)p_\theta(v^{(i)}|I^{(i)}, z)}{p(z|v^{(i)}, I^{(i)})}. \qquad (7.8)$$

It is hard to directly maximize the marginal distribution in Equation 7.7. We therefore maximize its variational upper-bound instead, as proposed by Kingma and Welling [Kingma and Welling, 2014]. Let $q_\phi(z|v^{(i)}, I^{(i)})$ be the variational distribution that approximates the posterior $p(z|v^{(i)}, I^{(i)})$. Then each term in the marginal distribution is upper bounded as

$$\log p(v^{(i)}|I^{(i)})$$
$$= \mathbb{E}_{q_\phi}\left[\log p(v^{(i)}|I^{(i)})\right]$$
$$= \mathbb{E}_{q_\phi}\left[\log \frac{p_z(z)p_\theta(v^{(i)}|I^{(i)}, z)}{p(z|v^{(i)}, I^{(i)})}\right]$$

$$= \mathbb{E}_{q_\phi} \left[\log \frac{p_z(z)}{q_\phi(z|v^{(i)}, I^{(i)})} \right] + \mathbb{E}_{q_\phi} \left[\log \frac{q_\phi(z|v^{(i)}, I^{(i)})}{p(z|v^{(i)}, I^{(i)})} \right] + \mathbb{E}_{q_\phi}[\log p_\theta(v^{(i)}|I^{(i)}, z)]$$

$$= - D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z)) + D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p(z|v^{(i)}, I^{(i)}))$$

$$+ \mathbb{E}_{q_\phi}[\log p_\theta(v^{(i)}|I^{(i)}, z)]$$

$$\geq - D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z)) + \mathbb{E}_{q_\phi}[\log p_\theta(v^{(i)}|I^{(i)}, z)]$$

$$\triangleq \mathcal{L}(\theta, \phi, v^{(i)}|I^{(i)}). \tag{7.9}$$

The first KL-divergence term in Equation 7.9 has an analytic form [Kingma and Welling, 2014]. To make the second term tractable, we approximate the variational distribution, $q_\phi(z|x^{(i)}, I^{(i)})$, by its empirical distribution. We have

$$\mathcal{L}(\theta, \phi, v^{(i)}|I^{(i)})$$

$$\approx - D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z)) + \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(v^{(i)}|z^{(i,l)}, I^{(i)})$$

$$= - D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z)) - \frac{1}{2\sigma^2 L} \sum_{l=1}^{L} \|v^{(i)} - f_{\mathrm{mean}}(z^{(i)}, I^{(i)})\| + C, \tag{7.10}$$

where $z^{(i,l)}$ are samples from the variational distribution $q_\phi(z|v^{(i)}, I^{(i)})$ and C is a constant. Equation 7.10 is the variation lower bound that our network minimizes during training.

In practice, we simply generate one sample of $z^{(i,l)}$ at each iteration (thus $L = 1$) of stochastic gradient descent, and different samples are used for different iterations. By defining $\lambda = 1/(2\sigma^2)$ and taking the negative of the right-hand side of Equation 7.10, we get the objective function to minimize in training (Equation 7.6):

$$D_{\mathrm{KL}}(q_\phi(z|v^{(i)}, I^{(i)})||p_z(z)) + \lambda \|v^{(i)} - f_{\mathrm{mean}}(z^{(i)}, I^{(i)})\|. \tag{7.11}$$

We will describe in Section 7.4 the neural networks that define the generative function f_{mean} and recognition function g_{mean} and g_{var}.

7.4 Learning Visual Dynamics

We present an end-to-end trainable neural network structure, defining the generative function f_{mean} and recognition functions g_{mean}, and g_{var}. Once trained, these functions can be used in conjunction with an input image to sample future frames. We first describe our newly proposed cross convolutional layer, which naturally characterizes a layered motion representation [Wang and Adelson, 1993]. We then explain our network structure and demonstrate how we integrate the cross convolutional layer into the network for future frame synthesis.
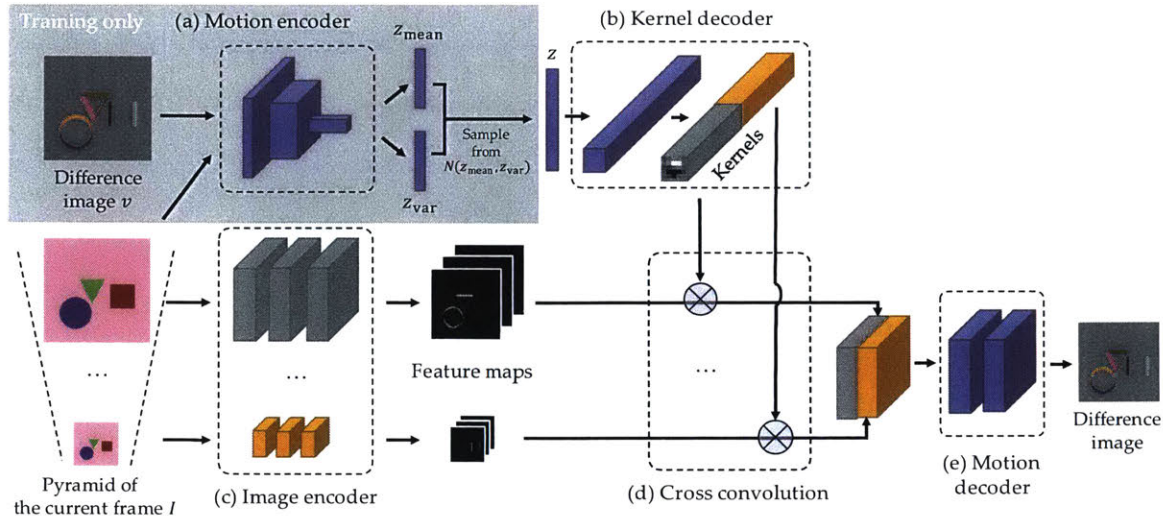
Figure 7-3: **Architecture of our future prediction network.** The network consists of five components: (a) a motion encoder, (b) a kernel decoder, (c) an image encoder, (d) a cross convolution layer, and (e) a motion decoder. Our image encoder takes images at four scales as input. For simplicity, we only show two scales in this figure. See Section 7.4 for details of our model. The motion encoder (the grayed region) is only used in training. At testing time, the motion vector z is sampled from its empirical distribution.

7.4.1 Layered Motion Representations and Cross Convolutional Nets

Motion can often be decomposed in a layer-wise manner [Wang and Adelson, 1993]. Intuitively, different semantic segments in an image should have different distributions over all possible motions; for example, a building is often static, but a car moves.

To model layered motion, we propose a novel cross convolutional network (Figure 7-3). The network first decomposes an input image pyramid into multiple feature maps (segments) through an image encoder (Figure 7-3c). It then convolves these maps with different kernels (Figure 7-3d), and uses the outputs to synthesize a difference image (Figure 7-3e). This network structure naturally fits a layered motion representation, as each feature map characterizes an image *segment* and the corresponding kernel characterizes the motion of that segment. In other words, we model motions as convolutional kernels, which are applied to feature maps of images at multiple scales.

Unlike a traditional convolutional network, these kernels should not be identical for all inputs, as different images should be associated with different motions (kernels). We therefore propose a cross convolutional layer to tackle this problem. The cross convolutional layer does not learn the weights of the kernels itself. Instead, it takes both kernel weights and image segments as input and performs convolution during a forward pass; for back propagation, it computes the gradients with respect to both convolutional kernels and image segments.

The characteristics of a cross convolutional layer naturally fit the layered motion

138

representation, as we can think of each feature map as an image segment, and the corresponding kernel characterizes the layer's motion. In other words, we model motions as convolutional kernels, which are applied to image segments (layers) at multiple scales. Concurrent papers [Finn et al., 2016, Brabandere et al., 2016] have also explored similar ideas. While they applied the learned kernels on input images, we jointly learn image segments and kernels without direct supervision.

7.4.2 Network Structure

As shown in Figure 7-3, our network consists of five components: (a) a motion encoder, which is a variational autoencoder that learns the compact representation, z, of possible motions; (b) a kernel decoder, which learns the motion kernels from the compact motion representation z; (c) an image encoder, which consists of convolutional layers extracting segments from the input image I; (d) a cross convolutional layer, which takes the output of the image encoder and the kernel decoder, and convolves the image segments with motion kernels; and (e) a motion decoder, which regresses the difference image from the combined feature maps. We now introduce each part in detail.

During training, our motion encoder (Figure 7-3a) takes the current frame and a difference image as input, both at resolution 128×128. The network then applies six 5×5 convolutional and batch normalization layers (number of channels are {96, 96, 128, 128, 256, 256}) to the concatenated images, with some pooling layers in between. The output has a size of 256×5×5. The kernel encoder then reshapes the output to a vector, and splits it into a 3,200-dimension mean vectors z_{mean} and a 3,200-dimension variance vector z_{var}, from which the network samples the latent motion representation $z \sim N(z_{mean}, z_{var})$. The motion encoder takes the current frame as input, in addition to the motion image, so that it can learn to model the *conditional* variational distribution ($q_\theta(\cdot)$ in Equation 7.6).

Next, the kernel decoder (Figure 7-3b) sends the $3{,}200 = 128{\times}5{\times}5$ tensor into two additional convolutional layers, each with 128 channels and a kernel size of 5. They are then split into four sets, each with 32 kernels of size 5×5.

Our image encoder (Figure 7-3c) operates on four different scaled versions of the input image I (256×256, 128×128, 64×64, and 32×32)[1]. At each scale, there are four sets of 5×5 convolutional and batch normalization layers (number of channels are {64, 64, 64, 32}), two of which are followed by a 2×2 max pooling layer. Therefore, the output size of the four channels are 32×64×64, 32×32×32, 32×16×16, and 32×8×8, respectively. This multi-scale convolutional network allows us to model both global and local structures in the image, which may have different motions.

---

[1]For the input image of size 128×128, we used five different scales instead. In that case, the size of motion vector is 4,000 ($= 5 \times 5 \times 32 \times 5$).

139

The core of our network is a cross convolutional layer (Figure 7-3d), which, as discussed in Section 7.4.1, applies the kernels learned by the kernel decoder to the feature maps (layers) learned by the image encoder. The cross convolutional layer has the same output size as the image encoder.

Our motion decoder (Figure 7-3e) starts with an up-sampling layer at each scale, making the output of all scales of the cross convolutional layer have a resolution of 64×64. This is then followed by one 9×9 and two 1×1 convolutional and batch normalization layers, with {128, 128, 3} channels. These final feature maps (layers) are then used to regress the output difference image.

Training and testing details. During training, the image encoder takes a single frame $I^{(i)}$ as input, and the motion encoder takes both input frame $I^{(i)}$ and the difference image $v^{(i)} = J^{(i)} - I^{(i)}$ as input, where $J^{(i)}$ is the next frame. The network aims to regress the difference image that minimizes the objective function Equation 7.6.

During testing, the image encoder still sees a single image I; however, instead of using a motion encoder, we directly sample motion vectors $z^{(j)}$ from the prior distribution $p_z(z)$ (therefore, the gray part in Figure 7-3 is not used in testing). In practice, we use an empirical distribution of z over all training samples as an approximation to the prior, a.k.a. the variational distribution $q_\phi(z)$ in the literature, as Doersch [2016] showed that this sampling strategy works better than sampling from the prior distribution $p_z(z)$. Our sampling of z is independent of the input image I, satisfying the independence assumption discussed in Section 7.3.3. The network then synthesizes possible difference images $v^{(j)}$ by taking the sampled latent representation $z^{(j)}$ and an RGB image I as input. We then generate a set of future frames $\{J^{(j)}\}$ from these difference images: $J^{(j)} = I + v^{(j)}$.

7.5   Evaluations

We now present a series of experiments to evaluate our method. We start with a dataset of 2D shapes, which serves to benchmark our model on objects with simple, yet nontrivial, motion distributions. Following Reed et al. [2015], we then test our method on a dataset of video game sprites with diverse motions. In addition to these synthetic datasets, we further evaluate our framework on real-world video datasets. Again, note that our model uses consecutive frames for training, requiring no supervision. Visualizations of our experimental results are also available on the project page.

7.5.1   Movement of 2D Shapes

We first evaluate our method using a dataset of synthetic 2D shapes. The dataset contains three types of objects: circles, squares, and triangles. Circles always move vertically, squares horizontally, and triangles diagonally. The motion of circles and

| (a) Input | (b) Next frame | (c) Reconstruction | (d) 3 sampled future frames | (e) Zoomed-in views |
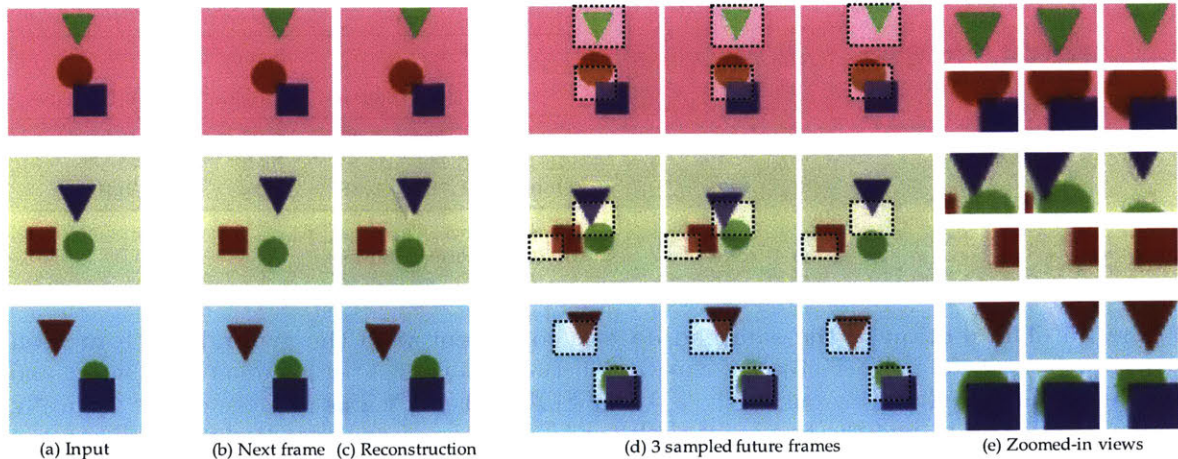
Figure 7-4: **Results on the Shapes dataset containing circles, squares, and triangles**. For a given frame (a) our goal is to predict probable motion. In (b) we show the ground truth future frame. Notice how squares move horizontally, circles vertically, triangles diagonally, and the triangle's motion is correlated with the circle's. Our model is able to reconstruct the motion (c) after encoding and decoding with the ground truth image pairs. By sampling from the latent representation, we can also synthesize additional novel future frames with probable motion (d). In (e), we show zoomed-in regions for these samples. Note the significant variation among the sampled frames.

squares are independent, while the motion of circles and triangles are correlated (when the triangle moves up, the circle moves down). The shapes can be heavily occluded, and their sizes, positions, and colors are chosen randomly. There are 20,000 pairs for training, and 500 for testing.

Figure 7-4 shows the results. Figure 7-4a and Figure 7-4b show a sample of consecutive frames in the dataset, and Figure 7-4c shows the reconstruction of the second frame after encoding and decoding with the ground truth image pairs. Figure 7-4d and Figure 7-4e show samples of the second frame; in these results the network only takes the first image as input, and the compact motion representation, z, is randomly sampled. Note that the network is able to capture the distinctive motion pattern for each shape, including the strong correlation of triangle and circle motion.

To quantitatively evaluate our algorithm, we compare the displacement distributions of circles, squares, and triangles in the sampled images with their ground truth distributions. We sample 50,000 images and use the optical flow package by Liu [2009] to calculate the mean movement of each object. We plot them in Figure 7-5 as well as the isolines using the *contour* function in MATLAB.

We also compute their KL-divergence. Here, we divide the region $[-5, 5] \times [-5, 5]$ into 41×41 bins and approximate the predicted distribution with the 2D histogram. We compare our algorithm with a simple baseline that copies the optical flow field of the closest image pairs from the training set ('Flow' in Figure 7-5); for each test image, we find its 10-nearest neighbors in the training set (the retrieval is based on ℓ^2-distance
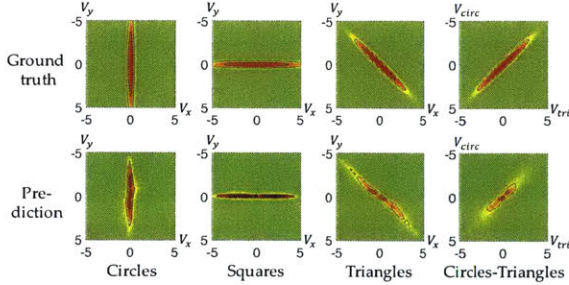
Method	Shapes			
	Circles	Squares	Triangles	Circles-Triangles
Flow	6.77	7.07	6.07	8.42
AE	8.76	12.37	10.36	10.58
Ours	**1.70**	**2.48**	**1.14**	**2.46**

KL divergence, $D_{\mathrm{KL}}(p_{\mathrm{gt}} \| p_{\mathrm{pred}})$, between predicted and ground truth distributions

Figure 7-5: **The motion of the sampled data is consistent with the ground truth motion distributions.** Left: for each object, comparison between its ground-truth motion distribution and the distribution predicted by our method. It shows the network learns to move circles vertically, squares, horizontally, and the motion of circles and triangles is correlated. Right: KL divergence between ground-truth distributions and distributions predicted by three different algorithms. Our network scores much better than a simple nearest-neighbor motion transfer algorithm.

Method	Resolution	
	32×32	64×64
Flow	29.7	21.0
Ours	**41.2**	**35.7**

Percentages (%) of results labeled as *real* by human subjects

(a) Input   (b) Next frame   (c) Reconstruction   (d) 2 sampled future frames   (e) Zoomed-in views
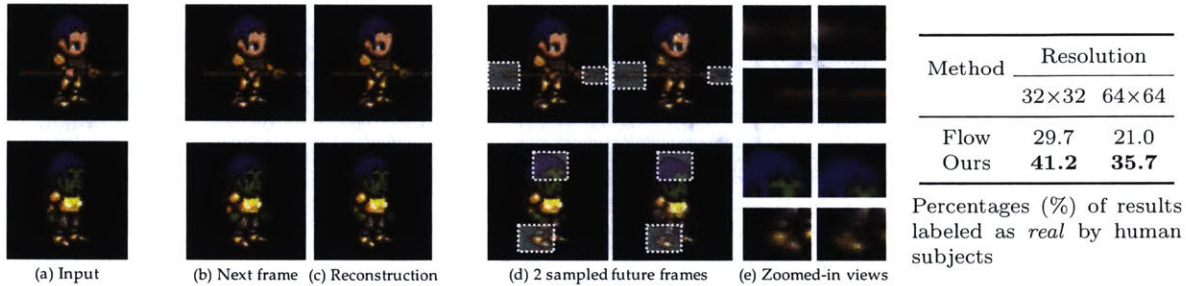
Figure 7-6: **Results on the Sprites dataset.** Left: We show input images (a), ground truth next frames (b), our reconstruction (c), two sampled future frames (d), and corresponding zoomed-in views (e). Right: Percentages (%) of synthesized results that were labeled as real by human subjects in two-way forced choices on Amazon Mechanical Turk, at resolution 32×32 and 64×64. A perfect algorithm would achieve a percentage around 50%.

between query image and images in the training dataset), and randomly transfer one of the corresponding optical flow fields. To illustrate the advantage of using a variational autoencoder (VAE) over a standard autoencoder, we also modify our network by removing the KL-divergence loss and sampling layer ('AE' in Figure 7-5). Figure 7-5 shows our predicted distribution is very close to the ground-truth distribution. It also shows that a VAE helps to capture the true distribution of future frames.

7.5.2   Movement of Video Game Sprites

We evaluate our framework on a video game sprites dataset[||], also used by Reed et al. [2015]. The dataset consists of 672 unique characters; for each character, there are 5 animations (spellcast, thrust, walk, slash, shoot) from 4 different viewpoints. The length of each animation ranges from 6 to 13 frames. We collect 102,364 pairs of neighboring frames for training, and 3,140 pairs for testing. The same character does

---

[||]Liberated pixel cup: http://lpc.opengameart.org

not appear in both the training and test sets. Sampled future frames are shown in Figure 7-6. From a single frame, our method captures various possible motions that are consistent with those in the training set.

As a quantitative evaluation on the success rate of our image synthesis algorithm, we conduct behavioral experiments on Amazon Mechanical Turk. We randomly select 200 images, sample a possible next frame using our algorithm, and show them to multiple human subjects as an animation side by side with the ground truth animation. We then ask the subject to choose which animation is real (not synthesized). An ideal algorithm should achieve a success rate of 50%. In our experiments, we present the animation in both the original resolution ($64{\times}64$) and a lower resolution ($32{\times}32$). We only evaluate on subjects that have a past approval rate of $>95\%$ and have also passed our qualification tests. Figure 7-6 shows that our algorithm significantly outperforms a baseline that warps the input by transferring one of closest flow fields from the training set. Our results are labeled as *real* by humans 41.2% of the time at $32{\times}32$, and 35.7% of the time at $64{\times}64$. Subjects are less easily fooled by $64{\times}64$ images, as it is harder to hallucinate realistic details in high-resolution images.

7.5.3 Movement in Real Videos Captured in the Wild

To demonstrate that our algorithm can also handle real videos, we collect 20 workout videos from YouTube, each about 30 to 60 minutes long. We first apply motion stabilization to the training data as a pre-processing step to remove camera motion. We then extract 56,838 pairs of frames for training and 6,243 pairs for testing. The training and testing pairs come from different video sequences.

Figure 7-7 shows that our framework works well in predicting the movement of the legs and torso. Specifically, the algorithm is able to synthesize reasonable motions of the human in various poses. The Mechanical Turk behavioral experiments also show that the synthesized frames are visually realistic. In particular, our synthesized images are labeled as real by humans 36.7% of the time at a resolution of $32{\times}32$, and 31.3% of the time at $64{\times}64$.

Given an input frame, how often can our algorithm sample realistic motion? We have run an additional human study to evaluate this: for each of 100 randomly selected test images, we sample 100 future frames at $128{\times}128$, and ask AMT subjects whether the two consecutive frames, one real and one synthesized, characterize realistic human motion. The success rate is 44.3% (averaged over the 100 test images), with a standard deviation of 4.3%.

Synthesizing images with realistic human motion is challenging. Traditional methods often require a high quality 3D model of the human body for real-time synthesis and rendering [Thies et al., 2016]. Although our algorithm does not require a 3D model, it can still simulate how a human moves between frames. Note that

143

Method	Resolution	
	32×32	64×64
Flow	31.3	25.5
Ours	**36.7**	**31.3**

Percentages (%) of results labeled as *real* by human subjects

| (a) Input | (b) Next frame | (c) Recon-struction | (d) 3 sampled future frames | (e) Zoomed-in views |
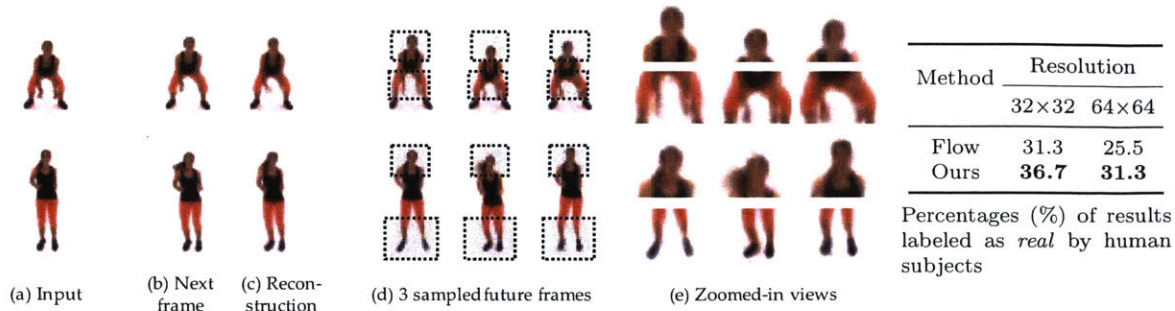
Figure 7-7: **Results on the Exercise dataset.** Left: We show input images (a), ground truth next frames (b), our reconstruction (c), three sampled future frames (d), and corresponding zoomed-in views (e). Right: Percentages (%) of synthesized results that were labeled as real by human subjects in two-way forced choices on Amazon Mechanical Turk, at resolution 32×32 and 64×64. A perfect algorithm would achieve a percentage around 50%.

| (a) Input | (b) Next frame | (c) Reconstruction | (d) 2 sampled future frames | (e) Zoomed-in views |
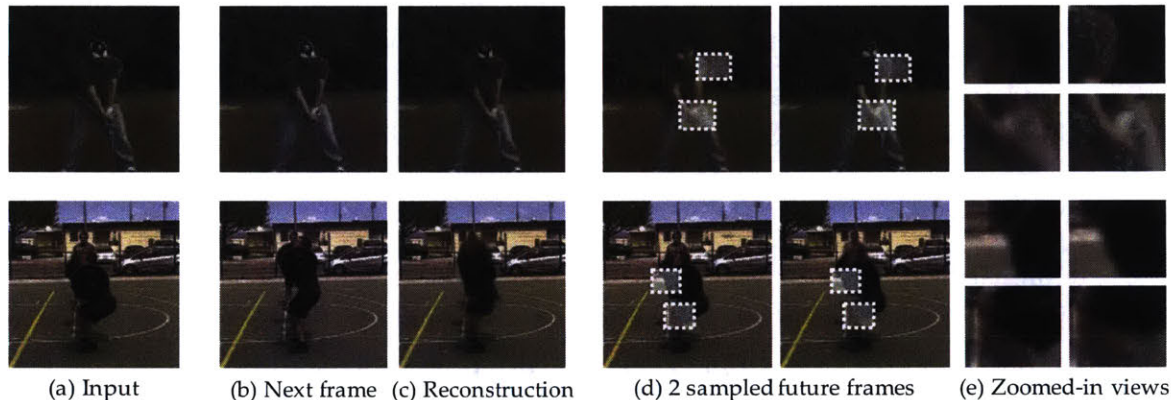
Figure 7-8: **Sampling results on the PennAction dataset [Zhang et al., 2013]**, where we show input images (a), ground truth next frames (b), our reconstruction (c), three sampled future frames (d), and corresponding zoomed-in views (e).

synthesized large motions in our results often happen around locations corresponding to skeletal joints, implying that our model has an implicit understanding of the structure and correlation between body parts.

At last, we test our algorithm on realistic videos in the wild. We use the PennAction dataset [Zhang et al., 2013], which contains 981 sequences of diverse human motion with complex backgrounds. We extract 7,705 pairs of frames for training and 987 pairs for evaluation. To remove trivial panning motions, we detect the human in each frame [Huang and Ramanan, 2017] and crop each frame to center the subject.

The results are shown in Figure 7-8. While in-the-wild videos are more challenging, our model synthesizes multiple realistic future frames from a single image. Especially, when synthesizing future frames, our model keeps the original background intact, suggesting the learned motion representation separates the region of interest from the background. In the future, we aim to develop prediction models that are able to

144
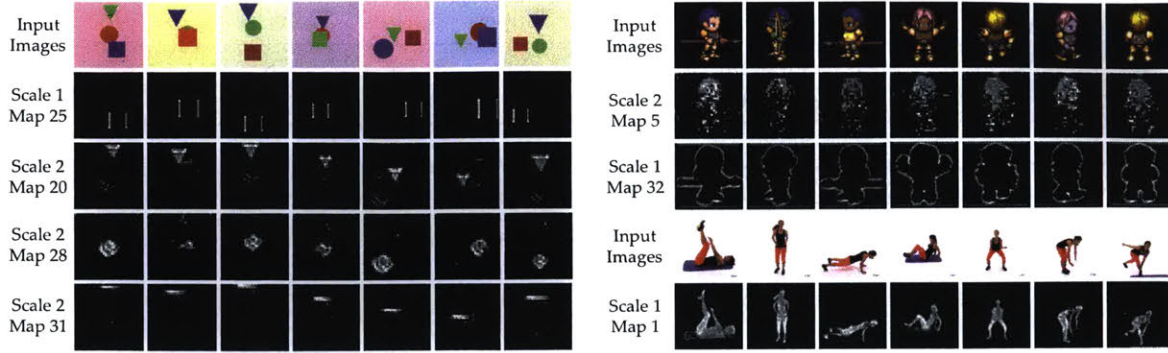
Figure 7-9: **Learned layers** on the Shapes dataset (left), the Sprites dataset (top right), and the Exercise dataset (bottom right). Our system is able to implicitly discover semantic structure from this self-supervised task. On the Shapes dataset, it learns to detect circles and triangles; it also detects vertical boundaries of squares, as squares always move horizontally. On the Exercise dataset, it learns to respond to only humans, not the carpet below them, as the carpet never moves.

better handle more complex visual scenes.

7.6 Analyses

In this section, we present an analysis of the learned network to demonstrate what it captures. In particular, we visualize the learned feature maps (Section 7.6.1), and show that the network implicitly learns object and edge detectors. Additionally, we compute statistics of the latent representation (Section 7.6.2) to verify that the network is learning a compact, informative representation of the motion manifold. We then visualize certain dimensions of the latent representation to reveal their meanings (Section 7.6.4).

7.6.1 Visualizing Learned Layers

Our network synthesizes the movement of objects by first identifying layers in the input and transferring each layer. Therefore, we expect that the these layers carry both low-level information such as object contours, and high-level information such as object parts.

To verify this, we visualize the learned feature maps (Figure 7-3b) in Figure 7-9. Even without supervision, our network learns to detect objects or contours in the image. For example, we see that the network automatically learns object detectors and edge detectors on the shapes dataset. It also learns a hair detector and a body detector on the sprites and exercise datasets, respectively. The part detectors have a sharper boundary on the shapes dataset than on the other two. This is because the shapes dataset has well-defined concepts of independently movable parts; in contrast, body parts in real videos have indistinguishable motion distributions.

145

Dataset	Shapes	Sprites	Exercise
Non-zero element in z_{mean}	299	54	978
Dominating PCA components	5	5	47

Table 7.1: **Statistics of the 3,200-dimensional motion vector** z. The network learns a sparse latent representation, encoding high-level knowledge using minimal bits.

(a) The Shapes dataset   (b) The Sprites dataset   (c) The Exercise dataset
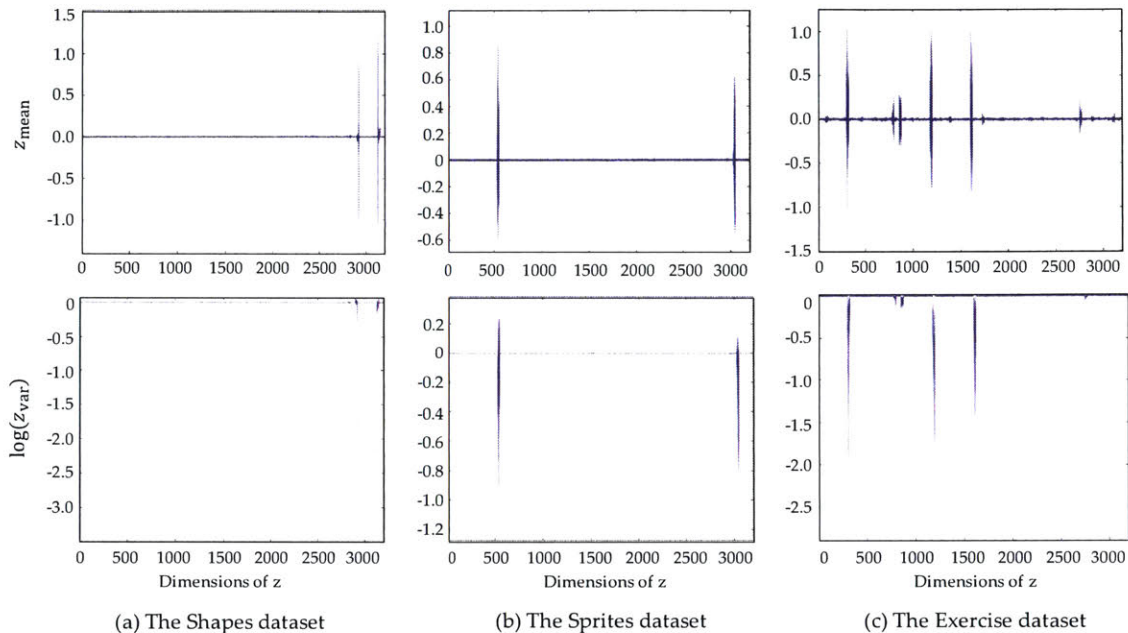
Figure 7-10: **Statistics of latent vectors** z_{mean} and z_{logvar} extracted from 1,000 image pairs from the Shapes, Sprites, and Exercise datasets, respectively. Each vertical line is for a single dimension in z. Although the z vector has 3,200 dimensions, only a small number of those have values deviating from the prior (mean 0, log-variance 0). This shows the system is learning to encode motion information in a compact way. Also, as the motion in the Exercise dataset is more complex, the system needs more bits for encoding (cf., Table 7.1).

7.6.2 The Sparsity of the Latent Representation

Although our latent motion representation, z, has 3,200 dimensions, its intrinsic dimensionality is much smaller. We show statistics in Table 7.1. There are two main messages. First, z_{mean} is very sparse. There are 299 non-zero elements of z_{mean} in the shapes dataset, 54 in sprites, and 978 in exercise. The sprites dataset requires fewer non-zero elements than the shapes dataset, because while the visual appearance of the characters are more complex, their motion falls into a few pre-defined categories (e.g., thrust) and is thus simpler than the continuous motion of the shapes. Second, the independent components of z are even fewer. We run principle component analysis (PCA) on the z_{mean}s obtained from a set of training images, and find that for each dataset, only a small fraction of components cover 95% of the variance in z_{mean} (5 in shapes, 5 in sprites, and 47 in exercise).

Figure 7-10 further shows detailed statistics of z_{mean} and z_{logvar} (a.k.a., $\log(z_{\mathrm{var}})$). For each dataset, we randomly select 1,000 train samples and calculate the corresponding z_{mean} and z_{logvar} through the motion encoder (Figure 7-3a). The first row shows the distribution of z_{mean} and the second row shows the distribution of z_{logvar}. The x-axis in each figure corresponds to the 3,200 dimensions in z_{mean} and z_{logvar}. For each dimension k, the blue region reflects the interval

$$[\mathrm{mean}(z_k) - \mathrm{std}(z_k), \mathrm{mean}(z_k) + \mathrm{std}(z_k)], \tag{7.12}$$

where $\mathrm{mean}(z_k)$ is the mean of the 1,000 values in the k dimension for the 1,000 samples, and $\mathrm{std}(z_k)$ is the standard deviation.

One interesting observation from Figure 7-10 is that, for most dimensions, z_{mean} is very close to 0 and z_{var} is very close to 1 (z_{logvar} equals to 0). This is because the training object Equation 7.2 minimizes the KL divergence between $N(\vec{0}, \boldsymbol{I})$ and $N(z_{\mathrm{mean}}, \mathrm{diag}(z_{\mathrm{var}}))^{**}$, and the minimizer of the KL divergence is $z_{\mathrm{mean}} = \vec{0}$ and $z_{\mathrm{var}} = \vec{1}$.

However, not all dimensions of z_{mean} and z_{logvar} are 0, and those non-zero dimensions actually encode the semantic information of how objects move between frames. Recall that to calculate the compact motion representation z, the motion encoder first estimates z_{mean} and z_{var} from two input frames, and then samples z from $N(z_{\mathrm{mean}}, \mathrm{diag}(z_{\mathrm{var}}))$. Let us rewrite the sampling as $z_k = z_{\mathrm{mean},k} + \epsilon_k z_{\mathrm{var},k}$, where k is a dimension index and ϵ_k is a random Gaussian noise with zero mean and unit variance. If $z_{\mathrm{var},k}$ is large, the corresponding dimension in the motion vector z is mostly corrupted by the random Gaussian noise, and only those dimensions with small $z_{\mathrm{var},k}$ are able to transmit motion information for next frame synthesis.

In other words, z_{mean} carries the actual motion information, while z_{var} is an indicator of whether a dimension is being actively used. This corresponds to our observation in Figure 7-10, where only uncorrupted dimensions (i.e., those with small z_{var}) have non-zero z_{mean}. Similar discussions were also presented by Hinton and Van Camp [1993] and concurrently by Higgins et al. [2016].

To further demonstrate how KL-divergence criterion ensures the sparsity of motion vector z, we also vary the weight of KL-divergence criterion λ during training. As shown in Figure 7-11, when λ is small, most of dimensions of $\log(z_{\mathrm{var}})$ are smaller than 0, and z_{mean} is not sparse (left of Figure 7-11). When we increase λ, the network is forced to encode the information in a more compact way, and the learned representation becomes sparser (right of Figure 7-11).

All these results suggest that our network has learned a compact representation of motion in an unsupervised fashion, and encodes high-level knowledge using a

---

$^{**}\mathrm{diag}(z_{\mathrm{var}})$ denotes a diagonal matrix whose diagonal elements are z_{var}.

(a) $\lambda = 10^{-5}$ (b) $\lambda = 10^{-4}$
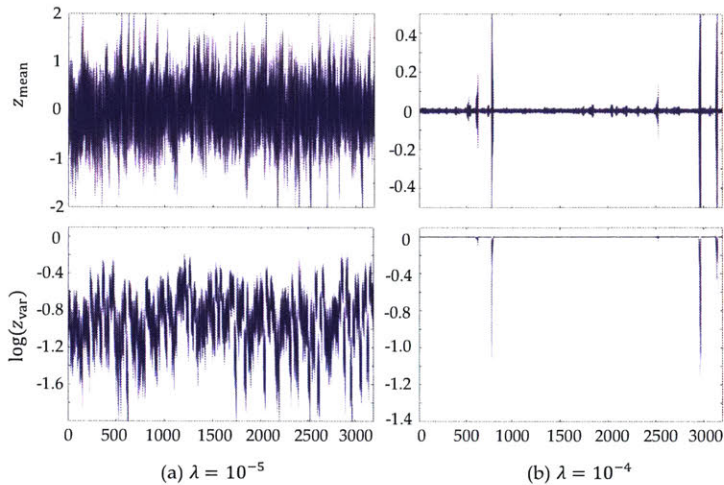
Figure 7-11: **Statistics of latent vectors z using different values of λ**, extracted from 1,000 image pairs from the Shapes dataset. The latent vector becomes sparser when λ is larger, i.e., the network is encoding motion in a more compact way.

Size of z	8	32	128	3,200
# components	3	3	3	5
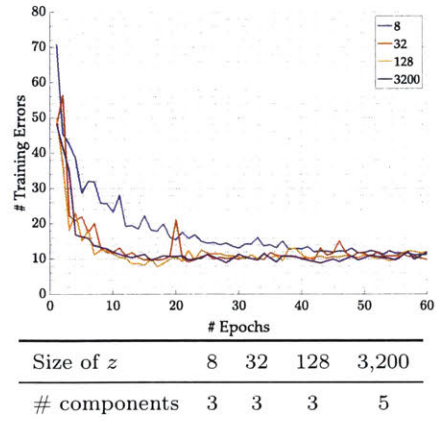
Figure 7-12: **Ablation study on the size of z.** Our model automatically discovers the underlying dimension of the motion (the number of dominating PCA components) in the *Shapes* dataset. When we shrink the size of z, the convergence gets a little slower (especially when z has only 8 dimensions), but the results are essentially the same.

small number of bits, rather than simply memorizing training samples. In the next subsection, we will also illustrate what motion information is actually learned.

7.6.3   Varying the Size of the Latent Representation

We have explored the sparsity of the latent representation z, suggesting the model learns to encode essential information using only a few bits, despite z itself has 3,200 dimensions. Here, we conduct an additional ablation study to understand the minimal number of dimensions that are necessary for the network to initialize the learning process.

We experiment on the *Shapes* dataset. Table 7.1 tells us that our model can learn to encode motion information using only 5 dominating PCA components. We therefore explore 4 different sizes of z: 8, 32, 128, and 3,200, as shown in Figure 7-12. All networks converge regardless of the dimensions of z, with a slight difference in convergence time (the network with smaller z takes longer to converge). Also, the number of dominating PCA components is always small (3–5), suggesting that as long as the dimension of the latent representation z is larger than its intrinsic dimension, the model consistently learns to encode the information compactly, regardless of its dimension.

148
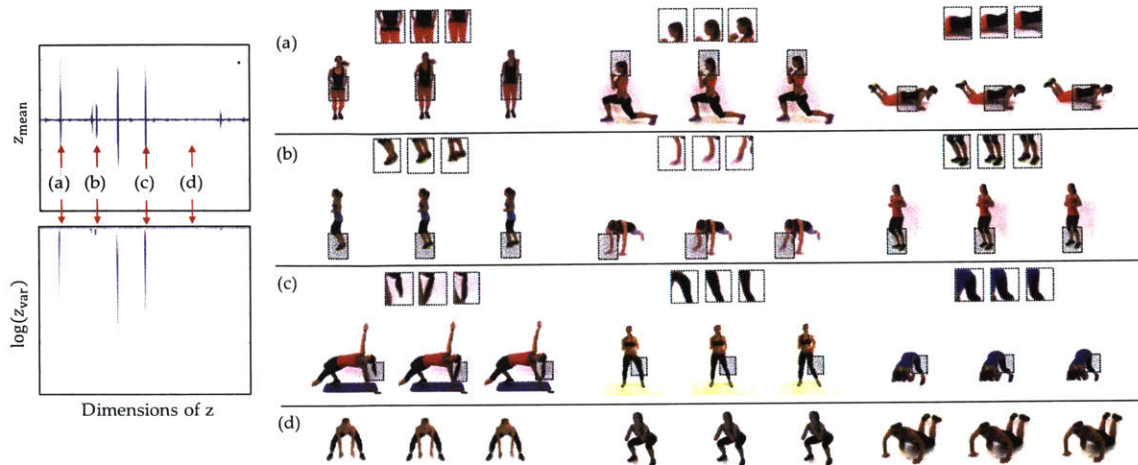
Figure 7-13: **We visualize the effect of varying individual dimensions in the latent representation** z, revealing the system is learning a disentangled, interpretable motion representation. For the dimensions whose log-variance is smaller than 0 (a–c), they record a certain type of motion. For example, dimension (a) corresponds to humans move upwards, dimensions (b) and (c) correspond to moving arms, hair, or legs to the left. For the dimensions whose log-variance is very close to 0, they record no motion information: changing the value of dimension (d) results in no motion in predicted frames.

7.6.4 Visualizing the Latent Representation

We visualize the encoded motion by manipulating individual dimensions of the representation z, while keeping other dimensions constant. Through this, we have found that each dimension corresponds to a certain type of motion. We show results in Figure 7-13. On the exercise dataset, varying one dimension of z causes the girl to stand-up, and varying another causes her to move her leg. The effect is consistent across input images, showing individual dimensions in the latent vector z carries abstract, higher-level knowledge. Also notice that only dimensions with smaller variance z_{var} contain semantic motion information (Figure 7-13a to Figure 7-13c). Manipulating dimensions with variances close to 1 results in no significant motion (Figure 7-13d).

7.6.5 Handling Disocclusions

Synthesizing future frames often involves handling disocclusions. Here we systematically evaluate it on a new dataset, *Shapes+Texture*, where the primitives in the *Shapes* dataset now have horizontal (squares), vertical (triangles), or checkerboard patterns (circles). Figure 7-14 shows our model handles disocclusions well, correctly synthesizing the object texture even if it's not visible in the input. This is further supported by Figure 7-9, showing the network learns feature maps that correspond to *amodal* segments of shapes.

(a) Input   (b) 2 sampled future frames

Reference

Input +
Prediction

Reference

Input +
Prediction
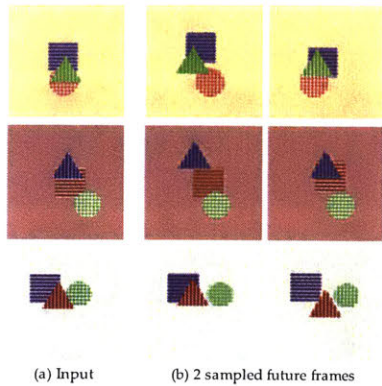
Figure 7-14: **Our model handles disocclusions well.** On the *Shapes+Texture* dataset, our model is able to complete shapes with their corresponding texture after hallucinating their motion.

Figure 7-15: **Results on visual analogy-making**, where we want to transfer the motion in a reference pair to a target image. We conduct experiments on two datasets, Sprites and Exercise, and mark the predicted frames in red. Our algorithm is able to transfer high-level motion (e.g. downwards motion of a human body, as opposed to pixel-wise motion) in a semantically plausible way.

Model	spellcast	thrust	walk	slash	shoot	average
Add	41.0	53.8	55.7	52.1	77.6	56.0
Dis	40.8	55.8	52.6	53.5	79.8	56.5
Dis+Cls	13.3	24.6	17.2	**18.9**	40.8	23.0
Ours	**9.5**	**11.5**	**11.1**	28.2	**19.0**	**15.9**

Table 7.2: **Mean squared pixel error on test analogies, by animation.** The first three models (Add, Dis, and Dis+Cls) are from Reed et al. [2015]. Compared with them, our model achieves lower errors.

7.7 Applications

Our framework models a general problem and therefore has a wide range of applications. Here we present two possible applications of our framework: visual analogy-making and extrapolation for generating video sequences.

7.7.1 Zero-Shot Visual Analogy-Making

Recently, Reed et al. [2015] studied the problem of inferring the relationship between a pair of reference images and synthesizing an image analogy by applying the inferred relationship to a test image. For example, the character shown in the top row of Figure 7-15a leans toward to the right; the task is to transfer its leaning motion to the target, bottom-left image.

The method by Reed et al. requires a set of quadruples during supervision (two source images and two target images). In contrast, our network is able to preform this task without first training using the quadruple sets. Specifically, we extract the motion vector, z, from two reference frames using our motion encoder (Figure 7-3a).
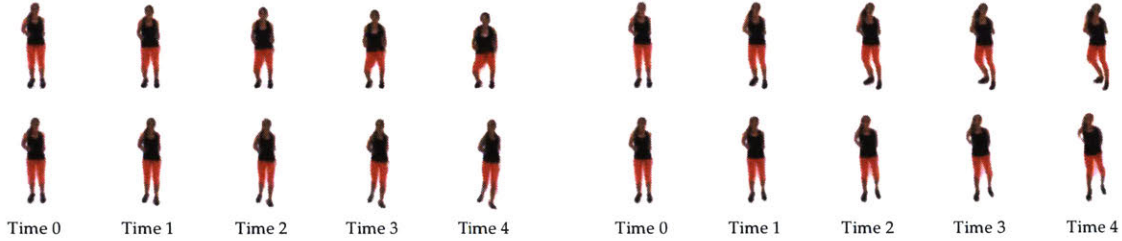
Figure 7-16: **Generating video sequences by repeatedly applying a sampled motion representation** z. Our model is able to synthesize reasonable videos of human moving in various ways, though artifacts gradually emerge over time (e.g. the thighs become bigger in the top-right example).

We then use the extracted motion vector z to synthesize an analogy-image given a new test image. In this way, our network learns to transfer motion from a source pair to the target image, without requiring any form of supervision (see Figure 7-15). As shown in Table 7.2, our algorithm outperforms that by Reed et al. [2015].

7.7.2 Extrapolation

Our network can also be used to generate video sequences by extrapolation. In Figure 7-16, we synthesize videos from a single image by simply repeatedly applying the sample motion. Given an input image frame I_1 and sampled motion vector z, we first synthesize the second frame I_2 from the first frame I_1 and motion vector z, and then synthesize the third frame I_3 from the second using the same motion vector. We see that our framework generates plausible video sequences. Recent work on modeling transitions across possible motions [Chao et al., 2017] could serve as alternative way to extend our framework to longer-term video generation. Compared with most of previous deterministic video synthesis networks [Srivastava et al., 2015, Mathieu et al., 2016], our approach can sample multiple possible ways a person can move, as shown Figure 7-16. One limitation of this approach is that artifacts gradually emerge over time. It may be possible to reduce these artifacts using a learned image prior [Zhang et al., 2017a].

7.8   Extending to Hierarchical Structure

When we see objects move, we not only discover object parts that move together, but also understand their relationships. Take a human body as an example. We want our model to parse human parts (e.g., torso, hands, and legs) and to learn their structure (e.g., hands and legs are both parts of the human body). In this section, we extend the visual dynamics model to also discover the hierarchical structure among object parts. We name our new model Parts, Structure, and Dynamics (PSD).

Formally, given a pair of images $\{\mathcal{I}_1, \mathcal{I}_2\}$, let $\mathcal{M}$ be the Lagrangian motion map (i.e. optical flow). Consider a system that learns to segment object parts and to

| (a) Pairs of images | (b) Global motions of parts | (c) Local motions of parts |
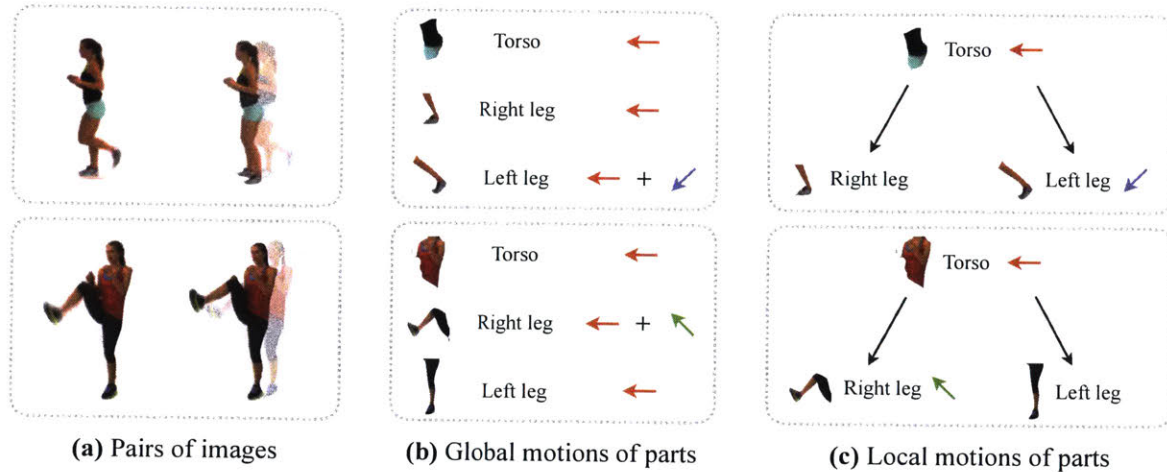
Figure 7-17: **Hierarchical motion decomposition.** Knowing that the legs are part of human body, the legs' motion can be decomposed as the sum of the body's motion and the legs' local motion.

capture their motions, without modeling their structure. Its goal is to find a segment decomposition of $\mathcal{I}_1 = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_n\}$, where each segment $\mathcal{O}_k$ corresponds to an object part with distinct motion. Let $\{\mathcal{M}_1^{\mathrm{g}}, \mathcal{M}_2^{\mathrm{g}}, \ldots, \mathcal{M}_n^{\mathrm{g}}\}$ be their corresponding motions.

Beyond that, we assume that these object parts form a hierarchical tree structure: each part k has a parent p_k, unless itself is the root of a motion tree. Its motion $\mathcal{M}_k^{\mathrm{g}}$ can therefore be decomposed into its parent's motion $\mathcal{M}_{p_k}^{\mathrm{g}}$ and a local motion component $\mathcal{M}_k^{\mathrm{l}}$ within its parent's reference frame. Specifically, $\mathcal{M}_k^{\mathrm{g}} = \mathcal{M}_{p_k}^{\mathrm{g}} + \mathcal{M}_k^{\mathrm{l}}$, if k is not a root. Here we make use of the fact that Lagrangian motion components $\mathcal{M}_k^{\mathrm{l}}$ and $\mathcal{M}_{p_k}^{\mathrm{g}}$ are additive.

Figure 7-17 gives an intuitive example: knowing that the legs are part of human body, the legs' motion can be written as the sum of the body's motion (e.g., moving to the left) and the legs' local motion (e.g., moving to lower or upper left). Therefore, the objective of our model is, in addition to identifying the object components $\{\mathcal{O}_k\}$, learning the hierarchical tree structure $\{p_k\}$ to effectively and efficiently explain the object's motion.

Such an assumption makes it possible to decompose the complex object motions into simple and disentangled local motion components. Reusing local components along the hierarchical structure helps to reduce the description length of the motion map $\mathcal{M}$. Therefore, such a decomposition should naturally emerge within a design with information bottleneck that encourages compact, disentangled representations. In the next section, we introduce the general philosophy behind our model design and the individual components within.
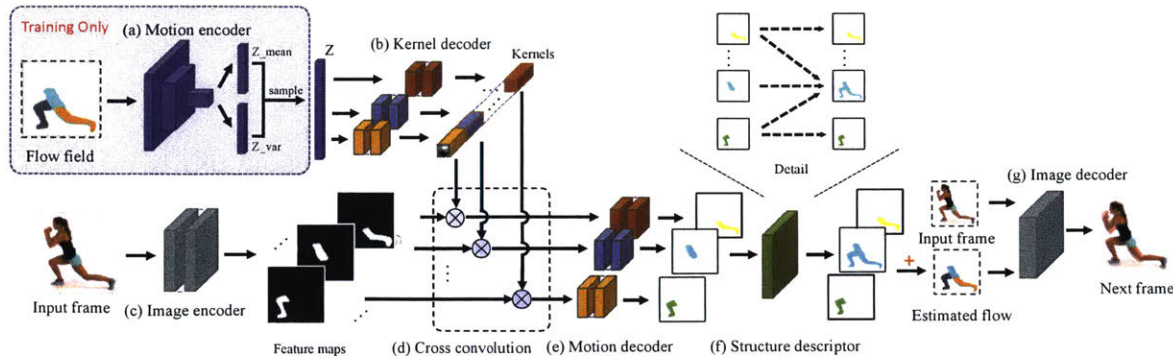
Figure 7-18: **Our PSD model has seven components**: **(a)** motion encoder; **(b)** kernel decoder; **(c)** image encoder; **(d)** cross convolution; **(e)** motion decoder; **(f)** structural descriptor; and **(g)** image decoder.

7.8.1 Learning Parts, Structure, and Dynamics

Figure 7-18 shows an overview of our Parts, Structure, and Dynamics (PSD) model. It is similar to the original visual dynamics model, with the addition of a structure descriptor.

Motion can be decomposed in a layer-wise manner, separately modeling different object component's movement [Wang and Adelson, 1993]. Motivated by this, our model first decomposes the input frame $\mathcal{I}_1$ into multiple feature maps using an *image encoder* (Figure 7-18c). Intuitively, these feature maps correspond to separate object components. Our model then performs convolutions (Figure 7-18d) on these feature maps using separate kernels obtained from a *kernel decoder* (Figure 7-18b), and synthesizes the local motions $\mathcal{M}_k^l$ of separate object components with a *motion decoder* (Figure 7-18e). After that, our model employs a *structural descriptor* (Figure 7-18f) to recover the global motions $\mathcal{M}_k^g$ from local motions $\mathcal{M}_k^l$, and then compute the overall motion $\mathcal{M}$. Finally, our model uses an *image decoder* (Figure 7-18g) to synthesize the next frame $\mathcal{I}_2$ from the input frame $\mathcal{I}_1$ and the overall motion $\mathcal{M}$.

Our PSD model can be seen as a conditional variational autoencoder. During training, it employs an additional *motion encoder* (Figure 7-18a) to encode the motion into the latent representation z; during testing, it instead samples the representation z from its prior distribution $p_z(z)$, which is assumed to be a multivariate Gaussian distribution, where each dimension is i.i.d., zero-mean, and unit-variance.

Architecture-wise, our model is the same as the original visual dynamics model, with the addition of the structure descriptor. Our structural descriptor recovers the global motions $\{\mathcal{M}_k^g\}$ from the local motions $\{\mathcal{M}_k^l\}$ and the hierarchical tree structure

153

$\{p_k\}$ using

$$\mathcal{M}_k^g = \mathcal{M}_k^l + \mathcal{M}_{p_k}^g = \mathcal{M}_k^l + \left(\mathcal{M}_{p_k}^l + \mathcal{M}_{p_{p_k}}^g\right) = \cdots$$

$$= \mathcal{M}_k^l + \sum_{i \neq k} [i \in P_k] \cdot \mathcal{M}_i^l, \quad \text{where } P_k \text{ is the set of ancestors of } \mathcal{O}_k. \quad (7.13)$$

Then, we define the structural matrix $\mathcal{S}$ as $\mathcal{S}_{ik} = [i \in P_k]$, where each binary indicator $\mathcal{S}_{ik}$ represents whether $\mathcal{O}_i$ is an ancestor of $\mathcal{O}_k$. This is what we aim to learn, and it is shared across different data points. In practice, we relax the binary constraints on $\mathcal{S}$ to $[0,1]$ to make this module differentiable: $\mathcal{S}_{ik} = \text{sigmoid}(\mathcal{W}_{ik})$, where $\mathcal{W}_{ik}$ are trainable parameters. Finally, the overall motion can be simply computed as $\mathcal{M} = \sum_k \mathcal{M}_k^g$.

Training details. Our objective function $\mathcal{L}$ is a weighted sum over three separate components:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \beta \cdot \mathcal{L}_{\text{reg}} + \gamma \cdot \mathcal{L}_{\text{struct}}, \quad \text{where } \beta \text{ and } \gamma \text{ are two weighting factors.} \quad (7.14)$$

The first component is the *pixel-wise reconstruction loss*, which enforces our model to accurately estimate the motion $\mathcal{M}$ and synthesize the future frame $\mathcal{I}_2$. We have $\mathcal{L}_{\text{recon}} = \|\mathcal{M} - \hat{\mathcal{M}}\|_2 + \alpha \cdot \|\mathcal{I}_2 - \hat{\mathcal{I}}_2\|_2$, where α is a weighting factor (which is set to 10^3 in our experiments).

The second component is the *variational loss*, which encourages our model to use as few dimensions in the latent representation z as possible [Xue et al., 2016, Higgins et al., 2017]. We have $\mathcal{L}_{\text{reg}} = \mathcal{D}_{\text{KL}}\big(\mathcal{N}(z_{\text{mean}}, z_{\text{var}}) \,\|\, p_z(z)\big)$, where $\mathcal{D}_{\text{KL}}(\cdot \,\|\, \cdot)$ is the KL-divergence, and $p_z(z)$ is the prior distribution of the latent representation (which is set to normal distribution in our experiments).

The last component is the *structural loss*, which encourages our model to learn the hierarchical tree structure so that it helps the motions $\mathcal{M}^l$ be represented in an efficient way: $\mathcal{L}_{\text{struct}} = \sum_{k=1}^d \|\mathcal{M}_k^l\|_2$. Note that we apply the structural loss on local motion fields, not on the structural matrix. In this way, the structural loss serves as a regularization, encouraging the motion field to have small values.

We implement our PSD model in PyTorch [Paszke et al., 2017]. Optimization is carried out using ADAM [Kingma and Ba, 2015] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use a fixed learning rate of 10^{-3} and mini-batch size of 32. We propose the two-stage optimization schema, which first learns the disentangled and then learns the hierarchical representation.

In the first stage, we encourage the model to learn a *disentangled* representation (without structure). We set the γ in Equation 7.14 to 0 and fix the structural matrix $\mathcal{S}$ to the identity $\mathcal{I}$. The β in Equation 7.14 is the same as the one in the β-VAE [Higgins et al., 2017], and therefore, larger β's encourage the model to learn a more disentangled

(a) Input　(b) Reconstruction (c) Ground truth (d) Offensive player　(e) Ball　(f) Defensive player　(g) Hierarchical tree structure
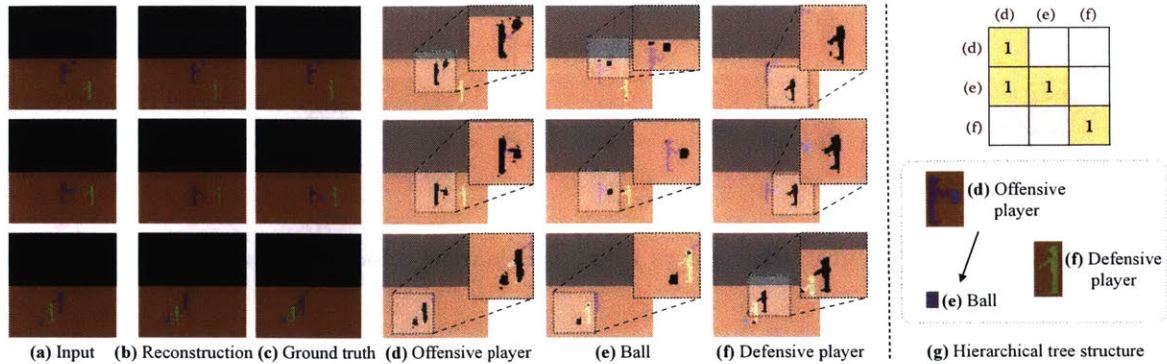
Figure 7-19: **Results of segmenting objects (d-f) and learning hierarchical structure (g) on Atari games**

representation. We first initialize the β to 0.1 and then adaptively double the value of β when the reconstruction loss reaches a preset threshold.

In the second stage, we train the model to learn the *hierarchical* representation. We fix the weights of motion encoder and kernel decoder, and set the β to 0. We initialize the structural matrix $\mathcal{S}$, and optimize it with the image encoder and motion decoder jointly. We adaptively tune the value of γ in the same way as the β in the first stage.

7.8.2　Experiments

We evaluate our model on Atari games of basketball playing and real-world human motions.

Atari games of playing basketball.　We evaluate our model on a dataset of Atari games. In particular, we select the Basketball game from the Atari 2600. In this game, there are two players competing with each other. Each player can move in eight different directions. The *offensive* player constantly dribbles the ball and throws the ball at some moment; while the *defensive* player tries to steal the ball from his opponent player. We download a video of playing this game from YouTube and construct a dataset with 5,000 pairs for training and 500 for testing.

Our PSD model discovers three meaningful dimensions in the latent representation z. We visualize the feature maps in these three dimensions in Figure 7-19. We observe that one dimension (in Figure 7-19d) is learning the *offensive player with ball*, another (in Figure 7-19e) is learning the *ball*, and the other (in Figure 7-19f) is learning the *defensive player*. We construct the hierarchical tree structure among these three dimensions from the structural matrix $\mathcal{S}$. As illustrated in Figure 7-19g, our PSD model is able to discover the relationship between the ball and the players: the offensive player *controls* the ball. This is because our model observes that the ball always moves along with the offensive player.

155
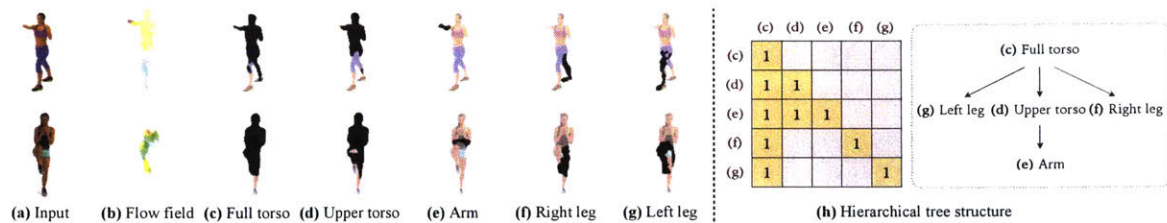
Figure 7-20: **Results of segmenting parts (c-g) and learning hierarchical structure (h) on human motions**
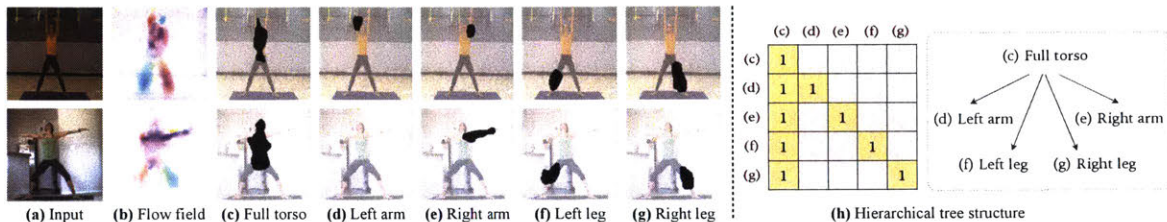
Figure 7-21: **Results of segmenting parts (c-g) and learning hierarchical structure (h) on human motions with complex backgrounds**

Moving humans. We then evaluate our method on two datasets of real-world human motions: the human exercise dataset used in Xue et al. [2016] and the yoga dataset used in Balakrishnan et al. [2018]. We estimate the optical flows between frames by an off-the-shelf package [Liu, 2009]. Compared with previous datasets, these two require much more complicated visual perception, and they have challenging hierarchical structures. In the human exercise dataset, there are 50,000 pairs of frames used for training and 500 for testing. As for the yoga dataset, there are 4,720 pairs of frames for training and 526 for testing.

In Figure 7-20 and Figure 7-21, we visualize the feature maps corresponding to the active latent dimensions. It turns out that each of these dimensions corresponds to one particular human part: full torsos (7-20c, 7-21c), upper torsos (7-20d), arms (7-20e), left arms (7-21d), right arms (7-21e), right legs (7-20f, 7-21g), and left legs (7-20g, 7-21f). Note that it is extremely challenging to distinguish different parts from motions, because different parts (e.g., arms and legs) might have similar motions (see Figure 7-20b). R-NEM is not able to segment any meaningful parts, let alone structure, while our PSD model gives imperfect yet reasonable part segmentation results.

For quantitative evaluation, we collect the ground truth part segmentation for 30 images and compute the *intersection over union* (IoU) between the ground-truth and the prediction of our model and the other two baselines (NEM, R-NEM). The quantitative results are presented in Table 7.3. Our PSD model significantly outperforms the two baselines.

We recover the hierarchical tree structure among these dimensions from the

156

	Full torso	Upper torso	Arm	Left leg	Right leg	Overall
NEM	0.298	0.347	0.125	0.264	0.222	0.251
R-NEM	0.321	0.319	0.220	0.294	0.228	0.276
PSD (ours)	**0.697**	**0.574**	**0.391**	**0.374**	**0.336**	**0.474**

Table 7.3: **Quantitative results (IoUs) of object segmentation on the human exercise dataset**

structural matrix $\mathcal{S}$. From Figure 7-20h, our PSD model is able to discover that the upper torso and the legs are part of the full torso, and the arm is part of the upper torso, and from Figure 7-21h, our PSD model discovers that the arms and legs are parts the full torso.

7.9 Discussion

In this chapter, we have proposed a novel framework that learns to approximate graphics and physics engines, which enables sampling future frames from a single image. Our method incorporates a variational autoencoder for learning compact motion representations and layer-based synthesis algorithm to generate realistic movement of objects. We have demonstrated that our framework works well on both synthetic and real-life videos.

The key component of our frame synthesis model is to decompose an input image into different layers and model the movement of each layer through convolution. This decomposition can well approximate the motion of a single deformable object, such as a human body. We have also demonstrated that our model can be extended to simultaneously capture the hierarchical structure among these layers. In the future, we would also like to extend it to handle more complicated and stochastic motion, e.g., water flowing, and we hope that the learned motion prior may find its use in other computer vision and computational photography applications.

THIS PAGE INTENTIONALLY LEFT BLANK

Part III

Reasoning: Learning with Program Executors

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

Learning to Discover Concepts
from Images and Language

In Part I and II, we have presented models that learn to see and predict by incorporating simulators in the loop. We have also seen models that learn to approximate simulators themselves. In Part III, we look into high-level cognitive reasoning problems: what is the relation among objects in the scene? are there any repetitive structures or regularities in the shapes, objects, or image segments we observe? can we exploit them for question answering or image manipulation? We aim to answer these questions in Part III via the help of program execution engines, in place of the graphics and physics engines we used in Part I and II.

In this chapter, we first study the problem of visual concept learning, as symbolic concepts are the basis for cognitive reasoning. We propose the Neuro-Symbolic Concept Learner (NS-CL), a model that learns visual concepts, words, and semantic parsing of sentences without explicit supervision on any of them; instead, our model learns by simply looking at images and reading paired questions and answers. Our model builds an object-based scene representation and translates sentences into executable, symbolic programs. To bridge the learning of two modules, we use a neuro-symbolic reasoning module that executes these programs on the latent scene representation. Analogical to human concept learning, the perception module learns visual concepts based on the language description of the object being referred to. Meanwhile, the learned visual concepts facilitate learning new words and parsing new sentences. We use curriculum learning to guide the searching over the large compositional space of images and language. Extensive experiments demonstrate the accuracy and efficiency of our model on learning visual concepts, word representations, and semantic parsing of sentences. Further, our method allows easy generalization to new object attributes, compositions, language concepts, scenes and questions, and even new program domains. It also empowers applications including visual question answering and bidirectional image-text retrieval.

This chapter includes materials previously published as Mao et al. [2019a], Yi et al.

[2018]. Jiayuan Mao, Kexin Yi, and Chuang Gan contributed significantly to the materials presented in this chapter.

8.1 Introduction

Humans are capable of learning visual concepts by jointly understanding vision and language [Fazly et al., 2010, Chrupała et al., 2015, Gauthier et al., 2018]. Consider the example shown in Figure 8-1-I. Imagine that someone with no prior knowledge of colors is presented with the images of the red and green cubes, paired with the questions and answers. They can easily identify the difference in objects' visual appearance (in this case, color), and align it to the corresponding words in the questions and answers (Red and Green). Other object attributes (e.g., shape) can be learned in a similar fashion. Starting from there, humans are able to inductively learn the correspondence between visual concepts and word semantics (e.g., spatial relations and referential expressions, Figure 8-1-II), and unravel compositional logic from complex questions assisted by the learned visual concepts (Figure 8-1-III, also see Abend et al. [2017]).

Motivated by this, we propose the neuro-symbolic concept learner (NS-CL), which jointly learns visual perception, words, and semantic language parsing from images and question-answer pairs. NS-CL has three modules: a neural-based perception module that extracts object-level representations from the scene, a visually-grounded semantic parser for translating questions into executable programs, and a symbolic program executor that reads out the perceptual representation of objects, classifies their attributes/relations, and executes the program to obtain an answer.

NS-CL learns from natural supervision (i.e., images and QA pairs), requiring no annotations on images or semantic programs for sentences. Instead, analogical to human concept learning, it learns via curriculum learning. NS-CL starts by learning representations/concepts of individual objects from short questions (e.g., What's the color of the cylinder?) on simple scenes (≤ 3 objects). By doing so, it learns object-based concepts such as colors and shapes. NS-CL then learns relational concepts by leveraging these object-based concepts to interpret object referrals (e.g., Is there a box right of a cylinder?). The model iteratively adapts to more complex scenes and highly compositional questions.

NS-CL's modularized design enables interpretable, robust, and accurate visual reasoning: it achieves state-of-the-art performance on the CLEVR dataset [Johnson et al., 2017a]. More importantly, it naturally learns disentangled visual and language concepts, enabling combinatorial generalization w.r.t. both visual scenes and semantic programs. In particular, we demonstrate four forms of generalization. First, NS-CL generalizes to scenes with more objects and longer semantic programs than those in the training set. Second, it generalizes to new visual attribute compositions, as

162

I. Learning basic, object-based concepts.

Q: What's the color of the object?
A: Red.
Q: Is there any cube?
A: Yes.

Q: What's the color of the object?
A: Green.
Q: Is there any cube?
A: Yes.

II. Learning relational concepts based on referential expressions.

Q: How many objects are right of the red object?
A: 2.
Q: How many objects have the same material as the cube?
A: 2

III. Interpret complex questions from visual cues.

Q: How many objects are both right of the green cylinder and have the same material as the small blue ball?
A: 3

Figure 8-1: **Human visual concept learning:** humans learn visual concepts, words, and semantic parsing jointly and incrementally.

demonstrated on the CLEVR-CoGenT [Johnson et al., 2017a] dataset. Third, it enables fast adaptation to novel visual concepts, such as learning a new color. Finally, the learned visual concepts transfer to new tasks, such as image-caption retrieval, without any extra fine-tuning.

8.2 Related Work

Our model is related to research on joint learning of vision and natural language. In particular, there are many papers that learn visual concepts from descriptive languages, such as image-captioning or visually-grounded question-answer pairs [Kiros et al., 2014, Shi et al., 2018, Mao et al., 2016, Vendrov et al., 2016, Ganju et al., 2017], dense language descriptions for scenes [Johnson et al., 2016b], video-captioning [Donahue et al., 2015] and video-text alignment [Zhu et al., 2015].

Visual question answering (VQA) stands out as it requires understanding both visual content and language. The state-of-the-art approaches usually use neural attentions [Malinowski and Fritz, 2014, Chen et al., 2016a, Yang et al., 2016, Xu and Saenko, 2016]. Beyond question answering, Johnson et al. [2017a] proposed the CLEVR (VQA) dataset to diagnose reasoning models. CLEVR contains synthetic visual scenes and questions generated from latent programs. Table 8.1 compares our model with state-of-the-art visual reasoning models [Andreas et al., 2016, Suarez et al., 2018, Santoro et al., 2017] along four directions: visual features, semantics, inference, and the requirement of extra labels.

For visual representations, Johnson et al. [2017b] encoded visual scenes into a convolutional feature map for program operators. Mascharka et al. [2018], Hudson and Manning [2018] used attention as intermediate representations for transparent program execution. Recently, Yi et al. [2018] explored an interpretable, object-based visual representation for visual reasoning, NS-VQA, which we will introduce in Section 8.3. It performs well, but requires fully-annotated scenes during training. Our model also adopts an object-based visual representation, but the representation is learned only based on natural supervision (questions and answers).

Models	Visual Features	Semantics	Extra Labels		Inference
			# Prog.	Attr.	
FiLM [Perez et al., 2018]	Convolutional	Implicit	0	No	Feature Manipulation
IEP [Johnson et al., 2017b]	Convolutional	Explicit	700K	No	Feature Manipulation
MAC [Hudson and Manning, 2018]	Attentional	Implicit	0	No	Feature Manipulation
Stack-NMN [Hu et al., 2018]	Attentional	Implicit	0	No	Attention Manipulation
TbD [Mascharka et al., 2018]	Attentional	Explicit	700K	No	Attention Manipulation
NS-VQA [Yi et al., 2018]	Object-Based	Explicit	0.2K	Yes	Symbolic Execution
NS-CL [Mao et al., 2019a]	Object-Based	Explicit	0	No	Symbolic Execution

Table 8.1: **Comparison with other frameworks on the CLEVR VQA dataset**, w.r.t. visual features, implicit or explicit semantics and supervisions

Anderson et al. [2018] also proposed to represent the image as a collection of convolutional object features and gained substantial improvements on VQA. Their model encodes questions with neural networks and answers the questions by question-conditioned attention over the object features. In contrast, NS-CL parses question inputs into programs and executes them on object features to get the answer. This makes the reasoning process interpretable and supports combinatorial generalization over quantities (e.g., counting objects). Our model also learns general visual concepts and their association with symbolic representations of language. These learned concepts can then be explicitly interpreted and deployed in other vision-language applications such as image caption retrieval.

There are two types of approaches in semantic sentence parsing for visual reasoning: implicit programs as conditioned neural operations (e.g., conditioned convolution and dual attention) [Perez et al., 2018, Hudson and Manning, 2018] and explicit programs as sequences of symbolic tokens [Andreas et al., 2016, Johnson et al., 2017b, Mascharka et al., 2018]. As a representative, Andreas et al. [2016] built modular and structured neural architectures based on programs for answering questions. Explicit programs gain better interpretability, but usually require extra supervision such as ground-truth program annotations for training. This restricts their application. We propose to use visual grounding as distant supervision to parse questions in natural languages into explicit programs, with *zero* program annotations. Given the semantic parsing of questions into programs, NS-VQA [Yi et al., 2018] used a purely symbolic executor for the inference of the answer in the logic space. Compared with that, we propose a quasi-symbolic executor for VQA.

Our work is also related to learning interpretable and disentangled representations for visual scenes using neural networks. Kulkarni et al. [2015b] proposed convolutional inverse graphics networks for learning and inferring pose of faces, while Yang et al. [2015] learned disentangled representation of pose of chairs from images. Siddharth et al. [2017], Higgins et al. [2018] learned disentangled representations using deep generative models. In Chapter 4, we have also introduced the neural scene de-rendering

(a) Input Image → **(b) Object Segments** → **(c) Structural Scene Representation**

ID	Size	Shape	Material	Color	x	y	z
1	Small	Cube	Metal	Purple	-0.45	-1.10	0.35
2	Large	Cube	Metal	Green	3.83	-0.04	0.70
3	Large	Cube	Metal	Green	-3.20	0.63	0.70
4	Small	Cylinder	Rubber	Purple	0.75	1.31	0.35
5	Large	Cube	Metal	Green	1.58	-1.60	0.70

I. Scene Parsing (de-rendering)

(d) Question — **II. Question Parsing (Program Generation)** — **(e) Program** — **III. Program Execution**

How many cubes that are behind the cylinder are large?

```
1. filter_shape(scene, cylinder)
2. relate(behind)
3. filter_shape(scene, cube)
4. filter_size(scene, large)
5. count(scene)
```

```
1. filter_shape      3. filter_shape
2. relate            4. filter_size      5. count
```

ID	Size	Shape	...
1	Small	Cube	...
2	Large	Cube	...
3	Large	Cube	...
5	Large	Cube	...

ID	Size	...
2	Large	...
3	Large	...
5	Large	...

Answer: 3

Figure 8-2: **The NS-VQA model has three components**: first, a scene parser (de-renderer) that segments an input image (a-b) and recovers a structural scene representation (c); second, a question parser (program generator) that converts a question in natural language (d) into a program (e); third, a program executor that runs the program on the structural scene representation to obtain the answer.

framework as an inverse process of any rendering process. In this chapter, we propose an alternative representation learning approach through joint reasoning with language.

8.3 Neuro-Symbolic Visual Question Answering (NS-VQA)

We start by proposing a model named Neuro-Symbolic Visual Question Answering (NS-VQA), which explicitly brings symbolic scene representations and reasoning into scene understanding. As we will see, the use of symbolic structure improves the model's accuracy, data-efficiency, and interpretability, but also poses challenges when we want to apply it to real-world images.

The NS-VQA model has three components: a scene parser (de-renderer), a question parser (program generator), and a program executor. Given an image-question pair, the scene parser de-renders the image to obtain a structural scene representation (Figure 8-2-I), the question parser generates a hierarchical program from the question (Figure 8-2-II), and the executor runs the program on the structural representation to obtain an answer (Figure 8-2-III).

The scene parser recovers a structural and disentangled representation of the scene in the image (Figure 8-2a), based on which we can perform fully interpretable symbolic reasoning. The parser takes a two-step, segment-based approach for de-rendering: it first generates a number of segment proposals (Figure 8-2b), and for each segment, classifies the object and its attributes. The final, structural scene representation is disentangled, compact, and rich (Figure 8-2c).

The question parser maps an input question in natural language (Figure 8-2d) to a latent program (Figure 8-2e). The program has a hierarchy of functional modules,

each fulfilling an independent operation on the scene representation. Using a hierarchical program as our reasoning backbone naturally supplies compositionality and generalization power.

The program executor takes the output sequence from the question parser, applies these functional modules on the abstract scene representation of the input image, and generates the final answer (Figure 8-2-III). The executable program performs purely symbolic operations on its input throughout the entire execution process, and is fully deterministic, disentangled, and interpretable with respect to the program sequence.

8.3.1   Model Details

Scene parser.   For each image, we use Mask R-CNN [He et al., 2017] to generate segment proposals of all objects. Along with the segmentation mask, the network also predicts the categorical labels of discrete intrinsic attributes such as color, material, size, and shape. Proposals with bounding box score less than 0.9 are dropped. The segment for each single object is then paired with the original image, resized to 224 by 224 and sent to a ResNet-34 [He et al., 2016] to extract the spacial attributes such as pose and 3D coordinates. Here the inclusion of the original full image enables the use of contextual information.

Question parser.   Our question parser is an attention-based sequence to sequence (seq2seq) model with an encoder-decoder structure similar to that in Luong et al. [2015] and Bahdanau et al. [2015]. The encoder is a bidirectional LSTM [Hochreiter and Schmidhuber, 1997] that takes as input a question of variable lengths and outputs an encoded vector e_i at time step i as

$$e_i = [e_i^F, e_i^B], \quad \text{where} e_i^F, h_i^F = \text{LSTM}(\Phi_E(x_i), h_{i-1}^F), \quad e_i^B, h_i^B = \text{LSTM}(\Phi_E(x_i), h_{i+1}^B).$$
$$(8.1)$$

Here Φ_E is the jointly trained encoder word embedding. (e_i^F, h_i^F), (e_i^B, h_i^B) are the outputs and hidden vectors of the forward and backward networks at time step i. The decoder is a similar LSTM that generates a vector q_t from the previous token of the output sequence y_{t-1}. q_t is then fed to an attention layer to obtain a context vector c_t as a weighted sum of the encoded states via

$$q_t = \text{LSTM}(\Phi_D(y_{t-1})), \qquad \alpha_{ti} \propto \exp(q_t^\top W_A e_i), \qquad c_t = \sum_i \alpha_{ti} e_i. \qquad (8.2)$$

Φ_D is the decoder word embedding. For simplicity we set the dimensions of vectors q_t, e_i to be the same and let the attention weight matrix W_A to be an identity matrix. Finally, the context vector, together with the decoder output, is passed to a fully connected layer with softmax activation to obtain the distribution for the predicted token $y_t \sim \text{softmax}(W_O[q_t, c_t])$. Both the encoder and decoder have two hidden layers

with a 256-dim hidden vector. We set the dimensions of both the encoder and decoder word vectors to be 300.

Program executor. We implement the program executor as a collection of deterministic, generic functional modules in Python, designed to host all logic operations behind the questions in the dataset. Each functional module is in one-to-one correspondence with tokens from the input program sequence, which has the same representation as in Johnson et al. [2017b]. The modules share the same input/output interface, and therefore can be arranged in any length and order. A typical program sequence begins with a `scene` token, which signals the input of the original scene representation. Each functional module then sequentially executes on the output of the previous one. The last module outputs the final answer to the question. When type mismatch occurs between input and output across adjacent modules, an `error` flag is raised to the output, in which case the model will randomly sample an answer from all possible outputs of the final module.

8.3.2 Training Paradigm

Scene parsing. Our implementation of the object proposal network (Mask R-CNN) is based on "Detectron" [Girshick et al., 2018]. We use ResNet-50 FPN [Lin et al., 2017] as the backbone and train the model for 30,000 iterations with eight images per batch. Please refer to He et al. [2017] and Girshick et al. [2018] for more details. Our feature extraction network outputs the values of continuous attributes. We train the network on the *proposed* object segments computed from the training data using the mean square error as loss function for 30,000 iterations with learning rate 0.002 and batch size 50. Both networks of our scene parser are trained on 4,000 generated CLEVR images.

Reasoning. We adopt the following two-step procedure to train the question parser to learn the mapping from a question to a program. First, we select a small number of ground truth question-program pairs from the training set to pre-train the model with direct supervision. Then, we pair it with our deterministic program executor, and use REINFORCE [Williams, 1992] to fine-tune the parser on a larger set of question-answer pairs, using only the correctness of the execution result as the reward signal.

During supervised pre-training, we train with learning rate 7×10^{-4} for 20,000 iterations. For reinforce, we set the learning rate to be 10^{-5} and run at most 2M iterations with early stopping. The reward is maximized over a constant baseline with a decay weight 0.9 to reduce variance. Batch size is fixed to be 64 for both training stages. All our models are implemented in PyTorch.

Methods	Count	Exist	Compare Integer	Compare Attribute	Query Attribute	Overall
Humans [Johnson et al., 2017b]	86.7	96.6	86.4	96.0	95.0	92.6
CNN+LSTM+SAN [Johnson et al., 2017b]	59.7	77.9	75.1	70.8	80.9	73.2
N2NMN* [Hu et al., 2017]	68.5	85.7	84.9	88.7	90.0	83.7
Dependency Tree [Cao et al., 2018]	81.4	94.2	81.6	97.1	90.5	89.3
CNN+LSTM+RN [Santoro et al., 2017]	90.1	97.8	93.6	97.1	97.9	95.5
IEP* [Johnson et al., 2017b]	92.7	97.1	98.7	98.9	98.1	96.9
CNN+GRU+FiLM [Perez et al., 2018]	94.5	99.2	93.8	99.0	99.2	97.6
DDRprog* [Suarez et al., 2018]	96.5	98.8	98.4	99.0	99.1	98.3
MAC [Hudson and Manning, 2018]	97.1	99.5	99.1	99.5	99.5	98.9
TbD+reg+hres* [Mascharka et al., 2018]	97.6	99.2	99.4	99.6	99.5	99.1
NS-VQA (ours, 90 programs)	64.5	87.4	53.7	77.4	79.7	74.4
NS-VQA (ours, 180 programs)	85.0	92.9	83.4	90.6	92.2	89.5
NS-VQA (ours, 270 programs)	**99.7**	**99.9**	**99.9**	**99.8**	**99.8**	**99.8**

Table 8.2: **Quantitative results of NS-VQA on the CLEVR dataset.** Our model (NS-VQA) outperforms current state-of-the-art methods on CLEVR and achieves near-perfect question answering accuracy. The question-program pairs used for pretraining our model are uniformly drawn from the 90 question families of the dataset: 90, 180, 270 programs correspond to 1, 2, 3 samples from each family respectively. (*): trains on all program annotations (700K).

8.3.3 Data-Efficient, Interpretable Reasoning

Setup. We evaluate our NS-VQA on CLEVR [Johnson et al., 2017a]. The dataset includes synthetic images of 3D primitives with multiple attributes—shape, color, material, size, and 3D coordinates. Each image has a set of questions, each of which associates with a program (a set of symbolic modules) generated by machines based on 90 logic templates.

Our structural scene representation for a CLEVR image characterizes the objects in it, each labeled with its shape, size, color, material, and 3D coordinates (see Figure 8-2c). We evaluate our model's performance on the validation set under various supervise signal for training, including the numbers of ground-truth programs used for pre-training and question-answer pairs for REINFORCE. Results are compared with other state-of-the-art methods including the IEP baseline [Johnson et al., 2017b]. We not only assess the correctness of the answer obtained by our model, but also how well it recovers the underlying program. An interpretable model should be able to output the correct program in addition to the correct answer.

Results. Quantitative results on the CLEVR dataset are summarized in Table 8.2. Our NS-VQA achieves near-perfect accuracy and outperforms other methods on all five question types. We first pre-train the question parser on 270 annotated programs sampled across the 90 question templates (3 questions per template), a number below the weakly supervised limit suggested by Johnson et al. [2017b] (9K), and then run REINFORCE on all the question-answer pairs. Repeated experiments starting from

(a) Acc. vs. # pre-training programs   (b) Program acc. vs. # programs   (c) Acc. vs. # training data
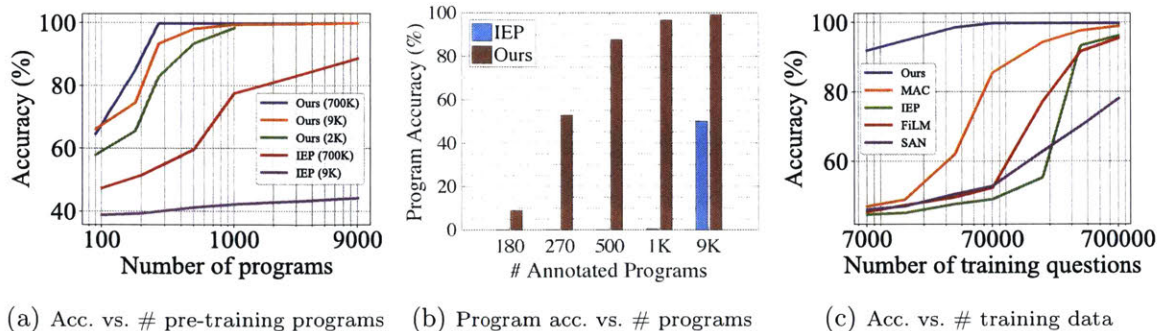
Figure 8-3: **Our NS-VQA model exhibits high data efficiency while achieving state-of-the-art performance and preserving interpretability.** (a) QA accuracy vs. number of programs used for pretraining; different curves indicate different numbers of question-answer pairs used in the REINFORCE stage. (c) QA accuracy vs. total number of training question-answer pairs; our model is pretrained on 270 programs.
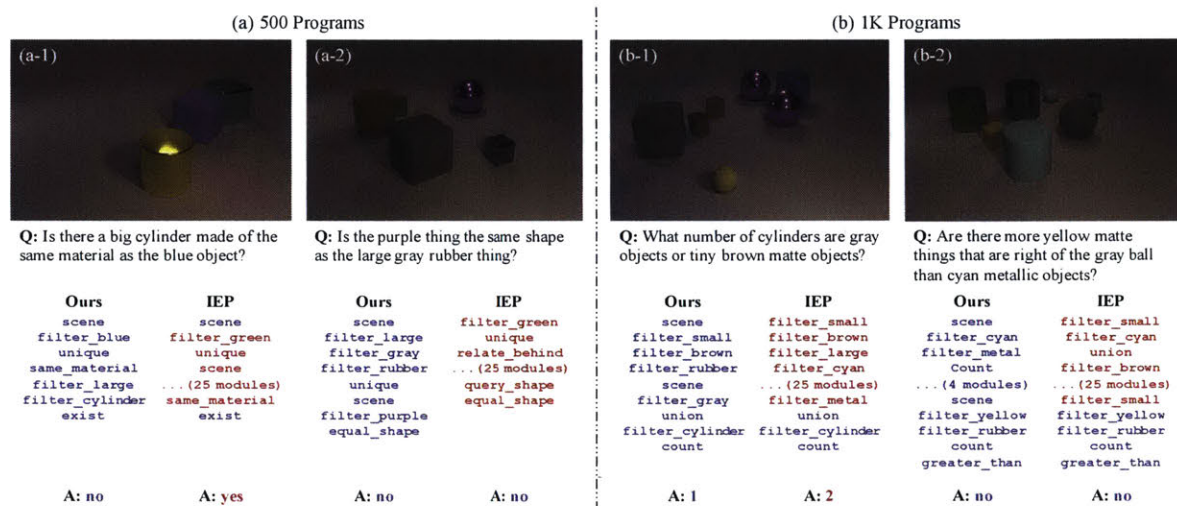
Figure 8-4: **Qualitative results of NS-VQA on CLEVR.** Blue color indicates correct program modules and answers; red indicates wrong ones. Our model is able to robustly recover the correct programs compared to the IEP baseline.

different sets of programs show a standard deviation of less than 0.1 percent on the results for 270 pre-training programs (and beyond). The variances are larger when we train our model with fewer programs (90 and 180). The reported numbers are the mean of three runs.

We further investigate the data-efficiency of our method with respect to both the number of programs used for pre-training and the overall question-answer pairs used in REINFORCE. Figure 8-3a shows the result when we vary the number of pre-training programs. NS-VQA outperforms the IEP baseline under various conditions, even with a weaker supervision during REINFORCE (2K and 9K question-answer pairs in REINFORCE). The number of question-answer pairs can be further reduced by pre-training the model on a larger set of annotated programs. For example, our model

169

achieves the same near-perfect accuracy of 99.8% with 9K question-answer pairs with annotated programs for both pre-training and REINFORCE.

Figure 8-3b compares how well our NS-VQA recovers the underlying programs compared to the IEP model. IEP starts to capture the true programs when trained with over 1K programs, and only recovers half of the programs with 9K programs. Qualitative examples in Figure 8-4 demonstrate that IEP tends to fake a long wrong program that leads to the correct answer. In contrast, our model achieves 88% program accuracy with 500 annotations, and performs almost perfectly on both question answering and program recovery with 9K programs.

Figure 8-3c shows the QA accuracy vs. the number of questions and answers used for training, where our NS-VQA has the highest performance under all conditions. Among the baseline methods we compare with, MAC [Hudson and Manning, 2018] obtains high accuracy with zero program annotations; in comparison, our method needs to be pre-trained on 270 program annotations, but requires fewer question-answer pairs to reach similar performance.

Our model also requires minimal memory for offline question answering: the structural representation of each image only occupies less than 100 bytes; in comparison, attention-based methods like IEP requires storing either the original image or its feature maps, taking at least 20K bytes per image.

8.4   Neuro-Symbolic Concept Learner

As discussed in the previous section, the NS-VQA model works well on synthetic scenes such as the CLEVR dataset [Johnson et al., 2017a]; however, as it requires full supervision on object attributes and semantic programs, NS-VQA faces challenges generalizing to natural images and language, where these annotations are prohibitively hard to obtain.

In this section, we present our neuro-symbolic concept learner (NS-CL), which uses a symbolic reasoning process to bridge the learning of visual concepts, words, and semantic parsing of sentences without explicit annotations for any of them. We first use a visual perception module to construct an object-based representation for a scene, and run a semantic parsing module to translate a question into an executable program. We then apply a quasi-symbolic program executor to infer the answer based on the scene representation. We use paired images, questions, and answers to jointly train the visual and language modules.

Shown in Figure 8-5, given an input image, the visual perception module detects objects in the scene and extracts a deep, latent representation for each of them. The semantic parsing module translates an input question in natural language into an executable program given a domain specific language (DSL). The generated programs
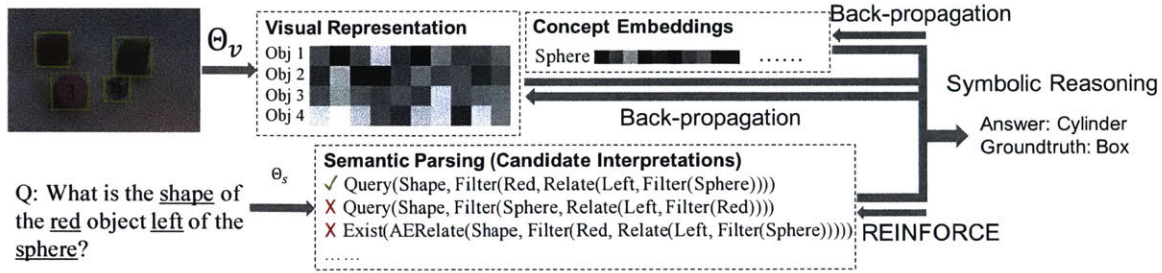
Figure 8-5: **We propose to use neuro-symbolic reasoning** as a bridge to jointly learn visual concepts, words, and semantic parsing of sentences.
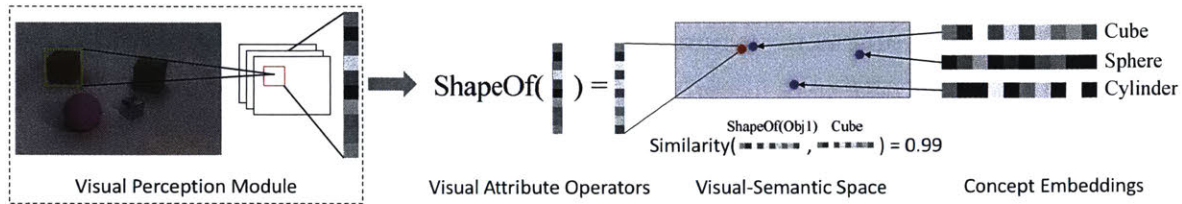
Figure 8-6: **We treat attributes such as Shape and Color as neural operators.** The operators map object representations into a visual-semantic space. We use similarity-based metric to classify objects.

have a hierarchical structure of symbolic, functional modules, each fulfilling a specific operation over the scene representation. The explicit program semantics enjoys compositionality, interpretability, and generalizability.

The program executor executes the program upon the derived scene representation and answers the question. Our program executor works in a symbolic and deterministic manner. This feature ensures a transparent execution trace of the program. Our program executor has a fully differentiable design w.r.t. the visual representations and the concept representations, which supports gradient-based optimization during training.

8.4.1 Model Details

Visual perception. Shown in Figure 8-5, given the input image, we use a pretrained Mask R-CNN [He et al., 2017] to generate object proposals for all objects. The bounding box for each single object paired with the original image is then sent to a ResNet-34 [He et al., 2016] to extract the region-based (by RoI Align) and image-based features respectively. We concatenate them to represent each object. Here, the inclusion of the representation of the full scene adds the contextual information, which is essential for the inference of relative attributes such as size or spatial position.

Concept quantization. Visual reasoning requires determining an object's attributes (e.g., its color or shape). We assume each visual attribute (e.g., shape) contains a set of visual concept (e.g., Cube). In NS-CL, visual attributes are implemented as

171

neural operators, mapping the object representation into an attribute-specific embedding space. Figure 8-6 shows an inference an object's shape. Visual concepts that belong to the shape attribute, including `Cube`, `Sphere` and `Cylinder`, are represented as vectors in the shape embedding space. These concept vectors are also learned along the process. We measure the cosine distances $\langle \cdot, \cdot \rangle$ between these vectors to determine the shape of the object. Specifically, we compute the probability that an object o_i is a cube by

$$\sigma \left(\langle \text{ShapeOf}(o_i), v^{\texttt{Cube}} \rangle - \gamma \right) / \tau, \tag{8.3}$$

where $\text{ShapeOf}(\cdot)$ denotes the neural operator, $v^{\texttt{Cube}}$ the concept embedding of `Cube` and σ the Sigmoid function. γ and τ are scalar constants for scaling and shifting the values of similarities. We classify relational concepts (e.g., `Left`) between a pair of objects similarly, except that we concatenate the visual representations for both objects to form the representation of their relation.

DSL and semantic parsing.   The semantic parsing module translates a natural language question into an executable program with a hierarchy of primitive operations, represented in a domain-specific language (DSL) designed for VQA. The DSL covers a set of fundamental operations for visual reasoning, such as filtering out objects with certain concepts or query the attribute of an object. The operations share the same input and output interface, and thus can be compositionally combined to form programs of any complexity. We include a complete specification of the DSL used by our framework in the Appendix B.1.

Our semantic parser generates the hierarchies of latent programs in a sequence to tree manner [Dong and Lapata, 2016]. We use a bidirectional GRU [Cho et al., 2014] to encode an input question, which outputs a fixed-length embedding of the question. A decoder based on GRU cells is applied to the embedding, and recovers the hierarchy of operations as the latent program. Some operations takes concepts their parameters, such as `Filter(` Red `)` and `Query(` Shape `)`. These concepts are chosen from all concepts appeared in the input question. Figure 8-7B shows an example, while more details can be found in Appendix B.2.

Quasi-symbolic program execution.   Given the latent program recovered from the question in natural language, a symbolic program executor executes the program and derives the answer based on the object-based visual representation. Our program executor is a collection of deterministic functional modules designed to realize all logic operations specified in the DSL. Figure 8-7B shows an illustrative execution trace of a program.

To make the execution differentiable w.r.t. visual representations, we represent the intermediate results in a probabilistic manner: a set of objects is represented by a vector, as the attention mask over all objects in the scene. Each element,

A. Curriculum concept learning

○ **Initialized** with DSL and executor.

□ **Lesson1**: Object-based questions.

Q: What is the <u>shape</u> of the <u>red</u> object?
A: Cube.

□ **Lesson2**: Relational questions.

Q: How many <u>cubes</u> are <u>behind</u> the <u>sphere</u>?
A: 3

□ **Lesson3**: More complex questions.

Q: Does the red object left of the green cube have the same shape as the purple matte thing?
A: No

□ **Deploy**: complex scenes, complex questions

Q: Does the matte thing behind the big sphere have the same color as the cylinder left of the small matte cube?
A: No.

B. Illustrative execution of NS-CL

Q: Does the <u>red</u> object <u>left</u> of the <u>green cube</u> have the same <u>shape</u> as the <u>purple matte</u> thing?

Step1: Visual Parsing

Step2, 3: Semantic Parsing and Program Execution

○ Program	Representations	Concepts	Outputs
Filter		Green Cube	
Relate	Object 2	Left	
Filter		Red	
Filter		Purple Matte	
AEQuery	Object 1 Object 3	Shape	No (0.98)
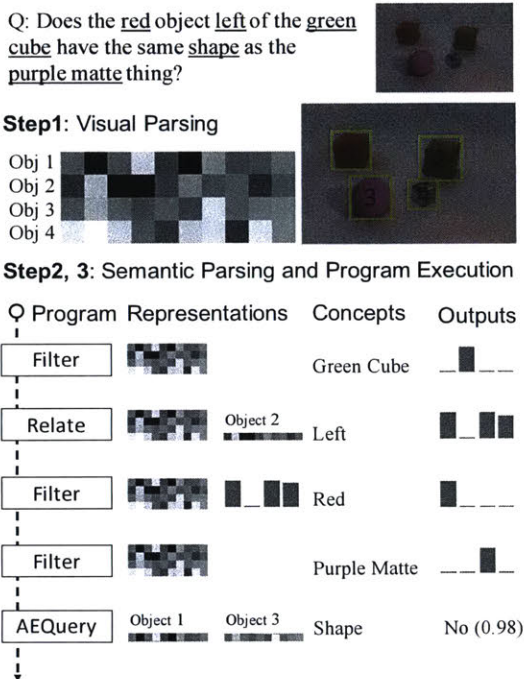
Figure 8-7: **A. Demonstration of the curriculum learning** of visual concepts, words, and semantic parsing of sentences by watching images and reading paired questions and answers. Scenes and questions of different complexities are illustrated to the learner in an incremental manner. **B. Illustration of our neuro-symbolic inference model for VQA.** The perception module begins with parsing visual scenes into object-based deep representations, while the semantic parser parse sentences into executable programs. A symbolic execution process bridges two modules.

$Mask_i \in [0, 1]$ denotes the probability that the i-th object of the scene belongs to the set. For example, shown in Figure 8-7B, the first `Filter` operation outputs a mask of length 4 (there are in total 4 objects in the scene), with each element representing the probability that the corresponding object is selected out (i.e., the probability that each object is a green cube). The output "mask" on the objects will be fed into the next module (`Relate` in this case) as input and the execution of programs continues. The last module outputs the final answer to the question. We refer interested readers to Appendix B.3 for the implementation of all operators.

8.4.2 Training Paradigm

Optimization objective. The optimization objective of NS-CL is composed of two parts: concept learning and language understanding. Our goal is to find the optimal parameters Θ_v of the visual perception module Perception (including the ResNet-34 for extracting object features, attribute operators. and concept embeddings) and Θ_s of the semantic parsing module SemanticParse, to maximize the likelihood of answering

the question Q correctly:

$$\Theta_v, \Theta_s \leftarrow \arg \max_{\Theta_v, \Theta_s} \mathbb{E}_P \left[\Pr[A = \text{Executor}(\text{Perception}(S; \Theta_v), P)] \right], \qquad (8.4)$$

where P denotes the program, A the answer, S the scene, and Executor the quasi-symbolic executor. The expectation is taken over $P \sim \text{SemanticParse}(Q; \Theta_s)$.

Recall the program executor is fully differentiable w.r.t. the visual representation. We compute the gradient w.r.t. Θ_v as

$$\nabla_{\Theta_v} \mathbb{E}_P \left[D_{\text{KL}}(\text{Executor}(\text{Perception}(S; \Theta_v), P) \| A) \right]. \qquad (8.5)$$

We use REINFORCE [Williams, 1992] to optimize the semantic parser Θ_s:

$$\nabla_{\Theta_s} = \mathbb{E}_P \left[r \cdot \log \Pr[P = \text{SemanticParse}(Q; \Theta_s)] \right], \qquad (8.6)$$

where the reward $r = 1$ if the answer is correct and 0 otherwise. We also use off-policy search to reduce the variance of REINFORCE, the detail of which can be found in Appendix B.4.

Curriculum visual concept learning. Motivated by human concept learning as in Figure 8-1, we employ a curriculum learning approach to help joint optimization. We heuristically split the training samples into four stages (Figure 8-7A): first, learning object-level visual concepts; second, learning relational questions; third, learning more complex questions with perception modules fixed; fourth, joint fine-tuning of all modules. We found that this is essential to the learning of our neuro-symbolic concept learner. We include more technical details in Appendix B.5.

8.5 Experiments

We demonstrate the following advantages of our NS-CL. First, it learns visual concepts with remarkable accuracy; second, it allows data-efficient visual reasoning on the CLEVR dataset [Johnson et al., 2017a]; third, it generalizes well to new attributes, visual composition, and language domains.

We train NS-CL on 5K images (<10% of CLEVR's 70K training images). We generate 20 questions for each image for the entire curriculum learning process. The Mask R-CNN module is pre-trained on 4K generated CLEVR images with bounding box annotations, following Yi et al. [2018].

8.5.1 Visual Concept Learning

Classification-based concept evaluation. Our model treats attributes as neural operators that map latent object representations into an attribute-specific embedding

	Visual Features	Mean	Color	Material	Shape	Size
IEP	Convolutional	90.6	91.0	90.0	89.9	90.6
MAC	Attentional	95.9	98.0	91.4	94.4	94.2
TbD (high-res.)	Attentional	96.5	96.6	92.2	95.4	92.6
NS-CL	Object-Based	**98.7**	**99.0**	**98.7**	**98.1**	**99.1**

Table 8.3: **We evaluate the learned visual concepts using a diagnostic question set** containing simple questions such as "How many red objects are there?". NS-CL outperforms both convolutional and attentional baselines. The suggested object-based visual representation and symbolic reasoning approach perceives better interpretation of visual concepts.

space (Figure 8-6). We evaluate the concept quantization of objects in the CLEVR validation split. Our model can achieve near perfect classification accuracy (~99%) for all object properties, suggesting it effectively learns generic concept representations. The result for spatial relations is relatively lower, because CLEVR does not have direct queries on the spatial relation between objects. Thus, spatial relation concepts can only be learned indirectly.

Count-based concept evaluation. The SOTA methods do not provide interpretable representation on individual objects [Johnson et al., 2017a, Hudson and Manning, 2018, Mascharka et al., 2018]. To evaluate the visual concepts learned by such models, we generate a synthetic question set. The diagnostic question set contains simple questions as the following form: "How many red objects are there?". We evaluate the performance on all concepts appeared in the CLEVR dataset.

Table 8.3 summarizes the results compared with strong baselines, including methods based on convolutional features [Johnson et al., 2017b] and those based on neural attentions [Mascharka et al., 2018, Hudson and Manning, 2018]. Our approach outperforms IEP by a significant margin (8%) and attention-based baselines by >2%, suggesting object-based visual representations and symbolic reasoning helps to interpret visual concepts.

8.5.2 Data-Efficient and Interpretable Visual Reasoning

NS-CL jointly learns visual concepts, words and semantic parsing by watching images and reading paired questions and answers. It can be directly applied to VQA.

Table 8.4 summarizes results on the CLEVR validation split. Our model achieves the state-of-the-art performance among all baselines using zero program annotations, including MAC [Hudson and Manning, 2018] and FiLM [Perez et al., 2018]. Our model achieves comparable performance with the strong baseline TbD-Nets [Mascharka et al., 2018], whose semantic parser is trained using 700K programs in CLEVR (ours need 0). The NS-VQA model we just introduced achieves better performance on CLEVR;

Model	Program Annotations	Overall	Count	Compare Integer	Exist	Query Attribute	Compare Attribute
Human	N/A	92.6	86.7	86.4	96.6	95.0	96.0
NMN	700K	72.1	52.5	72.7	79.3	79.0	78.0
N2NMN	700K	88.8	68.5	84.9	85.7	90.0	88.8
IEP	700K	96.9	92.7	98.7	97.1	98.1	98.9
DDRprog	700K	98.3	96.5	98.4	98.8	99.1	99.0
TbD	700K	**99.1**	**97.6**	**99.4**	**99.2**	**99.5**	**99.6**
RN	0	95.5	90.1	93.6	97.8	97.1	97.9
FiLM	0	97.6	94.5	93.8	99.2	99.2	99.0
MAC	0	98.9	97.2	**99.4**	**99.5**	99.3	**99.5**
NS-CL	0	**98.9**	**98.2**	99.0	98.8	**99.3**	99.1

Table 8.4: **Accuracy on visual question answering.** Our model outperforms all baselines using no program annotations. It achieves comparable results with models trained by full program annotations such as TbD.

however, it requires annotated visual attributes and program traces during training, while our NS-CL needs no extra labels.

Here, the visual perception module is pre-trained on ImageNet [Deng et al., 2009]. Without pre-training, the concept learning accuracies drop by 0.2% on average and the QA accuracy drops by 0.5%. Meanwhile, NS-CL recovers the underlying programs of questions accurately (>99.9% accuracy). NS-CL can also detect ambiguous or invalid programs and indicate exceptions. Please see Appendix B.6 for more details. NS-CL can also be applied to other visual reasoning testbeds. Please refer to Appendix B.7.1 for our results on the Minecraft dataset [Yi et al., 2018].

For a systematic study on visual features and data efficiency, we implement two variants of the baseline models: TbD-Object and MAC-Object. Inspired by Anderson et al. [2018], instead of the input image, TbD-Object and MAC-Object take a stack of object features as input. TbD-Mask and MAC-Mask integrate the masks of objects by using them to guide the attention over the images.

Table 8.5 summarizes the results. Our model outperforms all baselines on data efficiency. This comes from the full disentanglement of visual concept learning and symbolic reasoning: how to execute program instructions based on the learned concepts is programmed. TbD-Object and MAC-Object demonstrate inferior results in our experiments. We attribute this to the design of model architectures and have a detailed analysis in Appendix B.6.3. Although TbD-Mask and MAC-Mask do not perform better than the originals, we find that using masks to guide attentions speeds up the training.

NS-CL performs well on visual reasoning by leveraging both object-based representation and symbolic reasoning; it also learns fully interpretable visual concepts.

Model	Visual Features	Accuracy (100% Data)	Accuracy (10% Data)
TbD	Attentional	99.1	54.2
TbD-Object	Object-Based	84.1	52.6
TbD-Mask	Attentional	99.0	55.0
MAC	Attentional	98.9	67.3
MAC-Object	Object-Based	79.5	51.2
MAC-Mask	Attentional	98.7	68.4
NS-CL	Object-Based	**99.2**	**98.9**

Table 8.5: **We compare with baselines using various visual features on data efficiency.** Using only 10% of the training images, our model is able to achieve a comparable results with the baselines trained on the full dataset.

Please see Appendix B.8 for qualitative results on various datasets.

8.5.3 Generalizing to New Attributes and Compositions

Generalizing to new visual compositions. The CLEVR-CoGenT dataset is designed to evaluate models' ability to generalize to new visual compositions. It has two splits: Split A only contains gray, blue, brown and yellow cubes, but red, green, purple, and cyan cylinders; split B imposes the opposite color constraints on cubes and cylinders. If we directly learn visual concepts on split A, it overfits to classify shapes based on the color, leading to a poor generalization to split B.

Our solution is based on the idea of seeing attributes as operators. Specifically, we jointly train the concept embeddings (e.g., Red, Cube, etc.) as well as the semantic parser on split A, keeping pre-trained, frozen attribute operators. As we learn distinct representation spaces for different attributes, our model achieves an accuracy of 98.8% on split A and 98.9% on split B.

Generalizing to new visual concepts. We expect the process of concept learning can take place in an incremental manner: having learned 7 different colors, humans can learn the 8-th color incrementally and efficiently. To this end, we build a synthetic split of the CLEVR dataset to replicate the setting of incremental concept learning. Split A contains only images without any purple objects, while split B contains images with at least one purple object. We train all the models on split A first, and finetune them on 100 images from split B. We report the final QA performance on split B's validation set. All models use a pre-trained semantic parser on the full CLEVR dataset.

Our model performs a 93.9% accuracy on the QA test in Split B, outperforming the convolutional baseline IEP [Johnson et al., 2017b] and the attentional baseline TbD [Mascharka et al., 2018] by 4.6% and 6.1% respectively. The acquisition of Color operator brings more efficient learning of new visual concepts.

	Split A	Split B	Split C	Split D

Q: What's the shape of the big yellow thing?

Q: What size is the cylinder that is left of the cyan thing that is in front of the big sphere?

Q: What's the shape of the big yellow thing?

Q: What size is the cylinder that is left of the cyan thing that is in front of the gray cube?
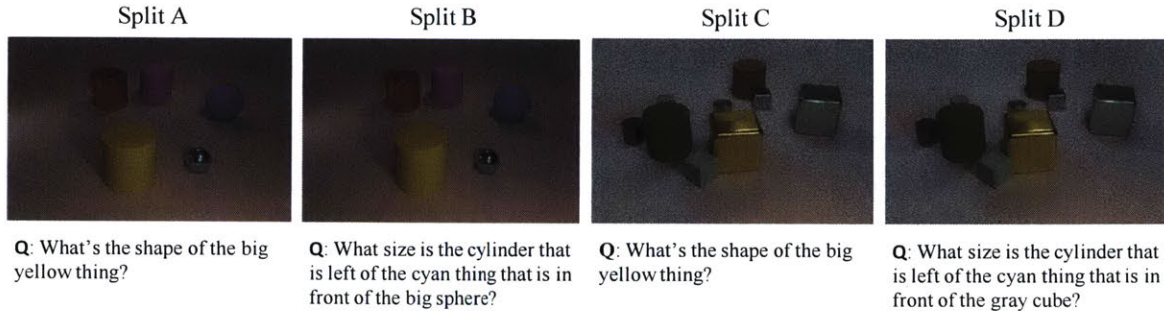
Figure 8-8: **Samples collected from four splits** in Section 8.5.3. Models are trained on split A but evaluated on all splits for testing the combinatorial generalization.

Model	Test			
	Split A	Split B	Split C	Split D
MAC	97.3	N/A	92.9	N/A
IEP	96.1	92.1	91.5	90.9
TbD	98.8	94.5	94.3	91.9
NS-CL	**98.9**	**98.9**	**98.7**	**98.8**

Table 8.6: **We test combinatorial generalizations** w.r.t. the number of objects in scenes and the complexity of questions (i.e., the depth of the program trees). We makes four split of the data containing various complexities of scenes and questions. Our object-based visual representation and explicit program semantics enjoys the best (and almost-perfect) combinatorial generalization compared with strong baselines.

8.5.4 Generalizing to New Scenes and Questions

Having learned visual concepts on small-scale scenes (containing only few objects) and simple questions (only single-hop questions), we humans can easily generalize the knowledge to larger-scale scenes and to answer complex questions. To evaluate this, we split the CLEVR dataset into four parts: **Split A** contains only scenes with less than 6 objects, and questions whose latent programs having a depth less than 5; **Split B** contains scenes with less than 6 objects, but arbitrary questions; **Split C** contains arbitrary scenes, but restricts the program depth being less than 5; **Split D** contains arbitrary scenes and questions. Figure 8-8 shows some illustrative samples.

As VQA baselines are unable to count a set of objects of arbitrary size, for a fair comparison, all programs containing the "count" operation over > 6 objects are removed from the set. For methods using explicit program semantics, the semantic parser is pre-trained on the full dataset and fixed. Methods with implicit program semantics [Hudson and Manning, 2018] learn an entangled representation for perception and reasoning, and cannot trivially generalize to more complex programs. We only use the training data from the Split A and then quantify the generalization ability on other three splits. Shown in Table 8.6, our NS-CL leads to almost-perfect generalization to

Caption: There is a big yellow cylinder in front of a gray object.

(a) An illustrative pair of image and caption in our synthetic dataset.

Model	Accuracy
IEP	95.5
TbD	**97.0**
NS-CL	96.9

(b) Image-caption retrieval accuracy on a subset of data. Our model archives comparable results with VQA baselines.

Model	Accuracy
CNN-LSTM	68.9
NS-CL	**97.0**

(c) Image-caption retrieval accuracy on the full dataset. Our model outperforms baselines and requires no extra training or fine-tuning of the visual perception module.

Table 8.7: **Results on image-caption retrieval.** We introduce a new simple DSL for image-caption retrieval to evaluate how well the learned visual concepts transfer. Due to the difference between VQA and caption retrieval, VQA baselines are only able to infer the result on a partial set of data. The learned object-based visual concepts can be directly transferred into the new domain for free.

larger scenes and more complex questions, outperforming all baselines by at least 4% in QA accuracy.

8.5.5 Generalizing to a New Program Domain

The learned visual concepts can also be used in other domains such as image retrieval. With the visual scenes fixed, the learned visual concepts can be directly transferred into the new domain. We only need to learn the semantic parsing of natural language into the new DSL.

We build a synthetic dataset for image retrieval and adopt a DSL from scene graph–based image retrieval [Johnson et al., 2015]. The dataset contains only simple captions: "There is an <object A> <relation> <object B>." (e.g., There is a box right of a cylinder). The semantic parser learns to extract corresponding visual concepts (e.g., `box`, `right`, and `cylinder`) from the sentence. The program can then be executed on the visual representation to determine if the visual scene contains such relational triples.

For simplicity, we treat retrieval as classifying whether a relational triple exists in the image. This functionality cannot be directly implemented on the CLEVR VQA program domain, because questions such as "Is there a box right of a cylinder" can be ambiguous if there exist multiple cylinders in the scene. Due to the entanglement of the visual representation with the specific DSL, baselines trained on CLEVR QA can not be directly applied to this task. For a fair comparison with them, we show the result in Table 8.7b on a subset of the generated image-caption pairs where the underlying programs have no ambiguity regarding the reference of object B. A separate semantic parser is trained for the VQA baselines, which translates captions into a CLEVR QA-compatible program (e.g., `Exist(Filter(Box, Relate(Right, Filter(Cylinder)))`).
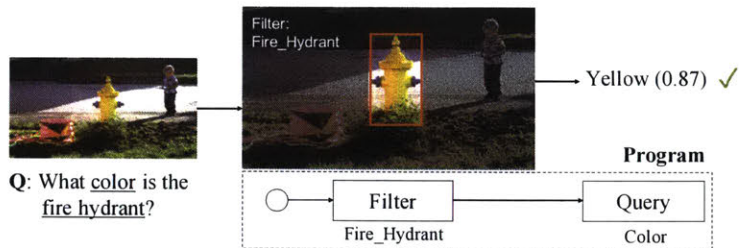
Figure 8-9: **Demo from the VQS dataset.** An example image-question pair from the VQS dataset and the corresponding execution trace of NS-CL.

Model	Accuracy
MLP	43.9
MAC	**46.2**
NS-CL	44.3

Table 8.8: **Results on the VQS test set.** Quantitatively, our model achieves a comparable results with the baselines.
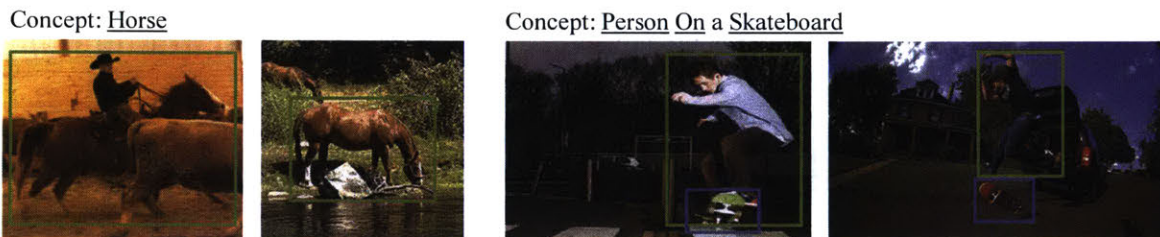
Figure 8-10: **Concepts learned from the VQS dataset**, including object categories, attributes, and relations

Table 8.7c compares our NS-CL against typical image-text retrieval baselines on the full image-caption dataset. Without any annotations of the sentence semantics, our model learns to parse the captions into the programs in the new DSL. It outperforms the CNN-LSTM baseline by 30%.

8.5.6 Generalizing to Natural Images and Language

We further conduct experiments on MS-COCO [Lin et al., 2014] images. Results are presented on the VQS dataset [Gan et al., 2017]. VQS contains a subset of images and questions from the original VQA 1.0 dataset [Antol et al., 2015]. All questions in the VQS dataset can be visually grounded: each question is associated with multiple image regions, annotated by humans as essential for answering the question. Figure 8-9 illustrates an execution trace of NS-CL on VQS.

We use a syntactic dependency parser to extract programs and concepts from language [Andreas et al., 2016, Schuster et al., 2015]. The object proposals and features are extracted from models pre-trained on the MS-COCO dataset and the ImageNet dataset, respectively. Illustrated in Figure 8-9, our model shows competitive performance on QA accuracy, comparable with the MLP baseline [Jabri et al., 2016] and the MAC network [Hudson and Manning, 2018]. Additional illustrative execution traces of NS-CL are in Appendix B.8. Beyond answering questions, NS-CL effectively learns visual concepts from data. Figure 8-10 shows examples of the learned visual concepts, including object categories, attributes, and relations. Experiment setup and

implementation details are in Appendix B.7.2.

We focus on a neuro-symbolic framework that learns visual concepts about object properties and relations. Indeed, visual question answering requires AI systems to reason about more general concepts such as events or activities [Levin, 1993]. We leave the extension of NS-CL along this direction and its application to general VQA datasets [Antol et al., 2015] as future work.

8.6 Discussion

In this chapter, we have presented a method that jointly learns visual concepts, words, and semantic parsing of sentences from natural supervision. The proposed framework, NS-CL, learns by looking at images and reading paired questions and answers, without any explicit supervision such as class labels for objects. Our model learns visual concepts with remarkable accuracy. Based upon the learned concepts, our model achieves good results on question answering, and more importantly, generalizes well to new visual compositions, new visual concepts, and new domain specific languages.

The design of NS-CL suggests multiple research directions. First, constructing 3D object-based representations for realistic scenes needs further exploration [Anderson et al., 2018, Baradel et al., 2018]. Second, our model assumes a domain-specific language for describing formal semantics. The integration of formal semantics into the processing of complex natural language would be meaningful future work [Artzi and Zettlemoyer, 2013, Oh et al., 2017]. We hope our model could motivate future research in visual concept learning, language learning, and compositionality.

Our framework can also be extended to other domains such as video understanding and robotic manipulation. Here, we would need to discover semantic representations for actions and interactions (e.g., push) beyond static spatial relations. Along this direction, researchers have studied building symbolic representations for skills [Konidaris et al., 2018] and learning instruction semantics from interaction [Oh et al., 2017] in constrained setups. Applying neuro-symbolic learning frameworks for concepts and skills would be meaningful future work toward robotic learning in complex interactive environments.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 9

Learning to Organize Concepts into 3D Shape Programs

In Chapter 8, we have seen a model that discovers entry-level visual concepts (e.g., geometric primitives) with minimal supervision. But human perception of 3D shapes goes much beyond reconstructing them as a set of points or a composition of geometric primitives: we also effortlessly understand higher-level shape structure such as the repetition and reflective symmetry of object parts. In contrast, recent advances in 3D shape sensing focus more on low-level geometry but less on these higher-level relationships.

In this chapter, we propose *3D shape programs*, integrating bottom-up recognition systems with top-down, symbolic program structure to capture both low-level geometry and high-level structural priors for 3D shapes. Because there are no annotations of shape programs for real shapes, we develop neural modules that not only learn to infer 3D shape programs from raw, unannotated shapes, but also to execute these programs for shape reconstruction. After initial bootstrapping, our end-to-end differentiable model learns 3D shape programs by reconstructing shapes in a self-supervised manner. Experiments demonstrate that our model accurately infers and executes 3D shape programs for highly complex shapes from various categories. It can also be integrated with an image-to-shape module to infer 3D shape programs directly from an RGB image, leading to 3D shape reconstructions that are both more accurate and more physically plausible.

This chapter includes materials previously published as Tian et al. [2019]. Yonglong Tian contributed significantly to the materials presented in this chapter.

9.1   Introduction

Given the table in Figure 9-1, humans are able to instantly recognize its parts and regularities: there exist sharp edges, smooth surfaces, a table top that is a perfect circle, and two lower, square layers. Beyond these basic components, we also perceive

```
Draw("Top", "Circle", position, geometry)

for(i < 2, "translation", a)
  for(j < 2, "translation", b)
    Draw("Leg", "Cub", position + i*a + j*b, geometry)

for(i < 2, "translation", c)
  Draw("Layer", "Rec", position + i*c, geometry)
```
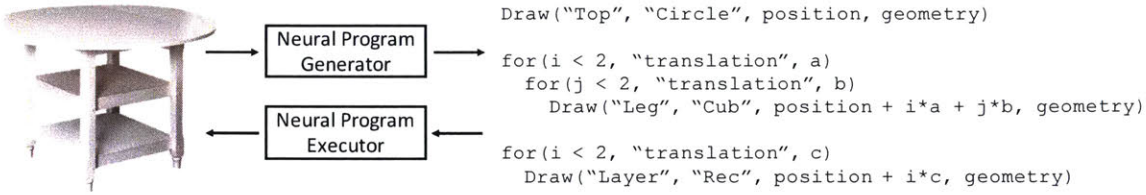
Figure 9-1: **A 3D shape can be represented by a program** via a program generator. This program can be executed by a neural program executor to produce the corresponding 3D shape.

higher-level, abstract concepts: the shape is bilateral symmetric; the legs are all of equal length and laid out on the opposite positions of a 2D grid. Knowledge like this is crucial for visual recognition and reasoning [Koffka, 2013, Dilks et al., 2011].

Recent AI systems for 3D shape understanding have made impressive progress on shape classification, parsing, reconstruction, and completion [Qi et al., 2017, Tulsiani et al., 2017a], many making use of large shape repositories like ShapeNet [Chang et al., 2015]. Popular shape representations include voxels [Wu et al., 2015b], point clouds [Qi et al., 2017], and meshes [Wang et al., 2018a]. While each has its own advantages, these methods fall short on capturing the strong shape priors we just described, such as sharp edges and smooth surfaces.

A few recent papers have studied modeling 3D shapes as a collection of primitives [Tulsiani et al., 2017a], with simple operations such as addition and subtraction [Sharma et al., 2018]. These representations have demonstrated success in explaining complex 3D shapes. In this chapter, we go beyond them to capture the high-level regularity within a 3D shape, such as symmetry and repetition.

Our key contribution is to represent 3D shapes as *shape programs*. We define a domain-specific language (DSL) for shapes, containing both basic shape primitives for parts with their geometric and semantic attributes, as well as statements such as loops to enforce higher-level structural priors.

Because 3D shape programs are a new shape representation, there exist no annotations of shape programs for 3D shapes. The lack of annotations makes it difficult to train an inference model with full supervision. To overcome this obstacle, we propose to learn a shape program executor that reconstructs a 3D shape from a shape program. After initial bootstrapping, our model can then learn in a self-supervised way, by attempting to explain and reconstruct unlabeled 3D shapes with 3D shape programs. This design minimizes the amount of supervision needed to get our model off the ground.

With the learned neural program executor, our model learns to explain input shapes without ground truth program annotations. Experiments on ShapeNet show that our model infers accurate 3D shape programs for highly complex shapes from

various categories. We further extend our model by integrating with an image-to-shape reconstruction module, so it directly infers a 3D shape program from a color image. This leads to 3D shape reconstructions that are both more accurate and more physically plausible.

Our contributions are three-fold. First, we propose 3D shape programs: a new representation for shapes, building on classic findings in cognitive science and computer graphics. Second, we propose to infer 3D shape programs by explaining the input shape, making use of a neural shape program executor. Third, we demonstrate that the inference model, the executor, and the programs they recover all achieve good performance on ShapeNet, learning to explain and reconstruct complex shapes. We further show that an extension of the model can infer shape programs and reconstruct 3D shapes directly from images.

9.2 Related Work

Inverse procedural graphics. The problem of inferring programs from voxels is closely related to inverse procedural graphics, where a procedural graphics program is inferred from an image or declarative specification [Ritchie et al., 2016, Šťava et al., 2010]. Where the systems have been most successful, however, are when they leverage a large shape/component library [Chaudhuri et al., 2011, Schulz et al., 2017] or when they are applied to a sparse solution space [van den Hengel et al., 2015]. Kulkarni et al. [2015a] approached the problem of inverse graphics as inference in a probabilistic program for generating 2D images, or image contours, from an underlying 3D model. They demonstrated results on several different applications using parametric generative models for faces, bodies, and simple multi-part objects based on generalized cylinders. In this chapter, we extend the idea of inverse procedural graphics to 3D voxel representations, and show how this idea can apply to large data sets like ShapeNet. We furthermore do not have to match components to a library of possible shapes, instead using a neural network to directly infer shapes and their parameters.

A few recent papers have explored the use of simple geometric primitives to describe shapes [Tulsiani et al., 2017a, Zou et al., 2017, Liu et al., 2018b], putting the classic idea of generalized cylinders [Roberts, 1963, Binford, 1971] or geons [Biederman, 1987] in the modern context of deep learning. In particular, Sharma et al. [2018] extended these papers and addressed the problem of inferring 3D CAD programs from perceptual input. We find this work inspiring, but also feel that a key goal of 3D program inference is to reconstruct a program in terms of semantically meaningful parts and their spatial regularity, which we address here. Some other graphics papers also explore regularity, but without using programs [Mitra et al., 2013, Zhu et al.,

2018a, Nishida et al., 2018, Li et al., 2017a].

Work in the HCI community has also addressed the problem of inferring parametric graphics primitives from perceptual input. For example, Nishida et al. [2016] proposed to learn to instantiate procedural primitives for an interactive modeling system. In our work, we instead learn to instantiate multiple procedural graphics primitives simultaneously, without assistance from a human user.

Program synthesis. In the AI literature, Ellis et al. [2018] leveraged symbolic program synthesis techniques to infer 2D graphics programs from images, extending their earlier work by using neural nets for faster inference of low-level cues such as strokes [Ellis et al., 2015]. Here, we show how a purely *end-to-end* network can recover 3D graphics programs from voxels, conceptually relevant to RobustFill [Devlin et al., 2017], which presents a purely end-to-end neural program synthesizer for text editing. The very recent SPIRAL system [Ganin et al., 2018] also takes as its goal to learn structured program–like models from (2D) images. An important distinction from our work here is that SPIRAL explains an image in terms of paint-like "brush strokes", whereas we explain 3D voxels in terms of high-level objects and semantically meaningful parts of objects, like legs or tops. Other tangential related work on program synthesis includes Balog et al. [2017], Devlin et al. [2017], Parisotto et al. [2017], Gaunt et al. [2016], Sun et al. [2018a], Liu et al. [2019].

Learning to execute programs. Neural Program Interpreters (NPI) have been extensively studied for programs that abstract and execute tasks such as sorting, shape manipulation, and grade-school arithmetic [Reed and De Freitas, 2016, Cai et al., 2017, Bošnjak et al., 2017]. In NPI [Reed and De Freitas, 2016], the key insight is that a program execution trace can be decomposed into pre-defined operations that are more primitive; and at each step, an NPI learns to predict what operation to take next depending on the general environment, domain specific state , and previous actions. Cai et al. [2017] improved the generalization of NPIs by adding recursion. Johnson et al. [2017b] learned to execute programs for visual question and answering. In comparison, our model also learns a 3D shape program executor that renders 3D shapes from programs as a component of our model.

9.3 3D Shape Programs

In this section, we define the domain-specific language for 3D shapes, as well as the problem of shape program synthesis.

Table 9.1 shows our DSL for 3D shape programs. Each shape program consists of a variable number of program statements. A program statement can be either `Draw`, which describes a shape primitive as well as its geometric and semantic attributes, or `For`, which contains a sub-program and parameters specifying how the subprogram

186

Program	→	Statement; Program
Statement	→	Draw(Semantics, Shape, Position_Params, Geometry_Params)
Statement	→	For(For_Params); Program; EndFor
Semantics	→	semantics 1 | semantics 2 | semantics 3 | ...
Shape	→	Cuboid | Cylinder | Rectangle | Circle | Line | ...
Position_Params	→	(x, y, z)
Geometry_Params	→	$(g_1, g_2, g_3, g_4, ...)$
For_Params	→	Translation_Params | Rotation_Params
Translation_Params	→	(times i, orientation u)
Rotation_Params	→	(times i, angle θ, axis a)

Table 9.1: **The domain specific language (DSL) for 3D shapes.** Semantics depends on the types of objects that are modeled, i.e., semantics for *vehicle* and *furniture* should be different. For details of DSL in our experimental setting, please refer to Appendix C.

should be repeatedly executed. The number of arguments for each program statement varies. We tokenize programs for the purpose of neural network prediction.

Each shape primitive models a semantically-meaningful part of an object. Its geometric attributes (Table 9.1: Geometry_Params, Position_Params) specify the position and orientation of the part. Its semantic attributes (Table 9.1: Semantics) specify its relative role within the whole shape (e.g., top, back, leg). They do not affect the geometry of the primitive; instead, they associate geometric parts with their semantic meanings conveying how parts can be shared across object categories semantically and functionally (e.g., a chair and a table may have similar legs).

Our For statement captures high-level regularity across parts. For example, the legs of a table can be symmetric with respect to particular rotation angles. The horizontal bars of a chair may lay out regularly with a fixed vertical gap. Each For statement can contain sub-programs, allowing recursive generation of shape programs.

The problem of inferring a 3D shape program is defined as follows: predicting a 3D shape program that reconstructs the input shape when the program is executed. We use voxelized shapes as input with a resolution of $32 \times 32 \times 32$.

9.4   Inferring and Executing 3D Shape Programs

Our model, called *Shape Programs*, consists of a program generator and a neural program executor. The program generator takes a 3D shape as input and outputs a sequence of primitive programs that describe this 3D shape. The neural program executor takes these programs as input and generates the corresponding 3D shapes. This allows our model to learn in a self-supervised way by generating programs from input shapes, executing these programs, and back-propagating the difference between the generated shapes and the raw input.

187
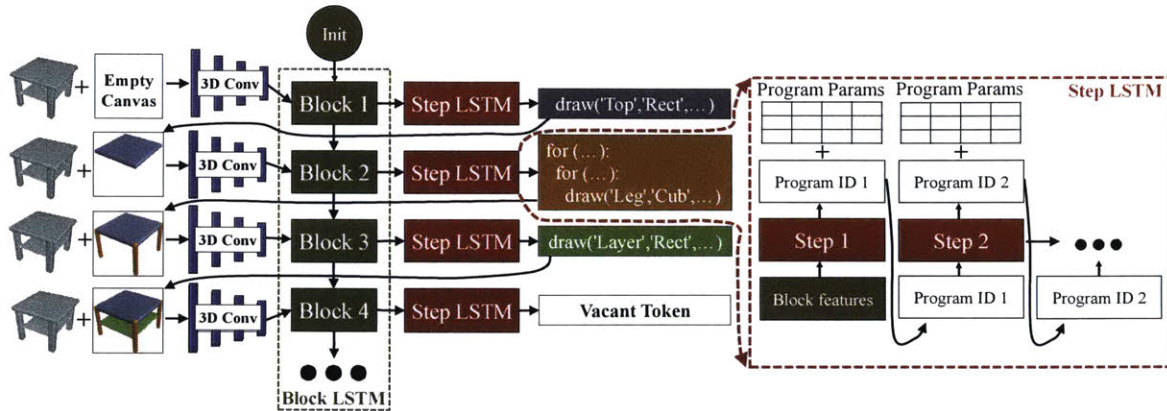
Figure 9-2: **Architecture of the 3D shape program generator.** The core of our 3D shape program generator are two LSTMs. The Block LSTM emits features for each program block. The Step LSTM takes these features as input and outputs programs inside each block, which includes either a single drawing statement or compound statements.

9.4.1 Program Generator

We model program generation as a sequential prediction problem. We partition full programs into two types of subprograms, which we call blocks: (1) a single drawing statement describing a semantic part, e.g. circle top; and (2) compound statements, which are a loop structure that interprets a set of translated or rotated parts, e.g. four symmetric legs. This part-based, symmetry-aware decomposition is inspired by human perception [Fleuret et al., 2011].

Our program generator is shown in Figure 9-2. The core of the program generator consists of two orthogonal LSTMs. The first one, the Block LSTM, connects sequential blocks. The second one, the Step LSTM, generates programs for each block. At each block, we first render the shape described by previous program blocks with a graphics engine. Then, the rendered shape and the raw shape are combined along the channel dimension and fed into a 3D ConvNet. The Block LSTM takes the features output by the 3D ConvNet and outputs features of the current block, which are further fed into the step LSTM to predict the block programs. The reason why we need the step LSTM is that each block might have a different length (*e.g.*, loop bodies of different sizes).

Given block feature h_{blk}, the Step LSTM predicts a sequence of program tokens, each consisting of a program id and an argument matrix. The i-th row of the argument matrix serves for the i-th primitive program. From the LSTM hidden state h_t, two decoders generate the output. The softmax classification probability over program sets is obtained by $f_{\text{prog}} : \mathbb{R}^M \to \mathbb{R}^N$. The argument matrix is computed by $f_{\text{param}} : \mathbb{R}^M \to \mathbb{R}^{N \times K}$, where N is the total number of program primitives and K is the maximum possible number of arguments. The feed-forward steps of the Step LSTM are summarized as

$$h_t = f_{\text{lstm}}(x_t, h_{t-1}), \tag{1}$$

$$p_t = f_{\text{prog}}(h_t), \ a_t = f_{\text{param}}(h_t), \tag{2}$$

where the p_t and a_t corresponds to the program probability distribution and argument matrix at time t. After getting the program ID, we obtain its arguments by retrieving the corresponding row in the argument matrix. At each time step, the input of the Step LSTM x_t is the embedding of the output in the previous step. For the first step, the block feature h_{blk} is used instead.

We pre-train our program generator on a synthetic dataset with a few pre-defined simple program templates. The set of all templates for tables are shown in Section C-1. These templates are much simpler than the actual shapes. The generator is trained to predict the program token and regress the corresponding arguments via the following loss $l_{\text{gen}} = \sum_{b,i} w_p l_{\text{cls}}(p_{b,i}, \hat{p}_{b,i})) + w_a l_{\text{reg}}(a_{b,i}, \hat{a}_{b,i})$, where $l_{\text{cls}}(p_{b,i}, \hat{p}_{b,i}))$ and $l_{\text{reg}}(a_{b,i}, \hat{a}_{b,i})$ are the cross-entropy loss of program ID classification and the $\mathcal{L}$-2 loss of argument regression, in step i of block b, respectively. The weights w_p and w_a balance the losses between classification and regression.

9.4.2 Neural Program Executor

We propose to learn a neural program executor, an approximate but differentiable graphics engine, which generates a shape from a program. The program executor can then be used for training the program generator by back-propagating gradients. An alternative is to design a graphics engine that explicitly executes a symbolic program to produce a voxelized 3D shape. Certain high-level program commands, such as For statements, will make the executor non-differentiable. Our use of a neural, differentiable executor enables gradient-based fine-tuning of the program synthesizer on unannotated shapes, which allows the model to generalize effectively to novel shapes outside training categories.

Learning to execute a long sequence of programs is difficult, since an executor has to learn to interpret not only single statements but also complex combinations of multiple statements. We decompose the problem by learning an executor that executes programs at the block level, e.g., either a single drawing statement or a compound statements. Afterwards, we integrate these block-level shapes by max-pooling to form the shape corresponding to a long sequence of programs. Our neural program executor includes an LSTM followed by a deconv CNN, as shown in Figure 9-3. The LSTM aggregates the block-level program into a fixed-length representation. The following deconv CNN takes this representation and generates the desired shape.

To train the program executor, we synthesize large amounts of block-level programs and their corresponding shapes. During training, we minimize the sum of the weighted
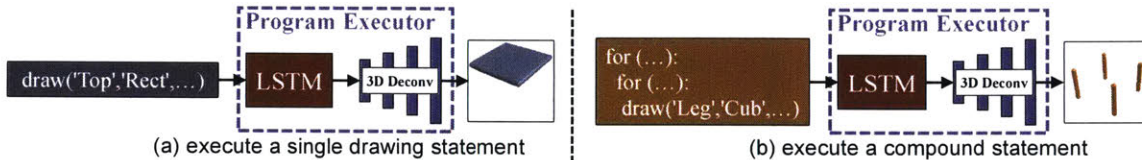
Figure 9-3: **Architecture of the program executor.** The learned program executor consists of an LSTM, which encodes multiple steps of programs, and a subsequent 3D DeconvNet which decodes the features to a 3D shape.
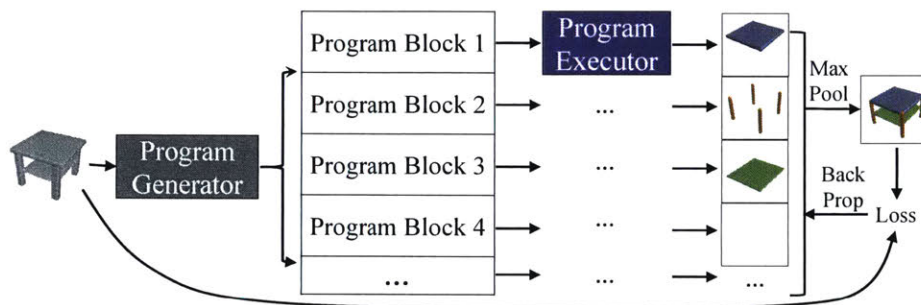
Figure 9-4: **Overview of inference and execution.** Given an input 3D shape, the neural program executor executes the generated programs. Errors between the rendered shape and the raw input are back-propagated.

binary cross-entropy losses over all voxels via

$$\mathcal{L} = \sum_{v \in V} -w_1 y_v \log \hat{y}_v - w_0 (1 - y_v) \log(1 - \hat{y}_v), \qquad (9.1)$$

where v is a single voxel of the whole voxel space V, y_v and $\hat{y}_v$ are the ground truth and prediction, respectively, while w_0 and w_1 balance the losses between vacant and occupied voxels. This training leverages only synthetic data, not annotated shape and program pairs, which is a blessing of our disentangled representation.

9.4.3 Guided Adaptation

A program generator trained only on a synthetic dataset does not generalize well to real-world datasets. With the learned differentiable neural program executor, we can adapt our model to other datasets such as ShapeNet, where program-level supervision is not available. We execute the predicted program by the learned neural program executor and compute the reconstruction loss between the generated shape and the input. Afterwards, the program generator is updated by the gradient back-propagated from the learned program executor, whose weights are frozen.

This adaptation is guided by the learned program executor and therefore called *guided adaptation (GA)*, and is shown in Figure 9-4. Given an input shape, the program generator first outputs multiple block programs. Each block is interpreted as 3D shapes by the program executor. A max-pooling operation over these block-level

190

shapes generates the reconstructed shape. The use of max-pooling also enables our executor to handle programs of variable length. Vacant tokens are also executed and pooled. Gradients can then propagate through vacant tokens and the model can learn to add new program primitives accordingly. Here, the loss for *Guided Adaptation* is the summation of the binary cross-entropy loss over all voxels.

9.5 Experiments

We present program generation and shape reconstruction results on three datasets: our synthetic dataset, ShapeNet [Chang et al., 2015], and Pix3D [Sun et al., 2018b].

Setup. In our experiments, we use a single model to predict programs for multiple categories. Our model is first pre-trained on the synthetic dataset and subsequently adapted to target dataset such as ShapeNet and Pix3D under the guidance of the neural program executor. All components of our model are trained with Adam [Kingma and Ba, 2015].

9.5.1 Evaluation on Synthetic Data

Program generator. We first pre-train our program generator on our synthetic dataset with simple templates. The synthetic training set includes 100,000 chairs and 100,000 tables. The generator is evaluated on 5,000 chairs and tables. More than 99.9% of the programs are accurately predicted. The shapes rendered by the predicted programs have an average IoU of 0.991 with the input shapes. This high accuracy is due to the simplicity of the synthetic dataset.

Program executor. Our program executor is trained on 500,000 pairs of synthetic block programs and corresponding shapes, and tested on 30,000 pairs. The IoU between the shapes rendered by the executor and the ground truth is 0.93 for a single drawing statement and 0.88 for compound statements. This shows the neural program executor is a good approximation of the graphics engine.

9.5.2 Guided Adaptation on ShapeNet

Setup. We validate the effectiveness of *guided adaptation* by testing our model on unseen examples from ShapeNet. For both tables and chairs, we randomly select 1,000 shapes for evaluation and all the remaining ones for *guided adaptation.*

Quantitative results. After our model generates programs from input shapes, we execute these programs with a graphics engine and measure the reconstruction quality. Evaluation metrics include IoU, Chamfer distance (CD) [Barrow et al., 1977], and Earth Mover's distance (EMD) [Rubner et al., 2000]. While the pre-trained model achieves 0.99 IoU on the synthetic dataset, the IoU drops below 0.5 on ShapeNet, showing the significant disparity between these two domains. As shown in Table 9.2,

Models	IoU ↑		CD ↓		EMD ↓	
	table	chair	table	chair	table	chair
CSGNet-original	0.111	0.154	0.216	0.175	0.205	0.177
Tulsiani et al. [2017a]	0.357	0.406	0.083	0.079	0.073	0.072
CSGNet-augmented	0.406	0.365	0.072	0.077	0.069	0.076
Nearest Neighbour	0.445	0.389	0.083	0.084	0.084	0.084
Shape Programs w/o GA	0.487	0.422	0.067	0.072	0.063	0.072
Shape Programs	**0.591**	**0.516**	**0.058**	**0.063**	**0.056**	**0.060**

Table 9.2: **Shape reconstruction results on ShapeNet**, evaluated in intersection over union (IoU, higher is better), Chamfer distance (CD, lower is better), and Earth Mover's distance (EMD, lower is better). Our model outperforms the baselines.

all evaluation metrics suggests improvement after *guided adaptation*. For example, the IoUs of table and chair increase by 0.104 and 0.094, respectively. We compare our method with Tulsiani et al. [2017a], which describes shapes with a set of primitives; and CSGNet [Sharma et al., 2018], which learns to describe shapes by applying arithmetic over primitives. For CSGNet, we evaluate two variants: first, CSGNet-original, where we directly test the model released by the original authors; second, CSGNet-augmented, where we retrain CSGNet on our dataset with the additional shape primitives we introduced. We also introduce a nearest neighbor baseline, where we use Hamming distance to search for a nearest neighbour from the training set for each testing sample.

Our model without *guided adaptation* outperforms Tulsiani et al. [2017a] and CSGNet by a margin, showing the benefit of capturing regularities such as symmetry and translation. The NN baseline suggests that simply memorizing the training set does not generalize well to test shapes. With the learned neural program executor, we try to directly train our program generator on ShapeNet without any pre-training. This trial failed, possibly because of the extremely huge and complicated combinatorial space of programs. However, the initial programs for pre-training can be very simple: e.g., 10 simple table templates (Fig. A1) are sufficient to initialize the model, which later achieves good performance under execution-guided adaptation.

Qualitative results. Figure 9-5 shows some program generation and shape reconstruction results for tables and chairs, respectively. The input shapes can be noisy and contain components that are not covered by templates in our synthetic dataset. After guided adaption, our model is able to extract more meaningful programs and reconstruct the input shape reasonably well.

Our model can be adapted to either add or delete programs, as shown in Figure 9-5. In (a), we observe an addition of translation describing the armrests. In (b) the "cylinder support" program is removed and a nested translation is added to describe
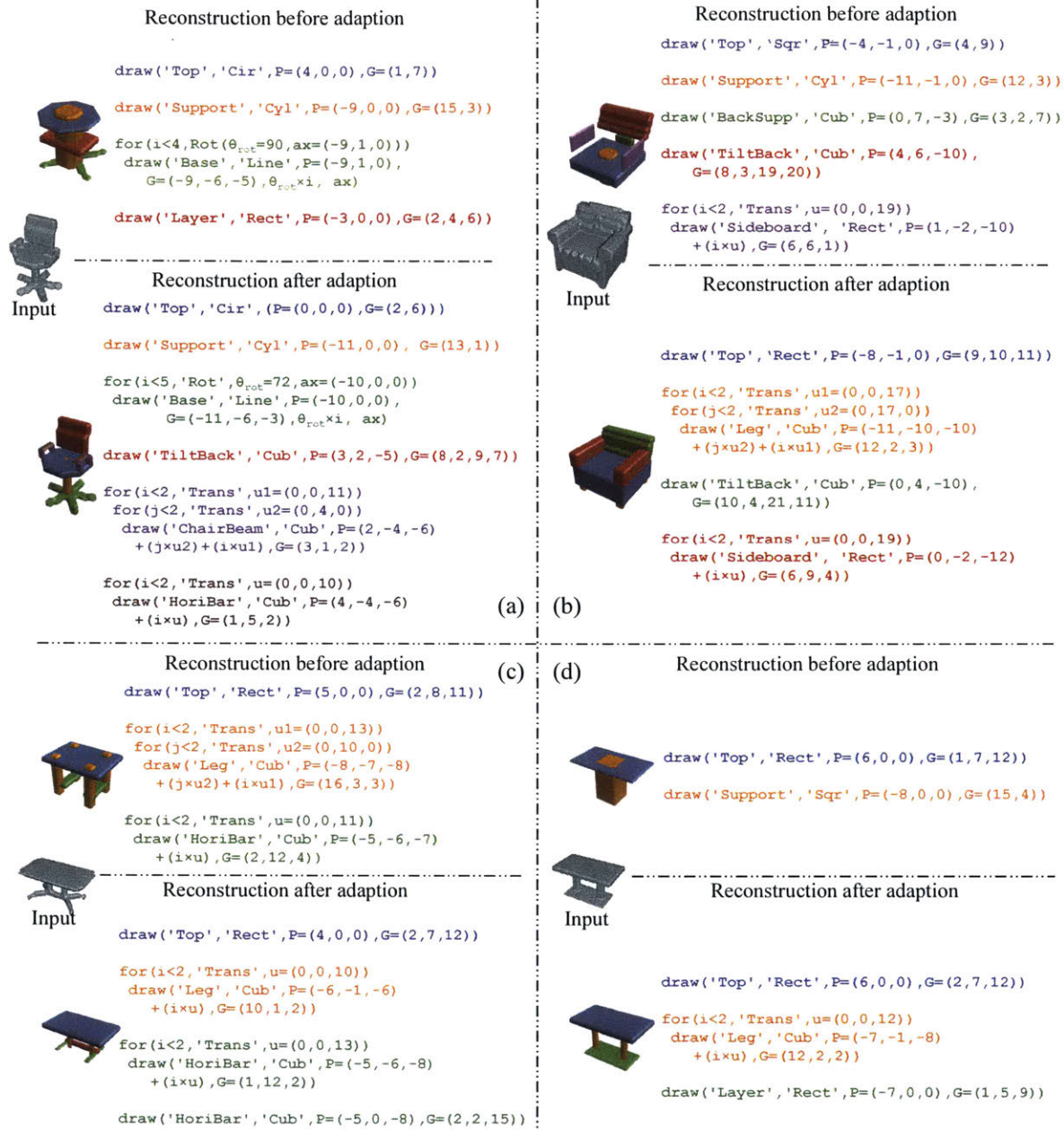
Figure 9-5: **Inferred programs for ShapeNet chairs and tables.** For each shape, the first and second rows represent results before and after *guided adaptation*. Best viewed in color.

four legs. In (c) and (d), the addition of "Horizontal bar" and "Rectangle layer" leads to more accurate representation. Improvements utilizing modifications to compound programs are not restricted to translations, but can also be observed in rotations, e.g., the times of rotation in (a) is increased from 4 to 5. We also notice new templates emerges after adaptation, e.g., tables in (c) and (d) are not in the synthetic dataset (check the synthetic templates for tables in Appendix C). These changes are significant because it indicates the generator can map complex, non-linear relationships to the

193

Models	Stable (%)		Connected (%)		Stable & Connected (%)	
	table	chair	table	chair	table	chair
Tulsiani et al. [2017a]	36.7	31.3	37.1	**68.9**	15.4	19.6
Shape Programs w/o GA	94.7	95.1	76.6	54.2	73.7	51.6
Shape Programs	**97.0**	**96.5**	**78.4**	68.5	**77.0**	**66.0**
Ground Truth	98.9	97.6	98.8	97.8	97.7	95.5

Table 9.3: **Measurement of stability and connectivity.** Our model is able to capture shape regularity such as symmetry. Therefore, shapes represented by our programs are more stable and better connected.

Models	IoU $\uparrow$				CD $\downarrow$			
	bed	sofa	cabinet	bench	bed	sofa	cabinet	bench
Shape Programs w/o GA	0.234	0.296	0.251	0.176	0.126	0.103	0.104	0.098
Shape Programs	**0.367**	**0.597**	**0.478**	**0.418**	**0.096**	**0.067**	**0.092**	**0.059**

Table 9.4: **Shape reconstruction results on unseen categories.** Results with or without *guided adaptation* in intersection over union (IoU, higher is better) and Chamfer distance (CD, lower is better).

program space.

9.5.3 Stability and Connectivity Measurement

Stability and connectivity are necessary for the functioning of many real-world shapes. This is difficult to capture using purely low-level primitives, but are better suited to our program representations.

We define a shape as stable if its center of mass falls within the convex hull of its ground contacts, and we define a shape as connected if all voxels form one connected component. In Table 9.3, we compare our model against Tulsiani et al. [2017a] and observe significant improvements in the stability of shapes produced by our model when compared to this baseline. This is likely because our model is able to represent multiple identical objects by utilizing translations and rotations. Before GA, our model produces chairs with lower connectivity, but we observe significant improvements with GA. This can be explained by the significant diversity in the ShapeNet dataset under the "chair" class. However, the improvements with GA also demonstrate an ability for our model to generalize. Measured by the percentage of produced shapes that are stable and connected, our model gets significantly better results, and continues to improve with *guided adaptation*.

194

Figure 9-6: **ShapeNet objects from unseen categories reconstructed with shape programs** before and after *guided adaptation*. Shape Programs can learn to adapt and explain objects from novel classes.
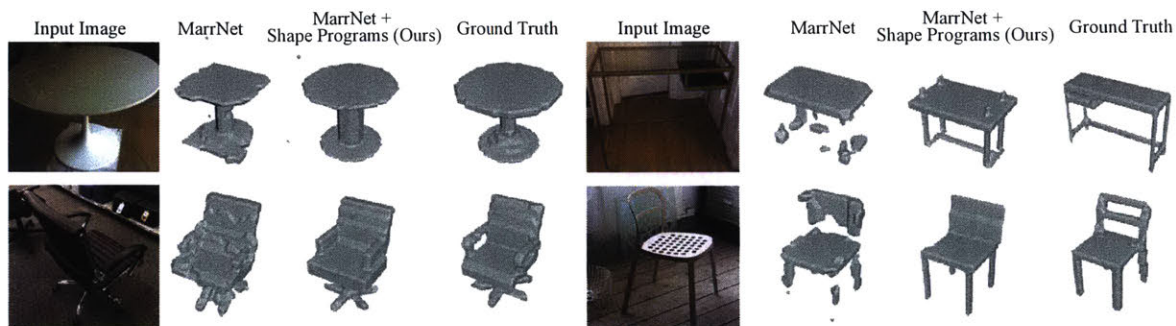
Figure 9-7: **3D reconstruction results on the Pix3D dataset.** MarrNet generates fragmentary shapes and our model further smooths and completes such shapes.

9.5.4 Generalization on Other Shapes

While our program generator is pre-trained only on synthetic chairs and tables, generalization on other shape categories is desirable. We further demonstrate that with *guided adaptation*, our program generator can be transferred to other unseen categories.

We consider *Bed, Bench, Cabinet,* and *Sofa*, which share similar semantics with table and chair but are unseen during pre-training. We split 80% shapes of each category for *guided adaptation* and the remaining for evaluation. Table 9.4 suggests the pre-trained model performs poorly for these unseen shapes but its performance improves with this unsupervised *guided adaptation*. The IoU of bed improves from 0.23 to 0.37, sofa from 0.30 to 0.60, cabinet from 0.25 to 0.48, and bench from 0.18 to 0.42. This clearly illustrates the generalization ability of our framework. Visualized examples are show in Figure 9-6.

9.5.5 Shape Completion and Smoothing by Programs

One natural application of our model is to complete and smooth fragmentary shapes reconstructed from 2D images. We separately train a MarrNet [Wu et al., 2017c] model for chairs and tables on ShapeNet, and then reconstruct 3D shapes from 2D images on the Pix3D dataset. As shown in Figure 9-7, MarrNet can generate

195

(a) Manipulation of dimension 27

(b) Manipulation of dimension 25

(c) Manipulation of dimension 41
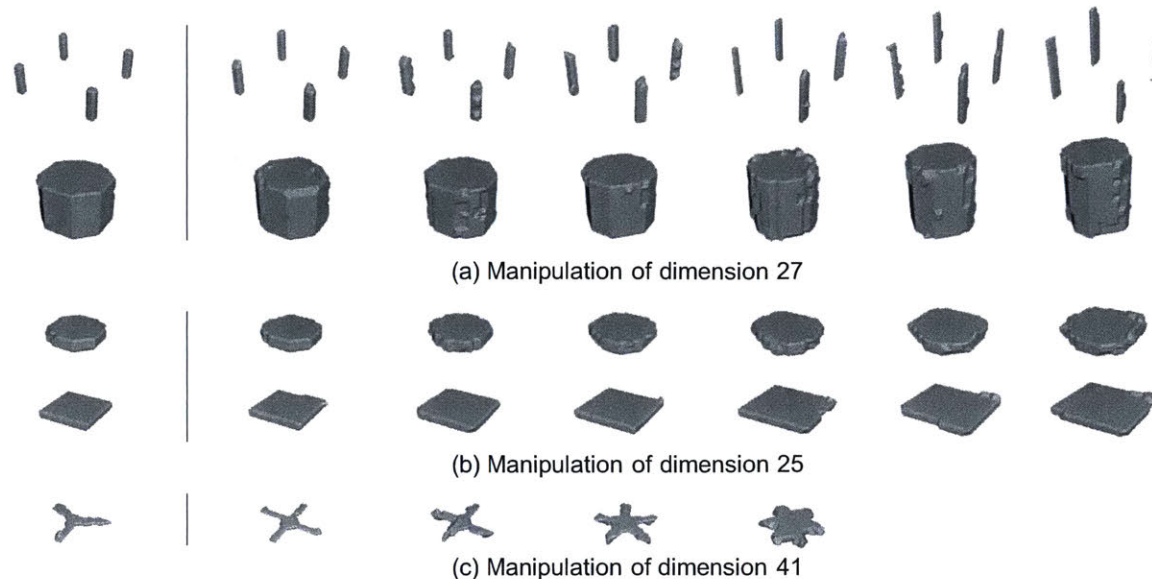
Figure 9-8: **We visualize the effect of manipulating individual dimensions** in the intermediate representation of neural program executor. For example, dimension 27 corresponds to the height of primitives, dimension 25 to the radius of primitives, and dimension 41 to the times of primitive repetition.

fragmentary shapes, which are then fed into our model to generate programs. These programs are executed by the graphics engine to produce a smooth and complete shape. For instance, our model can complete the legs of chairs and tables, as shown in Figure 9-7.

While stacking our model on top of MarrNet does not change the IoU of 3D reconstruction, our model produces more visually appealing and human-perceptible results. A user study on AMT shows that 78.9% of the participant responses prefer our results rather than MarrNet's.

9.6 Discussion

We have introduced 3D shape programs as a new shape representation. We have also proposed a model for inferring shape programs, which combines a neural program synthesizer and a neural executor. Experiments on ShapeNet show that our model successfully explains shapes as programs and generalizes to shapes outside training categories. Further experiments on Pix3D show our model can be extended to infer shape programs and reconstruct 3D shapes directly from color images. We now discuss key design choices and future work.

Analyzing the neural program executor. We look deep into the intermediate representation of the neural program executor, which is a 64-dimensional vector output by the LSTM (see Figure 9-3). We manipulate individual dimensions and visualize

the generated voxels. Figure 9-8 shows that these dimensions capture interpretable geometric features (e.g., height, radius, and number of repetitions).

Design of the DSL. Our design of the DSL for shape programs makes certain semantic commitments. A DSL with these semantics has advantages and disadvantages: it naturally supports semantic correspondences across shapes and enables better in-class reconstructions; on the other hand, it may limit the ability to generalize to shapes outside training classes. Our current instantiation focuses on the semantics of furniture (a superclass, whose subclasses share similar semantics). Within this superclass, our model generalizes well: trained on chairs and tables, it generalizes to new furniture categories such as beds. In future work, we are interested in learning a library of shape primitives directly from data, which will allow our approach to adapt automatically to new superclasses or domains of shape.

Structure search vs. amortized inference. For program synthesis, we use neural nets for amortized inference rather than structure search, due to the large search space and our desire to return a shape interpretation nearly instantaneously, effectively trading neural net training time for fast inference at test time. Our model takes about 5ms to infer a shape program with a Titan X GPU. We also considered various possible approaches for structured search over the space of shape programs, but decided that these would most likely be too our slow for our goals.

One approach to structured search is constraint solving. Ellis et al. [2015] used the Z3 SMT solver [De Moura and Bjørner, 2008] to infer 2D graphics programs, taking 5–20 minutes for problems arguably simpler than our 3D shape programs. Other approaches could be based on stochastic search, such as MCMC in the space of programs. For the related problem of inverse graphics from 2D images, MCMC, like constraint solving, takes too long for perception at a glance [Kulkarni et al., 2015b]. Efficient integration of discrete search and amortized inference, however, is a promising future research direction.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 10

Learning Scene Programs

Humans are capable of building holistic scene representations at various levels, from local objects and parts, to pairwise relations, to global structures. Finding global scene structures involves reasoning about the higher-order relationship (e.g., repetition) among multiple objects in the scene. In Chapter 8 and Chapter 9, we have presented models that automatically discover entry-level visual concepts and, in turn, combine them to explain the 3D shape of a single object. The focus of this chapter is to extend these models for multi-object scenes.

Our main contribution in this chapter is the Program-Guided Image Manipulator (PG-IM), building neuro-symbolic, program-like representations for image manipulation. Given an image, PG-IM detects repeated patterns, induces symbolic programs, and manipulates the image using a neural network that is guided by the program. PG-IM learns from a single image, exploiting its internal statistics. Despite trained only on image inpainting, PG-IM is directly capable of extrapolation and regularity editing in a unified framework. Extensive experiments show that PG-IM achieves superior performance on all these tasks.

This chapter includes materials previously published as Mao et al. [2019b], Liu et al. [2019]. Jiayuan Mao, Xiuming Zhang, and Yunchao Liu contributed significantly to the materials presented in this chapter.

10.1   Introduction

Looking at the images in Figure 10-1, we effortlessly identify the objects (pieces of cereal) in the image, interpret their pairwise relations, and reason over the global *regularity*: all pieces of cereal are organized on a 2D lattice with a triangular boundary. This holistic representation empowers our imagination of unseen objects: we can inpaint missing pixels in images, extrapolate images while preserving the regularity [Rock and Palmer, 1990], and reduce or exaggerate the regularity.

While tremendous progress has been made in object recognition [He et al., 2016] and visual relation detection [Lu et al., 2016], a global representation for *structural*

```
for i in range(0, 5):
    for j in range(0, 9):
        if i + j >= 8:
            draw(
                33 + 45 * i + 24 * j,
                6 + 45 * j,
                █,
            )
```

A. Inpainting   **B. Extrapolation**   **C. Regularity Editing**
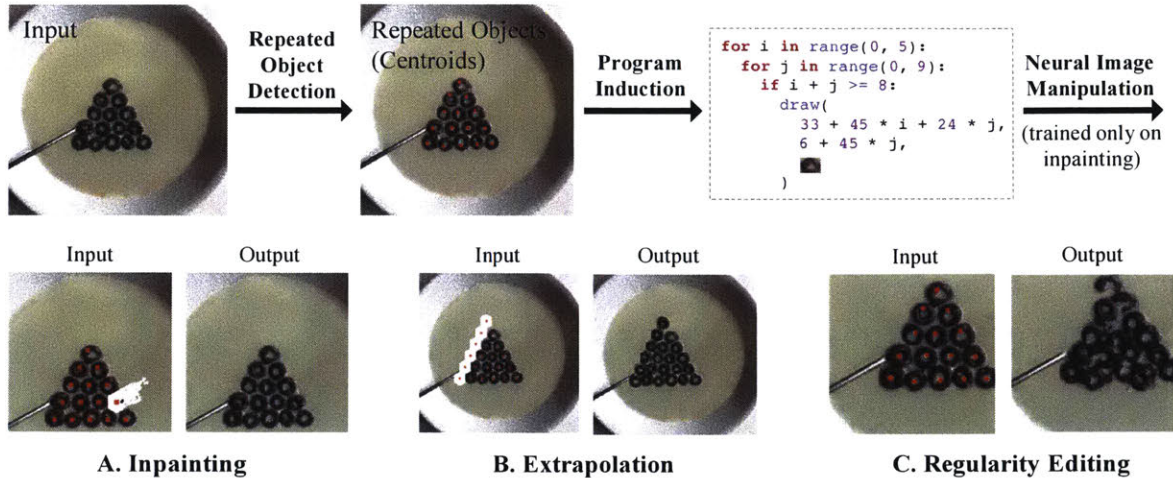
Figure 10-1: **Overview of the Program-Guided Image Manipulator (PG-IM).** Given an input image, PG-IM detects repeated entities in the image (pieces of cereal) and then infers a program-like representation for describing the regularity of the image. The regularity representation empowers multiple downstream tasks, such as image inpainting, extrapolation, and regularity editing.

regularity is still missing in these studies. In this chapter, we propose to augment deep networks, which are very powerful in pixel-level recognition, with symbolic programs, which are flexible to capture high-level regularity within the image. The intuition is that the disentanglement between perception and reasoning will enable complex image manipulation, preserving both high-level scene structure and low-level object appearance.

Our model, the Program-Guided Image Manipulator (PG-IM), induces symbolic programs for global regularities and manipulates images with deep generative models guided by the programs. PG-IM consists of three modules: a neural module that detects repeated patterns within the input image, a symbolic program synthesizer that infers programs for spatial regularity (lattice structure) and content regularity (object attributes), and a neural generative model that manipulates images based on the inferred programs.

We demonstrate the effectiveness of PG-IM on two datasets: the Nearly-Regular Pattern dataset [Lettry et al., 2017] and the Facade dataset [Teboul et al., 2010]. Both datasets contain nearly-regular images with lattice patterns of homogeneous objects. We also extend our experiments to a collection of Internet images with non-lattice patterns and variations in object appearance. Our neuro-symbolic approach robustly outperforms neural and patch-matching-based baselines on multiple image manipulation tasks, such as inpainting, extrapolation, and regularity editing.

200

10.2 Related Work

Image manipulation. Image manipulation is a long-standing problem in computer vision, graphics, and computational photography, most often studied in the context of image inpainting. Throughout decades, researchers have developed numerous inpainting algorithms operating at various levels of image representations: pixels, patches, and most recently, holistic image features learned by deep networks. Pixel-based methods often rely on diffusion [Ashikhmin, 2001, Ballester et al., 2001] and work well when the holes are small; later, patch-based methods [Efros and Freeman, 2001, Barnes et al., 2009] accelerate pixel-based methods and achieve better results. Both methods do not perform well in cases that require high-level information beyond background textures.

Deep networks are good at learning semantics from large datasets, and the learned semantic information has been applied to image manipulation [Xie et al., 2012, Pathak et al., 2016, Ulyanov et al., 2018]. Many follow-ups have been proposed to improve the results via multi-scale losses [Iizuka et al., 2017, Yang et al., 2017], contextual attention [Yu et al., 2018], partial convolution [Liu et al., 2018a], gated convolution [Yu et al., 2019], among others [Zhou et al., 2018, Yan et al., 2018]. Although these methods achieve impressive inpainting results with the learned semantic knowledge, they have two limitations: first, they rely on networks to learn object structure implicitly, and may fail to capture explicit, global object structures, such as the round shape of a clock [Xiong et al., 2019]; second, the learned semantics is specific to the training set, while real-world test images are likely to be out-of-distribution. Very recently, Xiong et al. [2019] and Nazeri et al. [2019] tackled the first problem by explicitly modeling contours to help the inpainting system preserve global object structures. In this chapter, we propose to tackle both problems using a combination of bottom-up deep recognition networks and the top-down neuro-symbolic program induction. We apply our approach to scenes with an arbitrary number of objects.

Program induction and procedural modeling. The idea of using procedural modeling for visual data has been a well-studied topic in computer graphics, mostly for indoor scenes [Wang et al., 2011, Li et al., 2019a, Niu et al., 2018] and 3D shapes [Li et al., 2017a]. More recently, with deep recognition networks, researchers have studied converting 2D images to line-drawing programs [Ellis et al., 2018], primitive sets [Sharma et al., 2018], markup code [Deng et al., 2017, Beltramelli, 2018], or symbolic programs with attributes [Liu et al., 2019]. As we will see in Section 10.3, these papers tackle synthetic images in a constrained domain, while here we study natural images.

SPIRAL [Ganin et al., 2018] used reinforcement learning to derive "drawing commands" for natural images. Their commands are, however, not interpretable, and

it is unclear how they can be extended to handle complex relations among a set of objects. Most recently, Young et al. [2019] integrated formal representations with deep generative networks and applied it to natural image inpainting. Still, our model differs from theirs in two aspects. First, we use neural modules for discovering repeated patterns in images, which does not require the patch of interest to repeat itself over the entire image (an assumption made in Young et al. [2019]). Second, their algorithm requires learning semantics on a pre-defined dataset for manipulation (e.g., image extrapolation); in contrast, our model exploits the idea of internal learning [Shocher et al., 2018] and requires no training data during image manipulation other than the image itself.

Single-image learning. Because visual entropy inside a single image is lower than in a diverse collection of images [Zontak and Irani, 2011], many approaches have exploited image-level (instead of dataset-level) statistics for various image editing tasks including deblurring [Bahat et al., 2017, Michaeli and Irani, 2014], super-resolution [Glasner et al., 2009, Freedman and Fattal, 2011, Huang et al., 2015], and dehazing [Bahat and Irani, 2016]. The same philosophy has also been proven successful in deep learning, where neural networks are trained on (and hence overfit to) a single image. Such image-specific networks effectively encode image priors unique to the input image [Ulyanov et al., 2018]. They can be used for super-resolution [Shocher et al., 2018], layer decomposition [Gandelsman et al., 2019], texture modeling [Bergmann et al., 2017, Zhou et al., 2018], and even generation tasks [Shaham et al., 2019, Shocher et al., 2019].

Powerful as these approaches are, they often lack a high-level understanding of the input image's global structure (such as the triangular shape formed by the cereal in Figure 10-1). Consequently, there is usually no guarantee that the original structure gets preserved after the manipulation (e.g., Row 2 of Figure 10-6). This work augments single-image learning methods with symbolic reasoning about the input image's global structure, not only providing a natural way of preserving such structure, but also enabling higher-level, semantic manipulation based on the structure (e.g., extrapolating an additional row of cereal following the triangular structure in the teaser figure).

10.3   Program Synthesis for Synthetic Scenes

As a first attempt, we look into synthetic scenes of geometric primitives as shown in Figure 10-2a. Here we develop a model that combines convolutional nets for object recognition and recurrent nets for program synthesis. The model first uses an object parser to predict the segmentation mask and attributes for each object in the image. A group recognizer then predicts the group that each object belongs to. Finally, a program synthesizer generates a program block for each object group. Figure 10-2

```
for(i<3)
  for(j<3-i)
    sphere(pos=(1+i,1+j,0),
           color=6-j)
for(i<4)
    cylinder(pos=(3,3,i),
             color=7)
cylinder(pos=(4,2,0),
         color=3)
```

(sphere, small, metal, green, x=2, y=2, z=0) ...

(a) Input image   (b) Object parsing & group detection   (c) Program
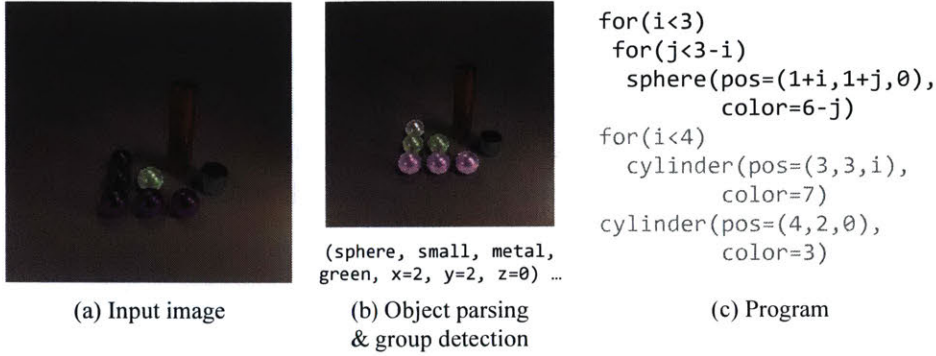
Figure 10-2: **Visual program synthesis for synthetic scenes.** (a) The input is an image consisting of multiple objects with ordered arrangements. We also perform instance segmentation to get object masks. (b) We use two vision models to extract object attributes and predict object groups, respectively. (c) These representations are then sent to a sequence model to predict the program.

Program	$\rightarrow$	Statement; $\cdots$; Statement
Statement	$\rightarrow$	cube(pos=Expression1, color=Expression2)
Statement	$\rightarrow$	sphere(pos=Expression1, color=Expression2)
Statement	$\rightarrow$	cylinder(pos=Expression1, color=Expression2)
Statement	$\rightarrow$	for($0 \leq$ Var1 $<$ Expression1){Program}
Statement	$\rightarrow$	rotate($0 \leq$ Var1 $<$ Expression1, start=Z, center=(Z, Z, Z)){Program}
Expression1	$\rightarrow$	$Z \times$ Var1 $+ \cdots + Z \times$ Var1 $+ Z$
Expression2	$\rightarrow$	$Z \times$ Var2 $+ \cdots + Z \times$ Var2 $+ Z$
Var1	$\rightarrow$	a free variable
Var2	$\rightarrow$	Var1 | Var1 % Z | Var1 / Z
Z	$\rightarrow$	integer

Table 10.1: **The DSL for programs on synthetic scenes.** Primitive commands (cube, sphere, cylinder) can be placed inside loop structures, where the position and color of each object are determined by the loop indices.

shows an example of synthesizing programs from an input image, where a sphere is selected at random (highlighted) and the group that this object belongs to is predicted, which consists of six spheres. Then the program for this group (highlighted) is synthesized.

A domain-specific language (DSL) for scenes. In order to constrain the program space to make it tractable for our models, we introduce human prior on scene regularities that can be described as programs. More specifically, we introduce a Domain Specific Language (DSL) which explicitly defines the space of our scene programs. We present the grammar of our DSL in Table 10.1, which contains three primitive commands (cube, sphere, cylinder) and two loop structures (for, rotate). The positions for each object are defined as affine transformations of loop indices, while the colors are more complicated functions of the loop indices, displaying alternating (modular) and repeating (division) patterns.

Furthermore, since the DSL allows unbounded program depth, we define *program blocks* to further reduce complexity. Each type of program block is an production instance of the Statement token, and objects that belong to the same block form a *group*. For example, in this work the program blocks include single objects, layered for loops of depth ≤ 3, and single-layer rotations of ≤ 4 objects.

Object parsing. Following the spirit of *the trace hypothesis* [Ellis et al., 2018], we use object attributes as an intermediate representation between image space and structured program space. Parsing individual objects from the input image consists of two steps: mask prediction and attribute prediction. For each object, its instance segmentation mask is predicted by a Mask R-CNN [He et al., 2017]. Next, the mask is concatenated with the original image, and sent to a ResNet-34 [He et al., 2016] to predict object attributes. In our work, object attributes include shape, size, material, color and 3D coordinates. Each attribute is encoded as a one-hot vector, except for coordinates. The overall representation of an object is a vector of length 18. The networks are trained with ground truth masks and attributes, respectively. For the attribute network, we minimize the mean-squared error between output and ground truth attributes.

Group detection. When we identify a distinct visual pattern, we first know which objects in the image form the pattern before we can tell what the pattern is. Motivated by this idea, we develop a group recognizer that tells us which objects form a group that can be described by a single program block. The group recognizer works after mask prediction is performed, and answers the following specific question: given an input object, which objects are in the same group with this object?

The input to the model consists of three parts: the original image, the mask of the input object, and the mask of all objects. These three parts are concatenated and sent to a ResNet-152 followed by fully connected layers. The output contains two parts: a binary vector g where $g[i] = 1$ denotes object i in the same group with the input object, and the category c of the group, representing the type of program block that this group belongs to. The network is trained to minimize the binary cross entropy loss for group recognition, and the cross entropy loss for category classification.

Neural program synthesis. With the object attributes and groups obtained from the vision models, the final step in our model is to generate program sequences describing the input image. Since we have already detected object groups, what remains is to generate a program block for each group. For this goal we train a sequence to sequence (seq2seq) LSTM with an encoder-decoder structure and attention mechanism [Luong et al., 2015, Bahdanau et al., 2015]. The input sequence is a set of object attributes that form a group, which are sorted by their 3D coordinates. The output program consists of two parts: program tokens are predicted as a sequence

as in neural machine translation, and program parameters are predicted by a MLP from the hidden state at each time step. At each step, we predict a token t as well as a parameter matrix P, which contains predicted parameters for all possible tokens. Then we use $P[t]$ as the output parameter for this step.

Since the program synthesizer only works for a single group, a method for combining the group prediction with program synthesis is needed. Consider the simplest case where we randomly choose an object and describe the group it belongs to. In practice, by default we sample 10 times and stop when a correct program is generated. Here correct means that we can recover the scene attributes successfully by executing the program.

10.4 Program-Guided Image Manipulators

The model introduced in Section 10.3 works well on synthetic scenes as those in Figure 10-2, but cannot easily generalize to natural images, where it's hard to obtain supervision on object attributes such as color and shape. In this section, we introduce Program-Guided Image Manipulator (PG-IM) as an extension to the original model, combining deep recognition and generative networks with program synthesis for natural image manipulation. PG-IM contains three modules, as shown in Figure 10-1. First, it detects repeated objects and make them a variable-length stack (Section 10.4.1). Then, it infers a program to describe the global regularity among the objects (Section 10.4.2), with program tokens such as for-loops for repetition and symmetry. Finally, the inferred program facilitates image manipulation, which is performed by a neural painting network (Section 10.4.3).

10.4.1 Repeated Object Detection

PG-IM detects repeated objects in the input image with a neural module based on Lettry et al. [2017]. Given the input image, it extracts convolutional feature maps from a pre-trained convolutional neural network (i.e., AlexNet [Krizhevsky et al., 2012]). A morphological filter is then applied to the feature maps for extracting activated neurons, resulting in a stack of *peakmaps*. Next, assuming the lattice pattern of repeated objects, a voting algorithm is applied to compute the displacements between nearby objects. Finally, an implicit pattern model (IPM) is employed to fit the centroids of objects. Please see Lettry et al. [2017] and Appendix D for details of the algorithm.

10.4.2 Program Synthesizer

The program synthesizer takes the centroids of the repeated objects as input and infers a latent program describing the pattern. The input image is partitioned into object patches by constructing a Voronoi graph of all pixels. That is, each pixel

```
for i in range(0, 6):
    for j in range(0, 3):
        draw(70 + 110 * i, 130 + 150 * j, ▨ )
```

```
for i in range(0, 9):
    for j in range(0, 5):
        draw(48 + 72 * i, 132 + 72 * j, ▦ )
```

```
for i in range(0, 5):
  for j in range(0, 9):
    if i + j >= 8:
        draw(33 + 45 * i + 24 * j, 6 + 45 * j, ● )
```
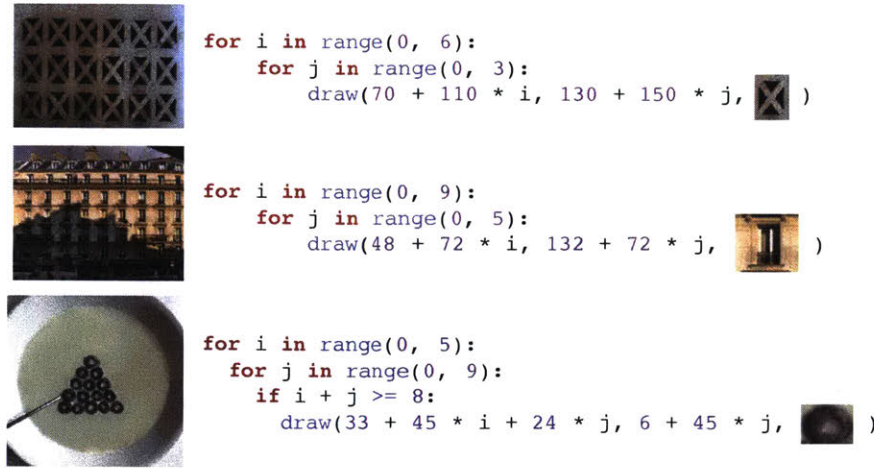
Figure 10-3: **Illustrative programs** inferred from (top row) the Nearly-Regular Pattern dataset [Lettry et al., 2017], (middle row) the Facade dataset [Teboul et al., 2010], and (bottom row) Internet images. The DSL of the inferred programs supports for-loops, conditions, and attributes.

Program	$\longrightarrow$ For1Stmt
For1Stmt	$\longrightarrow$ For (i in range(Integer, Integer))
{ For2Stmt }	
For2Stmt	$\longrightarrow$ For (i in range(Integer, Integer))
{ CondDrawStmt }	
CondDrawStmt	$\longrightarrow$ If (Expr $\geq$ 0) { CondDrawStmt }
CondDrawStmt	$\longrightarrow$ DrawStmt
DrawStmt	$\longrightarrow$ Draw (x=Expr, y=Expr,
attribute=AttributeExpr)	
AttributeExpr	$\longrightarrow$ Expr // Integer
AttributeExpr	$\longrightarrow$ 1 If (Expr == 0) else 0
AttributeExpr	$\longrightarrow$ 1 If (Expr == 0 and Expr == 0) else 0
AttributeExpr	$\longrightarrow$ 1 If (Expr % Integer == 0) else 0
AttributeExpr	$\longrightarrow$ 1 If (Expr % Integer == 0 and Expr % Integer == 0) else 0
Expr	$\longrightarrow$ Integer $\times i$ + Integer $\times j$ + Integer

Table 10.2: **The domain-specific language (DSL) for describing image regularities.** Language tokens including `For`, `If`, `Integer` and arithmetic/logical operators follow the convention of Python.

is assigned to its nearest centroid, under the metric of Euclidean distance between pixel coordinates. Meanwhile, objects are clustered into multiple groups. When the program reconstructs an object with the `Draw` command, it is allowed to specify both the coordinate of the object's centroid (x, y) and an integer (namely, the *attribute*), indicating which group this object belongs to. We implement our program synthesizer as a search-based algorithm that finds the *simplest* program that reconstructs the pattern.

Domain-specific language. We summarize the domain-specific language (DSL) used by PG-IM for describing object repetition in Table 10.2. In a nutshell, `ForStmt1`
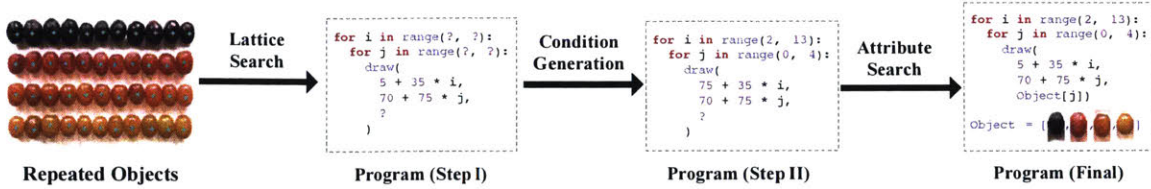
Figure 10-4: **The three-step inference of a program describing the shown repeated pattern**. Assuming the input keypoints follow a lattice pattern, we first search for parameters defining the lattice, such as the distance between nearby keypoints and the origin. Next, we fit boundary conditions for the program. Finally, we cluster objects into groups by their visual appearance, and fit an expression describing the variation.

and `ForStmt2` jointly define a lattice structure; `CondDrawExpr` defines the boundary of the lattice; `Draw` places an object at a given coordinate. `AttributeExpr` allows the attribute of the object to be conditioned on the loop variables (i and j). Figure 10-3 shows illustrative programs inferred from different datasets.

Program search. Finding the *simplest* program for describing a regularity pattern involves searching over a large compositional space of possible programs, which contains for-loops, if-conditions, coordinate expressions, and attribute expressions. To accelerate the search, we heuristically divides the search process into three steps, as illustrated in Figure 10-4. First, we search over all possible expressions for the coordinates, and find the one that fits the detected centroids the best. Second, we determine the conditions (the boundary). Finally, we find the expression for attributes.

Lattice search. The lattice search finds the expressions for coordinates x and y, ignoring all potential conditions and attribute expressions. Thus, the search process can be simplified as finding a 5-tuple $(b_x, b_y, d_{x,i}, d_{x,j}, d_{y,j})$ that satisfies $x = b_x + i \cdot d_{x,i} + j \cdot d_{x,j}$ and $y = b_y + j \cdot d_{y,j}$.

Each tuple defines a set of centroids $\mathbf{P}$ containing all (x, y) pairs whose coordinates are within the boundary of the whole image. We compare these sets with the centroids $\mathbf{C}$ detected by the repeated pattern detector. We find the optimal tuple as the one that minimizes a cost function

$$\mathcal{L}_{\text{lat}} = \sum_{(x,y)\in\mathbf{C}} \min_{(u,v)\in\mathbf{P}} \left[(x-u)^2 + (y-v)^2 \right] + \lambda|\mathbf{P}|, \tag{10.1}$$

where $\lambda = 5$ is a hyperparameter for regularization. It matches each detected centroid with the nearest one reconstructed by the program. The goal is to minimize the distance between them and a regularization term over the size of $\mathbf{P}$. From a Bayesian inference perspective, $\mathbf{P}$ defines a mixture of Gaussian distribution over the 2D plane. $\mathcal{L}_{\text{lat}}$ approximates the log-likelihood of the observation $\mathbf{C}$ and a prior distribution over possible $\mathbf{P}$'s, which favors small ones.

207

Condition search. In the next step, we generate the conditions of the program, assuming all centroids fit in a convex hull. This assumption covers both rectangular lattices and triangular lattices (see Figure 10-3 for examples). Since all pairs (x, y) are computed by an affine transformation of all (i, j)'s, the conditions can be determined by computing the convex hull of all (i, j)'s that are matched with detected centroids.

Specifically, we first match each coordinate in $\mathbf{P}$ with $\mathbf{C}$ by computing a minimum cost assignment between two sets, where the distance metric is the Euclidean distance in the 2D coordinate space. We then find the convex hull of all assigned pairs (i, j). We use the boundary of the convex hull as the conditions. The conditions include the boundary conditions of for-loops as well as optional if-conditions.

Attribute search. The last step is to find the expression that best describes the variance in object appearance (i.e., their attributes). Attributes are represented as a set of integers. Instead of clustering, we assign discrete labels to individual patches. The label of the patch in row p_i, column p_j is a function of (p_i, p_j). Shown in Table 10.2, each possible `AttributeExpr` defines an attribute assignment function $A(p) \triangleq A(p_i, p_j)$ for all centroids $p = (p_i, p_j) \in \mathbf{P}$. We say an expression fits the image if patches of the same label share similar visual appearance. Formally, we find the optimal parameters for the attribute expression that minimizes

$$\mathcal{L}_{\text{attr}} = \sum_{p \in \mathbf{P}} \sum_{q \in \mathbf{P}} \big(\text{sgn}(A(p), A(q)) \cdot d(p, q) \big) + \mu |A(\mathbf{P})|, \tag{10.2}$$

where $\text{sgn}(A(p), A(q)) = 1$ if $A(p) = A(q)$, and -1 otherwise. d computes the pixel-level difference between two patches centered at (p_i, p_j) and (q_x, q_y), respectively. $\mu = 10$ is a scalar hyperparameter of the regularization strength. $|A(\mathbf{P})|$ computes the number of distinct values of $A(p)$ for all $p \in \mathbf{P}$. The inference is done by searching over possible integer templates (e.g., $ai + bj + c$) and binary templates (e.g., $(ai + bj + c \; // \; d \; \% \; e) == 0$), and the coefficients $(a, b, c, \dots)$.

10.4.3 Neural Painting Networks

We propose the neural painting network (NPN), a neural architecture for manipulating images with the guidance of programs. It unifies three tasks: inpainting, extrapolation, and regularity editing in a single framework. The key observation is that all three tasks can be cast as filling pixels in images. For illustrative simplicity, we first consider the task of inpainting missing pixels in the image, and then discuss how to perform extrapolation and regularity editing using the same inpainting-trained network.

Patch aggregation. We first aggregate all pixels from other "objects" (loosely defined by the induceted program) to inpaint the missing pixels. Denote all object

208
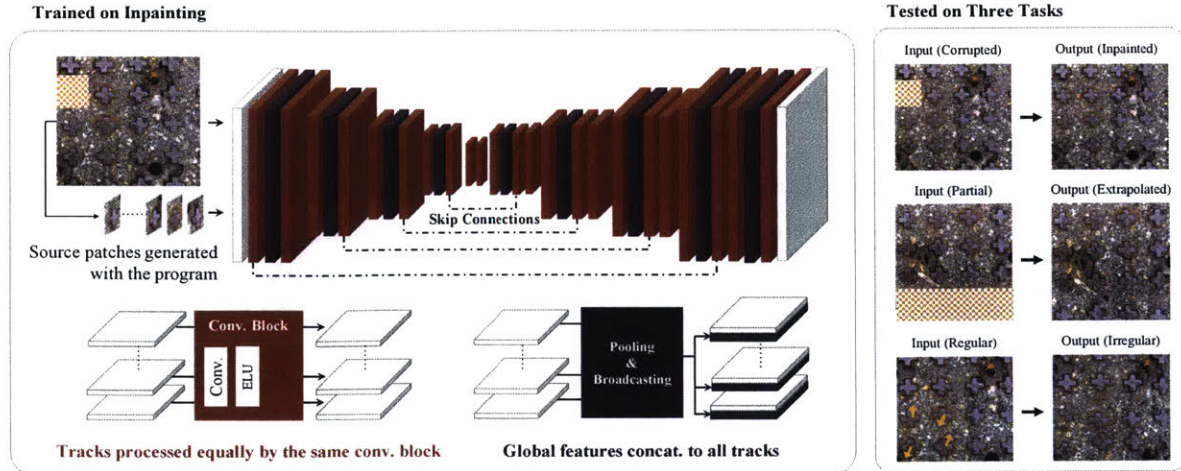
Figure 10-5: Architecture of a neural painting network (NPN). An NPN takes as input an image and a set of source patches, derived from the image with its program description, and outputs a manipulated image. An NPN learns from a single image, exploiting the image's internal statistics. Trained only on inpainting, it can directly extrapolate and edit the regularity of the input image in a unified inference framework, without any finetuning.

centroids reconstructed by the program as $\mathbf{P}$, the centroid of the object patch containing missing pixels (x_0, y_0), and all other centroids $\mathbf{P}^- = \mathbf{P} \setminus \{(x_0, y_0)\}$. The aggregation is performed by generating $|\mathbf{P}^-|$ images, the i-th of which is obtained by translating the original image such that the centroid of the i-th object in $\mathbf{P}^-$ is centered at (x_0, y_0). Pixels without a value after the shift are treated as 0. We stack the input image with missing pixels plus all the $|\mathbf{P}^-|$ images (the "patch source") as the input to the network.

Architecture. Our neural painting network (NPN) has a U-Net [Ronneberger et al., 2015] encoder-decoder architecture, designed to handle a variable number of input images and be invariant to their ordering. Demonstrated in Figure 10-5, the network contains a stack of shared-weight convolution blocks and max-pooling layers that aggregate information across all inputs. Paired downsampling and upsampling layers (convolution layers with strides) are skip-connected. The input of the network is the stack of the corrupted input image plus source patches, and the output of the network is the inpainted image. A detailed printout of the generator's architecture can be found in Appendix D.

The key insight of our design of the NPN is that it handles a variable number of input images in any arbitrary order. To this end, inspired by Aittala and Durand [2018] and Qi et al. [2017], we have a single encoder-decoder that processes the $|\mathbf{P}^-|+1$ images equally ("tracks"), and the intermediate feature maps from these tracks get constantly max-pooled into a "global" feature map, which is then broadcast back to the $|\mathbf{P}^-|+1$ tracks and concatenated to each track's local feature map to be processed

by the next block. Intuitively, the network is guided to produce salient feature maps that will "survive" the max-pooling, and the tracks exchange information by constantly absorbing the global feature map.

Extrapolation and regularity editing as recurrent inpainting. A key feature of program-guided NPNs is that although they are trained only on the inpainting task, they are able to be used directly for image extrapolation and regularity editing. With the program description of the image, NPNs are aware of where the entities are in the image, and hence able to cast extrapolation as recurrent inpainting of multiple corrupted objects. For instance, to extrapolate a 64-pixel wide stripe to the right, an NPN first queries the program description for where the new peaks are, and then recurrently inpaints each object given all the previously inpainted ones. Similarly for image regularity editing, when the (regularly spaced) centroids provided by the program get randomly perturbed, the pixels falling into their Voronoi cells move together with them accordingly, leaving many "cracks" on the image, which the NPN then inpaints recurrently.

Training. We train our NPNs with the same training paradigm as Isola et al. [2017]. We compute an L1 loss and a patch-based discriminator loss, between the generated (inpainted) image and the ground-truth image. We train image-specific NPNs for each individual image in the dataset. While only training the network to inpaint missing pixels, we show that the network can perform other tasks such as image extrapolation and regularity editing, by only changing the input to the network during inference. Other implementation details such as the hidden dimensions, convolutional kernel sizes, and training hyperparameters can be found in the Appendix D.

10.5 Experiments and Applications

We provide both quantitative and qualitative comparisons with the baselines on two standard image manipulation tasks: inpainting and extrapolation. We also show the direct application of our approach to image regularity editing, a task where the regularity of an image's global structure gets exaggerated or reduced. It is worth mentioning that these three problems can be solved with a single model trained for inpainting (see Section 10.4.3 for details). Finally, we demonstrate how our program induction easily incorporates object attributes (e.g., colors) in Internet images, in turn enabling our NPNs to manipulate images with high-level reasoning in an attribute-aware fashion. Please see Appendix D for ablation studies that evaluate each major component of PG-IM. We start with an introduction to the datasets and baseline methods we consider.

10.5.1 Dataset

We compare the performance of PG-IM with other baselines on two datasets: the Nearly-Regular Pattern (NRP) dataset [Lettry et al., 2017] and the Facade dataset [Teboul et al., 2010]. The Nearly-Regular Pattern dataset contains a collection of 48 rectified images with a grid or nearly grid repetition structure. The Facade dataset, specifically the CVPR 2010 subset, contains 109 rectified images of facades.

10.5.2 Baselines

We consider two groups of baseline methods: non-learning-based and learning-based. Among the non-learning-based methods are Image Quilting [Efros and Freeman, 2001] and PatchMatch [Barnes et al., 2009], both of which are based on the stationary assumption of the image structure. Intuitively, to inpaint a missing pixel, they fill it with the content of another existing pixel with the most similar context. Being unaware of the objects in the image, they rely on human-specified hyperparameters, such as the context window size, to produce reliable results. More importantly, in the case of extrapolation, the user needs to specify which pixels to paint, implicitly conveying the concept of objects to the algorithms. For PatchMatch and Image Quilting, we search for one set of optimal hyperparameters and apply that to the entire test set.

We also compare PG-IM with a learning-based, off-the-shelf algorithm for image inpainting: GatedConv [Yu et al., 2019]. They use neural networks for inpainting missing pixels by learning from a large-scale dataset (Place365 [Zhou et al., 2017a]) of natural images. GatedConv is able to generate novel objects that do not appear in the input image, which is useful for semantic photo editing. However, this may not be desired when the image of interest contains repeated but *unique* patterns: although a pattern appears repeatedly in the image of interest, it may not appear anywhere else in the dataset.

Therefore, we also consider another learning-based baseline, originally designed for image extrapolation: Non-Stationary Texture Synthesis (Non-Stationary) [Zhou et al., 2018]. In their framework, an image-specific neural network is trained for each input image. Its objective is to extrapolate a small (usually unique) patch ($k \times k$) into a large one ($2k \times 2k$). Although both of their method and PG-IM use single-image training for generating missing pixels, PG-IM uses symbolic programs as the guidance of the networks, enjoying both interpretability and better performance for complex structures. We also implement a variant of Non-Stationary, which keeps the neural architecture and training paradigm as the original version for texture synthesis, but use the same inpainting data as our method for inpainting. For a fair comparison, we train Non-Stationary and PG-IM with single sets of optimal hyperparameters on all test images.

Method	L1 Mean (Std.)	Inception Score
Nearly-Regular Patterns [Lettry et al., 2017]		
Image Quilting [Efros and Freeman, 2001]	**12.30 (2.903)**	**1.253**
PatchMatch [Barnes et al., 2009]	83.91 (17.62)	1.210
GatedConv [Yu et al., 2019]	50.45 (16.46)	1.196
Non-Stationary [Zhou et al., 2018]	103.7 (23.87)	1.186
PG-IM (ours)	**21.48 (5.375)**	**1.229**
Facade [Teboul et al., 2010]		
Image Quilting [Efros and Freeman, 2001]	**13.50 (6.379)**	1.217
PatchMatch [Barnes et al., 2009]	81.35 (25.28)	**1.219**
GatedConv [Yu et al., 2019]	26.26 (133.9)	1.186
Non-Stationary [Zhou et al., 2018]	133.9 (39.75)	1.199
PG-IM (ours)	**14.40 (7.781)**	**1.218**

Table 10.3: **We compare PG-IM against off-the-shelf neural baselines for image inpainting on both datasets.** Our method outperforms neural baselines with a remarkable margin across all metrics.

10.5.3 Inpainting

We compare PG-IM with GatedConv, Image Quilting, and PatchMatch on the task of image inpainting. For quantitative evaluations, we use the NRP and Facade datasets, each of whose images gets randomly corrupted 100 times, giving us a total of around 15,000 test images.

Table 10.3 summarizes the quantitative scores of different methods. Following Liu et al. [2018a], we compare the L1 distance between the inpainted image and the original image, as well as Inception score (IS) of the inpainted image. For all the approaches, we hold out a test patch whose pixels are never seen by the networks during training, and use that patch for testing. Quantitatively, PG-IM outperforms the other learning-based methods by large margins across both datasets in both metrics. PG-IM recovers missing pixels a magnitude more faithful to the ground-truth images than Non-Stationary in the L1 sense. It also has a small variance across different images and input masks. For comparisons with non-learning-based methods, although Image Quilting achieves the best L1 score, it tends to break structures in the images, such as lines and grids (see Figure 10-6 for such examples). Note that the reason why PatchMatch has worse L1 scores is that it also modifies pixels around the holes to achieve better image-level consistency. In contrast, the other methods including PG-IM only inpaint holes and modify nothing else in the images.

Qualitative results for inpainting are presented in Figure 10-6. Overall, our approach is able to preserve the "objects" in the test images even if the objects are completely missing, while other learning-based approaches either miss the intricate structures (Non-Stationary on Images 1 and 2), or produce irrelevant patches (learned

Figure 10-6: **Corrupted input images and inpainting results (zoomed-in) by PG-IM and the baselines.** The white pixels in the leftmost column are missing pixels to inpaint. The rightmost column shows the ground-truth patches. PG-IM inpaints realistic image patches that are consistent with the intricate global regularity and meanwhile different from the original, ground-truth patches.

from largely diverse image datasets) that break the global structure of this particular image (e.g., GatedConv on Image 2). Note how the image patches inpainted by our approach is realistic and meanwhile quite different from the ground-truth patches (compare our inpainting with the ground-truth Image 4). For the non-learning-based approaches, the baselines suffer from blurry outputs and sometimes produce inconsistent connections to the original image on boundaries. Moreover, as we will demonstrate in Figure 10-9, unlike our approach that combines high-level symbolic reasoning and lower-level pixel manipulations, PatchMatch fails to manipulate the pixels in an attribute-aware fashion.

Runtime-wise, learning-methods including PG-IM, once trained, inpaint an image in a forward pass (around 100ms on GPUs), whereas non-learning-based approaches take around 15 minutes to inpaint one image.

10.5.4 Extrapolation

Figure 10-7 shows the extrapolation results by PG-IM and the baselines. With the program description of the images, PG-IM naturally knows *where* to extrapolate to, e.g., by incrementing the for-loop range. This contrasts with the baselines that either require the user to specify which pixels to extrapolate (PatchMatch, Image Quilting,

Partial Images (Input)	PatchMatch	Image Quilting	GatedConv	Non-Stationary	PG-IM (Ours)
	Non-Learning-Based		**Learning-Based**		
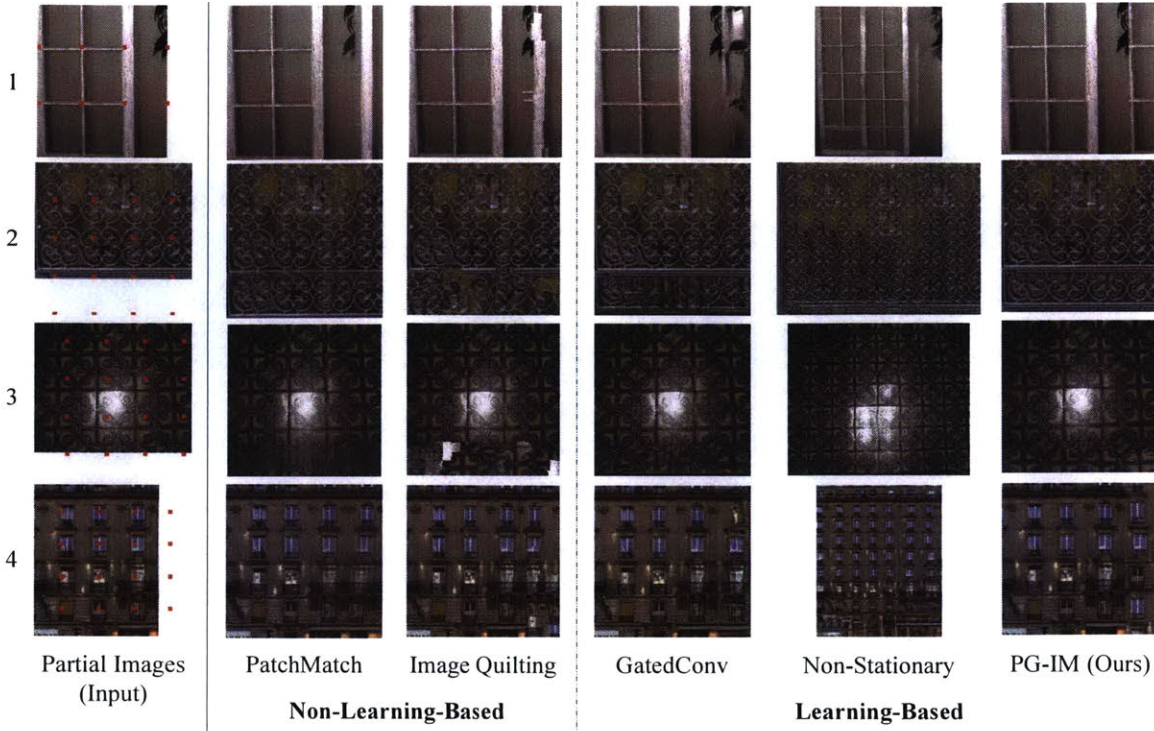
Figure 10-7: **Extrapolation results by PG-IM and the baselines.** The white pixels in the leftmost column indicate the pixels to be extrapolated. PG-IM generates realistic images while preserving global regularity. In contrast, GatedConv fails to capture the regularity; Non-Stationary does not preserve the original image contents; PatchMatch tends to generate blurry images in smoothing the transition; Image Quilting does not guarantee the global structure gets preserved.

and GatedConv), or simply extrapolate to every possible direction (Non-Stationary). Knowing where to extrapolate is particularly crucial for images where the objects do not scatter all over. Take the pieces of cereal in Figure 10-1B as an example. PG-IM reasons about the global structure that the pieces of cereal form, decides where to extrapolate to by relaxing its program conditions, and finally extrapolates a new row.

As PatchMatch greedily "copies from" patches with the most similar context, certain pixels may come from different patches, therefore producing blurry extrapolation results (Images 1, 3, and 4). Learning from large-scale image datasets, GatedConv fails to capture the repeated patterns specific to each individual image, thus generating patterns that do not connect to the image boundary consistently (Images 2 and 3). Non-Stationary treats the entire image as consisting of only patterns of interest and expands the texture along all four directions; artifacts show up when the image contains more than the texture (bottom of Image 4). Also interesting is that Non-Stationary can be viewed as a super-resolution algorithm, in the sense that it is interpolating among the replicated objects. As the rightmost column shows, during extrapolation, PG-IM produces realistic and sharp patches (Image 1), preserves the images' global regularity, and connects consistently to the image boundary (Images 2-4).
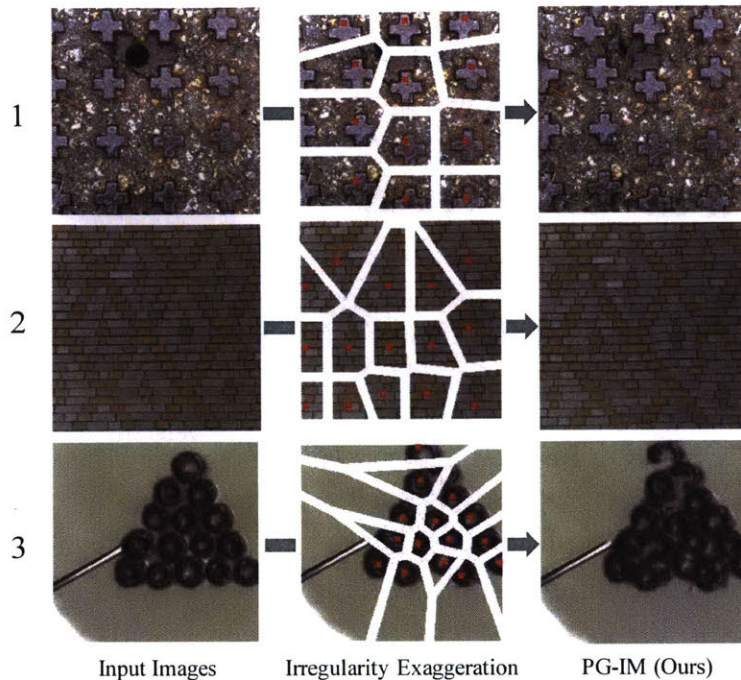
Figure 10-8: **PG-IM enables automated and semantic-aware irregularity exaggeration.** By comparing the centroids of the detected objects and the ones reconstructed by the program, we can measure and exaggerate the structural irregularity of input images.

10.5.5 Image Regularity Editing

With a program describing the image's *ideal* global regularity, PG-IM is able to exaggerate imperfections in the global regularity by magnifying the discrepancy between what the program depicts and the detected object centroids. A similar task has been discussed by Dekel et al. [2015]. In Figure 10-8, we magnify the displacement vectors between the program-provided and detected centroids by two, and shift the Voronoi cells together with their respective centroids, leaving missing values among the cells. An NPN then fills in the gaps by recurrent inpainting.

10.5.6 Attribute Regularity

Beyond using for-loops and if-conditions to capture the global regularity of objects, PG-IM can also reason about the regularity of object appearance variations (i.e., the *attribute* regularity). Our model automatically clusters objects into groups. Beyond knowing where to extrapolate to, with the attribute regularity described by the program, our NPNs generate new pixels from only patches of the correct attributes.

Figure 10-9 illustrates this idea. We show the image extrapolation results on images with attribute regularities, and compare PG-IM with a variant that does not consider object attributes, as well as a strong baseline: PatchMatch. Without explicit modeling of object attributes, the color of the new objects generated by PG-IM

215

| Partial Image (Input) | PG-IM w/o Attributes | PG-IM (Ours) | PatchMatch | Partial Image (Input) | PG-IM w/o Attributes | PG-IM (Ours) | PatchMatch |
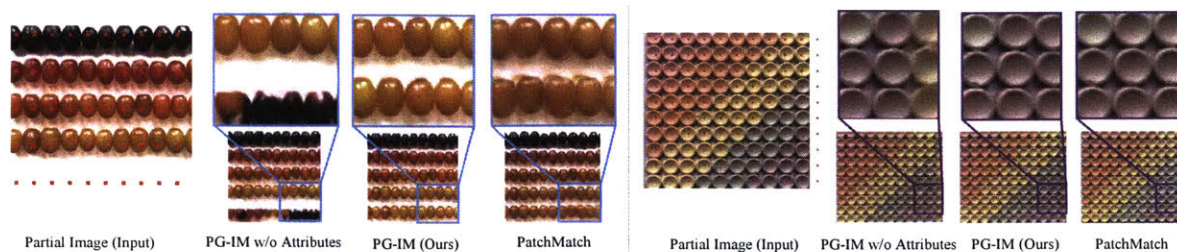
Figure 10-9: **PG-IM can reason about the attribute regularity of images**, which supports object appearance–aware image extrapolation. PG-IM w/o Attributes denotes a variant of PG-IM that does not include attributes. See the main text for detailed analysis and comparison.

without attributes fails to preserve the global attribute regularity. Meanwhile, due to the existence of objects with similar colors, PatchMatch mixes up two different colors, resulting in blurry output patches (Figure 10-9, left) or extrapolation results that break the global attribute regularity (the central object in the top-right row of zoom-in windows in Figure 10-9 should be purple, not green).

10.6 Discussion

In this chapter, we have introduced a neuro-symbolic approach to describing and manipulating natural images with repeated patterns. It combines the power of program induction—as symbolic tools for describing repetition, symmetry, and attributes—and deep neural networks—as powerful image generative models. PG-IM supports various tasks including image inpainting, extrapolation, and regularity editing.

Our results also suggest multiple future directions. First, variations in object appearance are currently handled as discrete properties. We leave the interpretation of continuous attributes, such as the color spectrum in Figure 10-9, as future work. Combining regularity inference and data-driven approaches for scene semantics is also a meaningful direction. As an example, from a facade image of merely windows, humans can extrapolate the image by not only adding windows, but also doors at the bottom and roofs at the top. Finally, the representational power of PG-IM is limited by the DSL. PG-IM currently does not generalize to unseen patterns, such as rotational patterns. Future work includes adding a more flexible DSL and discovering new patterns or the DSL itself from data.

Chapter 11

Conclusion

In this dissertation, we have discussed the general paradigm of integrating machine learning with simulation engines, as well as concrete realizations with graphics, physics, and program execution engines across multiple areas (vision, learning, NLP, symbolic reasoning, rule learning and program induction, planning, and control). In the era of big data, large computing resources, and advanced learning algorithms, these once separated areas across computer science have begun to reintegrate. I believe we should now take an more integrative view toward these areas and actively explore their interactions for a more general AI landscape. Below, I outline a few open challenges and future directions.

One such direction is to achieve more fundamental integration of perception, reasoning, and planning. While most computational models have treated them as disjoint modules, we observe that having them communicate with each other facilitates model design and leads to better performance [Sun et al., 2019, Janner et al., 2019]. The key factor that connects these modules is *belief space*—our belief of partially observable, uncertain world states. AI researchers have been integrating perception and planning in belief space [Kaelbling and Lozano-Pérez, 2013]. Building upon these insightful ideas, it becomes possible to explore interactive perception by integrating both classic and modern AI tools: probabilistic inference for managing uncertainty; causal and counterfactual reasoning in generative models, for explainability, imagination, and planning; and hierarchical inference for learning to learn, so knowledge builds progressively. Also, discovering the cognitive and neural basis of belief space perception, reasoning, and planning will be of significant value for understanding human intelligence.

Another direction is to integrate symbolic priors with deep representation learning via program synthesis for concept and structure discovery. Neuro-symbolic methods enjoy both the recognition power from neural nets and the combinatorial generalization from symbolic structure; therefore, they have great potential in scaling up current intelligent systems to large-scale, complex physical scenes in the real life, for which pure

bottom-up, data-driven models cannot work well due to the exponentially increasing complexity. As a preliminary study, our recent research has shown that they can learn to discover concepts, meta-concepts (e.g., synonyms), and use them to answer questions, all from natural supervision (images and question-answer pairs) as humans do [Mao et al., 2019a, Yi et al., 2018, Han et al., 2019]. Future work includes exploring the use of symbolic languages for knowledge representation and abstraction, and how they can be integrated with deep networks' pattern recognition abilities for flexible physical scene understanding.

Beyond physical objects and scenes, we need computational models that understand an agent's goals, beliefs, intentions, and theory of mind, and use these knowledge for planning and problem solving, drawing inspiration from *intuitive psychology*. While we have been inferring physical object properties from interactions, can we also build computational models that, just like 10-month-old infants [Liu et al., 2017], infer object values in agents' beliefs from their behaviors? Research along this direction would be incredible valuable for developing human-like and human-centered autonomous systems.

More generally, we should connect computer science with other disciplines such as cognitive science, neuroscience, social science, linguistics, and mechanical engineering. Research in cognitive science and neuroscience has been offering intuitions for AI researchers for decades; now we're entering a new stage, where contemporary research in intelligent systems or computer science in general may help us better understand human intelligence [Fischer et al., 2016, Yamins et al., 2014]. Our recent research has suggested that computational models that combine bottom-up neural recognition networks and top-down simulation engines shed light on understanding cognitive and neural processes in the brain [Yildirim et al., 2018b, Zhang et al., 2016]. Much more work needs to be done in these areas. With the right integration of probabilistic inference methods, deep learning, and generative models, we can build more powerful computational models for both neural activities and cognitive, behavioral data. The same applies to developmental psychology. We have been constructing benchmarks that carefully replicate classic developmental psychology experiments for evaluating modern computational models [Smith et al., 2019]. Our research compares and contrasts human and artificial intelligence on understanding *core knowledge*—knowledge about object permanence, solidity, continuity, and containment, and concepts such as gravity and momentum [Spelke, 2000]. Such interdisciplinary research deepens our understanding of multiple research areas and suggests future research topics.

We're in a unique and exciting time: the development of data, hardware, and algorithms (e.g., deep networks, graphical models, probabilistic programs) has enabled more flexible and expressive computational models. For the next decade, I believe building structured models for machine physical scene understanding, as well as inves-

tigating its connection with perception, reasoning, and interaction, will be incredibly valuable and essential for developing computational systems that contribute to broad fundamental and practical research across disciplines.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Data and Model Details
for Generalizable Reconstruction

In Appendix A, we supply additional details for the Generalizable Reconstruction (GenRe) model introduced in Chapter 3.

A.1 Data Preparation

We describe how we prepare our data for network training and testing.

Scene setup. The camera is fully specified by its azimuth and elevation angles, as its distance from the object is fixed at 2.2, its up vector is always the world $+y$ axis, and it always looks at the world origin, where the object center lies. Focal length of the camera is fixed at 50mm on a 35mm film. Depth values are measured from the camera center (i.e., ray depth), rather than from the image plane.

Rendering. We render 20 images of random views (or 200 fixed views in the viewpoint study) for each object of interest. To boost the rendering realism and diversity, we use three types of background: the SUN backgrounds [Xiao et al., 2010], high-dynamic-range environment lighting crawled on the web, and pure white backgrounds. Specifically, for each rendering, we randomly sample a background type and then a random instance of that type. We use Mitsuba [Jakob, 2010] for all of our rendering.

Data augmentation. For network training, we augment our RGB images with three techniques: color jittering, adding lighting noise, and color normalization. In color jittering, we multiply the brightness, contrast, and saturation, one by one in a random order, by a random factor uniformly sampled from [0.6, 1.4]. We then add AlexNet-style lighting noise [Krizhevsky et al., 2012] and perform the standard color normalization with statistics derived from the ImageNet dataset [Deng et al., 2009].

A.2 Model Details

We implement all of our networks in PyTorch 0.3.

A.2.1 Single-View Depth Estimator

We adopt an encoder-decoder architecture, where the encoder is a ResNet-18 [He et al., 2016] that encodes a 256×256 RGB image into 512 feature maps of size 1×1. Specifically, it consists of, in a sequential order,

```
Conv2d(3, 64, kernel=7, stride=2, pad=3)
BatchNorm2d(64, eps=1e-05, momentum=0.1)
ReLU(inplace)
MaxPool2d(kernel=3, stride=2, pad=1, dilation=1)
BasicBlock(
   (conv1): Conv2d(64, 64, kernel=3, stride=1, pad=1)
   (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1)
   (relu): ReLU(inplace)
   (conv2): Conv2d(64, 64, kernel=3, stride=1, pad=1)
   (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
BasicBlock(
   (conv1): Conv2d(64, 64, kernel=3, stride=1, pad=1)
   (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1)
   (relu): ReLU(inplace)
   (conv2): Conv2d(64, 64, kernel=3, stride=1, pad=1)
   (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
BasicBlock(
   (conv1): Conv2d(64, 128, kernel=3, stride=2, pad=1)
   (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1)
   (relu): ReLU(inplace)
   (conv2): Conv2d(128, 128, kernel=3, stride=1, pad=1)
   (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1)
   (downsample):
     Conv2d(64, 128, kernel=1, stride=2)
     BatchNorm2d(128, eps=1e-05, momentum=0.1)
)
BasicBlock(
   (conv1): Conv2d(128, 128, kernel=3, stride=1, pad=1)
   (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1)
   (relu): ReLU(inplace)
   (conv2): Conv2d(128, 128, kernel=3, stride=1, pad=1)
   (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1)
)
BasicBlock(
   (conv1): Conv2d(128, 256, kernel=3, stride=2, pad=1)
```

```
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1)
    (relu): ReLU(inplace)
    (conv2): Conv2d(256, 256, kernel=3, stride=1, pad=1)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1)
    (downsample):
      Conv2d(128, 256, kernel=1, stride=2)
      BatchNorm2d(256, eps=1e-05, momentum=0.1)
)
BasicBlock(
    (conv1): Conv2d(256, 256, kernel=3, stride=1, pad=1)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1)
    (relu): ReLU(inplace)
    (conv2): Conv2d(256, 256, kernel=3, stride=1, pad=1)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1)
)
BasicBlock(
    (conv1): Conv2d(256, 512, kernel=3, stride=2, pad=1)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1)
    (relu): ReLU(inplace)
    (conv2): Conv2d(512, 512, kernel=3, stride=1, pad=1)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1)
    (downsample):
      Conv2d(256, 512, kernel=1, stride=2)
      BatchNorm2d(512, eps=1e-05, momentum=0.1)
)
BasicBlock(
    (conv1): Conv2d(512, 512, kernel=3, stride=1, pad=1)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1)
    (relu): ReLU(inplace)
    (conv2): Conv2d(512, 512, kernel=3, stride=1, pad=1)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1)
).
```

The decoder is a mirrored version of the encoder, with all convolution layers replaced by transposed convolution layers. Additionally, we adopt the U-Net structure [Ronneberger et al., 2015] by feeding the intermediate outputs of each encoder block to the corresponding decoder block. The decoder outputs an image of *relative* depth values in the original view at the same resolution as input. Specifically, the decoder comprises

```
RevBasicBlock(
    (deconv1): ConvTranspose2d(512, 256, kernel=3, stride=1, pad=1)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1)
    (relu): ReLU(inplace)
    (deconv2): ConvTranspose2d(256, 256, kernel=3, stride=2, pad=1, out_pad=1)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1)
    (upsample):
```

```
      ConvTranspose2d(512, 256, kernel=1, stride=2, out_pad=1)
      BatchNorm2d(256, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
  (deconv1): ConvTranspose2d(256, 256, kernel=3, stride=1, pad=1)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1)
  (relu): ReLU(inplace)
  (deconv2): ConvTranspose2d(256, 256, kernel=3, stride=1, pad=1)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
  (deconv1): ConvTranspose2d(512, 128, kernel=3, stride=1, pad=1)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1)
  (relu): ReLU(inplace)
  (deconv2): ConvTranspose2d(128, 128, kernel=3, stride=2, pad=1, out_pad=1)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1)
  (upsample):
      ConvTranspose2d(512, 128, kernel=1, stride=2, out_pad=1)
      BatchNorm2d(128, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
  (deconv1): ConvTranspose2d(128, 128, kernel=3, stride=1, pad=1)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1)
  (relu): ReLU(inplace)
  (deconv2): ConvTranspose2d(128, 128, kernel=3, stride=1, pad=1)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
  (deconv1): ConvTranspose2d(256, 64, kernel=3, stride=1, pad=1)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1)
  (relu): ReLU(inplace)
  (deconv2): ConvTranspose2d(64, 64, kernel=3, stride=2, pad=1, out_pad=1)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1)
  (upsample):
      ConvTranspose2d(256, 64, kernel=1, stride=2, out_pad=1)
      BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
  (deconv1): ConvTranspose2d(64, 64, kernel=3, stride=1, pad=1)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1)
  (relu): ReLU(inplace)
  (deconv2): ConvTranspose2d(64, 64, kernel=3, stride=1, pad=1)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
  (deconv1): ConvTranspose2d(128, 64, kernel=3, stride=1, pad=1)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1)
```

```
   (relu): ReLU(inplace)
   (deconv2): ConvTranspose2d(64, 64, kernel=3, stride=1, pad=1)
   (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1)
   (upsample):
     ConvTranspose2d(128, 64, kernel=1, stride=1)
     BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
RevBasicBlock(
   (deconv1): ConvTranspose2d(64, 64, kernel=3, stride=1, pad=1)
   (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1)
   (relu): ReLU(inplace)
   (deconv2): ConvTranspose2d(64, 64, kernel=3, stride=1, pad=1)
   (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
ConvTranspose2d(128, 64, kernel=3, stride=2, pad=1, out_pad=1)
BatchNorm2d(64, eps=1e-05, momentum=0.1)
ReLU(inplace)
ConvTranspose2d(64, 1, kernel=8, stride=2, pad=3, out_pad=0).
```

Relative depth values provided by the predicted depth images are insufficient for conversions to spherical maps or voxels, as there are still two degrees of freedom undetermined: the minimum and maximum (or scale). Therefore, we have an additional branch decoding, also from the 512 feature maps, the minimum and maximum of the depth values. Specifically, it contains

```
Conv2d(512, 512, kernel=2, stride=2)
Conv2d(512, 512, kernel=4, stride=1)
ViewAsLinear()
Linear(in_features=512, out_features=256, bias=True)
BatchNorm1d(256, eps=1e-05, momentum=0.1)
ReLU(inplace)
Linear(in_features=256, out_features=128, bias=True)
BatchNorm1d(128, eps=1e-05, momentum=0.1)
ReLU(inplace)
Linear(in_features=128, out_features=2, bias=True).
```

Using the pretrained ResNet-18 as our network initialization, we then train this network with supervision on both the depth image (relative) and the minimum as well as maximum values. Under this setup, our network predicts effectively the absolute depth values of the input view, which allows us to project these depth values to the spherical representation or voxel grid.

This network was trained with a batch size of 4. We used Adam [Kingma and Ba, 2015] with a learning rate of $1e-3$, $\beta_1 = 0.5$, and $\beta_2 = 0.9$ for optimization.

A.2.2 Spherical Map Inpainting Network

Our inpainting network shares the same architecture as the single-view depth estimator. To mimic the boundary conditions of spherical maps, we use replication padding for the vertical dimension (elevation) and periodic padding for the horizontal dimension (azimuth). The padding size is 16 for all dimensions.

This network was trained with a batch size of 4. We used Adam with a learning rate of $1e-4$, $\beta_1 = 0.5$, and $\beta_2 = 0.9$ for optimization.

A.2.3 Voxel Refinement Network

Our voxel refinement network adopts the U-Net structure [Ronneberger et al., 2015] and uses a sequence of 3D convolution and transposed convolution layers. The input tensor is of shape BatchSize$\times2\times128\times128\times128$, where one channel contains voxels projected from the predicted original-view depth map, and the other contains voxels projected from the inpainted spherical map. After fusion, the output tensor is of shape BatchSize$\times1\times128\times128\times128$. Specifically, the network is structured as

```
Unet(
    Conv3d_block(
        Conv3d(2, 20, kernel=8, stride=2, pad=3)
        BatchNorm3d(20, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Conv3d_block(
        Conv3d(20, 40, kernel=4, stride=2, pad=1)
        BatchNorm3d(40, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Conv3d_block(
        Conv3d(40, 80, kernel=4, stride=2, pad=1)
        BatchNorm3d(80, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Conv3d_block(
        Conv3d(80, 160, kernel=4, stride=2, pad=1)
        BatchNorm3d(160, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Conv3d_block(
        Conv3d(160, 320, kernel=4, stride=2, pad=1)
        BatchNorm3d(320, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Conv3d_block(
        Conv3d(320, 640, kernel=4, stride=1)
```

```
        BatchNorm3d(640, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    full_conv_block(
        Linear(in_features=640, out_features=640, bias=True)
        LeakyReLU(negative_slope=0.01)
    )
    Deconv3d_skip(
        ConvTranspose3d(1280, 320, kernel=4, stride=1)
        BatchNorm3d(320, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Deconv3d_skip(
        ConvTranspose3d(640, 160, kernel=4, stride=2, pad=1)
        BatchNorm3d(160, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Deconv3d_skip(
        ConvTranspose3d(320, 80, kernel=4, stride=2, pad=1)
        BatchNorm3d(80, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)

    )
    Deconv3d_skip(
        ConvTranspose3d(160, 40, kernel=4, stride=2, pad=1)
        BatchNorm3d(40, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Deconv3d_skip(
        ConvTranspose3d(80, 20, kernel=8, stride=2, pad=3)
        BatchNorm3d(20, eps=1e-05, momentum=0.1)
        LeakyReLU(negative_slope=0.01)
    )
    Deconv3d_skip(
        ConvTranspose3d(40, 1, kernel=4, stride=2, pad=1)
    )
).
```

This network was trained with a batch size of 4. We used Adam with a learning rate of 10^{-5}, $\beta_1 = 0.5$, and $\beta_2 = 0.9$ for optimization.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Details and Results
for Neuro-Symbolic Concept Learners

In Appendix B, we supply additional details for the Neuro-Symbolic Concept Learner (NS-CL) introduced in Chapter 8.

B.1 The CLEVR Domain-Specific Language

We first introduce the domain-specific language (DSL) designed for the CLEVR VQA dataset [Johnson et al., 2017a]. Table B.1 shows the available operations in the DSL, while Table B.2 explains the type system.

We note that some function takes Object as its input instead of ObjectSet. These functions require the uniqueness of the referral object. For example, to answer the question "What's the color of the red object?", there should be one and only one red object in the scene. During the program execution, the input object set will be implicitly cast to the single object (if the set is non-empty and there is only one object in the set). Such casting is named Unique in related work [Johnson et al., 2017b].

B.2 Semantic Parsing

As shown in Appendix B.1, a program can be viewed as a hierarchy of operations which take concepts as their parameters. Thus, NS-CL generates the hierarchies of latent programs in a sequence to tree manner [Dong and Lapata, 2016]. The semantic parser adopts an encoder-decoder architecture, which contains four neural modules: (1) a bidirectional GRU encoder IEncoder [Cho et al., 2014] to encode an input question into a fixed-length embedding, (2) an operation decoder OpDecoder that determines the operation tokens, such as Filter, in the program based on the sentence embedding, (3) a concept decoder ConceptDecoder that selects concepts appeared in the input question as the parameters for certain operations (e.g., Filter takes an object-level concept parameter while Query takes an attribute), and (4) a set of output encoders {OEncoder$_i$} which encode the decoded operations by OpDecoder

Operation	Signature	Semantics
Scene	() $\longrightarrow$ ObjectSet	Return all objects in the scene.
Filter	(ObjectSet, ObjConcept) $\longrightarrow$ ObjectSet	Filter out a set of objects having the object-level concept (e.g., red) from the input object set.
Relate	(Object, RelConcept) $\longrightarrow$ ObjectSet	Filter out a set of objects that have the relational concept (e.g., left) with the input object.
AERelate	(Object, Attribute) $\longrightarrow$ ObjectSet	(Attribute-Equality Relate) Filter out a set of objects that have the same attribute value (e.g., same color) as the input object.
Intersection	(ObjectSet, ObjectSet) $\longrightarrow$ ObjectSet	Return the intersection of two object sets.
Union	(ObjectSet, ObjectSet) $\longrightarrow$ ObjectSet	Return the union of two object sets.
Query	(Object, Attribute) $\longrightarrow$ ObjConcept	Query the attribute (e.g., color) of the input object.
AEQuery	(Object, Object, Attribute) $\longrightarrow$ Bool	(Attribute-Equality Query) Query if two input objects have the same attribute value (e.g., same color).
Exist	(ObjectSet) $\longrightarrow$ Bool	Query if the set is empty.
Count	(ObjectSet) $\longrightarrow$ Integer	Query the number of objects in the input set.
CLessThan	(ObjectSet, ObjectSet) $\longrightarrow$ Bool	(Counting LessThan) Query if the number of objects in the first input set is less than the one of the second set.
CGreaterThan	(ObjectSet, ObjectSet) $\longrightarrow$ Bool	(Counting GreaterThan) Query if the number of objects in the first input set is greater than the one of the second set.
CEqual	(ObjectSet, ObjectSet) $\longrightarrow$ Bool	(Counting Equal) Query if the number of objects in the first input set is the same as the one of the second set.

Table B.1: **All operations in the domain-specific language for the CLEVR VQA dataset**

and output the latent embedding for decoding the next operation. The operation decoder, the concept decoder, and the output encoders work jointly and recursively to generate the hierarchical program layout. Algorithm 1 illustrates the algorithmic outline of the semantic parser.

The function *parse* takes two inputs: the current decoding state f and all concepts appeared in the question, as a set $\{c_i\}$. The parsing procedure begins with encoding

Type	Example	Semantics
ObjConcept	`Red`, `Cube`, etc.	Object-level concepts.
Attribute	`Color`, `Shape`, etc.	Object-level attributes.
RelConcept	`Left`, `Front`, etc.	Relational concepts.
Object	●	A single object in the scene.
ObjectSet	{●,■}	A set of objects in the scene.
Integer	$0, 1, 2, \cdots$	A single integer.
Bool	`True`, `False`	A single boolean value.

Table B.2: **The type system of the domain-specific language for the CLEVR VQA dataset**

Algorithm 1: The String-to-Tree Semantic Parser

Function parse(f, $\{c_i\}$):

  $program \leftarrow$ EmptyProgram();
  $program.op \leftarrow$ OpDecoder(f);
  if $program.op$ requires a concept parameter **then**
  | $program.concept \leftarrow$ ConceptDecoder($f, \{c_i\}$);

  for $i = 0, 1, \cdots$ number of non-concept inputs of $program.op$ **do**
  | $program.input[i] \leftarrow$ parse (OEncoder$_i$(f, $program.op$) ,$\{c_i\}$);
  return $program$

---

the input question by IEncoder as f_0, extracting the concept set $\{c_i\}$ from the input question, and invoking $parse(f_0, \{c_i\})$.

The concept set $\{c_i\}$ is extracted using hand-coded rules. We assume that each concept (including object-level concepts, relational concepts, and attributes) is associated with a single word in the question. For example, the word "red" is associated with the object-level concept `Red`, while the word "shape" is associated with the attribute `Shape`. Informally, we call these words *concept words*. For a given question Q, the corresponding concept set $\{c_i\}$ is composed of all occurrences of the *concept words* in Q. The set of *concept words* is known for the CLEVR dataset. For natural language questions, one could run POS tagging to find all *concept words* [Andreas et al., 2016, Schuster et al., 2015]. We leave the automatic discovery of concept words as a future work [Gauthier et al., 2018]. We use the word embedding of the *concept words* as the representation for the concepts $\{c_i\}$. Note that, these "concept embeddings" are only for the program parsing. The visual module has separate concept embeddings for aligning object features with concepts in the visual-semantic space.

We now delve into the main function $parse(f, \{c_i\})$: we first decode the root operation op of the hierarchy by OpDecoder(f). If op requires a concept parameter (an object-level concept, a relational concept, or an attribute), ConceptDecoder will be invoked to choose a concept from all concepts $\{c_i\}$. Assuming op takes two non-concept

inputs (e.g., the operation `Intersection` takes two object sets as its input), there will be two branches for this root node. Thus, two output encoders `OEncoder_0` and `OEncoder_1` will be applied to transform the current state f into two sub-states f_1 and f_2. *parse* will be recursively invoked based on f_1 and f_2 to generate the two branches respectively. In the DSL, the number of non-concept inputs for any operation is at most 2.

In our implementation, the input encoder `IEncoder` first maps each word in the question into an embedding space. The word embeddings are composed of two parts: a randomly initialized word embedding of dimension 256 and a positional embedding of dimension 128 [Gehring et al., 2017]. For a *concept word*, its word embedding only depends on which type it belongs to (i.e., object-level, relational or attribute). Thus, after being trained on a fixed dataset, the semantic parser can parse questions with novel (unseen) *concept words*. The sequence of word embeddings is then encoded by a two-layer GRU with a hidden dimension of 256×2 (bidirectional). The function *parse* starts from the last hidden state of the GRU, and works recursively to generate the hierarchical program layout. Both `OpDecoder` and `ConceptDecoder` are feed-forward networks. `ConceptDecoder` performs attentions over the representations of all concepts $\{c_i\}$ to select the concepts. Output encoders `OEncoder_0` and `OEncoder_1` are implemented as GRU cells.

Another pre-processing of the sentence is to group consecutive object-level *concept words* into a group and treat them together as a single concept, inspired by the notion of "noun phrases" in natural languages. The computational intuition behind this grouping is that, the latent programs of CLEVR questions usually contain multiple consecutive `Filter` tokens. During the program parsing and execution, we aim to fuse all such `Filters` into a single `Filter` operation that takes multiple concepts as its parameter.

Running example. As a running example, consider again the question "What is the color of the cube right of the red matte object?". We first process the sentence (by rules) as: "What is the <Attribute 1 (color)> of the <(ObjConcept 1 (cube)> <RelConcept 1 (right)> of the <ObjConcept 2 (red matte object)>?". The expected parsing result of this sentence is:

```
Query(<Attribute 1>,
      Filter(<ObjConcept 1>,
          Relate(<RelConcept 1>,
              Filter(<ObjConcept 2>, Scene)
          )
      )
).
```

The semantic parser encode the word embeddings with `IEncoder`. The last hidden

Step	Inputs	Outputs	Recursive Invocation
1	f_0	OpDecoder(f_0) → Query; ConceptDecoder(f_0) →< Attribute 1 >; OEncoder$_0$(f_0, Query) → f_1	$parse(f_1)$
2	f_1	OpDecoder(f_1) → Filter; ConceptDecoder(f_1) →< ObjConcept 1 >; OEncoder$_0$(f_1, Filter) → f_2	$parse(f_2)$
3	f_2	OpDecoder(f_2) → Relate; ConceptDecoder(f_2) →< RelConcept 1 >; OEncoder$_0$(f_2, Relate) → f_3	$parse(f_3)$
4	f_3	OpDecoder(f_3) → Filter; ConceptDecoder(f_3) →< ObjConcept 2 >; OEncoder$_0$(f_3, Filter) → f_4	$parse(f_4)$
5	f_4	OpDecoder(f_3) → Scene;	(End of branch.)

Table B.3: **A step-by-step running example of the recursive parsing procedure.** The parameter $\{c_i\}$ is omitted for better visualization.

state of the GRU will be used as f_0. The word embeddings of the *concept words* form the set $\{c_i\}$ = {Attribute 1, ObjConcept 1, RelConcept 1, ObjConcept 2}. The function *parse* is then invoked recursively to generate the hierarchical program layout. Table B.3 illustrates the decoding process step-by-step.

B.3 Program Execution

In this section, we present the implementation of all operations listed in Table B.1. We start from the implementation of Object-typed and ObjectSet-typed variables. Next, we discuss how to classify objects by object-level concepts or relational concept, followed by the implementation details of all operations.

Object-typed and ObjectSet-typed variables. We consider a scene with n objects. An Object-typed variable can be represented as a vector Object of length n, where Object$_i$ ∈ $[0, 1]$ and $\sum_i$ Object$_i$ = 1. Object$_i$ can be interpreted as the probability that the i-th object of the scene is being referred to. Similarly, an ObjectSet-typed variable can be represented as a vector ObjectSet of length n, where ObjectSet$_i$ ∈ $[0, 1]$. ObjectSet$_i$ can be interpreted as the probability that the i-the object is in the set. To cast an ObjectSet-typed variable ObjectSet as an Object-typed variable Object (i.e., the Unique operation), we compute: Object = softmax(σ^{-1}(ObjectSet)), where $\sigma^{-1}(x) = \log(x/(1 - x))$ is the logit function.

Concept quantization. Denote o_i as the visual representation of the i-th object, OC the set of all object-level concepts, and A the set of all object-level attributes. Each object-level concept oc (e.g., Red) is associated with a vector embedding v^{oc}

233

and a L1-normalized vector b^{oc} of length $|A|$. b^{oc} represents which attribute does this object-level concept belong to (e.g., the concept Red belongs to the attribute Color). All attributes $a \in A$ are implemented as neural operators, denoted as u^a (e.g., $u^{\texttt{Color}}$). To classify the objects as being Red or not, we compute:

$$\Pr[\text{object } i \text{ is Red}] = \sigma \left(\sum_{a \in A} \left(b_a^{\texttt{Red}} \cdot \frac{\langle u^a(o_i), v_{\texttt{Red}} \rangle - \gamma}{\tau} \right) \right),$$

where σ denotes the Sigmoid function, $\langle \cdot, \cdot \rangle$ the cosine distance between two vectors. γ and τ are scalar constants for scaling and shifting the values of similarities. By applying this classifier on all objects we will obtain a vector of length n, denoted as ObjClassify(Red). Similarly, such classification can be done for relational concepts such as Left. This will result in an $n \times n$ matrix RelClassify(Left), where RelClassify(Left)$_{j,i}$ is the probability that the object i is left of the object j.

To classify whether two objects have the same attribute (e.g., have the same Color), we compute:

$$\Pr[\text{object } i \text{ has the same Color as object } j] = \sigma \left(\frac{\langle u^{\texttt{Color}}(o_i), u^{\texttt{Color}}(o_j) \rangle - \gamma}{\tau} \right),$$

We can obtain a matrix AEClassify(Color) by applying this classifier on all pairs of objects, where AEClassifier(Color)$_{j,i}$ is the probability that the object i and j have the same Color.

Quasi-symbolic program execution. Finally, Table B.4 summarizes the implementation of all operators. In practice, all probabilities are stored in the log space for better numeric stability.

B.4 Optimization of the Semantic Parser

To tackle the optimization in a non-smooth program space, we apply an off-policy program search process [Sutton et al., 2000] to facilitate the learning of the semantic parser. Denote $\mathbb{P}(s)$ as the set of all valid programs in the CLEVR DSL for the input question s. We want to compute the gradient w.r.t. Θ_s, the parameters of the semantic parser:

$$\nabla_{\Theta_s} = \nabla_{\Theta_s} \mathbb{E}_P[r \cdot \log \Pr[P]],$$

where $P \sim \text{SemanticParse}(s; \Theta_s)$. In REINFORCE, we approximate this gradient via Monte Carlo sampling.

An alternative solution is to exactly compute the gradient. Note that in the definition of the reward r, only the set of programs $\mathbb{Q}(s)$ leading to the correct answer will contribute to the gradient term. With the perception module fixed, the set $\mathbb{Q}$ can

Signature	Implementation
`Scene()` → *out*: ObjectSet	$out_i := 1$
`Filter`(*in*: ObjectSet, *oc*: ObjConcept) → *out*: ObjectSet	$out_i := \min(in_i, \text{ObjClassify}(oc)_i)$
`Relate`(*in*: Object, *rc*: RelConcept) → *out*: ObjectSet	$out_i := \sum_j (in_j \cdot \text{RelClassify}(rc)_{j,i})$
`AERelate`(*in*: Object, *a*: Attribute) → *out*: ObjectSet	$out_i := \sum_j (in_j \cdot \text{AEClassify}(a)_{j,i})$
`Intersection`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *out*: ObjectSet	$out_i := \min(in_i^{(1)}, in_i^{(2)})$
`Union`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *out*: ObjectSet	$out_i := \max(in_i^{(1)}, in_i^{(2)})$
`Query`(*in*: Object, *a*: Attribute) → *out*: ObjConcept	$\Pr[out = oc] := \sum_i in_i \cdot \dfrac{\text{ObjClassify}(oc)_i \cdot b_a^{oc}}{\sum_{oc'} \text{ObjClassify}(oc')_i \cdot b_a^{oc'}}$
`AEQuery`($in^{(1)}$: Object, $in^{(2)}$: Object, *a*: Attribute) → *b*: Bool	$b := \sum_i \sum_j (in_i^{(1)} \cdot in_j^{(2)} \cdot \text{AEClassify}(a)_{j,i})$
`Exist`(*in*: ObjectSet) → *b*: Bool	$b := \max_i in_i$
`Count`(*in*: ObjectSet) → *i*: Integer	$i := \sum_i in_i$
`CLessThan`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *b*: Bool	$b := \sigma\big((\sum_i in_i^{(2)} - \sum_i in_i^{(1)} - 1 + \gamma_c)/\tau_c\big)$
`CGreaterThan`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *b*: Bool	$b := \sigma\big((\sum_i in_i^{(1)} - \sum_i in_i^{(2)} - 1 + \gamma_c)/\tau_c\big)$
`CEqual`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *b*: Bool	$b := \sigma\big((-\lvert \sum_i in_i^{(1)} - \sum_i in_i^{(2)} \rvert + \gamma_c)/(\gamma_c \cdot \tau_c)\big)$

Table B.4: **All operations in the domain-specific language for the CLEVR VQA dataset.** $\gamma_c = 0.5$ and $\tau_c = 0.25$ are constants for scaling and shift the probability. During inference, one can quantify all operations as Yi et al. [2018].

be efficiently determined by an off-policy exhaustive search of all possible programs $\mathbb{P}(s)$. In the third stage of the curriculum learning, we search for the set $\mathbb{Q}$ offline based on the quantified results of concept classification and compute the exact gradient $\nabla\Theta_s$. An intuitive explanation of the off-policy search is that, we enumerate all possible programs, execute them on the visual representation, and find the ones leading to the correct answer. We use $\mathbb{Q}(s)$ as the "groundtruth" program annotation for the question, to supervise the learning, instead of running the Monte Carlo sampling-based REINFORCE.

Spurious program suppression. However, directly using $\mathbb{Q}(s)$ as the supervision by computing $\ell = \sum_{p \in \mathbb{Q}(S)} -\log \Pr(p)$ can be problematic, due to the spuriousness or the ambiguity of the programs. This comes from two aspects:

1) intrinsic ambiguity: two programs are different but equivalent. For example

P1: `AEQuery(Color,Filter(Cube),Filter(Sphere))` and

P2: `Exist(Filter(Sphere,AERelate(Color,Filter(Cube))))`

are equivalent.

2) extrinsic spuriousness: one of the program is incorrect, but also leads to the correct answer in a specific scene. For example,

P1: `Filter(Red,Relate(Left,Filter(Sphere)))` and

P2: `Filter(Red,Relate(Left,Filter(Cube)))`

may refer to the same red object in a specific scene. Motivated by the REINFORCE process, to suppress such spurious programs, we use the loss function:

$$\ell = \sum_{p \in \mathbb{Q}} \text{stop\_gradient}(\text{Pr}[p]) \cdot (-\log \text{Pr}[p]).$$

The corresponding gradient ∇_{Θ_s} is,

$$\nabla_{\Theta_s} = \sum_{p \in \mathbb{Q}} \text{Pr}[p] \cdot \nabla_{\Theta_s} (r \cdot \log \text{Pr}[P]) = \nabla_{\Theta_s} \left(\sum_{p \in \mathbb{Q}} r \cdot \text{Pr}[p] \right).$$

The key observation is that, given a sufficiently large set of scenes, a program can be identified as spurious if there exists at least one scene where the program leads to a wrong answer. As the training goes, spurious programs will get less update due to the sampling importance term $\text{Pr}[p]$ which weights the likelihood maximization term.

B.5 Curriculum Learning Setup

During the whole training process, we gradually add more visual concepts and more complex question examples into the model. Summarized in Figure 8-7(A), in general, the whole training process is split into 3 stages. First, we only use questions from lesson 1 to let the model learn object-level visual concepts. Second, we train the model to parse simple questions and to learn relational concepts. In this step, we freeze the neural operators and concept embeddings of object-level concepts. Third, the model gets trained on the full question set (lesson 3), learning to understand questions of different complexities and various format. For the first several iterations in this step, we freeze the parameters in the perception modules. In addition, during the training of all stages, we gradually increase the number of objects in the scene: from 3 to 10.

We select questions for each lesson in the curriculum learning by their depth of the

latent program layout. For eaxmple, the program "Query(Shape, Filter(Red, Scene))" has the depth of 3, while the program "Query(Shape, Filter(Cube, Relate(Left, Filter(Red, Scene))))" has the depth of 5. Since we have fused consecutive Filter operations into a single one, the maximum depth of all programs is 9 on the CLEVR dataset. We now present the detailed split of our curriculum learning lessons:

For lesson 1, we use only programs of depth 3. It contains three types of questions: querying an attribute of the object, querying the existence of a certain type of objects, count a certain type of objects, and querying if two objects have the same attribute (e.g., of the same color). These questions are almost about fundamental object-based visual concepts. For each image, we generate 5 questions of lesson 1.

For lesson 2, we use programs of depth less than 5, containing a number of questions regarding relations, such as querying the attribute of an object that is left of another object. We found that in the original CLEVR dataset, all Relate operations are followed by a Filter operation. This setup degenerates the performance of the learning of relational concepts such as Left. Thus, we add a new question template into the original template set: Count(Relate(· , Filter(· , Scene))) (e.g., "What's the number of objects that are left of the cube?"). For each image, we generate 5 questions of lesson 2.

For lesson 3, we use the full CLEVR question set.

Curriculum learning is crucial for the learning of our neuro-symbolic concept learner. We found that by removing the curriculum setup w.r.t. the number of object in the scenes, the visual perception module will get stuck at an accuracy that is similar to a random-guess model, even if we only use stage-1 questions. If we remove the curriculum setup w.r.t. the complexity of the programs, the joint training of the visual perception module and the semantic parser can not converge.

B.6   Ablation Study

We conduct ablation studies on the accuracy of semantic parsing, the impacts of the ImageNet pre-training of visual perception modules, the data efficiency of our model, and the usage of object-based representations.

B.6.1   Semantic Parsing Accuracy

We evaluate how well our model recovers the underlying programs of questions. Due to the intrinsic equivalence of different programs, we evaluate the accuracy of programs by executing them on the ground-truth annotations of objects. Invalid or ambiguous programs are also considered as incorrect. Our semantic parser archives $> 99.9\%$ QA accuracy on the validation split.

B.6.2 Impacts of the ImageNet Pre-training

The only extra supervision of the visual perception module comes from the pre-training of the perception modules on ImageNet [Deng et al., 2009]. To quantify the influence of this pre-training, we conduct ablation experiments where we randomly initialize the perception module following He et al. [2016]. The classification accuracies of the learned concepts almost remain the same except for **Shape**. The classification accuracy of **Shape** drops from 98.7 to 97.5 on the validation set while the overall QA accuracy on the CLEVR dataset drops to 98.2 from 98.9. We speculate that large-scale image recognition dataset can provide prior knowledge of shape.

B.6.3 Data Efficiency and Object-Based Representations

In this section, we study whether and how the number of training samples and feature representations affect the overall performance of various models on the CLEVR dataset. Specifically, we compare the proposed NS-CL against two strong baselines: TbD [Mascharka et al., 2018] and MAC [Hudson and Manning, 2018].

Baselines. For comparison, we implement two variants of the baseline models: TbD-Object and MAC-Object. Inspired by Anderson et al. [2018], instead of using a 2D convolutional feature map, TbD-Object and MAC-Object take a stack of object features as inputs, whose shape is $k \times d_{obj}$. k is the number of objects in the scene, and d_{obj} is the feature dimension for a single object. In our experiments, we fix $k = 12$ as a constant value. If there are fewer than 12 objects in the scene, we add "null" objects whose features are all-zero vectors.

We extract object features in the same way as NS-CL. Features are extracted from a pre-trained ResNet-34 network before the last residual block for a feature map with high resolution. For each object, its feature is composed of two parts: region-based (by RoI Align) and image-based features. We concatenate them to represent each object. As discussed, the inclusion of the representation of the full scene is essential for the inference of relative attributes such as size or spatial position on the CLEVR domain.

TbD and MAC networks are originally designed to use image-level attention for reasoning. Thus, we implement two more baselines: TbD-Mask and MAC-Mask. Specifically, we replace the original attention module on images with a mask-guided attention. Denotes the union of all object masks as M. Before the model applies the attention on the input image, we multiply the original attention map computed by the model with this mask M. The multiplication silences the attention on pixels that are not part of any objects.

Results. Table 8.5 summarizes the results. We found that TbD-Object and MAC-Object approach show inferior results compared with the original model. We attribute

this to the design of the network architectures. Take the `Relate` operation (e.g., finds all objects left of a specific object x) as an example. TbD uses a stack of dilated convolutional layers to propagate the attention from object x to others. In TbD-Object, we replace the stack of 2D convolutions by several 1D convolution layers, operating over the $k \times d_{obj}$ object features. This ignores the equivalence of objects (the order of objects should not affect the results). In contrast, MAC networks always use the attention mechanism to extract information from the image representation. This operation is invariant to the order of objects, but is not suitable for handling quantities (e.g., counting objects).

As for TbD-Mask and MAC-Mask, although the mask-guided attention does not improve the overall performance, we have observed noticeably faster convergence during model training. TbD-Mask and MAC-Mask leverage the prior knowledge of object masks to facilitate the attention. Such prior has also been verified to be effective in the original TbD model: TbD employs an attention regularization during training, which encourages the model to attend to smaller regions.

In general, NS-CL is more data-efficient than MAC networks and TbD. Recall that NS-CL answers questions by executing symbolic programs on the learned visual concepts. Only visual concepts (such as `Red` and `Left`) and the interpretation of questions (how to translate questions into executable programs) need to be learned from data. In contrast, both TbD and MAC networks need to additionally learn to execute (implicit or explicit) programs such as counting.

For the experiments on the full CLEVR training set, we split 3,500 images (5% of the training data) as the hold-out validation set to tune the hyperparameters and select the best model. We then apply this model to the CLEVR validation split and report the testing performance. Our model reaches an accuracy of 99.2% using the CLEVR training set.

B.7 Extending to Other Scene and Language Domains

B.7.1 Minecraft Dataset

We also extend the experiments to a new reasoning testbed: Minecraft worlds [Yi et al., 2018]. The Minecraft reasoning dataset differs from CLEVR in both visual appearance and question types. Figure B-1 gives an example instance from the dataset.

Setup. Following Yi et al. [2018], we generate 10,000 Minecraft scenes using the officially open-sourced tools by Wu et al. [2017b]. Each image contains 3 to 6 objects. The objects are chosen from 12 categories, with 4 different facing directions (front, back, left and right). They stand on a 2D plane.

Besides different 3D visual appearance and image contexts, the Minecraft reasoning
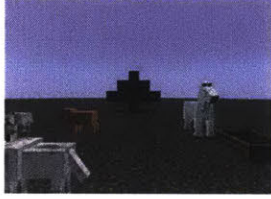
Q: What direction is the closest creature facing?
A: Left.
P: Query(Direction, FilterMost(Closest,
   Filter(Creature)
   ))

Figure B-1: **An example image and a related question-answering pair from the Minecraft dataset**

Model	Overall	Count	Exist	Belong	Query
NS-VQA	87.7	83.3	91.5	91.1	86.4
NS-CL	**93.3**	**91.3**	**95.6**	**93.9**	**94.3**

Table B.5: **Results on the Minecraft dataset.** Our model achieves comparable results on the Minecraft dataset with baselines trained by full program annotations.

dataset introduces two new types of reasoning operations. We add them to our domain-specific language:

1. `FilterMost`(ObjectSet, Concept) $\rightarrow$ ObjectSet: Given a set of objects, finds the "most" one. For example, `FilterMost`(`Closest`, set) locates the object in the input set that is cloest to the camera (e.g., what is the direction of the closest animal?)

2. `BelongTo`(Object, ObjectSet) $\rightarrow$ Bool: Query if the input object belongs to a set.

Results. Table B.5 summarizes the results and Figure B-4 shows sample execution traces. We compare our method against the NS-VQA baseline [Yi et al., 2018], which uses strong supervision for both scene representation (e.g., object categories and positions) and program traces. In contrast, our method learns both by looking at images and reading question-answering pairs. NS-CL outperforms NS-VQA by 5% in overall accuracy. We attribute the inferior results of NS-VQA to its derendering module. Because objects in the Minecraft world usually occlude with each other, the detected object bounding boxes are inevitably noisy. During the training of the derendering module, each detected bounding box is matched with one of the ground-truth bounding boxes and uses its class and pose as supervision. Poorly localized bounding boxes lead to noisy labels and hurt the accuracy of the derendering module. This further influences the overall performance of NS-VQA.

B.7.2 VQS Dataset

We conduct experiments on the VQS dataset [Gan et al., 2017]. VQS is a subset of the VQA 1.0 dataset [Antol et al., 2015]. It contains questions that can be visually

Q: Does this man have any pens on him?
A: Yes.
P: Exist(Filter(Man,
　　Relate(Have, Filter(Pen))
　　))
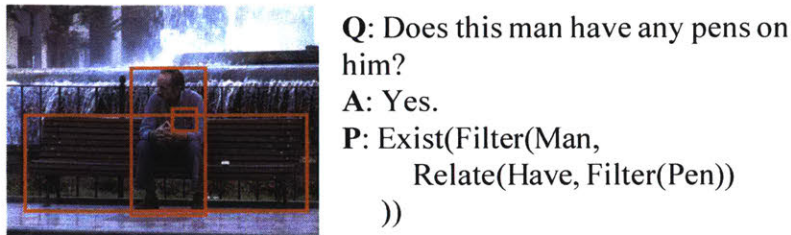
Figure B-2: **An example image from the VQS dataset.** The orange bounding boxes are object proposals. On the right, we show the original question and answer in natural language, as well as the latent program recovered by our parser. To answer this question, models are expected to attend to the man and his pen in the pocket.

grounded: each question is associated with multiple image regions, annotated by humans as necessary for answering the question.

Setup. All models are trained on the first 63,509 images of the training set, and tested on the test split. For hyper-parameter tuning and model selection, the rest 5,000 images from the training set are used for validation. We use the multiple-choice setup for VQA: the models choose their most confident answer from 18 candidate answers for each question.

To obtain the latent programs from natural languages, we use a pre-trained syntactic dependency parser [Andreas et al., 2016, Schuster et al., 2015] for extracting programs and concepts that need to be learned. A sample question and the program obtained by our parser is shown in Figure B-2. The concept embeddings are initialized by the bag of words (BoW) over the GloVe word embeddings [Pennington et al., 2014].

Baselines. We compare our model against two representative baselines: MLP [Jabri et al., 2016] and MAC [Hudson and Manning, 2018].

MLP is a standard baseline for visual-question answering, which treats the multiple-choice task as a ranking problem. For a specific candidate answer, a multi-layer perceptron (MLP) model is used to encode a tuple of the image, the question, and the candidate answer. The MLP outputs a score for each tuple, and the answer to the question is the candidate with the highest score. We encode the image with a ResNet-34 pre-trained on ImageNet and use BoW over the GloVe word embeddings for the question and option encoding.

We slightly modify the MAC network for the VQS dataset. For each candidate answer, we concatenate the question and the answer as the input to the model. The MAC model outputs a score from 0 to 1 and the answer to the question is the candidate with the highest score. The image features are extracted from the same ResNet-34 model.

Results. Table 8-9 summarizes the results. NS-CL achieves comparable results with the MLP baseline and the MAC network designed for visual reasoning. Our model also

241

brings transparent reasoning over natural images and language. Example execution traces generated by NS-CL are shown in Figure B-5. Besides, the symbolic reasoning process helps us to inspect the model and diagnose the error sources. See the caption for details.

B.8 Visualization of Execution Traces and Visual Concepts

Another appealing benefit is that our reasoning model enjoys full interpretability. Figure B-3, Figure B-4, and Figure B-5 show NS-CL's execution traces on CLEVR, Minecraft, and VQS, respectively. As a side product, our system detects ambiguous and invalid programs and throws out exceptions. As an example (Figure B-3), the question "What's the color of the cylinder?" can be ambiguous if there are multiple cylinders or even invalid if there are no cylinders.

Figure B-6 and Figure B-7 include qualitative visualizations of the concepts learned from the CLEVR and Minecraft datasets, including object categories, attributes, and relations. We choose samples from the validation or test split of each dataset by generating queries of the corresponding concepts. We set a threshold to filter the returned images and objects. For quantitative evaluations of the learned concepts on the CLEVR dataset, please refer to Table 8.3 and Table 8.7.

Example A.

Q: Do the cyan cylinder that is behind the gray cylinder and the gray cylinder have the same material?

Concept	Program	Result
Gray Cylinder	o→ Filter	
Behind	Relate	
Cyan Cylinder	Filter	
Gray Cylinder	o→ Filter	
Material	AEQuery	Yes (0.92)
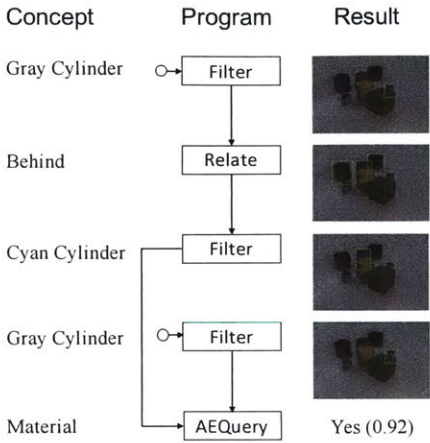
Example B.

Q: There is a small blue object that is to the right of the small red matte object; what shape is it?

Concept	Program	Result
Small Red Matte Object	o→ Filter	
Right	Relate	
Small Blue Object	Filter	
Shape	Query	Cube (0.85)
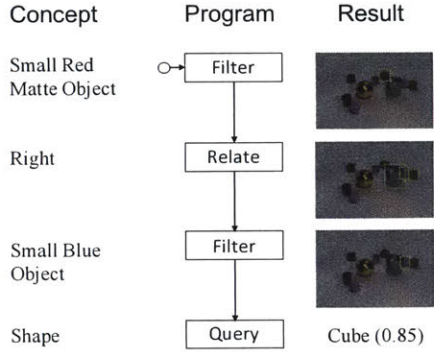
Example C. Failure Case

Q: What is the color of the big box left of the blue metal cylinder?

Concept	Program	Result
Blue Metal Cylinder	o→ Filter	No such object found!
	Execution Abort ✖	
Left	Relate	
Big Box	Filter	
Color	Query	

Color: Blue ✓
Material: Rubber ✗
Shape: Cylinder ✓
Size: Small ✓
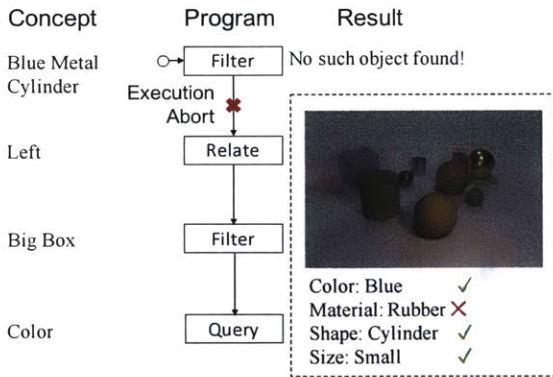
Example D. Ambiguous Program Case

Q: What is the color of the big metal object?

Concept	Program	Result
Big Metal Object	o→ Filter	
	Execution Abort ✖	
Color	Query	Ambiguous Referral!
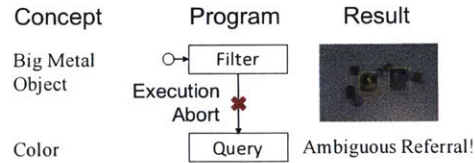
Figure B-3: **Exemplar execution trace generated by our Neuro-Symbolic Concept Learner on the CLEVR dataset.** Example A and B are successful executions that generate correct answers. In example C, the execution aborts at the first operator. To inspect the reason why the execution engine fails to find the corresponding object, we can read out the visual representation of the object, and locate the error source as the misclassification of the object material. Example D shows how our symbolic execution engine can detect invalid or ambiguous programs during the execution by performing sanity checks.

Example A.

Q: Are there <u>sheep</u> <u>near</u> the <u>wolf</u>?

Concept	Program	Result
Wolf	o→ Filter	
Near	Relate	
Sheep	Filter	
	Exist	No (0.98) ✓
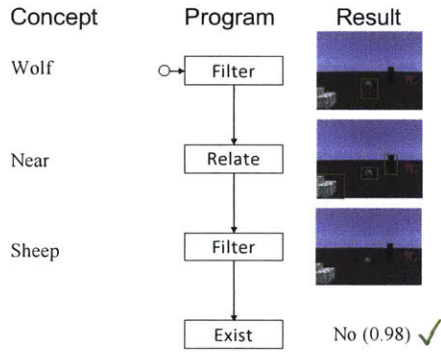
Example B.

Q: How many <u>animals</u> are <u>closer</u> to the camera than the <u>pig</u>?

Concept	Program	Result
Pig	o→ Filter	
Closer	Relate	
Animal	Filter	
	Count	1 ✓
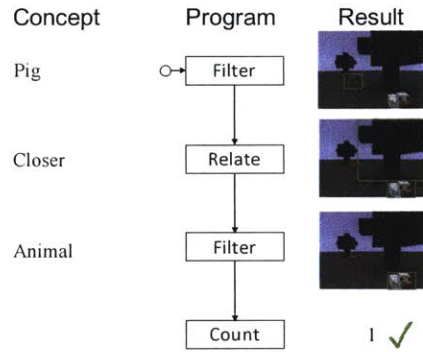
Example C.

Q: Which <u>direction</u> is the <u>closest animal</u> facing?

Concept	Program	Result
Animal	o→ Filter	
Closest	FilterMost	
Direction	Face	Right ✓
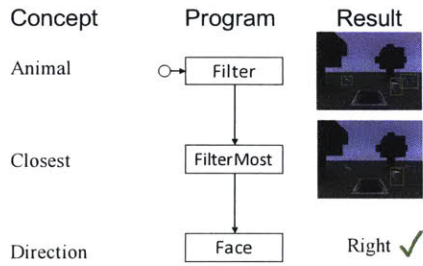
Example D. Failure Case

Q: How many <u>pigs</u> are there?

Concept	Program	Result
Pig	o→ Filter	
	Count	1 ✗

☐ Detected
☐ Missed
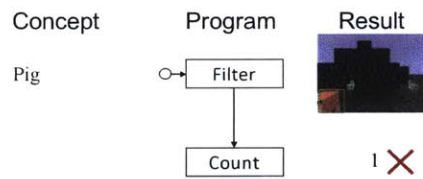
Figure B-4: **Exemplar execution trace generated by our Neuro-Symbolic Concept Learner on the Minecraft reasoning dataset.** Example A, B and C are successful execution. Example C demonstrates the semantics of the FilterMost operation. Example D shows a failure case: the detection model fails to detect a pig hiding behind the big tree.

244
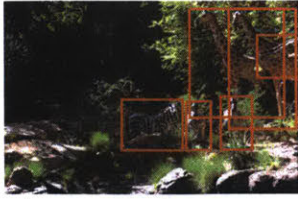
Example A.

Q: How many <u>zebras</u> are there?

Concept	Program	Result
Zebra	Filter	
	Count	3 ✓
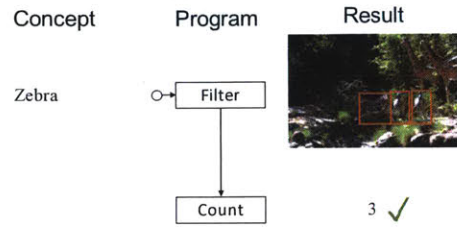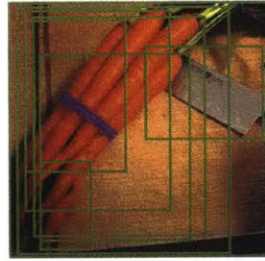
Example B.

Q: <u>What</u> is the <u>sharp</u> <u>object</u> <u>on</u> the <u>table</u>?

Concept	Program	Result
Table	Filter	
On	Relate	
Shape Object	Filter	
What	Query	Knife (0.85) ✓
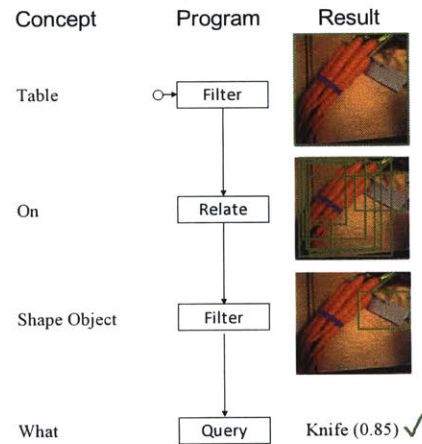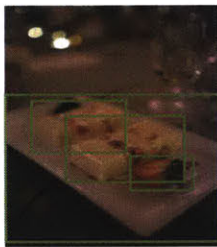
Example C.

Q: What <u>kind</u> of <u>desert</u> is <u>plated</u>?

Concept	Program	Result
Desert, Plated	Filter	
Kind	Query	Cake (0.68) ✓
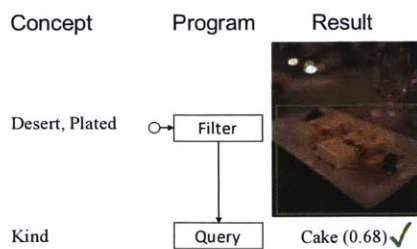
Example D.

Q: What are the kids doing?

Concept	Program	Result
Kids	Filter	
What	Query	Playing_Frisbee (0.70) ✗

Groundtruth: Playing_Baseball
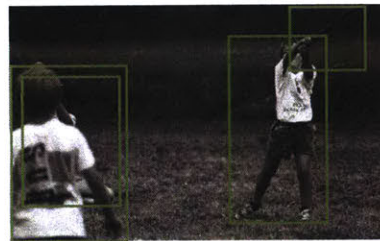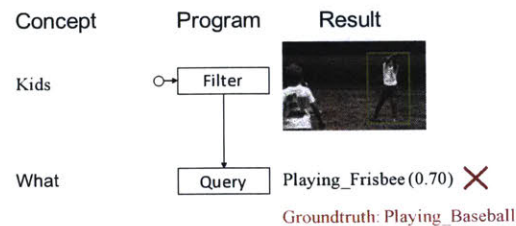
Figure B-5: **Exemplar execution trace generated by our Neuro-Symbolic Concept Learner on the VQS dataset.** Execution traces A and B shown in the figure leads to the correct answer to the question. Our model effectively learns visual concepts from data. The symbolic reasoning process brings transparent execution trace and can easily handle quantities (e.g., object counting in Example A). In Example C, although NS-CL answers the question correctly, it locates the wrong object during reasoning: a dish instead of the cake. In Example D, our model misclassifies the sport as frisbee.

Concept: <u>Cylinder</u>
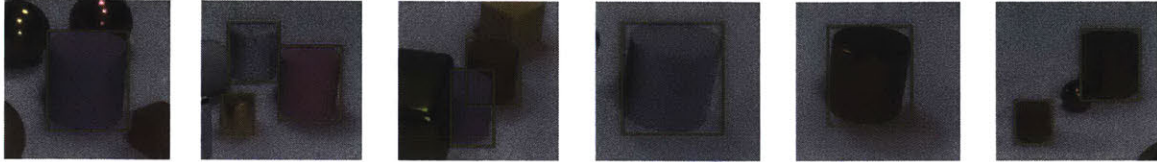
Concept: <u>Matte</u>
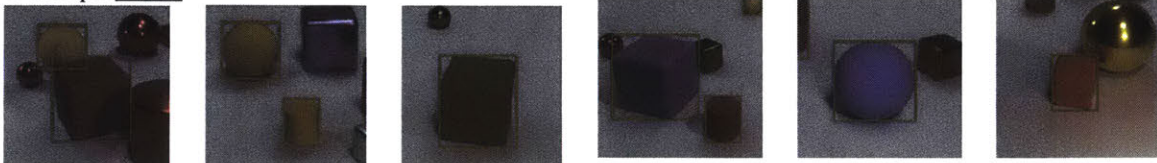
Concept: <u>Blue</u> <u>Sphere</u>
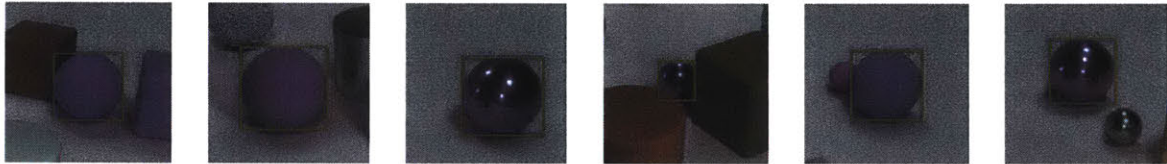
Concept: <u>Yellow</u> Object <u>Left</u> of <u>Cylinder</u>
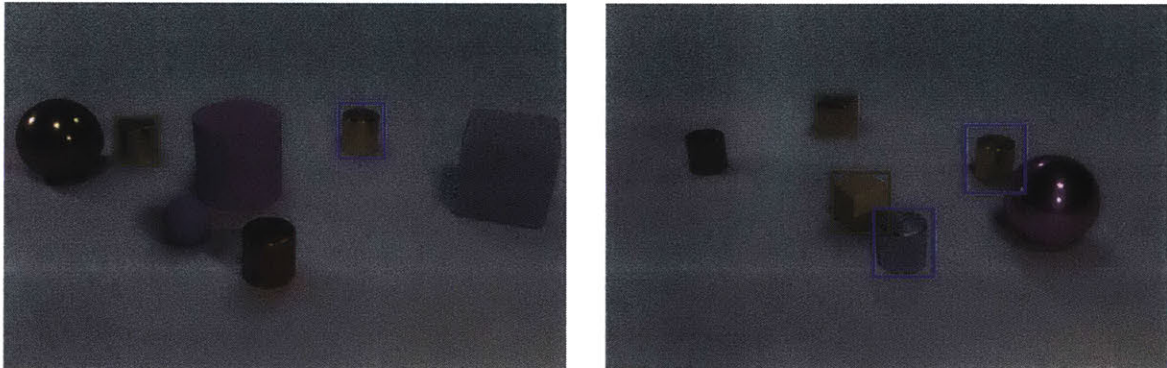
Figure B-6: **Concepts learned on the CLEVR dataset**

246

Concept: <u>Wolf</u>
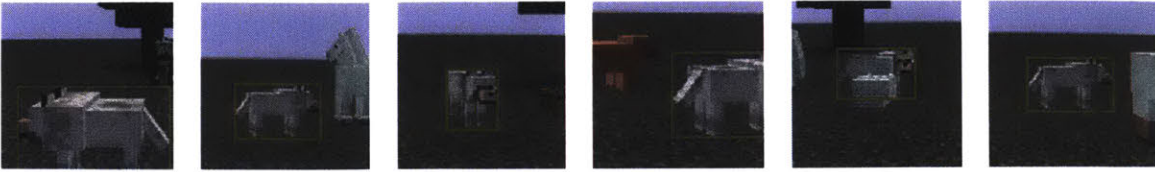
Concept: <u>Animal</u> That <u>Faces Right</u>
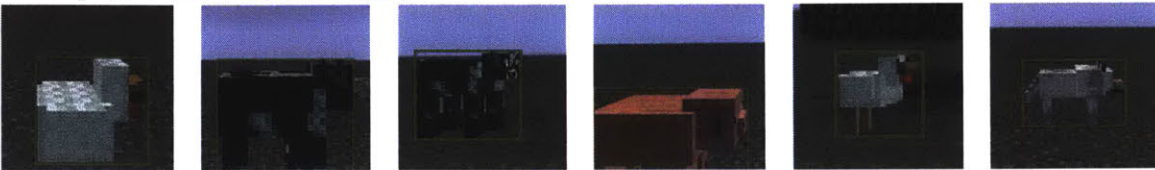
Concept: <u>Closest</u> <u>Living Thing</u>
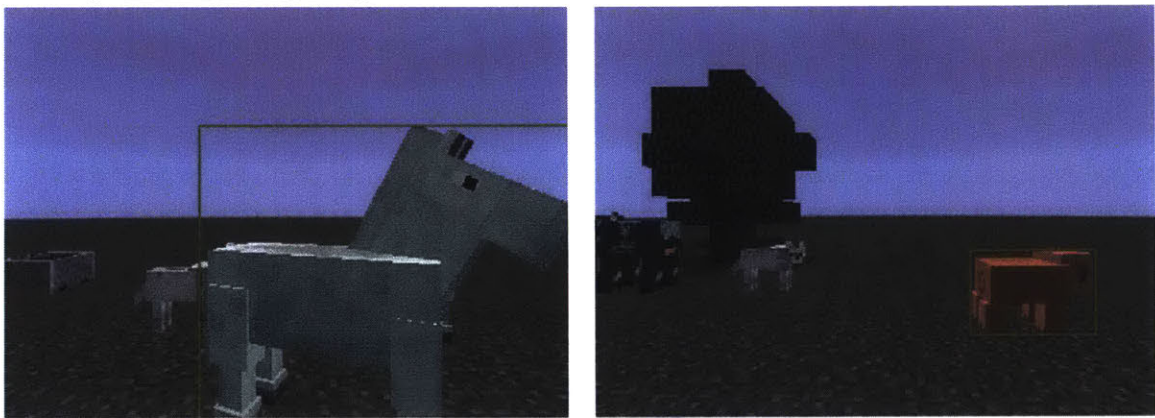
Concept: <u>Villager</u> <u>Closer</u> Than <u>Pig</u>

Figure B-7: **Concepts learned on the Minecraft dataset**

247

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix C

Details and Results for 3D Shape Programs

In Appendix C, we supply additional details for the Shape Programs introduced in Chapter 9.

C.1 Defined Programs

The details of semantics and shape primitives in our experimental setting for furniture are shown in Table C.1. Due to the semantic nature of objects, while a few semantics are category specific, e.g., "ChairBeam", other semantics are shared across different shape categories, e.g., "Leg" and "Top".

C.2 Architecture Details

Program generator. The program executor contains a 3D ConvNet and two LSTMs. *(1) 3D ConvNet.* This 3D ConvNet is the first part of the program generator model. It consists of 8 3D convolutional layers. The kernel size of each layer is 3 except for the first one whose kernel size is 5. The number of output channels are $(8, 16, 16, 32, 32, 64, 64, 64)$, respectively. The output of the last layer is averaged over the spatial resolution, which gives a 64-dimension embedding. *(2) Block LSTM and Step LSTM.* These two LSTMs share similar structure. Both are one-layer LSTMs. The dimensions of the input and hidden state are both 64.

Program executor. The program executor contains an LSTM and a 3D DeConvNet. *(1) LSTM.* This LSTM aggregates a block-level programs into a 64-dimensional vector. The dimension of the hidden state is also 64. The input of each time step is the concatenation of category distribution over programs and the corresponding parameters retrieved from the parameter matrix. *(2) 3D DeConvNet.* It consists of 7 layers. TransposedConv layer with kernel size 4 and Conv layer with kernel size 3 are alternating. The number of output channels are $(64, 64, 16, 16, 4, 4, 2)$, respectively. The output of the last layer is fed into a sigmoid function to generate the 3D voxel shape.

Semantics	Leg	Chair leg, table leg, etc. Usually long and used jointly for support
	Top	Seat top, table top, etc. Usually a broad and flat surface
	Layer	Shelf embedded in table, cabinet shelf etc. Usually a flat surface between other similar shapes
	Support	Chair support, table support etc. A monolithic object used to raise things off the ground
	Base	Base of an sofa, table etc. Usually a flat surface on the ground to help with stability
	Sideboard	Sideboard of a cabinet, table etc. A vertical, flat surface on the bottom half of an object
	Horizontal Bar	Horizontal bar of a chair etc. A thin bar used for structural integrity
	Vertical Board	Vertical board of a arm rest etc. A vertical, flat surface used by humans for arm support
	Locker	Table drawer etc. A boxy object used to put things in
	Back	Chair back, Sofa back etc. A surface used for resting backs on
	Back support	Office chair back support beam etc. A beam used to support a back rest
	ChairBeam	Arm rest support beam in chairs, benches etc. A long object used to support an arm rest
Shapes	Cylinder (Cyl)	$P = (x,y,z), G = (t,r)$, draw a cylinder at (x,y,z) with sizes (t,r)
	Cuboid (Cub)	$P = (x,y,z), G = (t,r_1,r_2,[ang])$, draw a cuboid at (x,y,z) with sizes (t,r_1,r_2) and optional ang of tilt along front/back
	Circle (Cir)	$P = (x,y,z), G = (t,r)$, draw a circle at (x,y,z) with sizes (t,r), t is usually small
	Square (Sqr)	$P = (x,y,z), G = (t,r)$, draw a square at (x,y,z) with sizes (t,r), t is usually small
	Rectangle (Rect)	$P = (x,y,z), G = (t,r_1,r_2)$, draw a rectangle at (x,y,z) with sizes (t,r_1,r_2), t is usually small
	Line (Line)	$P = (x_1,y_1,z_1), G = (x_2,y_2,z_2)$, draw a line from P to G

Table C.1: **The list of semantics and shapes**, as well as associated parameters used by our model

End-to-end differentiability. The end-to-end differentiability is obtained via our design of the neural program executor. The output of the program inference model is actually continuous. A real execution engine (not the neural executor) actually contains two steps: (1) discretize such output, and (2) execute the discretized program to generate the voxel. Our neural executor is learned to jointly approximate both steps, thus the whole pipeline can be differentiable in an end-to-end manner.

(a) 10 templates for synthetic dataset   (b) Raw and reconstructed shapes on ShapeNeT
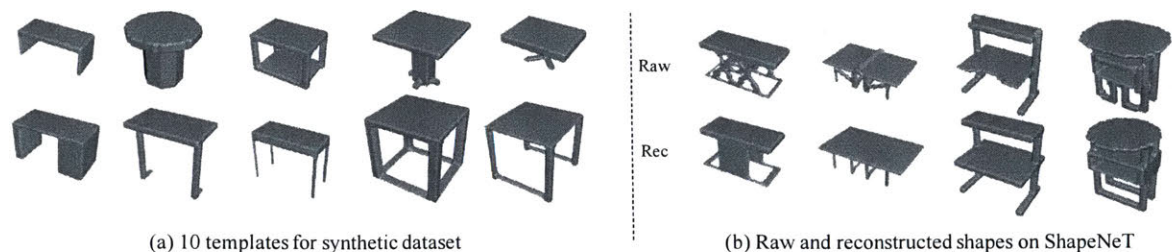
Figure C-1: **Templates and results on the ShapeNet dataset.** (a) shows random samples from all of our 10 table templates for synthetic dataset. (b) shows raw and reconstructed tables on ShapeNet.

C.3   Synthetic Templates vs. ShapeNet

ShapeNet was proposed to be the ImageNet of shapes; it is therefore highly diverse and intrinsically challenging. Our synthetic dataset were designed to provide minimal, simple guidance to the network. In Figure C-1, (a) shows sampled shapes from all of our 10 table templates, while (b) shows the ground truth and reconstructed tables in ShapeNet, which are significantly more complex. Such disparity of complexity explains why we saw a dramatic drop of IoU when we directly tested on ShapeNet with model only pre-trained on synthetic dataset. Our guided adaptation further adapt the pre-trained model.

C.4   Additional Results

In Figure C-2 through Figure C-8, we show the generated shape and programs using a network that is only pre-trained jointly on synthetic "table" and "chair" objects, and a network that is pre-trained then further enhanced by guided adaptation on ShapeNet data. Figure C-2 and Figure C-3 correspond to "chairs", Figure C-4 to "tables", Figure C-5 to "benches", Figure C-6 to "couches", Figure C-7 to "cabinets", and Figure C-8 to "beds". In Figure C-2, Figure C-3, and Figure C-4, even though "chair" and "table" have been seen in the synthetic dataset, we still note improvements in program inference and shape reconstruction on ShapeNet after guided adaptation. This is because our synthetic data is simpler than ShapeNet data. When directly using a model pre-trained on synthetic "chairs" and "tables" to other classes, it is not surprised some shapes are interpreted as tables or chairs. However, such mispredictions dramatically drop after our *guided adaptation*.

Panel (a)

Reconstruction before adaption
```
draw('Top','Rect',P=(5,0,0),G=(1,8,10))

for(i<2,'Trans',u=(0,0,16))
  draw('Sideboard', 'Rect',P=(-8,1,-8)+
    (i×u),G=(13,8,1))

draw('Layer','Rect',P=(-5,0,0),G=(1,7,8))
```

Reconstruction after adaption
```
draw('Top','Rect',P=(0,0,0),G=(2,8,9))

for(i<2,'Trans',u1=(0,0,14))
  for(j<2,'Trans',u2=(0,12,0))
    draw('Leg','Cub',P=(-11,-7,-8)
      +(j×u2)+(i×u1),G=(14,3,1))

draw('TiltBack','Cub',P=(1,5,-7),
  G=(10,2,15,11))

for(i<2,'Trans',u=(0,0,15))
  draw('HoriBar','Cub',P=(5,-8,-9)
    +(i×u),G=(2,14,2))
```

Input

Panel (b)

Reconstruction before adaption
```
draw('Top','Sqr',P=(1,0,0),G=(3,6))

for(i<2,'Trans',u1=(0,0,7))
  for(j<2,'Trans',u2=(0,9,0))
    draw('Leg','Cub',P=(-11,-6,-5)
      +(j×u2)+(i×u1),G=(14,2,3))

draw('Base','Sqr',P=(-8,0,0),G=(2,5))

draw('TiltBack','Cub',P=(2,3,-5),G=(10,3,10,6))
```

Reconstruction after adaption
```
draw('Top','Sqr',P=(-3,0,0),G=(3,5))

for(i<2,'Trans',u1=(0,0,7))
  for(j<2,'Trans',u2=(0,9,0))
    draw('Leg','Cub',P=(-11,-6,-5)
      +(j×u2)+(i×u1),G=(13,2,3))

draw('TiltBack','Cub',P=(1,2,-5),G=(10,3,10,6))

for(i<2,'Trans',u=(0,0,7))
  draw('ChairBeam','Cub',P=(1,-6,-5)
    +(i×u),G=(3,1,2))

for(i<2,'Trans',u=(0,0,6))
  draw('HoriBar','Cub',P=(3,-6,-5)+(i×u),G=(1,9,3))
```

Input

(a) (b)

(c) (d)

Panel (c)

Reconstruction before adaption
```
draw('Top','Cir',P=(-1,-1,0),G=(2,8))

draw('Support','Cyl',P=(-11,-1,0),G=(10,2))

for(i<4,'Rot',θ_rot=90,ax=(-12,-1,0))
  draw('Base','Line',P=(-12,-1,0),G=(-12,-7,-4),
    θ_rot×i,ax)

draw('TiltBack','Cub',P=(0,3,-5),G=(12,2,11,5))

for(i<2,'Trans',u=(0,0,13))
  draw('ChairBeam','Cub',P=(0,-3,-7  +(i×u),G=(3,1,1))

for(i<2,'Trans',u=(0,0,13))
  draw('HoriBar','Cub',P=(3,-3,-7)
    +(i×u),,G=(2,7,1))
```

Reconstruction after adaption
```
draw('Top','Cir',P=(-2,-1,0),G=(2,7))

draw('Support','Cyl',P=(-11,-1,0),G=(11,1))

for(i<5,'Rot',θ_rot=72,ax=(-11,-1,0))
  draw('Base','Line',P=(-11,-1,0),G=(-11,-7,-4),
    θ_rot×i,ax)

draw('TiltBack','Cub',P=(-1,3,-5),G=(12,2,10,5))

for(i<2,'Trans',u=(0,0,12))
  draw('ChairBeam','Cub',P=(0,-2,-7)
    +(i×u),G=(4,2,2))

for(i<2,'Trans',u=(0,0,12))
  draw('HoriBar','Cub',P=(4,-2,-7)
    +(i×u),G=(2,6,2))
```

Input

Panel (d)

Reconstruction before adaption
```
draw('Top','Rect',P=(5,-1,0),G=(1,11,12))

for(i<2,'Trans',u1=(0,0,22))
  for(j<2,'Trans',u2=(0,20,0))
    draw('Leg','Cub',P=(-7,-11,-12)
      +(j×u2)+(i×u1),G=(14,2,1))

for(i<2,'Trans',u=(0,0,23))
  draw('HoriBar','Cub',P=(-6,-10,-12)
    +(i×u),G=(1,21,1))

draw('HoriBar','Cub',P=(-5,-2,-13)
  +(i×u),G=(1,3,25))
```

Reconstruction after adaption
```
draw('Top','Rect',P=(-1,-1,0),G=(2,10,11))

for(i<2,'Trans',u1=(0,0,22))
  for(j<2,'Trans',u2=(0,19,0))
    draw('Leg','Cub',P=(-11,-11,-12)
      +(j×u2)+(i×u1),G=(12,2,2))

for(i<2,'Trans',u1=(0,0,22))
  draw('HoriBar','Cub',P=(-10,-11,-12)
    +(i×u),G=(2,20,2))

draw('BackSupp','Cub',P=(1,6,-4),G=(3,1,8))

draw('TiltBack','Cub',P=(3,6,-12),G=(7,2,23,30))

for(i<2,'Trans',u=(0,0,18))
  draw('ChairBeam','Cub',P=(1,-11,-10)
    +(i×u),G=(4,1,3))

for(i<2,'Trans',u=(0,0,18))
  draw('HoriBar','Cub',P=(5,-11,-11)
    +(i×u),G=(2,18,3))
```

Input

(e) (f)

Panel (e)

Reconstruction before adaption
```
draw('Top','Rect', P=(-5,-2,0),G=(4,7,9))

draw('Support','Cyl',P=(-11,-2,0),G=(10,2))

draw('Base','Cir',P=(-12,2,0),G=(1,6))

draw('TiltBack','Cub',P=(-2,4,-8),G=(13,3,15,17)

for(i<2,'Trans',u=(0,0,21))
  draw('Sideboard', 'Rect',P=(-2,0,11)
    +(i×u),G=(4,5,1))
```

Reconstruction after adaption
```
draw('Top','Rect',P=(-6,-1,0),G=(5,8,9))

draw('Support','Cyl',P=(-11,-2,0),G=(10,2))

for(i<4,'Rot',θ_rot=90,ax=(-12,-2,0))
  draw('Base','Line',P=(-12,-2,0),G=(-12,-9,-7),
    θ_rot×i,ax)

draw('TiltBack','Cub',P=(-2,3,-
7),G=(13,4,15,17))

for(i<2,'Trans',u=(0,0,20))
  draw('Sideboard', 'Rect',P=(-2,0,-12)
    +(i×u),G=(3,6,3))
```

Input

Panel (f)

Reconstruction before adaption
```
draw('Top','Cir',P=(3,0,0),G=(3,12))

for(i<2,'Trans',u1=(0,0,16))
  for(j<2,'Trans',u2=(0,16,0))
    draw('Leg','Cub',P=(-6,-9,-9)
      +(j×u2)+(i×u1),G=(13,2,2))

for(i<2,'Trans',u=(0,0,16))
  draw('HoriBar','Cub',P=(-5,-10,-9)
    +(i×u),G=(1,19,2))

draw('HoriBar','Cub',P=(-4,-2,-9),G=(1,3,18))
```

Reconstruction after adaption
```
draw('Top','Cir',P=(-1,-1,0),G=(3,11))

for(i<2,'Trans',u1=(0,0,15))
  for(j<2,'Trans',u2=(0,15,0))
    draw('Leg','Cub',P=(-11,-9,-8)
      +(j×u2)+(i×u1),G=(13,1,1))

for(i<2,'Trans',u=(0,0,15))
  draw('HoriBar','Cub',P=(-11,-10,-8)
    +(i×u),G=(1,17,1))

draw('TiltBack','Cub',P=(2,5,-9),G=(8,3,17,21))

for(i<2,'Trans',u=(0,0,18))
  draw('Sideboard', 'Rect',P=(3,-1,-10)
    +(i×u),G=(5,7,2))
```

Input

Figure C-2: **Generated shapes and programs for ShapeNet chairs**

(a)

Reconstruction before adaption

```
draw('Top','Rect',P=(2,0,0),G=(4,9,11))

for(i<2,'Trans',u1=(0,0,15))
 for(j<2,'Trans',u2=(0,16,0))
  draw('Leg','Cub',P=(-8,-10,-9)
  +(j×u2)+(i×u1),G=(15,2,3))

for(i<2,'Trans',u=(0,0,16))
 draw('HoriBar','Cub',P=(-8,-10,-9)
 +(i×u),G=(1,19,2))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-3,0,0),G=(5,9,10))

for(i<2,'Trans',u1=(0,0,13))
 for(j<2,'Trans',u2=(0,17,0))
  draw('Leg','Cub',P=(-12,-10,-8)
  +(j×u2)+(i×u1),G=(14,1,2))

for(i<2,'Trans',u=(0,0,13))
 draw('HoriBar','Cub',P=(-11,-10,-7)
 +(i×u),G=(2,18,2))

draw('TiltBack','Cub',P=(2,4,-7),G=(9,3,14,24))

for(i<2,'Trans',u1=(0,0,18))
 draw('Sideboard', 'Rect',P=(2,-3,-10)
 +(i×u),G=(3,5,2))
```

Input

(b)

Reconstruction before adaption

```
draw('Top','Rect',P=(-4,-2,0),G=(4,7,8))

draw('Support','Cyl',P=(-11,-2,0),G=(10,2))

for(i<2,'Trans',u=(0,0,10))
 draw('HoriBar','Cub',P=(-11,-7,-6)
 +(i×u), G=(1,12,2))

draw('TiltBack','Cub',P=(0,4,-8),G=(12,3,16,9))

for(i<2,'Trans',u=(0,0,16))
 draw('Sideboard', 'Rect',P=(-1,-2,-9)
 +(i×u),G=(2,5,2))
```

Reconstruction after adaption

```
draw('Top','Cir',P=(-5,-2,0),G=(4,8))

draw('Support','Cyl',P=(-11,-1,0),G=(9,1))

for(i<4,'Rot',θrot=90,ax=(-11,-2,0))
 draw('Base','Line',P=(-11,-2,0),G=(-11,-7,-6),
 θrot×i,ax)

draw('TiltBack','Cub',P=(-2,3,-8),G=(12,3,15,14))

for(i<2,'Trans',u=(0,0,15))
 draw('Sideboard', 'Rect',P=(-1,-1,-
8)+(i×u),G=(3,5,2))
```

Input

(c)

Reconstruction before adaption

```
draw('Top','Rect',P=(-2,-2,0),G=(1,8,10))

draw('VertBoard',P=(-10,-10,-10),G=(11,1,21))

draw('Base','Sqr',P=(-10,-1,0),G=(1,9))

draw('TiltBack','Cub',P=(1,6,-7),G=(10,1,14,17)
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-1,-2,0),G=(2,7,9))

for(i<2,'Trans',u=(0,0,20))
 draw('Sideboard', 'Rect',P=(-11,-2,-11)
 +(i×u),G=(11,9,3))

draw('Base','Sqr',P=(-10,-2,0),G=(1,8))

draw('TiltBack','Cub',P=(0,5,-7),G=(10,1,13,17))
```

Input

(d)

Reconstruction before adaption

```
draw('Top','Rect',P=(3,0,0),G=(3,12,13))

for(i<2,'Trans',u=(0,0,20))
 draw('Sideboard', 'Rect',P=(-7,0,-11)
 +(i×u),G=(10,10,2))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-6,0,0),G=(6,10,11))

for(i<2,'Trans',u1=(0,0,18))
 for(j<2,'Trans',u2=(0,13,0))
  draw('Leg','Cub',P=(-11,-9,-11)
  +(j×u2)+(i×u1),G=(11,2,2))

draw('TiltBack','Cub',P=(0,5,-10),G=(10,3,20,6))

for(i<2,'Trans',u=(0,0,21))
 draw('Sideboard', 'Rect',P=(1,-1,-11)
 +(i×u),G=(6,9,1))
```

Input

(e)

Reconstruction before adaption

```
draw('Top','Rect',P=(-2,-1,0),G=(4,6,8))

draw('Support','Cyl',P=(-11,-1,0),G=(12,3))

for(i<4,'Rot',θrot=90,ax=(-12,-2,0))
 draw('Base','Line',P=(-12,-2,0),G=(-12,-7,-5),
 θrot×i,ax)

draw('TiltBack','Cub',P=(0,4,-7),G=(11,3,13,10))

for(i<2,'Trans',u=(0,0,14))
 draw('ChairBeam','Cub',P=(1,-6,-8)+(i×u),G=(2,1,2))

for(i<2,'Trans',u=(0,0,14))
 draw('HoriBar','Cub',P=(3,-6,-8)+(i×u),G=(1,9,2))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-3,-1,0),G=(4,6,6))

draw('Support','Cyl',P=(-11,-1,0),G=(11,2))

for(i<5,'Rot',θrot=72,ax=(-11,-2,0))
 draw('Base','Line',P=(-11,-2,0),G=(-12,-8,-4),
 θrot×i,ax)

draw('TiltBack','Cub',P=(0,3,-6),G=(11,3,12,10))

for(i<2,'Trans',u=(0,0,13))
 draw('ChairBeam','Cub',P=(1,-5,-7)+(i×u),G=(3,1,2))

for(i<2,'Trans',u=(0,0,13))
 draw('HoriBar','Cub',P=(3,-4,-7)+(i×u),G=(2,9,2))
```

Input

(f)

Reconstruction before adaption

```
draw('Top','Rect',P=(2,-1,0),G=(5,9,11))

draw('Support','Cyl',P=(-8,-2,0),G=(13,5))
```

Reconstruction after adaption

```
draw('Top','Cir',P=(-5,-2,0),G=(5,10))

for(i<2,'Trans',u1=(0,0,12))
 for(j<2,'Trans',u2=(0,14,0))
  draw('Leg','Cub',P=(-10,-8,-8)
  +(j×u2)+(i×u1),G=(11,2,3))

draw('TiltBack','Cub',P=(0,3,-8),G=(10,3,15,22))

for(i<2,'Trans',u=(0,0,18))
 draw('Sideboard', 'Rect',P=(1,0,-10)
 +(i×u),G=(5,5,2))
```

Input

Figure C-3: **Generated shapes and programs for ShapeNet chairs**

Reconstruction before adaption

```
draw('Top','Sqr',P=(0,-1,0),G=(2,5))

draw('Support','Cyl',P=(-13,0,0),G=(16,2))

for(i<5,'Rot',θ_rot=72,ax=(-13,-1,0))
 draw('Base','Line',P=(-13,-1,0),G=(-15,-5,-5),
        θ_rot×i,ax)

draw('BackSupp','Cub',P=(1,2,-2),G=(4,1,3))

draw('TiltBack','Cub',P=(4,1,-4),G=(10,2,8,11))
```

Reconstruction after adaption

Input

```
draw('Top','Sqr',P=(9,0,0),G=(1,5))

draw('Support','Cyl',P=(-11,0,0),G=(22,2))

draw('Base','Sqr',P=(-12,0,0),G=(2,5))

draw('Locker','Cub',P=(-4,-3,-9),G=(10,5,2))
```

(a)

(b)

Reconstruction before adaption

```
draw('Top','Rect',P=(-6,-1,0),G=(4,5,8))

draw('VertBoard',P=(-14,-5,-10),G=(18,2,20))

draw('Layer','Rect',P=(-5,0,0),G=(2,3,11))

draw('Layer','Rect',P=(-2,0,0),G=(3,2,10))
```

Reconstruction after adaption

Input

```
draw('Top','Rect',P=(4,0,0),G=(4,3,12))

for(i<2,'Trans',u=(0,0,21))
 draw('Sideboard', 'Rect',P=(-9,0,-12)
   +(i×u),G=(12,3,2))

draw('Layer','Rect',P=(-8,0,0),G=(3,3,11))

draw('Locker','Cub',P=(-8,-3,-7),G=(12,5,4))
```

(c)

(d)

Reconstruction before adaption

```
draw('Top','Cir',P=(9,0,0),G=(2,13))

draw('Support','Cyl',P=(-11,0,0),G=(23,1))

for(i<5,'Rot',θ_rot=72,ax=(-11,0,0))
 draw('Base','Line',P=(-11,0,0),G=(-11,-8,-5),
        θ_rot×i,ax)
```

Reconstruction after adaption

Input

```
draw('Top','Sqr',P=(10,0,0),G=(1,11))

draw('Support','Cyl',P=(-11,0,0),G=(22,2))

for(i<4,'Rot',θ_rot=90,ax=(-11,0,0))
 draw('Base','Line',P=(-11,0,0),G=(-11,0,-10),
        θ_rot×i,ax)
```

Reconstruction before adaption

```
draw('Top','Rect',P=(4,0,0),G=(2,10,11))

for(i<2,'Trans',u1=(0,0,11))
 for(j<2,'Trans',u2=(0,13,0))
  draw('Leg','Cub',P=(-6,-8,-7)
   +(j×u2)+(i×u1),G=(13,2,3))

for(i<2,'Trans',u=(0,0,11))
 draw('HoriBar','Cub',P=(-7,-7,-6)
   +(i×u),G=(2,13,2))

draw('HoriBar','Cub',P=(-6,2,-7),G=(2,2,14))
```

Reconstruction after adaption

Input

```
draw('Top','Rect',P=(4,0,0),G=(2,10,12))

for(i<2,'Trans',u1=(0,0,12))
 for(j<2,'Trans',u2=(0,10,0))
  draw('Leg','Cub',P=(-6,-6,-7)
   +(j×u2)+(i×u1),G=(12,2,2))

for(i<2,'Trans',u=(0,0,12))
 draw('HoriBar','Cub',P=(-5,-8,-7)
   +(i×u),G=(2,16,2))

draw('HoriBar','Cub',P=(-6,-1,-7),G=(2,2,14))
```

(e)

(f)

Reconstruction before adaption

```
draw('Top','Rect',P=(0,0,0),G=(3,5,10))

for(i<2,'Trans',u=(0,0,21))
 draw('Leg','Cub',P=(-8,-5,-12)+(i×u),G=(10,2,2))

for(i<2,'Trans',u=(0,1,21))
 draw('HoriBar','Cub',P=(-8,-5,-11)
   +(i×u),G=(1,8,2))

for(i<2,'Trans',u1=(0,0,20))
 draw('HoriBar','Cub',P=(-3,-5,-11)
   +(i×u),G=(1,8,3))

draw('HoriBar','Cub',P=(-3,4,-12),G=(1,1,23))
```

Reconstruction after adaption

Input

```
draw('Top','Rect',P=(3,0,0),G=(2,4,12))

for(i<2,'Trans',u=(0,0,22))
 draw('Sideboard', 'Rect',P=(-5,0,-12)
   +(i×u),G=(7,4,2))

draw('VertBoard',P=(-3,3,-11),G=(6,1,23))

for(i<2,'Trans',u=(0,0,15))
 draw('Locker','Cub',P=(-3,-3,-10)+(i×u),G=(6,7,5))
```

Reconstruction before adaption

```
draw('Top','Rect',P=(5,0,0),G=(1,10,11))

for(i<2,'Trans',u1=(0,0,17))
 for(j<2,'Trans',u2=(0,19,0))
  draw('Leg','Cub',P=(-6,-11,-10)
   +(j×u2)+(i×u1),G=(13,4,3))

for(i<2,'Trans',u=(0,0,16))
 draw('HoriBar','Cub',P=(-5,-9,-10)
   +(i×u),G=(2,19,3))

draw('HoriBar','Cub',P=(-2,-6,-11),G=(2,5,20))
```

Reconstruction after adaption

Input

```
draw('Top','Rect',P=(4,0,0),G=(2,9,11))

for(i<2,'Trans',u=(0,0,16))
 draw('Sideboard', 'Rect',P=(-6,0,-9)
   +(i×u),G=(10,9,1))

for(i<2,'Trans',u=(0,18,2))
 draw('HoriBar','Cub',P=(-1,-11,-11)
   +(i×u),G=(1,3,20))
```

Figure C-4: **Generated shapes and programs for ShapeNet tables**

Reconstruction before adaption

```
draw('Top','Rect',P=(1,-1,0),G=(3,6,11))

for(i<2,'Trans',u=(0,0,14))
  draw('HoriBar','Cub',P=(-5,-3,-8)+(i×u),G=(2,7,2))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-1,0,0),G=(3,4,12))

for(i<2,'Trans',u=(0,0,16))
  draw('Leg','Cub',P=(-6,-2,-9)+(i×u),G=(9,1,2))

for(i<2,'Trans',u=(0,0,10))
  draw('ChairBeam','Cub',P=(2,-2,-6)+(i×u),G=(2,1,2))

for(i<2,'Trans',u1=(0,0,10))
  for(j<2,'Trans',u2=(0,3,0))
    draw('ChairBeam','Cub',P=(2,-3,-6)
    +(j×u2)+(i×u1),G=(3,1,2))

for(i<2,'Trans',u=(0,0,11))
  draw('HoriBar','Cub',P=(5,-2,-6)+(i×u),G=(1,5,2))
```

(a)

Reconstruction before adaption

```
draw('Top','Rect',P=(4,0,0),G=(1,8,11))

for(i<2,'Trans',u1=(0,0,20))
  for(j<2,'Trans',u2=(0,12,0))
    draw('Leg','Cub',P=(-5,-7,-11)
    +(j×u2)+(i×u1),G=(11,2,2))

draw('Layer','Rect',P=(0,0,0),G=(1,7,11))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(0,0,0),G=(1,5,11))

for(i<2,'Trans',u1=(1,0,21))
  for(j<2,'Trans',u2=(0,9,0))
    draw('Leg','Cub',P=(-6,-5,-11)
    +(j×u2)+(i×u1),G=(10,2,1))

for(i<2,'Trans',u=(0,0,20))
  draw('Sideboard','Rect',P=(0,1,-11)
  +(i×u),G=(3,6,2))

for(i<2,'Trans',u=(0,0,19))
  draw('Sideboard','Rect',P=(3,1,-11)
  +(i×u),G=(2,5,2))
```

(b)

Reconstruction before adaption

```
draw('Top','Rect',P=(-1,-1,0),G=(3,7,11))

for(i<2,'Trans',u1=(0,0,21))
  for(j<2,'Trans',u2=(0,9,0))
    draw('Leg','Cub',P=(-10,-6,-11)
    +(j×u2)+(i×u1),G=(13,3,1))

for(i<2,'Trans',u=(0,0,20))
  draw('HoriBar','Cub',P=(-5,-5,-11)
  +(i×u),G=(2,8,1))

draw('TiltBack','Cub',P=(3,4,-10),G=(8,3,20,7))
```

(c)

Reconstruction after adaption

```
draw('Top','Rect',P=(-2,-1,0),G=(2,4,12))

for(i<2,'Trans',u1=(1,0,21))
  for(j<2,'Trans',u2=(0,8,0))
    draw('Leg','Cub',P=(-8,-5,-12)
    +(j×u2)+(i×u1),G=(11,1,2))

for(i<2,'Trans',u=(0,0,20))
  draw('Sideboard','Rect',P=(-4,0,-12)
  +(i×u),G=(5,3,2))

draw('TiltBack','Cub',P=(2,2,-9),G=(7,2,19,16))

for(i<2,'Trans',u1=(1,0,21))
  draw('HoriBar','Cub',P=(3,-4,-11),G=(1,7,1))
```

Reconstruction before adaption

```
draw('Top','Rect',P=(-2,-1,0),G=(4,7,11))

for(i<2,'Trans',u=(0,0,18))
  draw('Sideboard','Rect',P=(-10,-2,-10)
  +(i×u),G=(11,6,1))

draw('TiltBack','Cub',P=(2,3,-10),G=(8,3,20,0))

for(i<2,'Trans',u=(0,0,15))
  draw('HoriBar','Cub',P=(2,-3,-9)+(i×u),G=(2,9,2))

for(i<2,'Trans',u=(0,0,16))
  draw('HoriBar','Cub',P=(5,-3,-9)+(i×u),G=(1,7,1))
```

(d)

Reconstruction after adaption

```
draw('Top','Rect',P=(-3,-1,0),G=(4,5,12))

for(i<2,'Trans',u1=(0,0,17))
  for(j<2,'Trans',u2=(0,7,0))
    draw('Leg','Cub',P=(-9,-5,-10)
    +(j×u2)+(i×u1),G=(10,1,2))

draw('TiltBack','Cub',P=(1,2,-9),G=(8,2,18,14))

draw('TiltBack','Cub',P=(2,4,-9),G=(8,2,19,7))
```

Reconstruction before adaption

```
draw('Top','Rect',P=(-1,0,0),G=(2,8,11))

for(i<2,'Trans',u=(0,0,21))
  draw('Sideboard','Rect',P=(-9,-1,-11)
  +(i×u),G=(10,6,1))

draw('TiltBack','Cub',P=(1,5,-10),G=(9,3,21,0))

for(i<2,'Trans',u=(0,0,21))
  draw('Sideboard','Rect',P=(0,2,-12)
  +(i×u),G=(4,6,2))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-1,0,0),G=(1,4,11))

for(i<2,'Trans',u1=(1,0,21))
  for(j<2,'Trans',u2=(0,6,0))
    draw('Leg','Cub',P=(-8,-5,-12)+(j×u2)
    +(i×u1),G=(10,2,1))

for(i<2,'Trans',u=(1,0,21))
  draw('Sideboard','Rect',P=(-5,0,-12)
  +(i×u),G=(5,3,1))

draw('TiltBack','Cub',P=(2,2,-10),G=(8,1,20,7))

for(i<2,'Trans',u=(1,0,22))
  draw('Sideboard','Rect',P=(1,-1,-12)
  +(i×u),G=(2,3,1))
```

(e)

Reconstruction before adaption

```
draw('Top','Rect',P=(0,0,0),G=(3,7,10))

for(i<2,'Trans',u=(0,0,19))
  draw('Sideboard','Rect',P=(-9,0,-10)
  +(i×u),G=(12,5,1))

draw('Locker','Cub',P=(-3,-3,2),G=(7,4,4))

for(i<2,'Trans',u=(0,0,15))
  draw('Sideboard','Rect',P=(1,0,-9)
  +(i×u),G=(4,3,2))
```

Reconstruction after adaption

```
draw('Top','Rect',P=(-1,0,0),G=(2,3,11))

for(i<2,'Trans',u1=(0,0,19))
  for(j<2,'Trans',u2=(0,6,0))
    draw('Leg','Cub',P=(-6,-3,-10)
    +(j×u2)+(i×u1),G=(10,2,1))

draw('TiltBack','Cub',P=(2,3,-8),G=(7,2,17,0))

for(i<2,'Trans',u=(1,0,20))
  draw('Sideboard','Rect',P=(2,1,-11)
  +(i×u),G=(3,2,1))
```

(f)

Figure C-5: **Generated shapes and programs for ShapeNet benches**
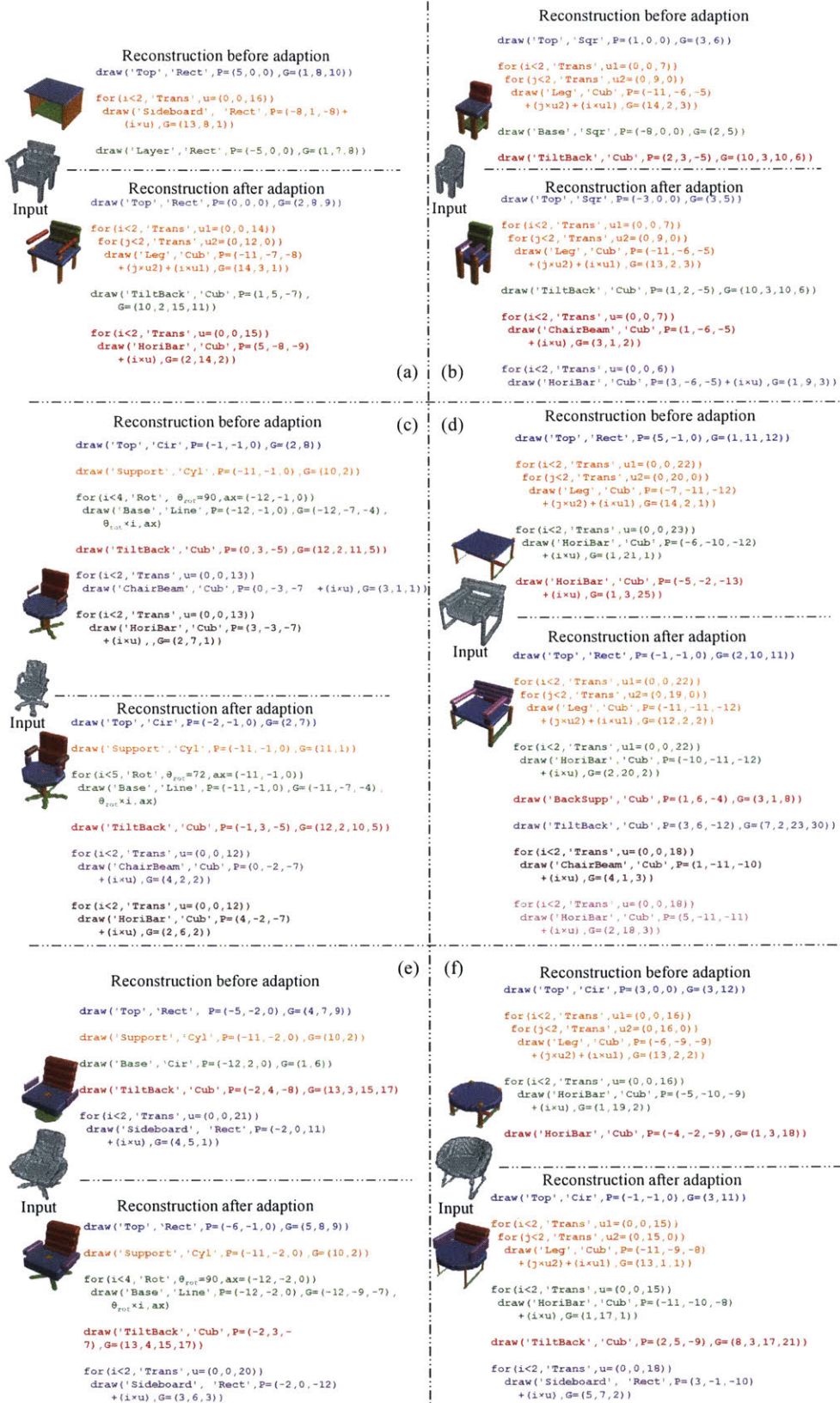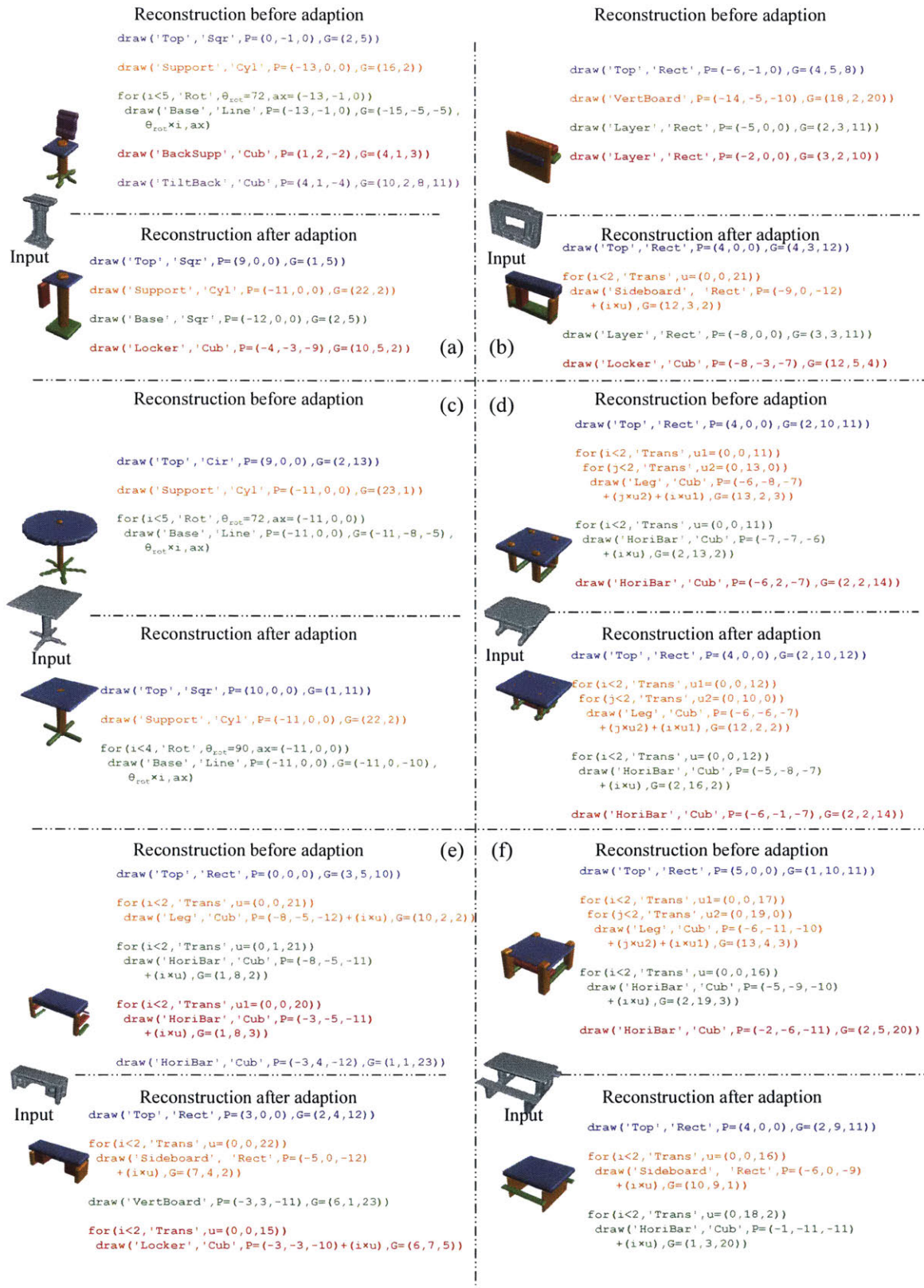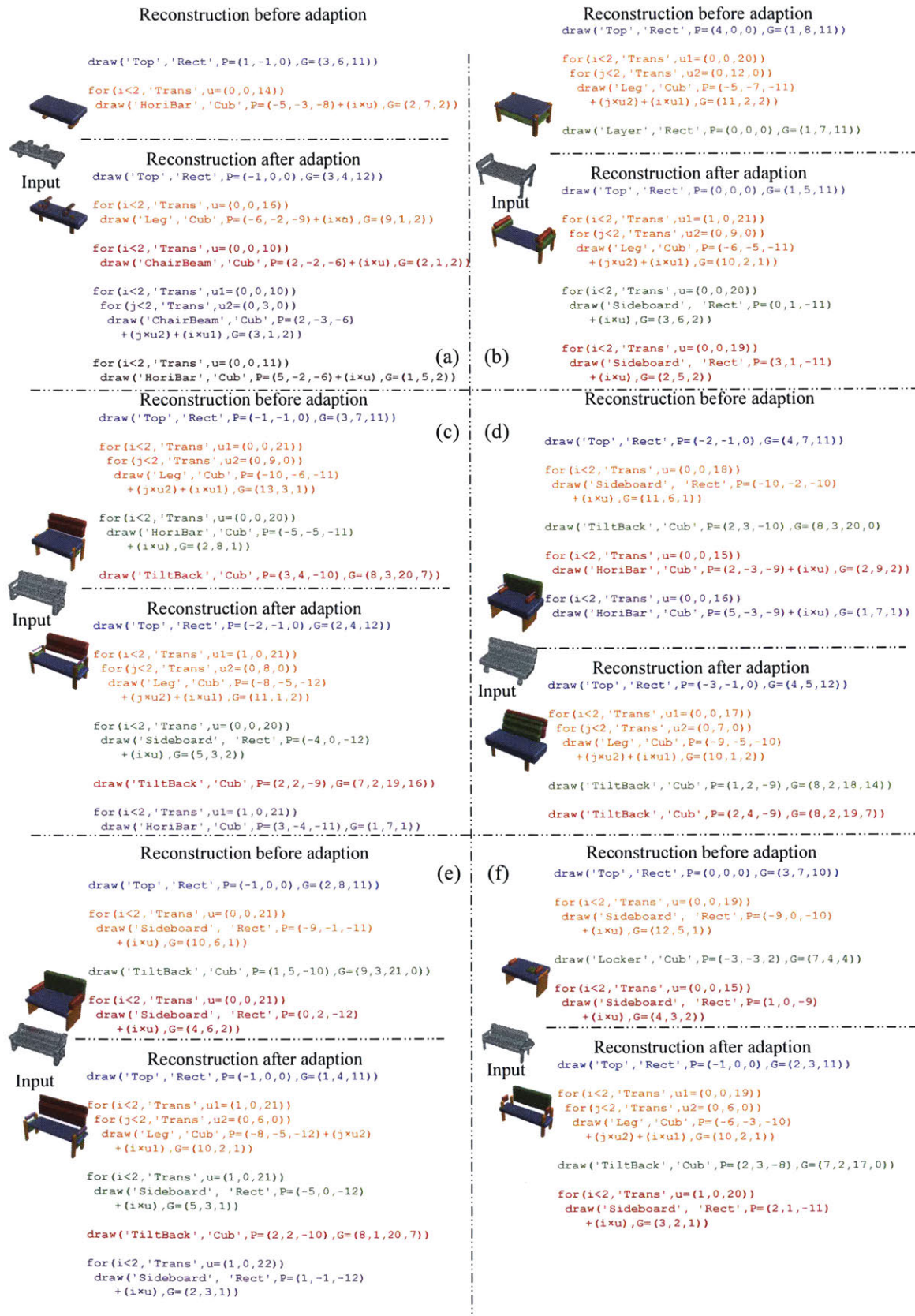
255

Figure C-6: **Generated shapes and programs for ShapeNet couches**

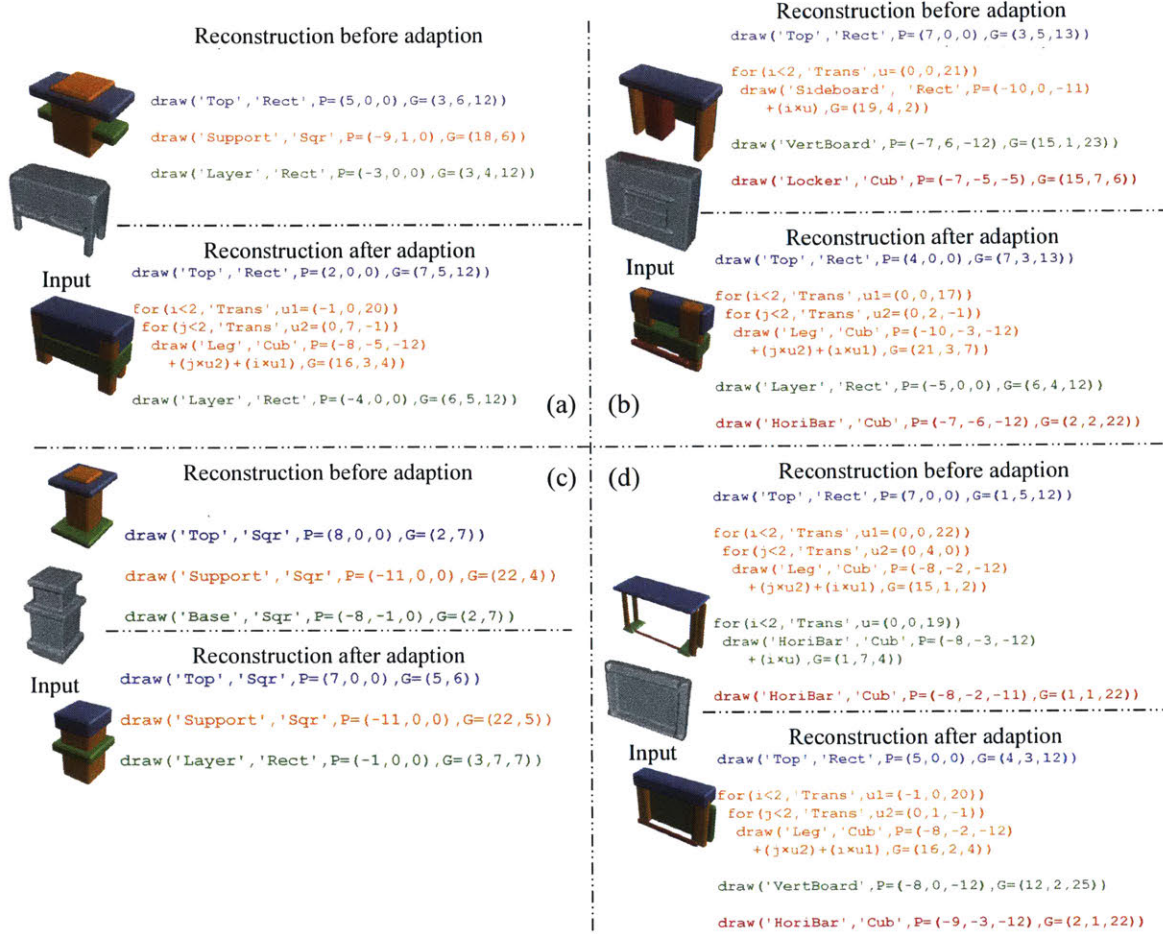Figure C-7: **Generated shapes and programs for ShapeNet cabinets**

(a)
Reconstruction before adaption
```
draw('Top','Rect',P=(5,0,0),G=(3,6,12))
draw('Support','Sqr',P=(-9,1,0),G=(18,6))
draw('Layer','Rect',P=(-3,0,0),G=(3,4,12))
```
Input
Reconstruction after adaption
```
draw('Top','Rect',P=(2,0,0),G=(7,5,12))
for(i<2,'Trans',u1=(-1,0,20))
 for(j<2,'Trans',u2=(0,7,-1))
  draw('Leg','Cub',P=(-8,-5,-12)
   +(j×u2)+(i×u1),G=(16,3,4))
draw('Layer','Rect',P=(-4,0,0),G=(6,5,12))
```

(b)
Reconstruction before adaption
```
draw('Top','Rect',P=(7,0,0),G=(3,5,13))
for(i<2,'Trans',u=(0,0,21))
 draw('Sideboard','Rect',P=(-10,0,-11)
  +(i×u),G=(19,4,2))
draw('VertBoard',P=(-7,6,-12),G=(15,1,23))
draw('Locker','Cub',P=(-7,-5,-5),G=(15,7,6))
```
Input
Reconstruction after adaption
```
draw('Top','Rect',P=(4,0,0),G=(7,3,13))
for(i<2,'Trans',u1=(0,0,17))
 for(j<2,'Trans',u2=(0,2,-1))
  draw('Leg','Cub',P=(-10,-3,-12)
   +(j×u2)+(i×u1),G=(21,3,7))
draw('Layer','Rect',P=(-5,0,0),G=(6,4,12))
draw('HoriBar','Cub',P=(-7,-6,-12),G=(2,2,22))
```

(c)
Reconstruction before adaption
```
draw('Top','Sqr',P=(8,0,0),G=(2,7))
draw('Support','Sqr',P=(-11,0,0),G=(22,4))
draw('Base','Sqr',P=(-8,-1,0),G=(2,7))
```
Input
Reconstruction after adaption
```
draw('Top','Sqr',P=(7,0,0),G=(5,6))
draw('Support','Sqr',P=(-11,0,0),G=(22,5))
draw('Layer','Rect',P=(-1,0,0),G=(3,7,7))
```

(d)
Reconstruction before adaption
```
draw('Top','Rect',P=(7,0,0),G=(1,5,12))
for(i<2,'Trans',u1=(0,0,22))
 for(j<2,'Trans',u2=(0,4,0))
  draw('Leg','Cub',P=(-8,-2,-12)
   +(j×u2)+(i×u1),G=(15,1,2))
for(i<2,'Trans',u=(0,0,19))
 draw('HoriBar','Cub',P=(-8,-3,-12)
  +(i×u),G=(1,7,4))
draw('HoriBar','Cub',P=(-8,-2,-11),G=(1,1,22))
```
Input
Reconstruction after adaption
```
draw('Top','Rect',P=(5,0,0),G=(4,3,12))
for(i<2,'Trans',u1=(-1,0,20))
 for(j<2,'Trans',u2=(0,1,-1))
  draw('Leg','Cub',P=(-8,-2,-12)
   +(j×u2)+(i×u1),G=(16,2,4))
draw('VertBoard',P=(-8,0,-12),G=(12,2,25))
draw('HoriBar','Cub',P=(-9,-3,-12),G=(2,1,22))
```


Figure C-8: **Generated shapes and programs for ShapeNet beds**

(a)
Reconstruction before adaption
```
draw('Top','Rect',P=(-4,-1,0),G=(4,10,12))
draw('VertBoard',P=(-10,-9,-11),G=(10,1,21))
draw('TiltBack','Cub',P=(0,9,-10),
 G=(10,2,21,0))
```
Input
Reconstruction after adaption
```
draw('Top','Rect',P=(-5,0,0),G=(5,12,10))
for(i<2,'Trans',u1=(0,0,18))
 for(j<2,'Trans',u2=(0,20,0))
  draw('Leg','Cub',P=(-8,-12,-10)
   +(j×u2)+(i×u1),G=(3,2,2))
draw('TiltBack','Cub',P=(-3,9,-11),
 G=(11,2,20,10))
```

(b)
Reconstruction before adaption
```
draw('Top','Sqr',P=(-4,-1,0),G=(4,10))
for(i<2,'Trans',u1=(0,0,18))
 draw('Sideboard','Rect',P=(-11,-1,-9)
  +(i×u),G=(10,8,1))
draw('TiltBack','Cub',P=(0,9,-9),G=(10,2,17,-2))
```
Input
Reconstruction after adaption
```
draw('Top','Sqr',P=(-4,0,0),G=(5,11))
draw('TiltBack','Cub',P=(-3,11,-9),G=(12,1,16,0))
draw('TiltBack','Cub',P=(0,9,-9),G=(10,1,16,6))
```

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix D

Details and Results
for Program-Guided Image Manipulators

In Appendix D, we supply additional details for the Program-Guided Image Manipulators (PG-IM) introduced in Chapter 10. In Section D.1, we first discuss the detailed implementation of PG-IM. We then, in Section D.2, provide ablation studies evaluating the effects of PG-IM's different modules or designs. Finally, we show more results produced by PG-IM in Section D.3, and also quantitatively compare them with results by the baselines.

D.1 Implementation Details

We discuss the implementation details necessary to reproduce our results.

D.1.1 Repeated Object Detection

We replicated the algorithm developed by Lettry et al. [2017]. The algorithm works on nearly-regular images and detects the repeated objects within the image. The model builds on the assumption that features learned from ImageNet [Deng et al., 2009] encode visual concepts at various levels: from edges to patches to high-level semantic objects. The algorithm takes as input the feature map from a pre-trained AlexNet [Krizhevsky et al., 2012] and outputs a set of 2D coordinates representing the centroids of repeated objects. In this section, we explain the algorithm as six-step procedure.

1) For each input feature map $\mathbf{f}$, it extracts all activated neurons at different locations. Denote the neuron at the 2D position (i, j) being activated by $act(\mathbf{f}_{i,j}) = 1$, which is given by

$$act(\mathbf{f}_{i,j}) = \mathbf{1}\left[\mathbf{f}_{i,j} == \max_{(u,v),|u-i|\leq K,|v-j|\leq K} \mathbf{f}_{u,v}\right], \qquad (\text{D.1})$$

where $\mathbf{1}[\cdot]$ is the indicator function, and K is a hyperparameter for "kernel size."

Feature maps at different layers of the AlexNet use different kernel sizes.

2) All activated 2D neurons are collected. For each pair of activated neurons from the same feature map, a displacement vector is computed. For simplicity, we use $d^{i,j} = (d_x^{i,j}, d_y^{i,j})$ to index the j-th displacement vector from the i-th feature map. A majority voting is performed over all displacement vectors. The output of this step is an image-level displacement vector $d^* = (d_x^*, d_y^*)$ between centroids of different objects

$$d_x^* = \arg\max_x \sum_{i,j} \exp(-\|d_x^{i,j} - x\|_2^2) \qquad \text{and} \qquad d_y^* = \arg\max_y \sum_{i,j} \exp(-\|d_y^{i,j} - y\|_2^2).$$

3) Based on the image-level displacement vector d^*, all displacement vectors are filtered. A vector is selected if

$$\|d^{i,j} - d^*\|_2 \leq 3 \times \alpha, \tag{D.2}$$

where α is a scalar hyperparameter of radius.

4) Each feature map is weighted by the number of selected displacement vectors from this feature map. Only highly weighted feature maps are selected.

5) For each 2D coordinate (x, y), we define a helper function

$$\mathcal{M}(x, y) \triangleq (x \bmod d_x^*, y \bmod d_y^*). \tag{D.3}$$

An offset o^* is then determined by optimizing the following objective:

$$o^* = \arg\min_o \sum_{i,j} \left(w_{i,j} \| M(d^{i,j} - o) - \frac{d^*}{2} \| \right), \tag{D.4}$$

where

$$w_{i,j} = \frac{1}{K_i + \phi} \cdot \exp\left(-\frac{\|d^{i,j} - d^*\|_2^2}{2\alpha^2} \right). \tag{D.5}$$

Here, K_i is the number of displacement vectors of feature map i. ϕ and α are hyperparameters. In our implementation, we use gradient descent to find the optimal o^*.

6) For each object region detected, the coordinates for all activated neurons within the region vote for the centroid. This step allows distortions of objects that may cause irregular lattices.

D.1.2   Adaptive Network Depth for PatchGAN

In order to encourage creative and sharp patch generation, we employ a discriminator loss provided by AdaPatchGAN, our adaptive variant of PatchGAN [Isola et al., 2017], in addition to the L1 loss. In the original PatchGAN, the classifier

architecture is fixed with a receptive field of a certain size. However, in images of our interest, the scale of texture varies violently across different images: the repeated pattern is sometimes a small entity but appearing many times, while other times the pattern is a larger object appearing only a few times. Given a single image, since its inferred program description provides a way of dividing it into patches, AdaPatchGAN computes a rough patch size and automatically adds convolutional layers until its receptive field is large enough to cover this patch size. In practice, we observe that our AdaPatchGAN enables the generator to produce more realistic patches than the original PatchGAN.

D.1.3 Extrapolation as Recurrent Inpainting

As mentioned in the main text, a single neural painting network (NPN) trained on the task of image inpainting is able to perform additional manipulation tasks including extrapolation without any finetuning. The straightforward way of doing this is treating the pixels to extrapolate as missing, and just inpainting them in one go. This approach of treating extrapolation without recurrent inpainting, however, results in distorted objects that fail to look realistic, as shown in an ablation study below. This is because NPNs are better at inpainting a localized patch (what they are trained to do as in inpainting) than inpainting a region spanning multiple objects.

Bearing this key observation in mind, our NPNs cast extrapolation into a task of *recurrent* inpainting, where it inpaints one object at a time, conditioned on the context provided by the previous inpainting. Note that this is only possible because the NPNs are guided by programs of images and therefore know how many objects should be extrapolated as well as where each of them should appear. Implementation-wise, this is achieved by repeatedly filling the current empty patch with the network output, and running network inference on the next empty patch. By this design, an empty patch gets inpainted so that it connects seamlessly to both the previously inpainted patches and the original image boundary.

D.1.4 Detailed Network Specifications and Training Parameters

We append to the end of this document a commented printout of the generator architecture, where channel numbers of the feature maps, kernel sizes, strides, padding, and more can be found. The training of PG-IM generally follows the procedure described in Pix2Pix [Isola et al., 2017], except that we do not decay our learning rate after 100 epochs like Pix2Pix does. Each of our epochs contains 1,000 training samples (i.e., random crops at different scales of the same image of interest), and the networks usually converge within 150 epochs.

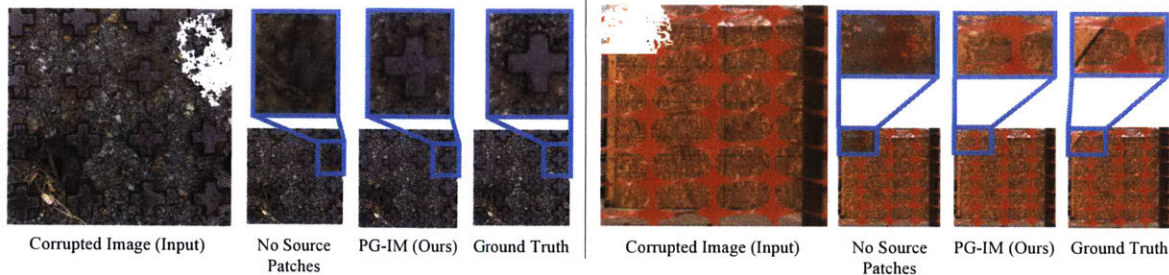| Corrupted Image (Input) | No Source Patches | PG-IM (Ours) | Ground Truth | | Corrupted Image (Input) | No Source Patches | PG-IM (Ours) | Ground Truth |

Figure D-1: **Inpainting with vs. without the source patches provided by program.** For each pane, the leftmost figure is the corrupted input to inpaint; to the right are results inpainted with and without source patches as well as the ground truth. With programs describing where the object centers are, our NPNs align the intact objects to the corrupted object and exploit such alignment to inpaint the missing pixels. Without such source patches, the networks have difficulty ensuring the "objectness" of the inpainted patch.

D.2   Ablation Studies

We report three ablations studies to provide intuitions as to why PG-IM works. Key to our approach is the use of program-like descriptions for images in guiding neural painting networks (NPNs) to perform pixel manipulations. The first study demonstrates why such descriptions facilitate the image manipulation tasks. Next, we justify a crucial design for image extrapolation: extrapolation should be cast into the task of *recurrent* inpainting for NPNs, rather than "one-go inpainting." Supplemental to Figure 10-9, the third study emphasizes the advantages of considering high-level attributes in the lower-level image manipulation tasks.

D.2.1   With vs. Without Source Patches

Our program-like description for an image informs NPNs of where the pattern of interest (repeatedly) appears in the image. NPNs can then align the available patterns to the corrupted or missing one in order to inpaint it (for simplicity, we discuss only image inpainting in this study, but the conclusion stands for the other tasks). These aligned image patches serve as "source patches," from which NPNs smartly copy to fill in the missing pixels.

Without program-like descriptions, NPNs would not have access to these source patches, which in turn leads to inferior performance such as results with the correct background but missing objects. As Figure D-1 shows, when programs (and hence source patches) are not available to the networks to smartly copy from, they fail to ensure the presence of the missing object in their inpainting (e.g., the cross in the left pane and the diamond in the right pane). In contrast, our NPNs exploit the source patches provided by programs and can inpaint the missing objects even if they are corrupted completely in the input. This result supports that PG-IM outperforms

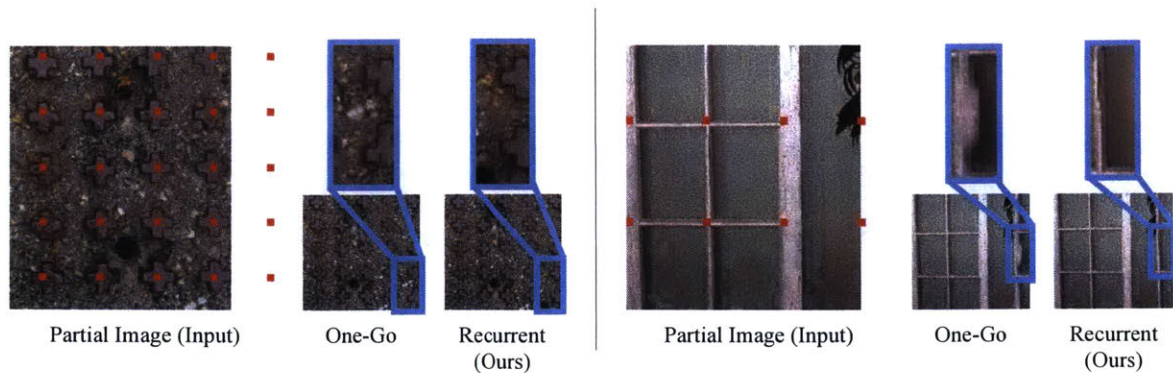| Partial Image (Input) | One-Go | Recurrent (Ours) | Partial Image (Input) | One-Go | Recurrent (Ours) |

Figure D-2: **Extrapolation as one-go vs. recurrent inpainting.** For each pane, the leftmost figure is the partial input to be extrapolated; to the right are results extrapolated with and without recurrent inpainting. Recall that the networks are trained only on inpainting one entity, so the networks tend to produce distorted objects if they are asked to inpaint the whole stripe spanning multiple entities. Instead, our NPNs treat extrapolation as recurrent inpainting, where they repeatedly solve the task they are good at and therefore produce better results.

repetition-aware but "programless" CNNs.

D.2.2 With vs. Without Recurrent Inpainting

In this study, we show empirical evidence that NPNs produce better results when extrapolation is cast into the task of recurrent inpainting rather than feedforward inpainting. As Figure D-2 shows, when the entire stripe gets extrapolated without recurrent inpainting, the extrapolated objects tend to be distorted, since our NPNs are trained only on more localized inpainting. By treating extrapolation as *recurrent* inpainting, our NPNs are essentially doing, repeatedly, what they are trained on and hence generate more realistic-looking results.

D.2.3 With vs. Without Considering Attributes

In Figure 10-9, we show our program-like descriptions of images can naturally incorporate attributes, enabling the NPNs to perform attribute-aware image manipulation, e.g., by referencing only patches with the same attributes as the patch to inpaint. We supplement two more examples demonstrating why attribute-aware image manipulations are important, and how PG-IM achieves superior performance on such tasks.

As shown in Figure D-3 left, the tomato to be inpainted is next to four orange tomatoes and one green tomato. Since convolutions are more of a local operation, one can expect a convolutional generator to produce a tomato with mixed colors of both orange and green, as is the case for PG-IM without attributes. However, we, as humans, reason at a higher level over the entire image, coming to the conclusions that the tomato to inpaint on the left should be orange, and that the candy to inpaint
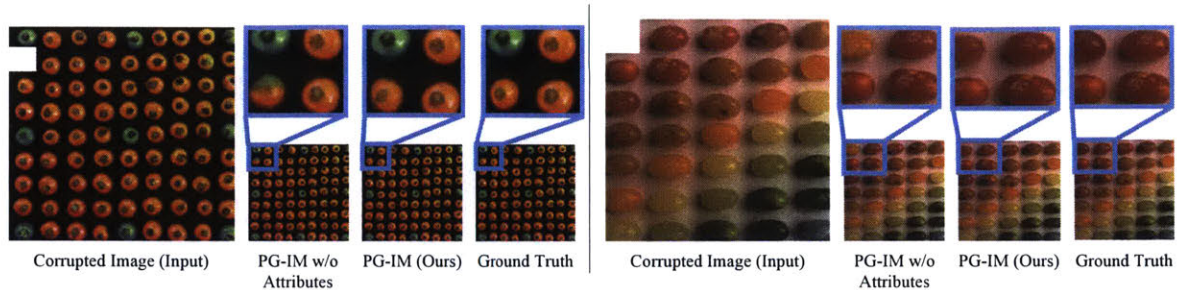
Figure D-3: **Inpainting with vs. without considering object attributes.** For each pane, the leftmost figure is the corrupted input to be inpainted; to the right are results inpainted with and without considering attributes as well as the ground truth. The attribute-agnostic networks tend to produce patches with mixed colors that break the global regularity of attributes. PG-IM is able to respect such regularity by referencing only patches with the same attribute as the patch to inpaint.

on the right should be red. With similar capabilities, PG-IM correctly preserves the global regularity of attributes in its inpainting.

D.3 More Experiments and Results

We supplement more results by PG-IM on the tasks of image inpainting, extrapolation, and regularity editing. For PatchMatch and Image Quilting, we search for one set of optimal hyperparameters and use that for the entire test set. Similarly, we train Non-Stationary [Zhou et al., 2018] and PG-IM with single sets of optimal hyperparameters on each test image. For GatedConv [Yu et al., 2019], we use the trained model released by the authors.

D.3.1 Inpainting

Figure D-4 shows how PG-IM compares in image inpainting with the baselines on another six test images. PG-IM is able to ensure the presence of the repeated object in its inpainting (e.g., the cross structure in Image 1 and the yellow tag in Image 5), while the baselines tend to have the object missing (e.g., PatchMatch completely misses the yellow tag in Image 5) or incomplete (e.g., the pig in Image 3) due to lack of the concept of objects.

D.3.2 Extrapolation

We supplement four more examples of image extrapolation by PG-IM and how they compare with results by the baselines. As Figure D-5 shows, extrapolation by PG-IM is sharp, respects the global regularity (e.g., the "pig array" in Image 3), and seamlessly connects to the original images. In contrast, PatchMatch tends to blur over the image boundaries for smooth transition to the extrapolated contents (e.g., the blurriness over the boundary in Image 2). Image Quilting and GatedConv tend
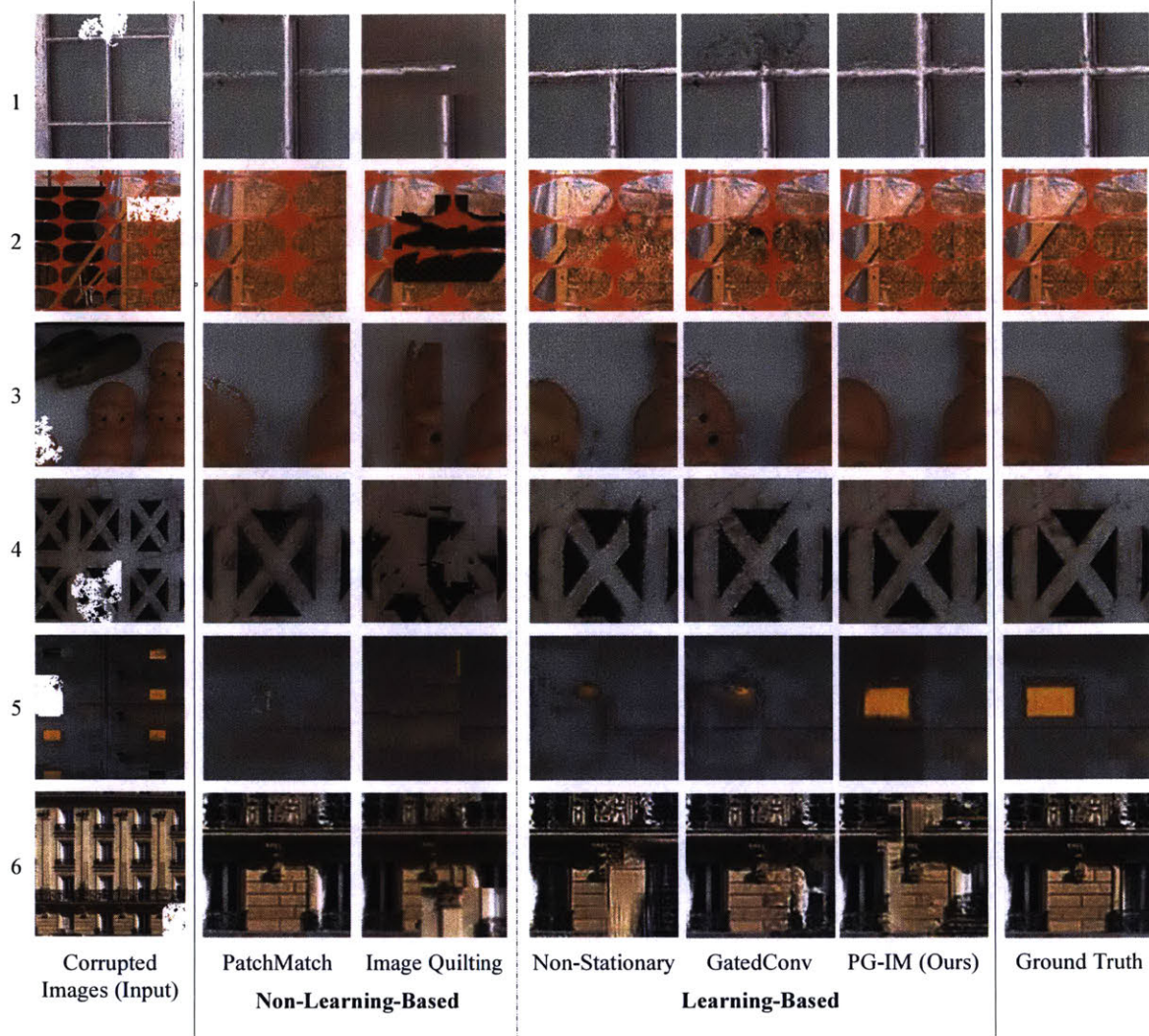
Figure D-4: **Additional inpainting results by PG-IM and the baselines.** The white pixels in the leftmost column are missing pixels to inpaint; the rightmost column shows the ground-truth patches. For easier comparison, we show only the close-up views of the inpainted region (and its proximity for some context). Aware of the images' global regularity, PG-IM inpaints the missing objects that maintain this regularity. Note this is not necessarily the case for other methods: baselines miss out the pig entity of Image 3 and the yellow tag of Image 5. Although PatchMatch gets the cross structure in Image 1, its result is less realistic than ours. Another advantage of PG-IM over the baselines is the ability to generate pixels that seamlessly connect to the original image contents (compare the results for Images 1, 2, and 4).

to break the global regularity (e.g. Images 2 and 4). Finally, Non-Stationary should be considered as a super-resolution method instead of an extrapolation one, because it essentially replicates the patterns on a larger canvas and interpolates in between; notice how it does *not* extrapolate beyond the texture boundaries in Images 2 and 4.

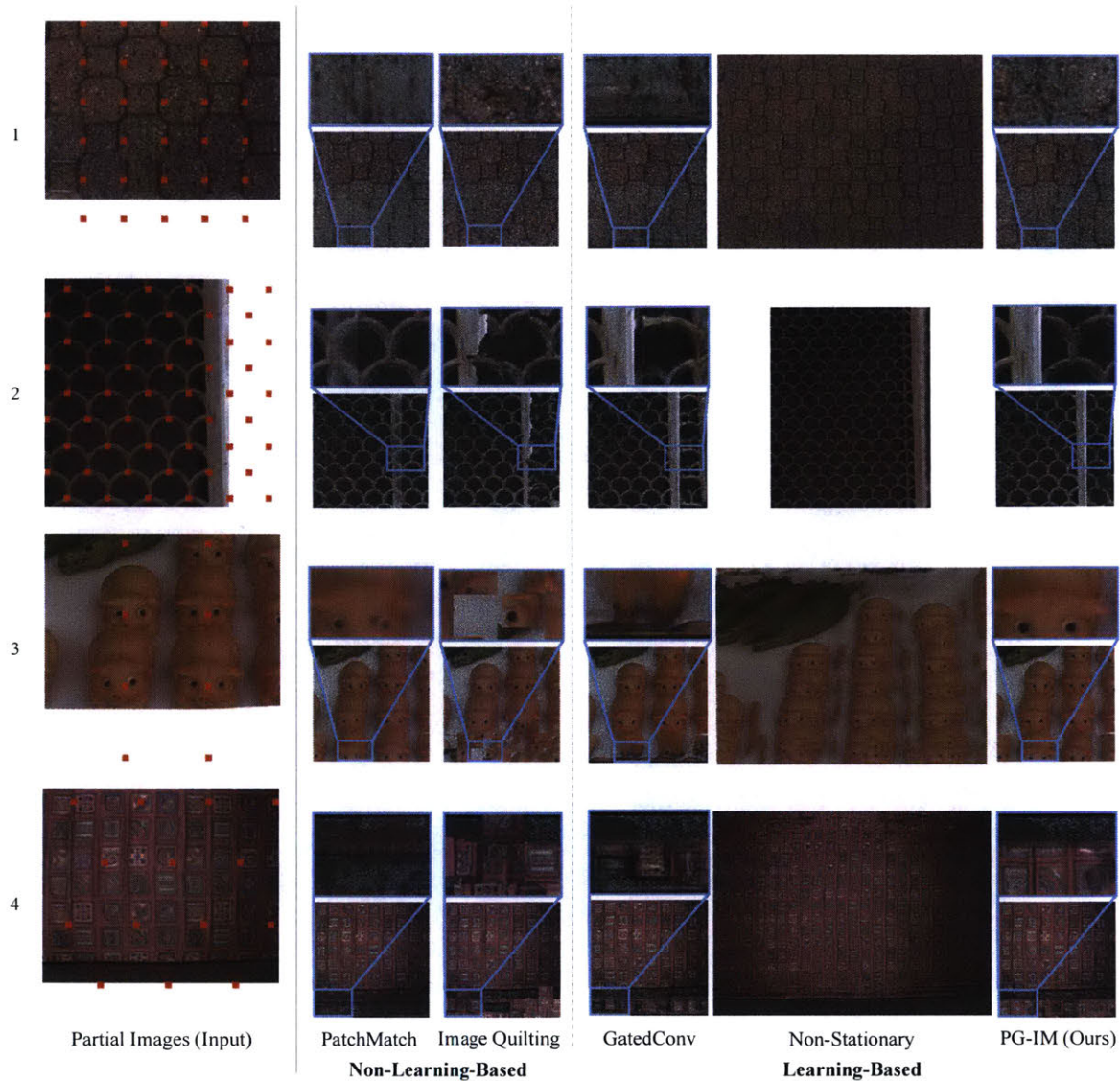| | PatchMatch | Image Quilting | GatedConv | Non-Stationary | PG-IM (Ours) |
| Partial Images (Input) | **Non-Learning-Based** | | **Learning-Based** | | |

Figure D-5: **Additional extrapolation results by PG-IM and the baselines.** The white pixels in the leftmost column indicate the pixels to be extrapolated, and the red dots are object centers given by the program descriptions. With such program descriptions, PG-IM knows where to extrapolate to automatically, whereas other methods require the user to specify where to extrapolate to. PG-IM generates realistic images while preserving the global regularity. In contrast, GatedConv fails to capture the regularity; Non-Stationary does not preserve the original content of images; non-learning-based baselines sometimes generate blurry images because of the repetition of similar objects in the partial image.

D.3.3 Regularity Editing

We supplement four more examples of image regularity editing by PG-IM in Figure D-6. Although PG-IM has been trained only on inpainting, with the program descriptions of images, it is naturally capable of exaggerating the irregularity in the images' global structures.

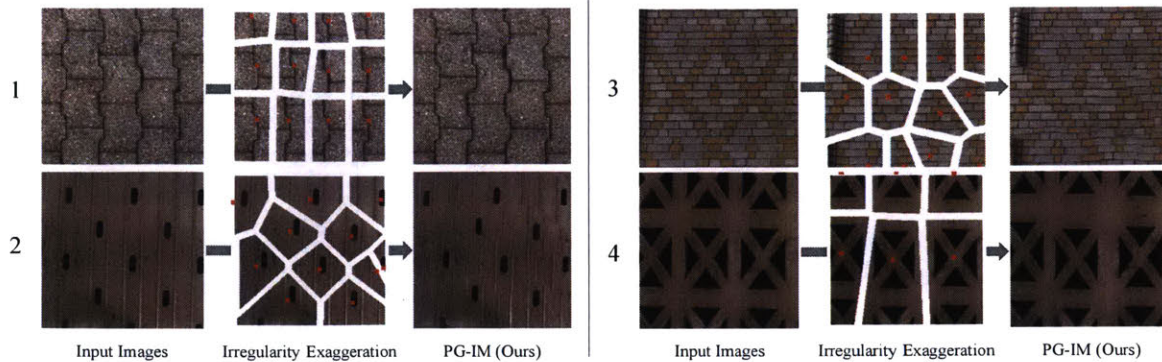| Input Images | Irregularity Exaggeration | PG-IM (Ours) | | Input Images | Irregularity Exaggeration | PG-IM (Ours) |

Figure D-6: **PG-IM enables automated and semantic-aware irregularity exaggeration.** By comparing the centroids of the detected objects and the ones reconstructed by the program, we can measure and exaggerate the structural irregularity of input images. In these examples, we first multiply by 2 the displacement vectors between the object centroids provided by the programs and the detected object centroids, and then randomly flip the sign for each displacement vector. According to these new displacement vectors, we shift the patches and then let the NPN fill in the missing pixels.

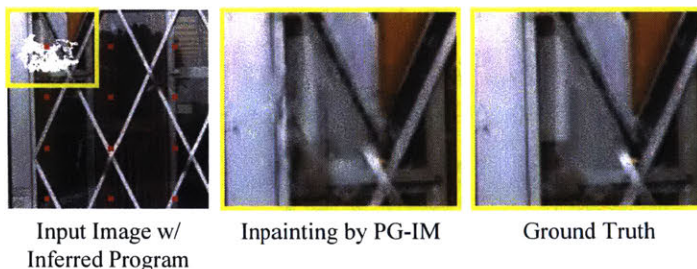| Input Image w/ Inferred Program | Inpainting by PG-IM | Ground Truth |

Figure D-7: **A failure case for inpainting.** In this challenging case, the transmission and reflection effects make the repetitive patterns less homogeneous, leading to non-photorealistic inpainting.

D.3.4 Failure Cases

Figure D-7 demonstrates a failure case for inpainting. In this challenging case, the transmission and reflection effects make the repetitive patterns less homogeneous, leading to non-photorealistic inpainting.

D.3.5 Baseline Finetuning

PG-IM learns from a single input image by exploiting the repetition structure, whereas other learning-based methods (GatedConv and PartialConv) learn from a large dataset (Places365). We supplement extra experiments, where we finetune the GatedConv model (pre-trained on Places365) on 50 Facade images and evaluate it on the held-out images (Figure D-8). Learning from similar facade images does help the model produce more photorealistic results (Inception score: $1.187 \rightarrow 1.191$). However, the model still fails on structured objects such as windows, which can be well handled by our patch-based, program-guided NPN (PG-IM's Inception score is 1.210).
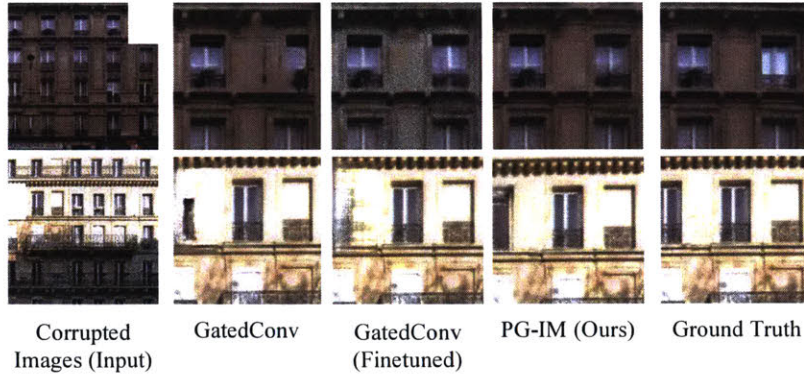
| Corrupted Images (Input) | GatedConv | GatedConv (Finetuned) | PG-IM (Ours) | Ground Truth |

Figure D-8: **Comparing with the GatedConv model finetuned on the Facade dataset.** Our PG-IM model is only trained on a single input image and still performs better.

D.4 Detailed Generator Architecture

```
('1', ReppadConv2d(3, 64, 3, stride=1, padding=1)),
('1a', nn.ELU()),
# 2: maxpool across tracks
# 3: concat 2
('4:+pool', ReppadConv2d(128, 96, 1, stride=1)),
('4a', nn.ELU()),
('5', ReppadConv2d(96, 96, 4, stride=2, padding=1)),
('5a', nn.ELU()),
# 6: maxpool across tracks
# 7: concat 6
('8:+pool', ReppadConv2d(192, 128, 1, stride=1)),
('8a', nn.ELU()),
('9', ReppadConv2d(128, 128, 4, stride=2, padding=1)),
('9a', nn.ELU()),
# 10: maxpool across tracks
# 11: concat 10
('12:+pool', ReppadConv2d(256, 256, 1, stride=1)),
('12a', nn.ELU()),
('13', ReppadConv2d(256, 256, 4, stride=2, padding=1)),
('13a', nn.ELU()),
# 14: maxpool across tracks
# 15: concat 14
('16:+pool', ReppadConv2d(512, 384, 1, stride=1)),
('16a', nn.ELU()),
('17', ReppadConv2d(384, 384, 4, stride=2, padding=1)),
('17a', nn.ELU()),
('18', ReppadConv2d(384, 384, 3, stride=1, padding=1)),
('18a', nn.ELU()),
# ----------------
('19', nn.ConvTranspose2d(384, 384, 4, stride=2, padding=1)),
('19a', nn.ELU()),
```

268

```python
# 20: maxpool across tracks
# 21: concat 16a, 20
('22:+pool+16a', ReppadConv2d(1152, 384, 1, stride=1)),
('22a', nn.ELU()),
('23', ReppadConv2d(384, 384, 3, stride=1, padding=1)),
('23a', nn.ELU()),
('24', nn.ConvTranspose2d(384, 256, 4, stride=2, padding=1)),
('24a', nn.ELU()),
# 25: maxpool across tracks
# 26: concat 12a, 25
('27:+pool+12a', ReppadConv2d(768, 256, 1, stride=1)),
('27a', nn.ELU()),
('28', ReppadConv2d(256, 256, 3, stride=1, padding=1)),
('28a', nn.ELU()),
('29', nn.ConvTranspose2d(256, 192, 4, stride=2, padding=1)),
('29a', nn.ELU()),
# 30: maxpool across tracks
# 31: concat 8a, 30
('32:+pool+8a', ReppadConv2d(512, 192, 1, stride=1)),
('32a', nn.ELU()),
('33', ReppadConv2d(192, 192, 3, stride=1, padding=1)),
('33a', nn.ELU()),
('34', nn.ConvTranspose2d(192, 96, 4, stride=2, padding=1)),
('34a', nn.ELU()),
# 35: maxpool across tracks
# 36: concat 1a, 35
('37:+pool+1a', ReppadConv2d(256, 96, 1, stride=1)),
('37a', nn.ELU()),
('38', ReppadConv2d(96, 96, 3, stride=1, padding=1)),
('38a', nn.ELU()),
# 39: maxpool across tracks
('40:pool', ReppadConv2d(96, 64, 3, stride=1, padding=1)),
('40a', nn.ELU()),
('41', ReppadConv2d(64, nch_out, 3, stride=1, padding=1)),
```

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

Omri Abend, Tom Kwiatkowski, Nathaniel J Smith, Sharon Goldwater, and Mark Steedman. Bootstrapping Language Acquisition. *Cognition*, 2017. 162

Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin, and Richard Szeliski. Panoramic Video Textures. *ACM Transactions on Graphics (TOG)*, 24(3):821–827, 2005. 131

Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to Poke by Poking: Experiential Learning of Intuitive Physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 101, 102

Miika Aittala and Frédo Durand. Burst Image Deblurring Using Permutation Invariant Convolutional Neural Networks. In *European Conference on Computer Vision (ECCV)*, 2018. 209

Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P. Kaelbling, Joshua B. Tenenbaum, and Alberto Rodriguez. Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018. 5

Anurag Ajay, Maria Bauza, Jiajun Wu, Nima Fazeli, Joshua B. Tenenbaum, Alberto Rodriguez, and Leslie P. Kaelbling. Combining Physical Simulators and Object-Based Networks for Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019. 5, 6

Zeynep Akata, Mateusz Malinowski, Mario Fritz, and Bernt Schiele. Multi-Cue Zero-Shot Learning with Strong Supervision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 39

Ijaz Akhter and Michael J. Black. Pose-Conditioned Joint Angle Limits for 3D Human Pose Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 15

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and Top-Down Attention for Image Captioning and Visual Question Answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 163, 176, 181, 238

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to Compose Neural Networks for Question Answering. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2016. 163, 164, 180, 231, 241

Stanislaw Antol, C. Lawrence Zitnick, and Devi Parikh. Zero-Shot Learning via Visual Abstraction. In *European Conference on Computer Vision (ECCV)*, 2014. 39

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 180, 181, 240

Yoav Artzi and Luke Zettlemoyer. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. *Transactions of the Association for Computational Linguistics (TACL)*, 2013. 181

Michael Ashikhmin. Synthesizing Natural Textures. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, 2001. 201

Mathieu Aubry, Daniel Maturana, Alexei Efros, Bryan Russell, and Josef Sivic. Seeing 3D Chairs: Exemplar Part-Based 2D-3D Alignment Using a Large Dataset of Cad Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 12, 15

Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. In *International Conference on Learning Representations (ICLR)*, 2015. 66, 68, 102

Yuval Bahat and Michal Irani. Blind Dehazing Using Internal Patch Recurrence. In *IEEE International Conference on Computational Photography (ICCP)*, 2016. 202

Yuval Bahat, Netalee Efrat, and Michal Irani. Non-Uniform Blind Deblurring by Reblurring. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 202

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations (ICLR)*, 2015. 166, 204

Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Selectively De-animating Video. *ACM Transactions on Graphics (TOG)*, 31(4):66–1, 2012. 102

Renée Baillargeon. Infants' Physical World. *Current Directions in Psychological Science*, 13 (3):89–94, 2004. 87

Guha Balakrishnan, Amy Zhao, Adrian V. Dalca, Fredo Durand, and John Guttag. Synthesizing Images of Humans in Unseen Poses. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 156

Coloma Ballester, Marcelo Bertalmio, Vicent Caselles, Guillermo Sapiro, and Joan Verdera. Filling-in by Joint Interpolation of Vector Fields and Gray Levels. *IEEE Transactions on Image Processing (TIP)*, 10(8):1200–1211, 2001. 201

Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. DeepCoder: Learning to Write Programs. In *International Conference on Learning Representations (ICLR)*, 2017. 186

Aayush Bansal and Bryan Russell. Marr Revisited: 2D-3D Alignment via Surface Normal Prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 15, 38

Fabien Baradel, Natalia Neverova, Christian Wolf, Julien Mille, and Greg Mori. Object Level Visual Reasoning in Videos. In *European Conference on Computer Vision (ECCV)*, 2018. 181

Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM Transactions on Graphics (TOG)*, 28(3):24, 2009. 201, 211, 212

Jonathan T. Barron and Jitendra Malik. Shape, Illumination, and Reflectance from Shading. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 37(8):1670–1687, 2015. 38

Harry G. Barrow and Jay M. Tenenbaum. Recovering Intrinsic Scene Characteristics from Images. In *Computer Vision Systems*. Elsevier, 1978. 33, 38

Harry G. Barrow, Jay M. Tenenbaum, Robert C. Bolles, and Helen C. Wolf. Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1977. 57, 191

Evgeniy Bart and Shimon Ullman. Cross-Generalization: Learning Novel Classes from a Single Example by Feature Replacement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 39

Peter W. Battaglia, Jessica B. Hamrick, and Joshua B. Tenenbaum. Simulation As an Engine of Physical Scene Understanding. *Proceedings of the National Acacdemy of Sciences (PNAS)*, 110(45):18327–18332, 2013. 4, 86, 101

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction Networks for Learning about Objects, Relations and Physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 101, 102

Peter N. Belhumeur, David W. Jacobs, David J. Kriegman, and Narendra Kumar. Localizing Parts of Faces Using a Consensus of Exemplars. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 35(12):2930–2940, 2013. 23

Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic Images in the Wild. *ACM Transactions on Graphics (TOG)*, 33(4):159, 2014. 38

Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. Material Recognition in the Wild with the Materials in Context Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 116

Tony Beltramelli. Pix2Code: Generating code from a graphical user interface screenshot. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, 2018. 201

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *International Conference on Machine Learning (ICML)*, 2009. 69

Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial GAN. In *International Conference on Machine Learning (ICML)*, 2017. 202

Thomas G. Bever and David Poeppel. Analysis by Synthesis: A (Re-)Emerging Program of Research for Language and Vision. *Biolinguistics*, 4(2-3):174–200, 2010. 15, 66

Irving Biederman. Recognition-by-Components: A Theory of Human Image Understanding. *Psychological Review*, 94(2):115, 1987. 185

Thomas O. Binford. Visual Perception by Computer. Invited talk at IEEE Conf. on Systems and Control, 1971. 185

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a Differentiable Forth Interpreter. In *International Conference on Machine Learning (ICML)*, 2017. 186

Lubomir Bourdev. *Poselets and Their Applications in High-Level Computer Vision*. PhD thesis, UC Berkeley, 2011. 23

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic Filter Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 139

Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer Science & Business Media, 2008. 57

Jonathon Cai, Richard Shin, and Dawn Song. Making Neural Programming Architectures Generalize via Recursion. In *International Conference on Learning Representations (ICLR)*, 2017. 186

Qingxing Cao, Xiaodan Liang, Bailing Li, Guanbin Li, and Liang Lin. Visual Question Reasoning on General Dependency Tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 168

Zhangjie Cao, Qixing Huang, and Karthik Ramani. 3D object classification via spherical projections. In *International Conference on 3D Vision (3DV)*, 2017. 38

Susan Carey. *The Origin of Concepts*. Oxford University Press, 2009. 87

Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human Pose Estimation with Iterative Error Feedback. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 15

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012*, 2015. 36, 37, 39, 56, 57, 75, 184, 191

Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A Compositional Object-Based Approach to Learning Physical Dynamics. In *International Conference on Learning Representations (ICLR)*, 2017. 100, 101, 102, 104, 118

Yu-Wei Chao, Jimei Yang, Brian Price, Scott Cohen, and Jia Deng. Forecasting Human Dynamics from Static Images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 151

Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic Reasoning for Assembly-Based 3D Modeling. *ACM Transactions on Graphics (TOG)*, 30(4):35, 2011. 185

Jiawen Chen, Shahram Izadi, and Andrew Fitzgibbon. KinÊtre: Animating the world with the human body. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2012. 33

Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. ABC-CNN: An attention based convolutional neural network for visual question answering. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshop)*, 2016a. 163

Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-Image Depth Perception in the Wild. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016b. 38

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016c. 64, 66

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 172, 229

Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A Unified Approach for Single and Multi-View 3D Object Reconstruction. In *European Conference on Computer Vision (ECCV)*, 2016. 15, 38, 52

Grzegorz Chrupała, Akos Kádár, and Afra Alishahi. Learning Language through Pictures. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015. 162

Taco Cohen, Mario Geiger, and Max Welling. Convolutional Networks for Spherical Signals. In *International Conference on Machine Learning Workshops (ICML Workshop)*, 2017. 38

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations (ICLR)*, 2018. 38, 54

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-Like Environment for Machine Learning. In *Advances in Neural Information Processing Systems Workshops (NeurIPS Workshop)*, 2011. 107

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 65, 76

275

Erwin Coumans. Bullet Physics Engine. *Open Source Software: http://bulletphysics. org*, 2010. 86, 92, 100, 104, 116

Jifeng Dai, Kaiming He, and Jian Sun. Instance-Aware Semantic Segmentation via Multi-Task Network Cascades. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 70

Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The Helmholtz Machine. *Neural Computation*, 7(5):889–904, 1995. 87

Leonardo De Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340. Springer, 2008. 197

Tali Dekel, Tomer Michaeli, Michal Irani, and William T. Freeman. Revealing and Modifying Non-Local Variations in a Single Image. *ACM Transactions on Graphics (TOG)*, 34(6): 227, 2015. 215

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 176, 221, 238, 259

Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-Markup Generation with Coarse-to-Fine Attention. In *International Conference on Machine Learning (ICML)*, 2017. 201

Misha Denil, Pulkit Agrawal, Tejas D. Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to Perform Physics Experiments via Deep Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2017. 102

Emily L. Denton, Soumith Chintala, and Rob Fergus. Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 64, 131

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural Program Learning under Noisy I/o. In *International Conference on Machine Learning (ICML)*, 2017. 186

Daniel D. Dilks, Joshua B. Julian, Jonas Kubilius, Elizabeth S. Spelke, and Nancy Kanwisher. Mirror-Image Sensitivity and Invariance in Object and Scene Processing Pathways. *Journal of Neuroscience*, 31(31):11305–11312, 2011. 184

Carl Doersch. Tutorial on Variational Autoencoders. *arXiv:1606.05908*, 2016. 140

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 163

Li Dong and Mirella Lapata. Language to Logical Form with Neural Attention. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016. 172, 229

276

Alexey Dosovitskiy and Thomas Brox. Generating Images with Perceptual Similarity Metrics Based on Deep Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 74

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to Generate Chairs with Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 12, 16

Yilun Du, Zhijian Liu, Hector Basevi, Ales Leonardis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to Exploit Stability for 3D Scene Parsing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 4

Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2001. 201, 211, 212

Sebastien Ehrhardt, Aron Monszpart, Niloy J. Mitra, and Andrea Vedaldi. Taking Visual Motion Prediction to New Heightfields. *Computer Vision and Image Understanding (CVIU)*, 181:14–25, 2019. 101

David Eigen and Rob Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 38

David Eigen, Christian Puhrsch, and Rob Fergus. Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 72

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised Learning by Program Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 186, 197

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to Infer Graphics Programs from Hand-Drawn Images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 186, 201, 204

S. M. Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 66, 102

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) equivariant representations with spherical CNNs. In *European Conference on Computer Vision (ECCV)*, 2018. 38

Haoqiang Fan, Hao Su, and Leonidas Guibas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 38, 56

Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing Objects by Their Attributes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 39

Nima Fazeli, Miquel Oller, Jiajun Wu, Zheng Wu, Joshua B. Tenenbaum, and Alberto Rodriguez. See, Feel, Act: Hierarchical Learning for Complex Manipulation Skills with Multisensory Fusion. *Science Robotics*, 4(26):eaav3123, 2019. 6

Afsaneh Fazly, Afra Alishahi, and Suzanne Stevenson. A Probabilistic Computational Model of Cross-Situational Word Learning. *Cognitive Science*, 2010. 162

Sanja Fidler, Sven J. Dickinson, and Raquel Urtasun. 3D Object Detection and Viewpoint Estimation with a Deformable 3D Cuboid Model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012. 16

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised Learning for Physical Interaction through Video Prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 101, 131, 139

Jason Fischer, John G. Mikhael, Joshua B. Tenenbaum, and Nancy Kanwisher. Functional Neuroanatomy of Intuitive Physical Inference. *Proceedings of the National Acacdemy of Sciences (PNAS)*, 113(34):E5072–E5081, 2016. 218

David J. Fleet, Michael J. Black, Yaser Yacoob, and Allan D. Jepson. Design and Use of Linear Models for Image Motion Analysis. *International Journal of Computer Vision (IJCV)*, 36(3):171–193, 2000. 128, 130

François Fleuret, Ting Li, Charles Dubout, Emma K Wampler, Steven Yantis, and Donald Geman. Comparing Machines and Humans on a Visual Categorization Test. *Proceedings of the National Acacdemy of Sciences (PNAS)*, 108(43):17621–17625, 2011. 188

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning Visual Predictive Models of Physics for Playing Billiards. In *International Conference on Learning Representations (ICLR)*, 2016. 4, 100, 101, 102, 105

Gilad Freedman and Raanan Fattal. Image and video upscaling from local self-Examples. *ACM Transactions on Graphics (TOG)*, 30(2):12, 2011. 202

William T. Freeman. The Generic Viewpoint Assumption in a Framework for Visual Perception. *Nature*, 368(6471):542, 1994. 60

Christopher Funk and Yanxi Liu. Beyond Planar Symmetry: Modeling Human Perception of Reflection and Rotation Symmetries in the Wild. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 39

Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual Worlds As Proxy for Multi-object Tracking Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 65, 76, 82

Chuang Gan, Yandong Li, Haoxiang Li, Chen Sun, and Boqing Gong. VQS: Linking Segmentations to Questions and Answers for Supervised Attention in VQA and Question-focused Semantic Segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 180, 240

Yossi Gandelsman, Assaf Shocher, and Michal Irani. "Double-DIP": Unsupervised Image Decomposition via Coupled Deep-Image-Priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 202

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Eslami, and Oriol Vinyals. Synthesizing Programs for Images Using Reinforced Adversarial Learning. In *International Conference on Machine Learning (ICML)*, 2018. 186, 201

Siddha Ganju, Olga Russakovsky, and Abhinav Gupta. What's in a Question: Using Visual Questions As a Form of Supervision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 163

Alexander L Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. TerpreT: A probabilistic programming language for program induction. In *Advances in Neural Information Processing Systems Workshops (NeurIPS Workshop)*, 2016. 186

Jon Gauthier, Roger Levy, and Joshua B Tenenbaum. Word Learning and the Acquisition of Syntactic–Semantic Overhypotheses. In *Annual Meeting of the Cognitive Science Society (CogSci)*, 2018. 162, 231

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional Sequence to Sequence Learning. In *International Conference on Machine Learning (ICML)*, 2017. 232

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are We Ready for Autonomous Driving? the Kitti Vision Benchmark Suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 76, 79

Rohit Girdhar, David F. Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a Predictable and Generative Vector Representation for Objects. In *European Conference on Computer Vision (ECCV)*, 2016. 38, 44, 47

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 14, 27

Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018. 167

Daniel Glasner, Shai Bagon, and Michal Irani. Super-Resolution from a Single Image. In *IEEE International Conference on Computer Vision (ICCV)*, 2009. 202

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 46, 47, 74, 131

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. DRAW: A Recurrent Neural Network for Image Generation. In *International Conference on Machine Learning (ICML)*, 2015. 66, 131

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. AtlasNet: A Papier-mâché Approach to Learning 3D Surface Generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 56, 58

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 50

Abhinav Gupta, Alexei A. Efros, and Martial Hebert. Blocks World Revisited: Image Understanding Using Qualitative Geometry and Mechanics. In *European Conference on Computer Vision (ECCV)*, 2010. 101

Jessica B. Hamrick, Andrew J. Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W. Battaglia. Metacontrol for Adaptive Imagination-Based Optimization. In *International Conference on Learning Representations (ICLR)*, 2017. 101

Chi Han, Jiayuan Mao, Chuang Gan, Joshua B. Tenenbaum, and Jiajun Wu. Visual Concept-Metaconcept Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 218

Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical Surface Prediction for 3D Object Reconstruction. In *International Conference on 3D Vision (3DV)*, 2017. 38

Bharath Hariharan and Ross Girshick. Low-Shot Visual Recognition by Shrinking and Hallucinating Features. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 39

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 43, 55, 75, 106, 108, 113, 166, 171, 199, 204, 222, 238

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 71, 75, 166, 167, 171, 204

M. Hejrati and D. Ramanan. Analysis by Synthesis: 3D Object Recognition by Object Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 15

Mohsen Hejrati and Deva Ramanan. Analyzing 3D Objects in Cluttered Images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012. 20

Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early Visual Concept Learning with Unsupervised Deep Learning. *arXiv:1606.05579*, 2016. 147

Irina Higgins, Loic Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations (ICLR)*, 2017. 154

Irina Higgins, Nicolas Sonnerat, Loic Matthey, Arka Pal, Christopher P. Burgess, Matthew Botvinick, Demis Hassabis, and Alexander Lerchner. SCAN: Learning Abstract Hierarchical Compositional Visual Concepts. In *International Conference on Learning Representations (ICLR)*, 2018. 164

Geoffrey E. Hinton and Zoubin Ghahramani. Generative Models for Discovering Sparse Distributed Representations. *Philos. Trans. Royal Soc. B*, 352(1358):1177–1190, 1997. 15

Geoffrey E. Hinton and Drew Van Camp. Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights. In *Conference on Learning Theory (COLT)*, 1993. 147

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. The "Wake-Sleep" Algorithm for Unsupervised Neural Networks. *Science*, 268(5214):1158, 1995. 66, 102, 108

Geoffrey F. Hinton. A Parallel Computation That Assigns Canonical Object-Based Frames of Reference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1981. 33

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. 64, 66, 166

Berthold K. P. Horn and Michael J. Brooks. *Shape from Shading*. MIT press, 1989. 38

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to Reason: End-to-End Module Networks for Visual Question Answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 168

Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable Neural Computation via Stack Neural Module Networks. In *European Conference on Computer Vision (ECCV)*, 2018. 164

Wenze Hu and Song-Chun Zhu. Learning 3D Object Templates by Quantizing Geometry and Appearance Spaces. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 37(6):1190–1205, 2015. 15

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019. 5, 6

Jonathan Huang and Kevin Murphy. Efficient Inference in Occlusion-Aware Generative Models of Images. In *International Conference on Learning Representations Workshops (ICLR Workshop)*, 2015. 66, 102

Qixing Huang, Hai Wang, and Vladlen Koltun. Single-View Reconstruction via Joint Analysis of Image and Shape Collections. *ACM Transactions on Graphics (TOG)*, 34(4):87, 2015. 12, 15, 16, 202

Shiyu Huang and Deva Ramanan. Expecting the Unexpected: Training Detectors for Unusual Pedestrians with Adversarial Imposters. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 144

Drew A. Hudson and Christopher D. Manning. Compositional Attention Networks for Machine Reasoning. In *International Conference on Learning Representations (ICLR)*, 2018. 163, 164, 168, 170, 175, 178, 180, 238, 241

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and Locally Consistent Image Completion. *ACM Transactions on Graphics (TOG)*, 36(4):107, 2017. 201

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 74, 77, 210, 260, 261

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard A. Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew J. Davison, and Andrew W. Fitzgibbon. KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. In Jeffrey S. Pierce, Maneesh Agrawala, and Scott R. Klemmer, editors, *ACM Symposium on User Interface Software and Technology (UIST)*, 2011. 38

Allan Jabri, Armand Joulin, and Laurens van der Maaten. Revisiting Visual Question Answering Baselines. In *European Conference on Computer Vision (ECCV)*, 2016. 180, 241

Max Jaderberg, Karen Simonyan, and Andrew Zisserman. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 20

Wenzel Jakob. Mitsuba Renderer, 2010. http://www.mitsuba-renderer.org. 57, 221

Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V. Gehler. The Informed Sampler: A Discriminative Approach to Bayesian Inference in Generative Computer Vision Models. *Computer Vision and Image Understanding (CVIU)*, 136:32–44, 2015. 66

Michael Janner, Jiajun Wu, Tejas D. Kulkarni, Ilker Yildirim, and Joshua B. Tenenbaum. Self-Supervised Intrinsic Image Decomposition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 3, 38

Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about Physical Interactions with Object-Oriented Prediction and Planning. In *International Conference on Learning Representations (ICLR)*, 2019. 5, 7, 99, 217

Dinesh Jayaraman and Kristen Grauman. Look-ahead before You Leap: End-to-end Active Recognition by Forecasting the Effect of Motion. In *European Conference on Computer Vision (ECCV)*, 2016. 68

Dinesh Jayaraman, Ruohan Gao, and Kristen Grauman. ShapeCodes: Self-Supervised Feature Learning by Lifting Views to Viewgrids. In *European Conference on Computer Vision (ECCV)*, 2018. 39, 56

Zhaoyin Jia, Andy Gallagher, Ashutosh Saxena, and Tsuhan Chen. 3D Reasoning from Blocks to Stability. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 37(5):905–918, 2015. 87, 101

Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. Image Retrieval Using Scene Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 179

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *European Conference on Computer Vision (ECCV)*, 2016a. 74, 120

Justin Johnson, Andrej Karpathy, and Li Fei-Fei. DenseCap: Fully convolutional localization networks for dense captioning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016b. 163

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017a. 162, 163, 168, 170, 174, 175, 229

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Inferring and Executing Programs for Visual Reasoning. In *IEEE International Conference on Computer Vision (ICCV)*, 2017b. 163, 164, 167, 168, 175, 177, 186, 229

Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. Cliplets: Juxtaposing Still and Dynamic Imagery. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2012. 131

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated Task and Motion Planning in Belief Space. *The International Journal of Robotics Research (IJRR)*, 32(9-10):1194–1227, 2013. 217

Abhishek Kar, Shubham Tulsiani, Joao Carreira, and Jitendra Malik. Category-Specific Object Reconstruction from a Single Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 12, 15, 17, 38

Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a Multi-View Stereo Machine. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 33

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D Mesh Renderer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 72, 73, 75

Michael Kazhdan, Bernard Chazelle, David Dobkin, Adam Finkelstein, and Thomas Funkhouser. A Reflective Symmetry Descriptor. In *European Conference on Computer Vision (ECCV)*, 2002. 38

Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Symmetry Descriptors and 3D Shape Matching. In *Eurographics Symposium on Geometry Processing (SGP)*, 2004. 38

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 47, 75, 76, 154, 191, 225

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014. 129, 131, 133, 134, 135, 136, 137

Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-Supervised Learning with Deep Generative Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 134, 135, 136

Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 73

Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models. In *Advances in Neural Information Processing Systems Workshops (NeurIPS Workshop)*, 2014. 163

Kris M. Kitani, De-An Huang, and Wei-Chiu Ma. Activity Forecasting. In *Group and Crowd Behavior for Computer Vision*, pages 273–294. Elsevier, 2017. 102

Kurt Koffka. *Principles of Gestalt Psychology*. Routledge, 2013. 184

George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research (JAIR)*, 2018. 181

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012. 31, 205, 221, 259

Tejas D. Kulkarni, Pushmeet Kohli, Joshua B. Tenenbaum, and Vikash Mansinghka. Picture: A Probabilistic Programming Language for Scene Perception. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015a. 15, 66, 102, 185

Tejas D. Kulkarni, William F. Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep Convolutional Inverse Graphics Network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015b. 15, 64, 66, 100, 164, 197

Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to Detect Unseen Object Classes by Between-Class Attribute Transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 39

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond Pixels Using a Learned Similarity Metric. In *International Conference on Machine Learning (ICML)*, 2016. 74

Yvan G. Leclerc and Martin A. Fischler. An Optimization-Based Approach to the Interpretation of Single Line Drawings As 3D Wire Frames. *International Journal of Computer Vision (IJCV)*, 9(2):113–136, 1992. 15

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 94

Alex X Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic Adversarial Video Prediction. *arXiv:1804.01523*, 2018. 121

Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal of Computer Vision (IJCV)*, 81(2):155, 2009. 42

Adam Lerer, Sam Gross, and Rob Fergus. Learning Physical Intuition of Block Towers by Example. In *International Conference on Machine Learning (ICML)*, 2016. 100, 101, 102, 108, 109, 110

Louis Lettry, Michal Perdoch, Kenneth Vanhoey, and Luc Van Gool. Repeated Pattern Detection Using CNN Activations. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017. 200, 205, 206, 211, 212, 259

Kenneth Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Q. Appl. Math.*, 2(2):164–168, 1944. 42

Beth Levin. *English Verb Classes and Alternations: A Preliminary Investigation.* University of Chicago Press, 1993. 181

Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient Implementation of Marching Cubes' Cases with Topological Guarantees. *J. Graph. Tools*, 8(2):1–15, 2003. 55

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM Transactions on Graphics (TOG)*, 36(4):52, 2017a. 186, 201

Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. GRAINS: Generative recursive autoencoders for INdoor scenes. *ACM Transactions on Graphics (TOG)*, 38(2): 12:1–12:16, 2019a. 201

Wenbin Li, Ales Leonardis, and Mario Fritz. Visual Stability Prediction for Robotic Manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017b. 102

Yangyan Li, Hao Su, Charles Ruizhongtai Qi, Noa Fish, Daniel Cohen-Or, and Leonidas J. Guibas. Joint Embeddings of Shapes and Images via CNN Image Purification. *ACM Transactions on Graphics (TOG)*, 34(6):234, 2015. 13

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. In *International Conference on Learning Representations (ICLR)*, 2019b. 5, 6

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation Networks for Model-Based Control under Partial Observation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019c. 5

Zicheng Liao, Neel Joshi, and Hugues Hoppe. Automated Video Looping with Progressive Dynamism. *ACM Transactions on Graphics (TOG)*, 32(4):77, 2013. 131

Joseph J. Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing Ikea Objects: Fine Pose Estimation. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. 14, 24, 25, 26, 27, 39, 40, 41

Joseph J. Lim, Aditya Khosla, and Antonio Torralba. FPM: Fine Pose Parts-Based Model with 3D CAD Models. In *European Conference on Computer Vision (ECCV)*, 2014. 16

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*, 2014. 180

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 167

Ce Liu. *Beyond Pixels: Exploring New Representations and Applications for Motion Analysis.* PhD thesis, Massachusetts Institute of Technology, 2009. 141, 156

Ce Liu, Jenny Yuen, and Antonio Torralba. SIFT Flow: Dense Correspondence across Scenes and Its Applications. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 33(5):978–994, 2011. 130

Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image Inpainting for Irregular Holes Using Partial Convolutions. In *European Conference on Computer Vision (ECCV)*, 2018a. 201, 212

Jiongxin Liu and Peter N. Belhumeur. Bird Part Localization Using Exemplar-Based Models with Enforced Pose and Subcategory Consistency. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. 22, 23

Shari Liu, Tomer D. Ullman, Joshua B. Tenenbaum, and Elizabeth S. Spelke. Ten-Month-Old Infants Infer the Value of Goals from the Costs of Actions. *Science*, 358(6366):1038–1041, 2017. 218

Yunchao Liu, Zheng Wu, Daniel Ritchie, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to Describe Scenes with Programs. In *International Conference on Learning Representations (ICLR)*, 2019. 6, 8, 186, 199, 201

Zhijian Liu, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Physical Primitive Decomposition. In *European Conference on Computer Vision (ECCV)*, 2018b. 3, 4, 7, 99, 185

Matthew M Loper and Michael J Black. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision (ECCV)*, 2014. 65

David G. Lowe. Three-Dimensional Object Recognition from Single Two-Dimensional Images. *Artificial Intelligence*, 31(3):355–395, 1987. 14, 15

Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual Relationship Detection with Language Priors. In *European Conference on Computer Vision (ECCV)*, 2016. 199

Hongjing Lu and Alan L. Yuille. Ideal Observers for Detecting Motion: Correspondence Noise. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2006. 130

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-Based Neural Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015. 166, 204

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *International Conference on Machine Learning (ICML)*, 2013. 47

M. Malinowski and M. Fritz. A Multi-World Approach to Question Answering about Real-World Scenes Based on Uncertain Input. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 163

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences from Natural Supervision. In *International Conference on Learning Representations (ICLR)*, 2019a. 6, 7, 161, 164, 218

Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *IEEE International Conference on Computer Vision (ICCV)*, 2019b. 8, 199

Junhua Mao, Jiajing Xu, Kevin Jing, and Alan L Yuille. Training and Evaluating Multimodal Word Embeddings with Large-Scale Web Annotated Images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 163

Donald W. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *J. Soc. Ind. Appl. Math.*, 11(2):431–441, 1963. 42

David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* W. H. Freeman and Company, 1982. 3, 36

David Mascharka, Philip Tran, Ryan Soklaski, and Arjun Majumdar. Transparency by Design: Closing the Gap between Performance and Interpretability in Visual Reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 163, 164, 168, 175, 177, 238

Michael Mathieu, Camille Couprie, and Yann LeCun. Deep Multi-Scale Video Prediction beyond Mean Square Error. In *International Conference on Learning Representations (ICLR)*, 2016. 102, 128, 131, 151

John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J. Davison. SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation? In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 16, 33, 38

Tomer Michaeli and Michal Irani. Blind Deblurring Using Internal Patch Recurrence. In *European Conference on Computer Vision (ECCV)*, 2014. 202

Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784*, 2014. 74

Niloy Mitra, Michael Wand, Hao Richard Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-Xing Huang. Structure-Aware Shape Processing. In *SIGGRAPH Asia Courses*. ACM, 2013. 185

Andriy Mnih and Danilo J. Rezende. Variational Inference for Monte Carlo Objectives. In *International Conference on Machine Learning (ICML)*, 2016. 68, 69, 104, 108

Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian Scene Understanding: Unfolding the Dynamics of Objects in Static Images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a. 4, 101

Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "What happens if..." learning to predict the effect of forces in images. In *European Conference on Computer Vision (ECCV)*, 2016b. 101

Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3D bounding box estimation using deep learning and geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. 79

Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. EdgeConnect: Generative image inpainting with adversarial edge learning. *arXiv:1901.00212*, 2019. 201

Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 33

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked Hourglass Networks for Human Pose Estimation. In *European Conference on Computer Vision (ECCV)*, 2016. 15

Gen Nishida, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive Sketching of Urban Procedural Models. *ACM Transactions on Graphics (TOG)*, 35(4):130, 2016. 186

Gen Nishida, Adrien Bousseau, and Daniel G. Aliaga. Procedural Modeling of a Building from a Single Image. *Computer Graphics Forum (CGF)*, 37(2):415–429, 2018. 186

Chengjie Niu, Jun Li, and Kai Xu. Im2Struct: Recovering 3d shape structure from a single rgb image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 201

Fakir S. Nooruddin and Greg Turk. Simplification and Repair of Polygonal Models Using Volumetric Techniques. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 9(2):191–205, 2003. 115

David Novotny, Diane Larlus, and Andrea Vedaldi. Learning 3D Object Categories by Looking around Them. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 38

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder Singh. Action-Conditional Video Prediction Using Deep Networks in Atari Games. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 131

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 2017. 181

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-Symbolic Program Synthesis. In *International Conference on Learning Representations (ICLR)*, 2017. 186

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems Workshops (NeurIPS Workshop)*, 2017. 154

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context Encoders: Feature Learning by Inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 201

Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning Deep Object Detectors from 3D Models. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 39

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 241

Bojan Pepik, Michael Stark, Peter Gehler, and Bernt Schiele. Teaching 3D Geometry to Deformable Part Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 27

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 164, 168, 175

Silvia L. Pintea, Jan C. van Gemert, and Arnold W. M. Smeulders. Deja Vu: Motion Prediction in Static Images. In *European Conference on Computer Vision (ECCV)*, 2014. 130

Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The Curious Robot: Learning Visual Representations via Physical Interactions. In *European Conference on Computer Vision (ECCV)*, 2016. 101, 102

Javier Portilla and Eero P. Simoncelli. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision (IJCV)*, 40 (1):49–70, 2000. 131

Mukta Prasad, Andrew Fitzgibbon, Andrew Zisserman, and Luc Van Gool. Finding Nemo: Deformable Object Class Modelling Using Curve Matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 12, 15

Marc Proesmans, Luc Van Gool, and André Oosterlinck. One-Shot Active 3D Shape Acquisition. In *International Conference on Pattern Recognition (ICPR)*, 1996. 39

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 184, 209

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2016. 47, 128, 131

Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Reconstructing 3D Human Pose from 2D Image Landmarks. In *European Conference on Computer Vision (ECCV)*, 2012. 15

Anurag Ranjan and Michael J. Black. Optical Flow Estimation Using a Spatial Pyramid Network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 106

Scott Reed and Nando De Freitas. Neural Programmer-Interpreters. In *International Conference on Learning Representations (ICLR)*, 2016. 186

Scott E. Reed, Yi Zhang, Yuting Zhang, and Honglak Lee. Deep Visual Analogy-Making. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 81, 140, 142, 150, 151

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 20, 72

Danilo Jimenez Rezende, S. M. Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised Learning of 3D Structure from Images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016a. 38, 44, 68, 72, 102, 104, 108

Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-Shot Generalization in Deep Generative Models. In *International Conference on Machine Learning (ICML)*, 2016b. 66

Gernot Riegler, Ali Osman Ulusoys, and Andreas Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 38

Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah Goodman. Neurally-Guided Procedural Models: Amortized Inference for Procedural Graphics Programs Using Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 185

Lawrence G. Roberts. *Machine Perception of Three-Dimensional Solids*. PhD thesis, Massachusetts Institute of Technology, 1963. 185

Irvin Rock and Stephen Palmer. The Legacy of Gestalt Psychology. *Scientific American*, 263(6):84–91, 1990. 199

Jason Rock, Tanmay Gupta, Justin Thorsen, JunYoung Gwak, Daeyun Shin, and Derek Hoiem. Completing 3D Object Shape from One Depth Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 39

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015. 55, 209, 223, 226

Stefan Roth and Michael J. Black. Fields of Experts: A Framework for Learning Image Priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 128, 130

Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning.* Springer-Verlag, Berlin, Heidelberg, 2004. 121

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The Earth Mover's Distance As a Metric for Image Retrieval. *International Journal of Computer Vision (IJCV)*, 40(2): 99–121, 2000. 191

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 12, 27

Fereshteh Sadeghi, C. Lawrence Zitnick, and Ali Farhadi. VISALOGY: Answering Visual Analogy Questions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 81

Ryusuke Sagawa, Hiroshi Kawasaki, Shota Kiyota, and Ryo Furukawa. Dense One-Shot 3D Reconstruction by Detecting Continuous Regions with Parallel Line Projection. In *IEEE International Conference on Computer Vision (ICCV)*, 2011. 39

Adam N. Sanborn, Vikash K. Mansinghka, and Thomas L. Griffiths. Reconciling Intuitive Physics and Newtonian Mechanics for Colliding Objects. *Psychological Review*, 120(2): 411, 2013. 86

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A Simple Neural Network Module for Relational Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 163, 168

Benjamin Sapp and Ben Taskar. Modec: Multimodal Decomposable Models for Human Pose Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. 14, 15, 21, 22, 23

Scott Satkin, Jason Lin, and Martial Hebert. Data-Driven Scene Understanding from 3D Models. In *British Machine Vision Conference (BMVC)*, 2012. 15

Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video Textures. *ACM Transactions on Graphics (TOG)*, 7(5):489–498, 2000. 131

John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking Deformable Objects with Point Clouds. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013. 87

Adriana Schulz, Ariel Shamir, Ilya Baran, David IW Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. Retrieval on Parametric Shape Collections. *ACM Transactions on Graphics (TOG)*, 36(1):11, 2017. 185

Sebastian Schuster, Ranjay Krishna, Angel Chang, Li Fei-Fei, and Christopher D. Manning. Generating Semantically Precise Scene Graphs from Textual Descriptions for Improved Image Retrieval. In *Conference on Empirical Methods in Natural Language Processing Workshops (EMNLP Workshop)*, 2015. 180, 231, 241

Thomas W Sederberg and Scott R Parry. Free-Form Deformation of Solid Geometric Models. *ACM Transactions on Graphics (TOG)*, 20(4):151–160, 1986. 72

Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a generative model from a single natural image. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 202

Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast Pose Estimation with Parameter-Sensitive Hashing. In *IEEE International Conference on Computer Vision (ICCV)*, 2003. 16

Tianjia Shao, Aron Monszpart, Youyi Zheng, Bongjin Koo, Weiwei Xu, Kun Zhou, and Niloy J. Mitra. Imagining the Unseen: Stability-Based Cuboid Arrangements for Scene Understanding. *ACM Transactions on Graphics (TOG)*, 33(6), 2014. 101

Abhishek Sharma, Oliver Grau, and Mario Fritz. VConv-DAE: Deep Volumetric Shape Learning without Object Labels. In *European Conference on Computer Vision Workshops (ECCV Workshop)*, 2016. 47

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 184, 185, 192, 201

Haoyue Shi, Jiayuan Mao, Tete Xiao, Yuning Jiang, and Jian Sun. Learning Visually-grounded Semantics from Contrastive Adversarial Samples. In *International Conference on Computational Linguistics (COLING)*, 2018. 163

Jian Shi, Yue Dong, Hao Su, and Stella X. Yu. Learning Non-Lambertian Object Intrinsics across ShapeNet Categories. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 38

Kevin J. Shih, Arun Mallya, Saurabh Singh, and Derek Hoiem. Part Localization Using Multi-proposal Consensus for Fine-Grained Categorization. In *British Machine Vision Conference (BMVC)*, 2015. 15, 22, 23

Daeyun Shin, Charless C. Fowlkes, and Derek Hoiem. Pixels, Voxels, and Views: A Study of Shape Representations for Single View 3D Object Shape Prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 39, 56, 57, 58, 59, 60

Assaf Shocher, Nadav Cohen, and Michal Irani. "Zero-Shot" Super-Resolution Using Deep Internal Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 202

Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. InGAN: Capturing and remapping the "DNA" of a natural image. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 202

Abhinav Shrivastava and Abhinav Gupta. Building Part-Based Object Detectors via 3D Geometry. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. 15

N. Siddharth, T. B. Paige, J. W. Meent, A. Desmaison, N. Goodman, P. Kohli, F. Wood, and P. Torr. Learning Disentangled Representations with Semi-Supervised Deep Generative Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 164

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *European Conference on Computer Vision (ECCV)*, 2012. 38

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 74, 75, 120

Kevin A. Smith, Lingjie Mei, Shunyu Yao, Jiajun Wu, Elizabeth Spelke, Joshua B. Tenenbaum, and Tomer D. Ullman. Modeling Expectation Violation in Intuitive Physics with Coarse Probabilistic Object Representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 218

Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning Structured Output Representation Using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 135

Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D. Kulkarni, and Joshua B. Tenenbaum. Synthesizing 3D Shapes via Modeling Multi-View Depth Maps and Silhouettes with Deep Generative Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 15, 56

Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. Semantic Scene Completion from a Single Depth Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 33, 38

Elizabeth S. Spelke. Core Knowledge. *American Psychologist*, 55(11):1233, 2000. 218

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised Learning of Video Representations Using LSTMs. In *International Conference on Machine Learning (ICML)*, 2015. 131, 151

Ondrej Št'ava, Bedrich Beneš, Radomir Měch, Daniel G Aliaga, and Peter Krištof. Inverse Procedural Modeling by Automatic Generation of L-systems. *Computer Graphics Forum (CGF)*, 29(2):665–674, 2010. 185

Hao Su, Qixing Huang, Niloy J. Mitra, Yangyan Li, and Leonidas Guibas. Estimating Image Depth Using Shape Collections. *ACM Transactions on Graphics (TOG)*, 33(4):37, 2014. 12, 15, 16

Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 13, 14, 16, 25, 26, 27

Joseph Suarez, Justin Johnson, and Fei-Fei Li. DDRprog: A CLEVR Differentiable Dynamic Reasoning Programmer. *arXiv:1803.11361*, 2018. 163, 168

Baochen Sun and Kate Saenko. From Virtual to Reality: Fast Adaptation of Virtual Object Detectors to Real Domains. In *British Machine Vision Conference (BMVC)*, 2014. 16

Chen Sun, Per Karlsson, Jiajun Wu, Joshua B. Tenenbaum, and Kevin Murphy. Stochastic Prediction of Multi-Agent Interactions from Partial Observations. In *International Conference on Learning Representations (ICLR)*, 2019. 217

Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. Neural Program Synthesis from Diverse Demonstration Videos. In *International Conference on Machine Learning (ICML)*, 2018a. 186

Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B. Tenenbaum, and William T. Freeman. Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018b. 7, 35, 51, 52, 57, 191

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 80

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2000. 234

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 109, 110

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Web-Scale Training for Face Identification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 20

Marshall F. Tappen, William T. Freeman, and Edward H. Adelson. Recovering Intrinsic Images from a Single Image. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2003. 38

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-View 3D Models from Single Images with a Convolutional Network. In *European Conference on Computer Vision (ECCV)*, 2016. 38

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 38, 52

Olivier Teboul, Loic Simon, Panagiotis Koutsourakis, and Nikos Paragios. Segmentation of Building Facades Using Procedural Shape Priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 200, 206, 211, 212

Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2Face: Real-Time face capture and reenactment of RGB videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 143

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to Infer and Execute 3D Shape Programs. In *International Conference on Learning Representations (ICLR)*, 2019. 6, 8, 183

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A Physics Engine for Model-Based Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. 121

Carlo Tomasi and Takeo Kanade. Detection and Tracking of Point Features. Technical report, Carnegie Mellon University, 1991. 92

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient Object Localization Using Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 15, 19, 20, 22, 23, 24

Jonathan J. Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 19, 22, 23

Antonio Torralba and Alexei A. Efros. Unbiased Look at Dataset Bias. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 25

Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Sharing Visual Features for Multiclass and Multiview Object Detection. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 29(5), 2007. 39

Lorenzo Torresani, Aaron Hertzmann, and Christoph Bregler. Learning Non-Rigid 3D Shape from 2D Motion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2003. 12, 15, 17, 20

Alexander Toshev and Christian Szegedy. DeepPose: Human Pose Estimation via Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 15, 23

Zhuowen Tu and Song-Chun Zhu. Image Segmentation by Data-Driven Markov Chain Monte Carlo. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 24(5): 657–673, 2002. 66

Shubham Tulsiani and Jitendra Malik. Viewpoints and Keypoints. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 14, 16, 27

Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning Shape Abstractions by Assembling Volumetric Primitives. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017a. 110, 112, 184, 185, 192, 194

Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-View Supervision for Single-View Reconstruction via Differentiable Ray Consistency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017b. 33, 38, 44, 51, 52, 56, 58

Tomer D. Ullman, Elizabeth Spelke, Peter Battaglia, and Joshua B. Tenenbaum. Mind Games: Game Engines As an Architecture for Intuitive Physics. *Trends in Cognitive Sciences (TiCS)*, 21(9):649–665, 2017. 86

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep Image Prior. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 201, 202

Anton van den Hengel, Chris Russell, Anthony Dick, John Bastian, Daniel Pooley, Lachlan Fleming, and Lourdes Agapito. Part-Based Modelling of Compound Scenes from Images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 185

Laurens Van der Maaten and Geoffrey Hinton. Visualizing Data Using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9(11):2579–2605, 2008. 32

Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-Embeddings of Images and Language. In *International Conference on Learning Representations (ICLR)*, 2016. 163

Sara Vicente, Joao Carreira, Lourdes Agapito, and Jorge Batista. Reconstructing PASCAL VOC. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 12, 15

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating Visual Representations from Unlabeled Video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a. 130

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating Videos with Scene Dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016b. 131

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 14, 15, 21

Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the Future: Unsupervised Visual Prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 128, 130

Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense Optical Flow Prediction from a Static Image. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 102, 130

Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders. In *European Conference on Computer Vision (ECCV)*, 2016. 130

John Y. A. Wang and Edward H. Adelson. Layered Representation for Motion Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1993. 137, 138, 153

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *European Conference on Computer Vision (ECCV)*, 2018a. 184

Peng Wang, Lingqiao Liu, Chunhua Shen, Zi Huang, Anton van den Hengel, and Heng Tao Shen. Multi-Attention Network for One Shot Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 39

Shaoxiong Wang, Jiajun Wu, Xingyuan Sun, Wenzhen Yuan, William T. Freeman, Joshua B. Tenenbaum, and Edward H. Adelson. 3D shape perception from monocular vision, touch, and shape priors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018b. 4

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018c. 73, 74, 75, 77, 79

Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing Deep Networks for Surface Normal Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 38

Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. Symmetry Hierarchy of Man-Made Objects. *Computer Graphics Forum (CGF)*, 30(2), 2011. 201

Yu-Xiong Wang and Martial Hebert. Learning to Learn: Model Regression Networks for Easy Small Sample Learning. In *European Conference on Computer Vision (ECCV)*, 2016. 39

Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. Visual Interaction Networks: Learning a Physics Simulator from Video. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 118

Yair Weiss. Deriving Intrinsic Images from Image Sequences. In *IEEE International Conference on Computer Vision (ICCV)*, 2001. 38

Yair Weiss and Edward H. Adelson. Slow and Smooth: A Bayesian Theory for the Combination of Local Motion Signals in Human Vision. Technical report, Massachusetts Institute of Technology, 1998. 128, 130

Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-Time Video Completion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. 131

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3-4):229–256, 1992. 65, 68, 69, 73, 103, 114, 167, 174

Jiajun Wu, Ilker Yildirim, Joseph J. Lim, William T. Freeman, and Joshua B. Tenenbaum. Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015a. 3, 4, 7, 15, 66, 85, 102, 130

Jiajun Wu, Joseph J. Lim, Hongyi Zhang, Joshua B. Tenenbaum, and William T. Freeman. Physics 101: Learning Physical Object Properties from Unlabeled Videos. In *British Machine Vision Conference (BMVC)*, 2016a. 3, 4, 7, 85, 102, 130

Jiajun Wu, Tianfan Xue, Joseph J. Lim, Yuandong Tian, Joshua B. Tenenbaum, Antonio Torralba, and William T. Freeman. Single Image 3D Interpreter Network. In *European Conference on Computer Vision (ECCV)*, 2016b. 7, 11, 66, 72

Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016c. 3, 7, 15, 35, 38, 44, 50

Jiajun Wu, Erika Lu, Pushmeet Kohli, William T. Freeman, and Joshua B. Tenenbaum. Learning to See Physics via Visual De-animation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017a. 3, 4, 7, 99, 130

Jiajun Wu, Joshua B. Tenenbaum, and Pushmeet Kohli. Neural Scene De-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017b. 3, 7, 63, 80, 239

Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T. Freeman, and Joshua B. Tenenbaum. MarrNet: 3D Shape Reconstruction via 2.5D Sketches. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017c. 3, 7, 15, 33, 35, 36, 38, 49, 53, 55, 56, 58, 72, 195

Jiajun Wu, Tianfan Xue, Joseph J. Lim, Yuandong Tian, Joshua B. Tenenbaum, Antonio Torralba, and William T. Freeman. 3D interpreter networks for Viewer-Centered wireframe modeling. *International Journal of Computer Vision (IJCV)*, 126(9):1009–1026, 2018a. 3, 7, 11

Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T. Freeman, and Joshua B. Tenenbaum. Learning Shape Priors for Single-View 3D Shape Completion and Reconstruction. In *European Conference on Computer Vision (ECCV)*, 2018b. 3, 7, 35, 38

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015b. 47, 48, 184

Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-Shot Learning-the Good, the Bad and the Ugly. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 39

Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond PASCAL: A Benchmark for 3D Object Detection in the Wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014. 14, 16, 24, 26, 27, 28, 29, 39, 50, 51, 52

Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. ObjectNet3D: A Large Scale Database for 3D Object Recognition. In *European Conference on Computer Vision (ECCV)*, 2016. 39, 42

Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun Database: Large-Scale Scene Recognition from Abbey to Zoo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 24, 28, 221

Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. Synthesizing Dynamic Textures and Sounds by Spatial-Temporal Generative Convnet. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 131

Junyuan Xie, Linli Xu, and Enhong Chen. Image Denoising and Inpainting with Deep Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012. 201

Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. In *European Conference on Computer Vision (ECCV)*, 2016. 131

Wei Xiong, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. Foreground-Aware Image Inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 201

Huijuan Xu and Kate Saenko. Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering. In *European Conference on Computer Vision (ECCV)*, 2016. 163

Zhenjia Xu, Zhijian Liu, Chen Sun, Kevin Murphy, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Unsupervised Discovery of Parts, Structure, and Dynamics. In *International Conference on Learning Representations (ICLR)*, 2019. 5, 7, 127

Tianfan Xue, Jianzhuang Liu, and Xiaoou Tang. Example-Based 3D Object Reconstruction from Line Drawings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 15

Tianfan Xue, Michael Rubinstein, Neal Wadhwa, Anat Levin, Fredo Durand, and William T. Freeman. Refraction Wiggles for Measuring Fluid Depth and Velocity from Video. In *European Conference on Computer Vision (ECCV)*, 2014. 130

Tianfan Xue, Jiajun Wu, Katherine Bouman, and William T. Freeman. Visual Dynamics: Probabilistic Future Frame Synthesis via Cross Convolutional Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 5, 7, 102, 127, 154, 156

Tianfan Xue, Jiajun Wu, Katherine Bouman, and William Freeman. Visual Dynamics: Stochastic Future Generation via Layered Cross Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 2019. 7, 127

Daniel L. K. Yamins, Ha Hong, Charles F. Cadieu, Ethan A. Solomon, Darren Seibert, and James J. DiCarlo. Performance-Optimized Hierarchical Models Predict Neural Responses in Higher Visual Cortex. *Proceedings of the National Acacdemy of Sciences (PNAS)*, 111 (23):8619–8624, 2014. 218

Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2Image: Conditional Image Generation from Visual Attributes. In *European Conference on Computer Vision (ECCV)*, 2016a. 129, 131, 134, 136

Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016b. 38, 44, 72

Zhaoyi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, and Shiguang Shan. Shift-Net: Image inpainting via deep feature rearrangement. In *European Conference on Computer Vision (ECCV)*, 2018. 201

Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. High-Resolution Image Inpainting Using Multi-Scale Neural Patch Synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 201

Jimei Yang, Scott E. Reed, Ming-Hsuan Yang, and Honglak Lee. Weakly-Supervised Disentangling with Recurrent Transformations for 3D View Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 64, 66, 164

Yi Yang and Deva Ramanan. Articulated Pose Estimation with Flexible Mixtures-of-parts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 15, 23

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked Attention Networks for Image Question Answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 163

Shunyu Yao, Tzu-Ming Harry Hsu, Jun-Yan Zhu, Jiajun Wu, Antonio Torralba, William T. Freeman, and Joshua B. Tenenbaum. 3D-Aware scene manipulation via inverse graphics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 3, 7, 56, 63

Hashim Yasin, Umar Iqbal, Björn Krüger, Andreas Weber, and Juergen Gall. A Dual-Source Approach for 3D Pose Estimation from a Single Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 15, 16

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 6, 7, 161, 163, 164, 174, 176, 218, 235, 239, 240

Ilker Yildirim, Winrich Freiwald, and Joshua Tenenbaum. Efficient Inverse Graphics in Biological Face Processing. *bioRxiv*, page 282798, 2018a. 87

Ilker Yildirim, Kevin Smith, Mario Belledonne, Jiajun Wu, and Joshua B. Tenenbaum. Neurocomputational Modeling of Human Physical Scene Understanding. In *Conference on Cognitive Computational Neuroscience (CCN)*, 2018b. 218

Halley Young, Osbert Bastani, and Mayur Naik. Learning Neurosymbolic Generative Models via Program Synthesis. In *International Conference on Machine Learning (ICML)*, 2019. 202

Fisher Yu, Vladlen Koltun, and Thomas A Funkhouser. Dilated Residual Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 73, 75

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative Image Inpainting with Contextual Attention. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 201

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-Form Image Inpainting with Gated Convolution. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 201, 211, 212, 264

Alan Yuille and Daniel Kersten. Vision As Bayesian Inference: Analysis by Synthesis? *Trends in Cognitive Sciences (TiCS)*, 10(7):301–308, 2006. 15, 66, 102

Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, and Jianxiong Xiao. 3DMatch: Learning the Matching of Local 3D Geometry in Range Scans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 15

Renqiao Zhang, Jiajun Wu, Chengkai Zhang, William T. Freeman, and Joshua B. Tenenbaum. A Comparative Evaluation of Approximate Probabilistic Simulation and Deep Neural Networks As Accounts of Human Physical Scene Understanding. In *Annual Meeting of the Cognitive Science Society (CogSci)*, 2016. 101, 218

Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-Time User-Guided Image Colorization with Learned Deep Priors. *ACM Transactions on Graphics (TOG)*, 36(4):119, 2017a. 151

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Networks As a Perceptual Metric. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018a. 77, 78

Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape-from-Shading: A Survey. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 21 (8):690–706, 1999. 38

Weiyu Zhang, Menglong Zhu, and Konstantinos G Derpanis. From Actemes to Action: A Strongly-Supervised Representation for Detailed Action Understanding. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. 144

Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Joshua B. Tenenbaum, William T. Freeman, and Jiajun Wu. Learning to Reconstruct Shapes from Unseen Classes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018b. 3, 7, 35

Zhoutong Zhang, Qiujia Li, Zhengjia Huang, Jiajun Wu, Joshua B. Tenenbaum, and William T. Freeman. Shape and Material from Sound. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017b. 4

Zhoutong Zhang, Jiajun Wu, Qiujia Li, Zhengjia Huang, James Traer, Josh H. McDermott, Joshua B. Tenenbaum, and William T. Freeman. Generative Modeling of Audible Shapes for Object Perception. In *IEEE International Conference on Computer Vision (ICCV)*, 2017c. 4

Bo Zheng, Yibiao Zhao, Joey Yu, Katsushi Ikeuchi, and Song-Chun Zhu. Scene Understanding by Reasoning Stability and Safety. *International Journal of Computer Vision (IJCV)*, 112 (2):221–238, 2015. 87, 101

David Zheng, Vinson Luo, Jiajun Wu, and Joshua B. Tenenbaum. Unsupervised Learning of Latent Physical Properties Using Perception-Prediction Networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018. 5, 130

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 Million Image Database for Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 40(6):1452–1464, 2017a. 211

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene Parsing through ADE20K Dataset. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017b. 73, 75

Tinghui Zhou, Philipp Krähenbühl, Mathieu Aubry, Qixing Huang, and Alexei A. Efros. Learning Dense Correspondence via 3D-Guided Cycle Consistency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a. 16

Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A. Efros. View Synthesis by Appearance Flow. In *European Conference on Computer Vision (ECCV)*, 2016b. 131

Xiaowei Zhou, Spyridon Leonardos, Xiaoyan Hu, and Kostas Daniilidis. 3D Shape Estimation from 2D Landmarks: A Convex Relaxation Approach. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 13, 14, 15, 24, 25, 26, 27

Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-Stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics (TOG)*, 37(4):49, 2018. 201, 202, 211, 212, 264

Chenyang Zhu, Kai Xu, Siddhartha Chaudhuri, Renjiao Yi, and Hao Zhang. SCORES: Shape Composition with Recursive Substructure Priors. *ACM Transactions on Graphics (TOG)*, 37(6):211:1–211:14, 2018a. 185

Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward Multimodal Image-to-Image Translation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017a. 73

Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Joshua B. Tenenbaum, and William T. Freeman. Visual Object Networks: Image Generation with Disentangled 3D Representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018b. 3, 38

Song-Chun Zhu and David Mumford. A Stochastic Grammar of Images. *Foundations and Trends® in Computer Graphics and Vision*, 2(4):259–362, 2007. 66, 102

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven Visual Navigation in Indoor Scenes Using Deep Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017b. 82

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 163

M. Zeeshan Zia, Michael Stark, Bernt Schiele, and Kaspar Schindler. Detailed 3D Representations for Object Recognition and Modeling. *IEEE Transactions on Pattern Analysis and Machine intelligence (TPAMI)*, 35(11):2608–2623, 2013. 15, 16

C. Lawrence Zitnick and Devi Parikh. Bringing Semantics into Focus Using Visual Abstraction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. 80

Maria Zontak and Michal Irani. Internal Statistics of a Single Natural Image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 202

Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 110, 185