Neural Scene De-rendering

Jiajun Wu
MIT CSAIL

Joshua B. Tenenbaum
MIT CSAIL

Pushmeet Kohli
Microsoft Research

Abstract

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. Previous works have used encoder-decoder based neural architectures to learn image representations; however, representations obtained in this way are typically uninterpretable, or only explain a single object in the scene.*

In this work, we propose a new approach to learn an interpretable distributed representation of scenes. Our approach employs a 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 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. Experiments demonstrate that our approach works well on scene de-rendering with two different graphics engines, and our learned representation can be easily adapted for a wide range of applications like image editing, inpainting, visual analogy-making, and image captioning.

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 [4, 21, 35]. 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.

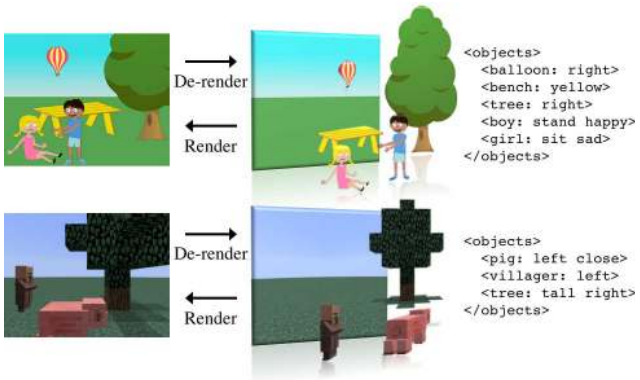


Figure 1: Our goal is to interpret an image in a holistic way. Assuming an image is rendered by a graphics engine on an indefinite length input, we aim to recover the input so that the the exact image can be reconstructed and manipulated. Here we show a simplified version of the XML we use.

In contrast to neural decoders like the ones used in [8, 21], 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 apply the representation to a range of applications like image editing.

This image de-rendering problem, however, is 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 works discussed the use of recurrent networks like LSTM [14] in these cases. However,

for a scene with many objects, it is unintuitive 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 that renderers can take. We evaluate our framework on two datasets with different rendering engines: one is the Abstract Scene dataset [39] and the other is a new dataset we build with Minecraft* images and its 3D renderer.

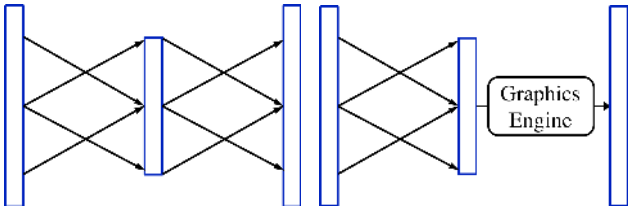
Third, the space of encoded representations and the space of images do not share the same metric: a pair of close latent representations may correspond to images with significantly different visual appearance, and vice versa. Thus, learning a direct mapping from images to labeled representations does not guarantee good performance in reconstruction. In this paper, 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 [22]. To overcome this problem, we use the multi-sample REINFORCE algorithm [32] for optimization.

Our contributions 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 enables multiple applications on two quite different datasets, one of which is a new dataset on the Minecraft platform.

2. Related Work

Our work is closely related to research on learning an interpretable representation with a neural network [13, 21, 35, 4, 33]. Kulkarni *et al*. [21] 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*. [35] and Wu *et al*. [33] explored learning disentangled representations of pose and content from chair images. Chen *et al*. [4] 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 [15, 11, 9, 27, 1], which typically involve recurrent networks like LSTM [14]. Many of these works 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.



(a) A standard autoencoder (b) A generalized autoencoder

Figure 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.

Our framework also relates to the field of "vision as inverse graphics", analysis-by-synthesis, or generative models with data-driven proposals [36, 37, 30, 20, 34, 16], 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 [3] for a nice review of analysis-by-synthesis methods.

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.

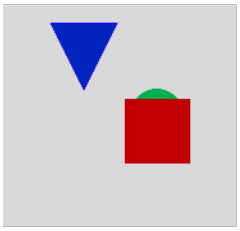
3.1. Generalized Encoding-Decoding Structure

Autoencoder Traditionally autoencoder have neural networks as both the encoder and the decoder, as shown in Figure 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.

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 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

*https://minecraft.net



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

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

representations of input images; and second, minimizing the unsupervised reconstruction error on the rendered images. In Section 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 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 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. We describe details of adapting scene XML to graphics engines in Section 4.

3.2. Black-Box Optimization via REINFORCE

As discussed in Section 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. 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 recent works [26, 1, 17], we formulate this as a reinforcement

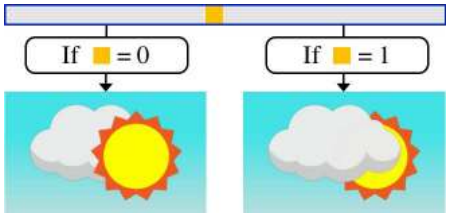


Figure 4: A small change in the latent space (*e.g.*, the depth of cloud) may lead to significant difference in rendered images. It is hence important to consider losses in both spaces.

learning problem, and adopt a multi-sample REINFORCE paradigm [23, 32] 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

$$\Delta w = \alpha(r - b)e, \tag{1}$$

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 [23, 32] 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

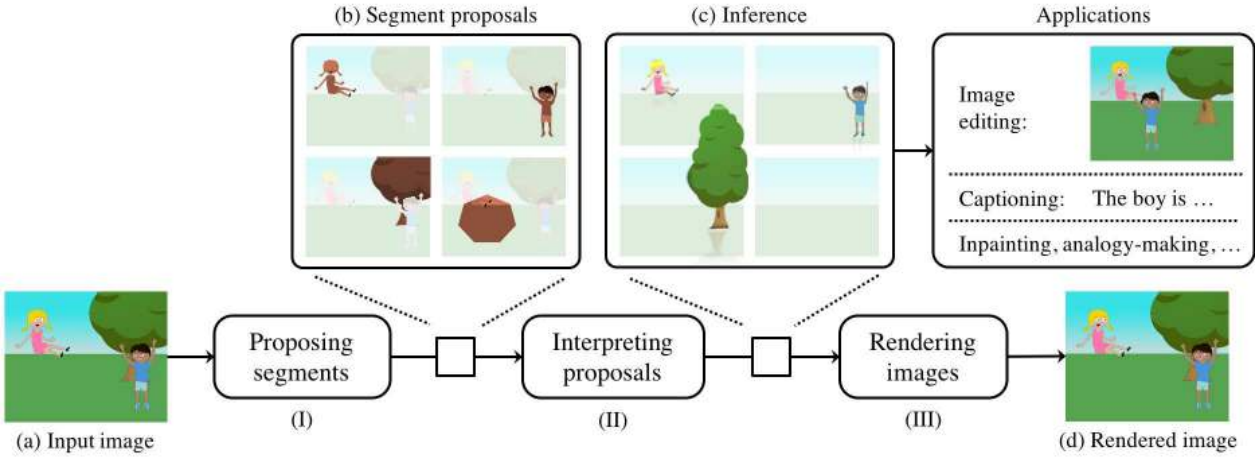


Figure 5: **Our neural scene de-rendering framework** consists of three component. Given an input image, it first generates a number of segment proposals (Stage I). It then tries to interpret if there is an object in the each proposal, and if so what its properties are (Stage II). Eventually, these inference 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*.

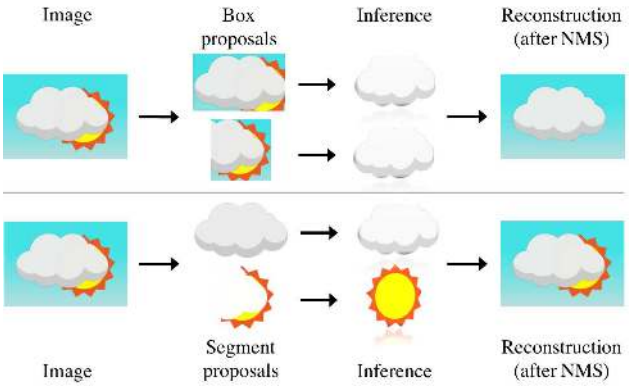


Figure 6: We use segment proposals instead of box proposals, as heavily occluded objects (like the sun in the example) cannot be correctly interpreted from box proposals. During reconstruction, we also need the segment for occluded objects to accurately compute their rewards for REINFORCE.

supervision required for training the encoder through curriculum learning [2]. This semi-supervised learning setting could be useful in practice, where labeled data are often scarce. We show results in Section 4.

3.3. Implementation Details

Network Structure Based on the generalized encoding-decoding structure, our framework has a neural encoder and a graphics engine as a generalized decoder, as shown in Figure 2b. We now describe our encoder in detail, and will provide the description of the two graphics engine decoders we explored for experiments later in Section 4.

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 5-I) produces segment proposals instead of bounding boxes. This is because heavily occluded objects cannot be correctly interpreted from box proposals. For example, in Figure 6, the network is not able to locate and interpret the heavily occluded sun, even with a perfect box proposal. 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 [6]. 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. Please refer to [6] for more details on the structure of the proposal generator. We compute 100 segment proposals for each image.

The object interpreter (Figure 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 5-III).

Analysis-by-Synthesis Refinement When a renderer is available, we may further refine our predictions via analysis-by-synthesis. Here we treat a network prediction as an initialization of a sampling algorithm, for which we use Gibbs sampling in this paper. In each iteration of sampling, we

draw a new sample of the latent representation, render it, and compare the reconstruction with the original image. We sample from a uniform distribution for discrete variables, and from a Gaussian distribution with the initialization as its mean for continuous variables. We run 10 iterations of sampling. Experiments demonstrate that this helps to obtain a more accurate explanation of images. The refinement helps to lower reconstruction error, but often only if most properties have already been inferred correctly. This makes it good for final fine-tuning, while the main de-rendering framework recovers most information.

Training Details Throughout our experiments, we use SGD for optimization, and set our batch size to 50, learning rate to 0.001, momentum to 0.9, and weight decay rate to 10^{-4}. We implement our framework in Torch [5].

4. Evaluations

We now present evaluation results. We start by describing the experimental setup; we then show how our framework performs on scene de-rendering with two different renderers, one used in the Abstract Scene dataset [39], and the other used in Minecraft. We also explain how we build a new Minecraft image dataset, of which Figure 7 shows samples.

4.1. Setup

Methods As described in Section 3, our model uses segment proposals with REINFORCE and analysis-by-synthesis refinement. We first compare our full neural scene de-rendering framework (NSD full) with four simplified ones as an ablation study to reveal how each component contribute to the results. The first two are our framework trained without either analysis-by-synthesis or REINFORCE, one using box proposals (box) and the other using segment proposals (seg). The third is is our segment-based model with only REINFORCE but not analysis-by-synthesis (seg+). The last is our framework in a semi-supervision setting (semi): we first train it using losses in both spaces on 10% randomly sampled training images with labels, and then fine-tune it on the entire training set, but using only the reconstruction loss without any labels of latent representations.

We also compare with two other frameworks: a traditional CNN with a fixed number of dimensions for the latent representation, and an end-to-end CNN+LSTM that aims to encode the image and then to sequentially explain objects from the encoding. Specifically,

- CNN: Our CNN baseline assumes there are no more than X objects in an image, and objects are ordered by their category indices. For an input image, it thus predicts an $X \times Y$ matrix, where Y is the dimension of an object representation in scene XML. Here we use an ResNet-18 model [12] without pre-training.
- CNN+LSTM: Our CNN+LSTM baseline is similar to the captioning model from Karpathy *et al*. [19]. The CNN component, again a ResNet-18 [12], learns an encoding of an input image; the recurrent net, which is an LSTM [14] with a 256-dim latent vector, generates objects sequentially from the image encoding. Here objects are also ordered by their category indices.

Figure 7: **Images in our new Minecraft dataset.** Objects in the dataset vary in size, position, and pose, and may be heavily occluded or cropped.

Evaluation Criteria As discussed in Sections 1 and 3, we would like to minimize both the error in representation inference, and the error in image reconstruction. Note that the reconstruction error, but not the inference error, puts an emphasis on large objects. During evaluation, we compute percentages of incorrectly inferred values in each of the two spaces for every method. For continuous variables, we quantize the space of each into 20 bins, we count an inferred value as correct if it lies in the same bin as the ground truth.

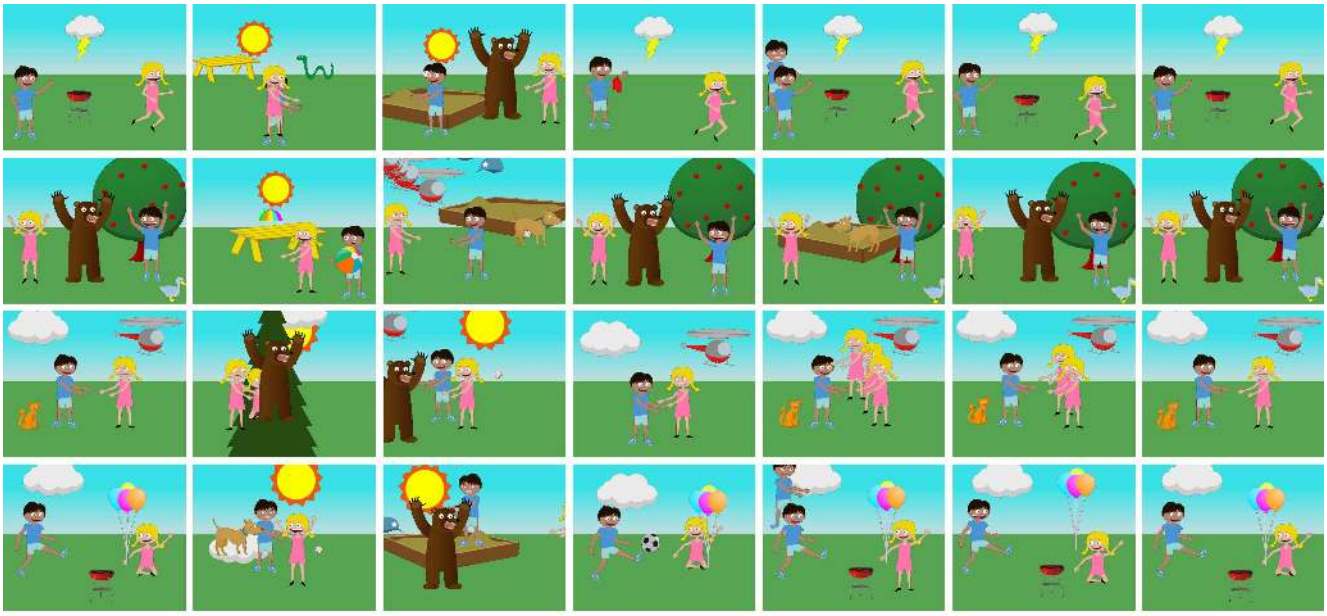
We also conduct a human study, where we present each test image and two reconstructions from different methods to five subjects on Amazon Mechanical Turk, and ask them which looks closer to the original. We then compute, for a pair of methods, how likely one is "preferred" over the other.

4.2. De-rendering Abstract Scenes

Abstract scenes have been an emerging research topic in computer vision [39], natural language processing [24], and reasoning [31]. Rendering engines for abstract scenes are usually efficient, yet still able to capture variations in object appearance and occlusions. We hence choose to first explore our scene de-rendering framework on abstract scenes.

Data We use the Abstract Scene dataset (V1.1), also known as the Mike and Jenny dataset [39]. The dataset contains 1,020 captions, each with 10 images, leading to a total of 10,020 images. Each image has 3 to 18 objects. We randomly sample 90% images for training, and use the rest 10% for testing. Objects are divided into 8 supercategories, each of which contains up to 34 subcategories. These objects have varied appearance, size (determined by depth), and pose; there are often heavy occlusions among them.

Scene XML To connect scene XML to the input for the abstract scene graphics engine, we select the following fields from the XML for each object: category (8-dim), subcategory (34-dim), position (2-dim), depth (quantized into 3 bins), and whether the object faces left or right (1-dim). Each object is therefore characterized by a 48-dim vector.



(a) Input image (b) CNN (c) CNN+LSTM (d) NSD (box) (e) NSD (seg) (f) NSD (seg+) (g) NSD (full)

Figure 8: **Results on the Abstract Scene dataset.** From left to right: (a) input images, and results of (b) the CNN model, (c) the CNN+LSTM model, (d) our de-rendering framework with box proposals, (e) our framework with segment proposals, (f) same as (e) but trained with REINFORCE, and (g) our full model with analysis-by-synthesis refinement on top of (f). See Section 4.1 for details of these methods and Section 4.2 analyses of the results.

	Abstract Scene		Minecraft	
	Inference	Recon	Inference	Recon
CNN	45.73	45.20	41.22	16.59
CNN+LSTM	45.31	41.38	43.52	20.22
NSD (box)	47.85	28.12	32.20	11.42
NSD (seg)	44.19	23.76	32.11	7.71
NSD (seg+)	45.09	22.44	28.79	5.73
NSD (semi)	45.22	21.96	30.05	7.62
NSD (full)	**42.74**	**21.55**	**26.41**	**5.05**

Table 1: **Quantitative results.** We show percentages (%) of incorrectly inferred representation values and reconstructed pixels, for both the Abstract Scene dataset and the Minecraft dataset. We compare methods explained in Section 4.1 and evaluated in Figures 8 and 9, and also a variant of our framework trained in a semi-supervised way. Our full model performs the best, while each component contributes to it.

	Abstract Scene		Minecraft	
	CNN+LSTM	NSD (seg)	CNN+LSTM	NSD (seg)
NSD (seg)	87.2	50.0	57.8	50.0
NSD (full)	96.6	68.6	59.6	53.4

Table 2: **Human study results.** Subjects see the original image and two reconstructed images from different methods. We show percentages (%) of how likely they prefer the left method to the top. We compare three different methods: CNN+LSTM, our framework with segment proposals (NSD seg), and our full model (NSD full). Our full model performs the best consistently. Margins are smaller on the Minecraft dataset because all algorithms perform better.

Results Figure 8 shows qualitative results. The CNN and CNN+LSTM baseline can capture some basic concepts (for example, there is a boy and a girl in the image), but can hardly go beyond those (Figure 8b and c). In contrast, the framework based on box proposals learns to decode most of the objects, though small objects, such as the grill in the first row, are likely to be left out (Figure 8d). Segment proposals help to reconstruct a complete set of objects (Figure 8e), but sometimes with duplicates. This issue gets mostly resolved with REINFORCE (Figure 8f). Analysis-by-synthesis further helps to correct minor deviations (Figure 8g).

We show quantitative results on Table 1. As expected, our full model outperforms the others by a margin in both the space of latent representations and the space of reconstructed images. Also, each component in the framework (segment proposals, REINFORCE, and analysis-by-synthesis) contributes to the performance. Our semi-supervised model performs almost equally well with the fully supervised one.

Table 2 shows results of the human study described in Section 4.1, where we compare three methods: CNN+LSTM, our segment-based framework (NSD seg), and our full model with REINFORCE and analysis-by-synthesis (NSD full). The majority of human subjects also prefer our full model to the one using segment proposals only, while both are better than the CNN+LSTM baseline.

Figure 9: **Results on the Minecraft dataset.** From left to right: (a) input images, and results of (b) the CNN+LSTM model, (c) our de-rendering framework with segment proposals, and (d) our full model with REINFORCE and analysis-by-synthesis. See Section 4.1 for details of these methods and Section 4.3 for analyses.

4.3. De-rendering Minecraft

Minecraft is a popular game where a player controls an agent to interact with a virtual 3D environment. Compared to the Abstract Scene dataset, which is mostly in the 2D space with limited depth information, the Minecraft environment is more realistic for its 3D rendering engine, and its modeling of lighting, shading, and physics to some extent.

Data We introduce a new dataset of Minecraft images using Malmo [18], which allows users to interact with Minecraft by perceiving the environment and sending commands. Our dataset contains 10,000 images, each consisting of 3 to 5 objects. These objects are from a set of 12 entities: pigs, cows, sheep, chicken, wolves, horses, villagers, armor stands, boats, minecarts, and two types of trees. This includes all entities available in Malmo that humans are familiar with (*i.e.*, we exclude entities like monsters).

For each object, we uniformly randomly sample its position and pose. Object positions are represented by r and θ in a polar coordinate system, where the player stands at the origin. Some objects also have their height as an attribute. We do not consider flying or floating objects at this moment, and we set the daytime to noon so that the sun is top in the sky. We then convert the position of each object to $\{x, y, z\}$ in the 3D space (round to 0.1) for Malmo to obtain the image rendered by the Minecraft graphics engine.

Scene XML To connect the scene XML to the Minecraft rendering engine, we select the following fields for each object: category (12-dim), position in the 2D plane (2-dim, $\{r, \theta\}$), height (1-dim), and rotation (3-dim for yaw, pitch, and roll). Each object is thus encoded as a 18-dim vector.

Results Figure 9 and Tables 1 and 2 show qualitative and quantitative results, respectively. Observations here are similar to those in Section 4.2 for the Abstract Scene dataset.

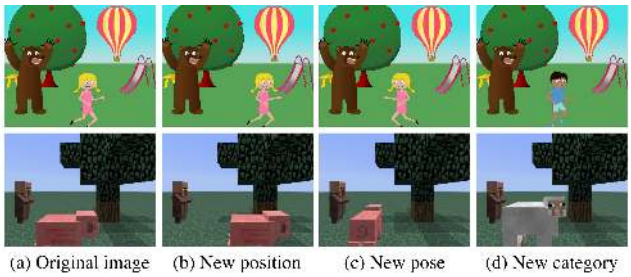


Figure 10: **Results on image editing.** Given an image, we can modify the position, pose, and category of objects with the inferred representation and the graphics engine.

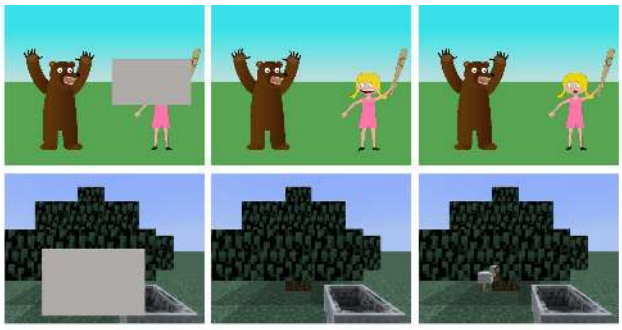


Figure 11: **Results on inpainting.** Our framework performs well, though it fails for almost fully occluded objects or parts. In the future, we may include context modeling to possibly correct some of the errors (*e.g.*, the girl in the first row, when facing a bear, should be surprised or afraid, not happy).

Our full model outperforms the others by obtaining more accurate latent representations and reconstructions.

5. Applications

Our learned representation has extensive applications due to its expressiveness and interpretability. We demonstrate exemplars in image editing, inpainting, visual analogy-making, and image captioning. Our framework obtains good performance in these seemingly irrelevant tasks.

Image Editing Given an image, we can easily make changes to it once we recover its interpretable latent representation. For instance, we show in Figure 10 that we can change the position, pose, and category of an object.

Inpainting Our framework can recover the original image from a corrupted one (Figure 11), even when objects are heavily cropped, *e.g.*, the trees in the second row. As expected, our framework fails to recover objects that are missing entirely from input, such as the girl's facial expression at the top, and the chicken at the bottom. In the future, we may incorporate context modeling to alleviate this issue.

Visual Analogy-Making Visual analogy-making [25], or visalogy [28], is an emerging research topic in AI and vision. A typical setting is to give a system a pair of images A and

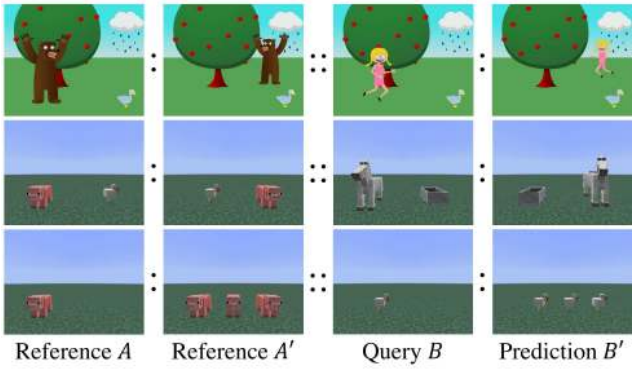


Figure 12: **Results on visual analogy-making.** Given a pair of reference images and a query, our framework can make analogies based on the position and pose of an object (top), and on the number of objects (bottom). See text for details.

A' and an additional source image B, and ask for an analogy B'. While previous works looked into learning analogies between objects, we study the problem of making scene analogies involving multiple objects.

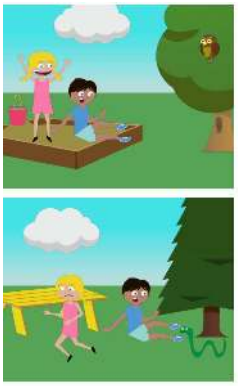
We consider a principled formulation for this seemingly ambiguous problem. Given two image representations Z_A and $Z_{A'}$, we consider their *minimum edit distance* — the minimum number of operations required to derive $Z_{A'}$ from Z_A. We then apply these operations on Z_B to get an analogy $Z_{B'}$. 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 12. 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.

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



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

Figure 13 shows qualitative results, where both LSTM and nearest neighbor perform better when using our distributed representations, compared to using raw pixels.

6. 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 [10, 38]. 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 paper we proposed a neural scene de-rendering framework for image representation learning and reconstruction.

We considered two simple yet rich graphics engines, and proposed a new dataset based on Minecraft. Results proved that our method performed well, and the learned representation has wide applications in a diverse set of vision tasks.

Extending our framework to real world images would require a more flexible scene representation, beyond the current object-attribute formulation, and a more powerful graphics engine, as we assume access to an accurate renderer. Alternatively, we may instead employ an approximate renderer, or both, for scene synthesis and recognition via a Helmholtz-style modeling of wake/sleep phases [7]. This opens up the possibility of an extension to general cases, even when the actual rendering function is not available.

Acknowledgements This work is supported by ONR MURI N00014-16-1-2007, the Center for Brain, Minds and Machines (NSF STC award CCF-1231216), and the Toyota Research Institute. J.W. is supported by an Nvidia fellowship. Part of this work was done when J.W. was an intern at Microsoft Research. We thank Donglai Wei and anonymous reviewers for helpful suggestions.

References

[1] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. In *ICLR*, 2015. 2, 3

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, 2009. 4

[3] T. G. Bever and D. Poeppel. Analysis by synthesis: a (re-) emerging program of research for language and vision. *Biolinguistics*, 4(2-3):174–200, 2010. 2

[4] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016. 1, 2

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. 5

[6] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016. 4

[7] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995. 8

[8] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015. 1

[9] S. Eslami, N. Heess, T. Weber, Y. Tassa, K. Kavukcuoglu, and G. E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *NIPS*, 2016. 2

[10] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016. 8

[11] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *ICML*, 2015. 2

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2015. 5

[13] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158, 1995. 2

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 1, 2, 5

[15] J. Huang and K. Murphy. Efficient inference in occlusion-aware generative models of images. In *ICLR Workshop*, 2015. 2

[16] V. Jampani, S. Nowozin, M. Loper, and P. V. Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *CVIU*, 136:32–44, 2015. 2

[17] D. Jayaraman and K. Grauman. Look-ahead before you leap: end-to-end active recognition by forecasting the effect of motion. In *ECCV*, 2016. 3

[18] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 2016. 7

[19] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, 2015. 5

[20] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. In *CVPR*, 2015. 2

[21] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015. 1, 2

[22] M. M. Loper and M. J. Black. Opendr: An approximate differentiable renderer. In *ECCV*, 2014. 2

[23] A. Mnih and D. J. Rezende. Variational inference for monte carlo objectives. In *ICML*, 2016. 3

[24] L. G. M. Ortiz, C. Wolff, and M. Lapata. Learning to interpret and describe abstract scenes. In *NAACL-HLT*, 2015. 5

[25] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee. Deep visual analogy-making. In *NIPS*, 2015. 7

[26] D. J. Rezende, S. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess. Unsupervised learning of 3d structure from images. In *NIPS*, 2016. 3

[27] D. J. Rezende, S. Mohamed, I. Danihelka, K. Gregor, and D. Wierstra. One-shot generalization in deep generative models. In *ICML*, 2016. 2

[28] F. Sadeghi, C. L. Zitnick, and A. Farhadi. Visalogy: Answering visual analogy questions. In *NIPS*, 2015. 7

[29] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014. 8

[30] Z. Tu and S.-C. Zhu. Image segmentation by data-driven markov chain monte carlo. *IEEE TPAMI*, 24(5):657–673, 2002. 2

[31] R. Vedantam, X. Lin, T. Batra, C. Lawrence Zitnick, and D. Parikh. Learning common sense through visual abstraction. In *ICCV*, 2015. 5

[32] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *MLJ*, 8(3-4):229–256, 1992. 2, 3

[33] J. Wu, T. Xue, J. J. Lim, Y. Tian, J. B. Tenenbaum, A. Torralba, and W. T. Freeman. Single image 3d interpreter network. In *ECCV*, 2016. 2

[34] J. Wu, I. Yildirim, J. J. Lim, W. T. Freeman, and J. B. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *NIPS*, 2015. 2

[35] J. Yang, S. E. Reed, M.-H. Yang, and H. Lee. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *NIPS*, 2015. 1, 2

[36] A. Yuille and D. Kersten. Vision as bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308, 2006. 2

[37] S.-C. Zhu and D. Mumford. A stochastic grammar of images. *Foundations and Trends® in Computer Graphics and Vision*, 2(4):259–362, 2007. 2

[38] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 8

[39] C. L. Zitnick and D. Parikh. Bringing semantics into focus using visual abstraction. In *CVPR*, 2013. 2, 5