Eurographics Symposium on Geometry Processing 2019
D. Bommes and H. Huang
(Guest Editors)

Volume 38 (2019), Number 5

A Convolutional Decoder for Point Clouds using Adaptive Instance Normalization

Isaak Lim† Moritz Ibing† Leif Kobbelt

Visual Computing Institute, RWTH Aachen University

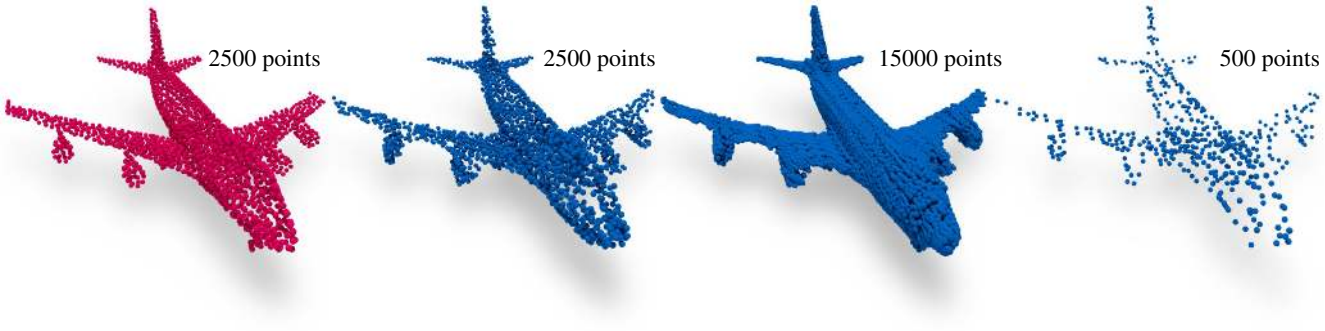

Figure 1: *We show decoding results (blue) for an input shape (red) from the test set. Our convolutional autoencoder with Adaptive Instance Normalization was trained to output 2500 points for inputs with 2500 points. We also visualize outputs from our decoder with a much higher (15000) or lower (500) number of points than the number used during training. Note that with 15000 points we are able to robustly and densely sample the underlying geometry of the input point cloud. Conversely, with 500 points our method is still able to capture the overall shape of the original input.*

Abstract

Automatic synthesis of high quality 3D shapes is an ongoing and challenging area of research. While several data-driven methods have been proposed that make use of neural networks to generate 3D shapes, none of them reach the level of quality that deep learning synthesis approaches for images provide. In this work we present a method for a convolutional point cloud decoder/generator that makes use of recent advances in the domain of image synthesis. Namely, we use Adaptive Instance Normalization and offer an intuition on why it can improve training. Furthermore, we propose extensions to the minimization of the commonly used Chamfer distance for auto-encoding point clouds. In addition, we show that careful sampling is important both for the input geometry and in our point cloud generation process to improve results. The results are evaluated in an auto-encoding setup to offer both qualitative and quantitative analysis. The proposed decoder is validated by an extensive ablation study and is able to outperform current state of the art results in a number of experiments. We show the applicability of our method in the fields of point cloud upsampling, single view reconstruction, and shape synthesis.

CCS Concepts

• ***Computing methodologies*** → ***Shape analysis;*** *Point-based models;*

1. Introduction

The question of how to represent 3D geometry as input for neural networks is still an ongoing field of research. Most recent papers (e.g. [QSMG17, QYSG17, AML18, FELWM18, LBS*18]) focus on how to encode the input in a manner such that its latent representation can then be used for tasks such as classification or segmentation. However, a smaller amount of work has been done on how high-fidelity 3D shapes can be generated by a decoder/generator network. We investigate the problem of generating 3D shapes in an auto-encoding setup. This allows us to evaluate results both qualitatively and quantitatively. While a number of previous works focus

† Equal Contribution

on the encoder, we mainly target the decoder/generator in this paper.

Synthesis of 3D shapes is a time consuming task (especially for non-expert users), which is why a number of data-driven approaches have been proposed to tackle this problem. Methods range from combining parts of a shape collection to create novel configurations over deformation based approaches to the full synthesis of voxelized, meshed or point sampled 3D shapes. While impressive results have been presented, generated 3D shapes have not yet reached a quality that is comparable to the state of the art in image generation, such as recently presented by Karras et al. [KLA19].

We are interested in the complete synthesis of 3D shapes. In particular we investigate the generation of 3D point clouds since voxelized representation incur a heavy memory cost. At the same time we want to benefit from recent advances in generating high-fidelity images. Thus, in this work we propose a convolutional decoder for point clouds. As shown by Groueix et al. [GFK*18], it is difficult to achieve high-quality auto-encoding results by training a naïve point cloud decoder (i.e. a simple multi-layer perceptron). In order to tackle this problem we propose several measures that allow for a better conditioning of the optimization problem.

Our contributions can be summarized as follows.

1. We propose a convolutional decoder for point clouds that is able to outperform current state of the art results on autoencoding tasks.
2. Our autoencoder is able to handle a varying number of points both for its input and output. This property makes it straightforward to apply our architecture to the task of point cloud upsampling.
3. To the best of our knowledge we are the first to apply Adaptive Instance Normalization as used in current image synthesis research [KLA19] to the area of point cloud generation. We give an intuition on why this technique is beneficial to training.
4. We propose several additional losses to the commonly used Chamfer distance that consider both voxel-based and point cloud differences.

Code and our sampling of the ShapeNet Core dataset (v2) [CFG*15] can be found at the project page †.

2. Related Work

Most work on content synthesis with neural networks has been done on images. The natural extension to 3D data is that of a voxel grid. This allows a straightforward transfer of many image based methods (e.g. by replacing 2D with 3D convolutions). Examples are methods that deal with tasks such as single image shape reconstruction [CXG*16], shape completion [HLH*17], and shape generation [WZX*16, LXC*17]. Another option is to represent geometry as planar patches inserted into an Octree [WSLT18]. However, as we are interested in point cloud methods we will restrict our discussion of related work to this domain.

† graphics.rwth-aachen.de/publication/03303

Point-Based Encoders Voxel-based approaches have its drawbacks when it comes to memory consumption, as the required memory scales cubically with the resolution of the grid. To deal with these problems several architectures have emerged, that give up the regular grid structure and instead work directly on unordered point clouds. PointNet [QSMG17] is one of the first among those approaches and does not take any structure or neighbourhood into account. The internal shape representation here is created by aggregating point descriptors. As the relation between nearby points is often important to characterize shape, this work has been extended in PointNet++ [QYSG17] where points are hierarchically grouped based on their neighbourhood and PointNet is applied to those local point clouds. On the other hand, dynamic graph CNNs [WSL*19] encode the information of a local neighborhood via graph convolutions. PCNNs [AML18] generalize convolutions over points via the extension of the convolution operation to continuous volumetric functions. In this manner they benefit from translational invariance and parameter sharing of convolutions, without the drawback of the memory size of high resolution voxel grids. Rethage et al. [RWS*18] propose to combine the advantages of point clouds and grid structures by extracting features from points in the local neighbourhood of each grid cell using a network similar to PointNet. On the resulting representation, 3D convolutions can be applied. As a single grid cell encodes details of the point cloud and not just a binary occupancy value, a low resolution grid is sufficient. This approach is most similar to the encoder used in our framework.

Point Set Generation Most current approaches [FSG17, NW17, ADMG18, GWM18, LCHL18] for the generation of point clouds employ fully connected layers, sometimes in combination with upsampling and convolution layers, to generate a fixed number of points. [LCHL18] employ both a convolution branch to recover the coarse shape and a fully connected branch for the details of the object. Unlike our approach they propose 2D convolutions that result in images with 3 channels, which are interpreted as point coordinates. A different approach is taken by [SUHR17] where instead of learning to output points directly, Sinha et al. propose to learn a mapping from 2D to 3D. By sampling the 2D domain one can obtain a point cloud. This allows the number of generated points to be flexible. Groueix et al. [GFK*18] propose a method that builds on this approach. However, instead of a single mapping a whole atlas of those is learned by training several networks in the style of [SUHR17] that do not share parameters. The loss then ensures that each network learns a different mapping and is responsible for a different part of the shape. We employ a similar point generation technique in the sense that we also learn a mapping from 2D to 3D, instead of using fully connected layers to directly generate a fixed number of points. However, we arrive at these maps in a different manner by generating them per grid cell with our proposed convolutional decoder. A different class of networks recently emerged to represent 3D shapes as an implicit function [PFS*19, CZ19, MON*19]. This function can then be sampled to reconstruct explicit geometry.

3. Convolutional Auto-Encoder for Point Clouds

We want to represent our geometry as point clouds since they can approximate 3D shapes at a higher resolution without incurring the

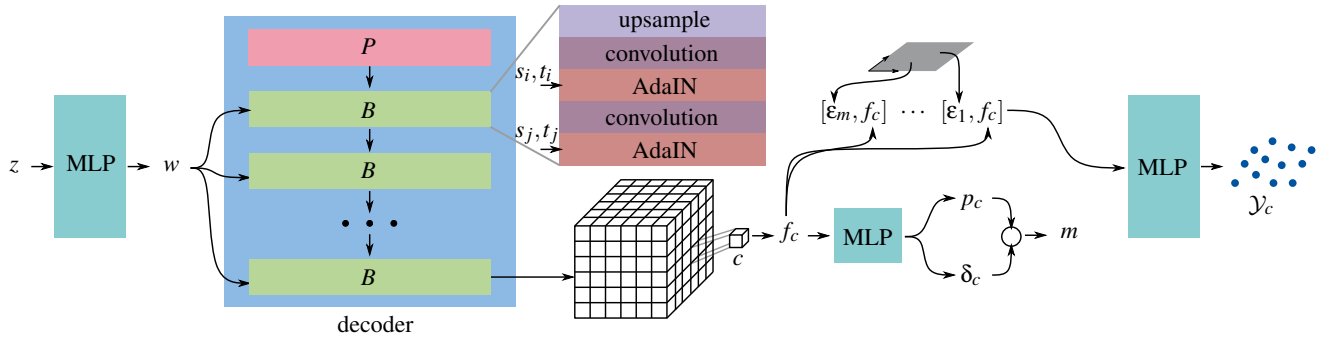

Figure 2: *Overview over our convolutional decoder: Given is some latent vector z produced by an encoder. Passing it through a multi-layer perceptron (MLP) produces w, which consists of a series of scaling and translation parameters $[(s_1,t_1),\dots,(s_l,t_l)]$. P is a learned constant parameter block (in our case it has dimension $512\times2\times2\times2$) used to kickstart the convolutional decoding process. The B blocks each contain an upsampling layer (trilinear by a factor of 2), followed by two convolution and AdaIN layers. The scaling and translation parameters from w are used for each of the l AdaIN layers in the convolutional decoder. The result is a voxel grid where each cell c has a feature vector f_c. Using f_c as input to a MLP we compute the probabilty p_c that c contains any point and the estimated local point cloud density δ_c, which are then used together with the required output size n to determine the number of points m that should be generated for c. We then sample a uniform 2-dimensional distribution (grey plane) m times to produce $\varepsilon_1,\dots,\varepsilon_m$. Each ε_i is concatenated with f_c as input to a MLP which produces a 3-dimensional point. Evaluating the MLP m times produces a point cloud $\mathcal{Y}_c$ for grid cell c.*

memory costs that voxel grids entail. However, we also want to benefit from the advantages of grid structures, enabling the use of convolutional layers and Adaptive Instance Normalization (AdaIN). To this purpose we propose our convolutional decoder (Section 3.1), which starts out with a low resolution grid and successively increases the resolution up to the final desired grid size. We then generate points for each grid cell. Conversely, for our encoder (Section 3.2) we embed the input point cloud into a voxel grid. A network then encodes and stores local parts of the point cloud for each corresponding (closest) grid cell. This voxel grid can then be encoded with a 3D convolutional network.

In traditional convolutional autoencoders the output of the encoder is passed to the decoder, who repeatedly upsamples it in order to produce the reconstruction of the input. This means that even the encoding of fine details of the shape has to pass through the entire decoder, since high- and low-level features are not distinguished. In contrast, our proposed decoder inserts the encoded shape information at various stages of the upsampling process. We will explain our decoder in detail first, followed by the encoder. In order to achieve high-quality results we introduce several additional losses.

3.1. Decoder

Inspired by Karras et al. [KLA19] we propose a convolutional decoder for point clouds based on Adaptive Instance Normalization (AdaIN) as used in a number of style-transfer methods [DSK17, GLK*17, HB17, DPS*18]. Given is an encoder that maps an input point cloud $\mathcal{X}\in\mathbb{R}^{n\times3}$ to a latent vector $z\in\mathbb{R}^{1024}$. A naïve decoder would map z to $\mathcal{Y}\in\mathbb{R}^{m\times3}$ via a multi-layer perceptron (MLP). One problem with this approach is that a series of fully connected layers means adding a large number of parameters to the network.

Another problem is that in order to reconstruct fine detail of $\mathcal{X}$ in $\mathcal{Y}$ every layer of the network is required to preserve the entire shape information. A small change in one of the parameters during back-propagation can have wide-reaching global effects on $\mathcal{Y}$. While, one can reduce the number of parameters used by introducing a convolutional decoder, the problem of the interplay of different parameters during back-propagation remains. Karras et al. [KLA19] show that using AdaIN with a convolutional decoder/generator can produce impressive results for images. An AdaIN layer works by first normalizing its input features and then applying an affine transformation per instance. The transformation parameters are an additional input (e.g. computed from z). In practise, this means that our decoder is constructed via a series of upsampling, convolution, instance normalization [UVL16] and affine feature transformation layers followed by a nonlinearity (see Figure 2). In contrast to traditional convolutional networks, the entire shape specific information is introduced through the affine transformations and is not passed through all layers of the decoder. Instead the upsampling process is applied to a learnable parameter block P. For more details on the architecture see Appendix A.

Thus a given z (by some encoder) is mapped to a vector w that contains the scaling and translation coefficients for each affine feature transformation layer. For every layer i with feature dimension d where AdaIN is applied we select a slice $w_i\in\mathbb{R}^{2d}$. We interpret $w_i=[s_i;t_i]$ such that $s_i,t_i\in\mathbb{R}^d$. As we regard only a single layer, we omit i in the following. The intermediate features $x=x^{(1)}\dots x^{(d)}$ are first normalized and then scaled and translated:

$$\hat{x}^{(k)}=\frac{x^{(k)}-\mu(x^{(k)})}{\sqrt{\sigma^2(x^{(k)})+\varepsilon}}\cdot s^{(k)}+t^{(k)}, \tag{1}$$

where $\mu(x^{(k)})$ and $\sigma^2(x^{(k)})$ are the mean and variance of $x^{(k)}$ over one instance. Since all operations are done for each channel separately, in the following we will omit k for readability.

As a result of this localized interaction the optimization problem becomes more well behaved. Let $\nabla_{\hat{x}}\mathcal{L}$ be the gradient of a loss

function (see Section 3.3) with respect to the output of an intermediate normalization layer. The gradient w.r.t. a single cell i of its input x is given as

$$\nabla_{x_i}\mathcal{L} = \frac{s}{\sqrt{\sigma^2(x)+\varepsilon}}\left(\nabla_{\hat{x}_i}\mathcal{L} - \frac{\mathbb{1}^\intercal(\nabla_{\hat{x}}\mathcal{L})}{m} - \frac{\hat{x}_i(\nabla_{\hat{x}}\mathcal{L})^\intercal\hat{x}}{m}\right), \quad (2)$$

where $\hat{x} \in \mathbb{R}^m$ and m is the number of cells. For a scaling $a \in \mathbb{R}$ and a constant translation $b \cdot \mathbb{1} \in \mathbb{R}^m$, consider the case where $\nabla_{\hat{x}}\mathcal{L} = a \cdot \hat{x} + b \cdot \mathbb{1}$. Then because $\hat{x}$ has zero mean and unit variance

$$\begin{aligned}\nabla_{x_i}\mathcal{L} &= \frac{s}{\sqrt{\sigma^2(x)+\varepsilon}}(a\cdot\hat{x}_i + b - b - a\cdot\hat{x}_i) \quad (3)\\ &= 0.\end{aligned}$$

Thus, there is no gradient w.r.t. a scaling and translation of x running through the normalization layer, which is only natural as such a transformation would be cancelled out by the normalization anyways. AdaIN allows us to set this affine transformation individually for each object. Therefore, the gradient w.r.t. its parameters does not have to pass through the entire decoder. Consequently, the convolutional layers only have to learn non-affine interactions.

3.1.1. Point Cloud Generation

Our proposed convolutional decoder so far only generates volumetric grids. We are however interested in generating point clouds. Therefore, as shown in Figure 2, for each cell c we feed its encoded information f_c into a simple MLP. This MLP predicts two values p_c and δ_c. p_c is a binary variable predicting whether a cell is filled or empty. δ_c is a probability density function (i.e. the likelihood, that a sample should be generated for a particular cell). Distributing the estimation of this information over two variables helps us in dealing with empty cells, as the density prediction seldom actually reaches zero and we thus would introduce points at unwanted locations. For all cells that are classified as filled we then distribute the total number of output samples proportionally to the density estimates of the cells. Thus our network is independent of the number of points we want to generate. This number can even be changed between training and inference (see Figure 1).

The actual generation of points is done in a similar manner to Groueix et al. [GFK*18] and Yin et al. [YHCOZ18]. The idea is to learn a parameterization from a k-dimensional domain to $\mathbb{R}^3$. Then by randomly sampling this domain from a uniform distribution and applying the map, we get our 3-dimensional points. In all our experiments we set $k = 2$, since we assume that locally the shape can be approximated with a surface patch. In practice we apply this map by concatenating the k-dimensional sample to the encoded cell information f_c and feeding the resulting vector into a MLP, which outputs a 3-dimensional point. Thus the MLP represents a map $m_{f_c} : \mathbb{R}^k \to \mathbb{R}^3$ conditioned on f_c. During inference we sample the k-dimensional domain uniformly and then apply a number of steps of Lloyd's algorithm [Llo82] to ensure an even coverage of the space. This further improves our results as shown in Section 4.1. The predicted samples of each cell are offset by the corresponding cell centers.

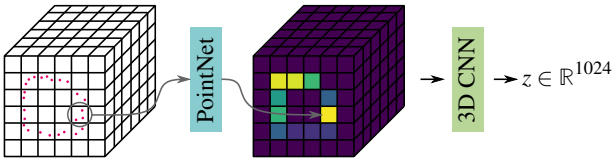

Figure 3: *Our convolutional encoder follows a similar method to Rethage et al. [RWS*18]. We embed the input point cloud (red) into a volumetric grid. For each cell we pass all points within a certain radius from the cell center into a small PointNet. This results in a grid where each cell encodes local point cloud information via a 32-dimensional feature vector. This is visualized as a multi-colored grid. The grid is then passed through a 3D CNN. Through a series of convolution and max-pooling layers we compute an encoding $z \in \mathbb{R}^{1024}$ of the input point cloud.*

3.2. Encoder

For our encoder (see Figure 3) we follow a similar approach to Rethage et al. [RWS*18]. We isotropically normalize the input point cloud such that the longest edge of its axis-aligned bounding box is scaled to the range $[-0.5, 0.5]$. This point cloud $\mathcal{X}$ is then embedded into a volumetric grid consisting of 32^3 cells. For each grid cell we encode the local neighborhood of $\mathcal{X}$ (all points within a radius $r = \frac{\sqrt{3}}{2}$ to the cell center) via a small PointNet (proposed by Qi et al. [QSMG17]). Apart from using fewer number of parameters we also aggregate the final encoding of point clouds by computing the mean of the point features instead of the maximum as proposed in the original paper. Since we make use of a PointNet we are able to handle input point clouds with varying number of points.

This results in a grid where each cell has an η-dimensional feature vector ($\eta = 32$ in all our experiments). This grid can then be passed through a 3D CNN, which consists of a series of convolution, batchnorm and max-pooling layers. The output is an encoding $z \in \mathbb{R}^{1024}$ of $\mathcal{X}$. For more details on the architecture see Appendix A.

3.3. Loss Functions

We define the distance of a point $s_\mathcal{X}$ to a point cloud $\mathcal{Y}$ as

$$d(s_\mathcal{X}, \mathcal{Y}) = \min_{s_\mathcal{Y} \in \mathcal{Y}} \|s_\mathcal{X} - s_\mathcal{Y}\|_2 . \quad (4)$$

In order to compare the input point cloud $\mathcal{X} \in \mathbb{R}^{n\times 3}$ to the reconstructed point cloud $\mathcal{Y} \in \mathbb{R}^{m\times 3}$ we measure the difference with the commonly used Chamfer distance as proposed for point clouds in [FSG17],

$$L_c(\mathcal{X},\mathcal{Y}) = \frac{1}{n}\sum_{s_\mathcal{X}\in\mathcal{X}} d(s_\mathcal{X},\mathcal{Y})^2 + \frac{1}{m}\sum_{s_\mathcal{Y}\in\mathcal{Y}} d(s_\mathcal{Y},\mathcal{X})^2. \quad (5)$$

This gives us a gradient for every point in $\mathcal{Y}$. However, we found that additionally formulating a sharper version of the Chamfer distance benefits training (see Section 4). With the formulation

$$L_p(\mathcal{X},\mathcal{Y}) = \frac{1}{n}\sqrt[p]{\sum_{s_\mathcal{X}\in\mathcal{X}} d(s_\mathcal{X},\mathcal{Y})^p} + \frac{1}{m}\sqrt[p]{\sum_{s_\mathcal{Y}\in\mathcal{Y}} d(s_\mathcal{Y},\mathcal{X})^p}. \quad (6)$$

the gradients of points that incur a larger error are weighted more heavily with $p > 2$. For high p this measure can be seen as similar to the Hausdorff distance. In our experiments we used $p = 5$.

Since $\mathcal{Y}$ is generated by offsetting generated per-cell point clouds $\mathcal{Y}_c$ by the corresponding cell centers c_o, we want to enforce a notion of locality (i.e. each cell only contributes to the part of $\mathcal{Y}$ in its vicinity). Thus we add a loss

$$L_o(\mathcal{Y}) = \sum_c \sum_{s_c \in \mathcal{Y}_c} \max\left(\operatorname{dist}(s_c, c_o) - m, 0\right), \tag{7}$$

This penalizes any generated points that are too far away from their cell centers. We choose $m = \sqrt{3}$ to allow points to be distributed within their generating cell and its direct neighbours.

We cannot directly train the density estimates and filled cell predictions using only the point-wise differences shown above. This is because the differences do not give a gradient w.r.t. the number of points per cell. For this reason we generate ground truth densities and label the filled cells based on the input. Training the MLP that predicts the density δ and probability that a cell c is filled p is done by using the mean squared error

$$L_d(\delta, \hat{\delta}) = \frac{1}{32^3} \sum_c \left(\delta_c - \hat{\delta}_c\right)^2, \tag{8}$$

and the binary cross entropy loss

$$L_f(p, \hat{p}) = -\frac{1}{32^3} \sum_c \hat{p}_c \cdot \log(p_c) + (1 - \hat{p}_c) \cdot \log(1 - p_c) \tag{9}$$

respectively. Here $\hat{\delta}$ and $\hat{p}$ denote the ground truth. Thus our loss during training is

$$\begin{aligned} &\lambda_1 L_c(\mathcal{X}, \mathcal{Y}) + \lambda_2 L_p(\mathcal{X}, \mathcal{Y}) + \lambda_3 L_d(\delta, \hat{\delta}) \\ &\quad + \lambda_4 L_f(p, \hat{p}) + \lambda_5 L_o(\mathcal{Y}) \end{aligned} \tag{10}$$

In all our experiments we chose $\lambda_1 = 1 \times 10^3$, $\lambda_2 = 1 \times 10^1$, $\lambda_3 = 1 \times 10^{10}$, $\lambda_4 = 1 \times 10^2$, and $\lambda_5 = 1$.

4. Experiments

We evaluate our decoder network both by showing the effectiveness of several design choices and by comparing our results with the current state of the art on the task of autoencoding 3D point clouds. All our experiments with our proposed method were done on the ShapeNet dataset [CFG*15], where we evaluated both our method and the methods proposed in [GFK*18,LCHL18]. Additionally, we performed experiments using their respective settings and datasets. This is necessary for a thorough comparison, since prior work employs different datasets, data normalization techniques and evaluation criteria. Furthermore, we can assume that their proposed network architectures were tuned according to the respective datasets. Our networks were trained using AMSGrad [RKK18] ($\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate $= 0.0046$). For evaluation on the testing set we used the network weights that performed best on the validation set. All other networks were trained using the hyperparameters settings suggested in the respective works.

Dataset For our experiments we made use of the official training, validation, and testing split of the ShapeNet Core dataset (v2), which consists of ca. 50k models in 55 different categories. We found that a high quality sampling is important to achieve good results (see Table 1), as the loss is strongly affected by it. Minimizing the Chamfer distance on a non-even sparse sampling does not necessarily mean that we are able to achieve a good approximation of the underlying surface. A large distance from a reconstructed point to the closest target point can be either caused by a great distance to the underlying surface (which we want to penalize) or by the lack of samples in this particular part of the surface (which we do not want to penalize).Therefore, it is desirable that the sampling is as even as possible over the entire shape. To achieve such a sampling, we strongly oversampled the objects uniformly (with roughly 80k points) and then chose a subset (16k points) of those with farthest point sampling.

method	Chamfer dist.
(1) with randomly sampled point clouds	0.387
(2) without AdaIN	0.385
(3) without regularization loss	0.401
(4) without p-norm	0.384
(5) all of the above	0.440
(6) with randomly sampled map	0.390
(7) our method (9 transformations)	0.401
(8) our method (3 transformations)	**0.376**
(9) random sampling	0.227

Table 1: *Evaluation of the different design choices for our network. As can be seen, each additional loss, sampling and architecture choice improves the final result (bold). The reported metric is the Chamfer distance as introduced in section 3.3 multiplied with 1000. To put the numbers into context we compare a random sampling of the same shape with the target.*

As our encoder sorts all points into a grid, we normalize the point clouds to the size of the unit cube centered in the origin. No further data augmentation is applied. All metrics are however computed on unnormalized shapes to simplify future comparisons. When not mentioned otherwise, all distances are reported between point clouds with 2500 points.

4.1. Ablation Study

To motivate our design choices we performed an extensive ablation study, reporting the Chamfer distance obtained on the testing set for different changes in our input, architecture or loss function (Table 1). To show the effect of an evenly distributed point cloud, we trained the network on a uniform random sampling (1) as used in [LCHL18]. We evaluated on our high quality point clouds. To motivate the use of AdaIN, we implemented a strong baseline in the form of a convolutional autoencoder. We used the same encoder as in our proposed network. However, for the decoder we used a convolutional decoder without AdaIN (2) (i.e. z is passed directly into the decoder and P is no longer necessary). To ensure a fair comparison we used a similar number of parameters.

While our proposed architecture enables the possible application of nine layers of AdaIN (7), we found that this lead to some overfitting on the training data. Therefore, we limit the number of affine feature transformations to the first three layers (8). All subsequent

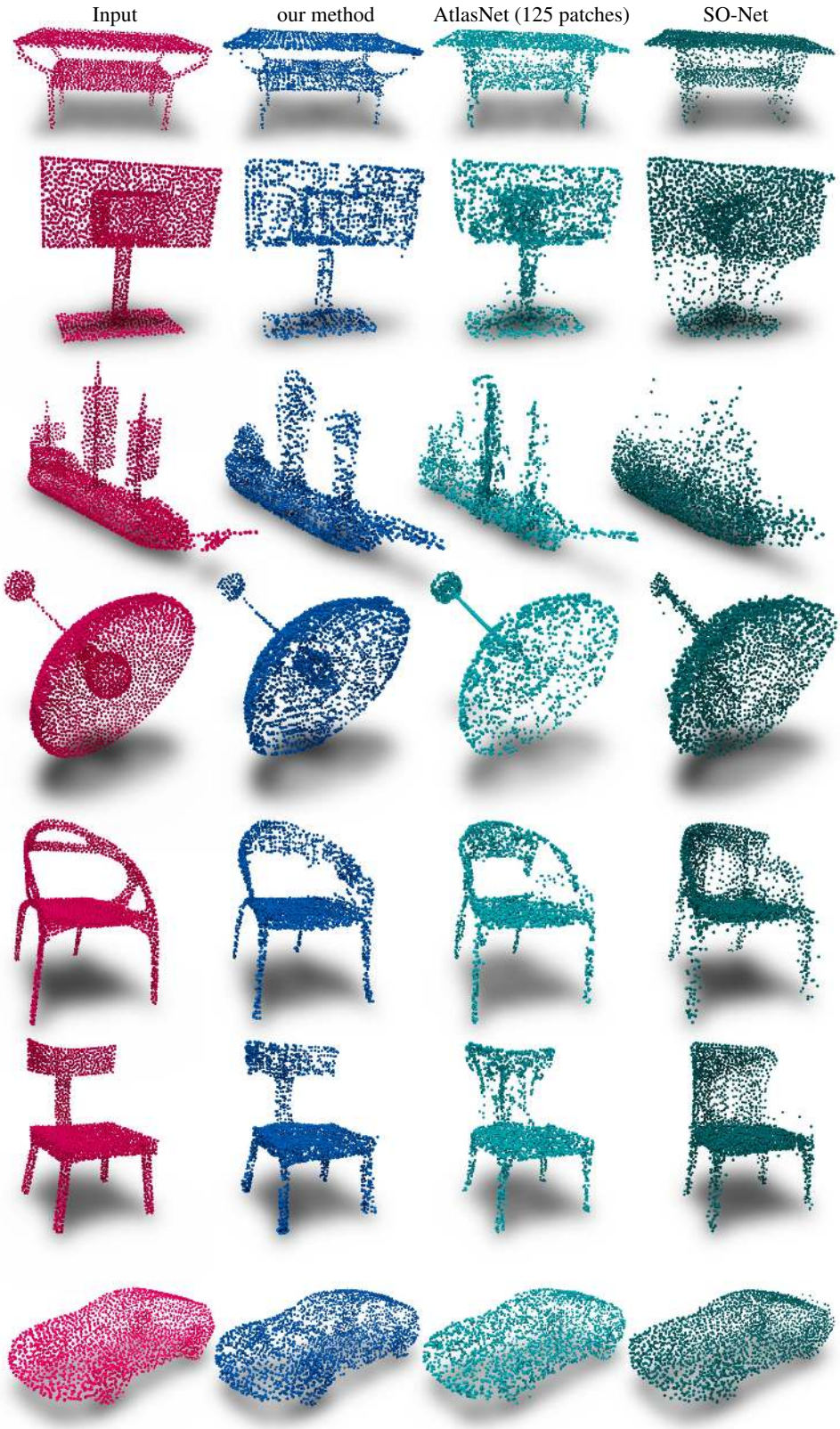

Figure 4: *Qualitive results for different autoencoder models. From left to right: ground truth, our results, [GFK*18], [LCHL18]. Note that our method produces less spurious points and reproduces sharper surface details.*

method	Chamfer dist.
SO-Net (4608 points)	0.603
SO-Net (2500 points via random subsampling)	0.708
SO-Net (2500 points via farthest point sampling)	0.691
AtlasNet (125 patches)	0.408
our method	**0.376**

Table 2: *Comparison of our method against SO-Net [LCHL18] and AtlasNet [GFK*18] on our dataset. The reported number is the Chamfer distance multiplied by 1000.*

outputs of instance normalization layers are not scaled and translated. This architecture achieved the best result (marked in bold in Table 1).

To show the effectiveness of the additionally introduced losses, we trained networks without them and show the difference in the resulting Chamfer distance (3,4). For further comparison, we trained a network in a fairly simple manner by only using the chamfer distance as a loss and no AdaIN on randomly sampled point clouds (5). Finally, we show that sampling the learned map from 2D to 3D at fixed, well distributed positions (as done in [GFK*18]) instead of randomly during inference further improves the results (6). Not using the cell classification loss has a minor negative impact on the results in the order of the fourth decimal. To put these numbers into context, we compare a random sampling of the shape with the ground truth (9).

4.2. Comparison

We compare against AtlasNet [GFK*18] and SO-Net [LCHL18] both on our own dataset (Table 2) as well as on their respective datasets (Table 3). For AtlasNet we trained their best performing network (125 Patches) on our dataset. SO-Net does not allow to output point clouds with 2500 points without changing the suggested architecture. Instead, we compare against the two presented versions of the network. One generates 1280 points (Table 3) and one has an output size of 4608 points (Table 2). The numbers reported in their paper are from a network outputting 1280 points, consequently we trained ours similarly (i.e. 1024 input points and 1280 output points). Furthermore, they use a slightly different definition of the Chamfer distance. They compute the Euclidean distance between closest points instead of its squared version. For a fair comparison on our dataset we report the Chamfer distance between a target of 2500 points and the entire point cloud (4608 points) as well as subsamplings (2500 points) of it.

Note that the computed distances are not comparable across datasets due to differences in normalization and evaluation methods. As can be seen in Tables 2 and 3 our method outperforms AtlasNet and SO-Net on our dataset as well as on the ones used by the respective authors. Qualitative results are shown in Figure 4. For these examples, our method is less prone to produce outliers and reconstructs the shape contours more faithfully.

5. Applications

To demonstrate the usefulness of our convolutional decoder we show results in three applications. Our hyper-parameters and architecture were not tuned particularly for these demonstrations. We expect that with more carefully chosen settings, better results could be achieved.

method	Chamfer dist.
AtlasNet (25 Patches)	1.56
AtlasNet (125 Patches)	1.51
our method	**1.42**
SO-Net (1280 points)	0.033
our method (1280 points)	**0.030**

Table 3: *Comparison against AtlasNet and SO-Net on their respective datasets. Our models were trained without any additional hyper-parameter tuning. The reported number for the comparison against AtlasNet is the Chamfer distance multiplied by 1000. The comparison against SO-Net is based on the Chamfer distance as reported in their paper [LCHL18].*

Figure 5: *We show some qualitative results for single view reconstruction. The input images are shown on the left. Reconstruction results are visualized in blue. The ground truth is rendered in green.*

Single View Reconstruction For single view reconstruction (see Figure 5) we follow [CXG*16] and use a subset of ShapeNet consisting of 13 classes. To be comparable we use their rendered views, as well as their sampling. Similar to [GWM18] we used a pretrained VGG-11 [SZ15] as an encoder. The rest of our network is unchanged to the autoencoder setting. We manage to achieve competitive quantitative results as shown in Table 4.

method	Chamfer dist.
Fan et al. [FSG17]	4.128
Lin et al. [LKL18]	3.547
MRTNet [GWM18]	**3.088**
our method	3.398

Table 4: *Quantitative results for Single View Reconstruction. The reported numbers are Chamfer distance (as defined in [LCHL18]), scaled by 100, computed on point clouds of size 4096*

Figure 6: *Qualitative results for our point cloud upsampling. Severely under-sampled input point clouds (50 points) are shown in red. The network predictions and ground truth point clouds are shown in blue and green respectively (16000 points).*

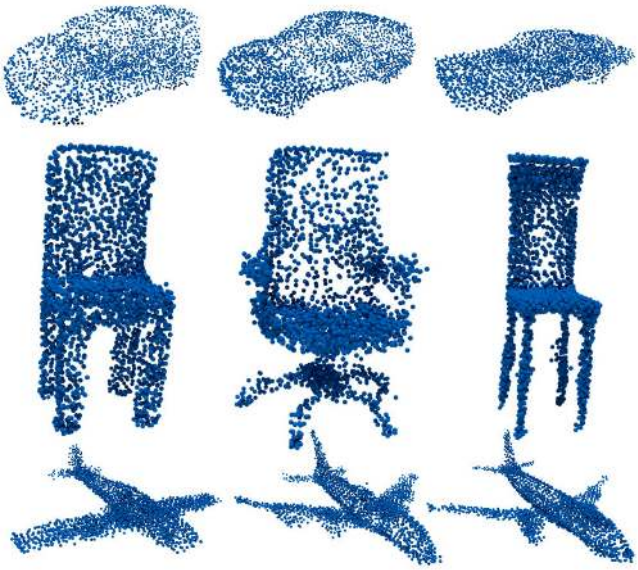

Figure 7: *Here we show some samples generated with our conditional VAE for the categories "car", "chair", and "airplane".*

Point Cloud Upsampling As our network architecture is indifferent to the number of input or output points, it is straightforward to use our model for the task of point cloud upsampling. We train our network on our training set to take between 50 and 500 input points, but output 5000. Although there are several methods that use neural networks for point cloud upsampling [YWH*19, YLF*18], their setting is different as they regard local patches of the geometry and compute a denser sampling there. In contrast, we regard the shape as a whole. As a result these methods require the input to be sampled densely enough that local patches convey geometric meaning. For our method it is sufficient that the general shape is conveyed in order to get results of a good quality. We demonstrate this on severely under-sampled point clouds of the test set with only 50 points as input (Figure 6). Note that our method is able to robustly output point clouds of size 16000 even though the network was trained to output 5000 points.

Point Cloud Synthesis Our decoder can not only be used to reconstruct point clouds for a given input but is also able to generate new shapes as well. A commonly used generative model is the variational autoencoder (VAE) as proposed by Kingma et al. [KW14]. We implemented a conditional VAE version of our network, with only minor changes to the original autoencoder. Conditioning on different classes is done by passing the category as a one-hot encoding vector into a MLP, which generates P (see Figure 2). The latent vector z is sampled from a multivariate Gaussian, whose parameters are predicted by the encoder. This allows us to sample the latent space in order to generate shapes for a specified category as shown in Figure 7.

6. Conclusion

In this work we have introduced a convolutional decoder that can generate high quality point clouds of arbitrary size. Our method is able to achieve state of the art results for auto-encoding tasks by making use of the benefits offered by AdaIN, careful consideration of even sampling, as well as several additions to the Chamfer distance as losses. We outline several possible applications for our method in the fields of single view reconstruction, point cloud upsampling and synthesis.

Our architecture inherits some of the common limitations that come with voxel-based representations. That is, our method is not invariant to rotations of the input and could incur a larger memory cost at higher grid resolutions. However, we show that with a relatively low resolution (32^3) we are able to generate results of a high quality. Furthermore, we approximate the geometry in each filled grid cell as a surface patch. For locally more complex geometries this might be a limitation.

Nevertheless, we are convinced that our method is useful in future research on 3D shape synthesis. One direction is the use of a generator similar to our decoder in the setting of generative adversial networks (GANs) as originally proposed by Goodfellow et al. [GPAM*14]. Another interesting research direction are more detailed shape modifications enabled by affine feature transformations at varying levels of detail.

Acknowledgements The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement n° [340884], as well as the Deutsche Forschungsgemeinschaft DFG – 392037563.

Appendix A: Network Architecture

Our encoder consists of a small PointNet and an 3D CNN. The PointNet is constructed as FC8-FC16-FC32-FC32. FCx is a fully connected layer (in this case without bias) with output dimensionality x. After every fully connected layer we apply batchnorm as

proposed by Ioffe and Szegedy [IS15]. We also apply the exponential linear unit (ELU) as an activation function as proposed by Clevert et al. [CUH16] after every batchnorm layer except for the last one. In order to construct the final 32-dimensional feature vector for each cell we compute the mean feature instead of taking the maximum.

The 3D CNN is constructed as C64-C64-C64-MP-C128-C128-MP-C256-C256-MP-C512-C512-MP-C512-C1024. Cx is a 3D convolution layer with kernel size $3 \times 3 \times 3$, zero-padding of 1, stride of 1, and output feature dimensionality x. For C1024 we use no padding and a kernel size of $2 \times 2 \times 2$ in order to reduce the output to a 1024-dimensional vector. We do not use bias for our convolution operations. After every convolution layer we apply batchnorm and ELU. MP refers to a max-pooling layer with kernel size $2 \times 2 \times 2$ and stride 1.

For our decoder we use a fully connected layer with bias to map z to w.The convolutional decoder is constructed as P-C512-U-C512-C256-U-C256-C128-U-C128-C64-U-C64-C62. P refers to the learnable constant parameter block of size $512 \times 2 \times 2 \times 2$. C$x$ refers to 3D convolution layers with output feature dimensionality x, kernel size $3 \times 3 \times 3$, stride of 1, and zero-padding of 1. We do not use bias for our convolution operations. After every convolution and P we apply dropout as proposed by Srivastava et al. [SHK*14] with a probabilty of 0.2. AdaIN is applied after every dropout layer and after P with the scaling and translation parameters provided by w. For every convolution layer we apply ELU after AdaIN.

Our point cloud generation MLP is structured as FC64-FC64-FC32-FC32-FC16-FC16-FC8-FC3. We apply ELU after every FC layer except for the last one.

The MLP that estimates the density and classifies whether a grid cell contains points or not is constructed as FC16-FC8-FC4-FC2. After every fully connected layer we apply batchnorm and ELU except for the last one.

References

[ADMG18] ACHLIOPTAS P., DIAMANTI O., MITLIAGKAS I., GUIBAS L. J.: Learning representations and generative models for 3d point clouds. *International Conference on International Conference on Machine Learning* (2018). 2

[AML18] ATZMON M., MARON H., LIPMAN Y.: Point convolutional neural networks by extension operators. *ACM Transactions on Graphics 37* (03 2018). 1, 2

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., ET AL.: Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015). 2, 5

[CUH16] CLEVERT D.-A., UNTERTHINER T., HOCHREITER S.: Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations* (2016). 9

[CXG*16] CHOY C. B., XU D., GWAK J., CHEN K., SAVARESE S.: 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *European Conference on Computer Vision* (2016), 628–644. 2, 7

[CZ19] CHEN Z., ZHANG H.: Learning implicit fields for generative shape modeling. *IEEE Conf. on Computer Vision and Pattern Recognition* (2019). 2

[DPS*18] DUMOULIN V., PEREZ E., SCHUCHER N., STRUB F., VRIES H. D., COURVILLE A., BENGIO Y.: Feature-wise transformations. *Distill 3*, 7 (2018), e11. 3

[DSK17] DUMOULIN V., SHLENS J., KUDLUR M.: A learned representation for artistic style. *International Conference on Learning Representations* (2017). 3

[FELWM18] FEY M., ERIC LENSSEN J., WEICHERT F., MÜLLER H.: Splinecnn: Fast geometric deep learning with continuous b-spline kernels. *IEEE Conf. on Computer Vision and Pattern Recognition* (2018), 869–877. 1

[FSG17] FAN H., SU H., GUIBAS L. J.: A point set generation network for 3d object reconstruction from a single image. *IEEE Conf. on Computer Vision and Pattern Recognition* (2017), 2463–2471. 2, 4, 7

[GFK*18] GROUEIX T., FISHER M., KIM V. G., RUSSELL B., AUBRY M.: AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. *IEEE Conf. on Computer Vision and Pattern Recognition* (2018). 2, 4, 5, 6, 7

[GLK*17] GHIASI G., LEE H., KUDLUR M., DUMOULIN V., SHLENS J.: Exploring the structure of a real-time, arbitrary neural artistic stylization network. *British Machine Vision Conference* (2017). 3

[GPAM*14] GOODFELLOW I., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative adversarial nets. *Advances in Neural Information Processing Systems* (2014), 2672–2680. 8

[GWM18] GADELHA M., WANG R., MAJI S.: Multiresolution tree networks for 3d point cloud processing. *European Conference on Computer Vision* (2018). 2, 7

[HB17] HUANG X., BELONGIE S.: Arbitrary style transfer in real-time with adaptive instance normalization. *IEEE International Conference on Computer Vision* (2017), 1501–1510. 3

[HLH*17] HAN X., LI Z., HUANG H., KALOGERAKIS E., YU Y.: High-resolution shape completion using deep neural networks for global structure and local geometry inference. *IEEE Conf. on Computer Vision and Pattern Recognition* (2017), 85–93. 2

[IS15] IOFFE S., SZEGEDY C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. *International Conference on International Conference on Machine Learning 37* (2015), 448–456. 9

[KLA19] KARRAS T., LAINE S., AILA T.: A style-based generator architecture for generative adversarial networks. *IEEE Conf. on Computer Vision and Pattern Recognition* (2019). 2, 3

[KW14] KINGMA D. P., WELLING M.: Auto-encoding variational bayes. *International Conference on Learning Representations* (2014). 8

[LBS*18] LI Y., BU R., SUN M., WU W., DI X., CHEN B.: Pointcnn: Convolution on x-transformed points. *Advances in Neural Information Processing Systems* (2018), 828–838. 1

[LCHL18] LI J., CHEN B. M., HEE LEE G.: So-net: Self-organizing network for point cloud analysis. *IEEE Conf. on Computer Vision and Pattern Recognition* (2018), 9397–9406. 2, 5, 6, 7

[LKL18] LIN C.-H., KONG C., LUCEY S.: Learning efficient point cloud generation for dense 3d object reconstruction. *Thirty-Second AAAI Conference on Artificial Intelligence* (2018). 7

[Llo82] LLOYD S.: Least squares quantization in pcm. *IEEE transactions on information theory 28*, 2 (1982), 129–137. 4

[LXC*17] LI J., XU K., CHAUDHURI S., YUMER E., ZHANG H., GUIBAS L.: Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics 36*, 4 (2017), 52. 2

[MON*19] MESCHEDER L., OECHSLE M., NIEMEYER M., NOWOZIN S., GEIGER A.: Occupancy networks: Learning 3d reconstruction in function space. *IEEE Conf. on Computer Vision and Pattern Recognition* (2019). 2

[NW17] NASH C., WILLIAMS C. K.: The shape variational autoencoder: A deep generative model of part-segmented 3d objects. *Computer Graphics Forum 36*, 5 (2017), 1–12. 2

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: Deepsdf: Learning continuous signed distance functions for shape representation. *IEEE Conf. on Computer Vision and Pattern Recognition* (2019). 2

[QSMG17] QI C. R., SU H., MO K., GUIBAS L. J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. *IEEE Conf. on Computer Vision and Pattern Recognition 1*, 2 (2017), 4. 1, 2, 4

[QYSG17] QI C. R., YI L., SU H., GUIBAS L. J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems* (2017), 5099–5108. 1, 2

[RKK18] REDDI S. J., KALE S., KUMAR S.: On the convergence of adam and beyond. *International Conference on Learning Representations* (2018). 5

[RWS*18] RETHAGE D., WALD J., STURM J., NAVAB N., TOMBARI F.: Fully-convolutional point networks for large-scale point clouds. *European Conference on Computer Vision* (2018). 2, 4

[SHK*14] SRIVASTAVA N., HINTON G., KRIZHEVSKY A., SUTSKEVER I., SALAKHUTDINOV R.: Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15* (2014), 1929–1958. 9

[SUHR17] SINHA A., UNMESH A., HUANG Q., RAMANI K.: Surfnet: Generating 3d shape surfaces using deep residual networks. *IEEE Conf. on Computer Vision and Pattern Recognition* (2017), 6040–6049. 2

[SZ15] SIMONYAN K., ZISSERMAN A.: Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations* (2015). 7

[UVL16] ULYANOV D., VEDALDI A., LEMPITSKY V.: Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (2016). 3

[WSL*19] WANG Y., SUN Y., LIU Z., SARMA S. E., BRONSTEIN M. M., SOLOMON J. M.: Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics* (2019). 2

[WSLT18] WANG P.-S., SUN C.-Y., LIU Y., TONG X.: Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. *ACM Transactions on Graphics 37*, 6 (2018). 2

[WZX*16] WU J., ZHANG C., XUE T., FREEMAN B., TENENBAUM J.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in Neural Information Processing Systems* (2016), 82–90. 2

[YHCOZ18] YIN K., HUANG H., COHEN-OR D., ZHANG H.: P2p-net: Bidirectional point displacement net for shape transform. *ACM Transactions on Graphics 37*, 4 (2018), 152:1–152:13. 4

[YLF*18] YU L., LI X., FU C.-W., COHEN-OR D., HENG P.-A.: Pu-net: Point cloud upsampling network. *IEEE Conf. on Computer Vision and Pattern Recognition* (2018), 2790–2799. 8

[YWH*19] YIFAN W., WU S., HUANG H., COHEN-OR D., SORKINE-HORNUNG O.: Patch-based progressive 3d point set upsampling. *IEEE Conf. on Computer Vision and Pattern Recognition* (2019). 8