This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Cloud Transformers: A Universal Approach To Point Cloud Processing Tasks

Kirill Mazur[1] Victor Lempitsky[1, 2*]
[1]Samsung AI Center Moscow [2]Skolkovo Institute of Science and Technology (Skoltech)

Abstract

We present a new versatile building block for deep point cloud processing architectures that is equally suited for diverse tasks. This building block combines the ideas of spatial transformers and multi-view convolutional networks with the efficiency of standard convolutional layers in two and three-dimensional dense grids. The new block operates via multiple parallel heads, whereas each head differentiably rasterizes feature representations of individual points into a low-dimensional space, and then uses dense convolution to propagate information across points. The results of the processing of individual heads are then combined together resulting in the update of point features. Using the new block, we build architectures for both discriminative (point cloud segmentation, point cloud classification) and generative (point cloud inpainting and image-based point cloud reconstruction) tasks. The resulting architectures achieve state-of-the-art performance for these tasks, demonstrating the versatility of the new block for point cloud processing.

1. Introduction

In this work, we consider recognition and generation tasks for point clouds, such as semantic segmentation or image-based reconstruction. Most state-of-the-art architectures for point cloud processing are derived from convolutional neural networks (ConvNets) [20] and are inspired by the success of ConvNets in image processing tasks. Such ConvNet adaptations are based on direct rasterization of point clouds onto regular grids followed by convolutional pipelines [38, 8], as well as on generalizations of the convolutional operators to irregularly sampled data [25, 50] or non-rectangular grids [18, 16].

In this work, we propose a new building block (a *cloud transform* block) for point cloud processing architectures that combines the ideas of ConvNets and Transformers [48] (Figure 3). Similarly to the (self)-attention layers within transformers, our cloud transform blocks take unordered sets of vectors as inputs, and process such input using multiple parallel *heads*. For an input set element, each head computes two- or three-dimensional *key* and a higher dimensional *value*, and then uses the computed keys to rasterize the respective values onto a regular grid. A two- or three-dimensional convolution is then used to propagate the information across elements. The results of parallel heads are then probed at key locations and are recombined together, producing an update to element features.

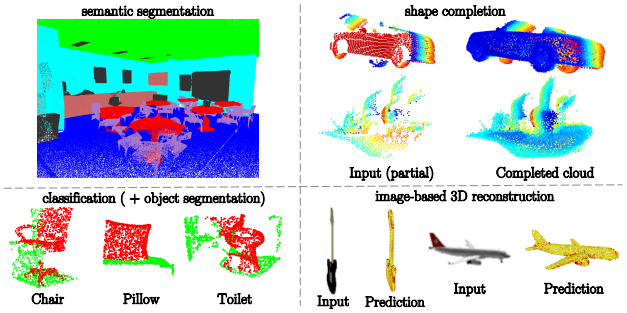

Figure 1: Sample outputs of cloud transformers across four diverse cloud processing tasks including recognition tasks (left) and generation tasks (right).

We show that multiple cloud transform blocks can be stacked sequentially and trained end-to-end, as long as special care is taken when implementing forward and backward pass through the rasterization operations. We then design *cloud transformer* architectures that concatenate multiple cloud transform blocks together with task-specific 3D convolutional layers. Specifically, we design a cloud transformer for semantic segmentation (which we evaluate on the S3DIS benchmark [1]), classification (which we evaluate on the ScanObjectNN benchmark [47]), point cloud inpainting (which we evaluate on ShapeNet-based benchmark [58]), and a cloud transformer for image-based geometric reconstruction (which we evaluate on a recently introduced ShapeNet-based benchmark [42]). In the evaluation, the designed cloud transformers achieve state-of-the-art accuracy for semantic segmentation and point cloud completion tasks and considerably outperform state-of-the-art for image-based reconstruction and point cloud classification (Figure 1). We note that such versatility is rare among previously introduced point cloud processing archi-

*VL is currently with Yandex and Skoltech.

tectures, which can handle either recognition tasks (such as semantic segmentation, classification) or generation tasks (such as inpainting and image-based reconstruction) but usually not both.

To sum up, our key contributions and novelty are:

- We propose a new approach to point cloud processing based on repeated learnable projection, rasterization and de-rasterization operations. We investgate how to make rasterizations and de-rasterizations repeatable *sequentially* within the same architecture through the gradient balancing trick. Additionally, we show that aggregating rasterizations via element-wise maximum performs better than additive accumulation at least in the context of our approach.
- We propose and validate an idea of multi-head self-attention for point clouds that performs *parallel* processing by rasterization and de-rasterization to separate low-dimensional grids. Additionally, we propose an idea of using both two-dimensional and three-dimensional grids in parallel with each other.
- Based on the two ideas above, we propose architectures for semantic segmentation, classification, point cloud inpainting, and image-based reconstruction. The proposed architectures are all based on the same Cloud Transform blocks and achieve state-of-the-art performance on standard benchmarks in each case despite the diversity of tasks.

2. Related work

A number of work use rasterizations of the point cloud over regular 3D grids [26, 8, 27, 25], where each point is rasterized at its original position within the point cloud. Multi-view ConvNets [38] project point clouds to multiple predefined 2D views. The approaches that use splat convolutions on permutohedral grid convolutions [17, 37] are perhaps most similar to ours (and have been an inspiration to us), as they also interleave rasterization (*splatting*), (permutohedral) convolution, and probing (*slicing*). In contrast to all above-mentioned works, which use initial positions or data-independent projections of points for rasterizations, our architectures use diverse data-dependent projections.

A dynamic graph ConvNet (DGCNN) architecture [51] uses a graph ConvNet. The graph is computed from spatial positions of the points that are modified in a data-dependent way within the architecture. In their case, the loss can not be backpropagated through the graph node position estimation since the spatial graph construction is non-differentiable. In contrast, our approach is based on regular grid convolutions and includes the backpropagation through position estimation (key computation). We also note that differentiable point cloud projection onto a 2D grid (from 3D space) has been used in [13] though in a different way and for a different purpose than in our case.

Our approach is also related to *spatial transformers* [15], i.e. neural blocks that warp signals on regular grids through data-dependent parametric warping and bilinear sampling. Our blocks also use bilinear sampling in the end of each head processing. Inspired by spatial transformers, [49] investigate how data-independent and data-dependent deformations of the original point clouds can be used to boost the performance of several recognition architectures including DGCNN [35], SplatNet [37], and VoxelNet [62]. Similarly to [51] and unlike [15], [49] do not propagate the loss fully through deformation computation (in the case of data-dependent deformations). Compared to [49], our architectures employ regular 2D and 3D convolutions, can handle both recognition and generative tasks (the latter not considered in [49]), and are trained with gradient propagation through key position computation. Our work is also related to [24, 40], as our method is also based on rasterizations and de-rasterizations. However, these methods are not applicable for data-dependent transformations of rasterizaton positions and, therefore, cannot handle generative tasks. Additionally, we employ a different method for rasterizations via element-wise maximum, instead of averaging.

Concurrently with us, the work [61] also adapted transformers to the point cloud domain. Their Point Transformer architecture handles the large size of point clouds by restricting self-attention blocks to considering nearest neighbors of individual points, and otherwise follows the original transformer architecture [48] closely. Our work replaces explicit self-attention mechanism with the combination of rasterization and convolution, and thus shares less similarity with [48] and more similarity with works that rely on ConvNets. Point Transformer significantly outperforms state-of-the-art for discriminative tasks, and outperforms our architecture on a semantic segmentation benchmark. At the same time, the adaptation of their approach to generative tasks is not immediately obvious.

3. Method

Overview. We describe our approach in a **bottom-up** way. In Section 3.1, we introduce the basic building operation of our processing pipeline that we call *cloud transform*. In Section 3.2, we discuss how cloud transforms can be assembled in a parallel fashion into blocks (which we call *multi-head processing blocks*). In Section 3.3, we discuss sequential stacking of multi-head processing blocks into bigger blocks (called *cascaded multi-head processing blocks*) that cycle through different spatial resolution and different numbers of feature channels. Finally, in Section 3.4, we introduce the architectures (that we call *cloud transformers*) build from cascaded multi-head processing blocks for four different point cloud processing tasks.

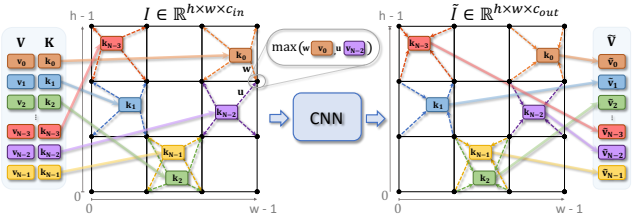

Figure 2: The cloud transform consists of rasterization (left) and de-rasterization(right) steps, with the convolutional part in between. It projects the high-dimensional point cloud onto low-dimensional (two-dimensional in this case) grid, applies convolutional processing, and lifts the result back to the high-dimensional space. Electronic zoom-in recommended.

3.1. Cloud Transform

The cloud transform contains three steps: rasterization, convolution, and de-rasterization.

Rasterization step. To rasterize each point $\boldsymbol{x}_i$, we predict the *value* $\boldsymbol{v}_i \in \mathbb{R}^{c_{in}}$ and the *key* $\boldsymbol{k}_i \in [0,1]^2$. These two vectors stand for *what* to rasterize and *where* to rasterize respectively.

The cloud transform (Figure 2) takes as an input an unordered set (to which we further refer as point cloud) $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \mid \boldsymbol{x}_i \in \mathbb{R}^f\}$, whose elements are vectors $\boldsymbol{x}_i \in \mathbb{R}^f$ of a potentially high dimension f. The Cloud Transform $\mathcal{T}(X)$ maps such input into a new g-dimensional point cloud $Y \in \mathbb{R}^{N \times g}$ of the same size N. Additionally, our layer uses an input point cloud positions $P = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_N \mid \boldsymbol{p}_i \in \mathbb{R}^3\}$.

The cloud transform first applies a learnable projection $\mathcal{P}_2$ (further called *rasterization*), which generates a two-dimensional feature map with c_{in} channels, i.e. $\mathcal{P}_2 \colon X \mapsto I \in \mathbb{R}^{w \times w \times c_{in}}$. In a volumetric setting, the cloud transform starts with a learnable projection $\mathcal{P}_3$, which generates a three-dimensional volumetric feature map, i.e. $\mathcal{P}_3 \colon X \mapsto I \in \mathbb{R}^{w \times w \times w \times c_{in}}$. In both cases, w stands for the spatial resolution of the grid, while c_{in} stands for the number of input channels.

Once an irregular point cloud X is projected onto a regular feature map, the cloud transform applies a single convolution or a more complex combination of convolutional operations. We denote the result of these convolutional layers as $\tilde{I} \in \mathbb{R}^{w \times w \times c_{out}}$ ($\tilde{I} \in \mathbb{R}^{w \times w \times w \times c_{out}}$ in the volumetric case). Note that we expect $\tilde{I}$ to be of the same spatial size as I. However, the channel dimension of $\tilde{I}$ might be changed from c_{in} to c_{out}.

The last step of our Cloud Transform operation is *de-rasterization* (also called *slicing*) $\tilde{\mathcal{P}} \colon \tilde{I} \to \tilde{V}$ from the processed feature map $\tilde{I}$ into a new *transformed* values $\tilde{V} \in \mathbb{R}^{N \times c_{out}}$. Note, that cloud transform passes information from $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ as long as these two points have been projected to sufficiently close positions. Thus, the cloud transform can be seen as a variant of self-attention layer with adaptive sparse attention mechanism. Below, for the sake of simplicity, we detail the steps of the cloud transform for a two-dimensional feature map case. The volumetric case is completely analogous.

Our method allows to directly predict keys via linear layer and stack these layers into deep architectures. However, in our experiments transforming positions via matrix multiplication (i.e predicting it with a point-wise MLP from $\boldsymbol{x}_i$) results in suboptimal performance. A better solution predicts deep residuals $d(\boldsymbol{x}_i)$ to the input positions $\boldsymbol{p}_i$ with a single layer peceptron d and apply a learnable transformation $T \in SE(3)$ afterwards (the transform T becomes the parameter of the layer). The keys are thus computed as:

$$\boldsymbol{k}_i = T\left(\boldsymbol{p}_i + d(\boldsymbol{x}_i)\right) \tag{1}$$

In the case of two-dimensional heads, we project $\boldsymbol{k}_i$ to the $z = 0$ plane (omitting the third coordinate). Finally, we apply the sigmoid activation to the keys $\boldsymbol{k}_i$ ensuring they are positioned between zero and one. As for the values $\boldsymbol{v}_i$ prediction, we use a single affine layer with the output dimension equal to c_{in}, followed by the normalization layer.

Depending on the architecture, the normalization layer can be batch normalization [14], instance normalization [46] or adaptive instance normalization [12].

We then rasterize the value $\boldsymbol{v}_i \in \mathbb{R}^c_{in}$ onto the grid $I = \mathbb{R}^{w \times w \times c_{in}}$ using the predicted key $\boldsymbol{k}_i$ as a position. Specifically, $\boldsymbol{k}_i = (k_i^0, k_i^1) \in [0,1]^2$ may be interpreted as a relative coordinate inside the spatial grid of I. Thus, the position defined by $\boldsymbol{k}_i$ falls into the enclosing integer cell (h_0, w_0), (h_0, w_1), (h_1, w_0), (h_1, w_1), where $h_0 = \lfloor (w-1) \cdot k_i^0 \rfloor$, $h_1 = \lceil (w-1) \cdot k_i^0 \rceil$, $w_0 = \lfloor (w-1) \cdot k_i^1 \rfloor$, $w_1 = \lceil (w-1) \cdot k_i^1 \rceil$. The value $\boldsymbol{v}_i$ is then rasterized into four neighbouring feature map pixels $I[h_0, w_0]$, $I[h_0, w_1]$, $I[h_1, w_0]$, $I[h_1, w_1] \in \mathbb{R}^{c_{in}}$ via bilinear assignment. In more detail, we compute bilinear weights $\boldsymbol{b}_i = (b_i^{00}, b_i^{01}, b_i^{10}, b_i^{11})$ of the key $\boldsymbol{k}_i$ with respect to the cell it falls to. The bilinear weights are then used to update the feature map I at corresponding locations:

$$\begin{aligned} I[h_0, w_0] &\leftarrow \max(I[h_0, w_0], b_i^{00} \boldsymbol{v}_i) \\ I[h_0, w_1] &\leftarrow \max(I[h_0, w_1], b_i^{01} \boldsymbol{v}_i) \\ I[h_1, w_1] &\leftarrow \max(I[h_1, w_0], b_i^{10} \boldsymbol{v}_i) \\ I[h_1, w_1] &\leftarrow \max(I[h_1, w_1], b_i^{11} \boldsymbol{v}_i) \end{aligned} \tag{2}$$

The feature map I is initialized with zeros, and the rasterization is repeated for every $\boldsymbol{x}_i \in X$, $i \in 1..N$, aggregating rasterized results via element-wise maximum at respective cells of the feature map I. While the choice of the maximum aggregator may seem unnatural compared to average or sum, we have found that it boosts the performance substantially.

Convolution step. As discussed above, after rasterization, we transform the feature map I into $\tilde{I}$ with any convolutional architecture that preserves the spatial resolution. In

practice, we use a single convolutional layer that keeps the number of channels unchanged.

De-rasterization step. As the last step, we perform the *de-rasterization* transform $\tilde{\mathcal{P}}_2 \colon \tilde{I} \rightarrow \tilde{V}$ produces the transformed feature cloud Y using standard bilinear grid sampling operation. Thus, the transformed values $\tilde{I}[h_0, w_0]$, $\tilde{I}[h_0, w_1]$, $\tilde{I}[h_1, w_0]$, $\tilde{I}[h_1, w_1] \in \mathbb{R}^{c_{out}}$ of the feature map are combined with bilinear weights $\boldsymbol{b}_i$ into the transformed value vector $\tilde{\boldsymbol{v}}_i$.

We apply the normalization layer, and the ReLU nonlinearity to the result of de-rasterization step, and further map each value from c_{out} dimensions back to g dimensions (g=512 unless noted otherwise) using a learnable affine transform.

Backpropagation through Cloud Transform. We have found that learning architectures with multiple sequentially-stacked Cloud Transform blocks via back-propagation [33] is highly unstable, as the gradients explode during the backward step. An ideal assumption on gradient variance during the back-propagation is to preserve its scale throughout the network [7, 10]. In the supplementary we demonstrate that a naïve verison of Cloud Transform block could not satisfy this assumption and suggest a *gradient balancing* trick to solve this issue. In our case, the instability can be tracked to the gradient of the bilinear weights $\boldsymbol{b}$ w.r.t. the key $\boldsymbol{k}$ at the rasterization and de-rasterization steps. According to the chain rule, the gradients' variance is multiplied by w during backpropagation through the keys, which results in the exponential resulting gradient variance (w.r.t. depth).

Gradient balancing trick Based on observation above, during back-propagation of $\mathcal{L}$ through keys, we simply divide the partial derivatives w.r.t. both coordinates of $\boldsymbol{k}_i$ by w, i.e. we apply:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{k}_i} \leftarrow \frac{1}{w} \frac{\partial \mathcal{L}}{\partial \boldsymbol{k}_i} \,. \tag{3}$$

We have found that this *gradient balancing* trick is sufficient to enable the learning of deep architectures containing multiple layers with cloud transforms.

3.2. Multi-Headed Cloud Transform block

The rasterization and de-rasterization operation may lead to the information loss due to the limited number of nodes in two dimensional and three dimensional lattices (we use up to w=128 for two-dimensional grids and up to w=32 for three-dimensional grids). We therefore build our architectures from blocks that combine multiple cloud transforms operating in parallel. This is reminiscent of both the multiple self-attention head in the Transformer architecture [48] and the multi-view convolutional networks [38]. Following [48], we call each of the parallel cloud transform

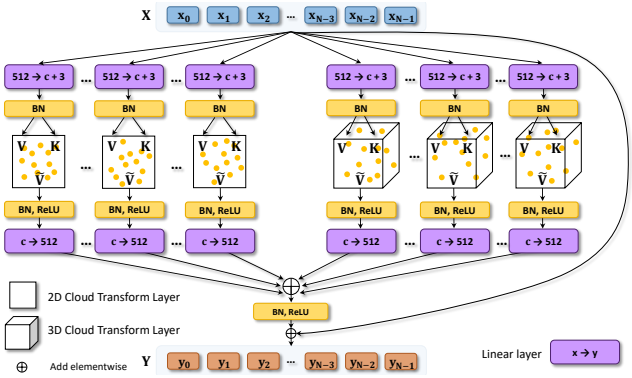

Figure 3: Our building block has several planar heads and several volumetric heads operating in parallel. Each head is a cloud transform, using a two-dimensional or a three-dimensional grid for rasterization, followed by convolutional operations, and de-rasterization (differentiable sampling). Electronic zoom-in recommended.

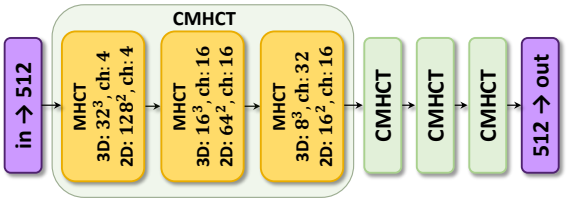

Figure 4: The architecture used for semantic segmentation is based on (1) a single point-wise layer, (2) four standard cascaded multi-headed cloud transform blocks, followed by (3) a final point-wise MLP.

modules a *head* and thus consider a multi-head architecture. Each head predicts keys and values independently, and may use its own spatial resolution w. In fact, two-dimensional and three-dimensional heads can operate in parallel. We note that the use of one-dimensional or higher-dimensional (e.g. four-dimensional) heads is also possible. One-dimensional grids however inevitably results in very strong conflation of the data, while higher-dimensional grids are computationally heavy and convolutions on them are not well supported. We therefore focus on two- and three-dimensional heads.

The results of the parallel heads for each point i are summed together, so that the resulting *multi-head cloud transform (MHCT) block* (Figure 3) still maps each input vector $\boldsymbol{x}_i$ to a g-dimensional vector $\boldsymbol{y}_i$. We add another normalization layer and ReLU nonlinearity after the results of the heads are summed, and complete the block with the residual skip connection from the start to the end.[10]. We note that the multi-head cloud transform block also resembles the Inception block [39], which uses heterogeneous parallel convolutions, as well as the blocks of the ResNeXt networks [54], which use grouped convolutions with small number of channels in each group.

In this work we use MHCT blocks with 16 two-dimensional heads and 16 three-dimensional heads.

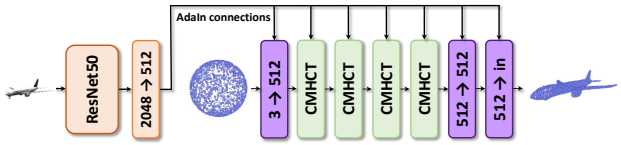

Figure 5: The architecture used for image-based reconstruction consists of a convolutional style encoder (1) and a generator (2). The generator is built of a linear layer, followed by four cascaded multi-headed cloud transform blocks (green), conditioned via adaptive instance normalizations on the style vector produced by the encoder (orange). The input to the generator is sampled from a uniform sphere S^2.

3.3. Cascaded Blocks

Our network does not use pooling and upsampling operations directly on points, as is done in most of the point cloud processing networks (e.g [45], [23]). Instead, we employ feature maps of different spatial sizes as a way of increasing or decreasing the receptive field. Specifically, following the pattern introduced in [36], we propose to stack three MHCT blocks sequentially into *Cascaded Multi-Headed Cloud Transform Block (CMHCT)*, decreasing spatial dimension, while increasing the number of channels. In practice, we set the spatial and channels dimensions as in the Figure 4, yellow blocks.

The order of these three MHCT blocks in CMHCT block is as on the figure. The CMHCT blocks can then be stacked sequentially.

3.4. Cloud Transformers

We now discuss the architectures that can be constructed from CMHCT blocks for specific point cloud processing tasks. We note that while the different nature of tasks requires different architectures, we strove to keep these architectures as similar as possible. Most importantly, all proposed architectures are build from CMHCT blocks that are built out of MHCT blocks that are based on Cloud Transforms.

Semantic segmentation. The semantic segmentation cloud transformer (Figure 4) consists of an initial one-layer perceptron, which is applied to each point independently and transforms its 3D coordinates and 3D color features to an f-dimensional vector (f=512). Afterwards, we apply four cascaded multi-headed cloud transform layers with default setting. And then conclude the architecture with a two-layer shared perceptron that maps the features of each point to the logits of segmentation classes. All normalization layers in the architectures are BatchNorm layers [14]. The architecture has 9.6M parameters and is trained with the cross-entropy loss.

Point cloud generation. To create the architecture that generates point cloud, we stack four CMHCT blocks sequentially, followed by a point-wise two-layer perceptron.

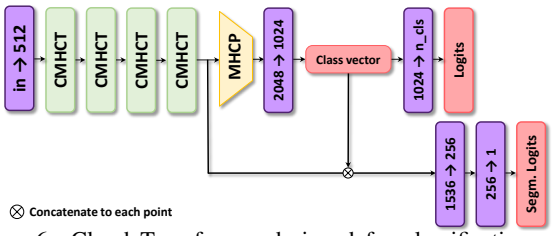

Figure 6: Cloud Transformer designed for classification. Primarily, the input point cloud is processed with a Cloud Transformer backbone, as in the segmentation setting. Afterward, we apply a Multi-headed Cloud Pooling (yellow), followed by a fully-connected layer to produce a classification vector (red). We also use a separate branch (bottom) for the background mask prediction as is common for the ScanObjectNN benchmark.

Finally, we add a $tanh$ non-linearity to produce 3D points coordinates. The input point cloud is sampled from a uniform 3D distribution on the unit sphere S^2 and then passed through point-wise linear layer, mapping each feature to f=512 dimensions. To solve the image-based geometry reconstruction task (recovering point clouds from images), we use adaptive instance normalization (AdaIN) layers [12] in the MHCT blocks. We create image encoder with ResNet-50 architecture [10] (pretrained on ImageNet [34]). The output of the encoder is a 512-dimensional vector, which is transformed into AdaIN coefficients via affine layer (Figure 5). The architecture is trained with approximate earth mover distance (EMD) loss [22]. Note, our generator architecture highly resembles our segmenter, apart from the normalization method.

Point cloud classification As in the segmentation model, we apply a "backbone" of a leading linear layer and four CMHCT blocks first (Figure 2). To solve a classification task, we introduce a multi-headed pooing layer. Similarly to the regular MHCT layer, this layer performs multiple rasterizations onto 2D and 3D feature maps of spatial size 8 and 16 respectively. The channel dimension of each head is 32 for three-dimensional heads and 16 for two-dimensional heads. Afterward, the resulting feature maps are processed with three standard convolutional residual blocks, each interchanged with a max-pooling (see the supplementary material for the exact architecture). The resulting vectors are aggregated across the heads via concatenation and processed with a dense layer to form a final classification vector $\boldsymbol{k}_{class}$ of dimension 1024. Following the original paper [47], we also predict an object's instance mask, using the point-wise features predicted with from the features extracted by the backbone and the class vector $\boldsymbol{k}_{class}$.

We train our architecture as in [47] with the two cross-entropy (CE) loss terms. The first one is a CE loss λ_{class} on object classification and the latter one is a point-wise CE loss λ_{seg} on foreground segmentation. The final loss is set to be $\lambda_{\text{full}} := 0.5 \cdot \lambda_{\text{class}} + 0.5 \cdot \lambda_{\text{seg}}$.

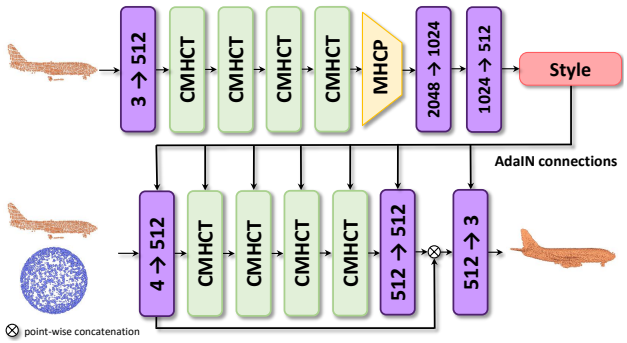

Figure 7: Our Cloud Transformer Inpainter differs from our single-view reconstruction model in two ways. First, the encoder part is taken from the classifier model, producing a style vector for adaptive normalization layers of the generator. Secondly, the input to the generator consists of the partial point cloud as wekk as of the points sampled from the unit sphere S^2 for "unknown" points.

Point cloud inpainting We also present a model for point cloud inpainting (completion). Given a partial point cloud, the goal is to infer the full shape. Our architecture for this task strongly resembles the one used for image-based reconstruction, apart from the encoder and the generator input. Namely, we apply a point cloud encoder to obtain a style vector. Our encoder is the Cloud Transformer model introduced above for classification. It extracts a vector of dimension 512.

To account for the input point cloud geometry, we use two sets of points in the generator branch input. First, we put in the input point cloud, thus giving the architecture opportunity to transfer its points to the output. Second, to account for the novel parts, we again sample random points from a unit sphere S^2. We augment the feature of each input point with a binary variable indicating whether is is sampled from the incomplete scan or from the sphere. As in the case of the image-based reconstruction, the generator branch is conditioned on the encoded vector via AdaIN connections. The model is trained with a sum of approximate EMD [22] and Chamfer Distance [6] losses. Afterwards, we fine-tine it on Chamfer Distance loss.

4. Experiments

We compare the performance of our approach to the state-of-the-art on the four considered tasks, using established benchmarks and metrics associated with those benchmarks. We then perform a short ablation study. In the **supplementary materials**, we provide visualizations of the cloud transformer operations and results, and the readers are encouraged to check the material to gain better intuition about cloud transformers.

Point Cloud Classification. We evaluate our Cloud Transformer for classification on a real-world dataset ScanObjectNN [47]. The dataset consists of 2902 unique objects obtained from ScanNet [5] scenes. The objects are categorized into 15 classes. Each object is obtained via cutting it from the scene using the ground truth bounding box. Note that the resulting cut may include background points as well. In the variant we use, each bounding box is randomly shifted and rotated to emulate real-world detection boxes. This process is repeated five times with different bounding box perturbations for each object. This process results in ~15k objects in total. An object is represented as a cloud of 2048 3D spatial points. Additionally, a binary pointwise mask is provided for training, indicating whether a point belongs to the background or to the object. We use the hardest variant of the dataset (PB_T50_RS), with the highest rate of bounding box perturbations. We adopt the original [47] train/test split.

	overall acc.	mean class acc.
3DmFV [2]	63	58.1
PointNet [28]	68.2	63.4
SpiderCNN [55]	73.7	69.8
PointNet++ [29]	77.9	75.4
DGCNN [51]	78.1	73.6
PointCNN [21]	78.5	75.1
DRNet [30]	80.3	-
GFNet [31]	80.5	77.8
DI-PointCNN [59]	81.3	79.6
CT (ours)	85.0	80.7
CT (ours) + scales	**85.5**	**83.1**

Table 1: Classification results on ScanObjectNN. Our method outperforms the others both in terms of overall accuracy and mean class accuracy.

We provide results in Table 1 and show that our method outperforms the existing ones both in overall and mean-class accuracy. Note, our method outperforms the current state-of-the-art in terms of the overall accuracy by a large margin (**+3.7%**). We also evaluate a CT variant with a learnable anisotropic scaling s applied after the transformation T in the equation 1. The resulting model outperforms the state-of-the-art by **3.5**% in the mean class accuracy.

For completeness, we have also evaluated our approach on the ModelNet40 benchmark [52], which has been saturated. Following the PointNet++ protocol [29], our approach (CT+scales) achieves 93.1 in Overall Accuracy (OA), and 90.8 in mean Class Accuracy (mAcc), which is on par or better than other methods that work with point clouds (rather than CAD meshes).

Single-View Object Reconstruction. In our generation experiments we follow the recently introduced benchmark [42] on 3D object reconstruction. The benchmark is based on ShapeNet [3] renderings. Unlike previous ShapeNet-based benchmarks for image-based reconstruction that used *canonical coordinate frames*, the new one argues that the re-

Methods	mAvg. F-score@1%	Top-1 cat.
AtlasNet [9]	0.252	2
Matryoshka [32]	0.217	0
OGN [41]	0.264	1
Retrieval	0.236	0
Retrieval (oracle)	0.290	3
CT (ours)	**0.359**	**49**

Table 2: F-score evaluation (@1%) of 3D shape reconstruction in the viewer-based coordinate frame, averaged by categories. The cloud transformer outperforms other methods including the retrieval based oracle considerably.

constructions should be evaluated in the *viewer-based coordinate frame*, where the task is more challenging and more realistic. The work [42] also provides evaluations of several recent methods on image-based reconstruction, as well as the retrieval-based oracle. The dataset consists of ShapeNet [3] models, where each model belongs to one of 55 classes. Each object has been rendered with ShapeNet-Viewer from five random view points. We employ the same train/val/test split as in [42].

In the benchmark, objects were rendered to 224×224 pixel images, which we resize to 128×128 pixels and then fed to our model. The ground truth is represented as a cloud of 10.000 points in the viewer-aligned coordinate system.

Our model outputs 8196 points to represent a reconstructed object. Since the protocol requires to predict exactly 10.000 points, we perform the reconstruction twice with different sphere noise and the same style vector $\boldsymbol{z}$ extracted by the convolutional encoder (Figure 5). This results in 16.392 points total from which we randomly select 10.000 points.

The main evaluation metric proposed in [42] is the F-score computed at a 1% volume distance threshold. The methods are compared with macro-averaged F-score @1% and by the number of classes, in which a method has the highest mean F-score @1%. Our quantitative results are summarized in Table 2. In is evident that our method outperforms all methods evaluated in [42] *including* the retrieval-based oracle very significantly.

Indoor Semantic Segmentation. The Stanford Indoor Dataset (S3DIS) [1] is a popular 3D point cloud segmentation benchmark that consists of large 3D point cloud scenes captured at three different buildings annotated with 13 semantic labels at the point level. The dataset comes with six splits. For the sake of fair comparison, we evaluate on S3DIS using a conventional protocol, established by [28], which chunks rooms into 1m×1m blocks. Each block consists of 4096 points and each point is represented with its 3D coordinates and its RGB color, which results in a six-dimensional input vector. In this setting, the average inference time is 0.5 sec on a Tesla P40 GPU card, with only 200 MB memory per chunk used. Following many previous works, we evaluate on the 'Area 5' split and train on the remaining five splits, as [43] advocate this fold as representative in measuring generalization ability due to being shot in a separate building.

Method	mIoU	Method	mIoU
PointNet [28]	41.1	SPGraph* [19]	58.0
Eff. 3D Conv [60]	51.8	Minkowski32* [4]	65.3
RNN Fusion [57]	57.3	KPConv† [45]	67.1
ParamConv [50]	58.3	JSENet† [11]	67.7
PointCNN [21]	57.2	Point Trans.* [61]	**70.4**
CT (ours)	63.7	**CT† (ours)**	67.9

Table 3: Semantic segmentation intersection-over-union scores on S3DIS *Area-5* split. The methods in the left column use the standard protocol with chunking of the scene into blocks, methods marked with † employ KPConv's [45] protocol. Other protocols are labeled with $*$. Cloud transformers outperform state-of-the-art in standard protocol and perform better than previously published works in other protocols (Note that Point Transformer is a concurrent line of work).

Since the current published state-of-the-art-method JSENet [11], uses a different protocol [45]), we also evaluate our model using 'KPConv' protocol. In it, at each step an input point cloud is dynamically sampled from a sphere of radius $2m$. Each point cloud contains up to 8192 points. During evaluation, the same data strategy is applied together with voting.

In the standard 1m $\times$ 1m protocol xyz spatial coordinates are augmented with random rotation, anisotropic scale, jitter and shifts. For color, on thhe other hand, we use chromatic autocontrast, jitter and translation (following [4]). As for the 'KPConv' protocol, we follow the original paper and augment point clouds with anisotropic random scaling, spatial gaussian jittering, random rotations around the z-axis and random color dropping. Table 3 shows that for the semantic segmentation task our method outperforms state-of-the-art in both considered protocols.

Point Cloud Completion. Finally, we evaluate Cloud Transformer Inpainter on the ShapeNet-based benchmark [58] for high-resolution point cloud completion. The benchmark is composed of the eight largest ShapeNet categories (airplane, cabinet, car, chair, lamp, sofa, table, vessel). This makes 30974 unique objects in total. In each category 100 of unique objects is reserved for validation and 150 for testing. The dataset consists of $(P_{\text{part}}, P_{\text{gt}})$ pairs, where P_{part} is a *partial* point cloud and P_{gt} is a *complete* point cloud. Partial point clouds are obtained via 2.5D depth image back-projection, taken from a random view. There are eight random views generated per each training object. The partial cloud consists of no more than 2048 points, while the complete point cloud consists of 16.384 points. In contrast to the image-based reconstruction, both partial and complete point clouds are provided in the object-based

Methods	mAvg. F-Score@1%	mAvg. CD
AtlasNet [9]	0.616	4.523
PCN [58]	0.695	4.016
FoldingNet [56]	0.322	7.142
TopNet [44]	0.503	5.154
MSN [22]	0.705	4.758
GRNet [53]	0.708	**2.723**
CT (ours)	**0.752**	3.392

Table 4: Point completion results on ShapeNet compared using F-Score@1% and CD. Note that both of them are computed on 16,384 points and macro-averaged.

coordinate system.

We predict a high-resolution reconstruction with 16.384 points in a single pass of our Cloud Transformer Inpainter network. Our model produces detailed reconstructions with complex geometries (see Figure 1). Regarding quantitative evaluation of our method, we report both F-Score@1% and CD (Chamfer Distance), both of them computed with the ground truth 16.384 point clouds. Following [42], we argue that the F-Score@1% should be regarded as a primal metric of the shape prediction quality, and in this metric our method again beats state-of-the-art considerably (Table 4).

Ablation Study. We also perform an ablation study to justify our architecture choices. We consider the following ablations:

- **Linear key prediction**: We replace key prediction procedure with a linear point-wise layer d, followed by BatchNormalization. Using the notations form 3, $\boldsymbol{k}_i = d(\boldsymbol{x}_i)$.
- **Mean agg.** and **Sum agg.**: The aggregation method in the rasterization step is replaced with element-wise mean and sum correspondingly. Note, in the latter case (sum) it makes our operation similar to the splatting, used in SplatNet [37].
- **Non-learnable keys:** In this ablation we use different non-learnable projections of the input positions as keys. More precisely, $\boldsymbol{k}_i = T(\boldsymbol{p}_i)$, where $T \in SO(3)$ is a *fixed* random transformation and no deep residual predicted. While this variant performs on par with *linear key prediction* for segmentation, it performs marginally better on classification.
- We also train an architecture **without planar heads** to see if using only volumetric heads might be sufficient.
- In the **Coarser feature maps** experiment the spatial dimensions of feature maps are halved.
- **No multihead**: We ablate our multi-headed architecture by replacing 16 headed CMHCT blocks with a single-headed block, where we increase the channel dimension 8 times to keep the model's capacity intact.
- Finally, we consider **2x shallower** and architectures, replacing four CMHCT blocks with two CMHCT blocks.

Methods	mIOU S3DIS	acc. ScanObjNN
Linear key prediction	62.5	81.9
Sum aggregation	57.9	82.1
Mean aggregation	61.3	83.4
Without planar heads	63.4	84.8
Coarser feature maps	62.2	84.4
No multihead	63.1	84.0
Non-learnable keys	62.5	84.9
2x shallower	62.2	84.1
CT (full)	63.7	85.0

Table 5: Ablation study on S3DIS semantic segmentation and ScanObjectNN classification. See text for discussion.

Observing the advantage of the full architecture over the shallower architecture in the S3DIS case, we have also evaluated a 2x deeper architecture with eight CMHCT blocks, achieving 64.1 mIOU score. With (T = Id) (see 2) ablation we observed 61.9 mIOU thus learnable projections are of great importance. Our preliminary ablation with $d = 0$ suggests that the effect is small for segmentation (drop of 0.04%), but we expect it to be higher for generative tasks where the input point cloud is trivial (spherical).

We have also evaluated the importance of the **gradient balancing trick**. On the S3DIS an ablation without gradient balancing trick achieved 62.8 mIOU. More importantly, when we tried to run the linear key prediction variant without the gradient balancing, the learning diverged for all reasonable learning rates, revealing the importance of gradient balancing.

5. Conclusion

We have presented a new block for neural architectures that process point clouds. Our block extends the ideas of Spatial Transformers, Transformers, and Multi-View CNNs on neural point cloud processing.

While there are some significant differences between our architecture and the Transformer, we want to highlight some interesting similarities. Both Transformer and our architecture operate on sets and use parallel heads. Most importantly, similarly to Transformer, our architecture achieves quick and long range information propagation without blowing up the number of learnable parameters. In the semantic segmentation 1×1 chunk case, an average point "interacts" with 39% of other points after the first MHCT block and with 100% of points after just one CMHCT block (i.e. just three MHCT blocks).

Based on the new block, we have presented architectures for point cloud semantic segmentation, point cloud classification, point cloud completion and single-image based geometry reconstruction that achieve state-of-the-art results.

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proc. CVPR*, 2016.

[2] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3(4):3145–3152, 2018.

[3] 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*, abs/1512.03012, 2015.

[4] Christopher Bongsoo Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proc. CVPR*, 2019.

[5] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. CVPR*, 2017.

[6] Haoqiang Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proc. CVPR*, 2017.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[8] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proc. CVPR*, 2018.

[9] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proc. CVPR*, 2018.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. ICCV*, 2015.

[11] Zeyu Hu, Mingmin Zhen, Xuyang Bai, Hongbo Fu, and Chiew-lan Tai. Jsenet: Joint semantic segmentation and edge detection network for 3d point clouds. In *Proc. ECCV*, 2020.

[12] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proc. ICCV*, 2017.

[13] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Proc. NeurIPS*, 2018.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In *Proc. NIPS*, 2015.

[16] Varun Jampani, Martin Kiefel, and Peter V. Gehler. Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks. In *Proc. CVPR*, 2016.

[17] Martin Kiefel, Varun Jampani, and Peter V. Gehler. Permutohedral lattice cnns. In *ICLR Workshop Track*, May 2015.

[18] Roman Klokov and Victor S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proc. ICCV*, 2017.

[19] Loïc Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *Proc. CVPR*, 2018.

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, Dec. 1989.

[21] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Proc. NeurIPS*, 2018.

[22] Minghua Liu, Lu Sheng, Shilin Yang, Jing Shao, and Shi-Min Hu. Morphing and sampling network for dense point cloud completion. In *Proc. AAAI*, 2020.

[23] Ze Liu, Han Hu, Yue Cao, Zheng Zhang, and Xin Tong. A closer look at local aggregation operators in point cloud analysis. In *Proc. ECCV*, 2020.

[24] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. In *Proc. NeurIPS*, 2019.

[25] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *Proc. ICCV*, 2019.

[26] Daniel Maturana and Sebastian A. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Proc. IROS*, pages 922–928. IEEE, 2015.

[27] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In *Proc. CVPR*, 2018.

[28] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. CVPR*, 2017.

[29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. NIPS*, 2017.

[30] Shi Qiu, Saeed Anwar, and Nick Barnes. Dense-resolution network for point cloud classification and segmentation. In *Proc. WACV*, 2021.

[31] Shi Qiu, Saeed Anwar, and Nick Barnes. Geometric back-projection network for point cloud classification. *IEEE Transactions on Multimedia*, 2021.

[32] Stephan R. Richter and Stefan Roth. Matryoshka networks: Predicting 3d geometry via nested shape layers. In *Proc. CVPR*, 2018.

[33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[34] 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)*, pages 211–252, 2015.

[35] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2015.

[37] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proc. CVPR*, 2018.

[38] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

[39] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015.

[40] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, J. Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *Proc. ECCV*, 2020.

[41] Maxim Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proc. ICCV*, 2017.

[42] Maxim Tatarchenko*, Stephan R. Richter*, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? In *Proc. CVPR*, 2019.

[43] Lyne P. Tchapmi, Christopher Bongsoo Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *Proc. 3DV*, 2017.

[44] Lyne P. Tchapmi, Vineet Kosaraju, Hamid Rezatofighi, Ian Reid, and Silvio Savarese. Topnet: Structural point cloud decoder. In *Proc. CVPR*, 2019.

[45] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proc. ICCV*, 2019.

[46] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proc. CVPR*, 2017.

[47] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proc. ICCV*, 2019.

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. NIPS*, 2017.

[49] Jiayun Wang, Rudrasis Chakraborty, and Stella X. Yu. Spatial transformer for 3d points. *ArXiv*, abs/1906.10887, 2019.

[50] S. Wang, S. Suo, W. Ma, A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *Proc. CVPR*, 2018.

[51] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[52] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proc. CVPR*, 2015.

[53] Haozhe Xie, Hongxun Yao, Shangchen Zhou, Jiageng Mao, Shengping Zhang, and Wenxiu Sun. Grnet: Gridding residual network for dense point cloud completion. In *Proc. ECCV*, 2020.

[54] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proc. CVPR*, 2016.

[55] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proc. ECCV*, 2018.

[56] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proc. CVPR*, 2018.

[57] Xiaoqing Ye, Jiamao Li, Hexiao Huang, Liang Du, and Xiaolin Zhang. 3d recurrent neural networks with context fusion for point cloud semantic segmentation. In *Proc. ECCV*, 2018.

[58] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. Pcn: Point completion network. In *Proc. 3DV*, 2018.

[59] R. Zhai, X. Li, Z. Wang, S. Guo, S. Hou, Y. Hou, F. Gao, and J. Song. Point cloud classification model based on a dual-input deep network framework. *IEEE Access*, 2020.

[60] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds. In *Proc. 3DV*, 2018.

[61] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Koltun Vladlen. Point transformer. In *Proc. ICCV*, 2021.

[62] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proc. CVPR*, 2018.