PAConv: Position Adaptive Convolution with Dynamic Kernel Assembling on Point Clouds

Mutian Xu[1]* Runyu Ding[1]* Hengshuang Zhao[2] Xiaojuan Qi[1]†
[1]The University of Hong Kong [2]University of Oxford
mino1018@outlook.com, {ryding, xjqi}@eee.hku.hk, hengshuang.zhao@eng.ox.ac.uk

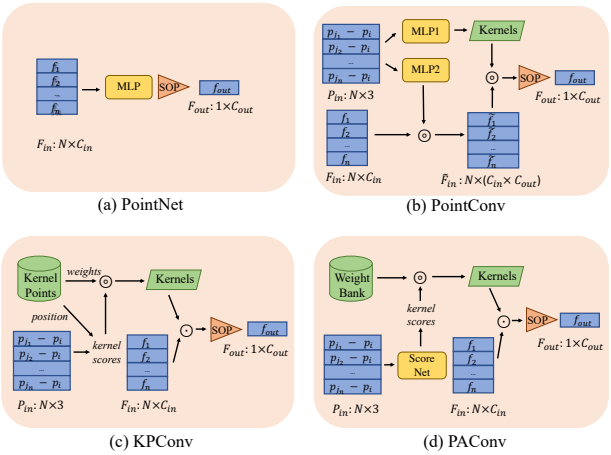
Abstract

*We introduce **P**osition **A**daptive Convolution (PAConv), a generic convolution operation for 3D point cloud processing. The key of PAConv is to construct the convolution kernel by dynamically assembling basic weight matrices stored in Weight Bank, where the coefficients of these weight matrices are self-adaptively learned from point positions through ScoreNet. In this way, the kernel is built in a data-driven manner, endowing PAConv with more flexibility than 2D convolutions to better handle the irregular and unordered point cloud data. Besides, the complexity of the learning process is reduced by combining weight matrices instead of brutally predicting kernels from point positions.*

*Furthermore, different from the existing point convolution operators whose network architectures are often heavily engineered, we integrate our PAConv into classical MLP-based point cloud pipelines **without** changing network configurations. Even built on simple networks, our method still approaches or even surpasses the state-of-the-art models, and significantly improves baseline performance on both classification and segmentation tasks, yet with decent efficiency. Thorough ablation studies and visualizations are provided to understand PAConv. Code is released on https://github.com/CVMI-Lab/PAConv.*

1. Introduction

In recent years, the rise of 3D scanning technologies has been promoting numerous applications that rely on 3D point cloud data, *e.g.*, autonomous driving, robotic manipulation and virtual reality [35, 40]. Thus, the approaches to effectively and efficiently processing 3D point clouds are in critical needs. While remarkable advancements have been obtained in 3D point cloud processing with deep learning [36, 37, 47, 25], it is yet a challenging task in view of the sparse, irregular and unordered structure of point clouds.



(a) PointNet (b) PointConv (c) KPConv (d) PAConv

Figure 1. Overview about convolutional sturctures of PointNet [36], PointConv [52], KPConv [47] and our PAConv. It illustrates the differences of these point-based convolutions. SOP denotes symmetric operations, like MAX.

To tackle these difficulties, previous research can be coarsely cast into two categories. The first line attempts to voxelize the 3D point clouds to form regular grids such that 3D grid convolutions can be adopted [33, 43, 39]. However, important geometric information might be lost due to quantization, and voxels typically bring extra memory and computational costs [10, 7].

Another stream is to directly process point cloud data. The pioneering work [36] proposes to learn the spatial encodings of points by combing Multi-Layer Perceptron (MLP) [13] and global aggregation as illustrated in Fig. 1 (a). Follow-up works [37, 38, 48, 20, 51] exploit local aggregation schemes to improve the network. Nonetheless, all the points are processed by the same MLP, which limits the capabilities in representing spatial-variant relationships.

Beyond MLP, most recent works design convolution-like operations on point clouds to exploit spatial correlations. To handle the irregularity of 3D point clouds, some works [58, 50, 29] propose to directly predict the kernel weights based on relative location information, which is

*M. Xu and R. Ding contribute equally.
†Corresponding author

further used to transform features just like 2D convolutions. One representative architecture [52] in this line of research is shown in Fig. 1 (b). Albeit conceptually effective, the methods severely suffer from heavy computation and memory costs caused by spatial-variant kernel prediction in practice. The efficient implementation also trade-offs its design flexibility, leading to inferior performance. Another group of works relate kernel weights with fixed kernel points [2, 47, 32] and use a correlation (or interpolation) function to adjust the weight of kernels when they are applied to process point clouds. Fig. 1 (c) illustrates one representative architecture [47]. However, the hand-crafted combination of kernels may not be optimal and sufficient to model the complicated 3D location variations.

In this paper, we present **P**osition **A**daptive **C**onvolution, namely PAConv, which is a plug-and-play convolutional operation for deep representation learning on 3D point clouds. PAConv (shown in Fig. 1 (d)) constructs its convolutional kernels by dynamically assembling basic weight matrices in Weight Bank. The assembling coefficients are self-adaptively learned from relative point positions by MLPs (*i.e*. ScoreNet). Our PAConv is flexible to model the complicated spatial variations and geometric structures of 3D point clouds while being efficient. Specifically, instead of inferring kernels from point positions [52] in a brute-force way, PAConv bypasses the huge memory and computational burden via a dynamic kernel assembling strategy with ScoreNet. Besides, unlike kernel point methods [47], our PAConv gains flexibility to model spatial variations in a data-driven manner and is much simpler without requiring sophisticated designs for kernel points.

We conduct extensive experiments on three challenging benchmarks on top of three generic network backbones. Specifically, we adopt the simple MLP-based point networks PointNet [36], PointNet++ [37] and DGCNN [51] as the backbones, and replace their MLPs with PAConv without changing other network configurations. With these simple backbones, our method still achieves the state-of-the-art performance on ModelNet40 [53] and considerably improves the baseline by 2.3% on ShapeNet Part [61] and 9.31% on S3DIS [1] with decent model efficiency. It's also worth noting that recent point convolution methods often use complicated architectures and data augmentations tailored to their operators [47, 25, 30] for evaluation, making it difficult to measure the progress made by the convolutional operator. Here, we adopt simple baselines and aim to minimize the influence of network architectures to better assess the performance gain from the operator – PAConv.

2. Related Work

Mapping point clouds into regular 2D or 3D grids (voxels). Since point cloud data has irregular structure in 3D space, early works [44, 21, 6] project point clouds to multi-view images and then utilize conventional convolutions for feature learning. Yet, this 3D-to-2D projection is not robust to occluded surfaces or density variations. Tatarchenko *et al*. [45] propose to map local surface points onto a tangent plane and further uses 2D convolutional operators, and FPConv [25] flattens local patches onto regular 2D grids with soft weights. However, they heavily rely on the estimation of tangent planes, and the projection process will inevitably sacrifice the 3D geometry information. Another technique is to quantize the 3D space and map points into regular voxels [39, 33, 3, 34], where 3D convolutions can be applied. However, the quantization will inevitably lose fine-grained geometric details, and the voxel representation is limited by the heavy computation and memory cost. Recently, to address the above issues, sparse representations [43, 10, 7] are employed to obtain smaller grids with better performance. Nevertheless, they still suffer from the trade-off between the quantization rate and the computational efficiency.

Point representation learning with MLPs. Many methods [36, 37, 18, 28, 14] process unstructured point clouds directly with point-wise MLPs. PointNet [36] is the pioneering work which encodes each point individually with shared MLPs and aggregates all point features with global pooling. However, it lacks the ability to capture local 3D structures. Several follow-up works address this issue by adopting hierarchical multi-scale or weighted feature aggregation schemes to incorporate local features [37, 19, 23, 16, 18, 28, 55, 17, 14, 60, 54, 56]. Other approaches use graphs to represent point clouds [38, 42, 51, 49, 57], and the point features are aggregated through local graph operations, aiming to capture local point relationships. Nonetheless, they all adopt the shared MLPs to transform point features, which limits the model capabilities in capturing spatial-variant information.

Point representation learning with point convolutions. More recently, lots of attempts [24, 58, 50, 52, 29, 47, 32, 30] focus on designing point convolutional kernels. PointCNN [24] learns an $\mathcal{X}$-transformation to relate points with kernels. However, this operation cannot satisfy permutation invariant, which is crucial for modeling un-ordered point cloud data. In addition, [41, 11, 58, 50, 52, 29] propose to directly learn the kernel of local points based on point positions. Nevertheless, these methods directly predict kernels, which has much higher complexity (memory and computation) in the learning process.

Another type of point convolutions associate weight matrices with pre-defined kernel points in 3D space [2, 5, 47, 32, 26, 22]. However, the positions of kernels have crucial influence on the final performance [47] and need to be specifically optimized for different datasets or backbone architectures. Besides, the above approaches [47, 32, 22] generate kernels through combining pre-defined kernels using hand-crafted rules which limit the model flexibility, lead-

ing to inferior performance [22]. Different from them, our method adaptively combines weight matrices in a learn-able manner, which improves the capability of the operator to fit irregular point cloud data.

Dynamic and conditioned convolutions. Our work is also related to dynamic and conditional convolutions [8, 9, 59]. Brabandere *et al*. [8] propose to dynamically generate position-specific filters on pixel inputs. In [9], through learning the offsets on kernel coordinates, the original kernel space is deformed to adapt to different scales of objects. Further, CondConv [59] generates the convolution kernel by combining several filters through a routing function that outputs the coefficients for filter combination, which is similar with our dynamic kernel assembly. Yet, the predicted kernels in CondConv [59] are not position-adaptive, while the unstructured point clouds require the weights that adapt to different point locations.

3. Method

In this section, we first revisit the general formulation of point convolutions. Then we introduce PAConv with dynamic kernel assembly. Finally, we compare PAConv with prior relevant works to demonstrate our advantages.

3.1. Overview

Given N points in a point cloud $\mathcal{P} = \{p_i | i = 1, ..., N\} \in \mathbb{R}^{N \times 3}$, the input and output feature map of $\mathcal{P}$ in a convolutional layer can be denoted as $F = \{f_i | i = 1, ..., N\} \in \mathbb{R}^{N \times C_{in}}$ and $G = \{g_i | i = 1, ..., N\} \in \mathbb{R}^{N \times C_{out}}$ respectively, where C_{in} and C_{out} are the channel numbers of the input and output. For each point p_i, the generalized point convolution can be formulated as:

$$g_i = \Lambda(\{\mathcal{K}(p_i, p_j) f_j | p_j \in \mathcal{N}_i\}), \tag{1}$$

where $\mathcal{K}(p_i, p_j)$ is a function which outputs convolutional weights according to the position relation between the center point p_i and its neighboring point p_j. $\mathcal{N}_i$ denotes all the neighborhood points, and Λ refers to the aggregation function in terms of MAX, SUM or AVG. Under this definition, 2D convolution can be regarded as a special case of the point convolution. For instance, for a 3×3 2D convolution, the neighborhood $\mathcal{N}_i$ lies in a 3×3 rectangular patch centered on pixel i , and $\mathcal{K}$ is a one-to-one mapping from a relative position (p_i, p_j) to the corresponding weight matrix $\mathcal{K}(p_i, p_j) \in \mathbb{R}^{C_{in} \times C_{out}}$ in a fixed set of 3×3 (Fig. 2. (a)).

However, the simple one-to-one mapping kernel function defined on images is not applicable for 3D point clouds owing to the irregular and unordered characteristics of point clouds. Specifically, the spatial positions of 3D points are continuous and thus the number of possible relative offsets (p_i, p_j) is infinite, which cannot be mapped into a finite-sized set of kernel weights. Therefore, we redesign the kernel function $\mathcal{K}$ to learn a position-adaptive mapping by dynamic kernel assembly. First, we define a Weight Bank composed of several weight matrices. Then, a ScoreNet is designed to learn a vector of coefficients to combine the weight matrices according to point positions. Finally, the dynamic kernels are generated by combining the weight matrices and its associated position-adaptive coefficients. The details are shown in Fig. 2. (b) and elaborated below.

3.2. Dynamic Kernel Assembling

Weight Bank. We first define a Weight Bank $\mathcal{B} = \{B_m | m = 1, ..., M\}$, where each $B_m \in \mathbb{R}^{C_{in} \times C_{out}}$ is a weight matrix, and M controls the number of weight matrices stored in the Weight Bank $\mathcal{B}$.

Intuitively, larger M contributes to more diversified weight matrices for kernel assembly. Yet, too many weight matrices may bring redundancies and cause heavy memory/computation overheads. We find that setting M to 8 or 16 is appropriate, which is discussed in Sec. 6.2. Equipped with Weight Bank, the next is to establish a mapping from discrete kernels to continuous 3D space. To this end, we propose ScoreNet to learn coefficients to combine weight matrices and produce dynamic kernels fitting to point cloud inputs, which is detailed as follows.

ScoreNet. The goal of ScoreNet is to associate relative positions with different weight matrices in Weight Bank $\mathcal{B}$. Given the specific position relation between a center point p_i and its neighbor point p_j, ScoreNet predicts the position-adaptive coefficients S_{ij}^m for each weight matrix B_m.

The inputs of ScoreNet are based on position relations. We explore different input representations as illustrated in Sec. 6.1. For the sake of clarity, here we denote this input vector as $(p_i, p_j) \in \mathbb{R}^{D_{in}}$. The ScoreNet outputs a normalized score vector as:

$$\mathcal{S}_{ij} = \alpha(\theta(p_i, p_j)), \tag{2}$$

where θ is a non-linear function implemented using Multi-layer Perceptrons (MLPs) [13] and α indicates Softmax normalization. The output vector $\mathcal{S}_{ij} = \{S_{ij}^m | m = 1, ..., M\}$, where S_{ij}^m represents the coefficient of B_m in constructing the kernel $\mathcal{K}(p_i, p_j)$. M is the number of weight matrices. Softmax ensures that the output scores are in range $(0, 1)$. This normalization guarantees that each weight matrix will be chosen with a probability, with higher scores implying stronger relations between the position input and the weight matrix. Sec. 6.1 presents the comparison of different normalization schemes.

Kernel generation. The kernel of PAConv is derived by softly combining weight matrices in Weight Bank $\mathcal{B}$ with the corresponding coefficients predicted from ScoreNet:

$$\mathcal{K}(p_i, p_j) = \sum_{m=1}^{M} (S_{ij}^m B_m). \tag{3}$$

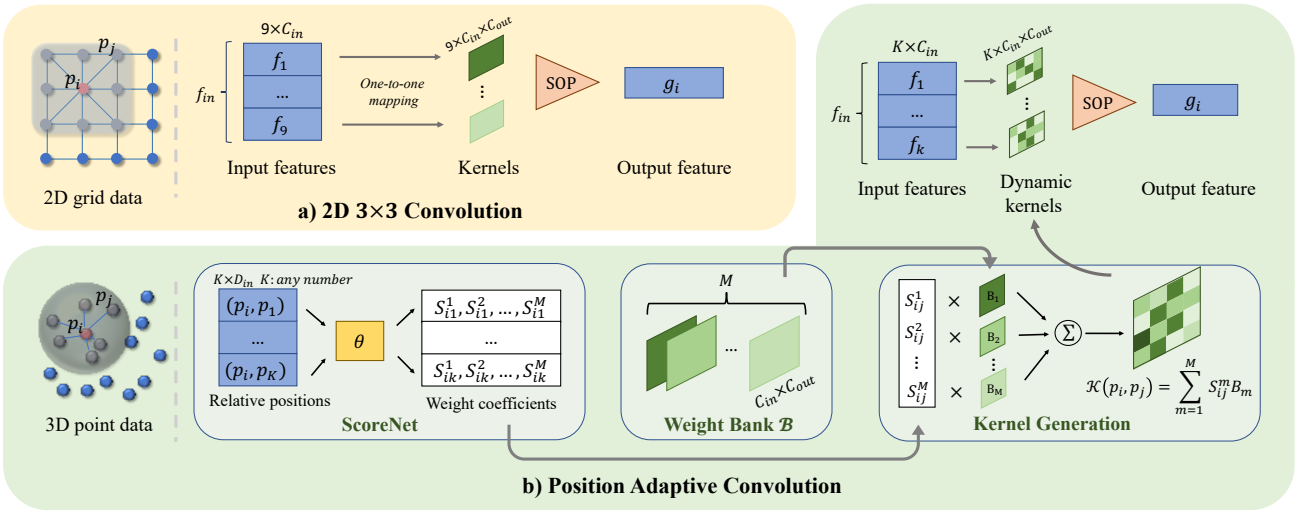


Figure 2. **PAConv**. (a) shows the traditional 2D convolution operators where SOP means symmetric operations, like MAX. (b) illustrates how our PAConv designs the kernel function $\mathcal{K}(p_i, p_j)$, including defining Weight Bank $\mathcal{B}$, learning ScoreNet and generating kernels.

By doing this, our PAConv constructs the convolution kernel in a dynamic data-driven manner, where the score coefficients S_{ij}^m are self-adaptively learned from point positions. Our position-adaptive convolution gains flexibility in modeling irregular geometric structures of 3D point clouds with the kernel assembly strategy.

3.3. Weight Regularization

While a large size of Weight Bank implies more weight matrices are available, the diversity of weight matrices is not ensured since they are randomly initialized and may converge to be similar with each other. To avoid this, we design a weight regularization to penalize the correlations between different weight matrices, which is defined as:

$$\mathcal{L}_{corr} = \sum_{B_i, B_j \in \mathcal{B}, i \neq j} \frac{|\sum B_i B_j|}{||B_i||_2 ||B_j||_2}. \quad (4)$$

This enforces weight matrices to be diversely distributed, further promises the diversity in the generated kernels.

3.4. Relation to Prior Work

• **Relation to PointCNN** [24]. PointCNN designs an MLP-based $\mathcal{X}$-transformation to permute point features and associate them with corresponding kernels by weighted combination. However, the operator cannot preserve permutation-invariance which is important for point cloud processing. Our PAConv, nevertheless, learns kernels from position relations, naturally maintaining shape information, and utilize the symmetric function to ensure permutation-invariant.

• **Relation to PointConv** [52]. PAConv differs from PointConv in the following folds: 1) PointConv treats convolutional kernels as nonlinear functions of point positions and densities. Instead, PAConv regards each weight matrix as a basis to capture certain spatial relations. These bases are further dynamically assembled via learnable ScoreNet to model continuous point position relations. 2) Our insight yields the following designs customized for PAConv, which is more flexible and effective: (a) *Softmax* normalization optimizes kernel scores as a whole, where higher scores imply stronger links between B_m and spatial relations. We can also use other norms (*e.g.* Sigmoid, Tanh in Table 5). (b) $\mathcal{L}_{corr}$ encourages B_m to be independent with each other; (c) More generic feature aggregation operation can be exploited: PAConv uses max-pooling, while Efficient PointConv can only realize sum-pooling.

• **Relation to KPConv** [47]. PAConv and KPConv both strive to design the kernel function in a position adaptive way, yet there exists two key differences: 1) KPConv generates fixed kernel points with corresponding weights offline by optimization, where the kernel point space may need to be specifically tuned for different point cloud datasets, which is sensitive to hyper-parameters. However, our PAConv defines weight matrices without requiring the estimation of kernel point locations. 2) KPConv uses hand-crafted relation to combine weight matrices, which may be sub-optimal and limited in flexibility. In contrast, PAConv defines a learnable ScoreNet to predict a vector coefficients adapted to point positions. PAConv is more flexible in both kernel design and weight learning, easily to be integrated with different architectures.

4. Backbone Network Architectures

The network configurations largely vary across recent point cloud networks [52, 18, 32, 47, 25], yet most of them

can be considered as different variants of the classical point-wise MLP-based networks [52, 18, 47]. To assess the effectiveness of PAConv and minimize the impact from complicated network architectures, we employ three classical and simple MLP-based network backbones for different 3D tasks, and integrate our PAConv without further modifications of network architectures.

Networks for object-level tasks. The object-level tasks deal with individual 3D objects, which can be effectively solved using lightweight networks without down-sampling layers. Thus the scale/resolution of the point cloud is fixed through the whole network. PointNet [36] and DGCNN [51] are two representatives, which are chosen as the backbones for object classification and shape part segmentation. We directly replace the MLPs in the encoders of PointNet and *EdgeConv* [51] of DGCNN with PAConv without changing the original network architectures.

DGCNN [51] computes pairwise distance in feature space and takes the closest k points for each point, which brings huge computational cost and memory usage. Instead, we search the k-nearest neighbors in 3D coordinate space.

Network for scene-level tasks. For large-scale scene-level segmentation tasks, it is necessary to employ the networks with encoder (downsampling) and decoder (upsampling). This effectively enlarges the receptive field of the network while achieving faster speed and less memory usage. PointNet++ [37] is such a pioneering architecture.

For the encoder, we follow PointNet++ which uses iterative farthest point sampling (FPS) to downsample point clouds. When building neighborhoods, PointNet++ finds all points within a ball centered at the query point. The ball radius is critical for performance and need to be tuned for different point cloud scales, thus we directly search k-nearest neighbor for flexibility. In addition, we adopt the simplest Single-scale grouping (SSG) approach instead of sophisticated MSG and MRG. The learned features are thus directly propagated to the next layer without feature fusion tricks.

Similar to object-level tasks, we directly replace the MLPs in the encoding layers of PointNet++ with PAConv. Our decoder is the same as PointNet++. The detailed network architectures are shown in the supplementary material.

5. Experiments

We integrate PAConv into different point cloud networks mentioned in Sec. 4 and evaluate it on object classification, shape part segmentation and indoor scene segmentation. We implement a CUDA layer to efficiently realize PAConv, which is presented in the supplementary material.

5.1. Object Classification

Dataset. First we evaluate our model on ModelNet40 [53] for object classification. It consists 3D meshed models from 40 categories, with $9,843$ for training and $2,468$ for testing.

Method (time order)	Input	Accuracy
MVCNN [44]	multi-view	90.1
OctNet [39]	hybrid grid octree	86.5
PointwiseCNN [15]	1K points	86.1
PointNet++ [37]	1K points	90.7
PointNet++ [37]	5K points+normal	91.9
SpecGCN [48]	2K points+normal	92.1
PCNN [2]	1K points	92.3
SpiderCNN [58]	1K points+normal	92.4
PointCNN [24]	1K points	92.5
PointWeb [18]	1K points+normal	92.3
PointConv [52]	1K points+normal	92.5
RS-CNN [29] w/o vot.	1K points	92.4
RS-CNN [29] w/ vot.	1K points	93.6
KPConv [47]	1K points	92.9
InterpCNN [32]	1K points	93.0
DensePoint [28]	1K points	93.2
Point2Node [12]	1K points	93.0
3D-GCN [26]	1K points	92.1
FPConv [25]	1K points	92.5
Grid-GCN [57]	1K points	93.1
PosPool [30]	5K points	93.2
PointNet [36]	1K points	89.2
PAConv (**PN*) w/o vot.	1K points	93.2 (**4.0**↑)
DGCNN [51]	1K points	92.9
PAConv (**DGC*) w/o vot.	1K points	93.6
PAConv (DGC*) w/ vot.**	**1K points**	**93.9 (1.0↑)**

Table 1. Classification accuracy (%) on ModelNet40 [53]. **PN* and **DGC* respectively denote using PointNet [36] and DGCNN [51] as the backbones. "vot." indicates multi-scale inference following [29]. PAConv obviously improves two baselines and surpasses other methods.

Implementation. As mentioned in Sec. 4, PAConv is utilized to replace the MLPs of the encoders in PointNet and *EdgeConv* of DGCNN. We sample $1,024$ points for training and testing following [36]. Following [51], the training data are augmented by randomly translating objects and shuffling points. We do not add $\mathcal{L}_{corr}$ (Sec. 3) while still achieving high performance due to the simplicity of the task.

Result. Table 1 summarizes the quantitative comparisons. PAConv significantly improves the classification accuracy with 4.0%↑ on PointNet and 1.0%↑ on DGCNN. Especially, the accuracy achieved by DGCNN+PAConv is 93.9%, which is an excellent result compared with recent works. Following RS-CNN [29], we perform voting tests with random scaling and average the predictions during test. Without voting, the accuracy of the released RS-CNN model drops to 92.4%, while PAConv still gets 93.6%. By eliminating the post-processing factor, the results without voting better reflects the performance gained purely from model designs and show the effectiveness of our PAConv.

5.2. Shape Part Segmentation

Dataset. PAConv is also evaluated on ShapeNet Parts [61] for shape part segmentation. It contains $16,881$ shapes with

Method (time order)	Cls. mIoU	Ins. mIoU
PointNet [36]	80.4	83.7
PointNet++ [37]	81.9	85.1
SynSpecCNN [62]	82.0	84.7
SPLATNet [43]	83.7	85.4
PCNN [2]	81.8	85.1
SpiderCNN [58]	82.4	85.3
SpecGCN [48]	-	85.4
PointCNN [24]	84.6	86.1
PointConv [52]	82.8	85.7
Point2Seq [27]	-	85.2
PVCNN [31]	-	86.2
RS-CNN [29] w/o vot.	84.2	85.8
RS-CNN [29] w/ vot.	84.0	86.2
KPConv [47]	**85.1**	**86.4**
InterpCNN [32]	84.0	86.3
DensePoint [28]	84.2	86.4
3D-GCN [26]	82.1	85.1
DGCNN [51]	82.3	85.2
PAConv (**DGC*) w/o vot.	84.2	86.0
PAConv (DGC*) w/ vot.**	84.6 (**2.3**↑)	86.1 (**0.9**↑)

Table 2. Shape part segmentation results (%) on ShapeNet Parts [61]. **DGC* indicates using DGCNN [51] as the backbone. "vot." indicates multi-scale inference following [29]. PAConv significantly improves both Class and Instance mIoU on DGCNN.

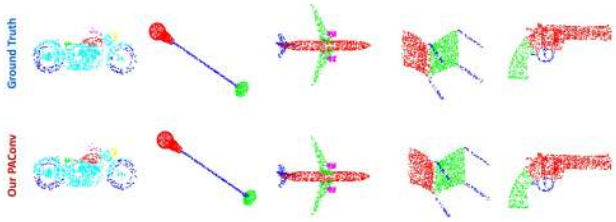


Figure 3. Visualization of shape part segmentation results on ShapeNet Parts. The first row is the ground truth, and the second row is the predictions of our PAConv. From left to right are motorbike, lamp, aeroplane, chair and pistol.

16 categories and is labeled in 50 parts where each shape has $2-5$ parts. $2,048$ points are sampled from each shape and each point is annotated with a part label.

Implementation. We displace *EdgeConv* in DGCNN [51] with PAConv and follow the official train/validation/test split of [51]. No data augmentations are used. Similar to the classification task, we do not employ $\mathcal{L}_{corr}$ and the same voting strategy during test is applied following [29].

Result. Table 2 lists the instance average and class average mean Inter-over-Union (mIoU), where PAConv notably lifts the performance of DGCNN on both class mIoU (**2.3**%↑) and instance mIoU (0.9%↑). PAConv also outperforms RS-CNN without voting (w/o vot.) Besides, our method surpasses or approaches other methods with much lower computational efficiency (analyzed in Sec. 5.3). Fig. 3 visualizes segmentation results. The mIoU of each class is shown in the supplementary material.

Method (time order)	Pre-proc.	mIoU	FLOPs
PointNet [36]	*BLK*	41.1	-
SegCloud [46]	*BLK*	48.9	-
TangentConv [45]	*BLK*	52.6	-
PointCNN [24]	*BLK*	57.26	-
ParamConv [50]	*BLK*	58.3	-
PointWeb [18]	*BLK*	60.28	-
PointEdge [17]	*BLK*	61.85	-
GACNet [49]	*BLK*	62.85	-
Point2Node [12]	*BLK*	62.96	-
KPConv *rigid* [47]	*Grid*	65.4	-
KPConv *deform*[47]	***Grid***	**67.1**	2042
FPConv [25]	*BLK*	62.8	-
SegGCN [22]	*BLK*	63.6	-
PosPool [30]	*Grid*	66.7	2041
PointNet++ [37]	*BLK*	57.27	991
PA w/o $\mathcal{L}_{corr}$ (**PN2*)	*BLK*	65.63	-
PA† w/ $\mathcal{L}_{corr}$ (**PN2*)	*BLK*	66.01	-
PA w/ $\mathcal{L}_{corr}$ (**PN2*) w/o vot.	*BLK*	66.33	-
PA w/ $\mathcal{L}_{corr}$ (PN2*) w/ vot.**	***BLK***	**66.58** (**9.31**↑)	**1253**

Table 3. Segmentation results (%) and #FLOPs/sample (M) on S3DIS Area-5 [1]. *BLK* and *Grid* signify using block sampling and grid sampling in data pre-processing, respectively. PA denotes PAConv, **PN2* refers to applying PointNet++ [37] as the backbone, and PA† symbolizes the CUDA implementation of PAConv. "vot." indicates multi-scale inference following [29].

5.3. Indoor Scene Segmentation

Dataset. Large-scale scene segmentation is a more challenging task. To further assess our method, we employ Stanford 3D Indoor Space (S3DIS) [1] following [18, 32, 30], which includes 271 rooms in 6 areas. 273 million points are scanned from 3 different buildings, and each point is annotated with one semantic label from 13 classes.

Implementation. We employ PAConv to replace the MLPs in the encoder of PointNet++ [37]. We follow [37] to prepare the training data, where the points are uniformly sampled into blocks of area size 1m $\times$ 1m, and each point is represented by a 9-dimensional vector (XYZ, RGB and a normalized location in the room). We randomly sample 4,096 points from each block on-the-fly, and all the points are adopted for testing. Following [46], we utilize Area-5 as the test scene and all the other areas for training. The data augmentations consist of random scaling, rotating, and perturbing points. The same voting test scheme as in the classification task is employed following [29].

NOTE: Different with our block sampling strategy, both KPConv [47] and PosPool [30] voxelize point clouds into grids. During training, the number of input points should be extremely large ($\approx 10\times$ ours) in their actual implementations. Although this brings more regular data structure and

more context information for better performance, it suffers from high memory usage during training.

Result. For the evaluation metrics, we use mean of class-wise intersection over union (mIoU). As shown in Table 3, our PAConv with $\mathcal{L}_{corr}$ (w/ $\mathcal{L}_{corr}$) achieves the best mIoU among all methods which use block sampling to pre-process data. PAConv also considerably promotes PointNet++ by **9.31**%↑. The result without voting (w/o vot.) is also listed. The visualization of segmentation results is shown in Fig. 4. The result of 6-fold cross-validation and the mIoU of each category is provided in the supplementary material.

Time complexity. Moreover, we take $4,096$ points as the input and test the time complexity (floating point operations/sample [‡]) of KPConv *deform* [47] and PosPool [30] as shown in Table 3. It demonstrates that our PAConv stands out with much less computational FLOPs (**38.6**%↓).

6. Ablation Studies

To better understand PAConv, ablation studies are conducted on S3DIS [1] dataset. Unless specified, *no* correlation loss (Sec. 3.3) is added to PAConv in all experiments.

6.1. ScoreNet

ScoreNet input. We firstly explore different input representations of ScoreNet. As illustrated in Table 4, when the ScoreNet input carries information from all three axes, PAConv can effectively utilize the rich relations to learn scores and achieve the best performance.

Input	mIoU
$(x_j - x_i, x_j, x_i, e_{ij})$	63.12
$(y_j - y_i, y_j, y_i, e_{ij})$	63.31
$(z_j - z_i, z_j, z_i, e_{ij})$	64.77
$(x_j - x_i, y_j - y_i, z_j - z_i, x_i, y_i, z_i, e_{ij})$	**65.63**

Table 4. Segmentation results (%) of PAConv with different ScoreNet input representations on S3DIS Area-5. While (x_j, y_j, z_j) represents the 3D coordinates of neighbor point, (x_i, y_i, z_i) indicates the center point position. e_{ij} refers to the Euclidean distance between neighbor point j and center point i.

Score normalization. We also investigate widely-used normalization functions in order to adjust the score distribution. Table 5 shows that Softmax normalization outperforms other schemes. It suggests that predicting scores for all weight matrices as a whole (Softmax) is superior than considering each score independently (Sigmoid and Tanh).

Score distribution in 3D space. More importantly, Fig. 5 shows the relationships between learned score distributions and different spatial planes. Notably, for each weight matrix B_i, B_j, B_k, the output scores are diversely distributed, indicating that different weight matrices capture different position relations. More explorations on ScoreNet are included in the supplementary material.

[‡] FLOPs from *torch.nn.module* is calculated by https://github.com/Lyken17/pytorch-OpCounter. FLOPs from *torch.nn.Parameter* is also added manually.

Normalization Function	mIoU
w/o normalization	64.28
Sigmoid	64.91
max(0, Tanh)	61.95
Softmax	**65.63**

Table 5. Segmentation results (%) on S3DIS Area-5 using PAConv with different normalization functions in ScoreNet. Normalization functions control the score distribution and determine the assembling of weight matrices.

6.2. The Number of Weight Matrices

We further conduct experiments to figure out the influence of the number of weight matrices as shown in Table 6. When the number of weight matrices is 2, the performance is 65.05%, only 0.58% apart from 16 weight matrices. This can be attributed to our kernel assembly strategy as diverse kernels will be generated even with only 2 weight matrices. This definitely demonstrates the power of our proposed approach. However, when the number becomes larger, the relative performance boost fluctuates due to optimization issues. Finally, we achieve the best and most stable performance when the number is 16.

# of weight matrices	mIoU	FLOPs(M)/ sample
2	65.05	**561.8**
4	64.86	651.1
8	64.39	839.1
16	**65.63**	1253

Table 6. Segmentation results (%) and #FLOPs/sample (M) of PAConv on S3DIS Area-5 using different numbers of weight matrices. Choosing more weight matrices ensures diversity of kernels selection and assembling.

6.3. Weight Bank Regularization

As mentioned in Sec. 3.3, weight regularization encourages weight matrices to have low correlations with each other, thus promising more diversity of kernel assembling. We utilize Pearson's R [4] to measure the correlation between different weight matrices and report the average Pearson's R (Lower Pearson's R value means lower correlations). As shown in Table 7, PAConv with the correlation loss outperforms the baseline with 0.95 mIoU on the scene segmentation task, and the Pearson's R between weight matrices remarkably drops.

Regularization	mIoU	Pearson's R [4]
w/o regularization	65.63	0.5393
w/ correlation loss	**66.58**	**-0.0333**

Table 7. Segmentation results (%) on S3DIS Area-5 and Pearson's R of PAConv with /without Weight Regularization. Regularized by correlation loss, weight matrices are low-correlated and diverse, bringing performance gains.

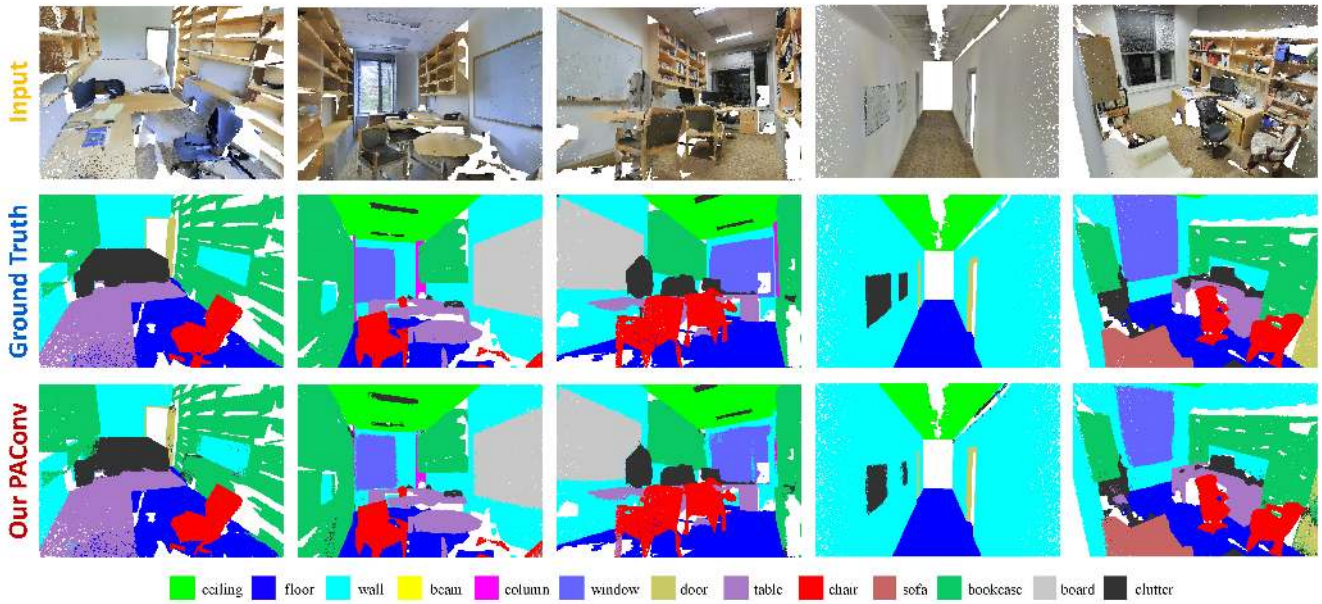


Figure 4. Visualization of semantic segmentation results on S3DIS Area-5. The first row shows original scene inputs, the second row shows the ground truth annotations, and the last row shows the scenes segmented by our PAConv. Each column denotes a scene in S3DIS Area-5.

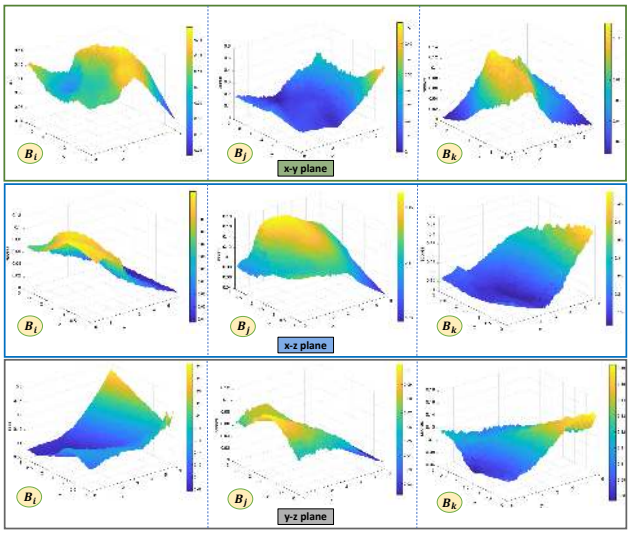


Figure 5. The spatial distribution of scores, where the input points are randomly initialized in x-y, x-z, y-z plane, and are sent to a trained ScoreNet. When the corresponding height of a point is higher (or the color is closer to yellow), the output score of this point is larger. It illustrates the relation between spatial positions and score distributions for each weight matrix B_i, B_j, B_k.

6.4. Robustness Analysis

PAConv uses a symmetric function to aggregate neighbor features, making it invariant to permutation and improving its robustness to rotation. Besides, the kernels are assembled by scores learned from diverse local spatial relations that may cover different transformations, which further enhances the robustness. We also evaluate our model in this respect. As shown in Table 8, PAConv performs stably well under various transformations.

Method	None	Perm.	90°	180°	270°	+0.2	-0.2	×0.8	×1.2	jitter
PN2	59.75	59.71	58.15	57.18	58.19	22.33	29.85	56.24	59.74	59.05
PAConv	65.63	65.64	61.66	63.48	61.8	**55.81**	**57.42**	64.20	63.94	65.12

Table 8. Test mIoU (%) on S3DIS Area-5 of perturbing the trained model. PN2 refers to our backbone PointNet++ [37]. We perform random permutation of points (Perm.), rotation around vertical axis (90°, 180°, 270°), translation in 3 directions (±0.2), scaling (×0.8, ×1.2) and Gaussian jittering (Jitter).

7. Conclusion

We have presented PAConv, a position adaptive convolution operator with dynamic kernel assembling for point cloud processing. PAConv constructs convolution kernels by combining basic weight matrices in Weight Bank, with the associated coefficients learned from point positions through ScoreNet. When embedded into simple MLP-based networks without modifications of network configurations, PAConv approaches or even surpasses the state-of-the-arts and significantly outperforms baselines with decent model efficiency. Extensive experiments and ablation studies illustrate the effectiveness of PAConv.

Acknowledgment

This work has been partially supported by HKU Start-up Fund and HKU Seed Fund for Basic Research.

References

[1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, 2016.

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 2018.

[3] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 2018.

[4] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*. 2009.

[5] Alexandre Boulch. Convpoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 2020.

[6] Alexandre Boulch, Bertrand Le Saux, and Nicolas Audebert. Unstructured Point Cloud Semantic Labeling Using Deep Segmentation Networks. In *Eurographics Workshop on 3D Object Retrieval*, 2017.

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

[8] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *NeurIPS*, 2016.

[9] Hang Gao, Xizhou Zhu, Stephen Lin, and Jifeng Dai. Deformable kernels: Adapting effective receptive fields for object deformation. In *ICLR*, 2020.

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

[11] Fabian Groh, Patrick Wieschollek, and Hendrik P. A. Lensch. Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In *ACCV*, 2018.

[12] Wenkai Han, Chenglu Wen, Cheng Wang, Xin Li, and Qing Li. Point2node: Correlation learning of dynamic-node for point cloud feature modeling. In *AAAI*, 2020.

[13] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 1991.

[14] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *CVPR*, 2020.

[15] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *CVPR*, 2018.

[16] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. Recurrent slice networks for 3d segmentation of point clouds. In *CVPR*, 2018.

[17] L. Jiang, H. Zhao, S. Liu, X. Shen, C. Fu, and J. Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *ICCV*, 2019.

[18] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *ICCV*, 2019.

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

[20] Artem Komarichev, Zichun Zhong, and Jing Hua. A-cnn: Annularly convolutional neural networks on point clouds. In *CVPR*, 2019.

[21] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Deep projective 3d semantic segmentation. In Michael Felsberg, Anders Heyden, and Norbert Krüger, editors, *Computer Analysis of Images and Patterns*, 2017.

[22] Huan Lei, Naveed Akhtar, and Ajmal Mian. Seggcn: Efficient 3d point cloud segmentation with fuzzy spherical kernel. In *CVPR*, 2020.

[23] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *CVPR*, 2018.

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

[25] Yiqun Lin, Zizheng Yan, Haibin Huang, Dong Du, Ligang Liu, Shuguang Cui, and Xiaoguang Han. Fpconv: Learning local flattening for point convolution. In *CVPR*, 2020.

[26] Zhi-Hao Lin, Sheng-Yu Huang, and Yu-Chiang Frank Wang. Convolution in the cloud: Learning deformable kernels in 3d graph convolution networks for point cloud analysis. In *CVPR*, 2020.

[27] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *AAAI*, 2019.

[28] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *ICCV*, 2019.

[29] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *CVPR*, 2019.

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

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

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

[33] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015.

[34] Hsien-Yu Meng, Lin Gao, Yu-Kun Lai, and Dinesh Manocha. Vv-net: Voxel vae net with group convolutions for point cloud segmentation. In *ICCV*, 2019.

[35] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2018.

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

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

[38] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *ICCV*, 2017.

[39] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017.

[40] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 2008.

[41] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *CVPR*, 2018.

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

[43] 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 *CVPR*, 2018.

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

[45] M. Tatarchenko, J. Park, V. Koltun, and Q. Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, 2018.

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

[47] 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 *ICCV*, 2019.

[48] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *ECCV*, 2018.

[49] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *CVPR*, 2019.

[50] S. Wang, S. Suo, W. Ma, A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *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 Trans. Graph.*, 2019.

[52] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019.

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

[54] Mutian Xu, Junhao Zhang, Zhipeng Zhou, Mingye Xu, Xiaojuan Qi, and Yu Qiao. Learning geometry-disentangled representation for complementary understanding of 3d object point cloud. *arXiv:2012.10921*, 2021.

[55] Mingye Xu, Zhipeng Zhou, and Yu Qiao. Geometry sharing network for 3d point cloud classification and segmentation. In *AAAI*, 2020.

[56] Mingye Xu, Zhipeng Zhou, Junhao Zhang, and Yu Qiao. Investigate indistinguishable points in semantic segmentation of 3d point cloud. *arXiv:2103.10339*, 2021.

[57] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-gcn for fast and scalable point cloud learning. In *CVPR*, 2020.

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

[59] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In *NeurIPS*, 2019.

[60] Zetong Yang, Yanan Sun, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Cn: Channel normalization for point cloud recognition. In *ECCV*, 2020.

[61] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Trans. Graph.*, 2016.

[62] Li Yi, Hao Su, Xingwen Guo, and Leonidas J. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, 2017.