Convolution in the Cloud: Learning Deformable Kernels in 3D Graph Convolution Networks for Point Cloud Analysis

Zhi-Hao Lin[1] Sheng-Yu Huang[1] Yu-Chiang Frank Wang[1,2]
[1]Graduate Institute of Communication Engineering, National Taiwan University, Taiwan
[2]ASUS Intelligent Cloud Services, Taiwan
{r08942062, r08942095, ycwang}@ntu.edu.tw

Abstract

Point clouds are among the popular geometry representations for 3D vision applications. However, without regular structures like 2D images, processing and summarizing information over these unordered data points are very challenging. Although a number of previous works attempt to analyze point clouds and achieve promising performances, their performances would degrade significantly when data variations like shift and scale changes are presented. In this paper, we propose ***3D Graph Convolution Networks (3D-GCN)****, which is designed to extract local 3D features from point clouds across scales, while shift and scale-invariance properties are introduced. The novelty of our 3D-GCN lies in the definition of learnable kernels with a graph max-pooling mechanism. We show that 3D-GCN can be applied to 3D classification and segmentation tasks, with ablation studies and visualizations verifying the design of 3D-GCN. Our code is publicly available at* https://github.com/j1a0m0e4sNTU/3dgcn.

1. Introduction

3D vision has been an active research topic, closely related to applications such as augmented reality, drones and self-driving vehicles [20, 15]. Existing 3D data representations include the use of voxel, mesh or point cloud features. Voxels [3, 33] describe 3D objects as voxel grids but generally suffer from insufficient resolution and high memory costs. Meshes are common in animations, while such representations are not directly associated with the 3D sensor outputs [16, 19]. Finally, 3D point clouds focus on describing shape information of 3D objects and can be easily acquired by 3D sensors, but the resulting unordered set of 3D points might limit the subsequent analysis tasks.

With the recent remarkable progress of deep learning techniques, in particular the Convolutional Neural Network (CNN), promising performances have been observed in a variety of computer vision tasks [6, 11]. However, image data are generally presented in terms of grid structures (e.g., pixels or cells over scales), which makes the convolution operation feasible. For irregular and unstructured data like 3D point clouds, it is not possible to learn and deploy kernels with fixed sizes or patterns on such data. In order to process such an unstructured/unordered set of points, PointNet [21] applies multiple fully connected layers to encode 3D point clouds, followed by global max-pooling operation, and shows impressive results for 3D data recognition and segmentation. Since global pooling operation is deployed, locally structured information of 3D point cloud might not be properly observed. Moreover, it is not designed to be invariant to global transformation like shift or scaling, as we later discuss and verify.

To better describe local information of 3D data, some research works utilize mesh data and their corresponding graph structures (i.e., vertices and associated edges) for extracting desirable information. For example, [27, 4] choose to propagate and aggregate features of adjacent vertices for deriving the final representation, while others [16, 2] define kernels with fixed patterns to observe local information.

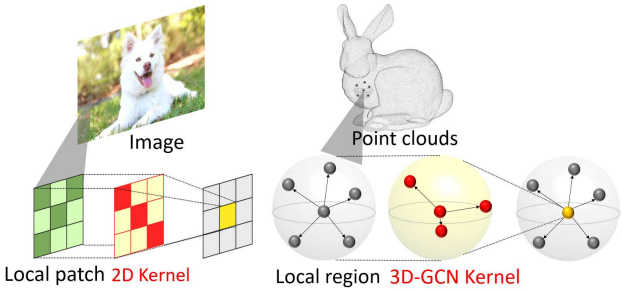


Figure 1: **Convolution in (a) 2D images and (b) 3D graphs.** Note that standard 2D CNN cannot be easily applied to handle 3D point cloud data, since the kernels in 3D graph convolution networks need to exhibit additional deformation in shape due to unstructured inputs.

Motivated by the above works, [26, 14, 1, 30, 32] attempt to construct graph-like structures for 3D point clouds. Since such methods consider global coordinates in representing their graphs, shift and scaling effects would degrade the performances of their models.

In this paper, we propose a novel deep learning model of 3D Graph Convolution Networks (3D-GCN) for processing and learning structural information of 3D point clouds. Motivated by 2D CNN, we aim at deriving *deformable 3D kernels*, whose shape and weights are learnable during the training stage. Moreover, similar to the max pooling operation in standard CNN, we perform a unique *graph max pooling* operation in 3D-GCN to summarize the processed features across different scales. As a result, our 3D-GCN is able to observe and extract structural information of unordered 3D point clouds with arbitrary shape and size. As detailed and confirmed later, our 3D-GCN is invariant to 3D point cloud shift and scaling changes, which are the key properties for real-world 3D vision applications.

We now summarize our main contributions as follows:

- We propose a 3D Graph Convolution Network (3D-GCN) for processing 3D point cloud data, exhibiting shift and scale-invariant properties for promising classification/segmentation performances.
- The shape and weights of each kernel in our 3D-GCN are learnable during training, which show capabilities in describing local structural information from unordered 3D point clouds.
- A novel graph max pooling is also introduced in 3D-GCN, allowing extraction and summarization of point cloud features across different scales for improved performances.

2. Related Work

Multi-view and voxelized methods. Since standard Convolutional Neural Networks cannot be directly applied to handle unstructured data like 3D point clouds, existing works typically choose to convert point clouds into proper representation for further processing purposes. For example, [33, 3, 18, 17, 25] register point features into voxel grid or directly use voxelized 3D shapes as the model input, so that standard 3D CNN can be performed on such input data. Although octree-based methods like [24, 31] have been proposed to refine the resolution of the predicted output, voxel-based methods are generally known to suffer from insufficient resolution and enormous memory consumption for 3D voxel representation.

Alternatively, a number of works [28, 9, 22, 29, 1] choose to project 3D shapes onto 2D planes in multiple views, followed by 2D CNN for feature extraction. While impressive results are reported, it cannot be easily extended to 3D data segmentation or reconstruction [11, 6].

Point-cloud based methods. In order to deal with 3D point cloud data, PointNet [21] has multiple shared fully connected layers to handle unordered 3D point inputs, followed by channel-wise max-pooling to extract global features to represent 3D point cloud data. While PointNet is able to handle 3D point cloud data without limitation to its unordered property, it essentially learns key point representation of the input object for deriving the final features. Local geometric information is not directly encoded, and this model would be sensitive to input translation and scaling variation, as we later verify.

To alleviate the aforementioned problem, researchers propose to sort the 3D points into an ordered list, where neighboring points have smaller euclidean distances in the 3D space. For example, [8] sorts all the points along different dimensions, then employ Recurrent Neural Networks (RNN) to extract features from the resulting sequences. [10, 5] convert the 3D points into an 1D list via kd-tree according to their coordinates, followed by 1D CNNs to extract the corresponding features. Nevertheless, sorting a 3D point set into an 1D list is not trivial; moreover, local geometric information may not be easily preserved in such ordered lists.

Geometry-based methods. Different from the aforementioned works which take the entire 3D data as the input, another branch of methods choose to learn local geometric information from a subset of 3D points [23, 32, 26, 14, 7, 13, 34, 30]. By dividing the 3D points into smaller groups, this type of approach extracts features from each local group for representation purposes. For example, PointNet++ [23] divides 3D point clouds into several ball regions, and apply [21] to each ball for local feature extraction. DGCNN [32] constructs local graphs by identifying the nearest neighbors of 3D points in the feature space, followed by the EdgeConv operation for feature extraction. Shen *et al*. [26] extend the above idea and additionally learn geometry information during feature aggregation. RS-CNN [14] applies weighted sum of neighboring point features, where each weight is learned with MLPs according to geometric relation between two points. These works attempt to extract geometrical information within local regions of 3D point clouds. However, existing methods typically use exact coordinates of points or distance vectors as the input features, and thus the model performance would be influenced by shifting and scaling effects, which would not be preferable for real-world applications like scene segmentation and multi-object detection. In this paper, we propose a novel 3D-GCN with learnable 3D graph kernels and Graph Max-Pooling mechanism, resulting in effective geometric features across different scales while exhibiting scale and shift-invariance.

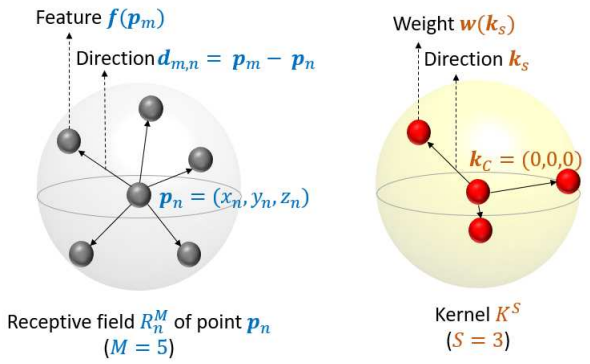


Figure 2: **Illustration of receptive field R_n^M and kernel K^S.** We have R_n^M indicates the M neighboring points for the nth point $\boldsymbol{p}_n$, and kernel K^S composes of S supports with center at $\boldsymbol{k}_C = (0, 0, 0)$. Note that directional vector $\boldsymbol{d}_{m,n}$ and $\boldsymbol{k}_s$ are used to measure the similarity in (4).

3. 3D Graph Convolution Networks

3.1. Notations

We present a novel 3D Graph Convolution Network (3D-GCN) to extract features from point cloud data, with applications to visual classification and semantic segmentation. For the sake of completeness, we define the notations used in this paper as follows. A point cloud instance is viewed as a set, which contains a total of N points $\mathcal{P} = \{\boldsymbol{p}_n \mid n = 1, 2, \ldots, N\}$ located on the surface of an object of interest. Note that $\boldsymbol{p}_n$ denotes the n-th point of this instance, and its attributes may describe coordinates (x_n, y_n, z_n), normal vector $(\nu_n^x, \nu_n^y, \nu_n^z)$, or RGB color information (r_n, g_n, b_n). In this work, the point cloud describes only the coordinates of each point on the object surface. In other words, we have $\boldsymbol{p}_n = (x_n, y_n, z_n)$, and thus a 3D point cloud object is represented by a matrix of size $N \times 3$.

For classification tasks, our 3D-GCN takes the point cloud input and produces the predicted output scores c for each class of interest. For semantic segmentation, we need to predict the part/scene label for each point in a 3D object. Thus, the output would be of size $N \times c$, which also indicates that the 3D-GCN performs point-wise classification in the task of semantic segmentation.

3.2. Deformable Kernels for 3D Graph Convolution

Receptive Fields in 3D-GCN A 3D point cloud object with N points is denoted as $\mathcal{P} = \{\boldsymbol{p}_n \mid n = 1, 2, \ldots, N\}, \boldsymbol{p}_n \in \mathbb{R}^3$. To describe the feature derived at each point in 3D-GCN, we have $\boldsymbol{f}(\boldsymbol{p}) \in \mathbb{R}^D$ denote the associated D-dimensional feature vector. To capture local geometric information of each point $\boldsymbol{p}_n$, we determine the 3D receptive field of $\boldsymbol{p}_n$ by a set of M neighboring points. As illustrated in Figure 2, we denote R_n^M as the receptive field of point $\boldsymbol{p}_n$ with size M as:

$$R_n^M = \{\boldsymbol{p}_n, \boldsymbol{p}_m \,|\, \forall \boldsymbol{p}_m \in \mathcal{N}(\boldsymbol{p}_n, M)\}, \tag{1}$$

where $\mathcal{N}(\boldsymbol{p}_n, M)$ denotes M nearest neighbors of $\boldsymbol{p}_n$ based on the distance $\|\boldsymbol{p}_m - \boldsymbol{p}_n\|$, and we have the corresponding *directional vector* $\boldsymbol{d}_{m,n} = \boldsymbol{p}_m - \boldsymbol{p}_n$ calculated for later convolution purposes. We note that, given a 3D point cloud object, we only need to determine the receptive fields for each point once. If the pooling operation is performed in later stages (as discussed in Sect. 3.3), receptive fields for the pooled point clouds at that scale needs to be constructed. In 3D-GCN, the features within the receptive field of $\boldsymbol{p}_n$ with size M is expressed as $\{\boldsymbol{f}(\boldsymbol{p}_n), \boldsymbol{f}(\boldsymbol{p}_m) \,|\, \forall \boldsymbol{p}_m \in \mathcal{N}(\boldsymbol{p}_n, M)\}$. These features will be calculated and updated during the convolution operation, as later discussed.

It is worth noting that, recent models for 3D point clouds [23, 30] select neighboring points for each point by a predefined radius r. While parameter r can be tuned to properly describe the local structures of 3D point clouds, their model cannot deal with scale variations as we later discuss and verify in Sect. 4.

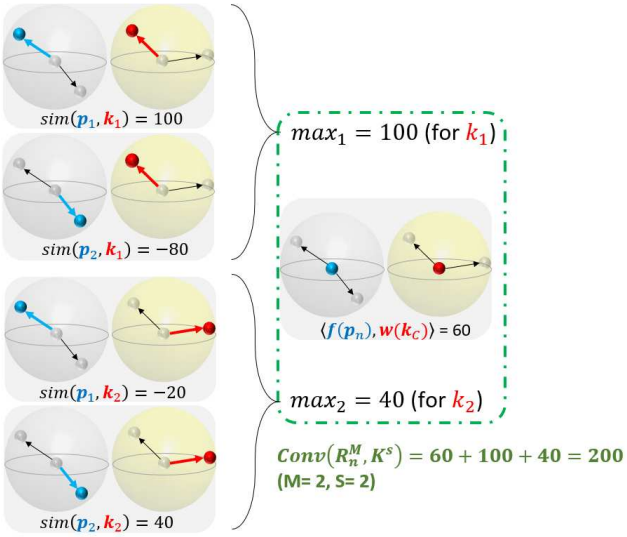


Figure 3: **3D Graph Convolution.** As in (4), $sim(\boldsymbol{p}_m, \boldsymbol{k}_s)$ calculates the inner product between $\boldsymbol{f}(\boldsymbol{p}_m)$ and $\boldsymbol{w}(\boldsymbol{k}_s)$ based on the cosine similarity between $\boldsymbol{d}_{m,n}$ and $\boldsymbol{k}_s$. For each support $\boldsymbol{k}_s$, the largest sim output among all neighbors $\boldsymbol{p}_m$ is obtained. Summing up with $\langle \boldsymbol{f}(\boldsymbol{p}_n), \boldsymbol{w}(\boldsymbol{k}_C) \rangle$ produces the final convolution output (i.e., (5)).

Learning Kernels in 3D-GCN In standard 2D CNN models, the kernel is composed of weight parameters in grids, sharing the same patterns across image patches (as depicted in Figure 1a). However, for 3D point cloud data, the data points are viewed as an unordered set, and no particular spatial 3D patterns can be observed.

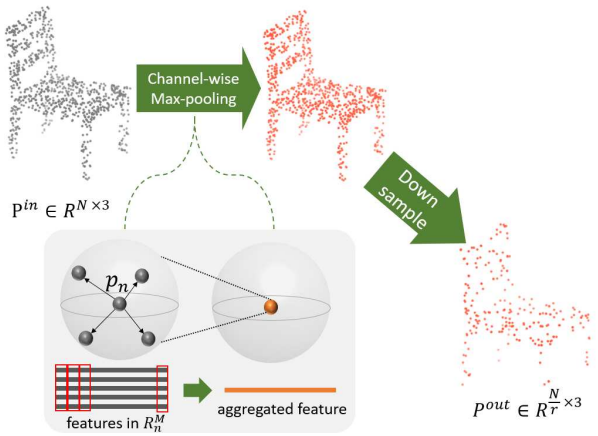


Figure 4: **Graph Max-Pooling.** This pooling process performs channel-wise max-pooling from the features in the receptive field of each $\boldsymbol{p}_n \in \mathcal{P}^{in}$, followed by randomly sampling a subset from $\mathcal{P}^{in}$ with a sampling rate r.

To perform convolution in 3D point cloud structures, we propose *3D Graph Convolution Kernel* K^S, where S denotes the number of supports in that kernel. More precisely, we have K^S composed of $S+1$ kernel points $\boldsymbol{k}_j \in \mathbb{R}^3$, i.e.,

$$K^S = \{\boldsymbol{k}_C, \boldsymbol{k}_1, \boldsymbol{k}_2, \ldots, \boldsymbol{k}_S\}. \tag{2}$$

Note that $\boldsymbol{k}_C = (0,0,0)$ is the center of the kernel, and $\boldsymbol{k}_1$ to $\boldsymbol{k}_S$ denote the associated supports.

In 2D CNN, each element in a kernel is the learned weight which describes the spatial patterns of interest. In our 3D-GCN, we define the weight vector $\boldsymbol{w}(\boldsymbol{k}) \in \mathbb{R}^D$ for each kernel point $\boldsymbol{k}$. Thus, the weighted-sum of features $\boldsymbol{f}(\boldsymbol{p})$ using the corresponding weights would achieve the convolution operation. Since the number and directions of supports might not be the same as the receptive fields of a 3D point, we need to define the directional vector $\boldsymbol{k}_s - \boldsymbol{k}_C = \boldsymbol{k}_s, s = 1, 2, \ldots, S$ for describing the pattern of the learned kernel. As illustrated in Figure 2, expect for $\boldsymbol{k}_C = (0,0,0)$, the kernel in 3D-GCN is now defined as $\{\boldsymbol{w}(\boldsymbol{k}_C), (\boldsymbol{k}_s, \boldsymbol{w}(\boldsymbol{k}_s)) \,|\, s = 1, 2, \ldots, S\}$, where each element is learned via training.

3D Graph Convolution In 2D CNN, the convolution operation can be regarded as calculating the *similarity* between the 2D kernel and the associated image patch. Larger output values indicate higher visual similarity. With the aforementioned receptive field and kernel definitions for 3D point cloud data, we define 3D Graph Convolution by calculating the similarity between R_n^M and K^S, denoted as $Conv(R_n^M, K^S)$.

However, unlike 2D CNN in which both kernel and image patches are of the same grid structures, it is not trivial to perform convolution in 3D graph structures. Thus, to measure the similarity between the features within the receptive field of $\boldsymbol{p_n}$ (i.e., $\boldsymbol{f}(\boldsymbol{p}_n), \boldsymbol{f}(\boldsymbol{p}_m), \forall \boldsymbol{p}_m \in \mathcal{N}(\boldsymbol{p}_n, M)$ as defined in (1)) and weight vectors of kernel K^S centered at $\boldsymbol{k}_C$ with S supports (i.e., $\boldsymbol{w}(\boldsymbol{k}_C), \boldsymbol{w}(\boldsymbol{k}_s), \forall s = 1, 2, \ldots, S$), we consider all possible pairs between $(\boldsymbol{p}_m, \boldsymbol{k}_s)$. Thus, $Conv(R_n^M, K^S)$ in 3D-GCN is defined as:

$$Conv(R_n^M, K^S) = \big\langle \boldsymbol{f}(\boldsymbol{p}_n), \boldsymbol{w}(\boldsymbol{k}_C) \big\rangle + g(\mathcal{A}), \tag{3}$$

where $\langle \cdot \rangle$ denotes the inner-product operation, and $\mathcal{A} = \{sim(\boldsymbol{p}_m, \boldsymbol{k}_s) \,|\, \forall m \in (1, M), \forall s \in (1, S)\}$. Note that the sim function in (3) is defined as:

$$sim(\boldsymbol{p}_m, \boldsymbol{k}_s) = \big\langle \boldsymbol{f}(\boldsymbol{p}_m), \boldsymbol{w}(\boldsymbol{k}_s) \big\rangle \frac{\langle \boldsymbol{d}_{m,n}, \boldsymbol{k}_s \rangle}{\|\boldsymbol{d}_{m,n}\| \|\boldsymbol{k}_s\|}, \tag{4}$$

which calculates the inner product between $\boldsymbol{f}(\boldsymbol{p}_n)$ and $\boldsymbol{w}(\boldsymbol{k}_C)$ based on their cosine similarity. The function g in (3) sums up the maximum similarity $sim(\boldsymbol{p}_m, \boldsymbol{k}_s)$ for each support $\boldsymbol{k}_s$ in that kernel. With the above definitions, the 3D Graph Convolution operation in our 3D-GCN is calculated as:

$$\begin{aligned} Conv(R_n^M, K^S) = &\big\langle \boldsymbol{f}(\boldsymbol{p}_n), \boldsymbol{w}(\boldsymbol{k}_C) \big\rangle \\ &+ \sum_{s=1}^{S} \max_{m \in (1,M)} \Big\{ sim(\boldsymbol{p}_m, \boldsymbol{k}_s) \Big\}. \end{aligned} \tag{5}$$

Recall that the neighbors M and supports S are the hyper-parameters (similar to the kernel sizes in 2D CNN). Note that (5) utilizes directional vectors $\boldsymbol{d}_{m,n}$ within R_n^M instead of global coordinates, which introduces *shift-invariant* property to our 3D-GCN model. In addition, the similarity function in (4) simply calculates the cosine similarity between $\boldsymbol{d}_{m,n}$ and $\boldsymbol{k}_s$ regardless of their lengths. Therefore, the scale-invariant property can be jointly observed by our 3D-GCN. Figure 3 illustrates 3D Graph Convolution operation in our 3D-GCN.

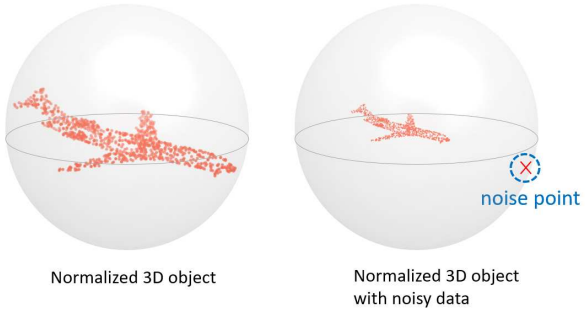
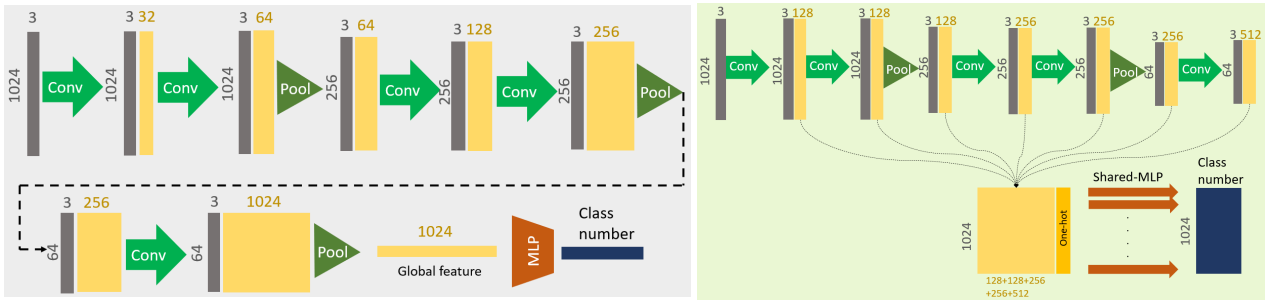


Figure 5: **Illustration of the lack of invariance property.** Recent models like PointNet [21] require techniques like zero-mean normalization for 3D point cloud representation, which might be sensitive to noisy 3D input points (as verified in Sect. 4).

3.3. Convolution and Pooling in 3D-GCN

3D Graph Convolution Layer A 3D Graph Convolution layer is composed of a pre-determined number L of



(a) **Classification** (b) **Part segmentation**

Figure 6: **Architecture of 3D-GCN for (a) classification and (b) part segmentation.** Note that grey and yellow blocks denote point and feature inputs, respectively. Green arrows denote 3D Graph Convolution Layers, while green triangles denote the Graph Max-Pooling layer. We have MLP and outputs denoted in brown and blue, respectively.

kernels K^S, each with a constant support number, which can be denoted as $\{K_i^S | i = 1, 2, \ldots, L\}$. Taking the 3D point cloud input $\mathcal{P} \in \mathbb{R}^{N\times 3}$ with the corresponding D-dimensional features $\mathcal{F}^{in} \in \mathbb{R}^{N\times D}$, our 3D Graph Convolution Layer applies (5) with each kernel respectively, producing output features $\mathcal{F}^{out} \in \mathbb{R}^{N\times L}$. Thus, each output layer $i = 1, 2, \ldots, L$ can be expressed as:

$$ConvLayer((\mathcal{P}, \mathcal{F}^{in}), K_i^S) = (\mathcal{P}, \mathcal{F}_i^{out}), \qquad (6)$$

where $\mathcal{F}_i^{out} \in \mathbb{R}^N$ is the i-th channel of $\mathcal{F}^{out}$. To initialize the convolution and learning process, we simply set $\boldsymbol{f}(\boldsymbol{p}) = 1, \forall \boldsymbol{p} \in \mathcal{P}$, and $\boldsymbol{w}(\boldsymbol{k}) = 1, \forall \boldsymbol{k}$ in the first input layer. In other words, we only consider directional information to initialize 3D-GCN operations.

3D Graph Max-Pooling Pooling operation plays an important role in standard 2D CNN, which summarizes the dominant responses within each scale for later high-level processing purposes, resulting in coarse-to-fine feature extraction. In 3D-GCN, we also propose a pooling operation, 3D Graph Max-Pooling, for performing similar mechanisms in 3D point clouds.

Our 3D Graph Max-Pooling layer takes the receptive field of each point R_n^M, and applies channel-wise max-pooling to aggregate features $\boldsymbol{f}(\boldsymbol{p}), \forall \boldsymbol{p} \in R_n^M$, followed by sampling a subset of $\mathcal{P}$ with a sampling rate r. Thus, this pooling process can be formulated as:

$$PoolLayer_r(\mathcal{P}^{in}, \mathcal{F}^{in}) = (\mathcal{P}^{out}, \mathcal{F}^{out}), \qquad (7)$$

where $\mathcal{P}^{in} \in \mathbb{R}^{N\times 3}, \mathcal{P}^{out} \in \mathbb{R}^{\frac{N}{r}\times 3}$, and $\mathcal{F}^{in} \in \mathbb{R}^{N\times D}, \mathcal{F}^{out} \in \mathbb{R}^{\frac{N}{r}\times D}$. As depicted in Figure 4, this pooling layer enables us to learn multi-scale 3D point cloud features, and make learning and calculation more efficient, which are crucial factors in 3D deep learning models.

Invariance Properties By learning the directional information within local receptive fields via 3D graph kernels, together with the pooling mechanism, our 3D-GCN exhibits promising shift and scale invariance properties. While existing works like [14, 21, 23, 26, 30] report promising performances, they typically consider global coordinates or require point cloud normalization to alleviate such data variances, which would limit their invariance properties (see examples in Figure 5). In Sect. 4), we will compare thorough experiments to confirm the robustness of our 3D-GCN when the above variants are presented.

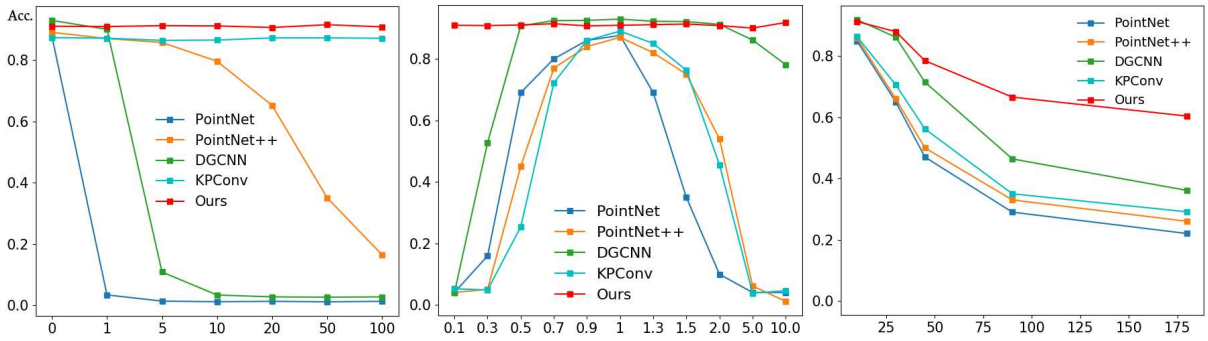
3.4. 3D Point Cloud Analysis

Classification To train 3D-GCN to recognize 3D point cloud data as particular categories, we apply and combine multiple 3D graph convolution and max-pooling layers, followed by adding multi-layer perceptron (MLP) for predicting the desirable outputs. Standard soft-max losses and backpropagation can be calculated to learn such 3D-GCN models (see Figure 6a for example architectures).

Semantic Segmentation One can also apply 3D-GCN to perform 3D point cloud semantic segmentation. To achieve this goal, we aggregate multi-scale features and advance a shared-MLP for point-wise classification. We note that, when aggregating features across different scales j and $j+1$, the number of 3D points do not match due to the pooling mechanism. Thus, for example, to concatenate the feature at $\boldsymbol{p}_n^j$ in scale j with that from $j+1$, we perform the following operation to identify the corresponding point of interest in scale $j+1$:

$$\boldsymbol{p}_{\tilde{n}}^{j+1} = \arg\min_{\boldsymbol{p}} \left\{ \|\boldsymbol{p} - \boldsymbol{p}_n^j\| \,|\, \forall \boldsymbol{p} \in \mathcal{P}^{j+1} \right\}. \qquad (8)$$

Thus, the concatenated features would be $\left(\boldsymbol{f}(\boldsymbol{p}_n^j), \boldsymbol{f}(\boldsymbol{p}_{\tilde{n}}^{j+1})\right)$ The architecture of 3D-GCN for semantic segmentation is illustrated in Figure 6b.



(a) (b) (c)

Figure 7: **Evaluation of invariance properties on ModelNet40. (a) Shift:** Objects randomly shifted within a distance along all directions (with unshifted version denoted as 0), **(b) Scale:** Objects scaled to different sizes (with the original size denoted as 1), **(c) Rotation:** Objects rotated along the upward direction (degree is denoted in this figure). Note that DGCNN in [32] was pre-trained on objects with scale variants (i.e., scale within [0.5, 1.5]), but it cannot handle unseen scale variants as shown in (b).

4. Experiments

4.1. 3D Model Classification

Dataset. We evaluate 3D-GCN for 3D shape classification on the ModelNet40 [33] dataset, which consists of 12311 CAD models of 40-categories, splitting into 9843 3D objects for training and 2468 for testing. To generate point clouds for training and testing, we sample 1024 points uniformly from the surface of each object without any normalization during training and testing.

Network configuration. Our 3D-GCN model structure for classification is shown in Figure 6a. The feature-extracting part is composed of 5 3D graph convolution layers, with kernel numbers $(32, 64, 128, 256, 1024)$ from low to high-level layers. We set the support number $S = 1$ for our kernels, and neighbor number $M = 25$ for the receptive fields. There are 3 3D Graph max-pooling layers in model structure, all with a fixed sample ratio $r = 4$. Following PointNet [21], the output feature of last 3D Graph Convolution Layer in our 3D-GCN is applied with global max-pooling, resulting a final feature representation of 1024 dimension. For classification, the MLP is of 2 layers, where Batchnorm and Dropout with drop ratio of 0.3 applied after the fist layer of MLP. We train our network with batch size 8, learning rate 0.0001 which is decayed half every 10 epochs, using the ADAM optimizer.

Results. The classification results of our 3D-GCN are listed in Table 1, in which we also compare our results with a number of recent approaches including PointNet [21], PointNet++ [23], DGCNN [32] and KPConv [30]. From this table, we see that our 3D-GCN is generally comparable or performs favorably against several state-of-the-art models when the test data are without any shift or scale variations presented.

Method	input	#points	Acc.(%)
ECC [27]	xyz	1k	87.4
PointNet [21]	xyz	1k	89.2
Kd-Net (depth=10) [10]	xyz	1k	90.6
PointNet++ [23]	xyz	1k	90.7
KCNet [26]	xyz	1k	91.0
MRTNet [5]	xyz	1k	91.2
DGCNN [32]	xyz	1k	92.9
SO-Net [12]	xyz	2k	90.9
KPConv rigid [30]	xyz	6.8k	92.9
PointNet++ [23]	xyz, normal	5k	91.9
SO-Net [12]	xyz, normal	5k	93.4
Ours	xyz	1k	92.1

Table 1: **Shape classification results on ModelNet40.** Note that "normal" denotes the normal vectors of object surfaces. We see that our method achieved comparable or improved results with inputs of size only 1k points.

To further evaluate the invariance properties of our model, we compare to the above models using 3D point cloud data with 1024 points, normalized to a unit sphere with zero mean, without data augmentation. We test them under three different situations: coordinate shift, shape scaling, and shape rotation. The results are shown in Figure 7a, 7b, and 7c, respectively. From the results shown in these figures, we see that the performance of PointNet and DGCNN significantly dropped with coordinate shifts, which is caused by extracting features from global coordinates. When scale variants are presented, only our model was able to perform recognition with satisfactory performances. As for shape rotation, better invariance ability was exhibited by our 3D-GCN. Thus, the above experiments confirm the effectiveness and robustness of our 3D-GCN.

method	class mIOU	instance mIOU	air plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skate board	table
Kd-Net [10]	77.4	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	84.9	87.4	86.7	78.1	51.8	69.9	80.3
MRTNet [5]	79.3	83.0	81.0	76.7	87.0	73.8	89.1	67.6	90.6	85.4	80.6	95.1	64.4	91.8	79.7	87.0	69.1	80.6
PointNet [21]	80.4	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
KCNet [26]	82.2	84.7	82.8	81.5	86.4	77.6	90.3	76.8	91.0	87.2	84.5	95.5	69.2	94.4	81.6	60.1	75.2	81.3
RS-Net [8]	81.4	84.9	82.7	86.4	84.1	78.2	90.4	69.3	91.4	87.0	83.5	95.4	66.0	92.6	81.8	56.1	75.8	82.2
SO-Net [12]	81.0	84.9	82.8	77.8	88.0	77.3	90.6	73.5	90.7	83.9	82.8	94.8	69.1	94.2	80.9	53.1	72.9	83.0
PointNet++ [23]	81.9	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
DGCNN [32]	82.3	85.2	84.0	83.4	86.7	77.8	90.6	74.7	91.2	87.5	82.8	95.7	66.3	94.9	81.1	63.5	74.5	82.6
KPConv deform [30]	85.1	86.4	84.6	86.3	87.2	81.1	91.1	77.8	92.6	88.4	82.7	96.2	78.1	95.8	85.4	69.0	82.0	83.6
Ours	82.1	85.1	83.1	84.0	86.6	77.5	90.3	74.1	90.9	86.4	83.8	95.6	66.8	94.8	81.3	59.6	75.7	82.8

Table 2: **Part segmentation results on ShapeNetPart.** Note that while our method achieved comparable results as state-of-the-art models did, our model complexity was significantly less than others as discussed in Sect. 4.4.

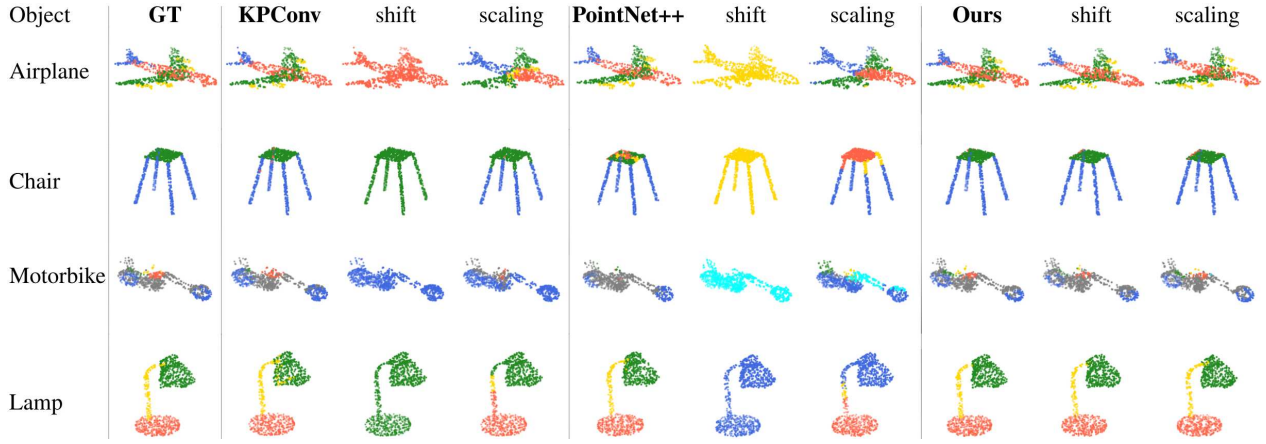


Figure 8: **Visualization of part segmentation on ShapeNetPart.** We compare our segmentation outputs to those produced by KPConv [30] and PointNet++ [23]. In addition, shift (by 100) and scale (by 10 times) variations are presented for evaluating the invariance capacity for each model. Note that GT denotes the ground truth part labels.

4.2. 3D Model Part Segmentation

Dataset. To evaluate the use of 3D-GCN for part segmentation, we consider the ShapeNetPart dataset [35], which consists of 16881 CAD models from 16 object types, with each point in an object corresponding to a part label. With a total of 50 categories, 2 to 6 part categories are available for each object type. In our work, we sample 1024 points from each 3D model for training and testing.

Network configuration. The model architecture is shown in Figure 6b. The feature-extracting part is composed of 5 layers with kernel numbers $(128, 128, 256, 256, 512)$ at the associated layers, and two 3D Graph Max-pooling layers with a fixed sample ratio $r = 4$ are deployed. We set the support number $S = 1$ for each kernel, and neighbor number $M = 50$ for the receptive field in 3D-GCN. The features for segmentation are concatenated from the layer outputs at different scales, as described in Section 3.4. Following PointNet [21], we also have one-hot vectors indicating object type concatenated to the above features, followed by 3 shared MLP layers to classify the segment label for each point. We train the 3D-GCN with learning rate 0.001 and decayed half every 10 epochs, using the ADAM optimizer.

Results. We evaluate the segmentation performance in terms of mean intersection over union (mIoU), which is the average IoU of each part type in that object category. Note that the mIoU of each category is calculated by averaging mIoUs of all the shape instances. More specifically, class mIoU is the average of mIoU over all 16 categories, while instance mIoU is the average of mIoU over all instances. The part segmentation results are listed in Table 2. Note that without using global coordinates, our 3D-GCN achieved comparable or better results than recent approaches did.

Furthermore, we demonstrate the robustness of 3D-GCN by visualizing the segmentation result under different transformations of an object, as shown in Figure 8. We shift the center/coordinates of each object by 100 and enlarge object size by 10 times, and Figure 8 compares our segmentation performances with others. We found that KPConv [30] and PointNet++ [23] failed to properly segment the corresponding parts in both cases. On the other hand, our 3D-GCN exhibited very promising invariance capabilities regardless of shift and scale variations.

Neighboring ratio (%)	5	10	30
Accuracy (%)	63.3	86.8	85.1

Table 3: **Effects on shape classification on ModelNet40 with varying neighboring ratio**. Note that insufficient numbers of neighbors are not expected to properly represent the receptive fields, while the performances would be less sensitive to larger numbers due to our learnable kernels.

Directional vector	A	B	C
Accuracy(%)	89.5	92.2	**93.9**

Table 4: **Effects on shape classification on ModelNet10 using learnable directional vector or not. A**: no directional information, **B**: assign three unit vectors (along 3 axes) as k_s, and **C**: our learnable directional vectors.

4.3. Ablation Study

Neighbor number M in receptive fields. We now conduct experiments on 3D-GCN by varying the neighbor number ratio when constructing the receptive fields in 3D-GCN (e.g., ratio 5% indicates we set the receptive field size as $M = 100 \times 0.05 = 5$). The results are shown in Table 3. From this table, we see that insufficient or excessive neighbor numbers would affect the performance of 3D-GCN in describing local structural information of 3D point clouds, thus moderate neighbor number leads to better performance.

Learning of directional vector k_s for each kernel. To demonstrate the power of learnable/deformable kernels in 3D-GCN, we consider three possible uses of directional vectors k_s in kernel K^S (we fix the support number S as 3). We first consider the inner product between the receptive field and the kernel as simply a correlation between the associated features, regardless of their geometry/cosine similarity. That is, (4) is simply replaced by $sim(\boldsymbol{p}_m, \boldsymbol{k}_s) = \langle \boldsymbol{f}(\boldsymbol{p}_m), \boldsymbol{w}(\boldsymbol{k}_s) \rangle$. The resulting accuracy is shown in the first column in Table 4. We next consider and assign 3 unit vectors along each axis (e.g., (1, 0, 0) along x-axis) as the 3 directional vectors k_s. Since these vectors are not learnable, and results shown in the second column of Table 4 was not satisfactory either. Finally, as shown in the last column of the table, we verify that the use of our learnable k_s would be desirable. Note that directional information is important for extracting geometric information, and learnable k_s makes kernel deformable and fitting the object of interest, which is why improved recognition performance can be achieved.

4.4. Visualization and Complexity analysis

In Figure 9, we visualize the points of an object which have large response values at each layer of our 3D-GCN. From low to high-level layers, we can see that responses were shifted from point to part levels, which confirms our ability in processing and summarizing 3D information across scales, which is equivalent to the use of 2D CNN in describing image data.

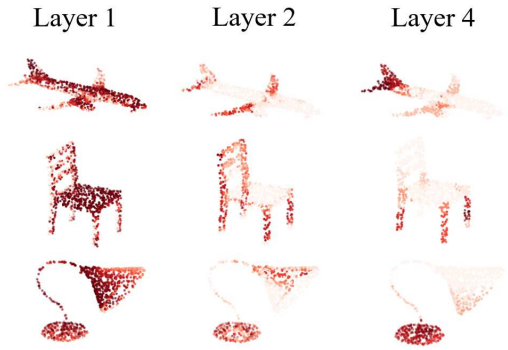


Figure 9: **Example kernel responses in different layers (segmentation on ShapeNetPart).** Note that points with larger responses are colored in darker red. As expected, the dominant responses are shifted from point (low) to part (high) levels in 3D-GCN.

method	#params	Acc. (%)
PointNet [21]	3.5M	89.2
PointNet++ [23]	1.48M	91.9
DGCNN [32]	1.81M	92.9
KPConv [30]	14.3M	92.9
Ours	**0.89M**	92.1

Table 5: **Number of parameters in different models (classification on ModelNet40).**

On the other hand, we compare the number of parameters of recent 3D point cloud models, and list the comparison outputs in Table 5. From this table, we see that our model achieves comparable recognition performances as state-of-the-art models did, while our model requires the fewest amount of parameters. This confirms both effectiveness and efficiency of our proposed 3D-GCN.

5. Conclusion

In this work, we introduced 3D-GCN which learns geometrical information of 3D point clouds across scales, and thus exhibits properties of shift and scale invariance. The technical contributions of our 3D-GCN lie in the design and learning of learnable kernels in 3D graphs, and the proposed scheme for graph max-pooling from 3D point clouds. While our model achieved comparable or improved classification and segmentation performances than recent state-of-the-art models did, we confirmed that our model is invariant to shift and scale changes and is computationally more efficient.

Acknowledgement This work is supported in part by the Ministry of Science and Technology of Taiwan under grant MOST 108-2634-F-002-018.

References

[1] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point Convolutional Neural Networks by Extension Operators. *arXiv preprint arXiv:1803.10091*, 2018. 2

[2] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning Shape Correspondence with Anisotropic Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, 2016. 1

[3] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 1, 2

[4] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. MeshNet: Mesh Neural Network for 3D Shape Representation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019. 1

[5] Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution Tree Networks for 3D Point Cloud Processing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2, 6, 7

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2

[7] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[8] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. Recurrent Slice Networks for 3D Segmentation of Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 7

[9] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[10] Roman Klokov and Victor Lempitsky. Escape From Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017. 2, 6, 7

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. 1, 2

[12] Jiaxin Li, Ben M Chen, and Gim Hee Lee. SO-Net: Self-Organizing Network for Point Cloud Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 6, 7

[13] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on X-Transformed Points. In *Advances in Neural Information Processing Systems*, 2018. 2

[14] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-Shape Convolutional Neural Network for Point Cloud Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2, 5

[15] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1

[16] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional Neural Networks on Riemannian Manifolds. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015. 1

[17] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for Real-time Object Recognition. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015. 2

[18] Hsien-Yu Meng, Lin Gao, YuKun Lai, and Dinesh Manocha. VV-Net: Voxel VAE Net with Group Convolutions for Point Cloud Segmentation. *arXiv preprint arXiv:1811.04337*, 2018. 2

[19] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[20] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1

[21] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 2, 4, 5, 6, 7, 8

[22] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and Multi-view CNNs for Object Classification on 3D Data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[23] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 2, 3, 5, 6, 7, 8

[24] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[25] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Classification of Point Cloud Scenes with Multiscale Voxel Deep Network. *arXiv preprint arXiv:1804.03583*, 2018. 2

[26] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining Point Cloud Local Structures by Kernel Correlation and

Graph Pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 5, 6, 7

[27] Martin Simonovsky and Nikos Komodakis. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 6

[28] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015. 2

[29] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent Convolutions for Dense Prediction in 3D. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[30] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 2, 3, 5, 6, 7, 8

[31] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 2

[32] 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)*, 38(5):146:1–146:12, 2019. 2, 6, 7, 8

[33] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 2, 6

[34] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2

[35] 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. *SIGGRAPH Asia*, 2016. 7