KPConv: Flexible and Deformable Convolution for Point Clouds

Hugues Thomas[1] Charles R. Qi[2] Jean-Emmanuel Deschaud[1] Beatriz Marcotegui[1]
François Goulette[1] Leonidas J. Guibas[2,3]

[1]Mines ParisTech [2]Facebook AI Research [3]Stanford University

Abstract

We present Kernel Point Convolution[1] (KPConv), a new design of point convolution, i.e. that operates on point clouds without any intermediate representation. The convolution weights of KPConv are located in Euclidean space by kernel points, and applied to the input points close to them. Its capacity to use any number of kernel points gives KPConv more flexibility than fixed grid convolutions. Furthermore, these locations are continuous in space and can be learned by the network. Therefore, KPConv can be extended to deformable convolutions that learn to adapt kernel points to local geometry. Thanks to a regular subsampling strategy, KPConv is also efficient and robust to varying densities. Whether they use deformable KPConv for complex tasks, or rigid KPconv for simpler tasks, our networks outperform state-of-the-art classification and segmentation approaches on several datasets. We also offer ablation studies and visualizations to provide understanding of what has been learned by KPConv and to validate the descriptive power of deformable KPConv.

1. Introduction

The dawn of deep learning has boosted modern computer vision with discrete convolution as its fundamental building block. This operation combines the data of local neighborhoods on a 2D grid. Thanks to this regular structure, it can be computed with high efficiency on modern hardware, but when deprived of this regular structure, the convolution operation has yet to be defined properly, with the same efficiency as on 2D grids.

Many applications relying on such irregular data have grown with the rise of 3D scanning technologies. For example, 3D point cloud segmentation or 3D simultaneous localization and mapping rely on non-grid structured data: point clouds. A point cloud is a set of points in 3D (or higher-dimensional) space. In many applications, the points are coupled with corresponding features like colors. In this work, we will always consider a point cloud as those two elements: the points $\mathcal{P} \in \mathbb{R}^{N \times 3}$ and the features $\mathcal{F} \in \mathbb{R}^{N \times D}$. Such a point cloud is a *sparse* structure that has the property to be *unordered*, which makes it very different from a grid. However, it shares a common property with a grid which is essential to the definition of convolutions: it is *spatially localized*. In a grid, the features are localized by their index in a matrix, while in a point cloud, they are localized by their corresponding point coordinates. Thus, the points are to be considered as structural elements, and the features as the real data.

Various approaches have been proposed to handle such data, and can be grouped into different categories that we will develop in the related work section. Several methods fall into the grid-based category, whose principle is to project the sparse 3D data on a regular structure where a convolution operation can be defined more easily [23, 28, 33]. Other approaches use multilayer perceptrons (MLP) to process point clouds directly, following the idea proposed by [47, 25].

More recently, some attempts have been made to design a convolution that operates directly on points [2, 44, 19, 14, 13]. These methods use the *spatial localization* property of

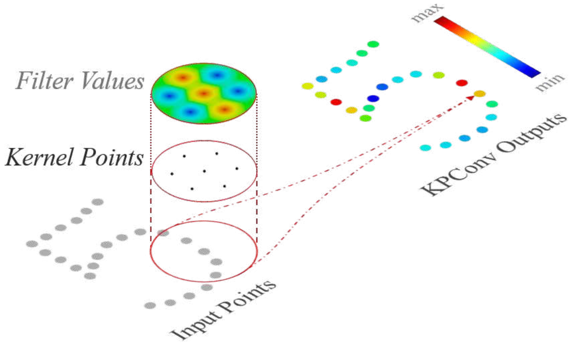

Figure 1. KPConv illustrated on 2D points. Input points with a constant scalar feature (in grey) are convolved through a KPConv that is defined by a set of kernel points (in black) with filter weights on each point.

[1] Project page: https://github.com/HuguesTHOMAS/KPConv

a point cloud to define point convolutions with spatial kernels. They share the idea that a convolution should define a set of customizable spatial filters applied locally in the point cloud.

This paper introduces a new point convolution operator named **Kernel Point Convolution** (KPConv). KPConv also consists of a set of local 3D filters, but overcomes previous point convolution limitations as shown in related work. KPConv is inspired by image-based convolution, but in place of kernel pixels, we use a set of kernel points to define the area where each kernel weight is applied, like shown in Figure 1. The kernel weights are thus carried by points, like the input features, and their area of influence is defined by a correlation function. The number of kernel points is not constrained, making our design very flexible. Despite the resemblance of vocabulary, our work differs from [31], which is inspired from point cloud registration techniques, and uses kernel points without any weights to learns local geometric patterns.

Furthermore, we propose a deformable version of our convolution [7], which consists of learning local shifts applied to the kernel points (see Figure 3). Our network generates different shifts at each convolution location, meaning that it can adapt the shape of its kernels for different regions of the input cloud. Our deformable convolution is not designed the same way as its image counterpart. Due to the different nature of the data, it needs a regularization to help the deformed kernels fit the point cloud geometry and avoid empty space. We use Effective Receptive Field (ERF) [21] and ablation studies to compare rigid KPConv with deformable KPConv.

As opposed to [40, 2, 44, 19], we favor radius neighborhoods instead of k-nearest-neighbors (KNN). As shown by [13], KNN is not robust in non-uniform sampling settings. The robustness of our convolution to varying densities is ensured by the combination of radius neighborhoods and regular subsampling of the input cloud [37]. Compared to normalization strategies [13, 14], our approach also alleviates the computational cost of our convolution.

In our experiments section, we show that KPConv can be used to build very deep architectures for classification and segmentation, while keeping fast training and inference times. Overall, rigid and deformable KPConv both perform very well, topping competing algorithms on several datasets. We find that rigid KPConv achieves better performances on simpler tasks, like object classification, or small segmentation datasets. Deformable KPConv thrives on more difficult tasks, like large segmentation datasets offering many object instances and greater diversity. We also show that deformable KPConv is more robust to a lower number of kernel points, which implies a greater descriptive power. Last but not least, a qualitative study of KPConv ERF shows that deformable kernels improve the network ability to adapt to the geometry of the scene objects.

2. Related Work

In this section, we briefly review previous deep learning methods to analyze point clouds, paying particular attention to the methods closer to our definition of point convolutions.

Projection networks. Several methods project points to an intermediate grid structure. Image-based networks are often multi-view, using a set of 2D images rendered from the point cloud at different viewpoints [34, 4, 17]. For scene segmentation, these methods suffer from occluded surfaces and density variations. Instead of choosing a global projection viewpoint, [35] proposed projecting local neighborhoods to local tangent planes and processing them with 2D convolutions. However, this method relies heavily on tangent estimation.

In the case of voxel-based methods, the points are projected on 3D grids in Euclidean space [23, 29, 3]. Using sparse structures like octrees or hash-maps allows larger grids and enhanced performances [28, 9], but these networks still lack flexibility as their kernels are constrained to use $3^3 = 27$ or $5^3 = 125$ voxels. Using a permutohedral lattice instead of an Euclidean grid reduces the kernel to 15 lattices [33], but this number is still constrained, while KPConv allows any number of kernel points. Moreover, avoiding intermediate structures should make the design of more complex architectures like instance mask detector or generative models more straightforward in future works.

Graph convolution networks. The definition of a convolution operator on a graph has been addressed in different ways. A convolution on a graph can be computed as a multiplication on its spectral representation [8, 46], or it can focus on the surface represented by the graph [22, 5, 32, 24]. Despite the similarity between point convolutions and the most recent graph convolutions [38, 42], the latter learn filters on edge relationships instead of points relative positions. In other words, a graph convolution combines features on local surface patches, while being invariant to the deformations of those patches in Euclidean space. In contrast, KPConv combines features locally according to the 3D geometry, thus capturing the deformations of the surfaces.

Pointwise MLP networks. PointNet [25] is considered a milestone in point cloud deep learning. This network uses a shared MLP on every point individually followed by a global max-pooling. The shared MLP acts as a set of learned spatial encodings and the global signature of the input point cloud is computed as the maximal response among all the points for each of these encodings. The network's performances are limited because it does not consider local spatial relationships in the data. Following PointNet, some hierarchical architectures have been developed to aggregate local neighborhood information with MLPs [26, 18, 20].

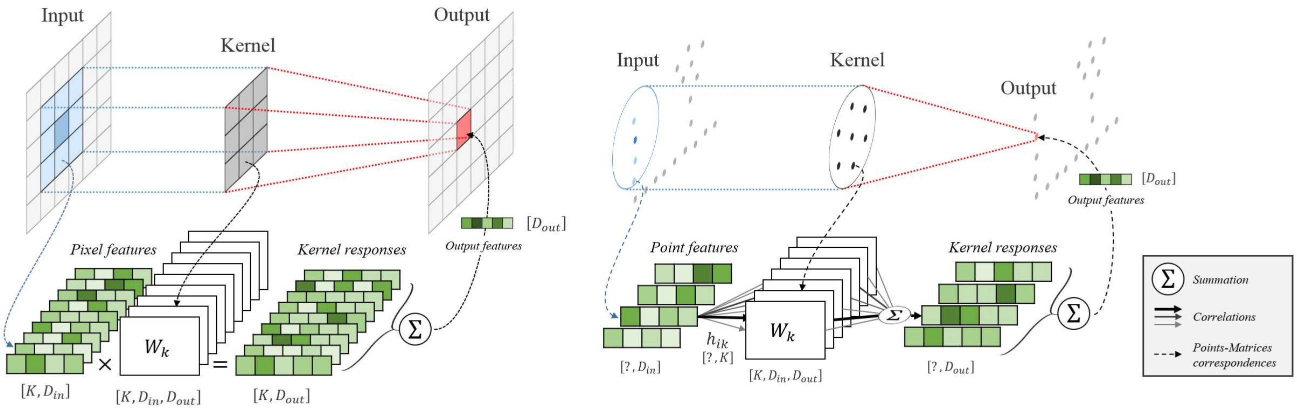

Figure 2. Comparison between an image convolution (left) and a KPConv (right) on 2D points for a simpler illustration. In the image, each pixel feature vector is multiplied by a weight matrix $(W_k)_{k<K}$ assigned by the alignment of the kernel with the image. In KPConv, input points are not aligned with kernel points, and their number can vary. Therefore, each point feature f_i is multiplied by all the kernel weight matrices, with a correlation coefficient h_{ik} depending on its relative position to kernel points.

As shown by [40, 19, 13], the kernel of a point convolution can be implemented with a MLP, because of its ability to approximate any continuous function. However, using such a representation makes the convolution operator more complex and the convergence of the network harder. In our case, we define an explicit convolution kernel, like image convolutions, whose weights are directly learned, without the intermediate representation of a MLP. Our design also offers a straightforward deformable version, as offsets can directly be applied to kernel points.

Point convolution networks. Some very recent works also defined explicit convolution kernels for points, but KPConv stands out with unique design choices.

Pointwise CNN [14] locates the kernel weights with voxel bins, and thus lacks flexibility like grid networks. Furthermore, their normalization strategy burdens their network with unnecessary computations, while KPConv subsampling strategy alleviates both varying densities and computational cost.

SpiderCNN [44] defines its kernel as a family of polynomial functions applied with a different weight for each neighbor. The weight applied to a neighbor depends on the neighbor's distance-wise order, making the filters spatially inconsistent. By contrast, KPConv weights are located in space and its result is invariant to point order.

Flex-convolution [10] uses linear functions to model its kernel, which could limit its representative power. It also uses KNN, which is not robust to varying densities as discussed above.

PCNN [2] design is the closest to KPConv. Its definition also uses points to carry kernel weights, and a correlation function. However, this design is not scalable because it does not use any form of neighborhood, making the convolution computations quadratic on the number of points. In addition, it uses a Gaussian correlation where KPConv uses a simpler linear correlation, which helps gradient backpropagation when learning deformations [7].

We show that KPConv networks outperform all comparable networks in the experiments section. Furthermore, to the best of our knowledge, none of the previous works experimented a spatially deformable point convolution.

3. Kernel Point Convolution

3.1. A Kernel Function Defined by Points

Like previous works, KPConv can be formulated with the general definition of a point convolution (Eq. 1), inspired by image convolutions. For the sake of clarity, we call x_i and f_i the points from $\mathcal{P} \in \mathbb{R}^{N\times 3}$ and their corresponding features from $\mathcal{F} \in \mathbb{R}^{N\times D}$. The general point convolution of $\mathcal{F}$ by a kernel g at a point $x \in \mathbb{R}^3$ is defined as:

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g(x_i - x) f_i \tag{1}$$

We stand with [13] advising radius neighborhoods to ensure robustness to varying densities, therefore, $\mathcal{N}_x = \left\{ \ x_i \in \mathcal{P} \ \middle| \ \|x_i - x\| \leqslant r \ \right\}$ with $r \in \mathbb{R}$ being the chosen radius. In addition, [37] showed that hand-crafted 3D point features offer a better representation when computed with radius neighborhoods than with KNN. We believe that having a consistent spherical domain for the function g helps the network to learn meaningful representations.

The crucial part in Eq. 1 is the definition of the kernel function g, which is where KPConv singularity lies. g takes the neighbors positions centered on x as input. We call them $y_i = x_i - x$ in the following. As our neighborhoods are defined by a radius r, the domain of definition of g is the ball $\mathcal{B}^3_r = \left\{ y \in \mathbb{R}^3 \mid \|y\| \leqslant r \right\}$. Like image convolution kernels (see Figure 2 for a detailed comparison between image convolution and KPConv), we want g to apply different weights to different areas inside this domain. There are many ways to define areas in 3D space, and points are the most intuitive as features are also localized by them. Let $\{\widetilde{x}_k \mid k < K\} \subset \mathcal{B}_r^3$ be the kernel points and $\{W_k \mid k < K\} \subset \mathbb{R}^{D_{in} \times D_{out}}$ be the associated weight matrices that map features from dimension D_{in} to D_{out}. We define the kernel function g for any point $y_i \in \mathcal{B}_r^3$ as :

$$g(y_i) = \sum_{k<K} h\left(y_i, \widetilde{x}_k\right) W_k \tag{2}$$

where h is the correlation between $\widetilde{x}_k$ and y_i, that should be higher when $\widetilde{x}_k$ is closer to y_i. Inspired by the bilinear interpolation in [7], we use the linear correlation:

$$h\left(y_i, \widetilde{x}_k\right) = \max\left(0, 1 - \frac{\|y_i - \widetilde{x}_k\|}{\sigma}\right) \tag{3}$$

where σ is the influence distance of the kernel points, and will be chosen according to the input density (see Section 3.3). Compared to a gaussian correlation, which is used by [2], linear correlation is a simpler representation. We advocate this simpler correlation to ease gradient backpropagation when learning kernel deformations. A parallel can be drawn with rectified linear unit, which is the most popular activation function for deep neural networks, thanks to its efficiency for gradient backpropagation.

3.2. Rigid or Deformable Kernel

Kernel point positions are critical to the convolution operator. Our rigid kernels in particular need to be arranged regularly to be efficient. As we claimed that one of the KPConv strengths is its flexibility, we need to find a regular disposition for any K. We chose to place the kernel points by solving an optimization problem where each point applies a repulsive force on the others. The points are constrained to stay in the sphere with an attractive force, and one of them is constrained to be at the center. We detail this process and show some regular dispositions in the supplementary material. Eventually, the surrounding points are rescaled to an average radius of 1.5σ, ensuring a small overlap between each kernel point area of influence and a good space coverage.

With properly initialized kernels, the rigid version of KPConv is extremely efficient, in particular when given a large enough K to cover the spherical domain of g. However it is possible to increase its capacity by learning the kernel point positions. The kernel function g is indeed differentiable with respect to $\widetilde{x}_k$, which means they are learnable parameters. We could consider learning one global set of $\{\widetilde{x}_k\}$ for each convolution layer, but it would not bring more descriptive power than a fixed regular disposition. Instead the network generates a set of K shifts $\Delta(x)$ for every convolution location $x \in \mathbb{R}^3$ like [7] and define deformable KPConv as:

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g_{deform}(x - x_i, \Delta(x)) f_i \tag{4}$$

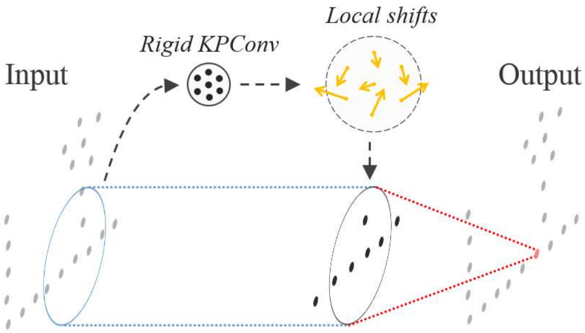

Figure 3. Deformable KPConv illustrated on 2D points.

$$g_{deform}(y_i, \Delta(x)) = \sum_{k<K} h\left(y_i, \widetilde{x}_k + \Delta_k(x)\right) W_k \tag{5}$$

We define the offsets $\Delta_k(x)$ as the output of a rigid KPConv mapping D_{in} input features to $3K$ values, as shown in Figure 3. During training, the network learns the rigid kernel generating the shifts and the deformable kernel generating the output features simultaneously, but the learning rate of the first one is set to 0.1 times the global network learning rate.

Unfortunately, this straightforward adaptation of image deformable convolutions does not fit point clouds. In practice, the kernel points end up being pulled away from the input points. These kernel points are lost by the network, because the gradients of their shifts $\Delta_k(x)$ are null when no neighbors are in their influence range. More details on these “lost” kernel points are given in the supplementary. To tackle this behaviour, we propose a “fitting” regularization loss which penalizes the distance between a kernel point and its closest neighbor among the input neighbors. In addition, we also add a “repulsive” regularization loss between all pair off kernel points when their influence area overlap, so that they do not collapse together. As a whole our regularization loss for all convolution locations $x \in \mathbb{R}^3$ is:

$$\mathcal{L}_{reg} = \sum_{x} \mathcal{L}_{fit}(x) + \mathcal{L}_{rep}(x) \tag{6}$$

$$\mathcal{L}_{fit}(x) = \sum_{k<K} \min_{y_i} \left(\frac{\|y_i - (\widetilde{x}_k + \Delta_k(x))\|}{\sigma}\right)^2 \tag{7}$$

$$\mathcal{L}_{rep}(x) = \sum_{k<K} \sum_{l \neq k} h\left(\widetilde{x}_k + \Delta_k(x), \widetilde{x}_l + \Delta_l(x)\right)^2 \tag{8}$$

With this loss, the network generates shifts that fit the local geometry of the input point cloud. We show this effect in the supplementary material.

3.3. Kernel Point Network Layers

This section elucidates how we effectively put the KPConv theory into practice. For further details, we have released our code using Tensorflow library.

Subsampling to deal with varying densities. As explained in the introduction, we use a subsampling strategy to control the density of input points at each layer. To ensure a spatial consistency of the point sampling locations, we favor grid subsampling. Thus, the support points of each layer, carrying the features locations, are chosen as barycenters of the original input points contained in all non-empty grid cells.

Pooling layer. To create architectures with multiple layer scales, we need to reduce the number of points progressively. As we already have a grid subsampling, we double the cell size at every pooling layer, along with the other related parameters, incrementally increasing the receptive field of KPConv. The features pooled at each new location can either be obtained by a max-pooling or a KPConv. We use the latter in our architectures and call it "strided KPConv", by analogy to the image strided convolution.

KPConv layer. Our convolution layer takes as input the points $\mathcal{P} \in \mathbb{R}^{N\times 3}$, their corresponding features $\mathcal{F} \in \mathbb{R}^{N\times D_{in}}$, and the matrix of neighborhood indices $\mathfrak{N} \in [\![1, N]\!]^{N'\times n_{max}}$. N' is the number of locations where the neighborhoods are computed, which can be different from N (in the case of "strided" KPConv). The neighborhood matrix is forced to have the size of the biggest neighborhood n_{max}. Because most of the neighborhoods comprise less than n_{max} neighbors, the matrix $\mathfrak{N}$ thus contains unused elements. We call them shadow neighbors, and they are ignored during the convolution computations.

Network parameters. Each layer j has a cell size dl_j from which we infer other parameters. The kernel points influence distance is set as equal to $\sigma_j = \Sigma \times dl_j$. For rigid KPConv, the convolution radius is automatically set to $2.5\,\sigma_j$ given that the average kernel point radius is $1.5\,\sigma_j$. For deformable KPConv, the convolution radius can be chosen as $r_j = \rho \times dl_j$. Σ and ρ are proportional coefficients set for the whole network. Unless stated otherwise, we will use the following set of parameters, chosen by cross validation, for all experiments: $K = 15$, $\Sigma = 1.0$ and $\rho = 5.0$. The first subsampling cell size dl_0 will depend on the dataset and, as stated above, $dl_{j+1} = 2 * dl_j$.

3.4. Kernel Point Network Architectures

Combining analogy with successful image networks and empirical studies, we designed two network architectures for the classification and the segmentation tasks. Diagrams detailing both architectures are available in the supplementary material.

KP-CNN is a 5-layer classification convolutional network. Each layer contains two convolutional blocks, the first one being strided except for the first layer. Our convolutional blocks are designed like bottleneck ResNet blocks [12] with a KPConv replacing the image convolution, batch normalization and leaky ReLu activation. After the last layer, the features are aggregated by a global average pooling and processed by the fully connected and softmax layers like in an image CNN. For the results with deformable KPConv, we only use deformable kernels in the last 5 KPConv blocks (see architecture details in the supplementary material).

KP-FCNN is a fully convolutional network for segmentation. The encoder part is the same as in KP-CNN, and the decoder part uses nearest upsampling to get the final pointwise features. Skip links are used to pass the features between intermediate layers of the encoder and the decoder. Those features are concatenated to the upsampled ones and processed by a unary convolution, which is the equivalent of a 1×1 convolution in image or a shared MLP in PointNet. It is possible to replace the nearest upsampling operation by a KPConv, in the same way as the strided KPConv, but it does not lead to a significant improvement of the performances.

4. Experiments

4.1. 3D Shape Classification and Segmentation

Data. First, we evaluate our networks on two common model datasets. We use ModelNet40 [43] for classification and ShapenetPart [45] for part segmentation. ModelNet40 contains 12,311 meshed CAD models from 40 categories. ShapenetPart is a collection of 16,681 point clouds from 16 categories, each with 2-6 part labels. For benchmarking purpose, we use data provided by [26]. In both cases, we follow standard train/test splits and rescale objects to fit them into a unit sphere (and consider units to be meters for the rest of this experiment). We ignore normals because they are only available for artificial data.

Classification task. We set the first subsampling grid size to $dl_0 = 2\text{cm}$. We do not add any feature as input; each input point is assigned a constant feature equal to 1, as opposed to empty space which can be considered as 0. This constant feature encodes the geometry of the input points. Like [2], our augmentation procedure consists of scaling, flipping and perturbing the points. In this setup, we are able to process 2.9 batches of 16 clouds per second on an Nvidia Titan Xp. Because of our subsampling strategy, the input point clouds do not all have the same number of points, which is not a problem as our networks accept variable input point cloud size. On average, a ModelNet40 object point cloud comprises 6,800 points in our framework. The other training parameters are detailed in the supplementary material, along with the architecture details. We also include the number of parameters and the training/inference speeds for both rigid and deformable KPConv.

As shown on Table 1, our networks outperform other state-of-the-art methods using only points (we do not take into account methods using normals as additional input). We also notice that rigid KPConv performances are slightly better. We suspect that it can be explained by the task simplicity. If deformable kernels add more descriptive power,

	ModelNet40	ShapeNetPart	
Methods	OA	mcIoU	mIoU
SPLATNet [33]	-	83.7	85.4
SGPN [41]	-	82.8	85.8
3DmFV-Net [9]	91.6	81.0	84.3
SynSpecCNN [46]	-	82.0	84.7
RSNet [15]	-	81.4	84.9
SpecGCN [39]	91.5	-	85.4
PointNet++ [26]	90.7	81.9	85.1
SO-Net [18]	90.9	81.0	84.9
PCNN by Ext [2]	92.3	81.8	85.1
SpiderCNN [44]	90.5	82.4	85.3
MCConv [13]	90.9	-	85.9
FlexConv [10]	90.2	84.7	85.0
PointCNN [19]	92.2	84.6	86.1
DGCNN [42]	92.2	85.0	84.7
SubSparseCNN [9]	-	83.3	86.0
KPConv *rigid*	**92.9**	85.0	86.2
KPConv *deform*	92.7	**85.1**	**86.4**

Table 1. 3D Shape Classification and Segmentation results. For generalizability to real data, we only consider scores obtained without shape normals on ModelNet40 dataset. The metrics are overall accuracy (OA) for Modelnet40, class average IoU (mcIoU) and instance average IoU (mIoU) for ShapeNetPart.

they also increase the overall network complexity, which can disturb the convergence or lead to overfitting on simpler tasks like this shape classification.

Segmentation task. For this task, we use KP-FCNN architecture with the same parameters as in the classification task, adding the positions (x, y, z) as additional features to the constant 1, and using the same augmentation procedure. We train a single network with multiple heads to segment the parts of each object class. The clouds are smaller (2,300 points on average), and we can process 4.1 batches of 16 shapes per second. Table 1 shows the instance average, and the class average mIoU. We detail each class mIoU in the supplementary material. KP-FCNN outperforms all other algorithms, including those using additional inputs like images or normals. Shape segmentation is a more difficult task than shape classification, and we see that KPConv has better performances with deformable kernels.

4.2. 3D Scene Segmentation

Data. Our second experiment shows how our segmentation architecture generalizes to real indoor and outdoor data. To this end, we chose to test our network on 4 datasets of different natures. Scannet [6], for indoor cluttered scenes, S3DIS [1], for indoor large spaces, Semantic3D [11], for outdoor fixed scans, and Paris-Lille-3D [30], for outdoor mobile scans. Scannet contains 1,513 small training scenes and 100 test scenes for online benchmarking, all annotated with 20 semantic classes. S3DIS covers six large-scale indoor areas from three different buildings for a total of 273 million points annotated with 13 classes. Like [36], we advocate the use of Area-5 as test scene to better measure the generalization ability of our method. Semantic3D is an online benchmark comprising several fixed lidar scans of different outdoor scenes. More than 4 billion points are annotated with 8 classes in this dataset, but they mostly cover ground, building or vegetation and there are fewer object instances than in the other datasets. We favor the *reduced-8* challenge because it is less biased by the objects close to the scanner. Paris-Lille-3D contains more than 2km of streets in 4 different cities and is also an online benchmark. The 160 million points of this dataset are annotated with 10 semantic classes.

Pipeline for real scene segmentation. The 3D scenes in these datasets are too big to be segmented as a whole. Our KP-FCNN architecture is used to segment small subclouds contained in spheres. At training, the spheres are picked randomly in the scenes. At testing, we pick spheres regularly in the point clouds but ensure each point is tested multiple times by different sphere locations. As in a voting scheme on model datasets, the predicted probabilities for each point are averaged. When datasets are colorized, we use the three color channels as features. We still keep the constant 1 feature to ensure black/dark points are not ignored. To our convolution, a point with all features equal to zero is equivalent to empty space. The input sphere radius is chosen as $50 \times dl_0$ (in accordance to Modelnet40 experiment).

Results. Because outdoor objects are larger than indoor objects, we use $dl_0 = 6\text{cm}$ on Semantic3D and Paris-Lille-3D, and $dl_0 = 4\text{cm}$ on Scannet and S3DIS. As shown in Table 2, our architecture ranks second on Scannet and outperforms all other segmentation architectures on the other datasets. Compared to other point convolution architectures [2, 19, 40], KPConv performances exceed previous scores by 19 mIoU points on Scannet and 9 mIoU points on S3DIS. SubSparseCNN score on Scannet was not reported in their original paper [9], so it is hard to compare without knowing their experimental setup. We can notice that, in the same experimental setup on ShapeNetPart segmentation, KPConv outperformed SubSparseCNN by nearly 2 mIoU points.

Among these 4 datasets, KPConv deformable kernels improved the results on Paris-Lille-3D and S3DIS while the rigid version was better on Scannet and Semantic3D. If we follow our assumption, we can explain the lower scores on Semantic3D by the lack of diversity in this dataset. Indeed, despite comprising 15 scenes and 4 billion points, it contains a majority of ground, building and vegetation points and a few real objects like car or pedestrians. Although this is not the case of Scannet, which comprises more than 1,500 scenes with various objects and shapes, our validation

Methods	Scannet	Sem3D	S3DIS	PL3D
Pointnet [25]	-	-	41.1	-
Pointnet++ [26]	33.9	-	-	-
SnapNet [4]	-	59.1	-	-
SPLATNet [33]	39.3	-	-	-
SegCloud [36]	-	61.3	48.9	-
RF_MSSF [37]	-	62.7	49.8	56.3
Eff3DConv [48]	-	-	51.8	-
TangentConv [35]	43.8	-	52.6	-
MSDVN [29]	-	65.3	54.7	66.9
RSNet [15]	-	-	56.5	-
FCPN [27]	44.7	-	-	-
PointCNN [19]	45.8	-	57.3	-
PCNN [2]	49.8	-	-	-
SPGraph [16]	-	73.2	58.0	-
ParamConv [40]	-	-	58.3	-
SubSparseCNN [9]	**72.5**	-	-	-
KPConv *rigid*	68.6	**74.6**	65.4	72.3
KPConv *deform*	68.4	73.1	**67.1**	**75.9**

Table 2. 3D scene segmentation scores (mIoU). Scannet, Semantic3D and Paris-Lille-3D (PL3D) scores are taken from their respective online benchmarks (reduced-8 challenge for Semantic3D). S3DIS scores are given for Area-5 (see supplementary material for k-fold).

studies are not reflected by the test scores on this benchmark. We found that the deformable KPConv outperformed its rigid counterpart on several different validation sets (see Section 4.3). As a conclusion, these results show that the descriptive power of deformable KPConv is useful to the network on large and diverse datasets. We believe KPConv could thrive on larger datasets because its kernel combines a strong descriptive power (compared to other simpler representations, like the linear kernels of [10]), and great learnability (the weights of MLP convolutions like [19, 40] are more complex to learn). An illustration of segmented scenes on Semantic3D and S3DIS is shown in Figure 4. More results visualizations are provided in the supplementary material.

4.3. Ablation Study

We conduct an ablation study to support our claim that deformable KPConv has a stronger descriptive power than rigid KPConv. The idea is to impede the capabilities of the network, in order to reveal the real potential of deformable kernels. We use Scannet dataset (same parameters as before) and use the official validation set, because the test set cannot be used for such evaluations. As depicted in Figure 5, the deformable KPConv only loses 1.5% mIoU when restricted to 4 kernel points. In the same configuration, the rigid KPConv loses 3.5% mIoU.

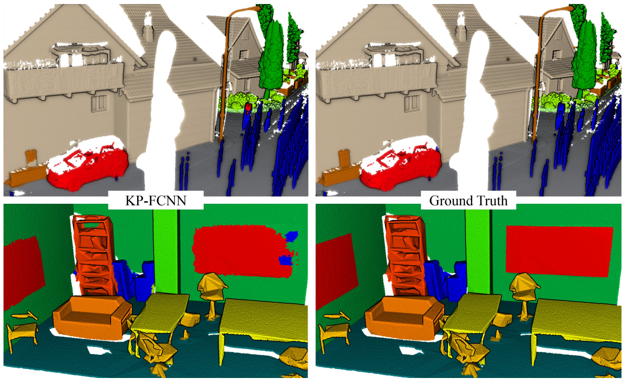

Figure 4. Outdoor and Indoor scenes, respectively from Semantic3D and S3DIS, classified by KP-FCNN with deformable kernels.

As stated in Section 4.2, we can also see that deformable KPConv performs better than rigid KPConv with 15 kernel points. Although it is not the case on the test set, we tried different validation sets that confirmed the superior performances of deformable KPConv. This is not surprising as we obtained the same results on S3DIS. Deformable KPConv seem to thrive on indoor datasets, which offer more diversity than outdoor datasets. To understand why, we need to go beyond numbers and see what is effectively learned by the two versions of KPConv.

4.4. Learned Features and Effective Receptive Field

To achieve a deeper understanding of KPConv, we offer two insights of the learning mechanisms.

Learned features. Our first idea was to visualize the features learned by our network. In this experiment, we trained KP-CNN on ModelNet40 with rigid KPConv. We added random rotation augmentations around vertical axis to increase the input shape diversity. Then we visualize each learned feature by coloring the points according to their level of activation for this features. In Figure 6, we chose input point clouds maximizing the activation for different features at the first and third layer. For a cleaner display, we

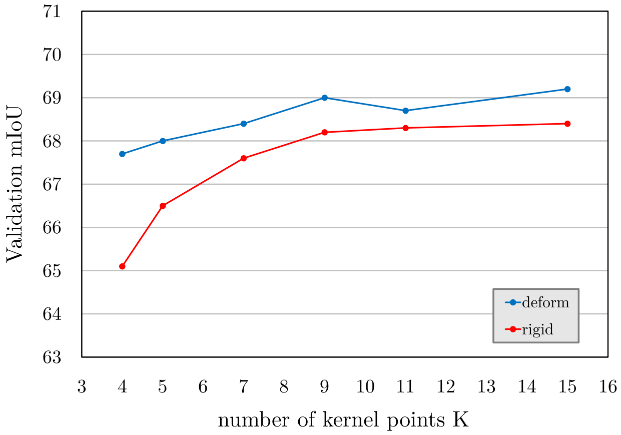

Figure 5. Ablation study on Scannet validation set. Evolution of the mIoU when reducing the number of kernel points.

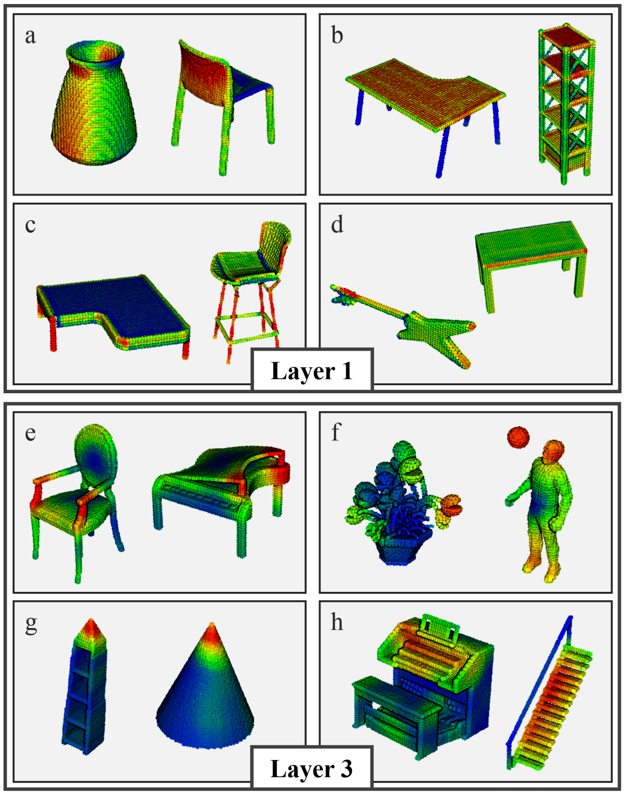

Figure 6. Low and high level features learned in KP-CNN. Each feature is displayed on 2 input point clouds taken from ModelNet40. High activations are in red and low activations in blue.

projected the activations from the layer subsampled points to the original input points. We observe that, in its first layer, the network is able to learn low-level features like vertical/horizontal planes (a/b), linear structures (c), or corners (d). In the later layers, the network detects more complex shapes like small buttresses (e), balls (f), cones (g), or stairs (h). However, it is difficult to see a difference between rigid and deformable KPConv. This tool is very useful to understand what KPConv can learn in general, but we need another one to compare the two versions.

Effective Receptive Field. To apprehend the differences between the representations learned by rigid and deformable KPConv, we can compute its Effective Receptive Field (ERF) [21] at different locations. The ERF is a measure of the influence that each input point has on the result of a KPConv layer at a particular location. It is computed as the gradient of KPConv responses at this particular location with respect to the input point features. As we can see in Figure 7, the ERF varies depending on the object it is centered on. We see that rigid KPConv ERF has a relatively consistent range on every type of object, whereas deformable KPConv ERF seems to adapt to the object size. Indeed, it covers the whole bed, and concentrates more on the chair that on the surrounding ground. When centered on a flat surface, it also seems to ignore most of it and reach for further details in the scene. This adaptive behavior shows that deformable KPConv improves the network ability to adapt to the geometry of the scene objects, and explains the better performances on indoor datasets.

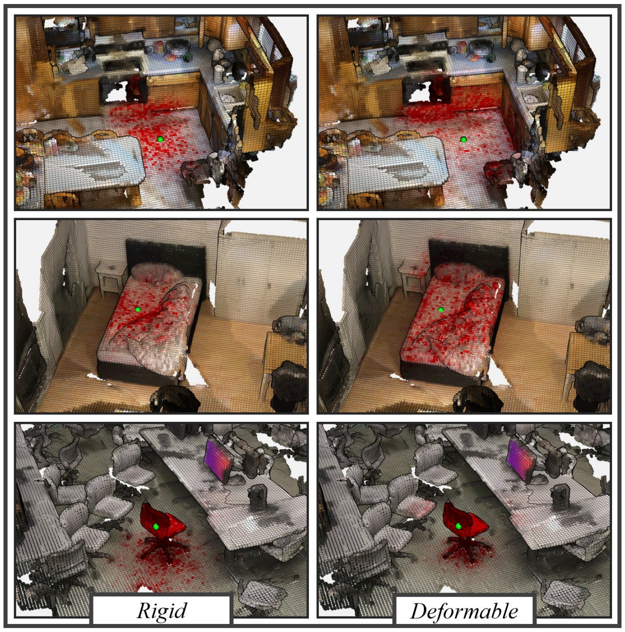

Figure 7. KPConv ERF at layer 4 of KP-FCNN, trained on Scannet. The green dots represent the ERF centers. ERF values are merged with scene colors as red intensity. The more red a point is, the more influence it has on the green point features.

5. Conclusion

In this work, we propose KPConv, a convolution that operates on point clouds. KPConv takes radius neighborhoods as input and processes them with weights spatially located by a small set of kernel points. We define a deformable version of this convolution operator that learns local shifts effectively deforming the convolution kernels to make them fit the point cloud geometry. Depending on the diversity of the datasets, or the chosen network configuration, deformable and rigid KPConv are both valuable, and our networks brought new state-of-the-art performances for nearly every tested dataset. We release our source code, hoping to help further research on point cloud convolutional architectures. Beyond the proposed classification and segmentation networks, KPConv can be used in any other application addressed by CNNs. We believe that deformable convolutions can thrive in larger datasets or challenging tasks such as object detection, lidar flow computation, or point cloud completion.

Acknowledgement. The authors gratefully acknowledge the support of ONR MURI grant N00014-13-1-0341, NSF grant IIS-1763268, a Vannevar Bush Faculty Fellowship, and a gift from the Adobe and Autodesk corporations.

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016. http://buildingparser.stanford.edu/dataset.html.

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics (TOG)*, 37(4):71, 2018.

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

[4] Alexandre Boulch, Bertrand Le Saux, and Nicolas Audebert. Unstructured point cloud semantic labeling using deep segmentation networks. In *Proceedings of the Workshop on 3D Object Retrieval (3DOR)*, 2017.

[5] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[6] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017. http://kaldir.vc.in.tum.de/scannet_benchmark.

[7] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international Conference on Computer Vision*, pages 764–773, 2017.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[9] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018.

[10] Fabian Groh, Patrick Wieschollek, and Hendrik P.A. Lensch. Flex-convolution. In *Asian Conference on Computer Vision*, pages 105–122. Springer, 2018.

[11] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D. Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017. http://www.semantic3d.net.

[12] 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*, pages 770–778, 2016.

[13] Pedro Hermosilla, Tobias Ristchel, Pere-Pau Vázquez, Álvaro Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)*, 37(6):235–1, 2018.

[14] 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*, pages 984–993, 2018.

[15] 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*, pages 2626–2635, 2018.

[16] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

[17] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Deep projective 3d semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns*, pages 95–107. Springer, 2017.

[18] 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*, pages 9397–9406, 2018.

[19] 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*, pages 820–830, 2018.

[20] 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. *arXiv preprint arXiv:1811.02565*, 2018.

[21] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pages 4898–4906, 2016.

[22] 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 workshops*, pages 37–45, 2015.

[23] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[24] 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*, pages 5115–5124, 2017.

[25] 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*, pages 652–660, 2017.

[26] Charles R. 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*, pages 5099–5108, 2017.

[27] Dario Rethage, Johanna Wald, Jurgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 596–611, 2018.

[28] 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*, volume 3, 2017.

[29] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Classification of point cloud scenes with multiscale voxel deep network. *arXiv preprint arXiv:1804.03583*, 2018.

[30] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, 2018. *http:// npm3d.fr*.

[31] 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*, volume 4, 2018.

[32] 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*, pages 3693–3702, 2017.

[33] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018.

[34] 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*, pages 945–953, 2015.

[35] 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*, pages 3887–3896, 2018.

[36] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)*, pages 537–547. IEEE, 2017.

[37] Hugues Thomas, François Goulette, Jean-Emmanuel Deschaud, and Beatriz Marcotegui. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. In *2018 International Conference on 3D Vision (3DV)*, pages 390–398. IEEE, 2018.

[38] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2598–2606, 2018.

[39] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–66, 2018.

[40] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

[41] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018.

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

[43] 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*, pages 1912–1920, 2015.

[44] 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)*, pages 87–102, 2018.

[45] Li Yi, Vladimir G. Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, Leonidas J. Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016.

[46] Li Yi, Hao Su, Xingwen Guo, and Leonidas J. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, pages 6584–6592, 2017.

[47] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R. Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[48] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds. In *2018 International Conference on 3D Vision (3DV)*, pages 399–408. IEEE, 2018.