SegGCN: Efficient 3D Point Cloud Segmentation with Fuzzy Spherical Kernel

Huan Lei Naveed Akhtar Ajmal Mian
School of Computer Science and Software Engineering
The University of Western Australia
huan.lei@research.uwa.edu.au, {naveed.akhtar,ajmal.mian}@uwa.edu.au

Abstract

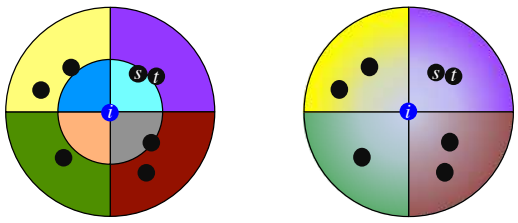
Fuzzy clustering is known to perform well in real-world applications. Inspired by this observation, we incorporate a fuzzy mechanism into discrete convolutional kernels for 3D point clouds as our first major contribution. The proposed fuzzy kernel is defined over a spherical volume that uses discrete bins. Discrete volumetric division can normally make a kernel vulnerable to boundary effects during learning as well as point density during inference. However, the proposed kernel remains robust to boundary conditions and point density due to the fuzzy mechanism. Our second major contribution comes as the proposal of an efficient graph convolutional network, SegGCN for segmenting point clouds. The proposed network exploits ResNet like blocks in the encoder and 1×1 *convolutions in the decoder. SegGCN capitalizes on the separable convolution operation of the proposed fuzzy kernel for efficiency. We establish the effectiveness of the SegGCN with the proposed kernel on the challenging S3DIS and ScanNet real-world datasets. Our experiments demonstrate that the proposed network can segment over one million points per second with highly competitive performance.*

1. Introduction

Learning directly from 3D point clouds is gaining increasing research interest due to its applications is autonomous vehicles and robotics in general. The main challenge here is that 3D point clouds captured by sensors (e.g. LiDAR) are unorganized, unlike images. Hence, the use of conventional CNN architectures is not feasible since they require organized grid-like inputs.

Currently, one of the most promising solution for feature learning from point clouds is to use the spatial-domain graph convolutional networks (GCNs). GCNs perform spatial convolutions on graph representations constructed from the point clouds. Most of the existing GCNs in the literature rely on mini-networks to achieve the graph convolution [24, 34, 43, 44], which incurs significant network complexity and computational overhead.

Discrete kernels have been recently proposed in the 3D Euclidean space [21, 40] that are well-suited for direct graph convolution without the need for mini-networks. Applications of these kernels to GCN architectures can make it significantly lightweight and efficient. Among these discrete kernels, the spherical kernel [21] partitions the local neighborhood in a compact way while preserving the attractive properties of translation-invariance and asymmetry similar to the standard CNN kernels [12, 16, 20, 35, 37], as well as permutation-invariance. More importantly, it does not require any online/offline learning of template points like the other kernels [33, 40]. Instead the kernel divides a spherical region systematically along its azimuth, elevation and radial directions into multiple non-overlapped bins.



(a) $4 \times 2 + 1$ hard kernel (b) $4 \times 1 + 1$ fuzzy kernel

Figure 1. Hard versus fuzzy spherical kernel (2D slices of 3D spheres are shown). The color gradient depicts fuzziness. Both kernels have 4 divisions along the azimuth. There is a split along the radial dimension for the hard kernel, assigning different coefficients to different bins. Hence, the neighbourhood points 's' and 't' of the target 'i' get very different weights. For the fuzzy kernel, these neighbors use similar parameters to compute the feature of 'i' as the product of their fuzzy coefficients and learnable parameters of their common bin.

However, since point coordinates in the 3D space ($\mathbb{R}^3$) are *real* numbers, such a discretization of the space into volumetric bins is potentially vulnerable to boundary effects, considering that many points will inevitably be very close to the boundaries. To address this issue and motivated by the success of fuzzy clustering in machine learning [4], we introduce the fuzzy mechanism into the spherical kernel. We remark that the standard CNN kernels do not suffer from the boundary effects because the image pixel coordinates are defined in Manhattan space. In Fig. 1, we provide a simple

example for unordered points in 2D to show how a fuzzy kernel can be robust to boundary effects during feature extraction. Note that, the spherical kernel degenerates to a circular kernel in 2D. As can be seen, the neighbor points s, t of the target point i locate very close to the boundary of radial bins. A hard kernel will use different parameters for s, t to compute the feature of the target point i. However, in a fuzzy kernel, the fuzzy coefficients result in s, t to use similar parameters to compute the feature of i.

In this paper, we propose fuzzy coefficients along the elevation and radial directions of the spherical kernel for robust 3D point cloud processing. The fuzzy spherical kernel avoids splits along the radial direction altogether. To apply our kernel effectively for semantic segmentation of point clouds, we also propose an encoder-decoder graph convolutional network SegGCN. The encoder exploits ResNet like blocks [12] for hierarchical feature learning, while the decoder comprises simple 1×1 convolutions to generate the final representations of the points. Within each ResNet block, we apply our fuzzy spherical kernel with depth-separable convolution [6] to aggregate the context information, which contributes significantly to the network efficiency. We construct the graph connections using range search [27], and coarsen the point cloud with farthest point sampling [29]. The point features are downsampled with max-pooling and upsampled with weighted interpolation. We demonstrate the performance of our SegGCN and fuzzy spherical kernel on two challenging real-world datasets, S3DIS and ScanNet. We also show that the fuzzy kernel is robust to decreasing point density. Our main contributions are summarized below:

- We propose a fuzzy spherical kernel by incorporating fuzzy mechanism into the spherical kernel of Lei *et al*. [21]. The new kernel applies spherical convolutions by separating the depth-wise and point-wise operations following Xception [6], and achieves superior performance in practical applications.
- We propose an efficient graph convolutional network architecture SegGCN for 3D semantic segmentation. With the use of separable fuzzy spherical convolution, SegGCN is able to segment over a million points per second with high accuracy.
- We provide CUDA implementations of the fuzzy spherical convolution in Tensorflow [1]. The source code is available on this Github link.

2. Related Work

3D-CNNs: 3D-CNNs learn features from voxel-grid representations of point clouds using grid-like kernels. The cubic growth of computational and memory requirements constrained early networks in this category to process low resolution inputs (e.g. 30×30×30 [45], 32×32×32 [26]). Engelcke *et al.* [10] proposed to reduce the computational overhead by making the input and intermediate feature maps sparse. However, their solution could not resolve the memory issue. The OctNet [30] improves the input resolution to 256×256×256 by reducing both the computational and memory costs with an octree-based representation. However, it is unable to avoid the redundant computations in the empty spaces. Recently, researchers transformed point clouds into other regular-grid representations such as tangent image [38] and high-dimensional lattice [36] so that standard CNNs can be applied.

MLPs: PointNet [28] is one of the first deep networks that directly takes the xyz coordinates of points as input features and learns representations of each point with multilayer perceptrons (MLPs). The limitation of PointNet is that MLPs cannot explore the geometric context during feature learning. PointNet++ [29] addresses this issue with hierarchical max-pooling. SO-Net [23] learns to reorganize the point cloud into a rectangular map, and exploits mini-PointNet to learn node-wise features within the map. KCNet [33] uses a distance-based kernel correlation between the local neighbors and the template points, but it is only applicable to point clouds with pure xyz coordinates. As the network goes deeper, it relies on MLPs to extract the features. Those template points are predefined and optimizable during network training. Kd-network [15] is a prominent tree structure based network for point clouds. It also uses point coordinates as the input and computes features of parent nodes by applying MLP to the concatenated features of their children. Despite the diversity of these networks, none of them contribute towards context learning with convolutional modules for point clouds.

GCNs: We group networks for point cloud processing into the GCN category as long as they define point-wise convolutions for context learning from the local neighbors. We ignore their originally explored architectures because those convolutional operations are readily adaptable to GCNs. Spectral graph convolutions entail high computational complexity for the signal transformation between different domains. It also requires a good alignment of the graph Laplacians of different point clouds. Yi *et al.* [47] addressed that for synthetic models with SpecTN, which is rather difficultly applied to real data. Most of the other networks focus on spatial graph convolutions.

ECC [34] is a pioneering work for point cloud analysis with spatial graph convolutions. Motivated by the dynamic filter networks [8], it introduces mini-networks composed of MLPs to generate the convolution filters dynamically on graph edges. Subsequently, Flex-Conv [11], SpiderCNN [46] and DGCNN [41] generate the edge filters using more effective parameterizations. Instead of generating the filters in an edge-wise convention, generating a complete convolution kernel using the mini-networks has also been explored [24, 44]. Li *et al.* [24] designed the $\mathcal{X}$-Conv

module for point clouds. However, their method is sensitive to the permutations of points. PointConv [44] computes the convolution kernel as a product of the local weight function and inverse density function. Unlike PointCNN, it uses 1×1 convolutions in the mini-networks making the kernel permutation-invariant. Wang *et al.* [41] proposed a graph attention convolutional network (GACNet) by inserting the attention mechanism in GCN. It still defines the point features as a weighted sum of its neighborhood features but the difference is that GACNet learns the weights with MLPs based not only on spatial proximity between the neighboring points and the target point, but also their feature similarity. Similar to CRF [17], such an attention mechanism encourages the neighbors to have consistent labels, leading to accurate semantic segmentation. However, the dependency on dynamic filter generation makes these networks computationally expensive.

Discrete kernel is an attractive alternative to avoid the computational overhead of dynamic networks. Lei *et al.* [21] proposed a spherical convolutional kernel for feature learning from point clouds, and demonstrated its effectiveness with octree-adapted architectures. Later, the kernel is also shown to be applied to graphs [22]. KPConv kernel [40] differs from the spherical kernel mainly in the decision of volumetric bins. It performs an offline data-independent training to obtain a number of template points, and uses these template points to divide the 3D metric space.

3. Method

Let $\mathcal{P} = \{\mathbf{x}_i \in \mathbb{R}^3\}_{i=1}^N$ be a point cloud and its feature maps be $\mathcal{F}^l = \{\mathbf{f}_i^l \in \mathbb{R}^{C_{in}}\}_{i=1}^N$. To perform convolution at a target point $\mathbf{x}_i$, we first construct its neighborhood set as $\mathcal{N}(\mathbf{x}_i) = \{\mathbf{x} \in \mathcal{P} : \|\mathbf{x} - \mathbf{x}_i\| \leq \rho\}$. With the neighborhood set $\mathcal{N}(\mathbf{x}_i)$, the general convolution kernel calculates the output feature of $\mathbf{x}_i$ in channel c as:

$$f_{ic}^{l+1} = \frac{1}{\mathcal{N}(\mathbf{x}_i)} \sum_{\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)} \langle \mathcal{K}_c(\mathbf{x} - \mathbf{x}_i), \mathbf{f}^l \rangle, \tag{1}$$

where $\mathcal{K}_c(\mathbf{x} - \mathbf{x}_i) \in \mathbb{R}^{C_{in}}$ is the kernel function related to channel c, $\mathbf{f}^l$ is the input features of neighbor point $\mathbf{x}$, and $\langle \cdot, \cdot \rangle$ represents the scalar product of two vectors. We omit the bias term in the equation for brevity, and follow the same convention in the subsequent equations. Here, the kernel function $\mathcal{K}_c$ can be either a continuous function [34, 46] or a set of learnable parameters defined in different volumetric regions [21, 40]. We use range search instead of the commonly used KNN to construct the neighbors because (1) the range neighbors provide consistent metric information in the space and it is robust to density changes [29, 40]; and (2) the range search is more efficient than KNN under brute-force CUDA implementations, especially when the number of desired neighbors is large.

3.1. Fuzzy Kernel Function

To compute the output features in channel c, a fuzzy kernel defines a set of learnable parameters $\mathcal{K}_c = \{\mathbf{w}_{\kappa c} \in \mathbb{R}^{C_{in}}\}_{\kappa=0}^K$ and a criterion to associate each neighbor point to the set elements $\mathbf{w}_{\kappa c}$ differently. Assume that each neighbor $\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)$ of the target convolution point $\mathbf{x}_i$ is associated to a coefficient vector $\boldsymbol{\xi} = \{\xi_\kappa\}_{\kappa=0}^K$. In general, the fuzzy kernel computes the feature f_{ic}^{l+1} of point $\mathbf{x}_i$ as:

$$f_{ic}^{l+1} = \frac{1}{\mathcal{N}(\mathbf{x}_i)} \sum_{\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)} \Big\langle \Big(\sum_{\kappa=0}^K \xi_\kappa \mathbf{w}_{\kappa c} \Big), \mathbf{f}^l \Big\rangle. \tag{2}$$

Without the loss of generality, we constrain the elements of the coefficients to be normalized, i.e. $\sum_{\kappa=0}^K \xi_\kappa = 1$, similar to the Gaussian Mixture Models [4].

In the fuzzy kernel function, each neighboring point makes use of all the kernel parameters to perform the convolution. In fact, the hard kernel can be considered as a special case of the fuzzy kernel where the coefficient vector $\boldsymbol{\xi}$ is simplified to be a one-hot vector. Thus, each neighbor $\mathbf{x}$ uses only a single particular $\mathbf{w}_{\kappa c}$ to compute the feature of the target point:

$$f_{ic}^{l+1} = \frac{1}{\mathcal{N}(\mathbf{x}_i)} \sum_{\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)} \langle \mathbf{w}_{\kappa c}, \mathbf{f}^l \rangle, \tag{3}$$

in which $\kappa = \kappa(\mathbf{x} - \mathbf{x}_i) \in \mathbb{Z}$.

The convolutions in Eq.(1)-(3) follow the definition of typical convolutions in deep learning, and calculate each output feature by conducting the depth-wise and spatial convolutions simultaneously. Inspired by the success of separable convolution [6, 24], we propose to separate the depth-wise and point-wise operations in Eq. (2) and (3) that makes the preformed convolutions more efficient. Since point-wise convolution is independent of neighbors, we apply the discrete kernels to depth-wise convolutions alone. In this situation, the kernel $\mathcal{K}_c$ is reduced to $\mathcal{K}_c = \{w_{\kappa c} \in \mathbb{R}\}_{\kappa=0}^K$. The fuzzy kernel convolution in Eq. (2) and the hard kernel convolution in Eq. (3) are respectively simplified to:

$$f_{ic}^{l+1} = \frac{1}{\mathcal{N}(\mathbf{x}_i)} \sum_{\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)} \Big(\sum_{\kappa=0}^K \xi_\kappa w_{\kappa c} \Big) f_c^l. \tag{4}$$

$$f_{ic}^{l+1} = \frac{1}{\mathcal{N}(\mathbf{x}_i)} \sum_{\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)} w_{\kappa c} f_c^l. \tag{5}$$

By applying such depth-wise convolutions λ times to each input channel, we obtain $C_{out} = \lambda C_{in}$ output features at point $\mathbf{x}_i$. Following this, point-wise convolution is readily achieved with 1×1 convolutions. In this paper, we apply all kernels under the separable convolutional convention which significantly contributes to our network efficiency.

3.2. Hard Spherical Kernel

The spherical kernel proposed by Lei *et al.* [21] divides the local spherical neighborhood into $n \times p \times q$ volumetric bins by partitioning the space uniformly along the azimuth (θ), elevation (ϕ) and radial (r) dimensions. It includes an additional bin for self-convolution of the target point. This results in a spherical convolution kernel of size $n \times p \times q + 1$, which can be represented as $\mathcal{K}_c = \{w_{\kappa c} \in \mathbb{R}\}_{\kappa=0}^{n \times p \times q}$. Here, w_{0c} is the parameter for self-convolution.

For the hard kernel, for each neighboring point $\mathbf{x}$, [21] computes a single index κ based on its coordinates $\boldsymbol{\psi} = (\theta, \phi, r)$ in the local spherical coordinate system centered at $\mathbf{x}_i$. The index value κ is 0 if $\mathbf{x} = \mathbf{x}_i$, otherwise, it is represented as

$$\kappa = k_r \times n \times p + k_\phi \times n + k_\theta + 1, \tag{6}$$

where $k_\theta \in \{0, \dots, n-1\}, k_\phi \in \{0, \dots, p-1\}, k_r \in \{0, \dots, q-1\}$. The constraint of uniform splitting results in the bin range along the three dimensions (θ, ϕ and r) respectively to be $\frac{2\pi}{n}, \frac{\pi}{p}$ and $\frac{\rho}{q}$. Therefore, k_θ, k_ϕ, k_r can be determined as:

$$\begin{cases} k_\theta = \min\left(n-1, \left\lfloor n \times \frac{(\theta+\pi)}{2\pi} \right\rfloor\right), \theta \in [-\pi, \pi], \\ k_\phi = \min\left(p-1, \left\lfloor p \times \frac{(\phi+\frac{\pi}{2})}{\pi} \right\rfloor\right), \phi \in [-\frac{\pi}{2}, \frac{\pi}{2}], \\ k_r = \min\left(q-1, \left\lfloor q \times \frac{r}{\rho} \right\rfloor\right), r \in (0, \rho]. \end{cases} \tag{7}$$

Though efficient, hard kernel suffers from ambiguity and sharp changes at the bin boundaries. For instance, given two points $\mathbf{x}_s, \mathbf{x}_t \in \mathcal{N}(\mathbf{x}_i)$ nearby the xy plane, and $0 < \|\mathbf{x}_s - \mathbf{x}_t\| < \epsilon, \epsilon \to 0^+$. Suppose the two coordinates of the points are exactly the same i.e. $\theta_s = \theta_t$, $r_s = r_t$, but the coordinates $\phi_s = \epsilon$ and $\phi_t = -\epsilon$ are different. This results in their indices k_{ϕ_s} and k_{ϕ_t} to be different, and hence they would use different parameters during the convolution. Another example is when a neighbor point $\mathbf{x}_s$ is very close to the target convolution point $\mathbf{x}_i$, i.e. $0 < \|\mathbf{x}_s - \mathbf{x}_i\| < \epsilon$. This results in the point $\mathbf{x}_s$ to use a certain parameter w_κ (where $\kappa > 0$) in the convolution, which is different from the parameter w_0 used by $\mathbf{x}_i$ for self convolution. To address these issues, we propose a fuzzy spherical as one of the major contributions of this work. We show that the new kernel is both more effective and efficient in practical applications.

3.3. Proposed Fuzzy Spherical Kernel

To avoid the boundary effects in radial direction, we first remove splitting of the sphere along its radius. Therefore, the fuzzy spherical kernel is of size $n \times p + 1$, and the computation of index κ in Eq. (6) gets reduced to $\kappa = k_\phi \times n + k_\theta + 1$. We introduce three new parameters α, β, γ along the elevation and radial dimensions to facilitate the construction of fuzzy coefficients for the kernel. Among them, α and γ are defined as:

$$\begin{cases} \alpha = p \times \frac{(\phi+\frac{\pi}{2})}{\pi}, \\ \gamma = 1 - \frac{\|\mathbf{x}-\mathbf{x}_i\|}{\rho}. \end{cases} \tag{8}$$

Here, γ controls the contribution of each neighbourhood point to the training of self-convolution bin. The closer the point to the target point, the more it contributes to self-convolution. We can then represent the index k_ϕ in Eq. (7) with α as:

$$k_\phi = \min(p-1, \lfloor \alpha \rfloor). \tag{9}$$

To this end, the parameter β becomes

$$\beta = 1 - |\alpha - k_\phi - 0.5|. \tag{10}$$

The β controls the contribution of each neighbourhood point to the training of the elevation bin it is located in. As can be derived, when the point is in the elevation bin center, β will be 1, while when the point is exactly on the boundary of two bins, β decreases to 0.5. The other bin that can cause boundary effects for point $\mathbf{x}$ in the elevation dimension is $\tilde{k}_\phi$, which can be computed as

$$\tilde{k}_\phi = \begin{cases} \max(0, \quad k_\phi - 1), \text{ if } (\alpha - k_\phi) \leq 0.5. \\ \min(p-1, k_\phi + 1), \text{ if } (\alpha - k_\phi) > 0.5. \end{cases} \tag{11}$$

The fuzzy coefficient vector $\boldsymbol{\xi}$ contains at most three non-zero values, which are

$$\xi_0 = \gamma, \ \xi_{\kappa'} = (1-\gamma)\beta, \ \xi_{\tilde{\kappa}'} = (1-\gamma)(1-\beta).$$

where $\kappa' = k_\phi \times n + k_\theta + 1$, $\tilde{\kappa}' = \tilde{k}_\phi \times n + k_\theta + 1$. It can be shown that $\boldsymbol{\xi}$ satisfies the normalization constraint:

$$\begin{aligned} \sum_{\kappa=0}^{n \times p} \xi_\kappa &= \xi_0 + \xi_{\kappa'} + \xi_{\tilde{\kappa}'} \\ &= \gamma + (1-\gamma)\beta + (1-\gamma)(1-\beta) = 1. \end{aligned} \tag{12}$$

Eventually, following Eq. (4), the fuzzy spherical kernel computes the feature of the target point $\mathbf{x}_i$ as:

$$f_{ic}^{l+1} = \frac{1}{\mathcal{N}(\mathbf{x}_i)} \sum_{\mathbf{x} \in \mathcal{N}(\mathbf{x}_i)} (\xi_0 w_{0c} + \xi_{\kappa'} w_{\kappa' c} + \xi_{\tilde{\kappa}'} w_{\tilde{\kappa}' c}) f_c^l. \tag{13}$$

In the proposed fuzzy kernel, all the neighboring points contribute to the training of self-convolution parameter w_0 based on their distances to the target point $\mathbf{x}_i$. This is in contrast to the original spherical kernel [21] that optimizes w_0 only based on the target convolution point. In the elevation direction, the parameter that each neighboring point uses in the convolution becomes a weighted combination of the parameters from its two nearest bins. Such a weighted combination alleviates the unreasonably sharp changes of convolutional parameters between the bin boundaries. Overall,

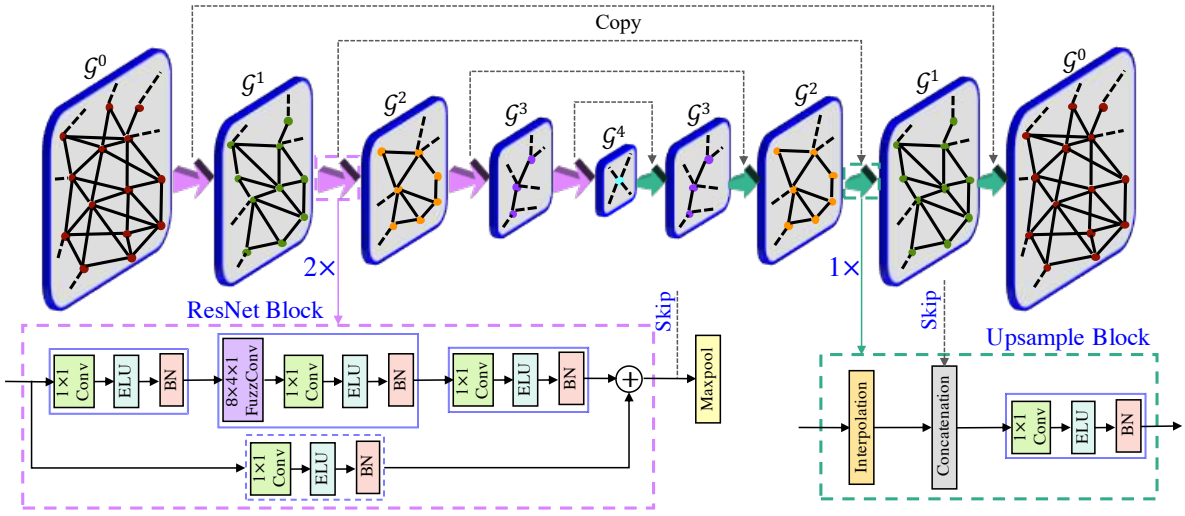


Figure 2. Proposed encoder-decoder graph convolutional network SegGCN. A graph $\mathcal{G}^0$ gets sequentially coarsened to $\mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3, \mathcal{G}^4$ and back again to $\mathcal{G}^0$. During encoding $2\times$ ResNet blocks with fuzzy convolutions are used between consecutive graphs. We omit self loops for clarity. During decoding, 1×1 convolutions are used for efficiency. Skip connections are used to copy features from the encoder that are concatenated with the features at the decoder of the similar graph stage. We use Batch Normalization (BN) after ELU activations.

the proposed fuzzy spherical kernel enables the convolutional parameters used by each neighbor point to change in a smooth and consistent way.

We do not incorporate the fuzziness in azimuth dimension because commonly performed arbitrary horizontal rotations for training data augmentation already resolves the ambiguity in this dimension. Hence, applying the fuzzy mechanism only in the radial and elevation directions keeps our kernel more efficient. This can be verified by comparing the fuzzy convolution computation in Eq. (4) to its hard counterpart in Eq. (5).

From Eq. (13), we can see that the proposed fuzzy spherical kernel requires only 3 more multiplications and 2 more additions in the convolution with each neighbor point, which is far less than the computations required by a dense vector $\boldsymbol{\xi}$. The sparseness of our fuzzy coefficient vector makes the spherical kernel more efficient without significantly affecting its effectiveness. We provide complexity analysis for the backward propagation of convolution in the supplementary material.

Comparison to KPConv: In contrast to the proposed kernel, the kernel used in KPConv [40] must generate a set of template points offline using optimization. The performance of that network is highly dependent on the kernel points returned by the offline training, which is sub-optimal. In contrast, the fuzzy spherical kernel divides the space occupied by the neighbors in a deterministic and compact manner. In § 4, we empirically demonstrate that the proposed fuzzy spherical kernel outperforms KPConv under similar settings.

3.4. Network Architecture

We propose an encoder-decoder graph convolutional network for point cloud segmentation. The network is composed of ResNet blocks in the encoder part and 1×1 convolutions in the decoder part. Figure 2 shows the architecture of our network. To construct graph representations from a point cloud $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with feature maps $\mathcal{F} = \{\mathbf{f}_1, \ldots, \mathbf{f}_N\}$, we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V} = \{1, 2, \ldots, N\}$ and $\mathcal{E} \subseteq |\mathcal{V}| \times |\mathcal{V}|$ respectively represent the sets of graph vertices and edges. Each vertex $i \in \mathcal{V}$ is associated with a point location $\mathbf{x}_i$ and its corresponding feature map $\mathbf{f}_i$. We derive the edge set $\mathcal{E}$ from the point neighbors.

Graph connection and coarsening: We use range search to get the spatial neighbors of each point within a specified radius ρ. We implement range search efficiently using CUDA programming to exploit GPU parallel computations. For practical reasons, we constrain the maximum number of neighbor connections to M. Random sampling is applied if the range search returns more than M points in the neighborhood. To coarsen the graph hierarchically into different resolutions, we exploit the point cloud subsampling method, Farthest Point Sampling (FPS) [29]. Compared to voxel-grid sampling [34, 40], FPS has the advantage of keeping the number of vertices/points fixed across different samples, which is helpful when applying standard batch normalization [13]. For efficiency, we avoid subsampling strategies that require separate training [9]. By alternating between connecting vertices and graph coarsening L times, we construct a pyramid of $L + 1$ graphs $\mathcal{G}^0 \rightarrow \mathcal{G}^1 \rightarrow \cdots \rightarrow \mathcal{G}^L$ with resolutions from fine to coarse.

Pooling and Unpooling: In the encoder, we use max-

pooling to compute vertex features for the coarsened graphs. This process defines the features of each vertex in graph $\mathcal{G}^{l+1}$ as the max-pooled features of their neighbors in graph $\mathcal{G}^{l}$. In the decoder part, we compute vertex features for the graph with higher resolution with unpooling operation, which is performed using weighted interpolation. In particular, the interpolation method upsamples the vertex features in graph $\mathcal{G}^{l}$ as a weighted sum of its neighborhood features in graph $\mathcal{G}^{l+1}$. The definition of our weights is similar to PointNet++ [29].

We construct a segmentation network, SegGCN, using the above techniques. SegGCN comprises a pyramid of five graph resolutions. To learn features at each level, we use two ResNet blocks with (separable) fuzzy spherical convolution in the encoder. Pooling operations are used to extract features when the graph structure alters. SegGCN also exploits the effective skip connections [3, 5, 31] by copying features from the encoder and concatenating them to the decoder features at a similar level in the graph hierarchy. The concatenated features at the decoder stage are processed with a 1×1 convolution, which is empirically shown to be more efficient and effective in § 4 than the ResNet blocks used for the encoding purposes. We provide illustration of our SegGCN including details of the ResNet block as well as the upsampling block in Fig. 2. With this paper, we also release CUDA implementation of the fuzzy spherical kernel and the efficient SegGCN architecture. The code is Tensorflow compatible [1], and can be found at this Github Link.

4. Experiments

We evaluate our SegGCN on the challenging semantic segmentation datasets S3DIS [2] and ScanNet [7] which consists of point clouds of large-scale indoor scenes. Color information are provided with (r, g, b) values on both datasets. We train our network with 6-dimensional input features (x, y, z, r, g, b). Before using the raw color values, we rescale them into the range $[-1, 1]$.

Our network is trained on a single GeForce RTX 2080 Ti GPU with Adam Optimizer [14]. For training, we set the initial learning rate to 0.001 with momentum 0.9. Throughout the experiments, we apply the fuzzy spherical convolution with a kernel size $8 \times 4 \times 1 + 1$. Our network takes point clouds of size $8,192$ as inputs, while the batch size are kept fixed to 16. This results in the total size of points processed by the network with each batch as 131K. We allow the maximum neighbor connections of vertices in all graphs to be $M = 64$. These hyper-parameters are empirically determined based on cross-validations. We exploit common data augmentations in the related literature in our experiments, including random scaling, shifting, noisy translation, random azimuth rotation (up to 360° degrees) and arbitrary rotation perturbations (up to 10° degrees). We apply these augmentations on-the-fly in the network training session. Point cloud flipping and color augmentation is not applied.

Network Configuration: We use identical network configurations for ScanNet and S3DIS. In specific, the vertex sizes of graphs $\mathcal{G}^0$, $\mathcal{G}^1$, $\mathcal{G}^2$, $\mathcal{G}^3$, $\mathcal{G}^4$ are 8192, 2048, 768, 384, 128 respectively. We construct the graph edges of $\mathcal{G}^0, \mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3$ using an increasing range search radius 0.1, 0.2, 0.4, 0.8. The output feature sizes of the three convolutions in a ResNet block can be represented as $(D, D, 4 \times D)$ [12]. We set the hyper-parameter D of our ResNet blocks of $\mathcal{G}^0, \mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3$ as $32, 64, 128, 256$. The fuzzy convolution in the block uses a constant multiplier $\lambda = 1$. In the decoder, we explore only the nearest three points for feature interpolation. The output feature sizes of the 1×1 convolutions in the upsampling blocks are half of their corresponding encoder features. Instead of feeding the raw input features to the graph architecture directly, we insert one shared MLP in-between to increase the point cloud feature size from 6 to 64. SegGCN classifies the feature representation obtained from $\mathcal{G}^0$ in the decoder directly without using further fully connected layers.

4.1. S3DIS

The Stanford large-scale 3D Indoor Spaces (S3DIS) dataset [2] is collected from three different buildings on the Standard campus using Matterport scanner. It is composed of colored 3D point clouds of 6 indoor areas. The task defined on this dataset is about labeling 13 semantic elements, which are *ceiling, floor, wall, beam, column, window, door, table, chair, sofa, bookcase, board*, and *clutter*. Any element that is not among the 12 well-defined classes, is considered *clutter*. Because Area 5 is related to a building not covered by the other areas [39], we perform experiments by using Area 5 as the test set, which is also the convention of the previous works [19, 24, 28, 39, 42]. The used evaluation metrics include the Overall Accuracy (OA), average Accuracy of all 13 categories (mAcc), per-category Intersection Over Union (IoU), and their average (i.e. mIoU). mIoU is considered the most reliable metric among these.

Due to the millions of points in each indoor scene, we firstly sub-sample the scene cloud using the VoxelGrid algorithm [32] with a grid size $3cm$. We then split each scene into overlapped blocks of size $1.5m \times 1.5m$. Similar to the PointCNN [24], we apply such splitting to the x, y dimensions but not to the height z dimension. The goal is to keep the height information complete. Our (x, y, z) coordinates in the input features are normalized by aligning them to the ground plane center of their corresponding blocks. The experiment results are provided in Table 1. It can be noticed that SegGCN performs better than other competitive convolutional networks, including SSP+SPG [18] and GACNet [42] whose performance pre-dominantly benefit from local labeling consistency. The size of learnable parameters of SegGCN is 3.0M. Its inference time for batch sizes of 16

Table 1. Performance on the fifth fold (Area 5) of S3DIS dataset. SSP+SPG and GACNet use the constraint of local labeling consistency while the proposed network does not.

	Methods	OA	mAcc	mIoU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
Area 5	PointNet [28]	-	49.0	41.1	88.8	97.3	69.8	0.1	3.9	46.3	10.8	58.9	52.6	5.9	40.3	26.4	33.2
	SEGCloud [39]	-	57.4	48.9	90.1	96.1	69.9	0.0	18.4	38.4	23.1	70.4	75.9	40.9	58.4	13.0	41.6
	Tangent-Conv [38]	82.5	62.2	52.8	-	-	-	-	-	-	-	-	-	-	-	-	-
	SPG [19]	86.4	66.5	58.0	89.4	96.9	78.1	0.0	42.8	48.9	61.6	75.4	84.7	52.6	69.8	2.1	52.2
	PointCNN [24]	85.9	63.9	57.3	92.3	98.2	79.4	0.0	17.6	22.8	62.1	74.4	80.6	31.7	66.7	62.1	56.7
	SSP+SPG [18]	87.9	68.2	61.7	-	-	-	-	-	-	-	-	-	-	-	-	-
	GACNet [41]	87.8	-	62.9	92.3	98.3	81.9	0.0	20.4	59.1	40.9	78.5	85.8	61.7	70.8	74.7	52.8
	SPH3D-GCN [22]	87.7	65.9	59.5	93.3	97.1	81.1	0.0	33.2	45.8	43.8	79.7	86.9	33.2	71.5	54.1	53.7
	KPConv [40]	-	70.9	65.4	92.6	97.3	81.4	0.0	16.5	54.5	69.5	90.1	80.2	74.6	66.4	63.7	58.1
	SegGCN (Prop.)	88.2	70.4	63.6	93.7	98.6	80.6	0.0	28.5	42.6	74.5	80.9	88.7	69.0	71.3	44.4	54.3

Table 2. 3D semantic labeling on ScanNet: All the networks use spatial coordinates and color values as input features.

Method	mIoU	floor	wall	chair	sofa	table	door	cab	bed	desk	toil	sink	wind	pic	bkshf	curt	show	cntr	fridg	bath	other
ScanNet [7]	30.6	78.6	43.7	52.4	34.8	30.0	18.9	31.1	36.6	34.2	46.0	31.8	18.2	10.2	50.1	0.2	15.2	21.1	24.5	20.3	14.5
PointNet++ [29]	33.9	67.7	52.3	36.0	34.6	23.2	26.1	25.6	47.8	27.8	54.8	36.4	25.2	11.7	45.8	24.7	14.5	25.0	21.2	58.4	18.3
SPLATNET_{3D} [36]	39.3	92.7	69.9	65.6	51.0	38.3	19.7	31.1	51.1	32.8	59.3	27.1	26.7	0.0	60.6	40.5	24.9	24.5	0.1	47.2	22.7
Tangent-Conv [38]	43.8	91.8	63.3	64.5	56.2	42.7	27.9	36.9	64.6	28.2	61.9	48.7	35.2	14.7	47.4	25.8	29.4	35.3	28.3	43.7	29.8
PointCNN [24]	45.8	94.4	70.9	71.5	54.5	45.6	31.9	32.1	61.1	32.8	75.5	48.4	47.5	16.4	35.6	37.6	22.9	29.9	21.6	57.7	28.5
PointConv [44]	55.6	94.4	76.2	73.9	63.9	50.5	44.5	47.2	64.0	41.8	82.7	54.0	51.5	18.5	57.4	43.3	57.5	43.0	46.4	63.6	37.2
SPH3D-GCN [22]	61.0	93.5	77.3	79.2	70.5	54.9	50.7	53.2	77.2	57.0	85.9	60.2	53.4	4.6	48.9	64.3	70.2	40.4	51.0	85.8	41.4
KPConv [40] †	68.4	93.5	81.9	81.4	78.5	61.4	59.4	64.7	75.8	60.5	88.2	69.0	63.2	18.1	78.4	77.2	80.5	47.3	58.7	84.7	45.0
SparseConvNet [25]	72.5	95.5	86.5	86.9	82.3	62.8	61.4	72.1	82.1	60.3	93.4	72.4	68.3	32.5	84.6	75.4	87.0	53.3	71.0	64.7	57.2
SegGCN (Prop.)	58.9	93.6	77.1	78.9	70.0	56.3	48.4	51.4	73.1	57.3	87.4	59.4	49.3	6.1	53.9	46.7	50.7	44.8	50.1	83.3	39.6

†KPConv [40] takes 9.3 seconds to segment 90K points in the inference stage using the same hardware, which is 100× slower than SegGCN.

and 32 is 150 and 250 milliseconds respectively. On a single RTX 2080 Ti GPU, SegGCN can process over a million points per-second. We show visualizations of representative segmentations generated by SegGCN in Fig. 3.

4.2. ScanNet

ScanNet [7] is an RGB-D video dataset collected from indoor environments. It contains a diversity of reconstructed rooms/offices with rich annotations for 3D semantic labeling L. The dataset provides labels for 40 common classes, while only 20 of them are used for performance evaluation, resulting in 21 classes to be labelled with the network. The training/validation/test set includes $1201, 312, 100$ scenes respectively, while the ground-truth labels of test set are not public. Researchers have to submit their test results online for the standard evaluation. We apply VoxelGrid subsampling and splitting strategies identical to S3DIS to prepare the block data. Table 1 summarizes the result of our experiments with this data. All the techniques in the table use the spatial coordinates and color values as input features. SegGCN outperforms the other competitive convolutional approaches on 16 out of 20 categories, resulting in a significant overall improvement in the mIoU. Our training/inference time with batch size 16 is 340/160 ms.

4.3. Discussion

4.3.1 Kernel comparison

In Table 3, we compare the performance of the proposed fuzzy spherical kernel to the hard spherical kernel of Lei et al. [21]. We also compare with the fuzzy KPConv kernel [40], which makes fuzzy assignments based on the distances between the neighbors and the template kernel points. Moreover, we also create a hard KPConv kernel that associates each neighbor point only to its closest template point (with coefficient 1). We calculate the template kernel points of KPConv based on the github code provided by the authors[1]. We apply all kernels under separable convolution configuration. The network used is *ours*, i.e. SegGCN. The training/test data as well as the hyper-parameters settings of the network are exactly the same for all experiments. The only difference is the discrete kernels used in the ResNet block. The original KPConv subsamples point clouds with grid-based sampling, and has to play implementation tricks for batch normalization because the point cloud size varies for different samples.

Table 3. Performance of different kernels on S3DIS. We use SPH as abbreviations of the spherical kernel. Hard and fuzzy SPH kernels are compared. We also compare with original fuzzy KPConv kernel to its hard counterpart implemented in this work. Proposed SegGCN is used as the network.

kernel	hard SPH	fuzzy SPH	hard KPConv	fuzzy KPConv
OA	88.0	**88.2**	86.9	87.1
mAcc	69.6	**70.4**	68.7	68.6
mIoU	62.9	**63.6**	60.5	61.0

The training/inference time for hard spherical kernel and fuzzy spherical kernel are 251/133 ms versus 314/153, where the fuzzy kernel performs slightly slower than the hard spherical kernel. However, it results in a significant performance gain at acceptable computational complexity.

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

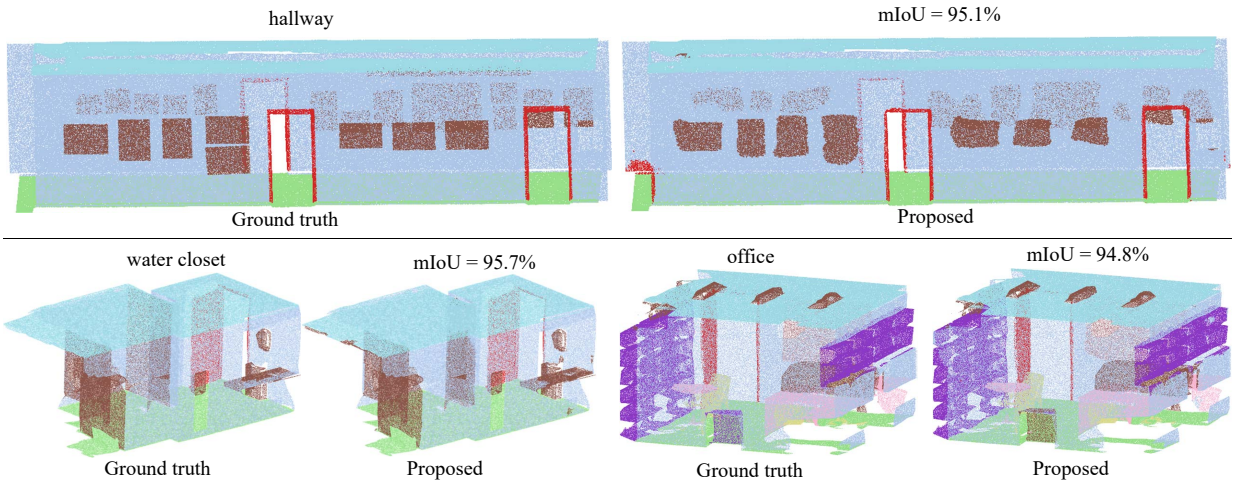


Figure 3. Visualization of representative segmentations by the proposed SegGCN on the S3DIS dataset. (Top) Segmentation result of a hallway of 13 meters. (Bottom) Results of two separate rooms. On left is a common water closet. On right is an office. For a diverse range of scenes (simple to complex), SegGCN is able to segment the point cloud semantics effectively.

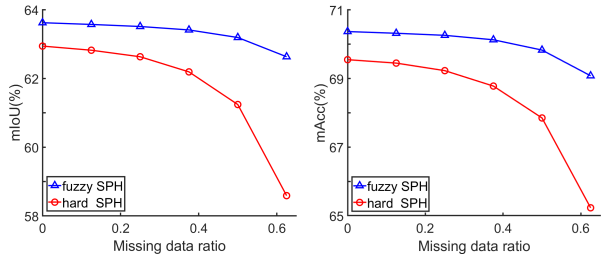


Figure 4. Robustness of fuzzy kernel vs. hard kernel to missing data. SegGCNs trained on $8,129$ points with fuzzy SPH kernel and hard SPH kernel [21] are tested by dropping a certain data percentage. The proposed fuzzy SPH kernel shows a clear relative stability over its hard counterpart over a considerable range of missing data. For the six different input sizes we test, the standard deviations of mIoU/mAcc metrics of the fuzzy kernel are 0.37/0.49, over 3 times better than 1.66/1.65 of the hard kernel.

We elaborate on further advantages of the fuzzy mechanism in the following section.

4.3.2 Robustness to point density (missing data)

Although the fuzzy kernel performs better than the hard kernel on mIoU, its real benefit comes in dealing with missing data or lower point density. To demonstrate that, we let two SegGCN networks use fuzzy and hard spherical kernels and train them on 8192 points. We test the robustness of those networks by inducing sparsity in the input. We vary the input sizes to 8192, 7168, 6144, 5120, 4096, 3072, corresponding to the data drop ratio of 0, 0.125, 0.25, 0.375, 0.5, 0.625, respectively. We show with both mIoU and mAcc metrics the performance of both kernels for these cases in Fig. 4. The results conclusively demonstrate the stability of our fuzzy kernel in the face of missing data, which is a major advantage of fuzzy kernel over hard kernels. We note that, in our technique this advantage comes in addition to the highly competitive performance on dense clouds, and the ability to efficiently process large point clouds.

Table 4. Performance comparison between the proposed 1×1 convolution and the choice dictated by the common practice of encoder-based $2\times$ ResNet blocks in the decoder. Despite the simplicity, the proposed 1×1 convolution is much more effective.

Decoder convolution	#params	mAcc	mIoU	time(ms)	
				train	infer
single 1×1 Conv	**3.0M**	**70.4**	**63.6**	**314**	**153**
$2\times$ ResNet blocks	5.6M	69.8	62.9	385	179

4.3.3 Decoder module choice

To justify our choice of 1×1 convolutions in the decoder part of SegGCN, as compared to the more popular choice of replicating similar encoder components, we perform an additional experiment. We replace all single 1×1 convolution in SegGCN with $2\times$ ResNet blocks similar to the encoder. Table 4 compares the results of the two networks. The proposed SegGCN is clearly more effective and efficient than its counterpart which is based on the more common choice of the decoder.

5. Conclusion

We introduce a fuzzy mechanism in spherical convolutional kernel to process 3D point clouds. We additionally propose an efficient Graph convolutional network, SegGCN for semantic segmentation task. The proposed fuzzy kernel demonstrates robustness against adverse boundary conditions by removing the weight assignment of the conventional discrete spherical kernel. It is also shown to be naturally robust against missing data. Our network exploits ResNet-like blocks in the encoder and 1×1 convolutions in the decoder. Experiments with S3DIS and ScanNet datasets demonstrate that our network can process over 1M points per second, achieving highly competitive performance.

Acknowledgments This research was supported by ARC Discovery Grant DP190102443. We thank NVIDIA Corporation for the GPU donation.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016. 2, 6

[2] 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. 6

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 6

[4] Christopher M Bishop. *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006. 1, 3

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision*, 2018. 6

[6] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017. 2, 3

[7] 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. 6, 7

[8] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, 2016. 2

[9] Oren Dovrat, Itai Lang, and Shai Avidan. Learning to sample. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2760–2769, 2019. 5

[10] M. Engelcke, D. Rao, D. Zeng Wang, C. Hay Tong, and I. Posner. Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. In *IEEE International Conference on Robotics and Automation*, June 2017. 2

[11] Fabian Groh, Patrick Wieschollek, and Hendrik PA Lensch. Flex-convolution. In *Asian Conference on Computer Vision*, pages 105–122. Springer, 2018. 2

[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. 1, 2, 6

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015. 5

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015. 6

[15] 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*, pages 863–872. IEEE, 2017. 2

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. 1

[17] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, page 282–289, 2001. 3

[18] Loic Landrieu and Mohamed Boussaha. Point cloud oversegmentation with graph-structured deep metric learning. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019. 6, 7

[19] 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*, 2018. 6, 7

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[21] Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9631–9640, 2019. 1, 2, 3, 4, 7, 8

[22] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical kernel for efficient graph convolution on 3d point clouds. *arXiv preprint arXiv:1909.09287*, 2019. 3, 7

[23] 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. 2

[24] 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. 1, 2, 3, 6, 7

[25] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015. 7

[26] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 922–928. IEEE, 2015. 2

[27] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012. 2

[28] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 2, 6, 7

[29] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in

a metric space. *Advances in Neural Information Processing Systems*, 2017. 2, 3, 5, 6, 7

[30] 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*, pages 3577–3586, 2017. 2

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015. 6

[32] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation*, pages 1–4, 2011. 6

[33] 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. 1, 2

[34] 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*, 2017. 1, 2, 3, 5

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015. 1

[36] 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. 2, 7

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015. 1

[38] 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. 2, 7

[39] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *International Conference on 3D Vision*, pages 537–547. IEEE, 2017. 6, 7

[40] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 1, 3, 5, 7

[41] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2019. 2, 3, 7

[42] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019. 6

[43] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 1

[44] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 1, 2, 3, 7

[45] 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. 2

[46] 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*, pages 87–102, 2018. 2, 3

[47] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017. 2