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 $1 \times 1$ 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 in 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.

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

![Figure 1](image_url)

(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.
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 \( 1 \times 1 \) 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. \( 30 \times 30 \times 30 \) [45], \( 32 \times 32 \times 32 \) [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 \( 256 \times 256 \times 256 \) 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 multi-layer 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 alternative approach 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 \( \mathcal{P} = \{x_i \in \mathbb{R}^3\}_{i=1}^{N} \) be a point cloud and its feature maps be \( \mathcal{F} = \{f_i^l \in \mathbb{R}^{C_{in}}\}_{i=1}^{N} \). To perform convolution at a target point \( x_i \), we first construct its neighborhood set as \( \mathcal{N}(x_i) = \{x \in \mathcal{P} : ||x - x_i|| \leq \rho\} \). With the neighborhood set \( \mathcal{N}(x_i) \), the general convolution kernel calculates the output feature of \( x_i \) in channel \( c \) as:

\[
f_{ic}^{l+1} = \frac{1}{\mathcal{N}(x_i)} \sum_{x \in \mathcal{N}(x_i)} \langle \kappa_c(x - x_i), f^l \rangle, \tag{1}
\]

where \( \kappa_c(x - x_i) \in \mathbb{R}^{C_{in}} \) is the kernel function related to channel \( c \), \( f^l \) is the input features of neighbor point \( x \), 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 \( \kappa_c \) 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 \( \kappa_c = \{w_{kc} \in \mathbb{R}^{C_{in}}\}_{k=0}^{K} \) and a criterion to associate each neighbor point to the set elements \( w_{kc} \) differently. Assume that each neighbor \( x \in \mathcal{N}(x_i) \) of the target convolution point \( x_i \) is associated to a coefficient vector \( \xi = \{\xi_k\}_{k=0}^{K} \). In general, the fuzzy kernel computes the feature \( f_{ic}^{l+1} \) of point \( x_i \) as:

\[
f_{ic}^{l+1} = \frac{1}{\mathcal{N}(x_i)} \sum_{x \in \mathcal{N}(x_i)} \left\langle \left( \sum_{k=0}^{K} \xi_ka_{xc}w_{kc} \right), f^l \right\rangle. \tag{2}
\]

Without the loss of generality, we constrain the elements of the coefficients to be normalized, i.e. \( \sum_{k=0}^{K} \xi_k = 1 \), 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 \( \xi \) is simplified to be a one-hot vector. Thus, each neighbor \( x \) uses only a single particular \( w_{kc} \) to compute the feature of the target point:

\[
f_{ic}^{l+1} = \frac{1}{\mathcal{N}(x_i)} \sum_{x \in \mathcal{N}(x_i)} \langle w_{kc}, f^l \rangle, \tag{3}
\]

in which \( \kappa = \kappa(x - x_i) \in \mathbb{Z} \).

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 \( \kappa_c \) is reduced to \( \kappa_c = \{w_{kc} \in \mathbb{R}^{C_{in}}\}_{k=0}^{K} \). The fuzzy kernel convolution in Eq. (2) and the hard kernel convolution in Eq. (3) are respectively simplified to:

\[
f_{ic}^{l+1} = \frac{1}{\mathcal{N}(x_i)} \sum_{x \in \mathcal{N}(x_i)} \left( \sum_{k=0}^{K} \xi_kw_{kc} \right)f_{ic}^l, \tag{4}
\]

\[
f_{ic}^{l+1} = \frac{1}{\mathcal{N}(x_i)} \sum_{x \in \mathcal{N}(x_i)} w_{kc}f_{ic}^l. \tag{5}
\]

By applying such depth-wise convolutions \( \lambda \) times to each input channel, we obtain \( \mathcal{C}_{out} = \lambda \mathcal{C}_{in} \) output features at point \( x_i \). Following this, point-wise convolution is readily achieved with \( 1 \times 1 \) 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 (\( \theta \)), elevation (\( \phi \)) 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 \( \mathcal{K}_s = \{ w_{k\theta} \in \mathbb{R} \}^{n \times p \times q}_{k\theta=0} \). Here, \( w_{00} \) is the parameter for self-convolution.

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

\[
\kappa = k_0 \times n + k_\phi \times n + k_\theta + 1,
\]

where \( k_\theta \in \{0, \ldots, n-1\}, k_\phi \in \{0, \ldots, p-1\}, k_\theta \in \{0, \ldots, q-1\} \). The constraint of uniform splitting results in the point \( x \) belonging to the local spherical neighborhood into \( \mathcal{K}_s \times \mathcal{K}_p \times \mathcal{K}_q \). Therefore, \( k_\theta, k_\phi, k_\theta \) can be determined as:

\[
\begin{align*}
k_\theta &= \min \left( n - 1, \left\lfloor \frac{(\theta + \pi)}{2\pi} \right\rfloor \right), \theta \in [-\pi, \pi], \\
k_\phi &= \min \left( p - 1, \left\lfloor \frac{(\phi + \pi)}{2\pi} \right\rfloor \right), \phi \in [-\pi, \pi], \\
k_\theta &= \min \left( q - 1, \left\lfloor \frac{r}{\rho} \right\rfloor \right), r \in (0, \rho].
\end{align*}
\]

(7)

Though efficient, hard kernel suffers from ambiguity and sharp changes at the bin boundaries. For instance, given two points \( x_s, x_t \in \mathcal{N}(x_i) \) nearly the \( xy \) plane, and \( 0 < \|x_s - x_t\| < \epsilon, \epsilon \to 0^+ \). 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 \( k_\phi \) and \( k_\phi \) to be different, and hence they would use different parameters during the convolution. Another example is when a neighbor point \( x_s \) is very close to the target convolution point \( x_t \), i.e. \( 0 < \|x_s - x_t\| < \epsilon \). This results in the point \( x_t \) using a different parameter \( w_{k\theta} \) (where \( k > 0 \)) in the convolution, which is different from the parameter \( w_{00} \) used by \( x_i \) 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 \( \kappa \) in Eq. (6) gets reduced to \( \kappa = k_\phi \times n + k_\theta + 1 \). We introduce three new parameters \( \alpha, \beta, \gamma \) along the elevation and radial dimensions to facilitate the construction of fuzzy coefficients for the kernel. Among them, \( \alpha \) and \( \gamma \) are defined as:

\[
\begin{align*}
\alpha &= p \times \frac{(\phi + \pi)}{2\pi}, \\
\gamma &= 1 - \frac{\|x - x_i\|}{\rho}.
\end{align*}
\]

(8)

Here, \( \gamma \) 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 \( k_\phi \) in Eq. (7) with \( \alpha \) as:

\[
k_\phi = \min(p - 1, \lfloor\alpha\rfloor).
\]

(9)

To this end, the parameter \( \beta \) becomes

\[
\beta = 1 - \lceil\alpha - k_\phi - 0.5\rceil.
\]

(10)

The \( \beta \) 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, \( \beta \) will be 1, while when the point is exactly on the boundary of two bins, \( \beta \) decreases to 0.5. The other bin that can cause boundary effects for point \( x \) in the elevation dimension is \( k_\phi, \) which can be computed as

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

(11)

The fuzzy coefficient vector \( \xi \) contains at most three non-zero values, which are

\[
\xi_0 = \gamma, \xi_0 = (1 - \gamma)\beta, \xi_0 = (1 - \gamma)(1 - \beta).
\]

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

\[
\sum_{k=0}^{n \times p} \xi_k = \xi_0 + \xi_{k'} + \xi_{k''}
\]

(12)

Eventually, following Eq. (4), the fuzzy spherical kernel computes the feature of the target point \( x_i \) as:

\[
f^{l+1}(x_i) = \frac{1}{N(x_i)} \sum_{x_j \in \mathcal{N}(x_i)} (\xi_0 w_{00} + \xi_{k'} w_{k'c} + \xi_{k''} w_{k''c}) f_c^{l}.
\]

(13)

In the proposed fuzzy kernel, all the neighboring points contribute to the training of self-convolution parameter \( w_{00} \) based on their distances to the target point \( x_i \). This is in contrast to the original spherical kernel [21] that optimizes \( w_{00} \) 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,
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 $\xi$. 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 $1 \times 1$ convolutions in the decoder part. Figure 2 shows the architecture of our network. To construct graph representations from a point cloud $P = \{x_1, \ldots, x_N\}$ with feature maps $F = \{f_1, \ldots, f_N\}$, we consider a graph $G = (V, E)$, in which $V = \{1, 2, \ldots, N\}$ and $E \subseteq |V| \times |V|$ respectively represent the sets of graph vertices and edges. Each vertex $i \in V$ is associated with a point location $x_i$ and its corresponding feature map $f_i$. We derive the edge set $E$ from the point neighbors.

**Graph connection and coarsening**: We use range search to get the spatial neighbors of each point within a specified radius $\rho$. 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 $G^0 \rightarrow G^1 \rightarrow \ldots \rightarrow G^L$ 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 $G^{l+1}$ as the max-pooled features of their neighbors in graph $G^l$. 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 $G^l$ as a weighted sum of its neighborhood features in graph $G^{l+1}$. 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 $1 \times 1$ 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 is 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 $360^\circ$ degrees) and arbitrary rotation perturbations (up to $10^\circ$ 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 $G^0, G^1, G^2, G^3$ are 8192, 2048, 768, 384, 128, respectively. We construct the graph edges of $G^0, G^1, G^2, G^3$ 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 $G^0, G^1, G^2, G^3$ 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 $1 \times 1$ 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 $G^0$ 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
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 best results are marked with †.

<table>
<thead>
<tr>
<th>Methods</th>
<th>OA</th>
<th>mAcc</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet [28]</td>
<td>-</td>
<td>49.0</td>
<td>41.1</td>
</tr>
<tr>
<td>SegCloud [39]</td>
<td>-</td>
<td>57.4</td>
<td>48.9</td>
</tr>
<tr>
<td>Tangent-Conv [38]</td>
<td>82.5</td>
<td>62.2</td>
<td>52.8</td>
</tr>
<tr>
<td>SPG [19]</td>
<td>86.4</td>
<td>66.5</td>
<td>58.0</td>
</tr>
<tr>
<td>PointCNN [24]</td>
<td>85.9</td>
<td>63.9</td>
<td>57.3</td>
</tr>
<tr>
<td>SSP+SPG [8]</td>
<td>87.9</td>
<td>68.2</td>
<td>61.7</td>
</tr>
<tr>
<td>GACNet [41]</td>
<td>87.8</td>
<td>62.9</td>
<td>59.6</td>
</tr>
<tr>
<td>SPH3D-GCN [22]</td>
<td>87.7</td>
<td>65.9</td>
<td>59.5</td>
</tr>
<tr>
<td>KPConv [40]</td>
<td>-</td>
<td>70.9</td>
<td>65.4</td>
</tr>
<tr>
<td>SegGCN (Prop.)</td>
<td>88.2</td>
<td>70.4</td>
<td>63.6</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Methods</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScanNet [7]</td>
<td>30.6</td>
</tr>
<tr>
<td>PointNet++ [29]</td>
<td>33.9</td>
</tr>
<tr>
<td>SPLATNetG [36]</td>
<td>39.3</td>
</tr>
<tr>
<td>Tangent-Conv [38]</td>
<td>43.8</td>
</tr>
<tr>
<td>PointCNN [24]</td>
<td>45.8</td>
</tr>
<tr>
<td>PointConv [44]</td>
<td>55.6</td>
</tr>
<tr>
<td>SPH3D-GCN [22]</td>
<td>61.0</td>
</tr>
<tr>
<td>KPConv [40]†</td>
<td>68.4</td>
</tr>
<tr>
<td>SparseConvNet [25]</td>
<td>72.5</td>
</tr>
<tr>
<td>SegGCN (Prop.)</td>
<td>58.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScanNet [7]</td>
<td>78.6</td>
</tr>
<tr>
<td>PointNet++ [29]</td>
<td>67.7</td>
</tr>
<tr>
<td>SPLATNetG [36]</td>
<td>92.7</td>
</tr>
<tr>
<td>Tangent-Conv [38]</td>
<td>91.8</td>
</tr>
<tr>
<td>PointCNN [24]</td>
<td>94.4</td>
</tr>
<tr>
<td>PointConv [44]</td>
<td>94.4</td>
</tr>
<tr>
<td>SPH3D-GCN [22]</td>
<td>93.5</td>
</tr>
<tr>
<td>KPConv [40]†</td>
<td>93.5</td>
</tr>
<tr>
<td>SparseConvNet [25]</td>
<td>95.5</td>
</tr>
<tr>
<td>SegGCN (Prop.)</td>
<td>93.6</td>
</tr>
</tbody>
</table>

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

### 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. 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 size varies for different samples.

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.

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 [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. 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 size varies for different samples.

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.

#### 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. 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 size varies for different samples.

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.

#### 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. 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 size varies for different samples.

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.

#### 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. 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 size varies for different samples.

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.

### 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.

<table>
<thead>
<tr>
<th>Methods</th>
<th>OA</th>
<th>mAcc</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet [28]</td>
<td>-</td>
<td>49.0</td>
<td>41.1</td>
</tr>
<tr>
<td>SegCloud [39]</td>
<td>-</td>
<td>57.4</td>
<td>48.9</td>
</tr>
<tr>
<td>Tangent-Conv [38]</td>
<td>82.5</td>
<td>62.2</td>
<td>52.8</td>
</tr>
<tr>
<td>SPG [19]</td>
<td>86.4</td>
<td>66.5</td>
<td>58.0</td>
</tr>
<tr>
<td>PointCNN [24]</td>
<td>85.9</td>
<td>63.9</td>
<td>57.3</td>
</tr>
<tr>
<td>SSP+SPG [8]</td>
<td>87.9</td>
<td>68.2</td>
<td>61.7</td>
</tr>
<tr>
<td>GACNet [41]</td>
<td>87.8</td>
<td>62.9</td>
<td>59.6</td>
</tr>
<tr>
<td>SPH3D-GCN [22]</td>
<td>87.7</td>
<td>65.9</td>
<td>59.5</td>
</tr>
<tr>
<td>KPConv [40]</td>
<td>-</td>
<td>70.9</td>
<td>65.4</td>
</tr>
<tr>
<td>SegGCN (Prop.)</td>
<td>88.2</td>
<td>70.4</td>
<td>63.6</td>
</tr>
</tbody>
</table>

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.
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 $1 \times 1$ 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.

<table>
<thead>
<tr>
<th>Decoder convolution</th>
<th>#params</th>
<th>mAcc</th>
<th>mIoU</th>
<th>time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>single $1 \times 1$ Conv</td>
<td>3.0M</td>
<td>70.4</td>
<td>63.6</td>
<td>314</td>
</tr>
<tr>
<td>$2 \times$ ResNet blocks</td>
<td>5.6M</td>
<td>69.8</td>
<td>62.9</td>
<td>385</td>
</tr>
</tbody>
</table>

**4.3.3 Decoder module choice**

To justify our choice of $1 \times 1$ 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 $1 \times 1$ 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 $1 \times 1$ 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.
References


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


[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


[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


