Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds

Huan Lei, Naveed Akhtar, and Ajmal Mian

Abstract—We propose a spherical kernel for efficient graph convolution of 3D point clouds. Our metric-based kernels systematically quantize the local 3D space to identify distinctive geometric relationships in the data. Similar to the regular grid CNN kernels, the spherical kernel maintains translation-invariance and asymmetry properties, where the former guarantees weight sharing among similar local structures in the data and the latter facilitates fine geometric learning. The proposed kernel is applied to graph neural networks without edge-dependent filter generation, making it computationally attractive for large point clouds. In our graph networks, each vertex is associated with a single point location and edges connect the neighborhood points within a defined range. The graph gets coarsened in the network with farthest point sampling. Analogous to the standard CNNs, we define pooling and unpooling operations for our network. We demonstrate the effectiveness of the proposed spherical kernel with graph neural networks for point cloud classification and semantic segmentation using ModelNet, ShapeNet, RueMonge2014, ScanNet and S3DIS datasets. The source code and the trained models can be downloaded from https://github.com/hlei-ziyang/SPH3D-GCN.

Index Terms—3D point cloud, spherical kernel, graph neural network, semantic segmentation.

1 INTRODUCTION

Convolutional neural networks (CNNs) [1] are known for accurately solving a wide range of Computer Vision problems. Classification [2], [3], [4], [5], image segmentation [6], [7], [8], object detection [9], [10], [11], and face recognition [12], [13] are just a few examples of the tasks for which CNNs have recently become the default modelling technique. The success of CNNs is mainly attributed to their impressive representational prowess. However, their representation is only amenable to the data defined over regular grids, e.g. pixel arrays of images and videos. This is problematic for applications where the data is inherently irregular [14], e.g. 3D Vision, Computer Graphics and Social Networks.

In particular, point clouds produced by 3D vision scanners (e.g. LiDAR, Matterport) are highly irregular. Recent years have seen a surge of interest in deep learning for 3D vision due to self-driving vehicles. This has also resulted in multiple public repositories of 3D point clouds [15], [16], [17], [18], [19]. Early attempts of exploiting CNNs for point clouds applied regular grid transformation (e.g. voxel grids [20], [21], multi-view images [22]) to point clouds for processing them with 3D-CNNs or enhanced 2D-CNNs [2], [3], [4], [5]. However, this line of action does not fully exploit the sparse nature of point clouds, leading to unnecessarily large memory footprint and computational overhead of the methods. Riegler et al. [23] addressed the memory issue in dense 3D-CNNs with an octree-based network, termed OctNet. However, the redundant computations over empty spaces still remains a discrepancy of OctNet.

Computational graphs are able to capitalize on the sparse nature of point clouds much better than volumetric or multi-view representations. However, designing effective modules such as convolution, pooling and unpooling layers, becomes a major challenge for the graph based convolutional networks. These modules are expected to perform point operations analogous to the pixel operations of CNNs, albeit for irregular data. Earlier instances of such modules exist in theoretical works [24], [25], [26], which can be exploited to form Graph Convolutional Networks (GCNs) [26]. Nevertheless, these primitive GCNs are yet to be seen as a viable solution for point cloud processing due to their inability to effectively handle real-world point clouds.

Based on the convolution operation, GCNs can be divided into two groups, namely; the spectral networks [24], [25], [26], [27] and the spatial networks [28], [29], [30], [31], [32]. The former perform convolutions using the graph Laplacian and adjacency matrices, whereas the latter perform convolutions directly in the spatial domain. For the spectral networks, careful alignment of the graph Laplacians of different samples is necessary [27]. This is not easily achieved for the real-world point clouds. Consequently, the spatial networks are generally considered more attractive than the spectral networks in practical applications.

The spatial GCNs are challenged by the unavailability of discrete convolutional kernels in the 3D metric space. To circumvent the problem, mini-networks [28], [29], [31] are often applied to dynamically generate edge-wise filters. This incurs significant computational overhead, which can be avoided in the case of discrete kernels. However, design and application of discrete kernels in this context is not straightforward. Beside effective discretization of the metric space, the kernel application must exhibit the properties of (a) translation-invariance that allows identification of similar local structures in the data, and (b) asymmetry for vertex pair processing to ensure that the overall representation remains compact.

Owing to the intricate requirements of discrete kernels...
for irregular data, many existing networks altogether avoid the convolution operation for point cloud processing [33], [34], [35], [36]. Although these techniques report decent performance on benchmark datasets, they do not contribute towards harnessing the power of convolutional networks for point clouds. PointCNN [37] is a notable exception that uses a convolutional kernel for point cloud processing. However, its kernel is again defined using mini-networks, incurring high computational cost. Moreover, it is sensitive to the order of the neighborhood points, implicating that the underlying operation is not permutation-invariant, which is not a desired kernel property for point clouds.

In this work, we introduce a discrete metric-based spherical convolutional kernel that systematically partitions a 3D region into multiple volumetric bins as shown in Fig. 1. The kernel is directly applied to point clouds for convolution. Each bin of the kernel specifies learnable parameters to convolve the points falling in it. The convolution defined by our kernel preserves the properties of translation-invariance, asymmetry, as well as permutation-invariance. The proposed kernel is applied to point clouds using Graph Networks. To that end, we construct the networks with the help of range search [38] and farthest point sampling [35]. The former defines edges of the underlying graph, whereas the latter coarsens the graph as we go deeper into the network layers. We also define pooling and unpooling modules for our graph networks to downsample and upsample the vertex features. The novel convolutional kernel and its application to graph networks are thoroughly evaluated for the tasks of 3D point cloud classification and semantic segmentation. We achieve highly competitive performance on a wide range of benchmark datasets, including ModelNet [20], ShapeNet [16], RueMonge2014 [39], ScanNet [18] and S3DIS [17]. Owing to the proposed kernel, the resulting graph networks are found to be efficient in both memory and computation. This leads to fast training and inference on high resolution point clouds.

This work is a significant extension of our preliminary findings presented in IEEE CVPR 2019 [32]. Below, we summarize the major directions along which the technique is extended beyond the preliminary work.

- **Separable convolution.** We perform the depth-wise and point-wise convolution operation separately in this work rather than simultaneously as in [32]. The separable convolution strategy is inspired by Xception [39], and significantly reduces the number of network parameters and computational cost.

- **Graph architecture.** Instead of the octree-guided network of [32], we use a more flexible graph-based technique to design our network architectures. This allows us to exploit convolution blocks and define pooling/unpooling operations independent of convolution. In contrast to the convolution-based down/upsampling, specialized modules for these operations are highly desirable for processing large point clouds. Moreover, this strategy also brings our network architectures closer to the standard CNNs.

- **Comprehensive evaluation on real-world data.** Compared to the preliminary work [32], we present a more thorough evaluation on real-world data. Highlights include 4.2% performance gain over [32] for the RueMonge2014 dataset, and comprehensive evaluation on two additional datasets, ScanNet and S3DIS. The presented results ascertain the computational efficiency of our technique with highly competitive performance on the popular benchmarks.

- **Tensorflow implementation.** While [32] was implemented in Matconvnet, with this article, we release cuda implementations of the spherical convolution and the pooling/unpooling operations for Tensorflow. The source code is available on Github (https://github.com/hlei-ziyan/SPH3D-GCN) for the broader research community.

## 2 Related Work

PointNet [33] is one of the first techniques to directly process point clouds with deep networks. It uses the $xyz$ coordinates of points as input features. The network learns point-wise features with shared MLPs, and extracts a global feature with max pooling. One limitation of this technique is that it does not explore the geometric context of points in representation learning. PointNet++ [35] addresses that by applying max-pooling to the local regions hierarchically. However, both networks must rely on max-pooling to aggregate any context information without convolution.

SO-Net [36] builds an $m \times m$ rectangular map from the point cloud, and hierarchically learns node-wise features within the map using mini-PointNet. However, similar to
the original PointNet, it also fails to exploit any convolution modules. KCNet \cite{41} learns features with kernel correlation between the local neighboring points and a template of learnable points. This can be optimized in a training session similar to convolutional kernels. In contrast to the image-like map used by the SO-Net, KCNet is based on graph representation. Kd-network \cite{34} is a prominent contribution that processes point clouds with tree structure based networks. This technique also uses point coordinates as the input and computes the feature of a parent node by concatenating the features of its children in a balanced tree. Despite their varied network architecture construction, none of the above methods contribute towards developing convolutional networks for point clouds. Approaches that advance research in that direction can be divided into two broad categories, discussed below.

\subsection*{2.1 3D Convolutional Neural Networks}
At the advent of 3D deep learning, researchers predominantly extracted features with 3D-CNN kernels using volumetric representations. The earlier attempts in this direction could only process voxel-grids of low resolution (e.g. $30\times30\times30$ in ShapeNets \cite{20}, $32\times32\times32$ in VoxNet \cite{21}), even with the modern GPUs. This issue also transcended to the subsequent works along this direction \cite{42, 43, 44, 45}. The limitation of low input resolution was a natural consequence of the cubic growth of memory and computational requirements associated with the dense volumetric inputs. Different solutions later appeared to address these issues. For example, Engelcke et al. \cite{46} introduced sparsity in the input and hidden neural activations. Their solution is effective in reducing the number of convolutions, but not the amount of required memory. Li et al. \cite{47} proposed a field probing neural network, which transforms 3D data into intermediate representations with a small set of probing filters. Although this network is able to reduce the computational and memory costs of fully connected layers, the probing filters fail to support weight sharing. Later, Riegler et al. \cite{23} proposed the octree-based OctNet, which represents point clouds with a hybrid of shallow grid octrees (depth = 3). Compared to its dense peers, OctNet reduces the computational and memory costs to a large degree, and is applicable to high-resolution inputs up to $256\times256\times256$. However, it still has to perform unnecessary computations in the empty spaces around the objects. Other recent techniques also transform the original point cloud into other regular representations like tangent image \cite{48} or high-dimensional lattice \cite{49} such that the standard CNNs can be applied to the transformed data.

\subsection*{2.2 Graph Convolutional Networks}
The demand of irregular data processing with CNN-like architectures has resulted in a recent rise of graph convolutional networks \cite{14}. In general, the broader graph-based deep learning has also seen techniques besides convolutional networks that update vertex features recurrently to propagate the context information (e.g. \cite{50, 51, 52, 53, 54}). However, here, our focus is on graph convolutional networks that relate to our work more closely.

Graph convolutional networks can be grouped into spectral networks (e.g. \cite{24, 25, 26}) and spatial networks (e.g. \cite{28, 29}). The spectral networks perform convolution on spectral vertex signals converted from Fourier transformation, while the spatial networks perform convolution directly on the spatial vertices. A major limitation of the spectral networks is that they require the graph structure to be fixed, which makes their application to the data with varying graph structures (e.g. point clouds) challenging. Yi et al. \cite{27} attempted to address this issue with Spectral Transformer Network (SpecTN), similar to STN \cite{55} in the spatial domain. However, the signal transformation from spatial to spectral domains and vice-versa has computational complexity $O(n^2)$, resulting in prohibitive requirements for large point clouds.

ECC\cite{28} is among the pioneering works for point cloud analysis with graph convolution in the spatial domain. Inspired by the dynamic filter networks \cite{56}, it adapts MLPs to generate convolution filters between the connected vertices dynamically. The dynamic generation of filters naturally comes with a computational overhead. DGCNN \cite{57}, FlexConv \cite{58} and SpiderCNN \cite{59} subsequently explore different parameterizations to generate the edge-dependent filters. Instead of generating filters for the edges individually, few networks also generate a complete local convolution kernel at once using mini networks \cite{31, 37}. Li et al. \cite{37} recently introduced PointCNN that uses a convolution module named $\mathcal{X}$-Conv for point cloud processing. The network achieves good performance on the standard benchmarks (e.g. ShapeNet and S3DIS). However, the generated kernels are sensitive to the order of neighborhood points indicating that the underlying representation is not permutation-invariant. Moreover, the strategy of dynamic kernel generation makes the technique computationally inefficient.

More recently, Wang et al. \cite{37} inserted an attention mechanism in graph convolutional networks to develop GACNet. Such an extension of graph networks is particularly helpful for semantic segmentation as it enforces the neighborhood vertices to have consistent semantic labels similar to CRF \cite{60}. Besides the convolution operation, graph coarsening and edge construction are two essential parts for the graph network architectures. We briefly review the methods along these aspects below.

**Graph coarsening:** Point cloud sampling methods are useful for graph coarsening. PointNet++ \cite{35} utilizes farthest point sampling (FPS) to coarsen the point cloud, while FlexConv \cite{58} samples the point cloud based on inverse densities (IDS) of each point. Random sampling is the simplest alternative to FPS and IDS, but it does not perform as well for the challenging tasks like semantic segmentation. Recently, researchers also started to explore the possibility of learning sampling with deep neural networks \cite{61}. In this work, we exploit FPS as the sampling strategy for graph coarsening, as it does not need training and it reduces the point cloud resolution relatively uniformly.

**Graph connections:** Point neighborhood search can be used to build edge connections in a graph. KNN search generates fixed number of neighborhood points for a given point, which results in a regular graph. Range search generates flexible number of neighborhood points, which may results in irregular graphs. Tree structures can also be seen as special kinds of graphs \cite{32, 34}, however, the default
absence of intra-layer connections in trees drastically limits their potential as graph networks. In a recent example, Rao et al. [62] proposed to employ spherical lattices for regular graph construction. Their technique relies on $1 \times 1$ convolution and max-pooling to aggregate the geometric context between neighbouring points.

In this paper, we use range search to establish the graph connections for its natural compatibility with the proposed kernel. Note that our spherical kernel does not restrict the graph vertex degrees to be fixed. Hence, unlike [31], [37], our kernel is applicable to both regular and irregular graphs.

### 3 Discrete Convolution Kernels

Given an arbitrary point cloud of $m$ points $\mathcal{P} = \{x_i \in \mathbb{R}^3\}_{i=1}^m$, we represent the neighborhood of each point $x_i$ as $\mathcal{N}(x_i)$. To achieve graph convolution on the target point $x_j$, the more common ‘continuous’ filter approaches [28], [29], [31], [37], [38], [59] parameterize convolution as a function of local coordinates. For instance, suppose $w$ is the filter that computes the output feature of channel $c$. These techniques may represent the filter as $w = h(x_j - x_i)$, where $h(\cdot)$ is a continuous function (e.g. MLP) and $x_i \in \mathcal{N}(x_j)$. However, compared to the continuous filters, discrete kernels is predefined and it does not need the above mentioned (or similar) intermediate computations. This makes a discrete kernel computationally more attractive.

Following the standard CNN kernels, a primitive discrete kernel for point clouds can be defined similar to the 3D-CNN kernels [20], [21]. For resolution $h$, this kernel comprises $h^3$ weight filters $w_{\kappa,\ell,\rho} \in \{1,\ldots,h\}$. By incorporating the notion of separable convolution [40] into this design, each weight filter is transformed from a vector $w_k$ to a scalar $w_k$. It is noteworthy that the application of a discrete kernel to ‘graph’ representation is significantly different from its volumetric counterpart. Hence, to differentiate, we refer to a kernel for graphs as CNN3D kernel. A CNN3D kernel indexes the bins and for the $\kappa$th bin, it uses $w_{\kappa}$ to propagate features from all neighbouring point $x_j$, $\forall j \in \mathcal{N}(x_i)$ to the target point $x_i$, see Fig. 2. It performs convolutions only at the point locations, avoiding unnecessary computations at empty spaces, which is in contrast to 3D-CNN kernels.

We make the following observation in relation to improving the CNN3D kernels. For images, the more primitive constituents, i.e. patches, have traditionally been used to extract hand-crafted features [63], [64]. The same principle transcended to the receptive fields of automatic feature extraction with CNNs, which compute feature maps using the activations of well-defined rectangular regions of images. Whereas rectangular regions are intuitive choice for images, spherical regions are more suited to process unstructured 3D data such as point clouds. Spherical regions are inherently amenable to computing geometrically meaningful features for such data [65], [66], [67]. Inspired by this natural kinship, we introduce the concept of spherical convolution kernel (termed SPH3D kernel) that considers a 3D sphere as the basic geometric shape to perform the convolution operation. We explain the proposed discrete spherical kernel in Section 3.1 and later contrast it to the existing CNN3D kernels in Section 3.2.

#### 3.1 Spherical Convolutions

We define the convolution kernel with the help of a sphere of radius $\rho \in \mathbb{R}^+$, see Fig 2. For a target point $x_i$, we consider its neighborhood $\mathcal{N}(x_i)$ to be the set of points within the sphere centered at $x_i$, i.e. $\mathcal{N}(x_i) = \{x : d(x, x_i) \leq \rho\}$, where $d(\cdot, \cdot)$ is a distance metric - $L_2$ distance in this work. We divide the sphere into $n \times p \times q$ ‘bins’ by partitioning the occupied space uniformly along the azimuth ($\theta$) and elevation ($\phi$) dimensions. We allow the partitions along the radial ($r$) dimension to be non-uniform because the cubic volume growth for large radius values can be undesirable. Our quantization of the spherical region is mainly inspired by 3DSC [65]. We also define an additional bin corresponding to the origin of the sphere to allow the case of self-convolution of points on the graph. To produce an output feature map, we define a learnable weight parameter $w_{\kappa,\ell,\rho} \in \mathbb{R}$ for each bin, where $w_0$ relates to self-convolution. Combined, the $n \times p \times q + 1$ weight values specify a single spherical convolution kernel.

To compute the activation value for a target point $x_i$, we first identify the relevant weight values of its neighboring points $x_j \in \mathcal{N}(x_i)$. It is straightforward to associate $w_0$ to $x_i$ for self-convolution. For the non-trivial cases, we first represent the neighboring points in terms of their spherical coordinates that are referenced using $x_k$ as the origin. That is, for each $x_j$ we compute $T(\Delta_{ji}) \rightarrow \psi_{ji}$, where $T(\cdot)$ defines the transformation from Cartesian to Spherical coordinates and $\Delta_{ji} = x_j - x_i$. Supposing that the bins of the quantized sphere are respectively indexed by $k_\theta$, $k_\phi$ and $k_r$ along the azimuth, elevation and radial dimensions, the weight values associated with each spherical bin can then be indexed as $\kappa = k_\theta + (k_\phi - 1) \times n + (k_r - 1) \times n \times p$, where $k_\theta \in \{1,\ldots,n\}$, $k_\phi \in \{1,\ldots,p\}$, $k_r \in \{1,\ldots,q\}$. Using this indexing, we relate the relevant weight value to each $\psi_{ji}$, and hence $x_j$. In the $i$th network layer, the activation for the $i$th point in channel $c$ gets computed as:

$$
\begin{align}
\psi_{ic} &= \frac{1}{|\mathcal{N}(x_i)|} \sum_{j=1}^{\mathcal{N}(x_i)} w_{\kappa,\ell,\rho}^{l-1} \ast_{ji} + b_c, \\
\psi_{ic} &= f(z_{ic}),
\end{align}
$$

where $w_{\kappa,\ell,\rho}^{l-1}$ is the feature of a neighboring point from layer $l-1$, $w_{\kappa}^{l-1}$ is the weight value, and $f(\cdot)$ is the non-linear activation function - ELU [69] in our experiments. By applying the spherical convolution $\lambda$ times for each input channel, we produce $\lambda C_{in}$ output features for the target convolution point $x_i$.

To elaborate on the characteristics of the spherical convolution kernel, we denote the boundaries along $\theta$, $\phi$ and $r$ dimensions of the kernel bins as follows:

$$
\begin{align}
\Theta &= [\Theta_1, \ldots, \Theta_{n+1}], \quad \Theta_1 < \Theta_{k+1}, \Theta_k \in [-\pi, \pi], \\
\Phi &= [\Phi_1, \ldots, \Phi_{p+1}], \quad \Phi_k < \Phi_{k+1}, \Phi_k \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \\
R &= [R_1, \ldots, R_{q+1}], \quad R_k < R_{k+1}, R_k \in (0, \rho].
\end{align}
$$

1. The term spherical in Spherical CNN [68] is used for surfaces (i.e. 360° images) not the ambient 3D space. Our notion of spherical kernel is widely dissimilar, and it is used in a different context. Also, note that, different from the preliminary work [32], here the spherical kernel is only used to perform depth-wise spatial convolutions.
The constraint of uniform splitting along the azimuth and elevation results in $\Theta_{k+1} - \Theta_k = \frac{2\pi}{n} \text{ and } \Phi_{k+1} - \Phi_k = \frac{\pi}{n}$.

**Lemma 2.1:** If $\Theta_k \cdot \Theta_{k+1} \geq 0$, $\Phi_k \cdot \Phi_{k+1} \geq 0$ and $n > 2$, then for any two points $x_a \neq x_b$ within the spherical convolution kernel, the weight value $w_{\kappa}, \forall \kappa > 0$, are applied asymmetrically.

**Proof:** Let $\Delta_{ab} = x_a - x_b = [\delta_x, \delta_y, \delta_z]^T$, then $\Delta_{ba} = [-\delta_x, -\delta_y, -\delta_z]^T$. Under the Cartesian to Spherical coordinate transformation, we have $T(\Delta_{ab}) = \psi_{ab} = [\theta_{ab}, \phi_{ab}, r]^T$, and $T(\Delta_{ba}) = \psi_{ba} = [\theta_{ba}, \phi_{ba}, r]^T$. Assume that the resulting $\psi_{ab}$ and $\psi_{ba}$ fall in the same bin indexed by $\kappa \leftarrow (k_\theta, k_\phi, k_r)$, i.e. $w_{\kappa}$ will have to be applied symmetrically to the original points. In that case, under the inverse transformation $T^{-1}(\cdot)$, we have $\delta_z = r \sin \phi_{ab}$ and $(-\delta_z) = r \sin \phi_{ba}$. The condition $\Phi_{k_\theta} \cdot \Phi_{k_\theta+1} \geq 0$ entails that $-\delta_z^2 \geq \delta_z \cdot (-\delta_z) = (r \sin \phi_{ab}) \cdot (r \sin \phi_{ba})$ entails $\delta_z^2 \geq 0 \implies \delta_z \geq 0$. Similarly, $\Theta_{k_\phi} \cdot \Theta_{k_\phi+1} \geq 0 \implies \delta_y = 0$. Thus, $\delta_z = 0$. The points falling in the $\kappa^{th}$ bin are propagated to $x_a$ with the weight $w_{\kappa}$. Multiple points falling in the same bin, e.g. $x_a$ and $x_b$, use the same weight for computing the output feature at $x_a$.

This division results in a kernel size (i.e. total number of bins) $4 \times 4 \times 3 + 1 = 49$, which is one of the coarsest multi-scale quantization allowed by Lemma 2.1.

Notice that, if we move radially from the center to periphery of the spherical kernel, we encounter identical number of bins (16 in this case) after each edge defined by $R$, where fine-grained bins are located close to the origin that can encode detailed local geometric information of the points. This is in sharp contrast to CNN3D kernels that must keep the size of all cells constant and rely on increased resolution to capture the finer details. This makes their number of parameters grow cubically, harming the scalability. The multi-scale granularity of spherical kernel (SPH3D) allows for more compact representation.

To corroborate, we briefly touch upon classification with CNN3D and SPH3D kernel, using a popular benchmark dataset ModelNet40 [20] in Table 1. We give further details on the dataset and experimental settings in Section 5. Here, we focus on the single aspect of representation compactness resulting from the non-uniform granularity of the bins in SPH3D. In the table, the only difference in the networks is in the used kernels. All the other experimental details are ‘exactly’ the same for all networks. Network 1 and 2 use CNN3D kernels that partition the space into $3 \times 3 \times 3 = 27$ and $5 \times 5 \times 5 = 125$ bins, respectively. The SPH3D kernel partitions the space into $8 \times 2 \times 2 + 1 = 33$ bins. Consequently, the kernel requires $1.22 \times$ parameters as compared to the Network-1 kernel, but only $0.26 \times$ parameters required by the Network-2 kernel. However, the performance of Network-3 easily matches Network-2. Such an advantage is a natural consequence of the non-uniform partitioning allowed by our kernel.
Fig. 3. Illustration of Encoder-decoder graph neural network for a toy example. A graph \( G_i \) of 12 vertices gets coarsened to \( G^{i+1} \) (8 vertices) and further to \( G^{i+2} \) (4 vertices), and expanded back to 12 vertices. The width variation of feature maps depicts different number of feature channels, whereas the number of cells indicates the total vertices in the corresponding graph. The pooling/unpooling operations compute features of the coarsened/expanded graphs. Consecutive convolutions are applied to form convolution blocks. The shown architecture for semantic segmentation uses skip connections for feature concatenation, similar to U-Net. For classification, the decoder and skip connections are removed and a global representation is fed to a classifier. We omit self loops in the shown graphs for clarity.

### 4 Graph Neural Network

In this work, we employ graph neural network to process point clouds. Compared to the inter-layer connectivity of the octree-guided network of our preliminary work \([32]\), graph representation additionally allows for intra-layer connections. This is beneficial in defining effective convolutional blocks as well as pooling/unpooling modules in the network. Let us consider a graph \( G = (V, E) \) constructed from a point cloud \( \mathcal{P} = \{x_1, \ldots, x_m\} \), where \( V = \{1, 2, \ldots, m\} \) and \( E \subseteq |V| \times |V| \) respectively represent the sets of vertices and edges. It is straightforward to associate each vertex \( i \in V \) of the graph to a point location \( x_i \) and its corresponding feature \( a_i \). However, the edge set \( E \) must be carefully established based on the neighborhood of the points.

**Edge construction:** We use range search with a specified radius \( \rho \) to get the spatial neighborhood of each point and construct the edge connections of each graph vertex. In the range search, neighborhood computations are independent of each other, which makes the search suitable for parallel processing and taking advantage of GPUs. The time complexity of the search is linear in the number of vertices \( |V| \). One potential problem of using range search is that large number of neighborhood points in dense clouds can cause memory issues. We sidestep this problem by restricting the number of neighboring points to \( K \in \mathbb{Z}^+ \) by randomly sub-sampling the neighborhood, if required. The edges are finally built on the sampled points. As a result, the neighborhood indices of the \( i^{th} \) vertex can be denoted as \( \mathcal{N}(i) = \{j : (j, i) \in E\} \), in which \( |\mathcal{N}(i)| \leq K \). With these sets identified, we can later compute features for vertices with spherical convolution.

**Graph coarsening:** We use Farthest Point Sampling (FPS) to coarsen the point graph in our network layer-by-layer. The FPS algorithm selects one random seed vertex, and iteratively searches for the point that is farthest apart from the previously selected points for the sampling purpose. The algorithm terminates when the desired number of sampled points are acquired, which form the coarsened graph. By alternately constructing the edges and coarsening the graph for \( l_{\text{max}} \) times, we construct a graph pyramid composed of \( G^0 \to G^1 \to \ldots \to G^{l_{\text{max}}-1} \to G^{l_{\text{max}}} \).

As compared to the octree structure based graph coarsening adopted in the preliminary work \([32]\), FPS coarsening has the advantage of keeping the number of vertices of each layer fixed across different samples, which is conducive for more systematic application of convolutional kernels.

**Pooling:** Once a graph is coarsened, we still need to compute the features associated with its vertices. To that end, we define max pooling and average pooling operations to sample features for the coarsened graph vertices. Inter-layer graph connections facilitate these operations. To be consistent, we denote the graphs before and after pooling layer \( l \in \{1, \ldots, l_{\text{max}}\} \) as \( G^{l-1} = (V^{l-1}, E^{l-1}) \) and \( G^l = (V^l, E^l) \) respectively, where \( V^{l-1} \supseteq V^l \). Let \( i^{l-1} \in V^{l-1} \) and \( i^l \in V^l \) be the two vertices associated with the same point location. The inter-layer neighborhood of \( i^l \) can be readily constructed from graph \( G^{l-1} \) as \( \mathcal{N}(i^l) = \{j : (j, i^{l-1}) \in E^{l-1}\} \). We denote the features of \( i^l \) and its neighborhood point \( j \in \mathcal{N}(i^l) \) as \( a_j = [a_{1j}, \ldots, a_{|\mathcal{N}(i^l)|j}]^T \) and \( a_{i^{l-1}} = [a_{1i^{l-1}}, \ldots] \) respectively. The max pooling operation then computes the feature of the vertex \( i^l \) as

\[
a_{i^l} = \max\{a_{j^{l-1}} : j \in \mathcal{N}(i^l)\},
\]

while the average pooling computes it as

\[
a_{i^l} = \frac{1}{|\mathcal{N}(i^l)|} \sum_{j \in \mathcal{N}(i^l)} a_{j^{l-1}}.
\]
We introduce both pooling operations in our source code release, but use max pooling in our experiments as it is commonly known to have superior performance in point cloud processing [33, 35, 41].

Unpooling: Decoder architectures with increasing neuron resolution are important for element-wise predictions in semantic segmentation [8], dense optical flow [70], etc. We build graph decoder by inverting the graph pyramid as \( G^{\text{max}} \rightarrow \cdots \rightarrow G^{1} \rightarrow G^{0} \). The coarsest graph \( G^{\text{max}} \) is omitted in the reversed pyramid because it is shared between encoder and decoder. We denote the graphs before and after an unpooling layer \( l \in \{1, \ldots, l_{\text{max}}\} \) as \( G^{l_{\text{max}}-l} \) and \( G^{l_{\text{max}}-l-1} \) respectively. To upsample the features from \( G^{l_{\text{max}}-l} \) to \( G^{l_{\text{max}}-l-1} \), we define two types of feature interpolation operations, namely; uniform interpolation and weighted interpolation. Notice that the neighborhood set \( \mathcal{N}(.) \) in Eqs. (4), (5) is readily available because of the relation \( V^{-1} \supseteq V \). However, the vertices of graphs \( G^{l_{\text{max}}-l} \) and \( G^{l_{\text{max}}-l-1} \) satisfy \( V^{l_{\text{max}}-l+1} \supseteq V^{l_{\text{max}}-l} \) on the contrary. Therefore, we have to additionally construct the neighborhood of \( i^{l_{\text{max}}-l} \) and its neighborhood points \( j \in \mathcal{N}(i^{l_{\text{max}}-l}) \). The features of \( i^{l_{\text{max}}-l} \) and its neighborhood points \( j \in \mathcal{N}(i^{l_{\text{max}}-l}) \) can be consistently denoted as \( a_{i}^{l_{\text{max}}-l} \) and \( q_{j}^{l_{\text{max}}-l} \). The uniform interpolation computes the feature of vertex \( i^{l_{\text{max}}-l} \) as the average features of its inter-layer neighborhood points, i.e.

\[
a_{i}^{l_{\text{max}}+l} = \frac{1}{|\mathcal{N}(i^{l_{\text{max}}-l})|} \sum_{j \in \mathcal{N}(i^{l_{\text{max}}-l})} a_{j}^{l_{\text{max}}+l-1}. \tag{6}
\]

The weighted interpolation computes the features of vertex \( i^{l_{\text{max}}-l} \) by weighing its neighborhood features based on their distance to \( i^{l_{\text{max}}-l} \). Mathematically,

\[
a_{i}^{l_{\text{max}}+l} = \frac{1}{\sum_{j \in \mathcal{N}(i^{l_{\text{max}}-l})} w_{ji}} \sum_{j \in \mathcal{N}(i^{l_{\text{max}}-l})} w_{ji} a_{j}^{l_{\text{max}}+l-1}, \tag{7}
\]

where \( w_{ji} = d(x_{i}^{l_{\text{max}}-l+1}, x_{j}^{l_{\text{max}}-l}) \). Here, \( d(\cdot, \cdot) \) is the distance function and the points \( x_{i}^{l_{\text{max}}-l+1} \) and \( x_{j}^{l_{\text{max}}-l} \) are associated to vertices \( j \) and \( i^{l_{\text{max}}-l} \), respectively. In our source code, we provide both types of interpolation functionalities for upsampling. However, the experiments in Section 5 are performed with uniform interpolation for its computational efficiency.

In Fig. 3, we illustrate an encoder-decoder graph neural network constructed by our technique for a toy example. In the shown network, a graph \( G^{6} \) of 12 vertices gets coarsened to \( 8 (G^{4+1}) \) and \( 4 (G^{4+2}) \) vertices in the encoder network, and later gets expanded in the decoder network. The pooling/unpooling operations are applied to learn features of the structure altered graphs. The graph structure remains unchanged during convolution operation. Notice, we apply consecutive spherical convolutions to form convolution blocks in our networks. In the figure, variation in width of the feature maps depicts different number of channels (e.g. 128, 256 and 384) for the features. The shown U-shape architecture for the task of semantic segmentation also exploits skipping connections similar to U-Net [7, 8]. These connections copy features from the encoder and concatenate them to the decoder features. For the classification task, these connections and the decoder part are removed, and a global feature representation is fed to a classifier comprising fully connected layers. The simple architecture in Fig. 3 graphically illustrates the application of the above-mentioned concepts to our networks in Section 5, where we provide details of the architectures used in our experiments.

Software for Tensorflow: With this article, we also release a cuda enabled implementation for the above presented concepts. The package is Tensorflow compatible [71]. As compared to the Matconvnet [72] source code of the preliminary work [32], Tensorflow compatibility is chosen due to the popularity of the programming framework. In the package, we provide cuda implementations of the spherical convolution, range search, max pooling, average pooling, uniform interpolation and weighted interpolation. The provided spherical kernel implementation can be used for convolutions on both regular and irregular graphs. Unlike existing methods (e.g. [31], [37]), we do not impose any constraint on the vertex degree of the graph allowing the graphs to be more flexible, similar to ECC [28]. In our implementation, the spherical convolutions are all followed by batch normalization [23]. In the preliminary work [32], the implemented spherical convolution does not separate the depth-wise convolution from the point-wise convolution [40], thereby performing the two convolutions simultaneously similar to a typical convolution operation. Additionally, the previous implementation is specialized to octree structures, and hence not applicable to general graph architectures. The newly released implementation for TensorFlow improves on all of these aspects. The source code and further details of the released package can be found at https://github.com/lei-ziyuan/SPH3D-GCN

5 Experiments

We evaluate our technique for classification and semantic segmentation tasks using clean CAD point clouds and large-scale noisy point clouds of real-world scenes. The dataset used in our experiments include ModelNet [20], ShapeNet [16], RueMonge2014 [39], ScanNet [18] and S3DIS [17], for which representative samples are illustrated in Fig. 4. We only use the \((x, y, z)\) coordinates of points to train our networks, except when the \((r, g, b)\) values are also available. In that case, we additionally use those values by rescaling them into the range \([-1, 1]\). We note that, a few existing methods also take advantage of normals as input features [33, 35, 36, 62]. However, normals are not directly sensed by the 3D sensors and must be computed separately, entailing additional computational burden. Hence, we avoid using normals as input features except for RueMonge2014, which already provides the normals.

Throughout the experiments, we apply the spherical convolution with a kernel size \(8 \times 2 \times 2 + 1\). Our network training is conducted on a single Titan Xp GPU with 12 GB memory. We use Adam Optimizer [74] with an initial learning rate of 0.001 and momentum 0.9 to train the network. The batch size is kept fixed to 32 in ModelNet and ShapeNet, and 16 the remaining datasets. The maximum neighborhood connections for each vertex is set to \(K = 64\). These hyper-parameters are empirically optimized with cross-validation.
We also employ data augmentation in our experiments. For that, we use random sub-sampling to drop points, and random rotation, which include azimuth rotation (up to $2\pi$ rad) and small arbitrary perturbations (up to $10^2\pi$ degrees). For change the view of point clouds. We also apply random scaling, shifting and noisy translation of points with std. dev $= 0.01$. These operations are commonly found in the related literature. We apply them on-the-fly in each training epoch of the network.

**Network Configuration:** Table 2 provides the summary of network configurations used in our experiments for the classification and segmentation tasks. We use identical configurations for semantic segmentation on the realistic datasets RueMonge2014, ScanNet and S3DIS, but a different one for the part segmentation of the synthetic ShapeNet. Our network for the realistic datasets takes input point clouds of size 8,192. To put this size into perspective, it is four times of 2,048 points accepted by PointCNN [37]. Further discussion on network configuration is also provided in the related sections below.

### 5.1 ModelNet40

The benchmark ModelNet40 dataset [20] is used to demonstrate the promise of our technique for object classification. The dataset comprises object meshes for 40 categories with 9,843/2,468 training/testing split. To train our network, we create the point clouds by sampling on mesh surfaces. Compared to the existing methods (e.g. [25], [33], [35], [41]), the convolutions performed in our network enable processing large input point clouds. Hence, our network is trained employing 10K input points. The channel settings of the first MLP and the six SPH3D layers is 32 and 64-64-64-128-128-128. We use the same classifier 512-256-40 as the previous works [32], [33], [41]. The Encoder4 in Table 2 indicates that the network learns a global representation of the point cloud using G-SPH3D. For that, we create a virtual vertex whose associated coordinates are computed as the average coordinates of the real vertices in the graph. We connect all the real vertices to the virtual vertex, and use a spherical kernel of size $8 \times 2 \times 1 + 1$ for feature computation. G-SPH3D computes the feature only at the virtual vertex, that becomes the global representation of point cloud for the classifier.

Following our preliminary work for $\Psi$-CNN [32], we boost performance of the classification network by applying max pooling to the intermediate layers, i.e. Encoder1, Encoder2, Encoder3. We concatenate these max-pooled features to the global feature representation in Encoder4 to form a more effective representation. This results in features with $832 = 64 + 128 + 128 + 512$ channels for the classifier. We use weight decay of $10^{-5}$ in the end-to-end network training, where 0.5 dropout [25] is also applied to the fully connected layers of the classifier to alleviate overfitting.

### 5.2 ShapeNet

The ShapeNet part segmentation dataset [16] contains 16,881 synthetic models from 16 categories. The models in
achieves the same instance mIoU as Ψ-CNN [32], but also outperforms the other approaches on 9 out of 16 categories, resulting in the highest class mIoU 84.9%. We also trained a single network with the configuration shown in Table 2 to segment the 50 parts of all categories together. In that case, the obtained instance and class mIoUs are 85.4% and 82.7%, respectively. These results are very close to highly competitive method SFCNN [62]. In all segmentation experiments, we apply the random sampling operation multiple times to ensure that every point in the test set is evaluated.

5.3 RueMonge2014

We test our technique for semantic segmentation of the real-world outdoor scenes using RueMonge2014 dataset [39]. This dataset contains 700 meters Haussmann style facades along an European street annotated with point-wise labelling. There are 7 classes in total, which include window, wall, balcony, door, roof, and shop. The point clouds are provided with normals and color features. We use xyz coordinates as well as normals and color values to form 9-dim input features for a point. The detailed network configuration used in this experiment is shown in Table 2 for which C = 7 for RueMonge2014. The original point clouds are split into smaller point cloud blocks following the pcl_split.mut indexing file provided with the dataset. We randomly sample 8,192 points from each block and use the sampled point clouds for training and testing. To standardize the points, we force their x and y dimensions to have zero mean values, and the z dimension is kept non-negative. In the real-world applications (here and the following sections), we use data augmentation but no weight decay or dropout. As compared to the preliminary work [32], we do not perform pre-processing in terms of alignment of the facade plane and gravitational axis correction. Besides, the processed blocks are also mostly much larger. Under the evaluation protocol of [80], Table 5 compares our current approach SPH3D-GCN with the recent methods, including Ψ-CNN [32]. It can be seen that SPH3D-GCN achieves very competitive performance, using only 0.4M parameters.

5.4 ScanNet

ScanNet [18] is an RGB-D video dataset of indoor environments that contains reconstructed indoor scenes with rich annotations for 3D semantic labelling. It provides 1,513

---

**Table 2**

Network configuration details: NN(\(\rho\)) denotes a range search with radius \(\rho\). SPH3D(\(\alpha, \beta, \lambda\)) represents a separable spherical convolution that takes \(\alpha\) input features, performs a depth-wise convolution with a multiplier \(\beta\) followed by a point-wise convolution to generate \(\beta\) features. When \(\lambda\) is omitted in the table, we use \(\lambda = 2\). MLP(\(\alpha, \beta\)) and FC(\(\alpha, \beta\)) indicate multilayer perceptron and fully connected layer taking \(\alpha\) input features, and \(\alpha\) output features. G-SPH3D denotes global spherical convolution that applies SPH3D once to a single point for global feature learning. The brackets [ ] are used to show feature concatenation. The pool(A, B) and unpool(A, B) operations transform vertices A into B, and C indicates the number of classes in a dataset.

| Layer Name | MLP1 | Encoder1 | Encoder2 | Encoder3 | Encoder4 | Decoder1 | Decorder2 | Decorder3 | Decorder4 | Output |
|------------|------|----------|----------|----------|----------|----------|----------|-----------|-----------|--------|
| ModelNet40 | MLP(3.32) | NN(p = 0.1) | SPH3D(64,128) | pool(1024,512) | SPH3D(512,128) | pool(2048,1024) | SPH3D(128,256) | pool(768,384) | SPH3D(256,128) | unpool(384,768) | unpool(1024,512) |
| ShapeNet | MLP(3.64) | NN(p = 0.1) | SPH3D(128,256) | pool(2048,1024) | SPH3D(512,256) | pool(768,384) | SPH3D(256,128) | pool(384,128) | SPH3D(128,256) | unpool(384,768) |
| RueMonge2014 | MLP(9.64) | NN(p = 0.1) | SPH3D(128,256) | pool(2048,1024) | SPH3D(512,256) | pool(768,384) | SPH3D(256,128) | pool(384,128) | SPH3D(128,256) | unpool(384,768) |
| ScanNet | MLP(6.64) | NN(p = 0.1) | SPH3D(128,256) | pool(2048,1024) | SPH3D(512,256) | pool(768,384) | SPH3D(256,128) | pool(384,128) | SPH3D(128,256) | unpool(384,768) |

**Table 3**

ModelNet40 classification: Average class and instance accuracies are reported along with the number of input points per sample (#points), the number of network parameters (#params), and the train/test time.

| Method | #point | #params | class | instance | time (ms) | training | testing |
|--------|--------|---------|-------|----------|-----------|----------|---------|
| ECC    | 1000   | 0.2M    | 83.2  | 87.4     | --        | --       | --      |
| PointNet | 1024  | 3.5M    | 86.2  | 89.2     | 7.9       | 2.5      |        |
| PointNet++ | 1024 | 1.5M    | 88.0  | 90.7     | 4.9       | 1.3      |        |
| KD-net(10) | 1024 | 3.5M    | 86.3  | 90.6     | 3.3       | 1.3      |        |
| SO-Net | 2048  | 2.4M    | 87.3  | 90.9     | --        | --       | --      |
| KCNet  | 2048  | 0.9M    | --    | 91.0     | --        | --       | --      |
| PointCNN | 1024 | 0.6M    | 88.0  | 91.7     | 19.4      | 7.5      |        |
| SFCNN  | 1024  | 8.6M    | --    | 91.4     | --        | --       | --      |
| Ψ-CNN  | 10000 | 3.0M    | 88.7  | 92.0     | 84.3      | 34.1     |        |
| SPH3D-GCN (Proposed) | 2048 | 0.7M    | 88.5  | 91.4     | 4.0       | 1.4      |        |

4. We remove parts represented with a single point in the range 0.3.
Scalability: The combination of discrete kernel, separable convolution and graph-based architecture adds to the scalability of the proposed SPH3D-GCN. In Table 8 we compare our network on computational and memory grounds with a highly competitive convolutional network PointCNN that is able to take 2,048 points as input. The reported values are for 3DIS, using the configuration in Table 2.
Table 6
3D semantic labelling on Scannet: All the techniques use 3D coordinates and color values as input features for network training.

| Method               | OA   | Acc  | mIoU | floor | wall | chair | sofa | table | door | cab | bed | desk | toilet | sink | wind | pic | bkshtf | curt | swtv | cnvr | fridg | bath | other |
|----------------------|------|------|------|-------|------|-------|------|-------|------|-----|-----|------|--------|------|------|-----|--------|------|------|------|-------|------|-------|
| ScanNet [18]         | 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++ [35]      | 33.9 | 67.7 | 52.3 | 36.0  | 34.6 | 23.2  | 26.1 | 25.6  | 27.8 | 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 [36, 60]    | 39.3 | 92.7 | 69.9 | 65.6  | 51.0 | 38.3  | 19.7 | 31.1  | 31.2 | 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 [48]    | 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 [57]        | 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 [31]       | 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 (Prop.)    | 61.0 | 93.5 | 77.3 | 79.2  | 70.5 | 54.9  | 50.7 | 55.2  | 77.2 | 57.0 | 85.9 | 60.2  | 53.4  | 4.6  | 48.9 | 64.3 | 70.2  | 70.4 | 51.0 | 85.8 | 41.4 |

Table 7
Performance on S3DIS dataset: Area 5 (top), all 6 folds (bottom). For the Area 5, SPH3D-GCN (9-dim) follows PointNet [33] to construct 9-dim input feature instead of 6-dim feature used by the proposed network.

| Methods              | Area 5 (All 6 Folds) | Area 5 (Prop.) | Office | mIoU= 94.4% | Ground truth | Proposed |
|----------------------|----------------------|----------------|--------|-------------|--------------|----------|
| PointNet [33]        | 90.1                 | 96.1           | 7.3    | 49.1        | 94.3         | 92.3     |
| SEGCloud [82]        | 94.4                 | 96.2           | 7.3    | 54.8        | 94.3         | 92.3     |
| Tangent-Conv [48]    | 94.4                 | 96.2           | 7.3    | 54.8        | 94.3         | 92.3     |
| SPC [63]             | 94.4                 | 96.2           | 7.3    | 54.8        | 94.3         | 92.3     |
| PointCNN [57]        | 94.4                 | 96.2           | 7.3    | 54.8        | 94.3         | 92.3     |
| SPH3D-GCN (9-dim)    | 94.4                 | 96.2           | 7.3    | 54.8        | 94.3         | 92.3     |
| SPH3D-GCN (Prop.)    | 94.4                 | 96.2           | 7.3    | 54.8        | 94.3         | 92.3     |

Fig. 5. Prediction visualization for two representative scenes of Area 5 in S3DIS dataset. Despite the scene complexity, the proposed SPH3D-GCN generally segments the points accurately.

for our network, where we vary the input point size. We show the memory consumption and training/testing time of our network. With a batch size 16 on 12GB GPU, our network can take point cloud of size up to 65,536, which is identical to the number of pixels in a 256 × 256 image. It is worth mentioning that the memory consumption of our ‘segmentation’ network for 32,678 input points is slightly lower than that of PointNet++ ‘classification’ network for 1,024 points (8.45GB vs. 8.57GB), using the same batch size, i.e. 16. Our 0.4M parameters are 10+ times less than the 4.4M of PointCNN. Considering that we use a larger batch size than the PointCNN, we include both the per-batch and per-sample training/testing time for a fair comparison. It can be seen that our per-sample running time for 2,048, 4,096, and 8,192 points is less than or comparable to that of PointCNN for 2,048 points. We refer to the websites [76, 77] for a speed comparison between Tesla P100 and Titan Xp. Although our SPH3D-GCN can take larger input size, we use point cloud of size 8,192 for S3DIS in Table 7 in the interest of time.

Graph coarsening visualization: We coarsen point cloud along our network with the Farthest Point Sampling (FPS) that reduces graph resolution layer-by-layer, similar to the image resolution reduction in the standard CNNs. We visualize the coarsening effects of the FPS in Fig. 6(top), using a chair from ModelNet40 as an example. The point clouds from left to right associate to the vertices of graphs visualized using a chair from ModelNet40 as an example.

TABLE 7
Performance on S3DIS dataset: Area 5 (top), all 6 folds (bottom). For the Area 5, SPH3D-GCN (9-dim) follows PointNet [33] to construct 9-dim input feature instead of 6-dim feature used by the proposed network.
not the 3D volume. Moreover, we also do not show the weight of self-loop.

7 CONCLUSION

We introduced separable spherical convolutional kernel for point clouds and demonstrated its utility with graph pyramid architectures. We built the graph pyramids with range search and farthest point sampling techniques. By applying the spherical convolution block to each graph resolution, the resulting graph convolutional networks are able to learn more effective features in larger contexts, similar to the standard CNNs. To perform the convolutions, the spherical kernel partitions its occupied space into multiple bins and associates a learnable parameter with each bin. The parameters are learned with network training. We down/upsample the vertex features of different graphs with pooling/unpooling operations. The proposed convolutional network is shown to be efficient in processing high resolution point clouds, achieving highly competitive performance on the tasks of classification and semantic segmentation on synthetic and large-scale real-world datasets.

ACKNOWLEDGMENTS

This research is supported by the Australian Research Council (ARC) grant DP190102443. The Titan Xp GPU used for this research is donated by the NVIDIA Corporation.

### References

1. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

3. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *International Conference on Learning Representations*, 2015.

4. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

5. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

6. J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

7. O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015, pp. 234–241.

8. V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

9. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

10. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSDL: Single shot detection,” in *European Conference on Computer Vision*, 2016, pp. 21–37.

11. S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

12. F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

13. S. Z. Gilani, A. Mian, F. Shafait, and I. Reid, “Dense 3d face correspondence,” *IEEE transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 7, pp. 1584–1598, 2018.

14. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

15. A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su et al., “ShapeNet: An information-rich 3D model repository,” arXiv preprint arXiv:1512.03385, 2015.

16. L. Yi, Y. G. Kim, D. Ceylan, J. Shen, M. Yan, H. Su, A. Lu, Q. Huang, A. Sheffer, L. Guibas et al., “A scalable active framework for region annotation in 3D shape collections,” *ACM Transactions on Graphics*, vol. 35, no. 6, p. 210, 2016.

17. I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, “3D semantic parsing of large-scale indoor
spaces,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1534–1543.

[18] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, “ScanNet: Richly-annotated 3d reconstructions of indoor scenes,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5828–5839.

[19] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, “Semantic3D.net: A new large-scale point cloud classification benchmark,” ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. IV-1-W1, pp. 91–98, 2017.

[20] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3D ShapeNets: A deep representation for volumetric shapes,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1912–1920.

[21] D. Maturana and S. Scherer, “VoxNet: A 3D convolutional neural network for real-time object recognition,” in IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2015, pp. 922–928.

[22] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3D shape recognition,” in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 945–953.

[23] G. Riegler, A. Osman Ulusoy, and A. Geiger, “OctNet: Learning deep 3D representations at high resolutions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3577–3586.

[24] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” in International Conference on Learning Representations, 2014.

[25] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on 3d point clouds,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2282–2290.

[26] M. Simonovsky and N. Komodakis, “Dynamic edge-conditioned filters in convolutional neural networks on graphs,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[27] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point clouds,” arXiv preprint arXiv:1801.07829, 2018.

[28] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan, “Graph attention convolution for point cloud semantic segmentation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10296–10305.

[29] W. Wu, Z. Qi, and L. Fuxin, “Pointconv: Deep convolutional networks on 3d point clouds,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 9621–9630.

[30] H. Lei, N. Akhtar, and A. Mian, “Octree guided cnn with spherical kernels for 3d point clouds,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 9631–9640.

[31] C. R Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3d classification and segmentation,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 652–660, 2017.

[32] R. Klokov and V. 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. IEEE, 2017, pp. 863–872.

[33] C. R Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep hierarchical feature learning on point sets in a metric space,” Advances in Neural Information Processing Systems, 2017.

[34] J. Li, B. M. Chen, and G. H. Lee, “So-net: Self-organizing network for point cloud analysis,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9397–9406.

[35] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “Pointcnn: Convolution on x-transformed points,” in Advances in Neural Information Processing Systems, 2018, pp. 820–830.

[36] F. P. Preparata and M. I. Shamos, Computational geometry: an introduction. Springer Science & Business Media, 2012.

[37] H. Riemeneschneider, A. Bódis-Szomoru, J. Weissenberg, and L. Van Gool, “Learning where to classify in multi-view semantic segmentation,” in European Conference on Computer Vision, 2014, pp. 516–532.

[38] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.

[39] Y. Shen, C. Feng, Y. Yang, and D. Tian, “Mining point cloud local structures by kernel correlation and graph pooling,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 4, 2018.

[40] J. Huang and S. You, “Point cloud labeling using 3D convolutional neural network,” in International Conference on Pattern Recognition, 2016, pp. 2670–2675.

[41] N. Sedaghat, M. Zolfaghari, and T. Brox, “Orientation-boosted voxel nets for 3D object recognition,” arXiv preprint arXiv:1604.03531, 2016.

[42] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, “3DMatch: Learning local geometric descriptors from RGB-D reconstructions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 199–208.

[43] Y. Zhang, M. Bai, P. Kohli, S. Izadi, and J. Xiao, “Deeppointnet: Context-encoding neural pathways for 3D holistic scene understanding,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1192–1201.

[44] 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.

[45] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, “FPFN: Field probing neural networks for 3D data,” in Advances in Neural Information Processing Systems, 2016, pp. 307–315.

[46] M. Tatschekno, J. Park, V. Koltun, and Q.-Y. Zhou, “Tangent convolutions for dense prediction in 3d,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 3887–3896.

[47] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, “SPLATNet: Sparse lattice networks for point cloud processing,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2530–2539.

[48] M. Golai, G. Monfardini, and F. Scarselli, “A new model for learning in graph domains,” in International Joint Conference on Neural Networks, vol. 2, 2005, pp. 729–734.

[49] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61–80, 2009.

[50] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” International Conference on Learning Representations, 2016.

[51] X. Q. R. Liao, J. Jia, S. Figler, and R. Urtasun, “3D graph neural networks for RGB-D semantic segmentation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2282–2290.

[52] X. Ye, J. Li, H. Huang, L. Du, and X. Zhang, “3D recurrent neural networks with context fusion for point cloud semantic segmentation,” in Proceedings of the European Conference on Computer Vision, 2018, pp. 403–417.

[53] M. Jaderberg, K. Simonyan, A. Zisserman et al., “Spatial transformer networks,” in Advances in Neural Information Processing Systems, 2015, pp. 2017–2025.

[54] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, “Dynamic filter networks,” in Advances in Neural Information Processing Systems, 2016.

[55] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan, “Graph attention convolution for point cloud semantic segmentation,” in The IEEE Conference on Computer Vision and Pattern Recognition, June 2019.

[56] F. Grohs, P. Wieschollek, and H. P. Lensch, “Flex-convolution,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 5199–5208.
[61] O. Dovrat, I. Lang, and S. Avidan, “Learning to sample,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2760–2769.

[62] Y. Rao, J. Lu, and J. Zhou, “Spherical fractal convolutional neural networks for point cloud recognition,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 452–460.

[63] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

[64] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, 2005, pp. 886–893.

[65] A. Frome, D. Huber, R. Kolluri, T. Bulo, and J. Malik, “Recognizing objects in range data using regional point descriptors,” European Conference on Computer Vision, pp. 224–237, 2004.

[66] F. Tombari, S. Salti, and L. Di Stefano, “Unique shape context for 3D data description,” in Proceedings of the ACM workshop on 3D object retrieval. ACM, 2010, pp. 57–62.

[67] ——, “Unique signatures of histograms for local surface description,” in European Conference on Computer Vision, 2010, pp. 356–369.

[68] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, “Spherical CNNs,” in International Conference on Learning Representations, 2018.

[69] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units,” International Conference on Learning Representations, 2016.

[70] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, “FlowNet: Learning optical flow with convolutional networks,” in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2758–2766.

[71] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., “TensorFlow: A system for large-scale machine learning,” in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI) 16, 2016, pp. 265–283.

[72] A. Vedaldi and K. Lenc, “Matconvnet: Convolutional neural networks for matlab,” in Proceedings of the 23rd ACM international conference on Multimedia. ACM, 2015, pp. 689–692.

[73] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in International Conference on Machine Learning, 2015.

[74] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” International Conference on Learning Representations, 2015.

[75] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.

[76] “FP32 Throughput Comparison,” http://www.cs.toronto.edu/~pekimenko/tbd/throughput.html accessed 23-Aug-2019.

[77] Z. Wu, R. Shou, Y. Wang, and X. Liu, “Interactive shape co-segmentation via label propagation,” Computers & Graphics, vol. 36, pp. 248–254, 2014.

[78] A. Marti, B. Knepp, H. Riemenschneider, and L. Van Gool, “3D all the way: Semantic segmentation of urban scenes from start to end in 3D,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 4456–4465.

[79] R. Gadde, V. Jampani, R. Marlet, and P. F. Gehler, “Efficient 2D and 3D facade segmentation using auto-context,” IEEE transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 5, pp. 1273–1280, 2018.

[80] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in IEEE International Conference on Robotics and Automation, 2011, pp. 1–4.

[81] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese, “Segcloud: Semantic segmentation of 3d point clouds,” in International Conference on 3D Vision. IEEE, 2017, pp. 537–547.

[82] L. Landrieu and M. Simonovsky, “Large-scale point cloud semantic segmentation with superpoint graphs,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018.

[83] F. Engelmann, T. Kontogianni, A. Herrmans, and B. Leibe, “Exploring spatial context for 3d semantic segmentation of point clouds,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 716–724.