Octree guided CNN with Spherical Kernels for 3D Point Clouds

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

Abstract

We propose an octree guided neural network architecture and spherical convolutional kernel for machine learning from arbitrary 3D point clouds. The network architecture capitalizes on the sparse nature of irregular point clouds, and hierarchically coarsens the data representation with space partitioning. At the same time, the proposed spherical kernels systematically quantize point neighborhoods to identify local geometric structures in the data, while maintaining the properties of translation-invariance and asymmetry. We specify spherical kernels with the help of network neurons that in turn are associated with spatial locations. We exploit this association to avert dynamic kernel generation during network training that enables efficient learning with high resolution point clouds. The effectiveness of the proposed technique is established on the benchmark tasks of 3D object classification and segmentation, achieving competetive performance on ShapeNet and RueMonge2014 datasets.

1. Introduction

Convolutional Neural Networks (CNNs) [17] are known to learn highly effective features from data. However, standard CNNs are only amenable to data defined over regular grids, e.g. pixel arrays. This limits their ability in processing 3D point clouds that are inherently irregular. Point cloud processing has recently gained significant research interest and large repositories for this data modality have started to emerge [1, 4, 12, 39, 40]. Recent literature has also seen many attempts to exploit the representation prowess of standard convolutional networks for point clouds by adaption [23, 39]. However, these attempts have often led to excessively large memory footprints that restrict the allowed input data resolution [29, 33]. A more attractive choice is to combine the power of convolution operation with graph representations of irregular data. The resulting Graph Convolutional Networks (GCNs) offer convolutions either in spectral domain [3, 7, 15] or spatial domain [33].

In GCNs, the spectral domain methods require the Graph Laplacian to be aligned, which is not straight forward to achieve for point clouds. On the other hand, the only prominent approach in spatial domain is the Edge Conditioned filters in CNNs for graphs (ECC) [33] that, in contrast to the standard CNNs, must generate convolution kernels dynamically entailing a significant computational overhead. Additionally, ECC relies on range searches for graph construction and coarsening, which can become prohibitively expensive for large point clouds. One major challenge in applying convolutional networks to irregular 3D data is in specifying geometrically meaningful convolution kernels in the 3D metric space. Naturally, such kernels are also required to exhibit translation-invariance to identify similar local structures in the data. Moreover, they should be applied to point pairs asymmetrically for a compact representation. Owing to such intricate requirements, few existing techniques altogether avoid the use of convolution kernels in computational graphs to process unstructured data [16, 27, 28]. Although still attractive, these methods do not contribute towards harnessing the potential of convolutional neural networks for point clouds.

In this work, we introduce the notion of spherical convolutional kernel that systematically partitions a spherical 3D region into multiple volumetric bins, see Fig. 1. Each bin of the kernel specifies a matrix of learnable parameters that weights the points falling within that bin for convolution. We apply these kernels between the layers of a Neural Network (Ψ-CNN) that we propose to construct by exploiting octree partitioning [24] of the 3D space. The sparsity guided octree structuring determines the locations to perform the convolutions in each layer of the network. The network architecture itself is guided by the hierarchy of the octree, having the same number of hidden layers as the tree depth. By exploiting space partitioning, the network avoids K-NN/range search and efficiently consumes high resolution point clouds. It also avoids dynamic generation of the proposed kernels by associating them to its neurons. At the same time, the kernels are able to share weights between similar local structures in the data. We theoretically establish that the spherical kernels are applied asymmetrically to points in our network just as the kernels in standard CNNs

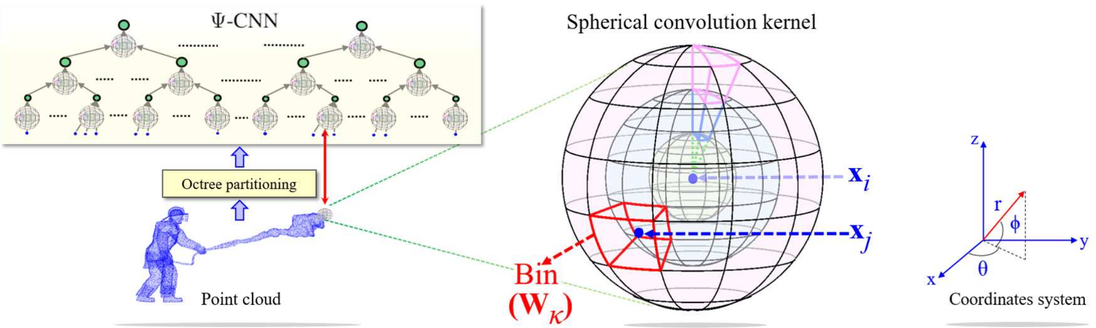

Figure 1. The proposed octree guided CNN, i.e. Ψ-CNN directly processes raw point clouds using octree partitioning information. The representation is hierarchically coarsened at each network layer (three layers depicted) by applying spherical convolutional kernels. A spherical kernel systematically splits the space around a point $\mathbf{x}_i$ into multiple volumetric bins. For the j^{th} neighboring point $\mathbf{x}_j$, a kernel first determines its relevant bin and uses the weight matrix $\mathbf{W}_\kappa$ defined for that bin to compute the activation value. The proposed spherical kernel preserves translation-invariance and asymmetry properties of standard 2D convolutional kernel in 3D point cloud domain.

are applied asymmetrically to image pixels. This ensures compact representation learning by the proposed network in the point cloud domain. We demonstrate the effectiveness of our method for 3D object classification, part segmentation and large-scale semantic segmentation. The major contributions of this work are summarized below:

- A novel concept of translation-invariant and asymmetric convolutional kernel is proposed and analyzed for point-wise feature learning from irregular point clouds.
- The resulting convolutional kernel is exploited with an octree guided neural network that, in contrast to the previous voxelization applications of octree to point clouds [29], hierarchically coarsens the data and constructs point neighborhoods using space partitioning to avoid time-consuming K-NN/range search.
- Efficacy of the proposed technique is established by experiments with ModelNets [39] for 3D object classification, ShapeNet [40] for part segmentation, and RueMonge2014 [30] for semantic segmentation, achieving competetive performance on the last two.

2. Related Work

PointNet [27] is one of the first instances of exploiting neural networks to represent point clouds. It directly uses x, y, z-coordinates of points as input features. The network learns point-wise features with shared MLPs, and extracts a global feature with max pooling. A major limitation of PointNet is that it explores no geometric context in point-wise feature learning. This was later addressed by PointNet++ [28] with hierarchical application of max-pooling to the local regions. The enhancement builds local regions using K-NN search as well as range search. Nevertheless, both PointNets [27, 28] aggregate the context information with max pooling, and no convolution modules are explored in the networks. In regards to processing point clouds with deep learning using tree structures, Kd-network [16] is among the pioneering prominent contributions. Kd-network also uses point coordinates as its input, and computes feature of a parent node by concatenating the features of its children in a *balanced* tree. However, its performance depends heavily on the randomization of the tree construction. This is in sharp contrast to our approach that uses deterministic geometric relationships between the points. Another technique, SO-Net [18] reorganizes the irregular point cloud into an $m \times m$ 2D rectangular map, and uses the PointNet architecture to learn node-wise features for the map. Similarly, KCNet [32] also builds on PointNet and introduces a point-set template to learn geometric correlations of local points in the point cloud. PointCNN [19] extracts permutation-invariant features by reordering the local points canonically with a learnable χ-transformation. All of these methods relate to our work in terms of directly accepting the spatial coordinates of points as input. However, they do not contribute towards the use of convolutional networks for processing 3D point clouds. Approaches advancing that research direction can be divided into two broad categories, discussed below.

A. Graph Convolutional Networks

Graph convolutional networks can be grouped into spectral networks [3, 7, 15] and spatial networks [33]. The spectral networks perform convolution in the spectral domain relying on the graph Laplacian and adjacency matrices, while the spatial networks perform convolution in the spatial domain. A major limitation of spectral networks is that they demand 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.* [41] attempted to address this issue with Spectral Transformer Network (SpecTN), similar to STN [14] in the spatial domain. However, the signal transformation from spatial to spectral domains and vice-versa results in computational complexity $\mathcal{O}(n^2)$. ECC [33] is among the pioneering works for point cloud analysis with graph convolution in the spatial domain.

Inspired by the dynamic filter networks [6], it adapts MLPs to generate convolution filters between the connected vertices dynamically. The dynamic generation of filters comes with computational overhead. Additionally, the neighborhood construction and graph coarsening in ECC must rely on range searches, which is not efficient. We achieve coarsening and neighborhood construction directly from the octree partitioning, thereby avoiding expensive range searching. Moreover, our spherical convolutional kernel effectively explores the geometric context of each point without requiring dynamic filter generation.

B. 3D Convolutional Neural Networks

3D-CNNs are applied to volumetric representations of 3D data. In the earlier attempts in this direction, only low input resolution could be processed, e.g. 30×30×30 [39], 32×32×32 [23]. This issue transcended to subsequent works as well [13, 31, 42, 43]. The limitation of low input resolution was a natural consequence of the cubic growth of memory and computational requirements associated with the volumetric input data. Later methods [8, 20] mainly aim at addressing these issues. Most recently, Riegler *et al.* [29] proposed OctNet, that 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 256×256×256. Whereas OctNet also exploits octrees, there are major differences between OctNet and our method. Firstly, OctNet must process point clouds as regular 3D volumes due to its 3D-CNN kernels. No such constraint is applicable to our technique due to the proposed spherical kernels. Secondly, we are able to learn point cloud representation with a single deep octree instead of using hybrid of shallow trees.

3. Spherical Convolutional Kernel

Our network derives its main strength from spherical convolutional kernels. Thus, it is imperative to first understand the proposed kernel before delving into the network details. This section explains our convolutional kernel for 3D point cloud processing.

For images, hand-crafted features have traditionally been computed over more primitive constituents, i.e. patches. In effect, the same principle transcended to automatic feature extraction with the standard CNNs that compute feature maps using the activations of well-defined rectangular regions. Whereas rectangular regions are a common choice to process data of 2D nature, 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 [9, 34, 35]. Inspired by this natural kinship, we introduce the concept of *spherical convolutional kernel*[1] that uses a 3D sphere as the basic geometric shape to perform the convolution.

Given an arbitrary point cloud $\mathcal{P} = \{\mathbf{x}_i \in \mathbb{R}^3\}_{i=1}^m$, where m is the number of points; we define the convolution kernel with the help of a sphere of radius $\rho \in \mathbb{R}^+$. For a target point $\mathbf{x}_i$, we consider its neighborhood $\mathcal{N}(\mathbf{x}_i)$ to comprise the points within the sphere centered at $\mathbf{x}_i$, i.e. $\mathcal{N}(\mathbf{x}_i) = \{\mathbf{x} : d(\mathbf{x}, \mathbf{x}_i) \leq \rho\}$, where $d(.,.)$ is a distance metric - ℓ_2 distance in this work. We divide the sphere into $n \times p \times q$ *'bins'* (see Fig. 1) by partitioning the occupied space uniformly along the azimuth (θ) and elevation (ϕ) dimensions. We allow the partitions along the radial dimension to be non-uniform because cubic volume growth for large radius values can become undesirable. Our quantization of the spherical region is mainly inspired by 3DSC [9]. We also define an additional bin corresponding to the origin of the sphere to allow the case of self-convolution of points. For each bin, we define a weight matrix $\mathbf{W}_{\kappa \in \{0,1,\dots,n \times p \times q\}} \in \mathbb{R}^{s \times t}$ of learnable parameters, where s-t are the number of output-input channels and $\mathbf{W}_0$ relates to self-convolution. Together, the $n \times p \times q + 1$ weight matrices specify a single spherical convolutional kernel.

To compute the activation value for a target point $\mathbf{x}_i$, we must identify the relevant weight matrices of the kernel for each of its neighboring points $\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i)$. It is straightforward to associate $\mathbf{x}_i$ with $\mathbf{W}_0$ for self-convolution. For the non-trivial cases, we first represent the neighboring points in terms of their spherical coordinates that are referenced using $\mathbf{x}_i$ as the origin. That is, for each $\mathbf{x}_j$ we compute $\mathcal{T}(\boldsymbol{\Delta}_{ji}) \rightarrow \boldsymbol{\psi}_{ji}$, where $\mathcal{T}(.)$ defines the transformation from Cartesian to Spherical coordinates and $\boldsymbol{\Delta}_{ji} = \mathbf{x}_j - \mathbf{x}_i$. Assuming that the bins of the quantized sphere are indexed by k_θ, k_ϕ and k_r along the azimuth, elevation and radial dimensions respectively, the weight matrices associated with the spherical kernel bins can be indexed as $\kappa = k_\theta + (k_\phi - 1) \times n + (k_r - 1) \times n \times p$, where $k_\theta \in \{1, \dots, n\}$, $k_\phi \in \{1, \dots, p\}$, $k_r \in \{1, \dots, q\}$. Using this indexing, we assign each $\boldsymbol{\psi}_{ji}$; and hence $\mathbf{x}_j$ to its relevant weight matrix. In the l^{th} network layer, the activation for the i^{th} point can then be computed as:

$$\mathbf{z}_i^l = \frac{1}{|\mathcal{N}(\mathbf{x}_i)|} \sum_{j=1}^{|\mathcal{N}(\mathbf{x}_i)|} \mathbf{W}_\kappa^l \mathbf{a}_j^{l-1} + \mathbf{b}^l, \tag{1}$$

$$\mathbf{a}_i^l = f(\mathbf{z}_i^l), \tag{2}$$

where $\mathbf{a}_j^{l-1}$ is the activation value of a neighboring point from layer $l-1$, $\mathbf{W}_\kappa^l$ is the weight matrix, $\mathbf{b}^l$ is the bias vector, and $f(\cdot)$ is the non-linear activation function - ReLU [25] in our experiments.

To elaborate on the characteristics of the proposed spherical convolutional kernel, let us denote the *edges* of the ker-

[1] Note that the term *spherical* in Spherical CNN [5] is used for spherical surfaces (i.e. 360° images) not the ambient 3D space. Our concept of spherical kernel widely differs from [5], and it is used in different context.

nel bins along θ, ϕ and r dimensions respectively as:

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

Due to the constraint of uniform splitting along the azimuth and elevation, we can write $\Theta_{k+1} - \Theta_k = \frac{2\pi}{n}$ and $\Phi_{k+1} - \Phi_k = \frac{\pi}{p}$.

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 $\mathbf{x}_a \neq \mathbf{x}_b$ within the spherical convolutional kernel, the weight matrices $\mathbf{W}_\kappa, \forall \kappa > 0$, are applied asymmetrically.*

Proof: Let $\mathbf{\Delta}_{ab} = \mathbf{x}_a - \mathbf{x}_b = [\delta_x, \delta_y, \delta_z]^\intercal$, then $\mathbf{\Delta}_{ba} = [-\delta_x, -\delta_y, -\delta_z]^\intercal$. Under the Cartesian to Spherical coordinate transformation, we have $\mathcal{T}(\mathbf{\Delta}_{ab}) = \boldsymbol{\psi}_{ab} = [\theta_{ab}, \phi_{ab}, r]^\intercal$, and $\mathcal{T}(\mathbf{\Delta}_{ba}) = \boldsymbol{\psi}_{ba} = [\theta_{ba}, \phi_{ba}, r]^\intercal$. We assert that $\boldsymbol{\psi}_{ab}$ and $\boldsymbol{\psi}_{ba}$ fall in the same bin indexed by $\kappa \leftarrow (k_\theta, k_\phi, k_r)$, i.e. $\mathbf{W}_\kappa$ is applied symmetrically to the points $\mathbf{x}_a$ and $\mathbf{x}_b$. In that case, under the inverse transformation $\mathcal{T}^{-1}(.)$, we have $\delta_z = r\sin\phi_{ab}$ and $(-\delta_z) = r\sin\phi_{ba}$. The condition $\Phi_{k_\phi} \cdot \Phi_{k_\phi+1} \geq 0$ entails that $-\delta_z^2 = \delta_z \cdot (-\delta_z) = (r\sin\phi_{ab}) \cdot (r\sin\phi_{ba}) = r^2(\sin\phi_{ab}\sin\phi_{ba}) \geq 0 \implies \delta_z = 0$. Similarly, $\Theta_{k_\theta} \cdot \Theta_{k_\theta+1} \geq 0 \implies \delta_y = 0$. Since $\mathbf{x}_a \neq \mathbf{x}_b$, for $\delta_x \neq 0$ we have $\cos\theta_{ab} = -\cos\theta_{ba} \implies |\theta_{ab} - \theta_{ba}| = \pi$. However, if θ_{ab}, θ_{ba} fall into the same bin, we have $|\theta_{ab} - \theta_{ba}| = \frac{2\pi}{n} < \pi$, which entails $\delta_x = 0$. Thus, the assertion can not hold, and $\mathbf{W}_\kappa$ can not be applied to any two points symmetrically unless both points are the same.

The asymmetry property of the spherical kernel is significant because it restricts the sharing of the same weights between point pairs, which facilitates in learning more effective features with finer geometric details. Lemma 2.1 also provides guidelines for the division of the convolution kernel into bins such that the asymmetry is always preserved. To elaborate further on this aspect, we provide few examples of kernel divisions that violate asymmetry in the supplementary material of the paper. Note that asymmetric application of kernel weights to pixels comes naturally in standard CNN kernels. However, the proposed kernel is able to ensure the same property in the point cloud domain.

Relation to 3D-CNN: Here, we briefly relate the proposed notion of spherical kernel to the existing techniques that exploit CNNs for 3D data. Pioneering works in this direction rasterize the raw data into uniform voxel grids, and then extract features using 3D-CNNs from the resulting volumetric representations [23, 39]. In 3D-CNNs, the convolution kernel of size $3 \times 3 \times 3 = 27$ is prevalently used, that splits the space in 1 cell/voxel for radius $r = 0$ (self-convolution); 6 cells for radius $r = 1$; 12 cells for radius $r = \sqrt{2}$; and 8 cells for radius $r = \sqrt{3}$. An analogous spherical convolution kernel for the same region can be specified with a radius $\rho = \sqrt{3}$, using the following edges for the bins:

$$\begin{aligned}
&\mathbf{\Theta} = [-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi];\\
&\mathbf{\Phi} = [-\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}];\\
&\mathbf{R} = [\epsilon, 1, \sqrt{2}, \rho], \epsilon \to 0^+.
\end{aligned} \tag{3}$$

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

Notice that, if we move radially from the center to the periphery of spherical kernel, we encounter identical number of bins (16 in this case) after each edge defined by $\mathbf{R}$, 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 3D-kernels that must keep the size of all cells constant and rely on increased input resolution of the data to capture finer details - generally entailing memory issues. The multi-scale granularity of spherical kernel makes it a natural choice for raw point clouds.

4. Neural Network

Most of the existing attempts to process point clouds with neural networks [18, 19, 28, 32, 33] rely on K-NN or range searches to define local neighborhood of points, that are subsequently used to perform operations like convolution or pooling. However, to process large point clouds, these search strategies become computationally prohibitive. For unstructured data, an efficient mechanism to define point neighbourhood is tree-structuring, e.g. Kd-tree [2]. The hierarchical nature of tree structures also provide guidelines for neural network architectures that can be used to process the point cloud. More importantly, a tree-structured data also possess the much desired attributes of permutation and translation invariance for neural networks.

A. Core Architecture

We exploit octree structuring [24] of point clouds and design a neural network based on the resulting trees. Our choice of using octree comes from its amenability to neural networks as the base data structure [29], and its ability to account for more data in point neighborhoods compared to, for example, Kd-tree. We illustrate 3D space partitioning under octree, the resulting tree, and the formation of neural network using the proposed strategy of network construction in Fig. 2 for a toy example. For an input point cloud $\mathcal{P}$, we construct an octree of depth L ($L = 3$ in the figure). In the construction, the splitting of nodes is fixed to use a maximum capacity of one point, with the exception of the last layer leaf nodes. The point in a parent node is computed as the Expected value of the points in its children. The allocation of multiple points in the last layer nodes directly results from the allowed finest partitioning of the space. For the sub-volumes in 3D space that are not densely populated,

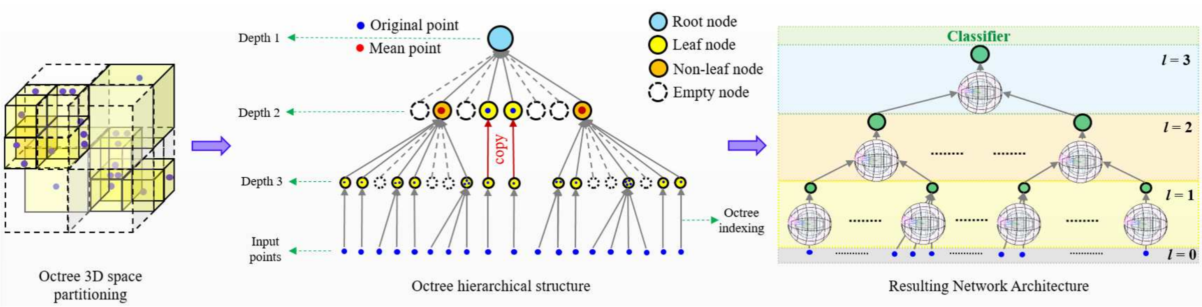

Figure 2. Illustration of octree guided network architecture using a toy example: The point cloud in 3D space is partitioned under an octree of depth 3. The corresponding tree representation allocates points to nodes at the maximum depth based on the space partitioning, and computes the location of each parent node as the Expected location of its children. Leaf nodes on shallow branches are replicated to match the maximum depth. The corresponding neural network has the same number of hidden layers as tree depth, and it learns spherical convolutional kernels for feature extraction.

our splitting strategy can result in leaf nodes before the tree reaches its maximum depth. In such cases, to facilitate mapping of the tree to a neural network, we replicate the leaf nodes to the maximum depth of the tree. We safely ignore the empty nodes while implementing the network, resulting in computational and memory benefits.

Based on the hierarchical tree structure, our neural network also has L hidden layers. Notice that, in Fig. 2 we use $l = 1$ for the first hidden layer that corresponds to Depth $= 3$ for the tree. We will use the same convention in the text to follow. For each non-empty node in the tree, there is a corresponding neuron in our neural network. Recall that, a spherical convolutional kernel is specified with a target point over whose neighborhood the convolution is performed. Therefore, to facilitate convolutions, we associate a single 3D point with each neuron, except for the leaf nodes at the maximum depth of the tree. For a leaf node, the associated point is the mean value of data points allocated to that node. A neuron uses its associated point/location to select the appropriate spherical kernel and later applies the non-linear activation (not shown in Fig. 2). In our network, all convolution layers before the last layer are followed by batch normalization and ReLU activations.

We denote the location associated with the i^{th} neuron in the l^{th} layer of the network as $\bar{\mathbf{x}}_i^l$. From $l = 1$ to $l = L$, we can represent the locations associated with all neurons as $\mathcal{Q}^1 = \{\bar{\mathbf{x}}_i^1\}_{i=1}^{m_1}, \ldots, \mathcal{Q}^L = \{\bar{\mathbf{x}}_1^L\}_{i=1}^{m_L}$. Denoting the raw input points as $\mathcal{Q}^0 = \{\bar{\mathbf{x}}_i^0\}_{i=1}^{m_0}$, $\bar{\mathbf{x}}_i^l$ is numerically computed by our network as:

$$\bar{\mathbf{x}}_i^l = \frac{\sum\limits_{\bar{\mathbf{x}}_j^{l-1}\in\mathcal{N}(\bar{\mathbf{x}}_i^l)} \bar{\mathbf{x}}_j^{l-1}}{|\mathcal{N}(\bar{\mathbf{x}}_i^l)|}, \qquad (4)$$

where $\mathcal{N}(\bar{\mathbf{x}}_i^l)$ contains locations of the relevant children nodes in the octree. It is worth noting that the strategy used for specifying the network layers also entails that $|\mathcal{Q}^{l-1}| > |\mathcal{Q}^l|$. Thus, from the first layer to the last, the features learned by our network move from lower to higher level of abstraction similar to the standard CNNs.

In relating the spherical nature of point neighborhood considered in our network to the cubic partitioning of space by octree, a subtle detail is worth considering. Say $\mathbf{x}_{\text{min}} = [x_{\text{min}}, y_{\text{min}}, z_{\text{min}}]^\intercal$, and $\mathbf{x}_{\text{max}} = [x_{\text{max}}, y_{\text{max}}, z_{\text{max}}]^\intercal$ determine the range of point coordinates in a given cubic volume resulting from our space partitioning. The spherical neighborhood associated with a neuron in the l^{th} layer is defined with the radius $\rho = 2^{l-L-1}||\mathbf{x}_{\text{max}} - \mathbf{x}_{\text{min}}||_2$. This neighbourhood may not strictly circumscribe all points of the corresponding cubic volume at this level due to shape dissimilarity. Although the number of such points is minuscule in practice, we still take those into account by assigning them to the outer-most bins of our kernels based on their azimuth and elevation values.

Our neural network performs inter-layer convolutions instead of intra-layer convolutions. This drastically reduces the operations required to process large point clouds when compared with graph-based networks [3, 7, 15, 33, 41]. We note that for all nodes with a single child, only self-convolutions are performed in the network. Note that due to its unconventional nature, spherical convolutional kernel is not readily implemented using the existing deep learning libraries, e.g. matconvnet [36]. Therefore, we implement it ourselves with CUDA C++ and mex interface[2]. For the other modules such as ReLU, batch normalization etc., we use matconvnet. See Sec. E of the supplementary material to understand the spherical kernel in a conventional way.

Comparison to OctNet [29]: OctNet [29] also makes use of octree structure. However, OctNet processes point clouds as regular 3D volumes - a 3D-CNN. In contrast, we process point clouds following their unstructured nature. Our network learns features for each point in the sets from $\mathcal{Q}^0$ to $\mathcal{Q}^L$, which is in contrast to OctNet that must account for occupied and unoccupied voxels, entailing complexity. We exploit octree structure to simultaneously construct neighborhoods of all points and coarsen the original point cloud layer-by-layer, while OctNet uses this structure to voxelize

[2] The implementation will be made public.

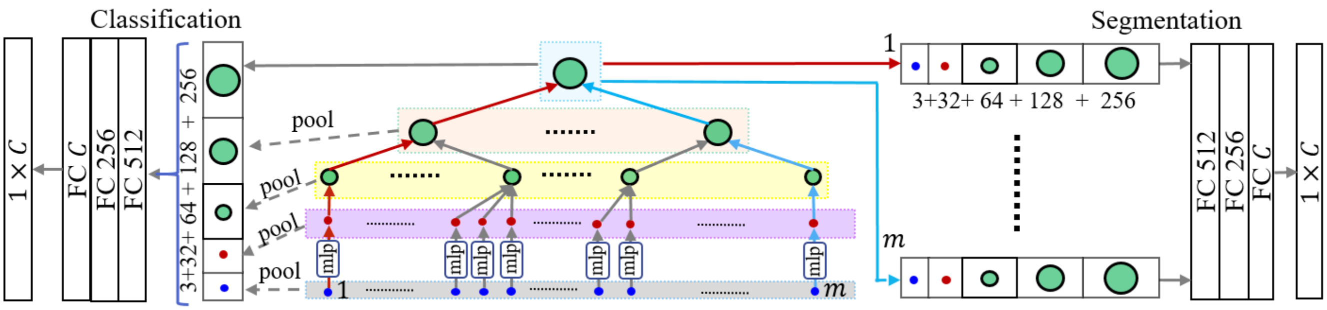

Figure 3. Classification and segmentation using the core network of Fig. 2. For classification, the features at the root node (top layer) are concatenated with the max-pooled (dashed lines) features at the remaining layers followed by FC layers. For segmentation, the representation of a point uses the layer-level features of all the ancestors along the path to the root node, e.g. red path for point '1' and blue path for point 'm'. Point-wise classification (segmentation) is performed using the concatenated raw point features ($xyz/xyz-rgb$), the MLP features and all the extracted layer-level features. A simple configuration MLP(32)-Octree(64-128-256) is shown for illustration.

the point cloud into different resolutions.

B. Classification and Segmentation

The classification and segmentation networks are basically variants of the same core architecture shown in Fig 2. However, we additionally insert an MLP layer prior to the octree structure to obtain more expressive point-wise features. This concept is inspired from Kd-Net [16]. Figure 3 shows the complete architectures for classification and segmentation. To fully exploit the hierarchical features learned at different octree levels, we use features from all octree layers. For classification, we max pool the features from intermediate layers, including the raw features, and concatenate them with the features at the root node to form a global representation of the complete point cloud. For segmentation, we need point wise features. The feature of each point is the concatenation of raw features, MLP features and layer-wise features without any pooling. The final classification or segmentation is performed using three fully connected layers.

5. Experiments

We conduct experiments on clean CAD Models as well as noisy point clouds to evaluate the performance of our method for the tasks of 3D object classification, part segmentation and semantic segmentation. Throughout the experiments, we keep the size of our convolution kernel fixed to $8 \times 2 \times 3 + 1$, in which the radial dimension is split uniformly. We use three fully connected layers (512-256-C) followed by softmax as the classifier for both the classification and segmentation tasks. Here, C denotes the number of classes/parts. The training of our network is conducted using a Titan Xp GPU with 12 GB memory. We use Stochastic Gradient Descent with momentum to train the network. The batch size is kept fixed to 16 in all our experiments. These hyper-parameters were empirically optimized using cross-validation. We use only the (x, y, z) coordinates of points provided by point clouds to train our network, and the (r, g, b) values when the color information is provided. Few existing methods in the literature also take advantage of normals, and use them as input features. However, normals are not directly sensed by 3D sensors and must be computed using the point coordinates. This also entails additional computational burden. Hence, we avoid using normals as input features. In our experiments, we follow the standard practice of taking advantage of data augmentation. To that end, we used random sub-sampling of the original point clouds, performed random azimuth rotation (up to $\frac{\pi}{6}$ rad) and also applied noisy translation (std. dev = 0.02) to increase the number of training examples. These operations were performed on the fly in each training epoch of the network.

A. Classification

We use the benchmark datasets ModelNet10 and ModelNet40 [39] to evaluate our technique for the classification task. These datasets are created using clean CAD models. ModelNet10 contains 10 categories of object meshes, and the samples are split into 3,991 training examples and 908 test instances. ModelNet40 contains object meshes for 40 categories with 9,843/2,468 training/testing split.

Compared to existing works (e.g. [27, 28, 32, 33]), the convolutions performed in our network allow the proposed method to consume large input point clouds. Hence, we train our network using 10K input points. For the classification task, we adopted a network with 6 levels of octree, whereas the number of feature channels are kept MLP(32)-Octree(64-64-64-128-128-128). The network comprises two components, octree based architecture for feature extraction and classification stage. We train the whole network in an end-to-end fashion. We standardize the input models by normalizing the 3D point clouds to fit into a cube of $[-1, 1]^3$ with zero mean.

Table 1 benchmarks the object classification performance of our approach that is abbreviated as Ψ-CNN[3]. Our method uses xyz coordinates of points as raw features to achieve these results. As can be seen, Ψ-CNN consistently

[3] A Greek alphabet is chosen as prefix to avoid duplication with other OCNNs and SCNNs, e.g. [21, 26, 37].

Table 1. Classification performance on ModelNets [39].

Method	ModelNet10		ModelNet40	
	class	instance	class	instance
OctNet [29]	90.1	90.9	83.8	86.5
ECC [33]	90.0	90.8	83.2	87.4
PointNet [27]	–	–	86.2	89.2
PointNet++ [28]	–	–	–	90.7
Kd-Net [16]	92.8	93.3	86.3	90.6
SO-Net [18]	93.9	94.1	87.3	90.9
KCNet [32]	–	94.4	–	91.0
Ψ-CNN	**94.4**	**94.6**	**88.7**	**92.0**

achieves the best performance on ModelNets. We note that, like our method Kd-Net [16] and OctNet [29] are also tree structure based networks. However, they require twice the number of parametric layers as required by our method to achieve the reported performance. This is a direct consequence of effective exploration of geometric information by the proposed kernel. We also provide an ablation study to support this in the supplementary material of the paper.

B. Part Segmentation

ShapeNet part segmentation dataset [40] contains 16,881 CAD models from 16 categories. The models in each category have two to five annotated parts, amounting to 50 parts in total. The point clouds are created with uniform sampling from 3D meshes. This dataset provides xyz coordinates of the points as raw features, and has 14007/2874 training/testing split defined. We use a 6-level octree for the segmentation network, with configuration MLP(64)-Octree(128-128-256-256-512-512). The output class number C of the classifier is determined by the number of parts in each category. We use the part-averaged IoU (mIoU) proposed in [27] to report the performance in Table 2. Similar to the classification task, we also standardize the input models of ShapeNet by normalizing input point clouds to $[-1, 1]^3$ cube with zero mean.

In Table 2, we compare our results with the popular methods that also take irregular point clouds as input. Yet, to achieve their results, some of these methods exploit normals besides xyz coordinates as input features, e.g. PointNet, PointNet++, SO-Net. It can seen that Ψ-CNN not only achieves the highest mIoU 86.8%, but also outperforms the other approaches on 11 out of 16 categories. To the best of our knowledge, Ψ-CNN records the new state-of-the-art performance on this part segmentation dataset that is $\sim 1\%$ higher than the specialized segmentation networks, SSCN [11] and SGPN [38].

In Fig. 4, we show few representative segmentation results. High mIoU is achieved by Ψ-CNN for the high-quality results, whereas the mIoU value is low for the other case. Examining the low-quality results, we found that most of these cases are caused by one of the two conditions. (1) Confusing ground truth labelling: E.g. the axle in *Skateboard* is labelled as a separate segment in most of the ground truth samples but part of the wheels in few other samples. Hence, the network learns the more dominant segmentation. Similar is the case for the legs of *Chairs*. (2) Small parts without clear boundaries: E.g. handles of a *Bag* are considered separate segments in the ground truth. We also provide further examples in the supplementary material. From these results, we can easily conclude the success of Ψ-CNN for the part segmentation task.

C. Semantic Segmentation

We also test our model for Semantic Segmentation of real world data with RueMonge2014 dataset [30]. This dataset contains 700 meter facades along a street annotated with point-wise labelling. The classes includes *window, wall, balcony, door, roof, sky* and *shop*. The point clouds are provided with color features. To train our network, we split both the training and testing data into $1m^3$ blocks. We align the facade plane of all the blocks into the same plane, and adjust the gravitational axis to be upright. We only force the x and y dimensions to have zero-means, but not the z axis. This processing strategy is adopted to avoid loosing the height information. We use xyz+rgb as input raw features to train our network. The used network configuration is MLP(64)-Octree(64-64-128-128-256-256). Table 3 compares the results of our approach with those of the recent methods on this dataset, under the evaluation protocol of [10]. With 7 parametric layers, we achieve better performance than OctNet, which uses 20 parametric layers to learn the final representation of each point. These results demonstrate the promises of Ψ-CNN in practical applications. Visualizations for the segmentation results are provided in the supplementary material.

D. Discussion

For geometrically meaningful convolutions, knowledge of local neighborhood of points is imperative. A related approach, ECC [33] exploits range search for this purpose. Another obvious choice is K-NN clustering. However, with tree structures, e.g. octree; the point neighborhood information is already readily available that adds to computational efficiency of Ψ-CNN. In Fig. 5, we report the timings of computing neighborhoods under different choices, and compare them to octree construction. As can be seen, for larger number of input points, octree structuring is more efficient as compared to K-NN and range searching. Moreover, its efficiency is also better than Kd-tree for large input sizes because the binary split in Kd-tree forces it to be much deeper than octree.

Running our classification network on 1K randomly selected samples from ModelNets, we compute the test time of our network for point clouds of sizes 10K, and report timings in Table 4. The test time for a sample consists of time required to construct the octree and performing the forward

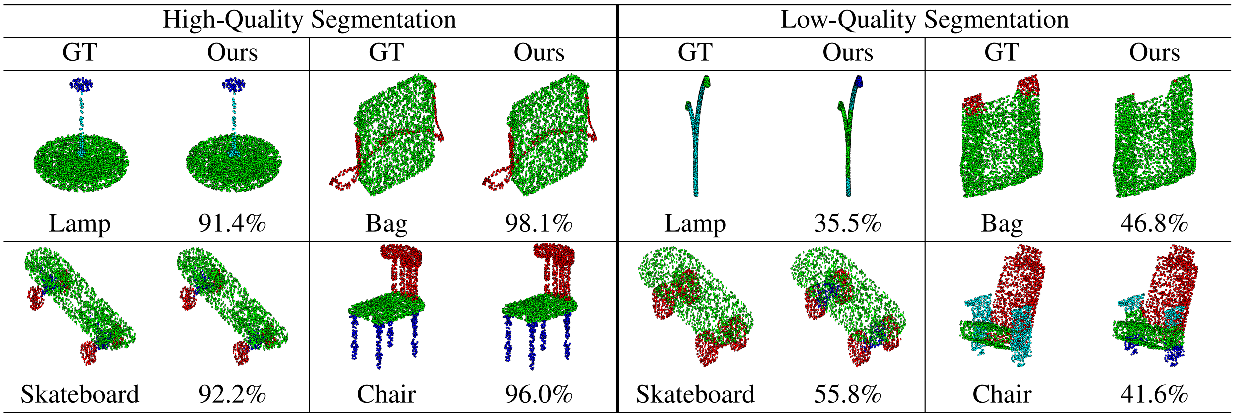

Figure 4. Representative examples of high- and low-quality segmentation results of Ψ-CNN. Computed mIoU is also given in each case. Low-quality segemetation generally result from: (1) confusing ground truth labeling, e.g. axles of *skateboards* are considered separate segments in most of the ground-truth labels, (2) small object parts with no clear boundaries, e.g. handles of *bags*. Color coding is within category (best viewed on screen).

Table 2. Results on ShapeNet part segmentation dataset

Method	mIoU	NO.	Airplane	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motorbike	Mug	Pistol	Rocket	Skateboard	Table
3D-CNN [27]	79.4	0	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6	74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
Kd-net [16]	82.3	0	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
PointNet [27]	83.7	0	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
SyncSpecCNN [41]	84.7	2	81.6	81.7	81.9	75.2	90.2	74.9	**93.0**	86.1	84.7	95.6	66.7	92.7	81.6	60.6	**82.9**	82.1
KCNet [32]	84.7	1	82.8	81.5	86.4	77.6	90.3	76.8	91.0	**87.2**	84.5	95.5	69.2	94.4	81.6	60.1	75.2	81.3
SO-Net [18]	84.9	1	82.8	77.8	**88.0**	77.3	90.6	73.5	90.7	83.9	82.8	94.8	69.1	94.2	80.9	53.1	72.9	83.0
PointNet++ [28]	85.1	0	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
Ψ-CNN	**86.8**	**11**	**84.2**	**82.1**	83.8	**80.5**	**91.0**	**78.3**	91.6	86.7	**84.7**	**95.6**	**74.8**	**94.5**	**83.4**	**61.3**	75.9	**85.9**

Table 3. Semantic Segmentation on RueMonge2014 dataset

Method	Average	Overall	IoU
Riemenschneider et al. [30]	–	–	42.3
Martinovic et al. [22]	–	–	52.2
Gadde et al. [10]	68.5	78.6	54.4
OctNet 256^3 [29]	73.6	81.5	59.2
Ψ-CNN	**74.7**	**83.5**	**63.6**

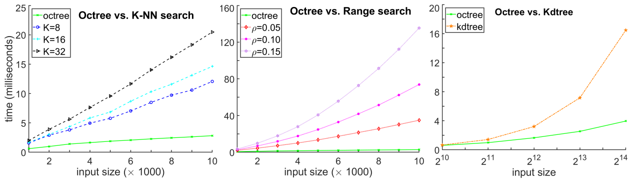

Figure 5. Comparison of octree structuring with K-NN, range search and Kd-tree for neighborhood computation.

Input size	Octree construction	Forward pass	Total	Normal computation
10K	3.5	30.6	34.1	27.4

Table 4. Per-sample test time (ms) for 10K input. The computing time for normals is included for reference only - indicated by red.

pass. We also show the time of normal computation in the table *for reference*. Our approach does not compute normals to achieve the results reported in the previous section. To put these timings into perspective, PointNet++ [28] requires roughly 115ms for a forward pass of input with 1024 points on the same machine. In Fig. 6, we also show a representative example of point cloud coarsening by our method under octree structuring. Our network gradually sparsifies the point cloud by applying spherical convolutional kernel at each level.

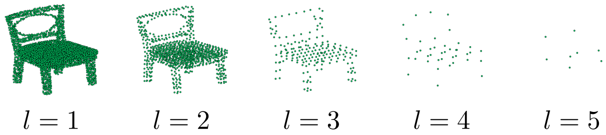

Figure 6. Point cloud coarsening example under octree structuring by our technique. 'l' is the octree level.

6. Conclusion

We introduced the notion of spherical convolutional kernels for point cloud processing and demonstrated its utility with a neural network guided by octree structure. The network successively performs convolutions in the neighborhood of its neurons, the locations of which are governed by the nodes of the underlying octree. To perform the convolutions, our spherical kernel divides its occupied space into multiple bins and associates a weight matrix to each bin. These matrices are learned with network training. We have shown that the resulting network can efficiently process large 3D point clouds in effectively achieving excellent performance on the tasks of 3D classification and segmentation on synthetic and real data.

Acknowledgments This research was supported by ARC Discovery Grant DP160101458 and partially by DP190102443. We also thank NVIDIA corporation for donating the Titan XP GPU used in our experiments.

References

[1] 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*, pages 1534–1543, 2016. 1

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 4

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014. 1, 2, 5

[4] 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:1512.03012*, 2015. 1

[5] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling. Spherical cnns. In *International Conference on Learning Representations*, 2018. 3

[6] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, 2016. 3

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016. 1, 2, 5

[8] 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. 3

[9] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. Recognizing objects in range data using regional point descriptors. *European Conference on Computer Vision*, pages 224–237, 2004. 3

[10] R. Gadde, V. Jampani, R. Marlet, and P. V. Gehler. Efficient 2D and 3D facade segmentation using auto-context. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 40(5):1273–1280, 2018. 7, 8

[11] B. Graham, M. Engelcke, and L. van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 18–22, 2018. 7

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

[13] J. Huang and S. You. Point cloud labeling using 3D convolutional neural network. In *ICPR*, pages 2670–2675, 2016. 3

[14] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 2

[15] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 1, 2, 5

[16] 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*, pages 863–872. IEEE, 2017. 1, 2, 6, 7, 8

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[18] 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*, pages 9397–9406, 2018. 2, 4, 7, 8

[19] Y. Li, R. Bu, M. Sun, and B. Chen. Pointcnn. *Advances in Neural Information Processing Systems*, 2018. 2, 4

[20] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. FPNN: Field probing neural networks for 3D data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016. 3

[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015. 6

[22] A. Martinovic, J. Knopp, 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*, pages 4456–4465, 2015. 8

[23] 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*, pages 922–928. IEEE, 2015. 1, 3, 4

[24] D. Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982. 1, 4

[25] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010. 3

[26] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 27–40. ACM, 2017. 6

[27] 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*, pages 652–660, 2017. 1, 2, 6, 7, 8

[28] 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. 1, 2, 4, 6, 7, 8

[29] 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*, pages 3577–3586, 2017. 1, 2, 3, 4, 5, 7, 8

[30] H. Riemenschneider, A. Bódis-Szomorú, J. Weissenberg, and L. Van Gool. Learning where to classify in multi-view semantic segmentation. In *European Conference on Computer Vision*, pages 516–532, 2014. 2, 7, 8

[31] N. Sedaghat, M. Zolfaghari, and T. Brox. Orientation-boosted voxel nets for 3D object recognition. In *British Machine Vision Conference*, 2017. 3

[32] 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*, volume 4, 2018. 2, 4, 6, 7, 8

[33] 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. 1, 2, 4, 5, 6, 7

[34] 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*, pages 57–62. ACM, 2010. 3

[35] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *European Conference on Computer Vision*, pages 356–369, 2010. 3

[36] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015. 5

[37] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics*, 36(4):72, 2017. 6

[38] W. Wang, R. Yu, Q. Huang, and U. Neumann. SGPN: Similarity group proposal network for 3D point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018. 7

[39] 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*, pages 1912–1920, 2015. 1, 2, 3, 4, 6, 7

[40] L. Yi, V. G. Kim, D. Ceylan, I. 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*, 35(6):210, 2016. 1, 2, 7

[41] L. Yi, H. Su, X. Guo, and L. J. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017. 2, 5, 8

[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*, pages 199–208, 2017. 3

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