A Survey on Hierarchical Structure-based Networks for Learning from Point Cloud

Nishan Tamang[1] and Ramy Battrawy[2]

[1] n_tamang19@cs.uni-kl.de
[2] ramy.battrawy@dfki.de

Abstract. 3D data can be represented in different formats such as regular/structured representation and irregular/unstructured representation (e.g. point clouds). Due to the unstructured nature of the point clouds, most of the existing methods for processing point clouds rasterize them into voxel-grid representation and forward it to convolutional neural networks as inputs where voxel-grids are convolved with 3D convolutional kernels. But these methods are computationally expensive and leads to excessive memory usage while processing them due to the sparsity of the voxels. There are some approaches that group the point clouds using search methods like k-NN which is then used to compute the local feature representation of the neighborhood. But this method still leads to excessive memory consumption and increase in time complexity. Therefore, processing point clouds is still a challenge. In this paper, we study different deep learning approaches based on hierarchical data structures for processing point clouds to extract features hierarchically and perform tasks such as classification and segmentation.

Keywords: point cloud, hierarchical structure, classification, segmentation, deep learning, kd-tree, self-organizing maps, octree

1 Introduction

With the availability of various sensor devices like LiDAR, RGB-D and stereo imaging systems, the acquisition of 3D data has increased. Along with the rise in applications such as autonomous driving, robotics and augmented reality, the usage of 3D data has seen its significance. In recent years, convolutional neural networks [13] with state-of-the-art performance have been successful in solving computer vision problems. The main reason for its success is its ability to learn highly effective features from data. However, their ability to learn is only limited to regular grids e.g., pixel arrays of images and videos. Also due to the irregular format of point clouds, convolutional neural networks cannot be used directly. This has prompted research in various techniques for effectively processing 3D data such as point clouds and to solve tasks such as 3D shape classification, 3D point cloud segmentation and 3D shape retrieval. With the availability of data sets such as ModelNet [15], ShapeNet [1], PartNet [4], KITTI Vision Benchmark Suite [3], research on different methods for processing point clouds is being proposed in increasing numbers.

3D data can be represented with different formats, regular data (e.g., multi-view images and 3D volumetric grids) and irregular data (e.g., point clouds and meshes). It is possible to process regular data by rasterizing 3D data to voxel representation and perform 3D convolutions ([10], [15]). But this leads to redundant processing due to the sparsity of 3D data which leads to increase in computation cost and excessive memory usage. On the other hand, processing irregular data such as point clouds is not trivial as there exists various challenges. Point clouds are a set of points in a 3D metric space represented by x, y and z coordinates. Point clouds are unordered and suffers from missing data points as they are not uniformly sampled across different regions of an object resulting in some regions containing dense points while others suffer from sparse points. The number of points in the point cloud is unknown in advance in case we want to sample number of points and unstructured as they are not on a regular grid.

This report will focus at different methods in processing point clouds by constructing CNN's based on different hierarchical structures where features are extracted in a hierarchical way as will be seen in Section 2. These methods are then evaluated on data sets such as ModelNet10/ModelNet40 [15] and ShapeNet-part [1] for tasks like classification and segmentation which is demonstrated in Section 3. We will then look at a brief comparison between the methods.

2 Overview of different methods

2.1 Kd-Networks

Kd-Network [5] constructs the deep learning architecture based on the construction of the kd-tree [2] on the input point clouds. The kd-tree [2] is constructed in a top-down fashion and splits the set of points into equally-sized subsets in a recursive manner. As a result, a balanced kd-tree of depth D consisting of leaf and non-leaf nodes is produced. The non-leaf nodes are associated with splitting directions along the x, y and z axis. The Kd-Network [5] performs computations at each of the nodes and extracts features at each nodes hierarchically. In Fig. 1, the Kd-Network [5] associated with the kd-tree [2] construction is demonstrated. The leaf nodes consists of the input point clouds and the parent nodes are the resulting non-linear activation of the leaf nodes. The circles corresponds to affine transformations with learnable parameters. Colors of the circles indicate their splitting direction along different axes and parameters are shared at each layer containing the same circle colors. In this bottom-up fashion, the root representation $\mathbf{v}_1$ is computed which is further used to learn a classifier.

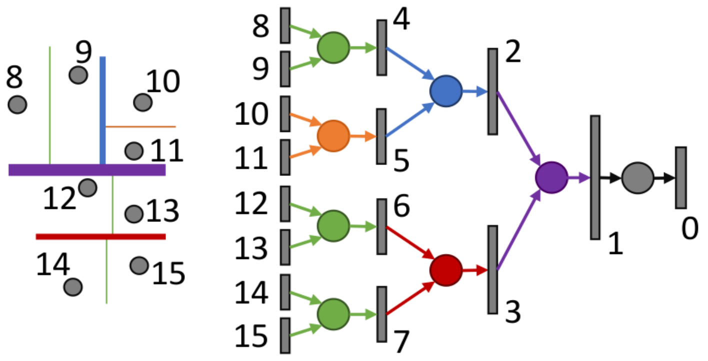

Fig. 1: Kd-Network [5] based on the construction of the kd-tree [2] using input point clouds. The leaf nodes consist of the input points. The arrows indicate the forward pass during training [5].

Given a node i at level $l(i)$ containing child nodes $c_1(i)$ and $c_2(i)$ at level $l(i+1)$ with its vector representation computed in advance $\mathbf{v}_{c_1(i)}$ and $\mathbf{v}_{c_2(i)}$, its vector representation $\mathbf{v}_i$ can be computed as follows:

$$\mathbf{v}_i = \phi(W^{l_i}_{d_i}[\mathbf{v}_{c_1(i)}; \mathbf{v}_{c_2(i)}] + \mathbf{b}^{l_i}_{d_i}) \tag{1}$$

where $\phi(.)$ is some non-linear activation function (e.g., ReLU), the square brackets indicate concatenation, W and $\mathbf{b}$ are the learnable parameters at level $l(i)$ and along the splitting direction $d_i \in \{x, y, z\}$. At last, the root representation $\mathbf{v}_1$ of the tree T can be computed in a bottom-up fashion. Given the root representation $\mathbf{v}_1$, we can directly learn a linear classifier as follows:

$$\mathbf{v}_0 = W^0\mathbf{v}_1(T) + \mathbf{b}^0 \tag{2}$$

where W^0 and $\mathbf{b}^0$ are the learnable parameters of the final multi-class classifier.

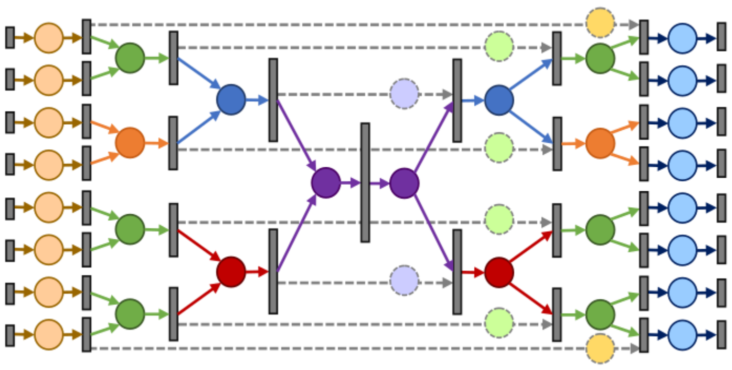

Fig. 2: The encoder-decoder network architecture for segmentation. The dashed lines indicate skip-connections and the light colored circles correspond to affine transformations consisting of learnable parameters [5].

As the kd-tree [2] plays a role in the construction of the Kd-Network [5], the performance of the network depends on the construction of the kd-tree [2] i.e., choosing the splitting direction of the point clouds and the order of the nodes. It is also non-invariant to rotation as the underlying kd-tree [2] structure are not invariant to them. These are some of the limitations of Kd-Network [5]. In Fig. 2, the Kd-Network [5] can be extended to perform segmentation where the Kd-Network [5] acts as the encoder. The decoder part of the architecture computes the second representation $\tilde{\mathbf{v}}_i$ at each nodes using the representation computed at the encoder part by skip-connections. Given the representation of a node $\tilde{\mathbf{v}}_i$ at level $l(i)$, its child nodes can be computed in a top-down fashion as follows:

$$\tilde{\mathbf{v}}_{c_1(i)} = \phi([\tilde{W}^{l_i}_{d_{c_1(i)}}\tilde{\mathbf{v}}_{(i)} + \tilde{\mathbf{b}}^{l_i}_{d_{c_1(i)}}; S^{l_i}\mathbf{v}_{c_1(i)} + \mathbf{t}^{l_i}]),$$

$$\tilde{\mathbf{v}}_{c_2(i)} = \phi([\tilde{W}^{l_i}_{d_{c_2(i)}}\tilde{\mathbf{v}}_{(i)} + \tilde{\mathbf{b}}^{l_i}_{d_{c_2(i)}}; S^{l_i}\mathbf{v}_{c_2(i)} + \mathbf{t}^{l_i}]), \tag{3}$$

where $\tilde{W}^{l_i}_{d_{c*(i)}}$ and $\tilde{\mathbf{b}}^{l_i}_{d_{c*(i)}}$ are the parameters of the affine transformation and S^{l_i} and $\mathbf{t}^{l_i}$ are the parameters within the skip-connections from $\mathbf{v}_{c_1(i)}$ to $\tilde{\mathbf{v}}_{c_1(i)}$.

2.2 SO-Net

SO-Net [9] performs feature extraction hierarchically on Self-Organizing Maps (SOM) [6]. The SO-Net [9] organizes the irregular point clouds into mxm rectangular Self-Organizing maps (SOM) [6]. SOM is used to model the spatial distribution of the input point clouds. Basically SO-Net [9] produces a lower dimensional representation of the input point cloud. The input points are then normalized by performing point-to-node k nearest neighbor search and each point is normalized as shown in Fig. 3. Given the trained SOM, the point-to-node k nearest neighbor (kNN) search on the SOM nodes S for each point p_i can be computed as follows:

$$s_{ik} = kNN(p_i|s_j, j = 0, \ldots, M-1) \tag{4}$$

where M is the number of SOM nodes. k is selected to systematically adjust the receptive field overlap. Point features are then learned from the normalized individual point clouds which are forwarded through a series of fully-connected layers. The points are normalized by subtracting each points into its k associated nodes as shown,

$$p_{ik} = p_i - s_{ik} \tag{5}$$

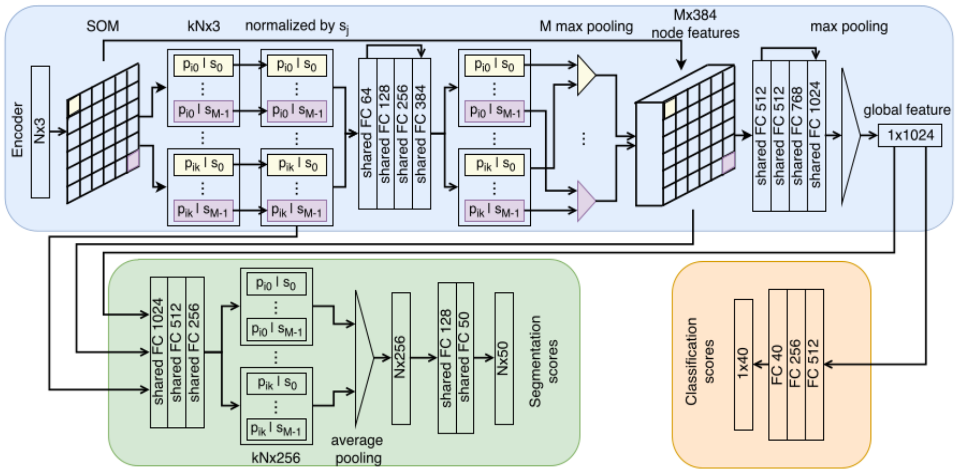

Fig. 3: The SO-Net [9] architecture used for classification and segmentation. For classification, the aggregated global feature vector is forwarded through a series of fully-connected layers to learn a classifier and get the classification score of an object. For segmentation, the global feature vector along with the node features and normalized points are concatenated and forwarded into a series of shared fully connected layers to get the segmentation score [9].

The final feature is then learned from the point features associated with its SOM-nodes using channel-wise max pooling and finally the nodes are forwarded to fully-connected layers again and aggregated into a final feature vector which ultimately represents the input point clouds. The channel-wise max pooling works by finding the maximum activation output for each of the point features associated to each of the SOM nodes S. The concatenation of the learned point features with the SOM nodes is performed as it plays the role of assembling these mini point clouds back into the original point cloud as SOM [6] reveals the spatial distribution of the input point cloud.

2.3 Ψ-CNN

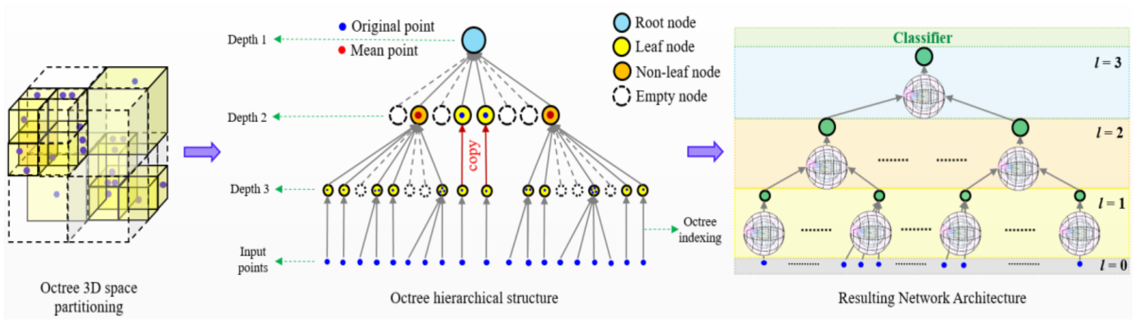

Fig. 4: Illustration of the Ψ-CNN [8] where the octree-space partitioning [11] takes place first to partition the input points into cubic volumes. The corresponding hierarchical tree is constructed and the resulting network has the same number of hidden layers as the tree depth and at each layer. It applies the spherical convolutional kernels to each neurons at different layers for extracting features [8].

Ψ-CNN [8] introduces the use of spherical convolutional kernels and the network architecture is constructed based on the octree partitioning of the point clouds in 3D space. For a given target point

$\mathbf{x}_i \in \mathbb{R}^3$, the radius of the spherical convolutional kernel is $\rho \in \mathbb{R}^3$ and its points are the points that are positioned within the radius of the spherical kernel i.e., $\mathcal{N}(\mathbf{x}_i) = \{\mathbf{x} : d(\mathbf{x}, \mathbf{x}_i) \leq \rho\}$, where $d(.)$ is a l_2 distance metric. The sphere is then divided into $nxpxq$ volumetric bins by the partitioning the space along the azimuth (θ) and radial (ϕ) dimensions. For each of the bins, the kernel specifies a matrix of learnable parameters and weights points that fall within the bin for convolution. To find the relevant weight matrices of the kernel for the points, we transform the neighboring points $\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i)$, to its spherical coordinates using $\mathbf{x}_i$ as the origin. For the weight matrices, we compute their indices by the help of indices along the azimuth, elevation and radial dimensions. Using this indexing, we can assign the weight matrices to its associated points in the bins. With the weight matrix and the target point situated in a bin, a non-linear activation function (ReLU) is used to compute its activation output.

In Fig. 4, the network is constructed based on the result of the octree partitioning [11]. The network layer is equal to the maximum depth of the tree. Every node except the leaf has maximum capacity of one point. For each non-empty nodes in the tree, there is a corresponding neuron in the neural network. To facilitate the convolutions, we assign a single 3D point to each neuron except the leafs. For leafs, its associated point is the mean value of the data points. A neuron uses its associated point/location to select the appropriate spherical kernel and applies non-linear activation. All convolutional layers before the last layer are followed by batch normalization and ReLU activations.

In Fig. 5, to perform classification we use the features of the root node concatenated with all the max-pooled features at all the depth and forward it to a series of fully-connected layer to get the classification score. To perform segmentation, we use the features of the root node and concatenate it with the raw features of all the ancestors along the path to the root node.

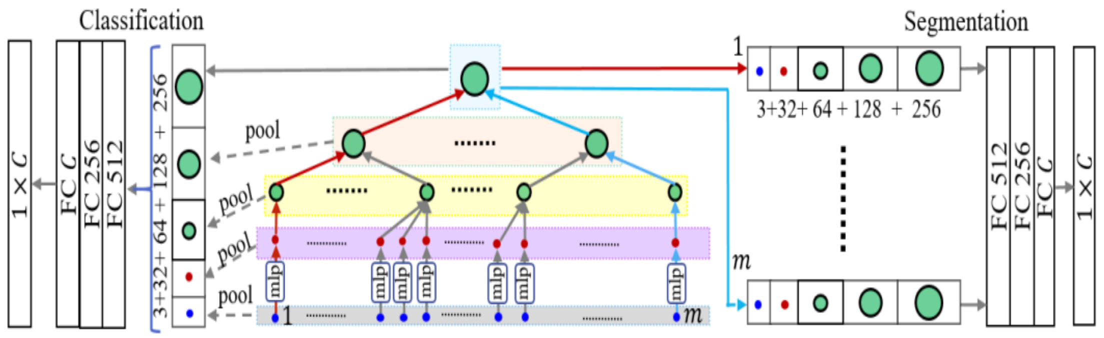

Fig. 5: Classification and Segmentation using the root representation obtained from the core network architecture in Fig. 4.

2.4 A-CNN

In the method proposed by Komarichev et al., [7], Annular convolutions are applied to point clouds. This approach mentions the use of ring-shaped structures to capture the local neighborhood geometric features of each point and specifies the directions in the computation. The ring structures consists of regular rings and dilated rings (Fig.6). The ring-based structure does not contain overlaps (duplicate neighboring points) at the query points neighborhood. Each ring contains unique points. Dilated rings contain empty space between rings making it different from the regular rings. The coverage of the dilated rings would be larger compared to regular rings. The idea of dilated rings is inspired from dilated convolutions on image processing [14]. To guarantee that non-overlapping neighboring points exist in each rings, a constrained k-NN search algorithm by using the Euclidean metric is proposed to ensure the closest and unique points will be found at each rings. Each ring is defined by its inner radius R_{inner} and outer radius R_{outer}. Given the rings, in order to learn the relation of neighboring points in a local region, the points need to be ordered (clockwise/counterclockwise) for applying the annular convolution. The ordering process consists of two steps: projection and ordering.

The idea for projection is to compute the normal $\mathbf{n}_i$ for a query point $\mathbf{q}_i$ and compute the projection $\mathbf{p}_j$ of the query point and its neighboring points on the tangential plane. Finally, we compute the angles for the neighboring points and order the points by sorting them based on the descending order of their angles to obtain the counterclockwise ordering. The ordered neighbors can now be represented in an array. Before performing annular convolution, we need to loop the array of neighbors with respect to the size of the kernel on each ring. We can then perform the standard convolutions on the resulting array. The complete network architecture for classification and segmentation is illustrated in Fig. 7 where the points are subsampled at the start using Farthest Point Sampling (FPS) algorithm [12]. For segmentation, the local and global features from the encoder network is used. The points from the encoder part are upsampled in order to get back to the original point cloud size.

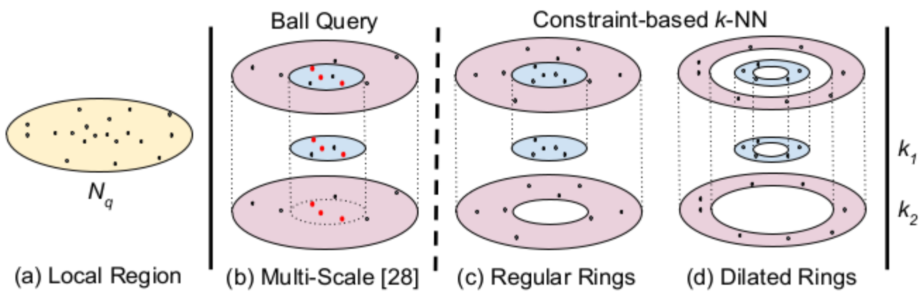

Fig. 6: Comparison of regular rings and dilated rings. The dilated rings cover large space by using the same number of points as the regular rings [7].

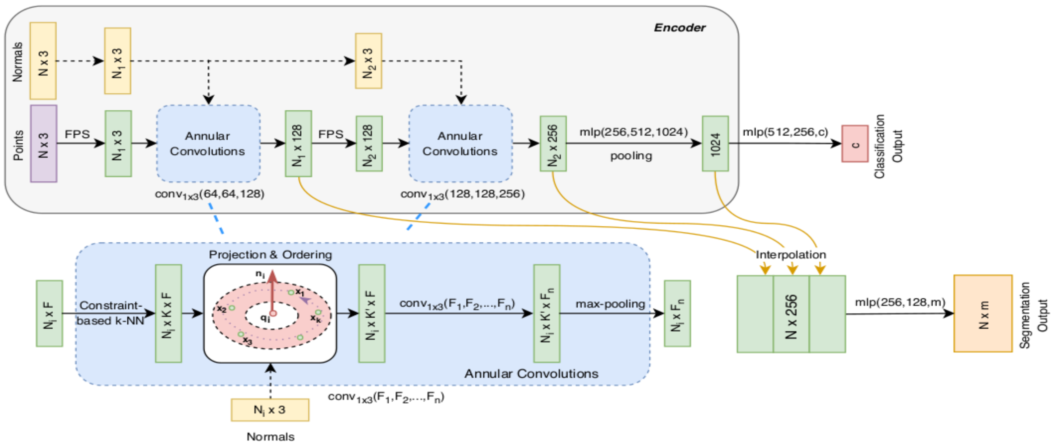

Fig. 7: The A-CNN [7] architecture for Classfication and Segmentation task. The normals are used only to determine the ordering of the points in the local region [7].

3 Comparison

3.1 Data sets for Object Classification and Segmentation

From Table 1. we can observe that A-CNN [7] outperforms other methods by giving the best result while evaluating on the ModelNet10 and ModelNet40 [15] data set for object classification.

	Datasets			
Methods	ModelNet10		ModelNet40	
	Class Accuracy	Instance Accuracy	Class Accuracy	Instance Accuracy
Kd-Networks [5]	92.8	93.8	86.3	90.6
SO-Net [9]	93.9	94.1	87.3	90.9
Ψ-CNN [8]	94.4	94.6	88.7	92.0
A-CNN [7]	**95.3**	**95.5**	**90.3**	**92.6**

Table 1: Comparison of the performance of different methods on ModelNet10 and ModelNet40 for Classification.

From Table 2. we can observe that Ψ-CNN [8] outperforms other methods by giving the best result while evaluating on the ShapeNet-part data set [1] for segmentation with mIoU of **86.8** and outperforms other methods on 8 out of 16 categories.

Methods	mIoU	Aero	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Bike	Mug	Pistol	Rocket	Skate	Table
Kd-Networks [5]	82.3	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
SO-Net [9]	84.9	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
Ψ-CNN [8]	**86.8**	**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**
A-CNN [7]	85.9	83.9	**86.7**	83.5	79.5	**91.3**	77.0	91.5	86.0	**85.0**	95.5	72.6	**94.9**	**83.8**	57.8	**76.6**	83.0

Table 2: Comparison of the performance of different methods on ShapeNet-part data set for Segmentation and the score for each class.

3.2 Time and Space Complexity

For Kd-Network (depth-10) model [5], it can take 16 hours (ModelNet40 [15]) for training and for depth-15 model, it takes upto 5 days for training using NVidia Titan Black to perform classification. The memory footprint for Kd-Network [5] on segmentation task for one example during training is less than 120 Mb. For SO-Net [9], it took 3 hours on ModelNet40 [15] with GTX1080Ti for classification. For Ψ-CNN [8], the test-time for classification on 1K randomly selected samples from ModelNets [15] with point clouds of size 10K took 34.1 ms. For the segmentation model of A-CNN [7], the training time is about 19 hours on a single NVIDIA Titan Xp GPU with 12 Gb GDDR5X. The size of the trained model for A-CNN is 22.3 Mb.

4 Conclusion

Based on our observation of different methods and their results while processing point clouds, A-CNN with ring-structures (regular and dilated) outperform methods such as Kd-network [5], SO-Net [9] and Ψ-CNN [8] on classification. In case of segmentation, Ψ-CNN [8] outperforms other methods with the mIoU of 86.8. Although A-CNN has better results than Ψ-CNN [8], the performance time of Ψ-CNN [8] is better than A-CNN [7]. Network architectures based on hierarchical structures are more effective and practical for processing irregular 3D data such as point clouds and the depth (hidden-layer) of the network is equal to the depth of the corresponding hierarchical structures of the network.

References

1. L. Guibas P. Hanrahan Q. Huang Z. Li S. Savarese M. Savva S. Song A. X. Chang, T. Funkhouser and H. Su. *ShapeNet: An information-rich 3D model repository.* arXivpreprintarXiv:1512.03012, 2015.
2. J. L. Bentley. *Multidimensional binary search trees used for associative searching.* Communications of the ACM, 18(9):509-517, 1975.
3. Andreas Geiger, Philip Lenz, and Raquel Urtasun. *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite.* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2012.
4. A. X. Chang L. Yi S. Tripathi L. J. Guibas K. Mo, S. Zhu and H. Su. *PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding.* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
5. Roman Klokov and Victor Lempitsky. *Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models.* Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.
6. T. Kohonen. *The self-organizing map.* Neurocomputing, 21(1):1–6, 1998.
7. Artem Komarichev, Zichun Zhong, and Jing Hua. *A-CNN: Annularly Convolutional Neural Networks on Point Clouds.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
8. Huan Lei, Naveed Akhtar, and Ajmal Mian. *Octree guided CNN with Spherical Kernels for 3D Point Clouds.* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
9. Jiaxin Li, Ben M Chen, and Gim Hee Lee. *SO-Net: Self-Organizing Network for Point Cloud Analysis.* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
10. Daniel Maturana and Sebastian Scherer. *VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition.* Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems, 2015.
11. D. Meagher. *Geometric modeling using octree encoding.* Computer graphics and image processing, 19(2):129–147, 1982.
12. C. Moenning and N. A. Dodgson. *Fast marching farthest point sampling.* Technical report, University of Cambridge, Computer Laboratory, 2003.
13. Y. Bengio Y. LeCun, L. Bottou and P. Haffner. *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11):2278–2324, 1998.
14. F. Yu and V. Koltun. *Multi-scale context aggregation by dilated convolutions.* In International Conference on Learning Representations, 2016.
15. A. Khosla F. Yu L. Zhang X. Tang Z. Wu, S. Song and J. Xiao. *3D shapeNets: A deep representation for volumetric shapes.* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2015.