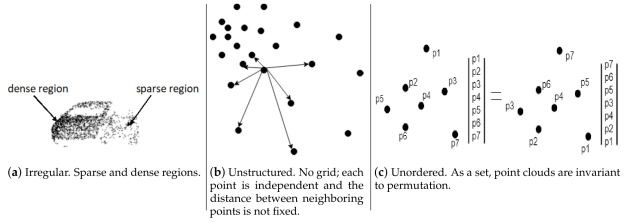
STATS Project Report

October 19, 2021

1 Introduction

Point clouds are a set of points represented by their spacial coordinates. They could been acquired by various types of 3D scanners, including everyday devices such as LiDARs and RGB-D cameras. Compared with 2D data, point clouds could provide geometric, shape, and scale information in the environment.

However, there are many challenges in point clouds analysis. Point clouds are irregular, unstructured, and orderless as shown in Figure 1. First, irregular means that point cloud data are not evenly sampled across the different regions of an object or scene. Some regions could have dense points while others have sparse points. Second, unstructured means that point cloud data are not placed on a regular grid with each point scanned independently. This also suggests that for each point, its distance to neighboring points are not fixed. Third, orderless means that, since point cloud is a set of points, we could have different permutations of the orders of the points while still representing the same point cloud.



(a) Irregular. Sparse and dense regions. (b) Unstructured. No grid; each point is independent and the distance between neighboring points is not fixed. (c) Unordered. As a set, point clouds are invariant to permutation.

Figure 1: Challenges of point cloud data. [1]

In this review, we will focus on a task called 3D point cloud semantic segmentation. Similar to 2D semantic segmentation, we want to associate each point in the point cloud with a class label. There are three paradigms for semantic segmentation: projection-based, discretization-based, and point-based. We choose some representative models in each categories to review its architecture and mean ideas.

The structure of this paper is as follows. Section 2 introduces the datasets and evaluation metrics for point cloud semantic segmentation. Section 3 presents the model architectures. Section 4 provide the performance comparison and an example on our custom dataset. Section 5 concludes the paper.

2 Background

2.1 Datasets

The datasets for 3D point cloud segmentation could be categorized by the device they were required, including Mobile Laser Scanners (MLS), Aerial Laser Scanners (ALS), static Terrestrial Laser Scanners (TLS), RGB-D cameras and mixed 3D scanners. We choose the Stanford 3D Indoor Scene Dataset (S3DIS) dataset as our benchmark dataset [2]. The dataset is acquired using a mixture of various 3D sensor on a platform called Matterport. It contains 6 large-scale indoor areas with 271 rooms. Each point in the point cloud is annotated with one of the 13 semantic categories. Class categories and some examples is shown in Figure 2.

2.2 Evaluation Metrics

The frequently used metrics in point cloud semantic segmentation is Overall Accuracy (OA) and mean Intersection Over Union (mIOU). OA is the average ac-

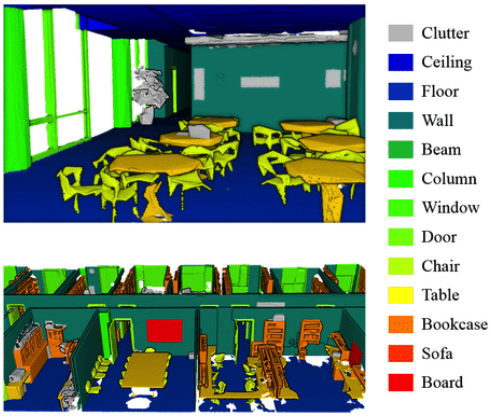


Figure 2: Examples of S3DIS dataset [3]

curacy for every point. mIOU first computes the IOU for each semantic class and then computes the average over classes. IOU is defined as follows:

$$\text{IOU} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive} + \text{False Negative}}$$

3 Methods

The Projection and Discretization-based Methods have the same principle. First, transform a point cloud to an intermediate regular representation. Second, segment the intermediate representation using traditional methods. Finally, project the intermediate segmentation results back to the point cloud.

However, the Point-based Methods directly work on raw point cloud without any transformation.

3.1 Project-based Methods

There are two paradigms of Project-based Methods: Multi-view Representation and Spherical Representation.

3.1.1 Multi-view Representation

The idea of multi-view representation is to project the 3D point cloud onto the 2D planes from multiple virtual camera views. Thus, a set of synthetic 2D-images is then generated and then be used as input to a 2D-CNN, designed for semantic segmentation to get a set of prediction scores. Finally, the obtained prediction scores are re-projected to the point cloud to obtain the segmentation results. Lawin et al.[4] were the first to propose the multi-view representation methods. An clear illustration of this idea is shown in Figure 3.

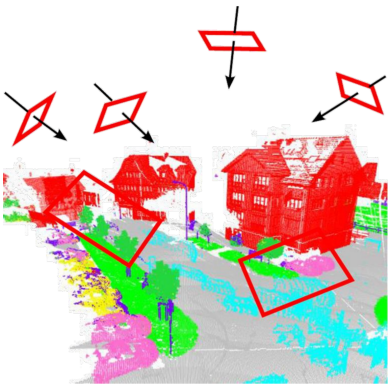


Figure 3: Multi-view Representation [5]

By projecting the 3D point clouds into 2D planes, one could employ many state-of-art 2D segmentation architectures to help segment the 2D picture and then reproject the data back to the 3D plane. However, there are several drawbacks of this multi-view representation method. First, the performance of multi-view segmentation methods is sensitive to viewpoint selection and occlusions. Second, the multi-view projection have not fully exploited the underlying geometric and structural information. Since they only choose several views which would not cover every perspective of the point cloud, the projection step inevitably introduces information loss.

3.1.2 Spherical Representation

Unlike the multi-view representation, spherical representation directly project the point cloud into the spherical space. One way of such projection used in RangeNet++[6] is shown below.

First, for each point $\mathbf{P}_i = (x, y, z)$,a mapping function Π: $\mathbb{R}^3 \to \mathbb{R}^2$ is used to convert the (x, y, z) to spherical coordinates and finally to image coordinates using the following equations:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 1/2[1 - \arctan(y, x)\pi^{-1}]w \\ [1 - \arctan(zr^{-1} + f_{up})f^{-1}]h \end{pmatrix}$$

where (u, v) are said image coordinates, (h, w) are the

height and width of the desired range image representation, $f = f_{up} + f_{down}$ is the vertical field-of-view of the sensor, and $r = ||\mathbf{P}_i||_2$ is the range of each point.

Finally, a list of (u, v) tuples containing a pair of image coordinates for each $\mathbf{P}_i$ is then generated to represent each point in the point cloud. $f = f_{up} + f_{down}$ are two parameters setted by the device one used to collect the 3D Point Cloud data. During this projection process, several points in the point cloud might be represented by the same (u, v). Thus, during the reprojection part, one would get a coarse segmentation since two or more points which were stored into the same range image pixel will get the same semantic label. To solve this problem, the authors propose a fast, GPU-enabled, k-nearest neighbor (kNN) search operating directly in the input point cloud to get a consensus vote of the k points in the scan that are the closest to it in 3D. Therefore, compared with the multi-view representation, spherical representation only use one range image to store the information of the original data. The drawback is that this might get coarse segmentation result. Whereas the benefit is that it has less computational cost compared with multi-view representation method.

3.2 Discretization-based Methods

Dense discretization and spare sidscretization are the two methods discussed in this section.

3.2.1 Dense Discretization Representation

For dense discretization, the 3D point could is first devided into a set of fine-grained occupancy voxels as shown in Figure 4. Then, 3D CNN would be applied in these voxels and did a voxel-wise segmentation. After that, all points with in a voxel are assigned the same segmantic label as the voxel and projected back to the 3D point cloud.

The benefit of using the dense discretization method is that volumetric-based networks are free to be trained and tested on point clouds with different spatial sizes

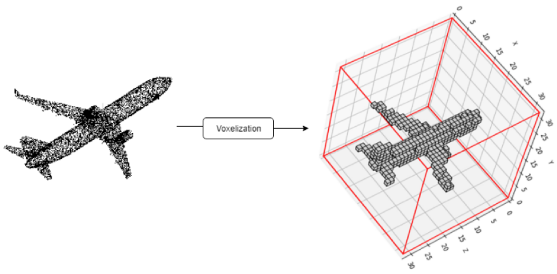


Figure 4: The point cloud of an airplane is voxelized to a $30 \times 30 \times 30$ volumetric occupancy grid. [1]

because of the scalability of 3D CNN. Moreover, the neighborhood structure of 3D point clouds is well preserved compared with the multi-view methods.

Although the voxel-based methods have shown good performance, this method is not efficient enough[1]. First, the voxels are in fact sparse. As we can see from Figure 4, only the airplane part have real value. Therefore, it is a consumption waste to convolve over the non-occupied regions. Second, since all 3D points within a voxel are assigned the same semantic label, the voxel size in fact limit the overall accuracy.

Thus, spares discretization is proposed.

3.2.2 Sparse Discretization Representation

Unlike the dense discretization, sparse discretization only convolve on the permutohedral lattice as shown in Figure 5.

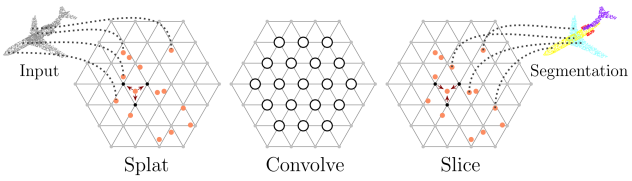


Figure 5: Bilateral Convolution Layer. [7]

A brief introduction of the Bilateral Convolution Layer (BCL) is learned from Su et.al. [7]. From my perspective, the BCL is similar to the Encoder Decoder model as we talked about in class. The key for the sparse discretization representation is that a signal is usually sparse in high-dimension. Therefore hash tables can be employed in BCL to index the populated

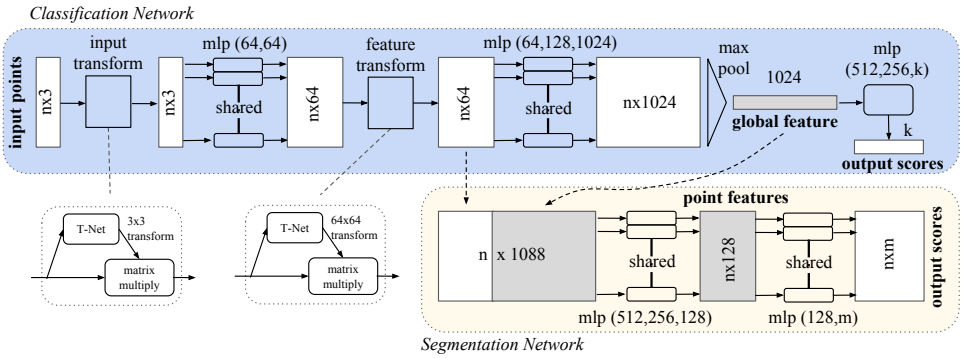


Figure 6: PointNet Architecture [8]

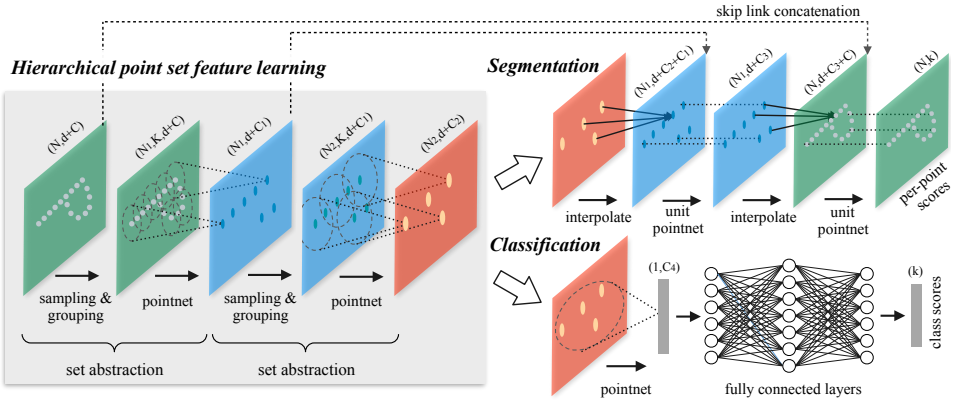


Figure 7: PointNet++ Architecture [9]

vertices and BCL would then do convolutions only at those locations. Thus the sparse inputs is efficiently processed [7].

3.3 Point-based Methods

There are four types of overall neural network architectures used in the point-based methods, which are point-wise MLP, RNN-based, point convolution based, and graph-based.

3.3.1 Point-wise MLP Methods

The idea of point-wise MLP is to use the same shared MLP for each point in the point cloud to extract their features individually. PointNet[8] was the pioneer work in this kind of methods. The architecture is shown in Figures 6. It used a permutation invariant global max pooling to obtain global features. They concatenated the individual features and global features to do the segmentation. Formally, taking an unordered point set $\{x_1, x_2, \cdots x_n\}$ and pass it to MLPs.

$$\text{Pointnet}(x_1, x_2, \cdots x_n) = \gamma(\max_{i=1\cdots n} h(x_i))$$

where h, γ are MLP.

This architecture is simple but works well. The main drawback is that it does not use local features for segment. They used one global feature for every point. This cased low precision in local detailed segmentation.

The work built on Pointnet is Pointnet++[9] and it addressed theses issues. Pointnet++ first uses sampling layer to sample some point of interest (centroids), and use grouping layer to find the neighbors of those points. Then, it uses a mini version of Pointnet on those centroids and their neighbors to reduce the neighborhood dimension. The above all operations is called a set abstraction unit, and could be repeatedly applied.

Specifically, the input of a set abstraction unit is a $N \times (d+C)$ matrix that from all the N points with coordinate dimension d and feature dimension C. The output is a $N_1 \times (d + C_1)$ matrix with abstracted M points

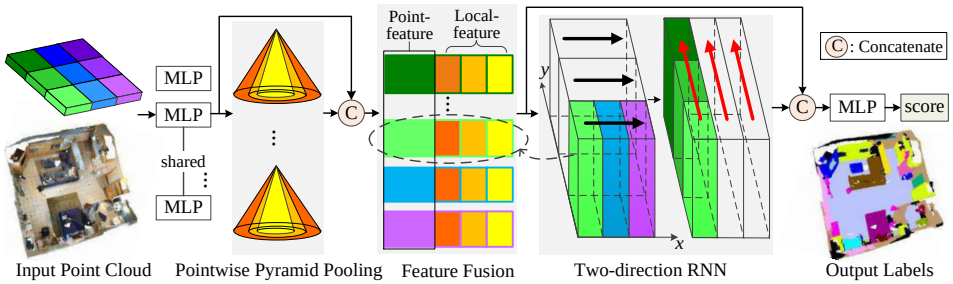


Figure 8: P3-RNN Architecture [10]

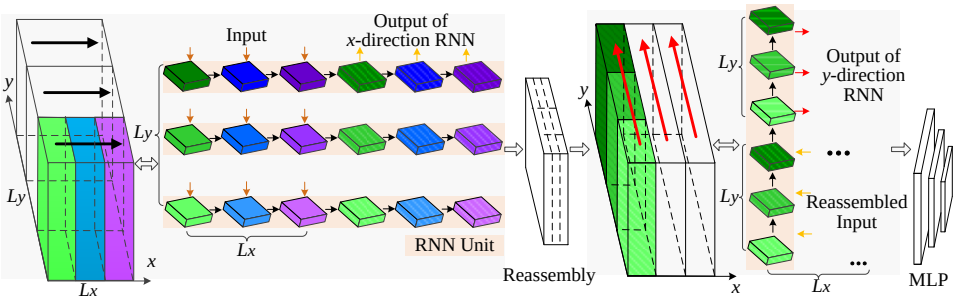


Figure 9: Illustration of bidirectional RNN [10]

with C' feature dimension. In the sampling layer, they used iterative farthest point sampling (FPS). Give a set of points $\{x_1, x_2, \cdots x_n\}$, they choose a subset of points $\{x_{i1}, x_{i2}, \cdots x_{im}\}$ such that x_{ij} is the most distant point from the set $\{x_{i1}, x_{i2}, \cdots x_{ij-1}\}$. In the grouping layer, the input is the all the points $N \times (d + C)$ and centroid points $N_1 \times d$. It will find K neighbors of each centroid using ball query. The output is a $N_1 \times K \times (d + C)$ matrix. Then, using the Pointnet (shared weights like CNN) in each neighborhood, we could reduce the dimension to $N_1 \times (d + C_1)$.

Before segmentation, we have to make the resolution back to the original points. They used interpolation and skip link concatenation to achieve that. The interpolation method they used was inverse distance weighted average based on k nearest neighbors shown in Equation 1. Skip link concatenation is take from Unet, which is just concate the value of set abstraction behind the interpolated value.

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \tag{1}$$

where

$$w_i(x) = \frac{1}{d\left(x, x_i\right)^p}, j = 1, \ldots, C$$

3.3.2 RNN-based Methods

To capture inherent context features from point clouds, RNN has also been used in segmentation. Based on Pointnet, Ye et al. proposed P3-RNN [10]. The overall architecture is shown in Figure 8.

The first contribution is that they proposed Pointwise Pyramid Pooling (3P). They used max pooling with different window size and stride 1 to slide over all the features. In this way, they could acquire local features. Formally, the 3P layer is

$$P(p_1, \ldots, p_n) = [\underset{p=p1,\cdots,p_n}{\text{MaxPool}}(f, k_1), \ldots, \underset{p=p1,\cdots,p_n}{\text{MaxPool}}(f, k_m)]$$

where f is the features after MLP layers, and k_i is the window size of the Maxpooling.

The second contribution is bidirectional RNN. After the 3P layer, the size of the feature is (L_x, L_y), where $L_x = D + F_1 + F_2 + F_3$ and $L_y = N$ (as shown in Figure 9). The author first train a LSTM using L_x as features dimension and L_y as time dimension. And

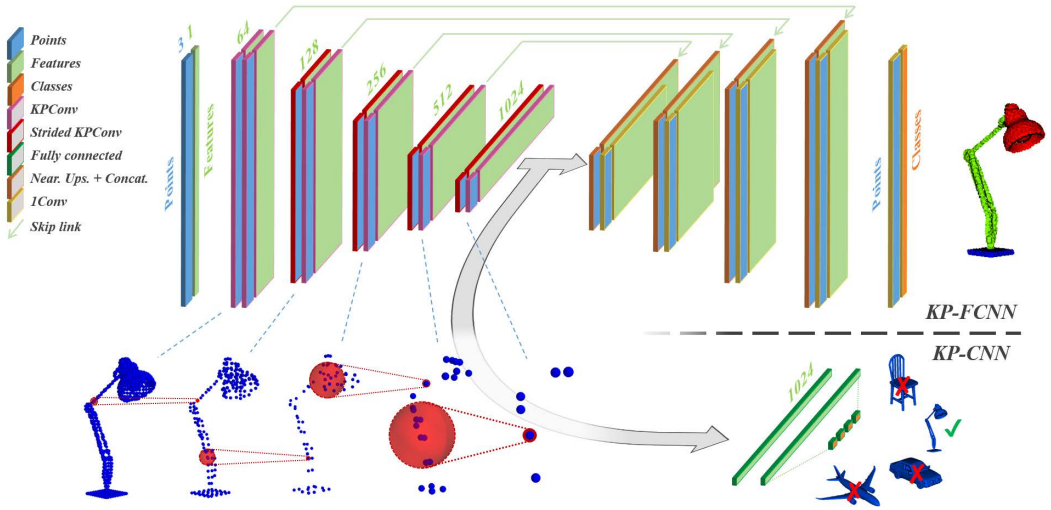


Figure 10: KP-FCNN architecture [11]

then do it again in the perpendicular direction. After that, it concatenates with the input and send to the MLP classifier. Since the points in the point cloud have common features, the RNN could extract them better.

3.3.3 Point Convolution Methods

Continuous Point Convolution Methods These methods define convolutional kernels on a continuous space. One of representative works is KPConv [11].

Suppose we have N points in point cloud as $\mathcal{P} \in \mathbb{R}^{N\times 3}$ and they have D dimensional features $\mathcal{F} \in \mathbb{R}^{N\times D}$. The neighbor of x is defined as all points in the ball with radius r,

$$\mathcal{N}_x = \{x_i \in \mathcal{P} \mid \|x_i - x\| \leq r\}$$

and the ball near x is defined as

$$y_i = x_i - x$$
$$\mathcal{B}_r^3 = \left\{y \in \mathbb{R}^3 \mid \|y\| \leqslant r\right\}$$

Then, we want to find K different points inside a ball as kernel points. The location of kernel points follows a specific rule than will introduce later. Suppose we have K kernel points $\tilde{x}_k$ inside a ball,

$$\{\tilde{x}_k \mid k < K\} \subset \mathcal{B}_r^3$$

and for each kernel point, their is a corresponding kernel matrix W_k

$$\{W_k \mid k < K\} \subset \mathbb{R}^{D_{in}\times D_{out}}$$

Finally, for a point x_i inside the ball $\mathcal{B}_r^3$, we could measure the distance between x and x_i, $y_i = x_i - x$. We define the kernel function g as

$$g\left(y_i\right) = \sum_{k=1}^{K} h\left(y_i, \tilde{x}_k\right) W_k$$

The kernel function is a weighted average of K kernel matrix with the weight defined as

$$h\left(y_i, \tilde{x}_k\right) = \max\left(0, 1 - \frac{\|y_i - \tilde{x}_k\|}{\sigma}\right)$$

The weight is correlated with the distance between the input point y_i and kernel point $\tilde{x}_k$. The closer they are, the larger the weights. The definition of σ will be mentioned later.

Based on the definition above, the KPConv at point x is define as

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g\left(x_i - x\right) f_i$$

where f_i is the feature of neighbor x_i. The whole procedure is shown in Figure 11.

The next problem is how to determine the position of the kernel points? The kernel points should distributed uniformly inside the ball. Authors converted it to an optimization problem. They defined two kind to loss to restrain the kernel points. The first loss is the repulsion between kernel points:

$$\forall x \in \mathbb{R}^3, \quad \mathcal{L}_k^{rep}(x) = \frac{1}{\|x - \tilde{x}_k\|}$$

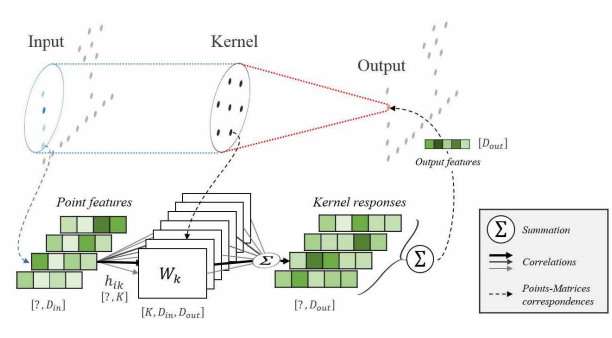


Figure 11: Illustration of KPConv [11]

and the second loss is the attraction between kernel points and the center points:

$$\forall x \in \mathbb{R}^3, \quad \mathcal{L}^{att}(x) = \|x\|^2$$

The tot loss function is

$$\mathcal{L}^{tot} = \sum_{k=1}^{K} \left(\mathcal{L}^{att}(\tilde{x}_k) + \sum_{i \neq k} \mathcal{L}_k^{rep}(x_i) \right)$$

With random start position, the kernel will converge to stable dispositions that are regular polyhedrons. The results are shown in Figure 12.

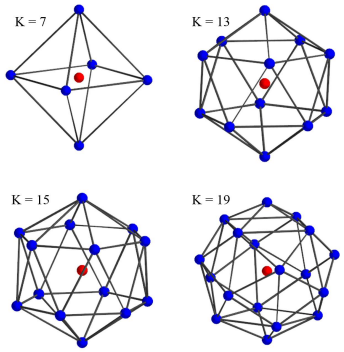


Figure 12: Illustration of the kernel points in stable dispositions [11]

Since this kernel position might not be optimal for every point cloud, the author made the kernel points deformable during training. They firstly use rigid KPConv to calculate an offset, apply the offset, and then do the convolution. Formally,

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}_x} g_{deform}\left(x_i - x, \Delta(x)\right) f_i$$

$$g_{deform}\left(y_i, \Delta(x)\right) = \sum_{k=1}^{K} h\left(y_i, \tilde{x}_k + \Delta(x)\right) W_k$$

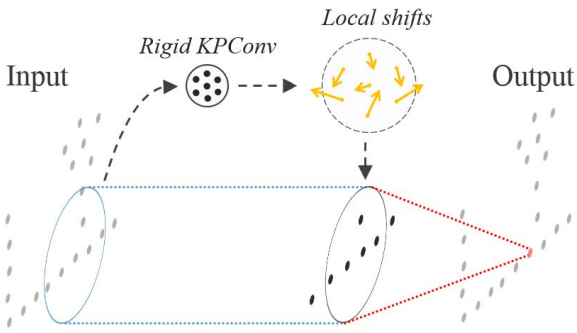


Figure 13: Illustration of KPConv [11]

The procedure is illustrated in Figure 13.

Directly use deformable kernel has a problem that if a kernel point has no near neighbor, its gradient will become NULL. To solve this issue, the authors proposed two loss. The first loss is to minimize the distance between kernel points and its nearest input point,

$$\mathcal{L}_{fit}(x) = \sum_{k=1}^{K} \min_{y_i} \left(\frac{\|y_i - (\tilde{x}_k + \Delta_k(x))\|}{\sigma} \right)^2$$

The second loss is to maximize the distances between kernel points,

$$\mathcal{L}_{rep}(x) = \sum_{k=1}^{K} \sum_{i \neq k} h\left(\tilde{x}_k + \Delta_k(x), \tilde{x}_i + \Delta_i(x)\right)$$

where h is defined as

$$h\left(y_i, \tilde{x}_k\right) = \max\left(0, 1 - \frac{\|y_i - \tilde{x}_k\|}{\sigma}\right)$$

The architecture of KP-FCNN is shown in Figure 10. In order to parallel the computation, the whole point cloud is divided into cells, and using the centroid as input. The strided KPConv layer acted as a pooling layer. The size of the cell is also doubled during the pooling. The σ in the h function is defined as $\sigma_j = \Sigma \times dl_j$ where dl_j is a the size of the cell.

Discrete Point Convolution Methods These methods define convolutional kernels on regular grids. One of representative works is PointCNN [13].

Suppose we have four points in a point cloud. Compared with a regular 2D image (Figure 15) The permutation of 2D grid is fixed, while the permutation of the point cloud could change. This is because the neighbors

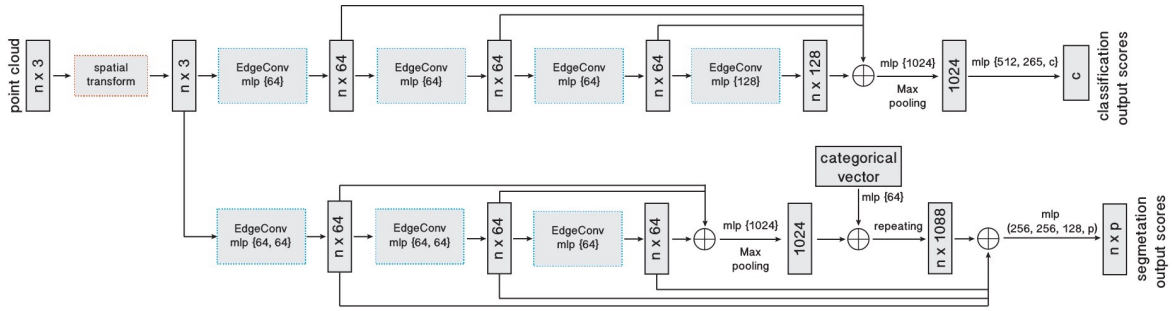


Figure 14: Architecture of DGCNN [12]

of point cloud is being sampled. Sampling could cause different index for the same point.

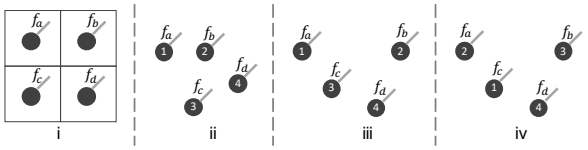


Figure 15: Convolution input from regular grids (i) and point clouds (ii-iv). [13]

If we apply a convolution on the point cloud,

$$f_{ii} = \text{Conv}\left(\mathbf{K}, [f_a, f_b, f_c, f_d]^T\right)$$
$$f_{iii} = \text{Conv}\left(\mathbf{K}, [f_a, f_b, f_c, f_d]^T\right)$$
$$f_{iv} = \text{Conv}\left(\mathbf{K}, [f_c, f_a, f_b, f_d]^T\right)$$

the output f_{iii} will not be the same with f_{iv}. Since they are in the same point cloud, we want them to have the same output. The permutation of the input is the main problem with point cloud convolution.

Authors tried to solve this problem by using a $\mathcal{X}$-transform matrix. After the transformation, the input will be permutation-invariant. Then, we could perform the convolution.

$$f_{ii} = \text{Conv}\left(\mathbf{K}, \mathcal{X}_{ii} \times [f_a, f_b, f_c, f_d]^T\right)$$
$$f_{iii} = \text{Conv}\left(\mathbf{K}, \mathcal{X}_{iii} \times [f_a, f_b, f_c, f_d]^T\right)$$
$$f_{iv} = \text{Conv}\left(\mathbf{K}, \mathcal{X}_{iv} \times [f_c, f_a, f_b, f_d]^T\right)$$

where $\mathcal{X}$ matrix is different for each input. For example, if $\mathcal{X}_{iii} = \mathcal{X}_{iv} \times \Pi$, and $[f_a, f_b, f_c, f_d]^T = \Pi \times [f_c, f_a, f_b, f_d]^T$, then f_{iii} will be equal to f_{iv}

The proposed operation is as follows. Suppose we have a query point p, and we want to calculate its feature. Its neighbors are $\mathbf{P} = (p_1, p_2, \cdots, p_K)^T$ and they have features $\mathbf{F} = (f_1, f_2, \cdots, f_K)^T$, which could be acquired by FPS or KNN. $\mathbf{K}$ is the kernel.

1. Convert the neighbor coordinates into the relative coordinates to p: $\mathbf{P'} \leftarrow \mathbf{P} - p$
2. Use a MLP_θ to convert neighborhood positional information to features: $\mathbf{F}_\theta \leftarrow \text{MLP}_\theta(\mathbf{P'})$
3. Concatenate the positional features to the original features: $\mathbf{F'} \leftarrow [\mathbf{F}, \mathbf{F}_\theta]$
4. Use a $\text{MLP}_\mathcal{X}$ to get a $\mathcal{X}$ matrix from neighborhood positional information: $\mathcal{X} \leftarrow \text{MLP}_\mathcal{X}(\mathbf{P'})$
5. Get a permutation-invariant features by multiplying matrix $\mathcal{X}$: $\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F'}$
6. Do convolution with kernel $\mathbf{K}$ and get output features at point p: $\mathbf{F}_p \leftarrow \text{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$

Or,

$$\begin{aligned}\mathbf{F}_p &= \mathcal{X}\text{-Conv}(\mathbf{K}, p, \mathbf{F}, \mathbf{P}) \\ &= \text{Conv}(\mathbf{K}, \text{MLP}_\mathcal{X}(\mathbf{P} - p \times [\text{MLP}_\theta(\mathbf{P} - p), \mathbf{F}]))\end{aligned}$$

3.3.4 Graph-based Methods

Graph convolution networks are better than CNN at point cloud data since it does not require data to be regular. It could also handle sparse data better and faster. Pioneer work in graph-based methods is DGCNN [12]. This work is also based on Pointnet.

The first contribution is that they proposed EdgeConv to aggregate local features (Figure 16). Suppose x_i is a point and we want to calculate its feature. Its

Methods			S3DIS			
			Area5 (OA)	Area5 (mIoU)	6-fold (OA)	6-fold (mIoU)
Projection-based Methods	Multi-view	DeePr3SS	-	-	-	-
	Spherical	RangeNet++	-	-	-	-
Discretization-based Methods	Volumetric	SEGCloud	-	48.9	-	-
	Permutohedral lattice	SPLATNet	-	-	-	-
		LatticeNet	-	-	-	-
Point-based Methods	Point-wise MLP	PointNet	78.2*	41.1	78.6	47.6
		PointNet++	80.1*	48.6*	81.0	54.5
	Point convolution	PointCNN	86.1*	57.3	88.3*	65.4
		KPConv	**87.5***	**67.1**	**89.7***	**70.6**
	RNN-based	3P-RNN	85.7	53.4	86.7*	56.3
	Graph-based	DGCNN	82.8*	54.6*	84.1	56.1

Table 1: Point cloud semantic segmentation results on the S3DIS (including both Area5 and 6-fold cross validation). Evaluation metrics are Overall Accuracy (OA) and Mean Intersection over Union (mIoU). The symbol '-' means the results are unavailable. The symbol '*' means the result is missing in the original paper or inconsistent with our testing.

k nearest neighbors are $\mathcal{N}_x = \{x_{j_{i1}}, x_{j_{i2}}, \cdots, x_{j_{ik}}\}$. We could construct a Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with x_i and $\mathcal{N}_x$ as vertexes and $\mathcal{E} = \{(i, j_{i1}), (i, j_{i2}), \cdots, (i, j_{ik})\}$. We define Edge features as $e_{ij} = h_\theta(x_i, x_j)$ where h_θ is a function with learnable parameters. Finally, we aggregate the edge features using a aggregation function $\square$:

$$x_i' = \underset{j:(i,j)\in\mathcal{E}}{\square} h_\theta(x_i, x_j)$$

As illustrated in Figure 16.

Here are some choices of h_θ and $\square$:

1. $h_\theta(x_i, x_j) = h_\theta(x_i)$ and $\square$ = MaxPool. This is Pointnet. It only summarizes global features and does not consider local features.
2. $h_\theta(x_i, x_j) = h_\theta(x_i - x_j)$ and $\square$ = MaxPool. This formulation only summarizes local features. It just consider point cloud as many pairs of points. The global geometry will be lost.
3. $h_\theta(x_i, x_j) = h_\theta(x_i, x_i - x_j)$ and $\square$ = MaxPool. This is the architecture authors chose. It considers both global and local features.

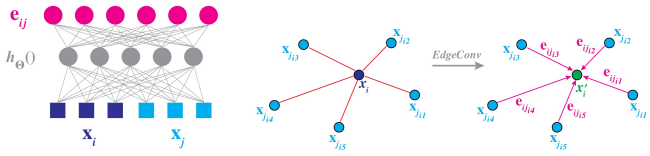


Figure 16: Illustration of EdgeConv [12]

The second contribution is the dynamic graph. After each EdgeConv, the features of each point changed to another space. Authors reconstructs the graph after each EdgeConv by KNN. Thus, the graph is dynamic. The overall architecture of the DGCNN is shown in Figure 14.

4 Experiments

We used S3DIS dataset as our benchmark dataset and tried the aforementioned models on a Nvidia RTX 3090 with 24 Gigabit of GPU memory. The results are shown in the Table 1. The S3DIS dataset has 6 different areas, each area contains some rooms. The Area-5 means left out Area-5 during the training and use all the rooms

in Area-5 as testing dataset. 6-fold method is splitting the dataset into 6 folds and do a 6-fold cross validation. The metrics used were OA and mIOU, which is introduced in Section 2. We marked the results that were missing in the original paper or inconsistent with our testing. According to the table, the best perform model is KPConv model.

We also tried to do prediction on our own point cloud data. We did a 3D scan using the LiDAR sensor on iPad Pro 2020 to our classroom. The scanning result is shown in Figure 17. We chose a KPConv model pre-trained on S3DIS dataset to do segmentation. The result is shown in Figure 18. The model did well in Desk, Chair, and Floor category, but failed on Door, Wall, and Ceiling. The low quality of the scan could cause these issues.

5 Conclusion

This paper presented a survey on point cloud segmentation. This field has been gaining more and more attention. Also, some ideas of point cloud could be used in different fields, such as molecular simulation. We hope this paper could offer some insights about point cloud and help researchers to further explore this field.

Member Contributions C completed the Section 3.1 (Projection-based methods), Section 3.2.2 (Discretization-based methods), and part of the Introduction. Q completed the rest.

Acknowledgement We thank Prof. Z and Prof. D for the awesome class!

Figure 17: 3D scan of our classroom

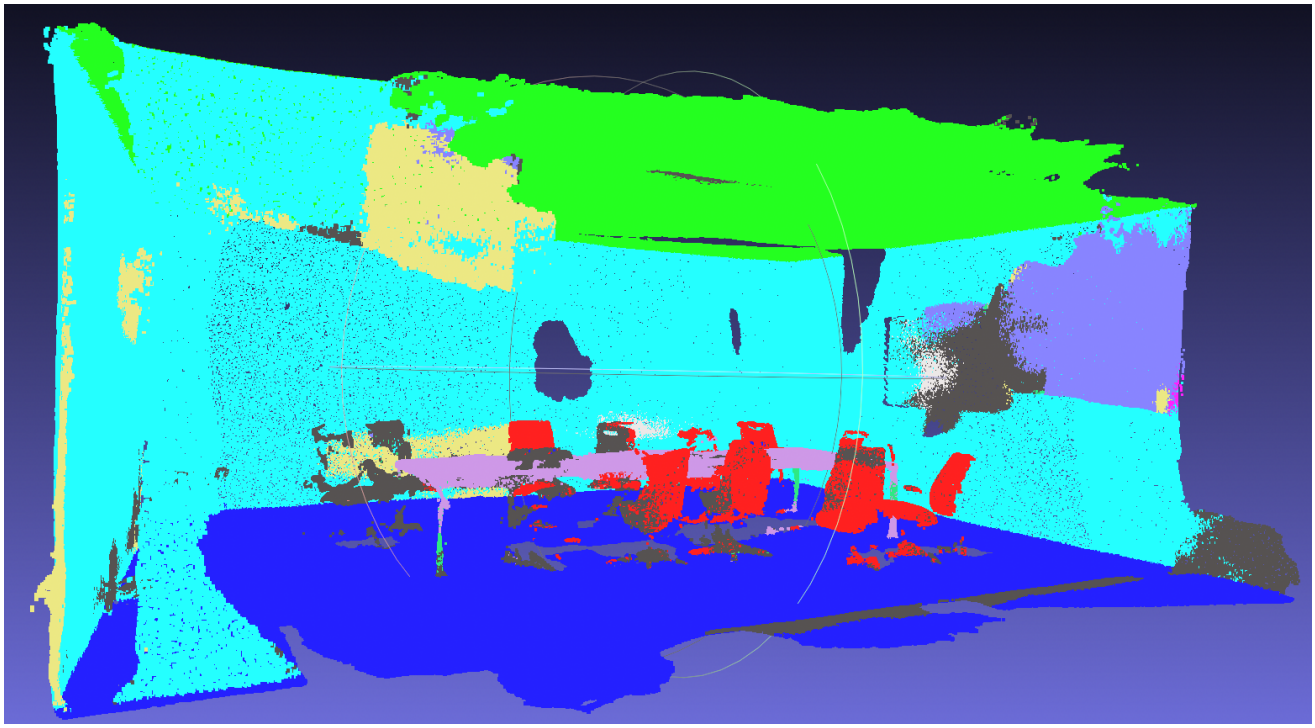


Figure 18: Result of segmentation. The colors ant their representing class are: Light Green → Ceiling. Cyan → Wall. Red → Chair. Light Purple → Desk. Blue → Floor. Yellow → Door.

References

[1] S. A. Bello, S. Yu, and C. Wang, "Review: deep learning on 3d point clouds," *CoRR*, vol. abs/2001.06280, 2020.

[2] 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 International Conference on Computer Vision and Pattern Recognition*, 2016.

[3] H. Thomas, F. Goulette, J.-E. Deschaud, B. Marcotegui, and Y. LeGall, "Semantic classification of 3d point clouds with multiscale spherical neighborhoods," in *2018 International conference on 3D vision (3DV)*, pp. 390–398, IEEE, 2018.

[4] F. J. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. S. Khan, and M. Felsberg, "Deep projective 3d semantic segmentation," in *CAIP*, 2017.

[5] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[6] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "Rangenet ++: Fast and accurate lidar semantic segmentation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4213–4220, 2019.

[7] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, "Splatnet: Sparse lattice networks for point cloud processing," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2530–2539, 2018.

[8] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

[9] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," 2017.

[10] 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 (ECCV)*, pp. 403–417, 2018.

[11] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6411–6420, 2019.

[12] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[13] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," *Advances in neural information processing systems*, vol. 31, pp. 820–830, 2018.