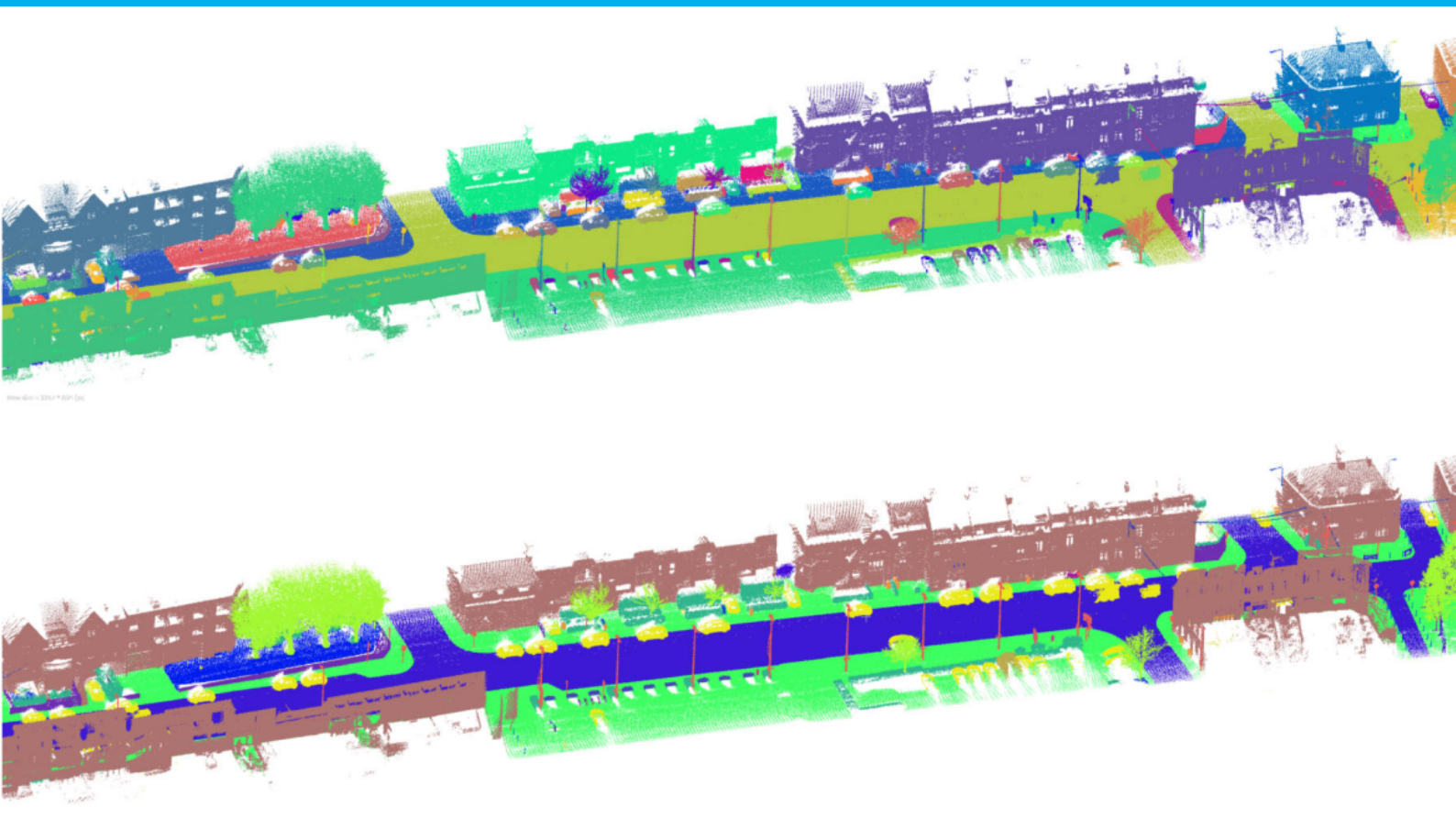
Master Thesis

Semantic Segmentation of Large-scale Urban Scenes from Point Clouds

Zhiwei Ai



TU Delft

Semantic Segmentation of Large-scale Urban Scenes from Point Clouds

by

Zhiwei Ai

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday July 29, 2019 at 10:30 am.

Student number:	4699076	
Project duration:	September, 2018 – July, 2019	
Thesis committee:	Prof. dr. D. M. Gavrila,	TU Delft
	Dr. L. Nan,	TU Delft
	Dr. R. C. Lindenbergh,	TU Delft

TU Delft

Abstract

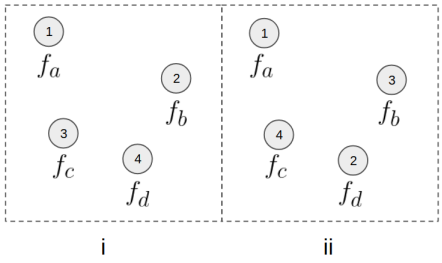
Deep learning methods have been demonstrated to be promising in semantic segmentation of point clouds. Existing works focus on extracting informative local features based on individual points and their local neighborhood. They lack consideration of the general structures and latent contextual relations of underlying shapes among points. To this end, we design geometric priors to encode contextual relations of underlying shapes between corresponding point pairs. Geometric prior convolution operator is proposed to explicitly incorporate the contextual relations into the computation. Then, GP-net, which contains geometric prior convolution and a backbone network is constructed. Our experiments show that the performance of our backbone network can be improved by up to 6.9 percent in terms of mean Intersection over Union (mIoU) with the help of geometric prior convolution. We also analyze different design options of geometric prior convolution and GP-net. The GP-net has been tested on the Paris and Lille 3D benchmark, and it achieves the state-of-the-art performance of 74.7% mIoU.

Contents

1 Introduction

Semantic segmentation of images assigns each pixel a class label, which has been applied to autonomous driving, surveillance, medical analysis, and other fields. Deep leaning methods [25, 27, 31, 37, 53, 54] achieve a success in this area, and pushes this area into a limit. Due to the lack of spatial information, the applications of image semantic segmentation are restricted.

As a supplement to images, point cloud data can provide spatial information directly. Point cloud semantic segmentation plays a crucial role in computer vision. It can serve as a clue for scene understanding to provide a robust perception of the environment. Autonomous driving requires HD maps for accurate localization, and semantic segmentation of point clouds can be applied to provide more detailed environment information. 3D reconstruction is important for the construction of 3D cities. Semantic segmentation of point clouds with good quality can significantly improve the accuracy of 3D reconstruction. Traditional methods of point cloud semantic segmentation require training a classifier on hand-crafted features, and the performance of these methods faces the bottleneck. Deep learning, as an alternative, has been attracting researchers' attention on the semantic segmentation of point clouds. It has been proved to be promising in this field. However, it is still a challenging task.



$$\mathbf{K} = \left[k_\alpha, k_\beta, k_\gamma, k_\delta\right]^T$$
$$f_i = \mathrm{Conv}\left(\mathbf{K}, [f_a, f_b, f_c, f_d]^T\right)$$
$$f_{ii} = \mathrm{Conv}\left(\mathbf{K}, [f_a, f_d, f_b, f_c]^T\right)$$
$$f_i \neq f_{ii}$$

Figure 1.1: An example of permutation variance of a regular convolution applied on a point cloud

On the one hand, a point cloud is irregular and unordered, which results in permutation variance when a regular convolution is applied to process the point cloud. In Figure 1.1, i and ii represent two same point clouds with corresponding features $\mathbb{F} = \{f_a, f_b, f_c, f_d\}$, but the index orders are different. When two point clouds are convolved with a same regular convolution kernel $\mathbf{K}$, it results in $f_i \neq f_{ii}$. For this issue, a group of methods [3, 39, 51] proposes to convert a point could into regular grids in images, and make use of well-developed image CNNs [1, 13] to make predictions. While some of the works [6, 14, 29, 55] focus on transforming a point cloud into regular grids in 3D voxels to process the point cloud. Generally, transformation of a point cloud into images or voxels causes loss in spatial information, which limits the resolution of predictions. In order to process a point cloud directly using neural networks, a set of works, such as PointNet [33], and [16, 26, 34, 42, 46], uses a symmetric function to achieve permutation invariance. PointCNN [23] proposes to learn a $\mathcal{X}$-transformation matrix

to permute and weight point features into a latent and potentially canonical order. SO-Net [22] learns a permutation invariant self-organizing map to achieve permutation invariance. In a word, permutation variance is solved by various ways. Compared with methods that require data transformation, point-based methods can preserve more original and internal information from points.

On the other hand, many of the current point-based methods [16, 22, 23, 34] focus on extracting effective local features in a local structure representation usually generated by point searching methods, such as kNN, and ball query. They lack consideration of latent contextual relations among points. Superpoint Graph [21] is the pioneering method harnessing the contextual relations in a point cloud to make predictions.

Inspired by current point-based methods and Superpoint Graph, we design permutation invariant geometric prior convolution operator that can explicitly incorporate contextual relations between the underlying shapes of corresponding point pairs. GP-net, a point-based U-type network containing a backbone network and geometric prior convolution, is proposed. We show that GP-net can achieve state-of-the-art performance on the point cloud semantic segmentation task. Compared to the current point-based methods, GP-net makes use of contextual relations between underlying shapes of points. Unlike Superpoint Graph, the point pairs to provide contextual relations are dynamically constructed, instead of pre-defined, for each layer of geometric prior convolution, and the final predictions are made point-wise.

2

Literature

Deep learning has been attracting researchers' attention since 2012. One neural network-based method [18] outperformed other traditional machine learning methods by a large margin on Imagenet benchmark [9]. Neural networks, as the hero behind the impressive performance, has been studied for decades. In this chapter, the basic components and relevant knowledge needed to understand the structure of neural networks are introduced. Then several state-of-the-art methods for semantic segmentation of point clouds are discussed.

2.1. Background on deep learning

2.1.1. Perceptron and activation function

The basic unit of neural networks is the artificial neuron. The artificial neuron, which imitates the process of information flowing through a biological neuron, was first proposed by McCulloch and Pitts's work [30] in 1943. Based on this work, a neuron named perceptron [38], was designed by Frank Rosenblatt in 1958.

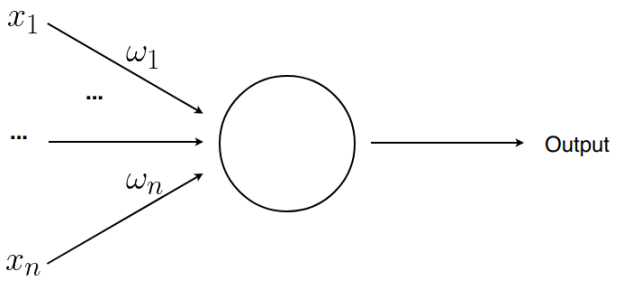


Figure 2.1: The model of a perceptron

A perceptron generally has several inputs and one output. For each input x_i, there is a corresponding weight ω_i to indicate how important the input is. The weighted sum of inputs are computed in the perceptron, and the output is decided by comparing the weighted sum of inputs with a threshold θ. The output is binary. When the weighted sum is greater than the threshold, the neuron is activated and the output is 1.

$$output = \begin{cases} 0, & \sum_{i=1}^{n} \omega_i x_i \leq \theta \\ 1, & \sum_{i=1}^{n} \omega_i x_i > \theta \end{cases} \tag{2.1}$$

The above mathematical model is the very initial representation of a perceptron. This representation can be changed into a more common version. If the inputs and their corresponding

weights are in two vectors, the weighted sum can be substituted by dot product as $\omega \cdot x$. Furthermore, the threshold can be moved to the left side of inequality and replaced by a bias. Then, the perceptron can be written as a more common version.

$$output = \begin{cases} 0, & \omega \cdot x + bias \leq 0 \\ 1, & \omega \cdot x + bias > 0 \end{cases} \tag{2.2}$$

To simplify the above function, it can be written as $sgn(\omega \cdot x + bias)$. $sgn(x)$ is denoted as a step function shown below, and it is here also called an activation function since it decides the final output.

$$sgn(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases} \tag{2.3}$$

Using a step function as an activation function has many negative properties. Step function is not continuous, and very small changes of weights and bias can lead to a completely different output, for example, from 0 to 1. These problems can be overcome by using other activation functions, for example sigmoid function $\sigma(x)$. Similar to perceptron, a sigmoid neuron can be denoted as $\sigma(\omega \cdot x + bias)$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

In modern neural networks, neurons are not initial perceptrons with step functions but neurons with advanced activation functions, such as sigmoid, tanh, ReLU. When many of the neurons are connected in various orders and structures, different neural networks can be constructed.

2.1.2. Multi-layer perceptrons

Multi-layer perceptrons comprise at least three layers, one input layer, one hidden layer, and one output layer. The depth of multi-layer perceptrons can be increased by stacking several hidden layers. For the example shown in the figure, the input layer has three input neurons, and the output layer has one output neuron. Two hidden layers have respectively 5 and 4 neurons. The number of neurons in each layer can be adjusted case by case. The neurons of previous layer are fully connected to the neurons in the subsequent layer. Even though this structure is called multi-layer perceptrons, the neurons applied here are not original perceptrons but neurons with advanced activation functions, such as sigmoid, and ReLU.

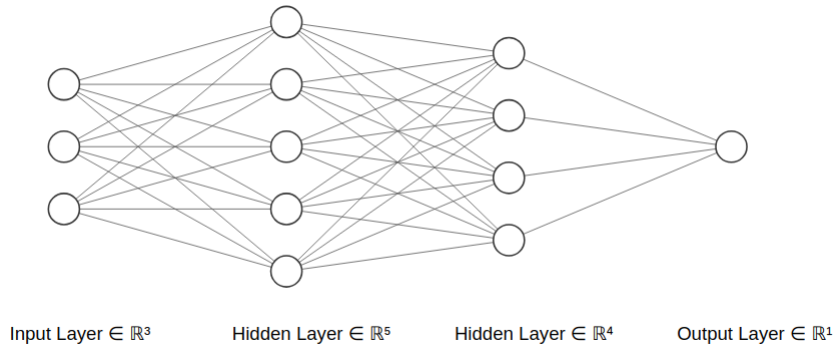


Figure 2.2: An example of MLPs

2.1.3. Convolution layer

Instead of full connections between neurons in the previous layer and successive layer, local connectivity of neurons are applied in convolution layers. This way, computation and parameters can be significantly reduced for a large number of input. One of the most successful

applications of convolution is on images. A common definition of image convolution is,

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t) \tag{2.5}$$

where $f(x,y)$ is a source image, and $g(x,y)$ is the image after convolution. The location of a pixel on the image is marked by x, and y values. ω is the convolution filter kernel, also called the weights for the image to convolve with. ω always has three dimensions, namely, width, height, and depth. In the equation, $2a$ is the width of convolution kernel and $2b$ is the height of the convolution kernel. The figure below shows the convolution when the depth of kernel is 1, and the depth of the kernel determines the output feature channel of the convolved image.

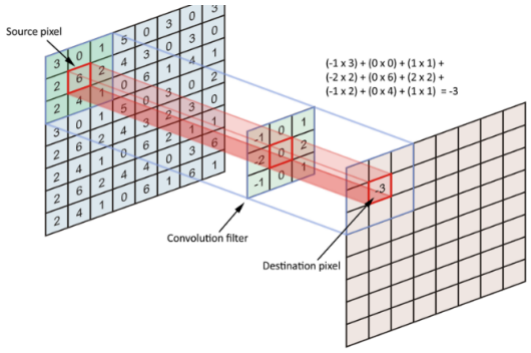


Figure 2.3: An example of image convolution

2.1.4. Pooling layer

The function of applying pooling layer in neural networks is to down-sample the size of input in order to save computation and aggregating features. The operation of pooling works on the width or height for images or on the number of points for point tasks, but not on the feature channels. Max pooling is a commonly used pooling operation. As in the figure below, max pooling with a 2×2 filter, and stride 2 can down sample a 4×4 image into 2×2 by selecting the maximum value corresponding to each filter region.

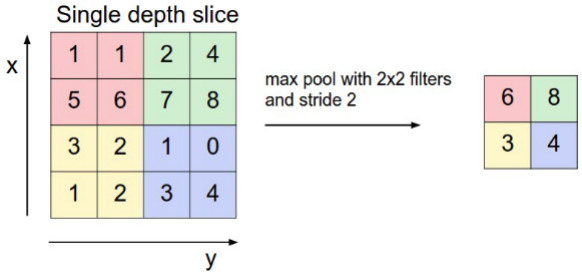


Figure 2.4: An example of max pooling

2.1.5. Softmax and loss function

Softmax

Given the task of semantic segmentation of point clouds. We need to classify p points into n classes, and it exists a class $c \in \mathcal{C}$. $\mathcal{C}$ is a vector of all classes. For classification tasks, the output channels at the end of the neural network are usually aligned with the number of classes n, so that the neural network will output a vector with dimensions of n for each point, X_i for point i. Softmax function is often used to project raw outputs of the neural network to

probability distribution. After using softmax to process each element in X_i, each element in the resulting vector represents the probability of this point belonging to a respective class.

$$f_i(c) = \frac{e^{X_i(c)}}{\sum_{c' \in \mathcal{C}} e^{X_i(c')}} \quad \forall i \in [1,p], \forall c \in \mathcal{C} \tag{2.6}$$

In the softmax function, for a point $i \in [1,p]$, f_i is the n dimension probability distribution vector after softmax. $f_i(c)$ denotes the probability of point i belonging to class $c \in \mathcal{C}$. $X_i(c)$ is the element in X_i that corresponds to class c. The results of softmax are normalized between $(0,1)$. Clearly, all the elements in the f_i add up to 1. The classification results or the loss can be computed after softmax.

Cross-entropy loss

When training a neural network, an objective is needed for adjusting the weights and biases. The objective is usually to minimize the loss function. In order to have a good performance in terms of metrics that evaluate the neural network, the loss function should be chosen carefully. Cross-entropy is a common loss function in neural networks.

$$C = -\frac{1}{p}\sum_{i=1}^{p} \log f_i\left(y_i^*\right) \tag{2.7}$$

For the task of semantic segmentation of point clouds, the cross-entropy loss C for all the p points, is the mean of negative log likelihood of all these points, with each point $i \in [1,p]$, y_i^* the ground truth of point i. Using an optimizer to minimize cross-entropy loss equals to applying maximum likelihood estimation on the neural network [10], which optimizes the parameters to maximize the log likelihood.

2.2. Related work

Point clouds are irregular, which makes semantic segmentation of point clouds a challenging problem. In traditional methods, a classifier is trained on hand-crafted features. In deep learning, different methods were proposed in recent years. Based on how neural networks consume the data, current deep learning methods can be roughly categorized into voxel-based methods, image-based methods, graph-based methods, and point-based methods.

2.2.1. Semantic segmentation based on hand-crafted features

The general pipeline of machine learning methods requires training a classifier on hand-crafted features, and then the trained classifier can be applied to label the test data. The performance of machine learning methods mainly relies on the quality of hand-crafted features as well as the choice of the classifier. Therefore, extracting informative and discriminant features that represent the properties of the data effectively and selecting a smart classifier that can make use of the features comprehensively remain challenges to overcome. In the following section, several hand-crafted features and two methods are introduced.

Hand-crafted features

For a specific point, and its selected neighbor points, a covariance matrix can be easily computed. Eigenvalues $\lambda_1 > \lambda_2 > \lambda_3$ can be calculated afterwards. Many of the features are based on the eigenvalues. Such a covariance matrix is sometimes mentioned as a structure tensor, such as in [47, 50], and their relevant features are called structure tensor features. Both the eigenvalues and their related features have abilities to represent the local structure geometric properties. According to [47], 3D features are listed in the Table 2.1.

Among the features, Verticality was initially defined in [8]. n_Z is the third component of the normal vector of the local structure. Verticality provides information about points and a flat vertical area [8]. Therefore, it has the potential to distinguish facades from grounds [47]. Change of curvature, also called surface variation, was defined in [32]. This measurement

Sum of eigenvalues Σ_λ	$\lambda_1 + \lambda_2 + \lambda_3$
Omnivariance O_λ	$(\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
Eigenentropy E_λ	$-\sum_{i=1}^{3} \lambda_i \cdot ln(\lambda_i)$
Anisotropy A_λ	$(\lambda_1 - \lambda_3)/\lambda_1$
Planarity P_λ	$(\lambda_2 - \lambda_3)/\lambda_1$
Linearity L_λ	$(\lambda_1 - \lambda_2)/\lambda_1$
Curvature C_λ	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
Sphericity S_λ	λ_3/λ_1
Verticality V	$1 - n_Z$
Local point density D	$(k+1)/(\frac{4}{3}\pi r_{k-NN}^3)$

Table 2.1: 3D features mentioned in [47]

demonstrates if the points of the local neighborhood are in a plane. If a change of curvature equals 0, all points are in the same plane. If it reaches the maximum, $\frac{1}{3}$, it means the points are isotropically distributed. Thus, it can help us discriminate between planar structures and non-planar structures. The linearity determines how well the points fit a linear structure, and the planarity describes the smoothness of the points. Sphericity provides information on the existence of a volumetric structure, and omnivariance also describes the volumetric distribution of points [45]. Eigenentropy indicates if the local point cloud is ordered or unordered [49].

Examples of methods

Covariance	Sum	$\lambda_1 + \lambda_2 + \lambda_3$
	Omnivariance	$(\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
	Eigenentropy	$-\sum_{i=1}^{3} \lambda_i \cdot ln(\lambda_i)$
	Anisotropy	$(\lambda_1 - \lambda_3)/\lambda_1$
	Planarity	$(\lambda_2 - \lambda_3)/\lambda_1$
	Linearity	$(\lambda_1 - \lambda_2)/\lambda_1$
	Surface Variation	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
	Sphericity	λ_3/λ_1
	Verticality	$1 - \lvert\langle[001], e_3\rangle\rvert$
Moment	$1^{st}order, 1^{st}axis$	$\sum_{i\in P} \langle P_i - P, e_1\rangle$
	$1^{st}order, 2^{nd}axis$	$\sum_{i\in P} \langle P_i - P, e_2\rangle$
	$2^{nd}order, 1^{st}axis$	$\sum_{i\in P} \langle P_i - P, e_1\rangle^2$
	$2^{nd}order, 2^{nd}axis$	$\sum_{i\in P} \langle P_i - P, e_2\rangle^2$
Height	Vertical range	$Z_{max} - Z_{min}$
	Height below	$Z - Z_{min}$
	Height above	$Z_{max} - Z$

Table 2.2: Features used in [12]

In the work [12] proposed by Hackel et al., in order to efficiently and effectively extract features for points, the neighborhood of a specific point is considered by a multi-scale scheme. The point cloud is down-sampled several times and the density of the point cloud is decreasing each time. The reduced point cloud density requires less computation for further processing. After each down-sampling, the neighborhood points in this scale are selected using kNN. The authors think that kNN can avoid the drawbacks caused by the varying density of the point cloud, and thus it can be considered an adaptive radius for searching. For one single point, 10 nearest points of 9 scales are taken into account, which results in 144 feature dimensions in total. The features for each scale is listed in Table 2.2. For the contour edges, the before-mentioned surface property features may be insufficient. Therefore, histogram-based descriptors are used specifically for contour edges. A random forest classifier is implemented

for classification, and its effectiveness has been approved in [4, 48].

Based on the previous work [12], Thomas et al. presented a new work [43] that modified the point neighborhood definition. Instead of kNN, multi-scale spherical neighborhood that contains more geometrical meaning is defined. Spherical neighborhood searches points in an area with a fixed radius, which may reach two extremes. When a point cloud is dense and the scale is large, included points may be too many to compute. When a point cloud is sparse and the scale is small, included points may be too less to extract features. The first problem can be solved by down-sampling, similar to [12]. The second problem can be solved by making use of different scales. The features used in this work are slightly different than previous one, as shown in Table 2.3. By also implementing a random forest classifier, the method achieves a competitive performance.

Covariance	Sum	$\lambda_1 + \lambda_2 + \lambda_3$
	Omnivariance	$(\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
	Eigenentropy	$-\sum_{i=1}^{3} \lambda_i \cdot ln(\lambda_i)$
	Linearity	$(\lambda_1 - \lambda_2)/\lambda_1$
	Planarity	$(\lambda_2 - \lambda_3)/\lambda_1$
	Surface Variation	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
	Verticality(×2)	$\left\lvert \frac{\pi}{2} - angle(e_i, e_z) \right\rvert_{i\in(0,2)}$
	Sphericity	λ_3/λ_1
Moment	Absolute moment (×6)	$\frac{1}{\lvert N\rvert} \left\lvert \sum \langle P - P_0, e_i \rangle^k \right\rvert_{i\in(0,1,2)}$
	Vertical moment (×2)	$\frac{1}{\lvert N\rvert} \sum \langle P - P_0, e_z \rangle^k$
Others	Number of points	$\lvert N\rvert$
	Average color (×3)	$\frac{1}{\lvert N\rvert} \sum c$
	Color variance (×3)	$\frac{1}{\lvert N\rvert - 1} \sum (c - \bar{c})^2$

Table 2.3: Features used in [43]

The performance of the traditional methods is constrained by the power of classifier and effectiveness of hand-crafted features. Deep learning demonstrates a more powerful representation ability and feature learning ability.

2.2.2. Voxel-based methods

VoxNet [29] is a pioneering effort using 3D convolutional networks for point cloud processing. The input to the network is pre-segmented objects, and the network can predict labels for them. The network consists of two parts. The first part is a volumetric grid representing. Volumetric occupancy grid is used because it can provide information about both occupied and unoccupied spaces, and it also can be calculated in an efficient data structure. The second part is a 3D convolution network that labels the object. 3D CNN is considered because spatial structures are expected to be learned explicitly by the network. A hierarchical network can be designed by adding more layers of convolution with different sizes of kernels, which can make the network concatenate information from different scales.

After VoxNet, Huang et al. presented a new 3D convolutional neural network [14] that can comsume large-scale point clouds for labelling. It is an end-to-end voxel-based method that requires no segmentation and hand-crafted features. They came up with two different voxelization methods for training and testing phases. For training, in order to avoid the bias caused by dense sampling, they select the same number of center points from each class of objects. After choosing the center points, cubic bounding boxes are generated and then cubes are separated into small grids of cells. For testing, dense voxelization is used. This method can classify the whole scene with a high accuracy and computation efficiency.

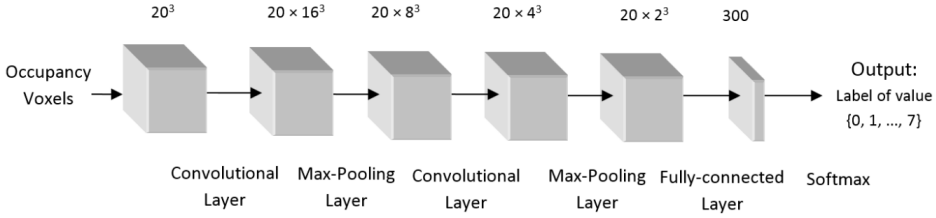


Figure 2.5: The architecture of network in [14]

In later works, ScanNet [6] proposed a 3D convolutional network that can predict labels for columns. When testing, the network is moving on the x-y plane to make the whole scene predicted. The predictions are based on information from the targeted voxel and its neighborhood voxels. Zhou et al. proposed a voxel-based end-to-end method, VoxelNet [55] that can learn feature representations automatically for region proposal networks to accomplish the point cloud object classification.

Though voxel-based methods have many applications, transforming a point cloud into voxel grids requires additional computation. Besides, transformation causes a loss in spatial information and a decrease in resolution. Some works are presented in order to produce a high resolution representation. For example, [36] uses Octree data structure to achieve a high resolution.

2.2.3. Image-based methods

Despite voxel-based representation methods, another way to transform a point cloud into an ordered data type is to generate images from a point cloud.

Yang et al. presented a semantic segmentation method [51] specifically for ALS point clouds. This work can be considered as a combination between traditional feature selection and deep learning methods. The method mainly consists of two parts, point-based feature image transformation, and CNNs that are intended to extract hierarchical features of the feature images. Three types of features, including local geometric features, global geometric features as well as full-waveform Lidar features are calculated and transferred into RGB values to generate feature images. After the feature images are generated, CNNs consume the feature images and abstract high-level features for labeling.

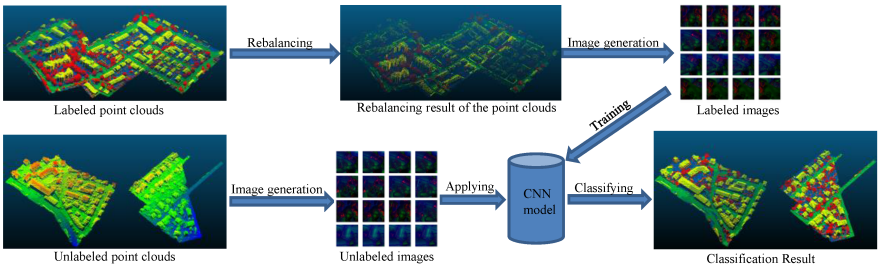


Figure 2.6: The pipeline of network in [51]

Another work that adopts the same idea of transferring a point cloud into meaningful feature images to do semantic segmentation is SnapNet [3]. As the pipeline shows in Figure 2.7, a point cloud is first pre-processed. In order to boost the efficiency of processing a point cloud, voxelization is applied and the closest point to the center of each voxel is kept so that the down-sampled point cloud has an uniform density. Then, the meshes are constructed with textures. Textures contain RGB, normal deviation to vertical, and a noise estimation. Snapshots are then generated containing two different images, RGB images, and images from

depth composite. The images are next fused and fed into SegNet [1] based network for pixel-wise segmentation. The point-wise labels are obtained by back-projection of 2D results.

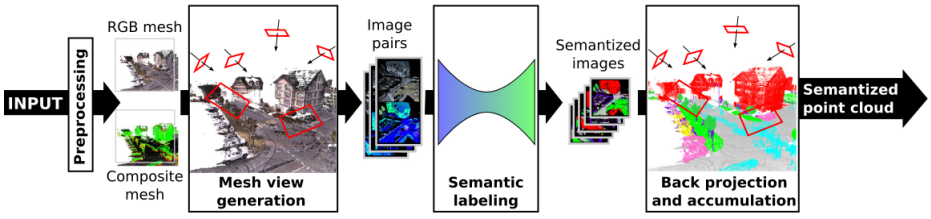


Figure 2.7: The pipeline of network in SnapNet [3]

Roveri et al. presented a novel end-to-end network [39] that can automatically generate informative depth images for a point cloud instead of selecting hand-crafted features to generate feature images. From the pipeline of the network, we can see that the network mainly consists of three stages. The network first takes an input point cloud and predicts its representative directions. In the second stage, the network generates depth images for each direction of the point cloud automatically. Then, several ResNet50 [13] architectures are used to predict class labels based on the depth images in the third stage.

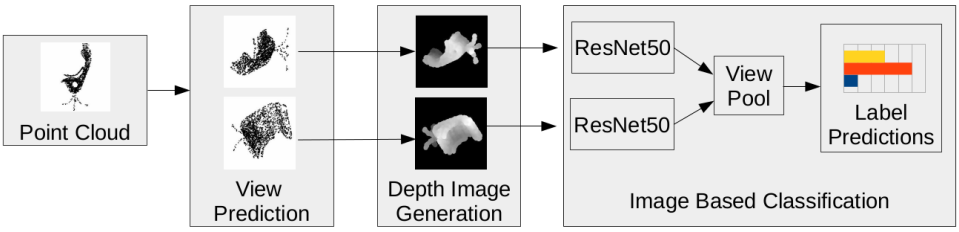


Figure 2.8: The pipeline of network in [39]

One of the advantages of before-mentioned image-based methods is that they can harness well-studied existing 2D methods. However, their performance is related to the quality of the feature images, and the network lacks consideration of spatial information provided by point clouds internally.

2.2.4. Point-based methods

Using neural networks to process point clouds can make use of spatial information directly. The challenge is to solve permutation variance. [33, 34, 42] use a symmetric function to achieve permutation invariance. [23] uses MLPs to learn a $\mathcal{X}$-transformation matrix to achieve permutation equivariance. [16] gives points specific orders corresponding to orientations, so that the computation does not suffer from permutation variance.

PointNet

PointNet [33] is the first attempt to use neural networks to process a point cloud directly, while the previous methods need to convert a point cloud into other data representations, for example, voxels. Processing a point cloud directly keeps rich spatial information in x, y, z coordinates, and prevents transforming a point cloud into a redundant data form. The key of PointNet is the symmetric function, max pooling. Max pooling helps the network achieve permutation invariance, which is a challenge of using deep learning on point clouds.

For segmentation tasks, PointNet takes a fixed number of points as input, so that a point cloud needs to be down-sampled first. T-nets in the network make the points invariant of geometric transformations. Multi-layer perceptrons lift each point into a high dimensional

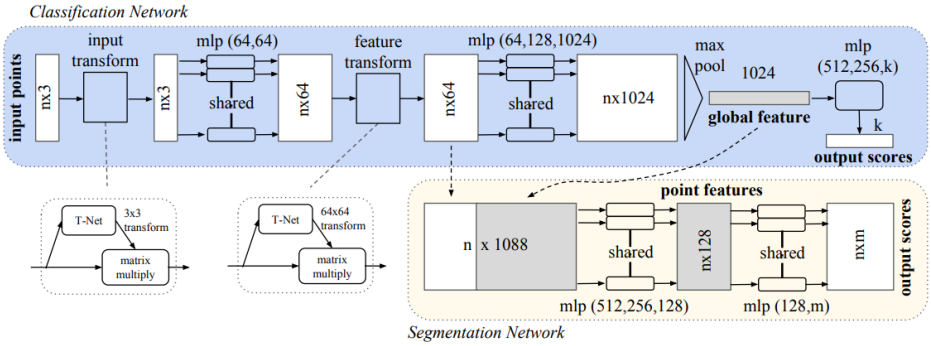


Figure 2.9: The architecture of PointNet [33]

feature vector. Max pooling extracts the maximum of each channel to generate a global feature vector for the input points. Then, global features are concatenated with point features to supply each point with global information. After several layers of multi-layer perceptrons, points are assigned with labels.

PointNet++

Though PointNet achieves an impressive performance on point cloud analysis, it is incompetent to consider local features and multi-scale features. This may result in a poor performance when dealing with data with fine-grained patterns. Otherwise, the features extracted by PointNet are in an uniform manner, which limits the performance on data with uneven densities. Based on the success of PointNet as an effective and robust local feature extractor, PointNet++ [34] was proposed to enhance the PointNet by extracting multi-scale local features. The network comprises several layers of set abstraction structure. Specifically, it consists of a sampling layer, a grouping layer, and a simplified PointNet.

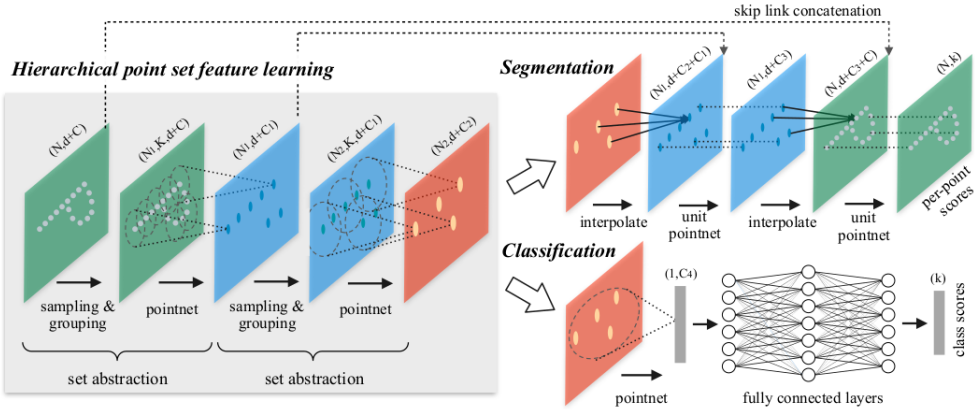


Figure 2.10: The architecture of PointNet++ [34]

Sampling layers select a set of central points using farthest point sampling. Grouping layers can find neighborhood points for selected central points, and group respective features for further processing. PointNet is utilized to extract local features for grouped local neighborhood points. For the semantic segmentation task, PointNet++ achieves a hierarchical structure by decreasing the number of central points while projecting local features into central points. The high level global features are then interpolated to original points with the help of skip connections. The experiments show that, PointNet++ achieves better performance than PointNet, and it is robust to uneven sampling densities.

PointSIFT

PointNet++ sets a new standard for point cloud semantic segmentation. However, the ball query searching used in the grouping layer of PointNet++ overlooks the possible unbalanced

distribution of a point cloud in different directions. PointSIFT [16] was proposed to solve this problem. PointSIFT is inspired by both point-based methods and SIFT [28] descriptor that is widely used in 2D images. Similar to SIFT in 2D, given a point p, PointSIFT module computes a description of a set of points centered at p. A difference from SIFT is that PointSIFT module can be trained end-to-end.

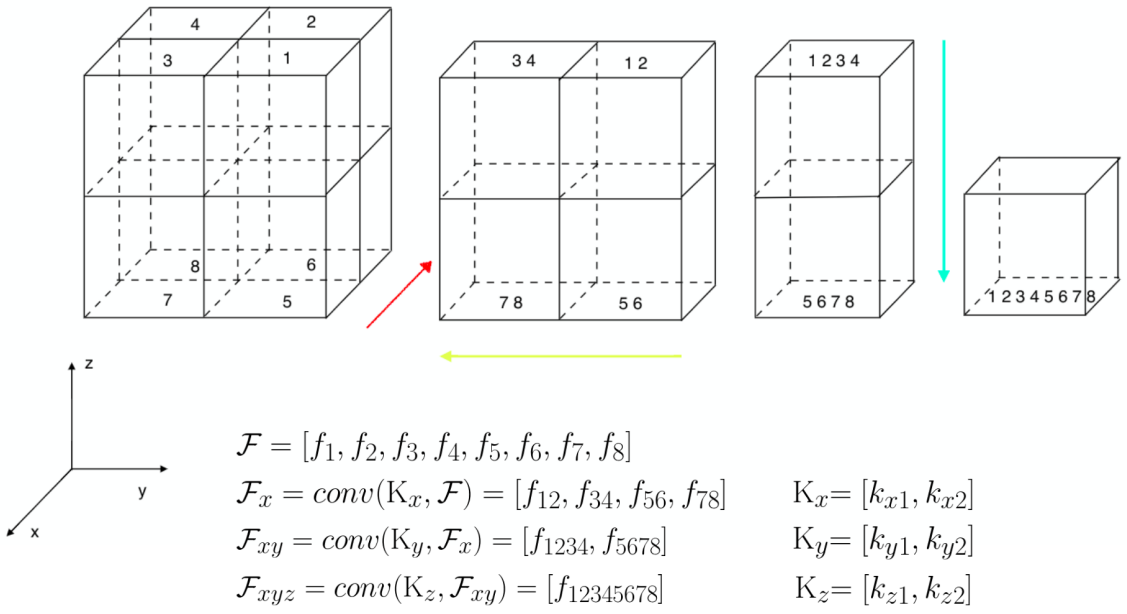


Figure 2.11: An illustration of orientation encoding unit [16]. $\mathcal{F}$ is the feature matrix of 8 points, with size $1 \times 8 \times d$. $\mathcal{F}_x$, $\mathcal{F}_{xy}$, $\mathcal{F}_{xyz}$ are corresponding feature matrices after convolutions. K_x, K_y, K_z, are three convolution kernels.

The key to the PointSIFT module is the orientation encoding unit, which consists of an 8 orientation search and a three-stage orientation encoding convolution. For a center point p, space is partitioned into 8 octants by x, y, z local coordinates. 8 orientation search searches one nearest point from each of the 8 orientations in a fixed radius to generate a feature matrix $\mathcal{F}$ of $1 \times 2 \times 2 \times 2 \times d = 1 \times 8 \times d$ dimensions. $2 \times 2 \times 2$ represents x, y, z axes, and there are two directions, positive as well as negative, for each axis. Every point has d dimension features. Since the point features are stored in a certain order, the orientation encoding convolution does not suffer from permutation variance. Three 2D convolutions with the filter size $1 \times 2 \times d$ and the stride 1×2 are applied on the point features to integrate information in x, y, z directions accordingly. The illustration of the three-stage convolution is demonstrated in Figure 2.11.

After the convolution, the features of local neighborhood points in 8 orientations are integrated on the center point. Though one orientation encoding unit takes merely 8 points into consideration, if i units are stacked in the PointSIFT module, theoretically 8^i points are computed. This property allows the network to choose the most adaptive scales.

PointCNN

CNNs cannot be applied on points directly because CNNs require ordered data type. Li et al. presented PointCNN [23] that overcomes the unordered property of point clouds. The key to the performance of PointCNN is $\mathcal{X}$-Conv.

For input points with features, farthest point sampling is used to find a set of representative points for input points. $\mathcal{X}$-Conv can be considered as a local feature extractor that projects the information of neighborhood points into the representative points. $\mathcal{X}$-Conv can be applied on the point clouds repeatedly, and their functions are similar to applying convolutions on images. As $\mathcal{X}$-Conv is applied recursively, the number of points is reduced gradually, while

Algorithm 1 $\mathcal{X}$-Conv Operator

Input: $\mathbf{K}, p, \mathbf{P}, \mathbf{F}$

Output: $\mathbf{F}_p$ — Features 'projected', or 'aggregated', into representative point p

1: $\mathbf{P}' \leftarrow \mathbf{P} - p$ — Move $\mathbf{P}$ to local coordinate system of p

2: $\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$ — Individually lift each point into C_δ dimensional space

3: $\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$ — Concatenate $\mathbf{F}_\delta$ and $\mathbf{F}$, $\mathbf{F}_*$ is a $K \times (C_\delta + C_1)$ matrix

4: $\mathcal{X} \leftarrow MLP(\mathbf{P}')$ — Learn the $K \times K$ $\mathcal{X}$-transformation matrix

5: $\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F}_*$ — Weight and permute $\mathbf{F}_*$ with the learnt $\mathcal{X}$

6: $\mathbf{F}_p \leftarrow \mathrm{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$ — Finally, typical convolution between $\mathbf{K}$ and $\mathbf{F}_\mathcal{X}$

the feature channels are increased.

In Algorithm 1, p is the representative point. $\mathbf{K}$ is a trainable kernel of $\mathcal{X}$-Conv. $\mathbf{P}$ is the K nearest local neighborhood point set of the representative point. $\mathbf{F}$ is the feature matrix of $\mathbf{P}$. $\mathbf{F}_p$ is the feature projected from local neighborhood points to the representative point. In the $\mathcal{X}$-Conv operator, neighborhood points are first transformed into a local coordinate system centered at the representative point. Because $\mathcal{X}$-Conv is designed to extract features for a local region, the extracted features should depend on relative locations. MLP_δ is applied to lift the features into C_δ dimensional space, and then the learned $\mathcal{X}$-transformation matrix is used to weight and permute the features. A convolution is then applied on the features. Achieving invariance using a symmetric function may result in the loss of information, while achieving equivariance solves this problem. Ideally, because of the approximation ability of MLPs, permutation equivariance should be achieved by learning the $\mathcal{X}$-transformation matrix from input points. An example of semantic segmentation network using $\mathcal{X}$-Conv is demonstrated in Figure 2.12.

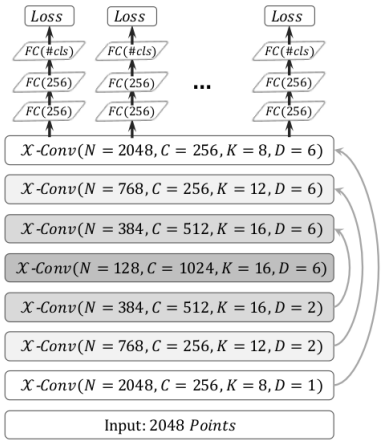


Figure 2.12: An example of semantic segmentation network with $\mathcal{X}$-Conv [23]

Similar to other point-based methods, the network takes a specific number of points as input. N and C stand for the number of output representative points and feature dimension. K is the number of local neighborhood points for each representative point. D is the dilation rate. When D is larger than 1, the K neighborhood points are uniformly sampled from $K \times D$ neighborhood points. Therefore, the receptive field of PointCNN can be defined as $(K \times D)/N$. It can be concluded that as $\mathcal{X}$-Conv is recursively applied, the resolution of each layer is decreasing, but the receptive field is increasing. The increasing receptive field determines the network has a larger view of the whole shape, which is beneficial for semantic segmentation.

Edge conditioned convolution

Following the idea that a point cloud can be regarded as graph-structured data, edge-conditioned convolution is proposed [42] and demonstrated to be effective on point cloud analysis. The

proposed operator is a generalization of convolution on graph-structured data. It is different from general convolution since the weights of ECC are dependent on edge features. This property makes the operation explicitly aware of relative relations encoded in edge features. A point graph of a given point i and its neighborhood points $j \in N(i)$ can be denoted as $G = (V, E)$, in which V are a set of points of the graph, and E are a set of edges of points. Every point comes with certain features $X^{l-1}(\cdot)$ as input to ECC, and $L(j,i)$ defines the edge relation of point $j \in N(i)$ and point i. The edge-specific weights Θ^l_{ji} are produced by Filter generating networks [15] $F^l(\cdot)$ based on edge features $L(j,i)$. The convolution output $X^l(i)$ of point i is then defined as below.

$$\begin{aligned} X^l(i) &= \frac{1}{|N(i)|} \sum_{j \in N(i)} F^l\left(L(j,i); w^l\right) X^{l-1}(j) + b^l \\ &= \frac{1}{|N(i)|} \sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) + b^l \end{aligned} \tag{2.8}$$

b^l and w^l are trainable parameters, and F^l is implemented using multi-layer perceptorns. By computing the weighted sum of $X^{l-1}(\cdot)$, ECC can achieve permutation invariance.

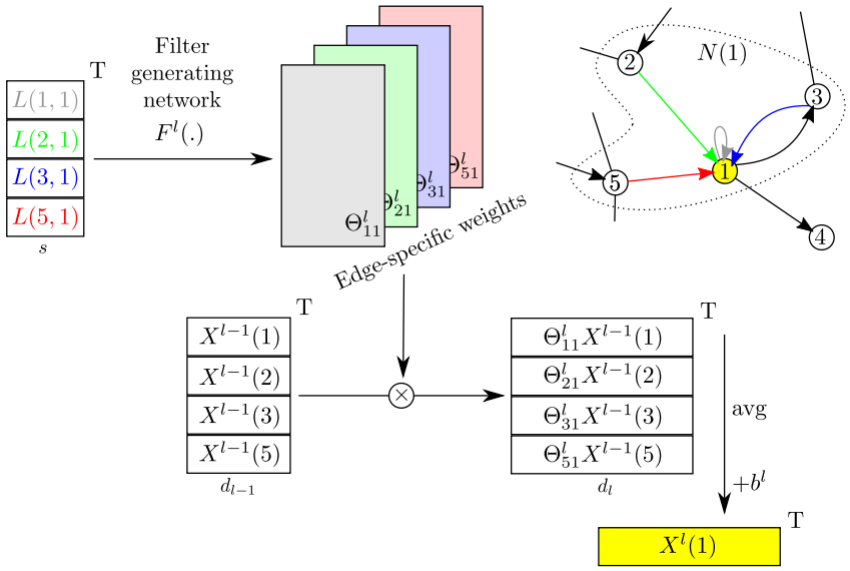


Figure 2.13: An illustration of ECC [42]

2.2.5. Graph-based method

Superpoint Graph

Point-based methods, such as PointNet and PointCNN, need to naively split a large-scale point cloud into small cubes before feeding points into the neural network. Landrieu et al. proposed a novel method [21] specifically designed for performing semantic segmentation at large-sclae point clouds.

Unlike other works that constrain the spatial linkages among points, the idea behind this work is that a geometrical representation called Superpoint graphs can effectively capture the internal connections between Superpoints. In order to obtain Superpoint graphs, Superpoints are first computed by an unsupervised method using hand-crafted features, including planarity, linearity, scattering, verticality, and elevation. The generated Superpoints are a set of points that represent a geometrically homogeneous shape. The homogeneous shape can be a large segment, for example, a wall, or it can be a small part of an object, for example, one component of a vehicle.

Then, the symmetric Voronoi adjacency graph of the whole point cloud is defined. And assume there are two Superpoints, S and T. If there is at least one edge in symmetric Voronoi adjacency graph that one end of the edge belongs to S and the other end belongs to T. S and T are considered adjacent. The edge of adjacent superpoints S and T is called Superedge. Then, the features of Superedges are obtained from the set of offsets for edges linking both Superpoints. Superedge features indicate the spatial relationship between Superpoints.

After we have Superedges and associated features, Superpoint embeddings are computed using a simplified PointNet. 128 points are selected on the fly from each Superpoint and fed into the PointNet. The authors believe a small number of points can represent the Superpoint because the Superpoint is considered as a semantically separable shape.

In the last phase, a graph neural network that contains an improved version of Edge-Conditioned Convolution [42] as well as Gated Recurrent Units [5] are used to assign each superpoint a class label based on its embeddings and edge features.

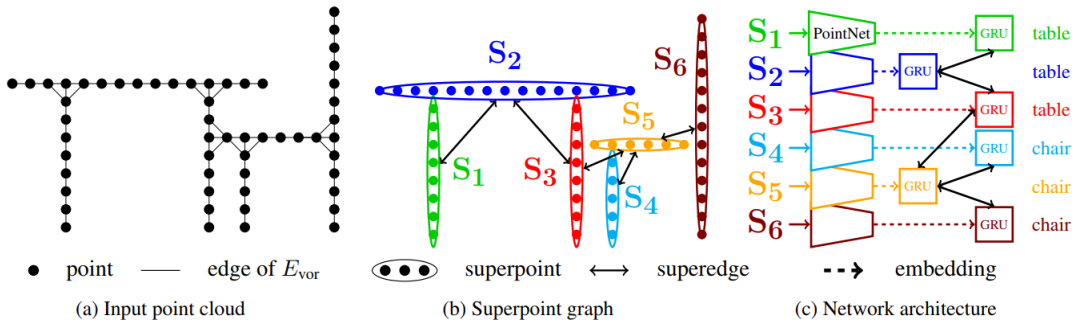


Figure 2.14: A pipeline of Superpoint Graph [21]

The design of Superpoint Graph significantly reduces the computation needed for a large-scale scene. The graph neural network can make use of contextual relations of superpoints. However, to a certain extent, the quality of the partitions determines the accuracy of final segmentation outcomes.

3

Methodology

In this chapter, the methodology details of my thesis project are introduced. The structure of my backbone network is introduced to demonstrate how a general point cloud semantic segmentation network processes the input data. Then, the rationale and architecture of the proposed geometric prior convolution are described.

3.1. Backbone network

$\mathcal{X}$-Conv [23] has been empirically verified to be an effective feature extractor by learning a $\mathcal{X}$-transformation matrix. Orientation-encoding unit [16] searches points from 8 different orientations to better capture the distribution of a point cloud. With the strong representation ability of $\mathcal{X}$-Conv, and the help of orientation-encoding unit, a hierarchical encoder-decoder semantic segmentation network can be constructed.

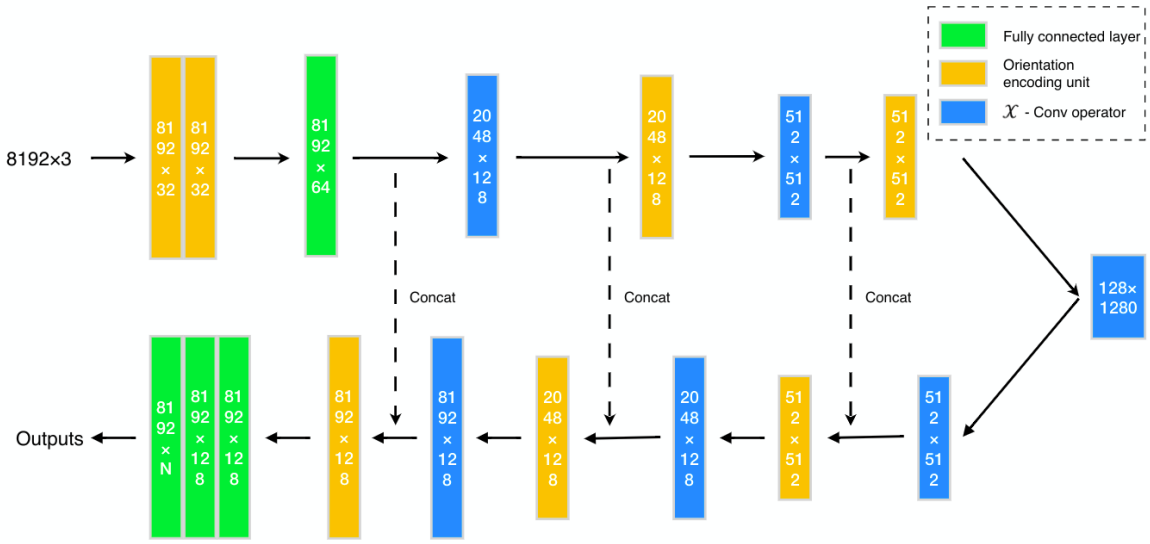


Figure 3.1: Backbone network with $\mathcal{X}$-Conv and orientation-encoding unit

The backbone network takes 8192 points with certain features as input, and outputs semantic labels. In each orientation-encoding unit, it treats every input point as a center point, and features from 8 orientations are projected on the center point. $\mathcal{X}$-Conv operators in encoder part down-sample the point cloud by finding a set of representative points using farthest point sampling. For a representative point, $\mathcal{X}$-Conv is applied on its k-Nearest Neighbor local point neighborhood. The features of the local point neighborhood are projected on the representative point. At the end of the encoder part, a point cloud is represented by 128 points with high

dimensional global features. In the decoder part, $\mathcal{X}$-Conv operators aim to propagate global features to original points following an idea of DeConv [31]. Representative points for $\mathcal{X}$-Conv are not searched by farthest point sampling, but simply adopted from corresponding representative points in the encoder part. Features, therefore, can be interpolated from a smaller number of points with higher dimensional features to a larger number of representative points with lower dimensional features. Skip connections are designed to supplement global features with detailed local features for classification.

The hierarchical structure enables the backbone network to enlarge the receptive field of each representative point. The computation of point features are under two structure representations. The first one is kNN local point neighborhood around a representative point in $\mathcal{X}$-Conv operator. The second is 8 points from different orientations in orientation-encoding unit. The two structure representations have slightly different focuses on the point cloud distribution. kNN local neighborhood can capture the local structure defined by k nearest points, but the structure representation can be constrained by local point distribution. For example, if a point is on a plane, the k nearest points of this point are likely to be on the same plane. 8 orientation search can solve this problem by finding points in 8 orientations, which captures a more general structure of the point cloud. In a way, the two operations can be considered supplementary to each other.

3.2. Geometric prior convolution

3.2.1. Rationale

Though the backbone network is well-designed, we see differences with human perception. When we look at a wall, our intuitive definition of a wall is an enormous vertical plane. When the backbone network processes the point cloud, the structure representations to extract the features are point sets, either generated by kNN or 8 orientation search. It has been empirically testified that the information embedded in x, y, z coordinates can be harnessed to yield a good point-wise classification result. However, two questions remain. The first is that if it's possible that a point cloud can be represented by meaningful shapes similar to what human perceives rather than independent points. The second is that if it's possible that the latent contextual relations of meaningful shapes can be harnessed to produce more accurate classification results. In order to address the two questions, we introduce geometric prior convolution that can weight the point features depending on geometric priors that encode the contextual relations between the underlying shapes of corresponding point pairs.

3.2.2. Geometric features

Superpoint Graph [21] shows a point cloud can be effectively represented by graphs of Superpoints. Superpoints are point cloud partitions generated by homogeneous partition process. In this subsection, we show how homogeneous partitions can be utilized to generate geometric features that represent the underlying shapes of points.

Homogeneous partition

The first step to generate geometric features is to segment the point cloud into semantically and geometrically homogeneous partitions. Ideally, the points reside in a partition will have same labels and similar geometric features. The shapes and sizes of partitions are not fixed. They vary with the shape complexity of the point cloud. The partitions are not complete objects in most of the cases. Instead, they are simpler shapes on objects. If one object is geometrically complex, for example a chair, the object is more likely to be divided into several small partitions. One partition contains a simple shape of the object, such as, the back or a leg of the chair. When the object is geometrically simple, for example a wall, it is likely to be segmented into one large partition. We follow the unsupervised method introduced in [21] to do homogeneous partition.

Given a point cloud with $\mathcal{N}$ points, eigenvalues are first computed for each point, following [7]. For a point $\mathbf{x}_i = \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T$ and its local point neighborhood $\mathcal{V}$ with n points,

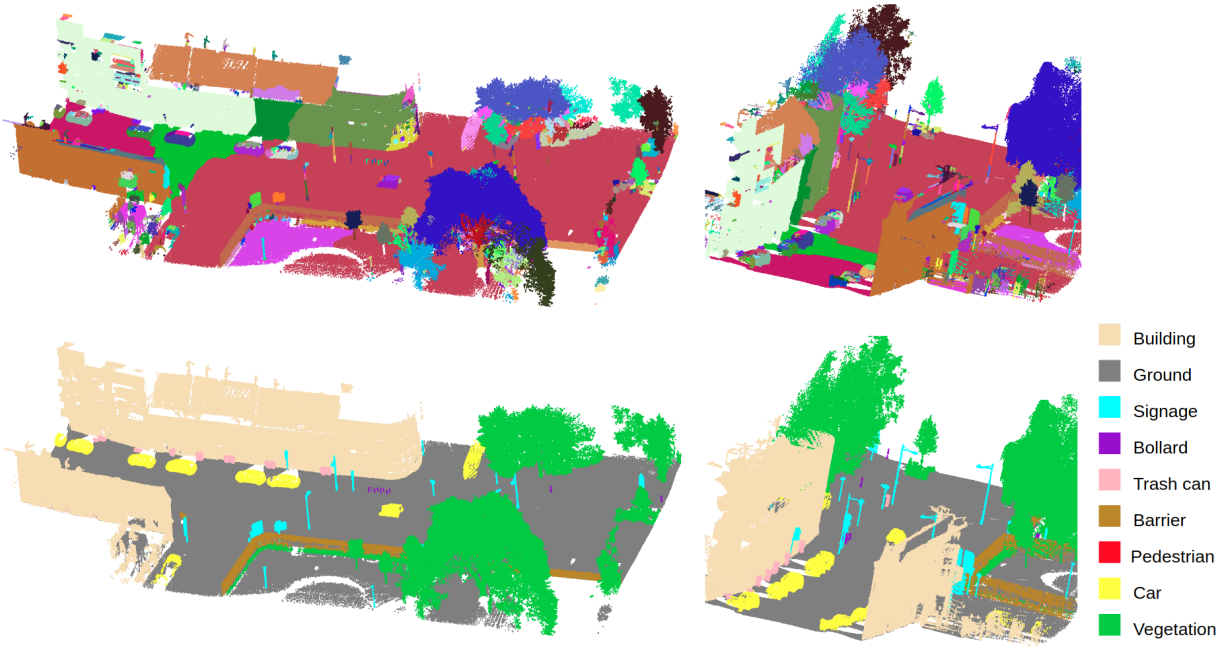


Figure 3.2: An example of homogeneous partitions on Paris and Lille 3D benchmark [40]. The upper row is the visualization of homogeneous partitions. Each color demonstrates a single partition. The lower row is the ground truth. Each color represents a class.

we have $\overline{\mathrm{x}} = \frac{1}{n}\sum_{i=1}^{i=n} \mathrm{x}_i$. The 3D structure tensor can be defined as $\mathbf{C} = \frac{1}{n}\mathbf{M}^T\mathbf{M}$, with $\mathbf{M} = \left(\mathrm{x}_1 - \overline{\mathrm{x}}, \quad \ldots \quad , \mathrm{x}_n - \overline{\mathrm{x}} \right)^T$. Here, $\mathbf{C}$ is a symmetric positive definite matrix, and eigenvalue decomposition can be applied. The eigenvalues can be arranged in a descending order as $\lambda_1 \geq \lambda_2 \geq \lambda_3 > 0$. Then, eigenvalue features can be defined following [11].

$$Linearity : \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{3.1}$$

$$Planarity : \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{3.2}$$

$$Scattering : \frac{\lambda_3}{\lambda_1} \tag{3.3}$$

Let u_1, u_2, u_3 be the corresponding eigenvectors of $\lambda_1, \lambda_2, \lambda_3$. The vertical component of the vector below is the verticality of a point.

$$[\hat{u}]_i \propto \sum_{j=1}^{3} \lambda_j \left| [u_j]_i \right|, \text{ for } i = 1,2,3 \text{ and } \|\hat{u}\| = 1 \tag{3.4}$$

The eigenvalue and eigenvector features reveal different structure properties. The linearity and planarity describes how well the point local structure fits a line or a plane. Scattering evaluates if the point local structure is an isotropic spherical neighborhood. Verticality can discriminate the vertical property of the point cloud. In addition to these four features, elevation is defined as the z coordinate of each point x_i normalized over the entire point cloud. Therefore, for each point x_i, we have a 5 dimension feature vector $f_i \in \mathbb{R}^5$ to describe the local structure. The points in a point cloud can be connected by an undirected graph, $G = (V, E)$. G here is the point graph. V contains nodes of the graph, which are the points in the point cloud. E represents the edges of the graph, which indicate the adjacency between point pairs. The homogeneous partition [7] is based on 10 nearest graph G of the point cloud. An optimization function for

homogeneous partition is defined as below.

$$g = \underset{g \in \mathbb{R}^{V \times 5}}{\arg\min} \sum_{i \in V} \|g_i - f_i\|^2 + \mu \sum_{(i,j) \in E} \omega_{i,j} \left[g_i - g_j \neq 0 \right] \tag{3.5}$$

g is the piece-wise constant approximation of $f \in \mathbb{R}^{V \times 5}$ that minimizes the optimization function. $[\cdot]$ is the Iverson bracket that returns 1 if the condition is satisfied, otherwise returns 0. The weight $\omega_{i,j}$ is inversely proportional to the distance between the point pairs. The first term of the optimization function ensures the resulting g is close to f. The second term adds more penalty to edges with long distances, which guarantees the simple shape of the partitions. μ is the regularization strength that makes a balance between two terms. ℓ_0-cut pursuit algorithm [20] can be used to find the approximate solution of the optimization function. The homogeneous partitions $\mathcal{P} = \{P_1, \cdots, P_k\}$ are defined as constant connected components of the piece-wise constant approximations.

Feature generation

After homogeneous partition, a point cloud is segmented into k semantically and geometrically homogeneous partitions. Geometric features are designed to describe the underlying shapes of points defined by the homogeneous partitions. The features can be adjusted flexibly depending on the data properties.

Feature	Dimension	Description
Centroid of the partition	3	$\text{mean}_{p_i \in P_k}\, p_i, p_i \in \mathbb{R}^3$
Length of the partition	1	$\lambda_1^{P_k}$
Surface of the partition	1	$\lambda_1^{P_k} \lambda_2^{P_k}$
Volume of the partition	1	$\lambda_1^{P_k} \lambda_2^{P_k} \lambda_3^{P_k}$
Point count of the partition	1	Point number $\lvert P_k \rvert$ of the partition
Linearity of the point	1	Linearity computed for homogeneous partition
Planarity of the point	1	Planarity computed for homogeneous partition
Scattering of the point	1	Scattering computed for homogeneous partition

Table 3.1: 10 dimensional geometric features to describe the underlying shape of each point

For each point $p_i \in P_k$, a geometric feature vector $f_g^{p_i} \in \mathbb{R}^{10}$ is constructed. The first three dimensions are the center coordinates of the partition P_k. Following [21], eigenvalues $\lambda_1^{P_k}, \lambda_2^{P_k}, \lambda_3^{P_k}$, are computed in a descending order for all the points in the partition. Three features, length, surface and volume, representing the general shape of the partition are derived based on the eigenvalues. Point count $|P_k|$ defines the size of the partition, which in a way assumes the point cloud is evenly distributed. The last three features are reused from the homogeneous partition step, and they can supplement the general shape features with more local considerations.

3.2.3. Learning from contextual relations

In human perception, we consider contextual relations among objects, such as chairs are usually close to desks, and a chair commonly has four legs attached. Homogeneous partition divides a point cloud into geometrically simple but meaningful shapes. Based on this, geometric features are designed to represent the underlying shapes of points, which makes it possible to make use of similar contextual relations explicitly. In order to incorporate these contextual relations into point-wise classification, we propose geometric prior convolution, which is a generalization of a convolution operator on points.

The logic behind geometric prior convolution is simple and similar to [41]. A center point with its neighborhood points can be structured as a directed graph where the neighborhood points are pointing towards the center point. In the directed graph, the nodes are points with features. The edges indicate the relationship between node pairs. Given this, geometric priors are constructed as edge features to demonstrate the contextual relations between node pairs.

A convolution-like operation can be applied to integrate the features of the neighborhood graph to the center point.

Graph construction

For a center point, not all the nearby points can bring contextual relations. The quality of point searching to construct the graph is crucial for the effectiveness of the proposed convolution. In order to find the points with meaningful contextual relations, an ideal candidate of point searching should have certain properties. First, the point searching should not be constrained by local structures, for example a plane. It should have a certain pattern that captures the 3D space, so that similar contextual relations can always be found in the point graph. 8 orientation search qualifies by searching nearest points in 8 octants around the center point. This functionality makes it overcome the constraints of a local structure, and it is more likely to capture points with different underlying shapes to provide contextual relations for the center point. Benefited from its orientation aware property, possible underlying shapes in different orientations are equally considered.

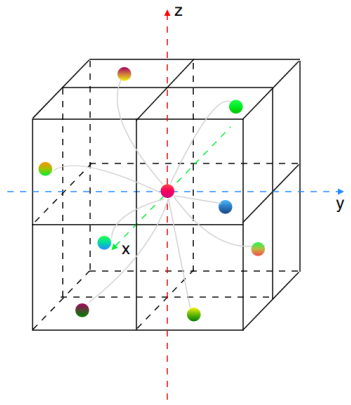


Figure 3.3: An illustration of searching in 8 different octants. The red dot is the center point, and other dots represent nearest points in 8 different orientations.

Geometric priors

Geometric priors are based on geometric features, which encodes the relationship of shapes underlying the points. We use subtraction and division to make the geometric priors aware of the shape relations. Given a center point x_i with geometric features $f_g^{x_i}$, we can find a neighborhood point x_j. x_j comes with geometric features $f_g^{x_j}$, and geometric priors $f_p(x_j, x_i)$ are denoted in Table 3.2.

Geometric priors	Dimension	Description
Centroid offset	3	Centroid of x_j - Centroid of x_i
Length ratio	1	log(Length of x_j / Length of x_i)
Surface ratio	1	log(Surface of x_j / Surface of x_i)
Volume ratio	1	log(Volume of x_j / Volume of x_i)
Point count ratio	1	log(Point count of x_j / Point count of x_i)
Linearity ratio	1	log(Linearity of x_j / Linearity of x_i)
Planarity ratio	1	log(Planarity of x_j / Planarity of x_i)
Scattering ratio	1	log(Scattering of x_j / Scattering of x_i)

Table 3.2: Geometric priors $f_p(x_j, x_i)$

Learning by permutation invariant convolution

Learning a discriminative representation of the point cloud is the key to the semantic segmentation. The rationale behind geometric prior convolution is to provide each point with

contextual relations of shapes underlying the points to enhance the point cloud structural representation. To this end, a more efficient version of ECC [42] is considered.

Given a point cloud $\mathcal{C}$, we have a point $x_i \in \mathcal{C}$. $\mathcal{N}_{x_i}^8$ is the 8 orientation neighborhood found by 8 orientation search centered at x_i, and it exists $x_j \in \mathcal{N}_{x_i}^8$. A general convolution can be defined as below.

$$\mathbf{F}_{x_i} = \sigma(\mathcal{A}(\omega \cdot \mathbf{F}_{x_j}, \forall x_j)), \forall x_j \in \mathcal{N}_{x_i}^8 \tag{3.6}$$

The representation $\mathbf{F}_{x_i}$ of $\mathcal{N}_{x_i}^8$ is obtained by performing element-wise multiplication, denoted by $\cdot$, between weight vector ω and point feature vector $\mathbf{F}_{x_j}, \forall x_j \in \mathcal{N}_{x_i}^8$. Then aggregation function $\mathcal{A}$ is applied following with a non-linear activation function σ. In a classic convolution, aggregation function is summation, Σ, and weights are order-dependent ω_j, so that we have:

$$\mathbf{F}_{x_i} = \sigma\left(\Sigma_{j=1}^{j=8}\left(\omega_j \cdot \mathbf{F}_{x_j}\right)\right) \tag{3.7}$$

For non-grid data, point clouds, the weights ω_j will introduce permutation variance. Therefore, we use ω_{ji} instead of ω_j, and we here explain how weights ω_{ji} based on geometric priors $f_p(x_j, x_i)$ can achieve permutation invariance. Weights ω_{ji} are generated by a Filter generating network [15], which can also be considered as a mapping $\mathcal{M}$ [26] from geometric priors to weights. The mapping can be defined as below.

$$\mathcal{M} : f_p(x_j, x_i) \rightarrow \omega_{ji} \tag{3.8}$$

Along with the mapping $\mathcal{M}$, we use average function as aggregation function. The geometric prior convolution then can be defined.

$$\begin{aligned} \mathbf{F}_{x_i} &= \sigma\left(\frac{1}{8}\Sigma_{j=1}^{j=8}\left(\mathcal{M}(f_p(x_j, x_i)) \cdot \mathbf{F}_{x_j}\right)\right) \\ &= \sigma\left(\frac{1}{8}\Sigma_{j=1}^{j=8}\left(\omega_{ji} \cdot \mathbf{F}_{x_j}\right)\right) \end{aligned} \tag{3.9}$$

In this formulation of convolution, weights ω_{ji} are conditioned on geometric priors $f_p(x_j, x_i)$ that encode the underlying shape relations between x_j and x_i. Because of the powerful approximation ability of multi-layer perceptrons (MLPs), we use shared MLPs to dynamically map $f_p(x_j, x_i)$ into ω_{ji} with the same channel number of $\mathbf{F}_{x_j}$. In this way, weights ω_{ji} are order specific, which realizes permutation awareness. By performing element-wise multiplication $\omega_{ji} \cdot \mathbf{F}_{x_j}$, the contextual relations are explicitly connected with point features. By using average function to aggregate features, the generated feature $\mathbf{F}_{x_i}$ considers both point features and contextual relations of shapes underlying the points. Because of the permutation awareness, weights ω_{ji} will always be the same for a certain point pair x_j and x_i, so that after symmetric aggregation function, the output is irrelevant with the input order.

3.3. GP-net

Since the structure of geometric prior convolution is differentiable and it can be applied on each of the input points, we insert three layers of it in the encoder part of the backbone network to provide each point with contextual relations of shapes underlying points. This is the structure of our GP-net, as shown in Figure 3.5.

3.3.1. Implementation details

In geometric prior convolution, the mapping $\mathcal{M}$ is implemented with three layers of MLPs with size $(32, 64, C)$, in which C is the dimension of point features. Every layer of MLPs has elu as activation function and batch normalization. After each of the geometric prior convolution, we also apply elu and batch normalization on the output. The implementation of other structures follows the original papers. For training the network, we use ADAM [17] optimizer with initial learning rate of 0.001. The loss function is combined Lovász-Softmax loss [2], and the details

will be discussed in the next chapter.

$$Loss = -\frac{1}{p}\sum_{i=1}^{p}\log f_i\left(y_i^*\right) + l\frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}}\overline{\Delta_{Jc}}(m(c)) + \frac{\lambda}{2n}\sum_{w} w^2 \tag{3.10}$$

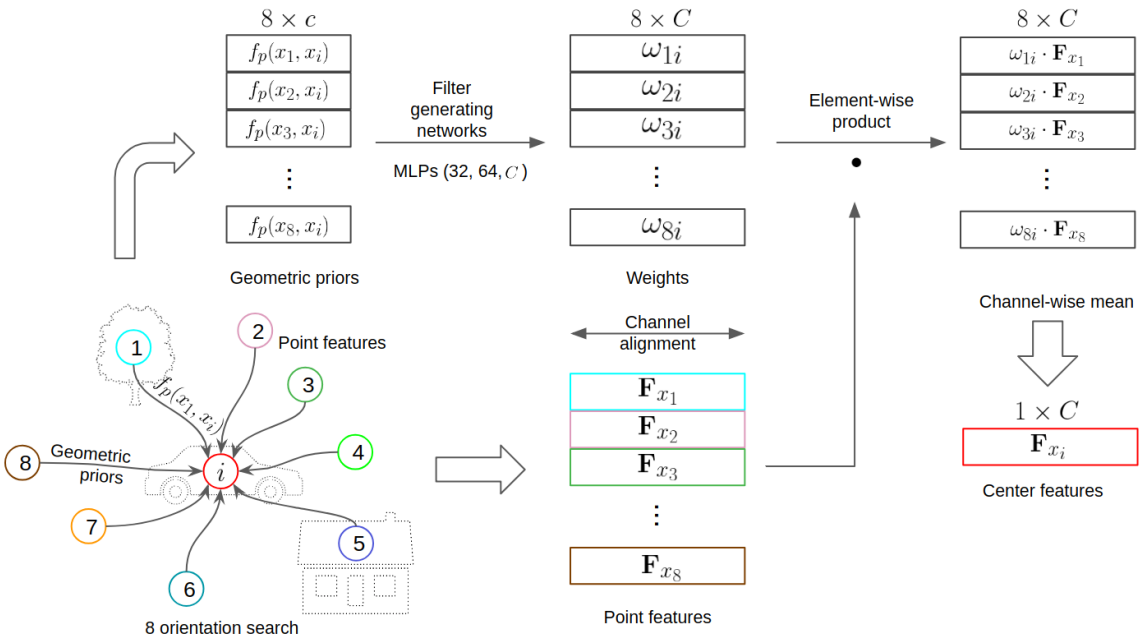


Figure 3.4: An illustration of geometric prior convolution. c is the dimension of geometric priors $f_p(x_j, x_i)$. C is the dimension of point features F_{x_j}, F_{x_i}, and weights ω_{ji}.

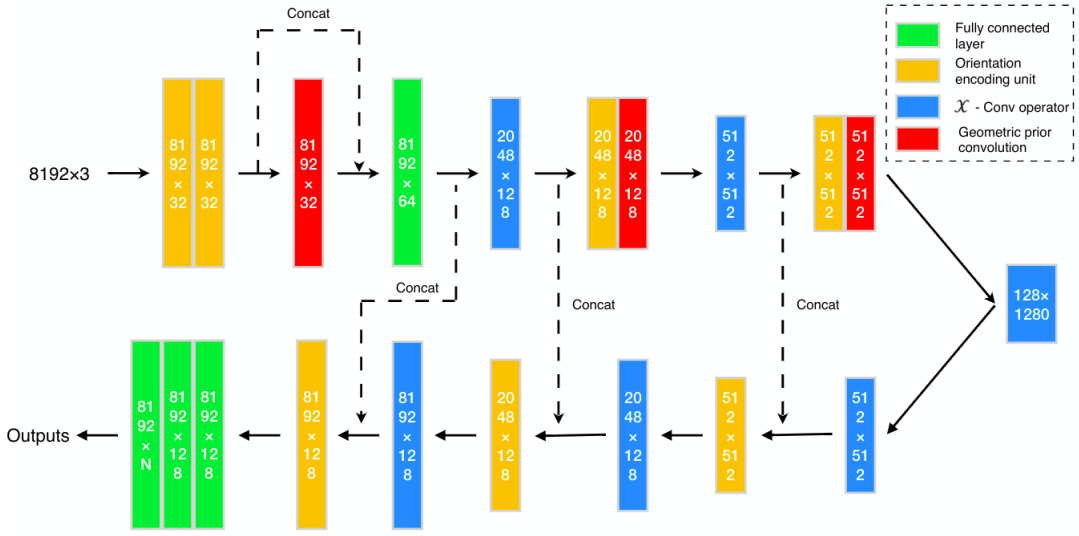


Figure 3.5: The structure of GP-net

4

Results and Discussion

In this chapter, we introduce dataset, metrics and experimental setups. We show the geometric prior convolution can improve the performance of our backbone network by a large margin. We also demonstrate alternatives of components in geometric prior convolution and analyze how they influence the performance.

4.1. Dataset and experimental setup

Paris and Lille dataset [40] is a large-scale point cloud dataset acquired using LiDAR equipped on the top of a moving truck. It covers 1940 meters of the urban road scene in Paris and Lille with over 140 million points. The points in the dataset are annotated with 9 classes, ground, pedestrian, building, vegetation, signage, bollard, trash can, barrier and car. The dataset has a relatively even density, which is around 1000 and 2000 points per m^2. However, due to the properties of the multi-beam LiDAR sensor, there are anisotropic patterns in the data.

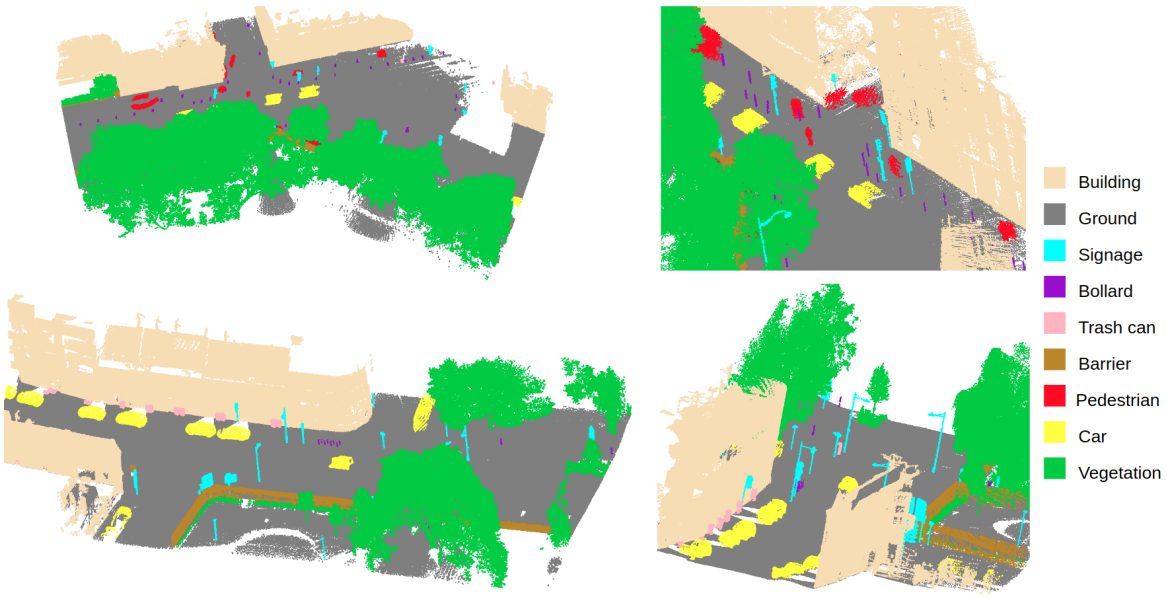


Figure 4.1: Examples of Paris and Lille dataset, every class is assigned an unique color.

As shown in the Table 4.1, Paris and Lille dataset consists of 3 large point clouds that are collected in two different cities. For experiments, the dataset is first divided into training set and validation set. We follow the $80:20$ ratio, and divide Lille1 as well as Paris into training set. Lille2 is the validation set. The training set is then split into blocks of $20m \times 5m$ along

Section	Length	Number of points	Number of objects
Lille1	1150 m	71.3 M	1349
Lille2	340 m	26.8 M	501
Paris	450 m	45.7 M	629
Total	1940 m	143.1 M	2479

Table 4.1: Overview of the Paris and Lille dataset

the road center line with a stride of $2m$. Since in our methods, we consider contextual relations of shapes underlying points, the completeness of objects is possible to influence the performance. Therefore, for the training set, we augment the data with a stride of $2m$, so that the information of objects can be thoroughly utilized. We naively split the validation set into $20m \times 5m$ blocks without a stride for an efficient and unbiased evaluation. Then, the homogeneous partition can be applied on the blocks. After homogeneous partition, we randomly subsample each block into 8192 points with geometric features for processing.

Paris and Lille 3D benchmark also has 30 million unlabeled points as the test set. The test set comprises 3 point clouds, Ajaccio2, Ajaccio 57, and Dijon 9. Each of them has 10 million points. In order to make use of the contextual relations among points completely to have a better performance, we split the test set into $20m \times 5m$ blocks along the road center line with stride of $5m$ in x direction, and $3m$ in y direction for further processing.

4.2. Results

Metrics

Mean Intersection over Union (mIoU) is the metric to evaluate our models. This is because it treats different classes evenly, and usually the number of points belonging to different classes is unbalanced. One extreme example is that we have 10000 points in class A, 100 points in class B and 100 points in class C. If one model classify all points as class A, the overall accuracy (OA) of this model is 98%, but mIoU is 32.7%. Clearly, this is a poor classifier. OA evaluates the model with bias, and mIoU is more reasonable. The equations of IoU and mIoU are shown in equation 4.1 and 4.2. N here is the number of classes.

$$IoU = \frac{TP}{TP + FP + FN} \tag{4.1}$$

$$mIoU = \frac{1}{N}\Sigma_{i=1}^{i=N} IoU_i \tag{4.2}$$

Results

Methods	mIoU (%)	Ground	Building	Sinage	Bollard	Trash can	Barrier	Pedestrian	Car	Vegetation
KP-FCNN [44]	75.9	99.5	93.2	69.3	82.2	48.8	44.3	62.0	93.6	90.4
HDGCN [24]	68.3	99.4	93.0	67.7	75.7	25.7	44.7	37.1	81.9	89.6
RF_MSSF [43]	56.3	99.3	88.6	47.8	67.3	2.3	27.1	20.6	74.8	78.8
GP-net (ours)	74.7	99.3	96.8	77.4	71.7	56.8	45.9	38.8	94.5	91.3

Table 4.2: Test result comparison of Paris and Lille 3D benchmark. mIoU and IoU of all classes are presented.

The test results in Table 4.2 show that our method achieves state-of-the-art performance. In several classes, our method outperforms the method with the highest mIoU. The visualization of test results are shown in Figure 4.2-4.4.

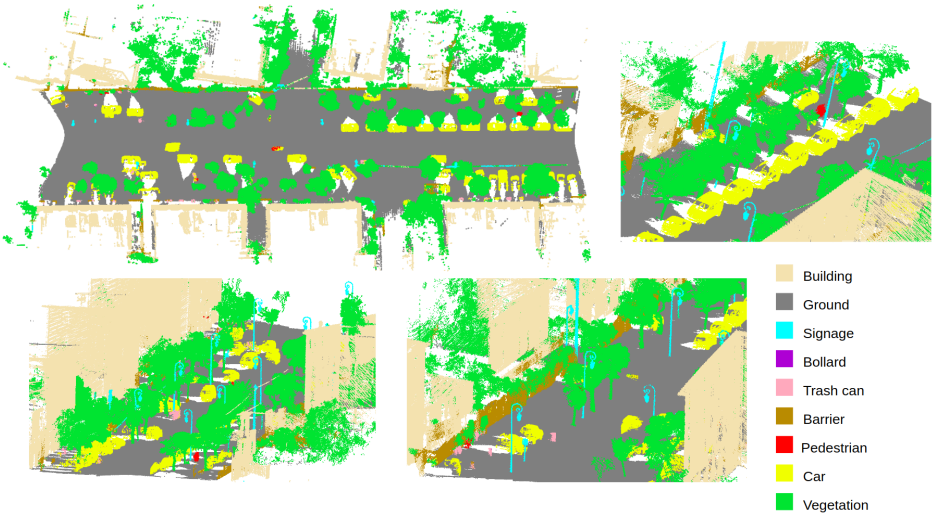


Figure 4.2: Test result of Paris and Lille 3D benchmark, Ajaccio 2 with 10 million points

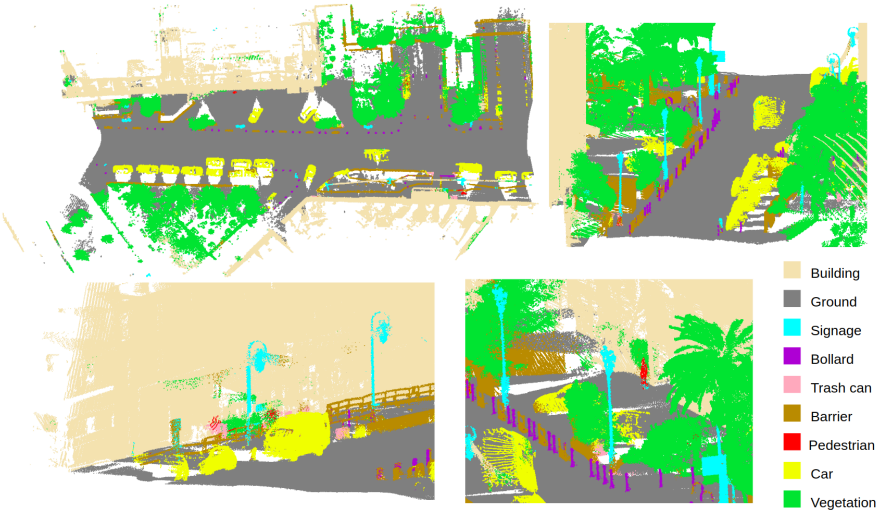


Figure 4.3: Test result of Paris and Lille 3D benchmark, Ajaccio 57 with 10 million points

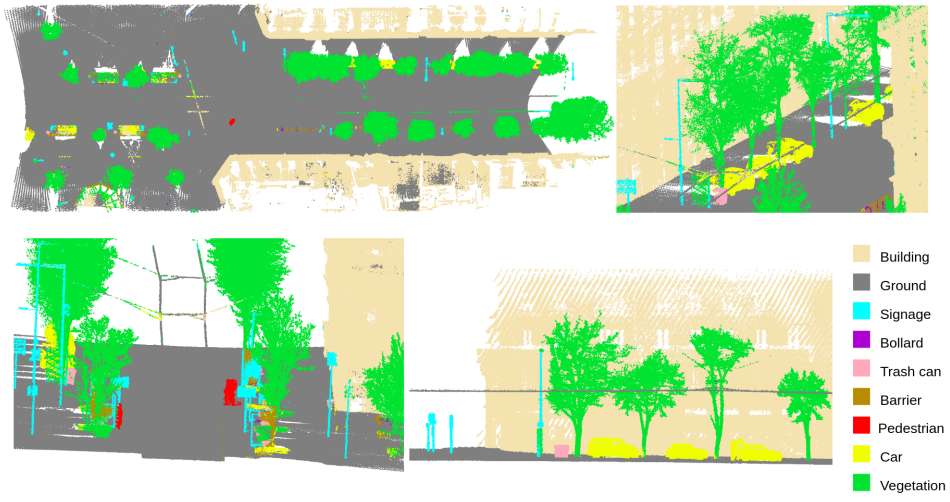


Figure 4.4: Test result of Paris and Lille 3D benchmark, Dijon 9 with 10 million points

4.3. Discussion

We analyze different design options of geometric prior convolution and GP-net by doing ablation studies. The experiment results of ablation studies are based on training set and validation set described in the previous section.

4.3.1. Structure of GP-net

Model	mIoU (%)
Backbone	68.6
Add the first layer of GP-conv	73.2
Add first two layers of GP-conv	73.9
Add three layers of GP-conv	75.5

Table 4.3: Performance of models with different layers of geometric prior convolution

	mIoU (%)	Ground	Building	Signage	Bollard	Trash can	Barrier	Pedestrian	Car	Vegetation
Backbone	68.6	98.2	95.4	60.3	75.3	55.1	16.7	53.4	96.1	66.7
GP-net	75.5	98.5	97.3	71.5	75.9	68.3	31.1	71.4	94.8	69.9

Table 4.4: The detailed IoU comparison between backbone network and GP-net

Since geometric prior convolution can be inserted into point-based networks, we exam each layer's contribution of geometric prior convolution to GP-net. Experiments show the first layer of geometric prior convolution plays a most important role in providing useful contextual information. This is probably because the first layer is applied on the point cloud before down-sampling, so that more effective points can be found in the relatively dense point cloud.

4.3.2. Geometric prior convolution

Point searching method

kNN and ball query are two most prevalent point neighborhood searching methods. We compare them with 8 orientation search to show 8 orientation search is more suitable for geometric prior convolution.

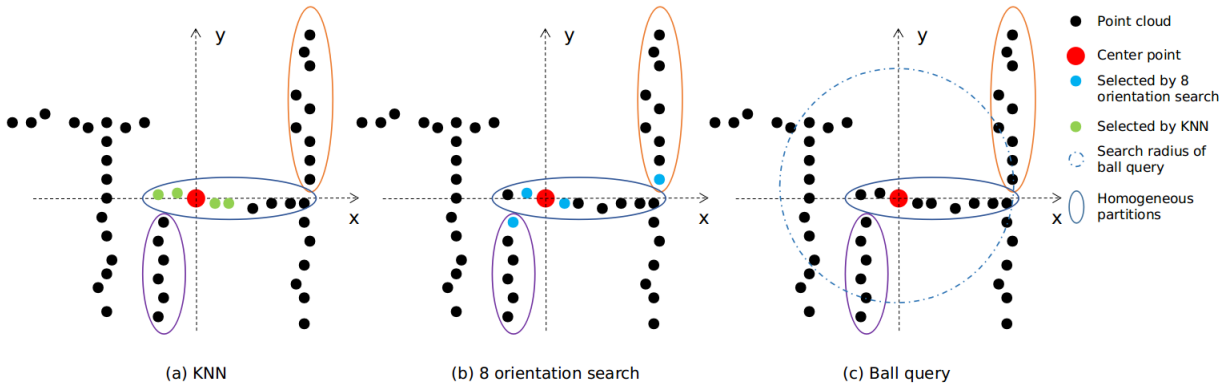


Figure 4.5: An illustration of different point neighborhood searching methods

Ideally, for a specific point, the points that can provide contextual relations should be in close but different homogeneous partitions. Only when two points reside in two different homogeneous partitions, can the geometric priors provide contextual relations. Thus, the point searching method can not be constrained by local structures, and it should perceive the distribution of the point cloud with a certain pattern. kNN is easily constrained by a small local structure. It is unable to capture a more general local structure. Ball query randomly selects

points in a search radius. The randomness makes it less reliable in providing consistent contextual relations under similar circumstances. 8 orientation search finds nearest points in 8 octants, which makes it more likely to find points in different homogeneous partitions and better capture the distribution of the point cloud.

Point searching method	mIoU (%)
8 orientation search	75.5
kNN	68.6
Ball query	72.3

Table 4.5: Performance of models with different point searching methods. Ball query and 8 orientation search have same searching radius. All of the three methods search 8 points for computation.

Our experiment results prove our analysis. kNN barely provides useful contextual information. The model with kNN has the same performance as the backbone network. Ball query helps the network to incorporate informative relations, but the randomness limits its performance.

Self-loops

Among other convolution operators, on pixels or points [42, 46], the information of the center element is usually carefully preserved. One important difference of geometric prior convolution is that the information of the center point is not involved in computation. We have three reasons for this design. First, the hierarchical structure of GP-net is to make sure at the end of the encoder part, each point has a large receptive field. Even though geometric prior convolution does not include the feature of the center point into computation, it can still effectively integrate features of 8 points together to enlarge the receptive field. Second, the purpose of geometric prior convolution is to incorporate contextual relations, when it comes to self-loops, similar to kNN, geometric priors of same homogeneous partitions are not able to provide effective information, so that it may even degrade the performance, as shown in Table 4.6. Third, the features of center points are not actually discarded. Because of the existence of skip-connections, the local features of each point are concatenated with high-level global features to provide detailed features in the decoder part. Since the point-wise classification is made at the end of the neural network, an effective integration of local point features and global point features is crucial for the classification.

Model	mIoU (%)
With self-loops	72.7
Without self-loops	75.5

Table 4.6: Performance of models with or without self-loops

Aggregation function

As discussed in [26], there are more symmetric functions that can be used as aggregation functions for geometric prior convolution. We exam the performance of GP-net with max pooling, average function, and summation. In our experiments, different aggregation functions do

Model	mIoU (%)
Average function	75.5
Max pooling	75.4
Summation	74.8

Table 4.7: Performance of models with different aggregation function

not differentiate from each other. Average function performs slightly better than others.

Geometric feature

Geometric features and relative geometric priors can be flexibly designed for different datasets. We here propose different combinations of geometric priors and show the experiment results. We tested two different sets of geometric priors on Paris and Lille dataset. One set is described

Geometric priors	mIoU (%)
With centroid offset	75.5
Without centroid offset	72.9

Table 4.8: Geometric priors with or without centroid offset

in the previous chapter, and the other set is more rotation invariant. In order to be more rotation invariant, we replace centroid offset with distance offset and z offset. Distance offset is the Euclidean distance of centers of two corresponding homogeneous partitions. For urban point cloud, rotation rarely happens in terms of z axis, so that the offset of z coordinate of two corresponding homogeneous partitions is also rotation invariant. Experiments show the geometric priors with centroid offset perform better. We therefore keep centroid offset for Paris and Lille dataset.

Besides the rotation invariant geometric priors, we try to replace some of the geometric priors with 1 at each time to exam their contribution to the general performance. The experiments show they all contribute to the performance.

Geometric priors	mIoU (%)
All priors kept	75.5
Centroid offsets replaced by 1	72.2
Length, surface, volume ratio replaced by 1	72.4
Point count ratio replaced by 1	72.6
Linearity, planarity, scattering ratio replaced by 1	72.3

Table 4.9: Performance under different geometric prior combinations

We described in the previous chapter the functions used to compute geometric priors from geometric features are responsible to find the relative relations of features. We remove the functions and only use geometric features to demonstrate the underlying shapes and see how the performance goes. The experiment results show without relative computation, the relations of underlying shapes of points can not be demonstrated, which barely contributes to the improvement of performance.

Model	mIoU (%)
With geometric priors	75.5
With geometric features	69.3

Table 4.10: Performance of models with geometric priors or geometric features

4.3.3. Combined Lovász-Softmax loss

Cross-entropy loss treats each point equally by summing the negative log-likelihood of all points together and divide it by the number of points. When a point has a high probability to be correctly classified, the loss caused by this point is nominal, otherwise is large. In a way, cross-entropy directly optimizes the overall accuracy, which has been proved to be a sub-optimal metric for semantic segmentation task. To evaluate a semantic segmentation neural network, we usually use mean Intersection over Union (mIoU) as the main metric. Given this, minimizing cross-entropy along may not be the perfect choice to optimize a semantic segmentation neural network.

Lovász-Softmax loss

In order to compensate the shortcoming of cross-entropy, we want to optimize the neural network directly on Jaccard index (intersection over union). Jaccard index of class c, J_c, can be defined.

$$J_c(y^*, \tilde{y}) = \frac{\left|\{y^* = c\} \cap \{\tilde{y} = c\}\right|}{\left|\{y^* = c\} \cup \{\tilde{y} = c\}\right|} = \frac{TP_c}{TP_c + FN_c + FP_c} \tag{4.3}$$

y^* is a vector of the ground truth, $\tilde{y}$ a vector of predictions, TP_c true positives of c, FN_c false negatives of c, FP_c false positives of c. A loss function in terms of Jaccard index can be defined as below.

$$\Delta_{J_c}(y^*, \tilde{y}) = 1 - J_c(y^*, \tilde{y}) \tag{4.4}$$

From equation (4.3), Jaccard index is count based [35], and we need TP, FN, FP to compute it. The Jaccard index loss is not differentiable, so that it can not be optimized directly by back propagation. Therefore, an alternative of Jaccard index loss needs to be found. The set of mispredictions of class c, $\mathbf{M}_c \in \{0,1\}^p$, can be defined as below.

$$\mathbf{M}_c(y^*, \tilde{y}) = \{y^* = c, \tilde{y} \neq c\} \cup \{y^* \neq c, \tilde{y} = c\} \tag{4.5}$$

$\mathbf{M}_c$ is a set with p elements of either 0 or 1. The Jaccard index loss can be defined with respect to $\mathbf{M}_c$ as below.

$$\Delta_{J_c} = \frac{|\mathbf{M}_c|}{|\{y^* = c\} \cup \mathbf{M}_c|} \tag{4.6}$$

Lovász-Softmax loss [2] is a surrogate to optimize a neural network directly on Jaccard index. The Lovász-Softmax loss in multi class setting is defined as below.

$$\mathfrak{L} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \overline{\Delta_{J_c}}(m(c)) \tag{4.7}$$

$m(c) \in [0,1]^p$ is an error vector that records the probability for each point $i \in [1,p]$ that is misclassified in terms of class $c \in \mathcal{C}$.

$$m_i(c) = \begin{cases} 1 - f_i(c) & \text{if } c = y_i^* \\ f_i(c) & \text{otherwise} \end{cases} \tag{4.8}$$

$\overline{\Delta_{J_c}}$ is the Lovász extension of Jaccard index loss Δ_{J_c}. Equation (4.6) has been proved to be submodular [52]. The Lovász extension of a submodular function is convex and continuous and it can be minimized efficiently. The minimization of Lovász extension of a submodular function is an ideal surrogate for minimizing a submodular set function. In addition, the operations in Lovász-Softmax loss are differentiable and can be implemented on GPU with high efficiency [2]. Lovász-Softmax loss considers all classes by computing the mean of $\overline{\Delta_{J_c}}$, $c \in \mathcal{C}$, which is similar to the operation in mIoU.

As mentioned in [2], optimizing batch mIoU is not completely equivalent to optimizing dataset mIoU, because of the absence of some of the classes under the mini-batch setting. We here adopt an idea to combine cross-entropy loss and Lovász-Softmax loss for a more accurate and stable training.

$$Loss = -\frac{1}{p} \sum_{i=1}^{p} \log f_i\left(y_i^*\right) + l \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \overline{\Delta_{J_c}}(m(c)) + \frac{\lambda}{2n} \sum_{w} w^2 \tag{4.9}$$

The final loss function is defined in equation (4.9). The first term is cross-entropy loss, which ensures the successful and stable convergence of the training process. The second term is Lovász-Softmax loss with empirical intensity parameter l. This term provides extra penalty in order to optimize the parameters in the direction of a higher mIoU. The intensity parameter can be adjusted to keep a balance between cross-entropy loss and Lovász-Softmax loss. In experiments, l is set to 2 for Paris and Lille 3D benchmark. The third term is L2 regularization

term. This term can prevent overfitting by minimizing the weights. λ is weight decay parameter. w represents all the weights in the neural network. n is the number of training samples. The experiments show that the combined loss can improve the performance by 6.6 mIoU(%).

Loss	mIoU(%)
Cross-entropy loss	68.9
Combined loss	75.5

Table 4.11: Performance of models with different loss functions

5

Conclusion and Future Work

In this work, we think outside the box of conventional point cloud deep learning. Our inspiration origins from human perception. We try to incorporate contextual relations of underlying shapes among points into computation. For this purpose, we design geometric features representing underlying shapes of points. Geometric priors are further generated to define contextual relations of point pairs. Permutation invariant geometric prior convolution with 8 orientation search is responsible for integrating these contextual relations on points. A backbone network is designed to accommodate geometric prior convolution layers to achieve point-wise classification. We call the final network GP-net. Our ablation studies show the architecture of GP-net has been optimized. The test results on Paris and Lille 3D benchmark show GP-net can achieve state-of-the-art performance. We also show Lovász-Softmax loss can be generalized to improve the performance of point cloud semantic segmentation neural networks.

Though GP-net has a good performance on Paris and Lille 3D benchmark, we believe the point searching method for providing the contextual relations can be further improved. 8 orientation search outperforms other point searching methods in our experiments, but a drawback is conspicuous. The points searched by 8 orientation search are not guaranteed to be in different homogeneous partitions. This implicit searching scheme may fail to provide effective contextual information on a different dataset. An alternative of 8 orientation search should be proposed. In addition, the quality of contextual information also relies on the quality of homogeneous partition process. The method that we follow is unsupervised, and it is computation-costly. [19] proposes to use neural networks to perform the oversegmentation on a point cloud in a supervised manner. A more efficient and effective homogeneous partition method is worth being further studied to provide a more meaningful shape representation of the point cloud. The current point-based neural networks also have a limitation. The neural networks can only process a fixed number of points. This limits its ability to process large scale point clouds, while making use of spatial information in a wider range. This can be resolved by researching a more efficient point cloud data representation without a sacrifice of its spatial information.

Bibliography

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[2] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4413–4421, 2018.

[3] Alexandre Boulch, Joris Guerry, Bertrand Le Saux, and Nicolas Audebert. Snapnet: 3d point cloud semantic labeling with 2d deep segmentation networks. *Computers & Graphics*, 71:189–198, 2018.

[4] Nesrine Chehata, Li Guo, and Clément Mallet. Airborne lidar feature selection for urban classification using random forests. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 3):W8, 2009.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas A Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, volume 2, page 10, 2017.

[7] Jerome Demantke, Clément Mallet, Nicolas David, and Bruno Vallet. Dimensionality based scale selection in 3d lidar point clouds. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*, 38(5):W12, 2011.

[8] Jérôme Demantké, Bruno Vallet, and Nicolas Paparoditis. Streamed vertical rectangle detection in terrestrial laser scans for facade database production. *IAPRS I-3*, pages 99–104, 2012.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[11] Stéphane Guinard and Loïc Landrieu. Weakly supervised segmentation-aided classification of urban scenes from 3d lidar point clouds. In *ISPRS Workshop 2017*, 2017.

[12] Timo Hackel, Jan D Wegner, and Konrad Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3), 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Jing Huang and Suya You. Point cloud labeling using 3d convolutional neural network. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2670–2675. IEEE, 2016.

[15] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

[16] Mingyang Jiang, Yiran Wu, and Cewu Lu. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. *arXiv preprint arXiv:1807.00652*, 2018.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Loic Landrieu and Mohamed Boussaha. Point cloud oversegmentation with graph-structured deep metric learning. *arXiv preprint arXiv:1904.02113*, 2019.

[20] Loic Landrieu and Guillaume Obozinski. Cut pursuit: Fast algorithms to learn piecewise constant functions on general weighted graphs. *SIAM Journal on Imaging Sciences*, 10 (4):1724–1766, 2017.

[21] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018.

[22] Jiaxin Li, Ben M Chen, and Gim Hee 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.

[23] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018.

[24] Zhidong Liang, Ming Yang, Liuyuan Deng, Chunxiang Wang, and Bing Wang. Hierarchical depthwise graph convolutional neural network for 3d semantic segmentation of point clouds. *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[25] Wei Liu, Andrew Rabinovich, and Alexander C Berg. Parsenet: Looking wider to see better. *arXiv preprint arXiv:1506.04579*, 2015.

[26] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[28] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[29] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[30] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[31] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[32] Mark Pauly, Markus Gross, and Leif P Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization'02*, pages 163–170. IEEE Computer Society, 2002.

[33] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[34] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017.

[35] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pages 234–244. Springer, 2016.

[36] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 3, 2017.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[38] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[39] Riccardo Roveri, Lukas Rahmann, A Cengiz Oztireli, and Markus Gross. A network architecture for point cloud classification via automatic depth images generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4176–4184, 2018.

[40] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, page 0278364918767506, 2017.

[41] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1): 61–80, 2008.

[42] Martin Simonovsky and Nikos Komodakis. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[43] Hugues Thomas, François Goulette, Jean-Emmanuel Deschaud, and Beatriz Marcotegui. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. In *2018 International Conference on 3D Vision (3DV)*, pages 390–398. IEEE, 2018.

[44] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. *arXiv preprint arXiv:1904.08889*, 2019.

[45] Christoph Waldhauser, Ronald Hochreiter, Johannes Otepka, Norbert Pfeifer, Sajid Ghuffar, Karolina Korzeniowska, and Gerald Wagner. Automated classification of airborne laser scanning point clouds. In *Solving Computationally Expensive Engineering Problems*, pages 269–292. Springer, 2014.

[46] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[47] Martin Weinmann, Boris Jutzi, and Clément Mallet. Feature relevance assessment for the semantic interpretation of 3d point cloud data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 5:W2, 2013.

[48] Martin Weinmann, Steffen Urban, Stefan Hinz, Boris Jutzi, and Clément Mallet. Distinctive 2d and 3d features for automated large-scale scene analysis in urban areas. *Computers & Graphics*, 49:47–57, 2015.

[49] Martin Weinmann, Michael Weinmann, Clément Mallet, and Mathieu Brédif. A classification-segmentation framework for the detection of individual trees in dense mms point cloud data acquired in urban areas. *Remote sensing*, 9(3):277, 2017.

[50] Karen F West, Brian N Webb, James R Lersch, Steven Pothier, Joseph M Triscari, and A Evan Iverson. Context-driven automated target detection in 3d data. In *Automatic Target Recognition XIV*, volume 5426, pages 133–144. International Society for Optics and Photonics, 2004.

[51] Zhishuang Yang, Wanshou Jiang, Bo Xu, Quansheng Zhu, San Jiang, and Wei Huang. A convolutional neural network-based 3d semantic labeling method for als point clouds. *Remote Sensing*, 9(9):936, 2017.

[52] Jiaqian Yu and Matthew B Blaschko. The lovász hinge: A novel convex surrogate for submodular losses. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[53] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7151–7160, 2018.

[54] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[55] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.