Semantic Segmentation of Urban Meshes Utilizing MeshCNN

Xiao Tan

Duration: 7 months
Completion: July 2021
Supervisor: M.Sc Laupheimer Dominik, Prof. Dr.-Ing. Norbert Haala
Examiner: Prof. Dr.-Ing. Norbert Haala

1 Introduction

3D model semantic segmentation have been increasingly important in the computer vision field thanks to the development of 3D acquisition sensors such as LIDARs and RGB-D cameras. The high demand for automatic analysis of 3D data is, therefore, increase rapidly. There have a lot of mature methods been proposed for 3D point cloud. Meanwhile, the 3D mesh semantic segmentation has been a tough task due to the special characters of 3D mesh. Fortunately, a novel method named MeshCNN has proposed a promising solution utilizing mesh data directly. The topology of mesh is taken into account as the most exciting point. Within the scope of this master thesis, we proposed a framework utilizing MeshCNN that could handle the real-world urban 3D mesh dataset.

The input mesh data is generated from nadir images and been simplified. The theory of mesh convolution is presented. The mesh pooling is discussed and improved. The whole structure of the network is established with the encoder decoder architecture. The performance of the network is tested and evaluated.

2 Methodology

2.1 Mesh Convolution

To use the topologic information of the meshes, MeshCNN takes the edges as the convolution unit. Hence MeshCNN defines a convolution operator on the edges, where the topologic information is defined using the four incident neighbors. The incident neighbor is defined as 1-ring neighbor. The 1-ring neighbor of an edge is defined novelly by MeshCNN as the topology neighbor of an edge that together consists a face. Since convolution is defined as dot production between kernel k and the neighborhood, the edge convolution is defined the same way for each feature type:

$$f_{out} = f \cdot k_0 + \sum_{j=1}^{4} k_j \cdot f^j \tag{1}$$

where f^j is the feature of the j^{th} convolutional neighbor of target edge.

MeshCNN built an unwrapped matrix to rearrange every mesh to a matrix consisted by an edge-neighborhood row vector. In short words the matrixed mesh looks like: $\begin{bmatrix} e^0 & e^1 & e^2 & e^3 & e^4 \\ e^1 & e^2 & e^0 & e^5 & e^6 \\ \ldots \end{bmatrix}$. The corresponding relations of edges are shown in Figure 1.

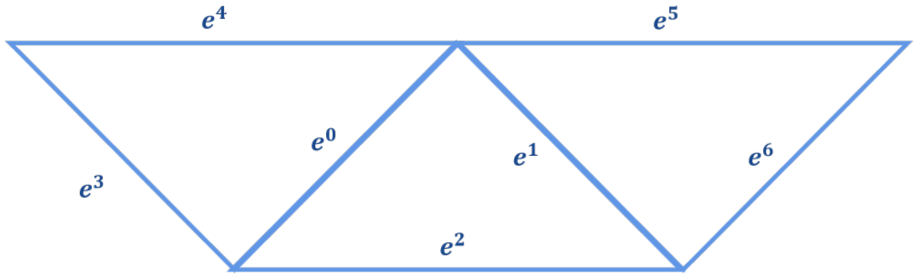


Figure 1: The relations of the edges presented in matrix.

Further symmetric calculation is implemented to the features, making the convolution invariant to the sorting orders.

2.2 Mesh Pooling-Unpooling

MeshCNN defines mesh pooling as a series of edge collapse operations. The entry edge for the pooling operation is called the target edge. The collapses are executed around it. For each target edge (e.g. edge 'a' in see Figure 2), the collapse operation detects if the 'adjacency shared' edge (e.g. edge 'e') exists. If yes, the collapse operation will delete all the edges in the large triangle (e.g. 'b', 'c', 'e')

The mesh unpooling, which shares the same purpose as image unpooling, is the inverse of the pooling operation. The topologic information has to be stored before pooling operations and used to recover the original shape of meshes. The matrix that used for storing the pooling result is shown in Figure 3

In our thesis, we improve the performance of the unpooling method by re-implement with sparse matrix and related functions. The improvement is shown in Figure 4

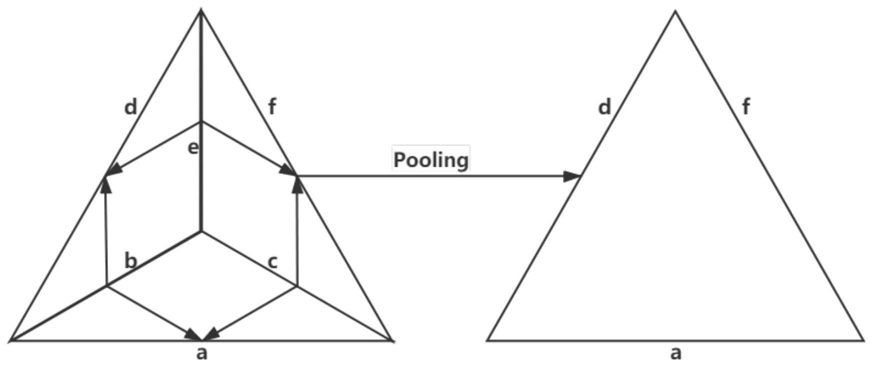


Figure 2: This kind of face structure is where the collapse occurs. The inside edges 'b', 'c', 'e' are deleted to meet the pooling requirements.

$$\begin{array}{c|ccccc} & e^0 & e^1 & e^2 & e^3 & e^4 \\ \hline e^0 & 1 & 0 & 0 & 0 & 0 \\ e^1 & 0 & 1 & 0 & 0 & 0 \\ e^2 & 0 & 0 & 1 & 0 & 0 \\ e^3 & 0 & 0 & 0 & 1 & 0 \\ e^4 & 0 & 0 & 0 & 0 & 1 \end{array} \xrightarrow[e^0+e^2]{e^0+e^1} \begin{array}{c|ccccc} & e^0 & e^1 & e^2 & e^3 & e^4 \\ \hline e^0 & 1 & 1 & 1 & 0 & 0 \\ e^1 & 0 & 1 & 0 & 0 & 0 \\ e^2 & 0 & 0 & 1 & 0 & 0 \\ e^3 & 0 & 0 & 0 & 1 & 0 \\ e^4 & 0 & 0 & 0 & 0 & 1 \end{array}$$

value	1	1	1	1	1
row	1	2	3	4	5
column	1	2	3	4	5

value	1	1	1	1	1	1	1
row	1	1	1	2	3	4	5
column	1	2	3	2	3	4	5

Figure 3: In the first row, the left matrix shows the before-pooling adjacency matrix. Every edge has no new adjacent edges. The right matrix shows the result after pooling that e^1 and e^2 are collapsed to e^0. The result matrix stores the pooling result by the corresponding values. In the second row, the left sparse matrix stores the same information as the before-pooling adjacency matrix. The right sparse matrix stores the same after-pooling information. The larger the original adjacency matrix is, the more memory we saved by utilizing the sparse matrix.

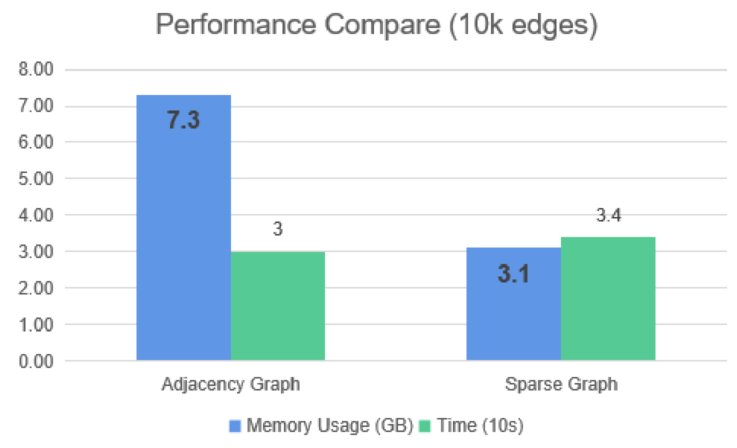


Figure 4: This chart shows the comparison while dealing with 10k edges. The GPU memory usage decrease from 7.3GB to 3.1GB while the totally performing time for one traing ephoch doesn't take much influence.

3 Result and Conclusion

The test result could be shown as Figure 5. We could see from Figure 5 that the valid labels take only a certain percentage (39.4% on average for our data) of the ground true labels. This is also the most important reason for slicing the original dataset mentioned in Section **??**. Under this situation, the test accuracy for validation dataset still reaches 72.35% and 69.283% for the 2 layers network and the 4 layers network accordingly.

Label	Color(RGB)
Invalid(-1)	Red(255,0,0)
Powerline(0)	Black(0, 0, 0)
Low Vegetation(1)	Cyan(0,255,255)
Impervious Surface(2)	Orange(255,97,0)
Car(3)	Yellow(255,255,0)
Fance(4)	Carrot(237,145,33)
Roof(5)	Pale white(250,235,215)
Facade(6)	Gray(192,192,192)
Shrub(7)	Aquamarine(127,255,0)
Tree(8)	Green(0,255,0)

Table 1: The original label-color chart.

For quantitative evaluation, the confusion matrixes are calculated for both training results of the test data set as Figure 6. As mentioned in Section **??**,

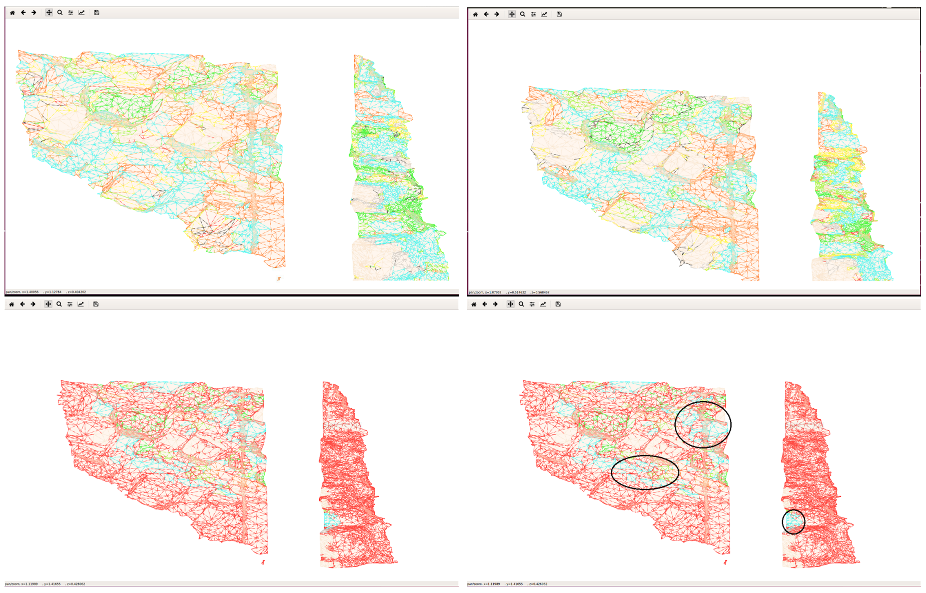
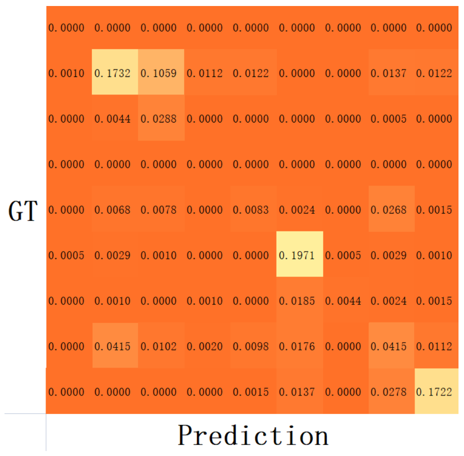
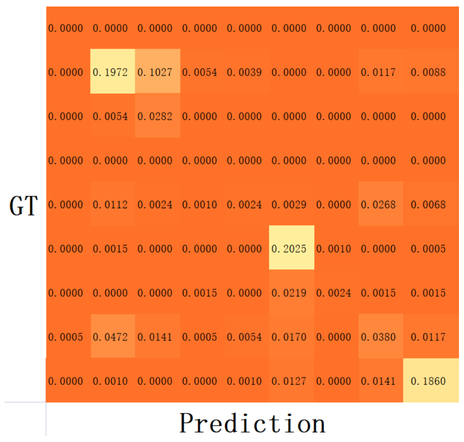


Figure 5: These images shows the results from our net works. The first row shows the result from two kinds of networks. The up-right image is trained from 4 layers network. The up-left image is trained from 2 layers network. The second row shows the ground truth images with or with not emphasis. The down-right image are the ground truth image. The down-left image is used for emphasizing the valid part of the ground true labels that take account in loss functions. We could clearly tell that in the emphasized part the prediction and the ground true labels are in high consistency. Although the percentage of GT labels are not large, the test accuracies still reach 72.35% and 69.283% for these two networks.



0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0010	0.1732	0.1059	0.0112	0.0122	0.0000	0.0000	0.0137	0.0122
0.0000	0.0044	0.0288	0.0000	0.0000	0.0000	0.0000	0.0005	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0068	0.0078	0.0000	0.0083	0.0024	0.0000	0.0268	0.0015
0.0005	0.0029	0.0010	0.0000	0.0000	0.1971	0.0005	0.0029	0.0010
0.0000	0.0010	0.0000	0.0010	0.0000	0.0185	0.0044	0.0024	0.0015
0.0000	0.0415	0.0102	0.0020	0.0098	0.0176	0.0000	0.0415	0.0112
0.0000	0.0000	0.0000	0.0000	0.0015	0.0137	0.0000	0.0278	0.1722



0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.1972	0.1027	0.0054	0.0039	0.0000	0.0000	0.0117	0.0088
0.0000	0.0054	0.0282	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0112	0.0024	0.0010	0.0024	0.0029	0.0000	0.0268	0.0068
0.0000	0.0015	0.0000	0.0000	0.0000	0.2025	0.0010	0.0000	0.0005
0.0000	0.0000	0.0000	0.0015	0.0000	0.0219	0.0024	0.0015	0.0015
0.0005	0.0472	0.0141	0.0005	0.0054	0.0170	0.0000	0.0380	0.0117
0.0000	0.0010	0.0000	0.0000	0.0010	0.0127	0.0000	0.0141	0.1860

Figure 6: The left confusion matrix is calculated from 2 layers network while the right confusion matrix is calculated from 4 layers network. Since the validation dataset are also choosen randomly, not all of the classes are contained.

invalid labels are not considered during calculation. The training results from the two networks are basically similar. The accuracy is 72.35% and 69.283% respectively. The low vegetation class (1) has the largest error number by misclassified with the surfaces and other vegetations. The fence class (4) has the largest error rate that can hardly tell from the other class. This may be caused by the simplification since these small objects suffer more geometric information loss.