DSC180B Report

Xinrui Zhan

Li Shang

3D Points Clouds Classification with Graph Neural Network

1. Abstract

This research focuses on 3D shape classification. Our goal is to predict the category of shapes consisting of 3D data points. We aim to implement Graph Neural Network models and compare the performances with PointNet, a popular architecture for 3d points cloud classification tasks. Not only will we compare standard metrics such as accuracy and confusion matrix, we will also explore the model's resilience of data transformation. Besides, we also tried combining PointNet with graph pooling layers. Our experiment shows that:

2. Introduction

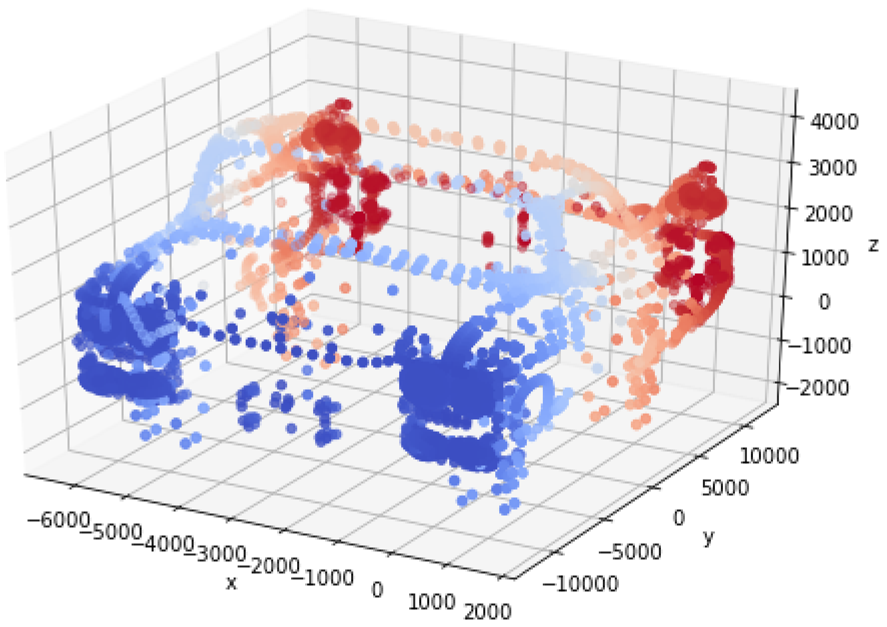
In this report, the Data section will discuss the dataset we used and the graph construct will include methods we used to transform the points to graphs. Mode and result section will discuss the details of our models and the running results and comparison of three models.

3. Data

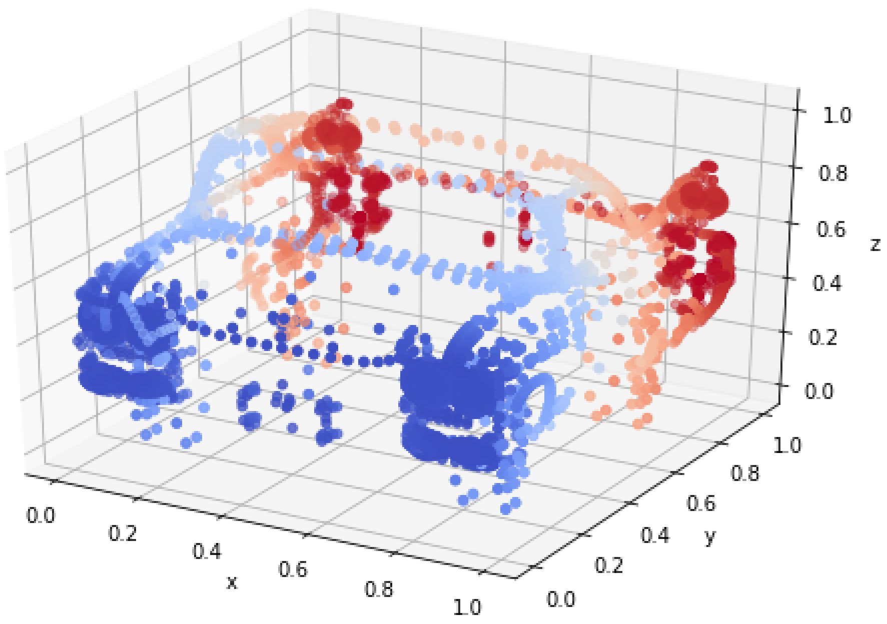
In this project, we will mainly use the ModelNet40 as our dataset. This dataset contains 40 categories such as airplane, TV_stand, guitar, etc. Each category has around 500 samples and every sample contains 3k-80k points. Points are all in the format of euclidean space coordinates (x,y,z). The range of the coordinates is not unified and varied a lot. Some samples may have points all in the range in (-100, -100, -100) to (100, 100, 100) while some may have (-4000, -4000, -4000) to (4000, 4000, 4000). This difference could result in imbalance models' parameters training and thus we used min-max normalization to normalize all points in all samples to range (0, 0, 0) to (1, 1, 1). Not only could normalization help us train a balance model, but also it will be convenient for us with the graph construction. Since the fix-radius graph construction needs a unique hyper-parameter r, instead of finding a best choice of r for all samples, we could simply use one number if we normalized our data first.

Sampling is also a key procedure. Samples with categories like air_plane, cars, or guitar usually have 60k+ points. Using 60k+ points will construct a graph with 60k+ nodes and 600k-1200k+ edges. The data dimension is too big to train the model efficiently and at the same time, increase the running time for data-loading and graph-construction. Not to mention that most of the points are unnecessary in model's training. In lots of cases, datasets include the inner points, which do not contribute to the determination of the category's shape. What is matter are those surface points which construct the shape of the sample. In our project, we decide to randomly select 1000 points for all samples. Some may doubt that the size may be too small while our baseline model PointNet shows that 1000 points are indeed sufficient. What's more, since we regularize the number of points, we do find implementing the model's architecture more easily since we have a unified input shape.

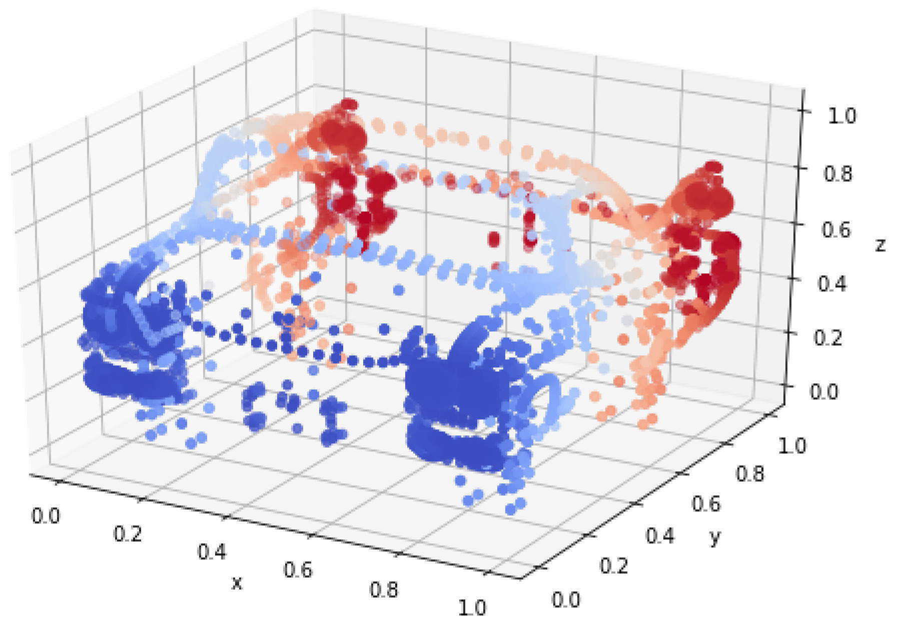
Below are three images of the raw data, the normalized data, and the sampled normalized data. We could see that even after the sampling process, we still could clearly tell the object's shape.



Raw data; "Car"



Normalized data; "Car"



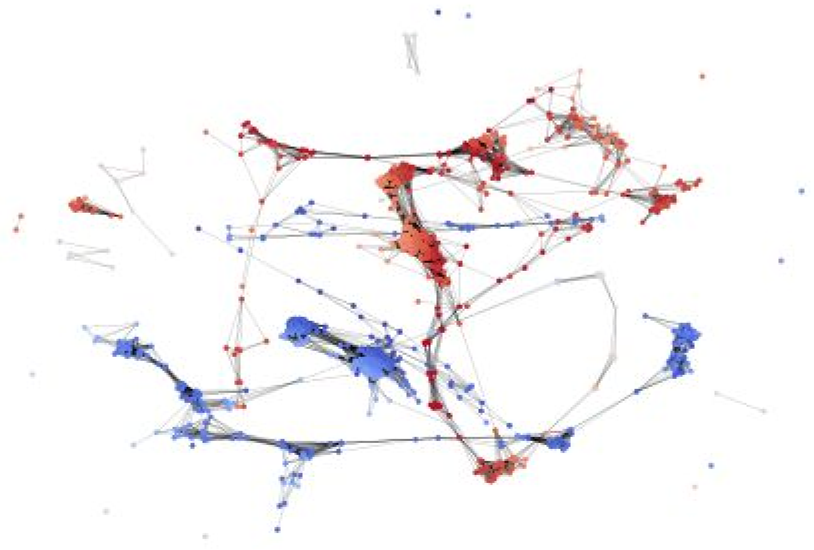
Normalized and Sampled data; ‘Car’

After models-training, we also tested our model’s performance on a different dataset ShapeNet. ShapeNet is a public 3d points cloud data and shares a lot of the same categories with ModelNet40. Through testing on samples that models have not seen in the training process, we could measure our model’s elasticity.

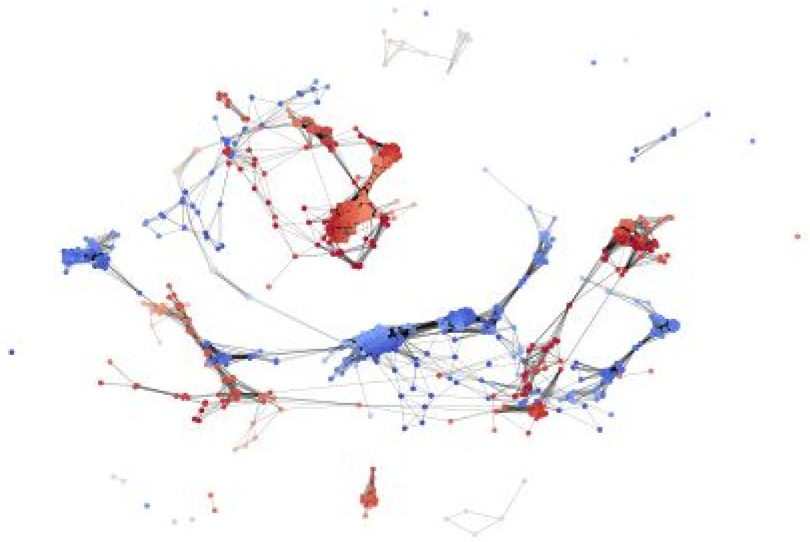
Data augmentation is also used in this project to improve the accuracy and measure models’ resilience. Our goal is to train models predicting the shape of a point cloud. Therefore, if the incoming points: 1. Globally translate with one direction; 2. Rotate a certain angle; 3. Stretch or squeeze with a degree in certain ranges, our model should be able to predict the same result. In this project, normalization will help us deal with transformation and we used the data augmentation testing the left two.

4. Graph Construct

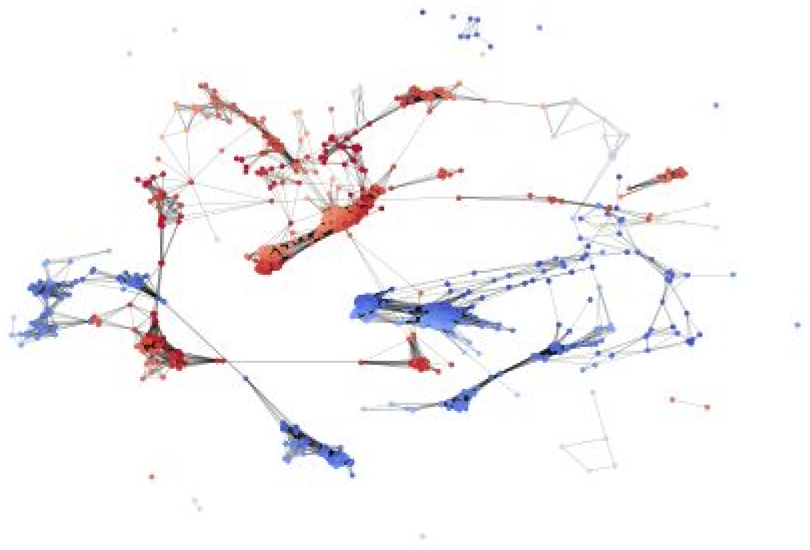
In order to train a Graph Neural Network, we need first convert our dataset into a collection of graphs. To be more specific, we need an adjacency matrix or adjacency list to represent the edges’ information. In this project, we used two different ways to construct the graph. The first one is the fixed-radius distance. We calculated the euclidean distance between each node and set a threshold. Any node that is farther away from one node than this distance will not be considered as one’s neighbor. For nodes that are the range of this distance, we will store the distance either by a distance matrix or an adjacency list. The second method is using k-nearest-neighbors to find k most nearest neighbors of one node. Then storing either the distance or simply binary information as a matrix or adjacency list. The advantage of using fix-radius is that for different nodes, we can have different connection density. Thus we could distinguish nodes' popularity. For example, a node with lots of neighbors may be located right in the center of the shape while a node with few neighbors is in the corner. From the model's training perspective, this could help us aggregate the node’s features' information. The features of nodes with more neighbors will be aggregated more often and thus play a more important role. The drawback of fix-radius is that the value of the hyper-parameter ‘r’ is hard to define. Different r values could result in a highly different graph and thus affect our model’s result. In general, a higher value of r could result in a more dense graph while a smaller number could result in a more sparse one. In our model’s comparison process, we will compare our models’ performance on datasets generated by different r values. Below are images of graphs that are constructed based on the same data points but with different r values.



Graph constructed with r = 0.1



Graph constructed with r = 0.2



Graph constructed with r = 0.3

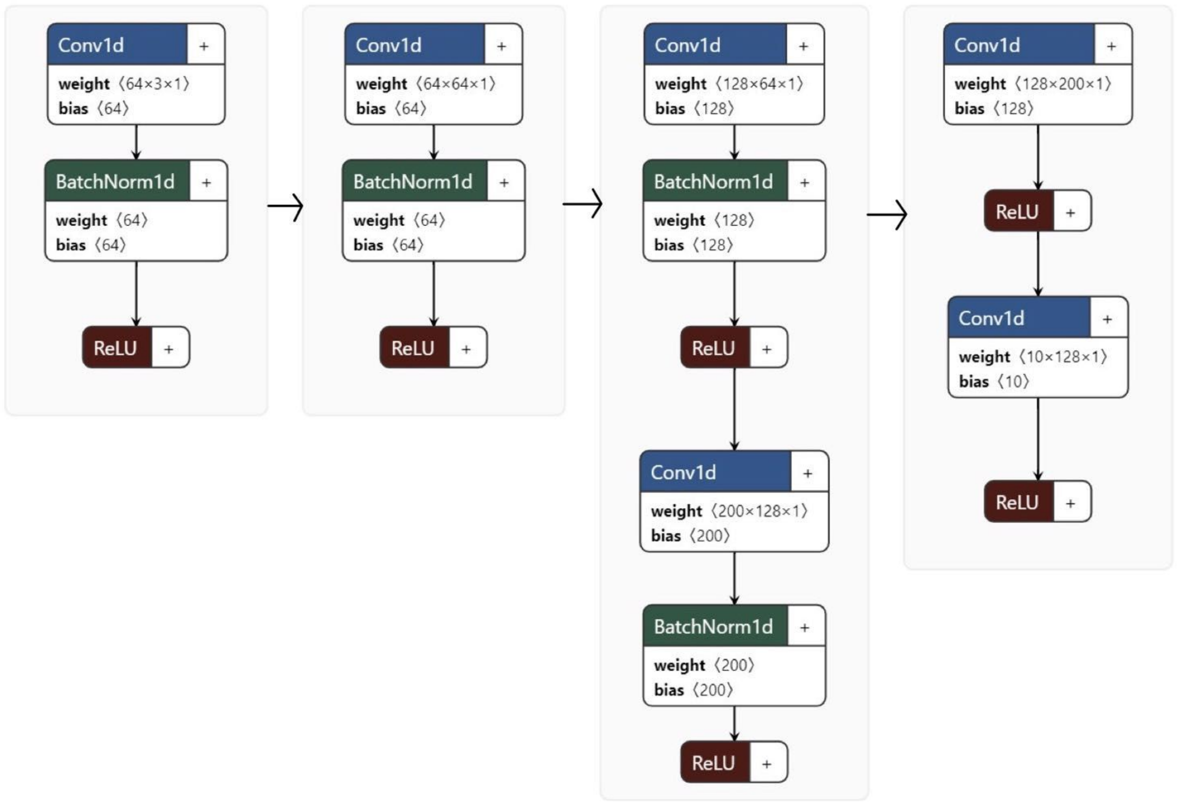
The advantage of using k-nearest neighbors is that we could have the same neighbors for all points thus we do need to worry about the hyper-parameter setting in-class influence.However, it is also the drawback since we could no longer apply the connectivity of one node to others in our models. We will use both methods to create datasets and train our models on each one. The comparison results will be discussed in the Result section.

5. Models

In this section we will discuss models' implementation details. Graph Convolution Network (GCN) is our main focus. PointNet is used as a baseline model and since it is only used for comparison, we designed a basic architecture. In the last part, we will discuss the effects of combining graph pooling layers with PointNet. All three models are predicting on a points set. Thus, all models should be invariant to permutation. In other words, if we change the order in which we pass in our data points, the predicting results should always be the same. Thus, maintaining this speciality is an important key in implementation of models. We will discuss the permutation invariance of the three models in the corresponding section.

5.1 PointNet

PointNet is a well known model's architecture and widely-used in 3d points data machine learning tasks. It fully used the Multi-Layer-Perceptron (MLP) and symmetric functions to maintain the model's permutation invariance. More specifically, MLP is a layer that is trained to learn a function for all the points. Letting δ be the function MLP learned. For all points, the parameters α^* of δ are all the same. Thus, set $\{ \delta(\alpha_1), \delta(\alpha_2), ..., \delta(\alpha_n) \}$ should always be the same for all permutation of set of points $\{ \alpha_1, \alpha_2, ..., \alpha_n \}$. Connected MLP with symmetric functions such as max-pooling or mean-pooling will help us to reduce the data dimension while at the same time still keep the permutation invariance. After multiple blocks of MLP and pooling layer, we transform our data points to a feature vector and use them to predict the class labels. When implementing the code, we choose to use the convolution 1d layer to represent the MLP layer. The full architecture is shown below:



PointNet is only used as a baseline model in our project. Thus we only keep its core layers and design it simply. The architecture is similar to the architecture of our Graph Convolutional Network.

5.2 Graph Convolutional Network

5.3 GNN-PointNet

6. Results

7. Conclusion

8. Appendix