University of Illinois FALL 2019

CS 598PS Term Project: 3D PointCloud Segmentation and Classification

Name	Jiayi Luo	Zhoutong Jiang	Zixu Zhao
NetID	jiayil5	zjiang30	zzhao39
UIN	676472385	651476677	665628393

GitHub

1 Introduction

The urbanization and rapid growth of lead to congestion in urban transportation system, fuel consumption, and CO2 emissions, posing threat to our environment as well as economy. It is believed that self-driving cars can be a potential solution to a large number of problems in our society. The merits of self-driving cars are discussed substantially in [2].

Although many efforts have already been made towards incorporating the self-driving cars into our communities in a safely manners, one major safety concern about the self-driving cars is whether they can successfully detect objects, for example, pedestrians, obstacles, and other vehicles on the roads, and avoid collisions with these objects. There is a large body of literature about 2D object detection/classification, interested readers are refer to [1] for a complete review/introduction of the 2D object detection/recognition models. While the 2D models are very reliable and accurate, they might not be totally suitable for object detection in self-driving industries given the complexity of real world condition. 3D object detection, on the other hand, could provide more degrees of freedom and thus can potentially provide a more complete representation of different objects and scenarios in real world [4]. However, as compared with the traditional 2D object detection methods, 3D object detection requires substantially different modeling approaches and input data. In this project, we will explore how 3D PointCloud Segmentation and Classification can be utilized in self-driving industries. The most relevant work we found in literature is [3], while the main difference is they mainly focus on the detection of objects and the application of data fusion techniques.

The key contributions of our project are summarized as follows,

- We proposed a customized 3D object segmentation architecture as well as its calibration approaches. Specifically, we design our models/metrics in the way that it can decide the number of objects in the street scenes as well as the classes for each objects simultaneously.
- We modified the classification model architecture proposed in [6] to achieve a better classification accuracy.
- We demonstrate the effectiveness and applicability of our proposed model through a series of numerical studies. Results show that our model can achieve a relatively high classification/segmentation accuracies in a reasonable computation time.

This report is structured as follows, Section 2 introduces the details of the data set we are using as well as the important pre-processing steps. Section 3 presents the modeling approaches for our classification tasks and segmentation tasks. The architectures of our proposed models as well as the evaluation process of our model are discussed. Section 4 demonstrates the details of the parameters tuning process. Section 5 shows how our model could be applied to real world cases through numerical results. Finally, Section 6 discusses potential future research directions/extensions for this project and summarizes the whole project.

2 Datasets Description

2.1 Data Structure

The dataset we use for this project is Lyft 3D object detection dataset on Kaggle challenge. The link for this dataset can be found in: https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles.

The dataset consists of the following fields:

- scene - Consists of 25-45 seconds of a car's journey in a given environment. Each scence is composed of many samples.
- sample - A snapshot of a scence at a particular time instance. Each sample is annoted with the objects presented.
- sample annotation - An annotated instance of an object.
- instance - An enumeration of all object instance we observed.
- category - Taxonomy of object categories (e.g. vehicle, human).
- attribute - Property of an instance that can vary within the same category.
- sensor - A specific sensor type.
- calibrated sensor - Definition of a particular sensor as calibrated on a particular vehicle.
- map - Map data that is stored as binary semantic masks from a top-down view.

The most important data is the LiDAR data. The dataset also consists of RGB .jpeg data that's used for 2D object detection, which complements the LiDar data since LiDAR is short for Light Detection and Ranging. LiDAR is a very common method used to generate accurate 3D representations of the surroundings in autonomous vehicle industry. The idea behind LiDAR is fairly simple. An object in 3D space is illuminated by a laser beam, and the laser light is reflected back to the LiDAR device. The device then collects the reflected light and the time required for the light to travel is used to infer distances. [5] Since LiDAR combines several sensors together, and each sensor may record different distances. This difference is used to calculate the depth of an object. The distance inferred from the travel of light provides depth information combined with the 2D RGB representation. This way provides an accurate 3D representation of the object. The entire process is quite similar to how human beings perceive the world. In this dataset, we have 6 classes: car, bus, truck, pedestrian, bicycle, and other vehicle. The majority of the objects are cars while other classes have similar quantities.

2.2 Data Preprocessing

The original dataset structure is complex, so we need to obtain data that can be used by the deep learning dataloader through data preprocessing. Firstly, we need to select the pointcloud from all the scenes and also select the corresponding labels. The dataset is saved in json files, a group of json files are identified by a 'Token' key. We access and obtain the corresponding scene data using 'Token'. Each scene consists of the pointclouds collected by 3 LiDARs during a 45s period. Each scene also has the corresponding label files. All the labels are provided by Lyft dataset, and they are hand labeled by experts.

Another thing we did is to remove the background noisy data points that don't belong to any

object. The processed data has about 20,000 objects in about 2,000 scenes. In each scene, the processed data is stored in a 5 by n matrix, in which we have x, y, and z coordinates, class id, and object index for all n points. This matrix is then passed to the dataloader for further processing.

2.3 DataLoader

We prepared two different types of dataloaders for this project, the classification task dataloader and the segmentation task dataloader.

2.3.1 Return Values

For classification task dataloader, it returns a 3 by n (i.e. point cloud) matrix and a n by 1 array (i.e. the class id for each point).

For segmentation task dataloader, it returns a 3 by n (i.e. point cloud) matrix and a n by 1 array (i.e. the object id for each point).

2.3.2 Implementation Details

For classification tasks, the incoming data typically has about 2 to 20 different objects. Some objects have a few thousand points, and some others may only have a few dozens. If the number of points for an individual object is too small, it's not meaningful for classification tasks, because an object has to have a certain number of data points to represent its features in order to be classified. We set a threshold for this purpose. In our experiment, this threshold is typically 200 to 500. What we do in this step is to save each individual object from each scene, with a fixed number of points to represent it. Each file also has an class id for its object.

For segmentation tasks, the concept is similar. The only difference is that we need to save an entire scene to a file. We first filter out the object whose number of points is small. Another thing we need to guarantee is that the number of points in each scene is fixed. To this end we randomly sample 2500 points from each scene.

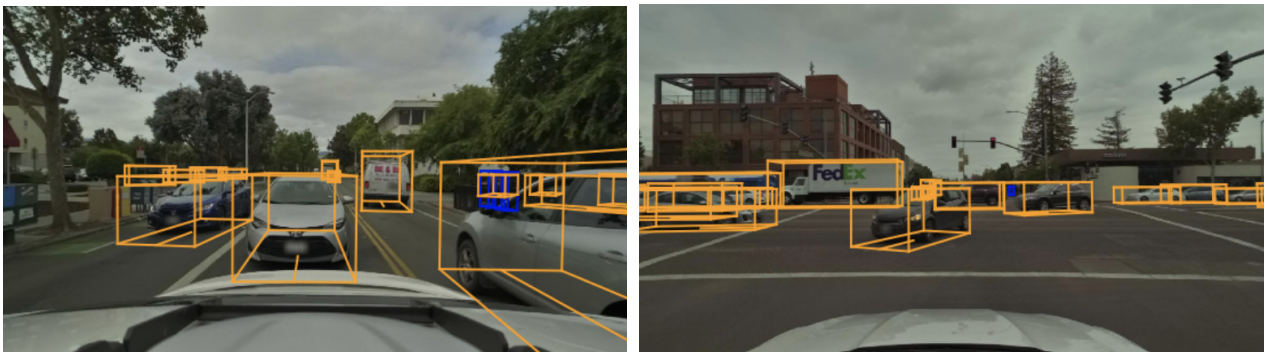

Figure 1: Sample images from cameras

Figure 2: Sample images from LiDAR

2.4 ShapeNet Dataset

ShapeNet dataset serves as the baseline in our implementation to evaluate the correctness of our algorithm.

ShapeNet dataset was the data used in the original paper [6]. We use this dataset for some preliminary testing of our algorithm. The ShapeNet dataset is a data collection of a few common objects in real life, like planes, chairs, hats, etc. The classification task is simply to classify an individual object. The segmentation task is to segment out the different parts of an object. For example, for a plane object, classification task is to classify this plane, as a plane. Segmentation task is segment out the wings, plane body, etc. The structure of this dataset and the problem definition is different from ours, but the Lyft dataset is more challenging than this. In Lyft dataset, we need to first pick out all the instances from a scene, and then we run classification to identify each instance.

3 Model Details

3.1 Classification Architecture

The code for classification is modified based on original pointnet implementation [6]. It mainly consists of three major components as shown in Figure 3:

3.1.1 Major Components

1. T-net: T-net is a subnet which aims to handle the intrinsic property of the point cloud. It is also the means to make the training invariant under spatial transformations. The detailed proof steps can be found in the original paper.

2. Dilated Convolution Layers: different from the original implementation, where the authors want to get rid of the interaction among adjacent points, we ensure that near-by points in the raw data will also be close to each other under the processed data format (numpy array) through our preprocessing procedure. Thus we want to introduce the local geometric information in our model. Instead of using Convolution layer with kernel size = 1, the dilated convolution layer is utilized here to receive a large reception field and to capture the local information without adding too many hyper-parameters.

3. Max Pool Layer: The purpose of this layer is to obtain the global information from all the point-wise values. It reduces the dimensions of the point cloud feature map to the global feature map before applying fully connected layer.

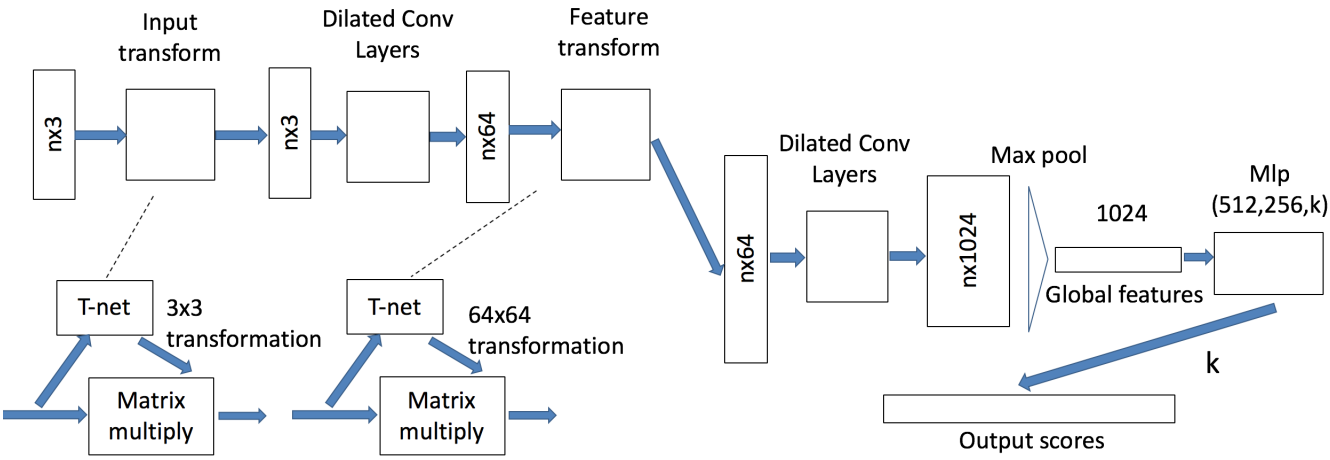

Figure 3: Classification Model Architecture (k = number of classes)

3.1.2 Implementation Details

- The T-Net also use dilated Convolution Layer to capture local feature.
- The T-Net includes a max filter operation among the feature depth direction to drop redundant features before the final fully connected layer.
- The reception field option is implemented for the dilated convolutional layer as adjustable parameters. The default value is 5%.
- The transform loss is added during the training process to regularize the T-Net's inputs and avoid over-fitting issues.
- The dropout layer is added with 0.2 dropout probability to reduce the impact of over-fitting.

3.2 Segmentation Architecture

We implement all the Segmentation architectures from scratch except the $n \times 1088$ point feature map part, which is borrowed from the Classification branch.

The original segmentation task described in pointnet implementation is totally different from than ours, which can be described as follows:

- In the original model for the original segmentation task, for a specific object, the number of classes and the corresponding labels are assumed to be fixed and given in the ground truth data, for example, we know that a cup will consist of the handle and its body.
- The label for each point in the original segmentation task is deterministic and predefined. For example, cup body is assigned with label 0 while cup handle is assigned with label 1.

While in our task, we want to handle a more general case which can be describe as follows (similar to the 3D pointnet instance segmentation):

- In the ground truth data, the labels of classes are not predetermined, instead, we can only differentiate different classes from the labels. Consider the previous cup as an example, if Model 1 labels the body as 1, handle as 0. Model 2 labels body as 0, handle as 1. Then these two models are considered equivalent.

3.2.1 Major Components

The architecture consists of three major components as shown in Figure 4:

- The head part, i.e. point feature map. This part is borrowed from the Classification Network architecture. To get the point feature map, after obtaining the global feature, we replicate it to increase its dimension and append it to the previous point local feature map. This part is used as the feature map input for the following two branches: (i) the object number prediction branch, and (ii) the point labeling branch, which are described below.
- The object number prediction branch. This branch aims to predict the maximum number of objects in the scene and trim the labeling branches to ensure the output label won't exceed the predicted boundary. The default maximum objects in one scene is assumed to be 40 in our model (which is an adjustable parameter). The loss for this branch is the the cross entropy loss between the predicted maximum number of objects and the ground truth number of objects.
- The point labeling branch. This branch aims to predict the label for each point after trimming (using output from the object number prediction branch).

  The loss for this branch is the summation of the following four loss functions:

  - 1. Ground-Truth based Matching Loss: to calculate this loss, for each ground truth object, we need to (i) Extract the label for this object from the ground truth data. (ii) Find all the corresponding predicted labels from the predicted output. (iii) Find the majority label of the predicted label set and assign index 0 to this label. (iv) Change the rest of the labels in the predicted label set to 1. (v) Compute the cross entropy loss between the modified predicted label set and the array of all 0s (ground truth). Notice that if a label was previously used for an object i, then it can no longer be used for a new object j and thus the next majority label will be used for object j.
  - 2. Prediction based Matching Loss: this loss is the mirror of the Ground-Truth based Matching Loss, in which for each object, we (i) Extract the label for a certain object

from the predicted output. (ii) Find all the corresponding labels in the ground truth dataset. (iii) Find the majority label of the ground truth data and assign index 0 to this label. (iv) Change the rest of the labels in the ground truth to 1. (v) Compute the cross entropy loss between the modified predicted label set and the array of all 0s (ground truth).

- 3. The Local Geometric Loss: This loss is specifically designed for the Lyft dataset, which is not included in the ShapeNet dataset. The loss increases with the distance between each point in the object and the center of the object. We considered this loss because we found that many point clouds in Lyft dataset contain several objects that are adjacent to each other.
- 4. Transformation loss: This loss is applied the same as the classification subroutine to avoid the over-fitting issue.

The detailed calculations of these loss are included in the GitHub link at the beginning of this report.

3.2.2 Accuracy Metric

The calculation of the accuracy metric is similar to the calculation of the two macthing loss functions described previously, which can be divided into two parts:

Part 1 (Precision Information). a) Extract one ground truth object's label set. b) Find all the corresponding predicted label set from the predicted output. c) Find the majority labels of the extracted predicted label and treat them as the correct ones. d) Repeat for all ground truth objects.

Part 2 (Recall Information). a) Extract one predicted object's label set. b) Find all the corresponding ground truth label set from the ground truth input. c) Find the majority labels of the extracted ground truth label set and treat them as the correct ones. d) Repeat for all predict objects.

The definition for accuracy is different from the typical definition. We tested our model with the loss from only Part 1, however, we realized that this metric only contains the precision information and ignore the recall information and will produce inaccurate results. Hence we finally decide to consider both of them, and the final accuracy is the average of the accuracies from these two parts.

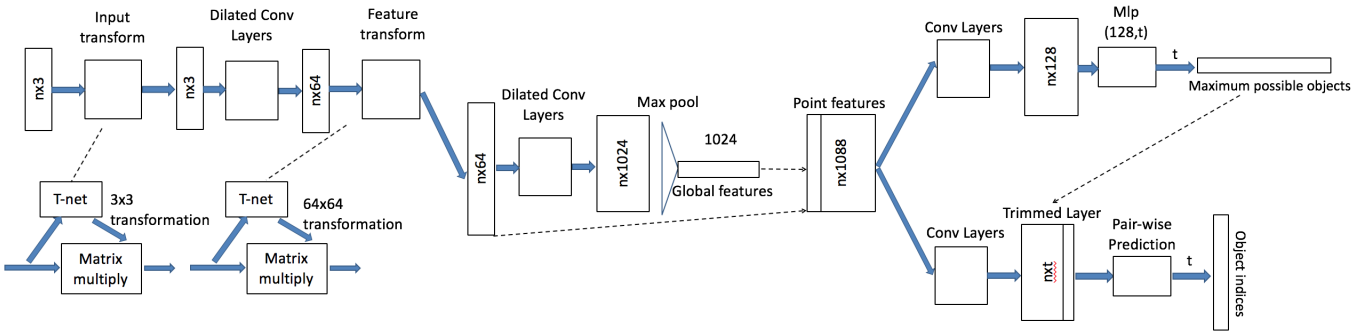

Figure 4: Classification Model Architecture (t = number of objects)

3.2.3 Implementation Details

- The matching loss for point labeling branch is divided into two parts to avoid the model predicting all labels to 1 (bias) same value.

- The accuracy term contains both precision information and recall information, not the typical accuracy information.
- The minimum accuracy for one scene is $0.5*\frac{100}{num\ of\ predicted\ objects}\%+0.5*\frac{100}{num\ of\ ground-truth\ objects}\%$, the maximum accuracy for one scene is 100%.
- Dropout layer is added with 0.2 probability to reduce the over-fitting problem.

4 Hyper-Parameters Tunning

Reception Field: The reception field parameter we select is 5%. We tested this value through 2%, 5%, 10% and 15%. It turns out 5% works the best after trading off between the speed and accuracy.

Coordinate Alignment: The data preprocess requires us to extract the point cloud coordinates from the raw data. Then apply the intrinsic and extrinsic LiDar sensor parameters to adjust the coordinates back to rectangular coordinate system.

Training Epoch: For classification task, 5 epoch is selected after stabilization. For segmentation task, 2 epoch is selected due to 1)the computation power limitation, 2)the somehow low quality of the dataset.

Input size: For classification task, we select 1200 as the input dimension and for segmentation we select 2500 as the input size.

Computation Resources: For classification, in total 20 GPU hours are used on Google Colab with GPU capability. For segmentation, in total 100 GPU hours are used on Google Colab with GPU capability.

5 Evaluation

The evaluation for this project is basically divided into two steps:

1. Apply our model to the original Shapenet dataset (Correctness Check).
2. Apply our model to the Lyft dataset (Application).

- For classification task, we trained the classification network with original shapenet dataset for 10 epoches and reached 95.4% accuracy at the end. Then we trained our model using Lyft dataset for 5 epochs. The test accuracy increased from 55% to 87%. The test loss decreased from 2.2462 to 0.3428.

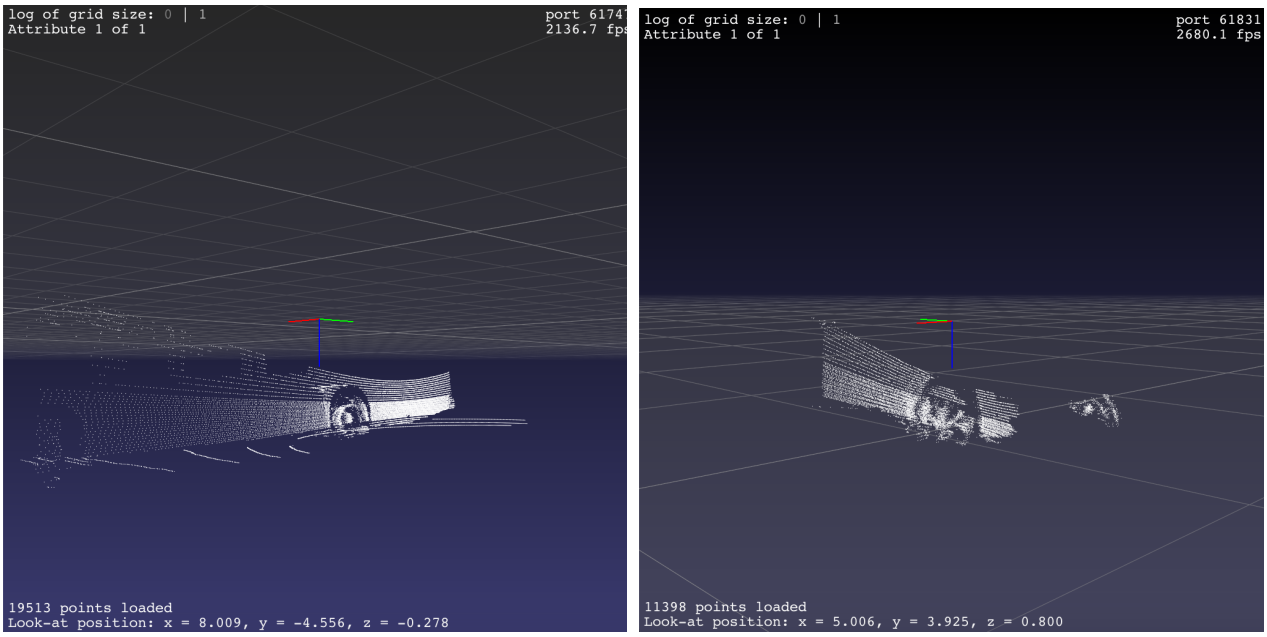

Figure 5: Samples of the successfully segmented and classified point clouds. Bus (left); Car (right)

- For segmentation network, we first trained the network with ShapeNet dataset for 4 epochs. The test accuracy started from 55% and reached almost 84%. The test loss dropped from 5.08943 to 0.23586 at end. Then we trained it using Lyft dataset for 2 epoches (we ignore the number of class information and use the whole dataset). The test accuracy increased from 54% to 89%. Due to the computation power and time limitation, we didn't record the test loss along the training process. Only the beginning and the final values were recorded. It started at 3.48737 and dropped to 0.54759 at end. The train loss and train/test accuracy plots can be found in Figure 7.
- The results of classification using Lyft dataset doesn't reach as high as 95% as compared with using ShapeNet dataset. This is due to the intrinsic difficulty of our task since most of the objects in the dataset only consists a small portion of the entire object. Like the bus in the Figure 5, the object only contains the lower right surface of the bus.
- The segmentation accuracy on Lyft dataset becomes stable after only 2 epochs of training. This is probably due to the property of Lyft dataset. In the dataset, more than 30% of the segmentation input contains only 1 object, while more than half of the datset contains only 2 objects. Additionally, since segmentation is a larger network as compared with the classification network, the performance of our model heavily relies on the quality and size of the dataset.

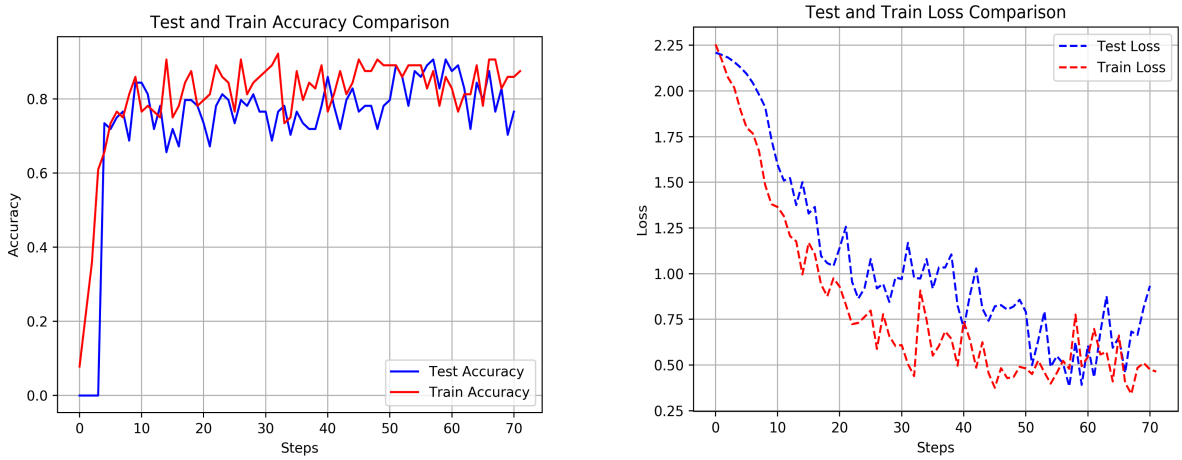

Figure 6: Classification accuracy and loss

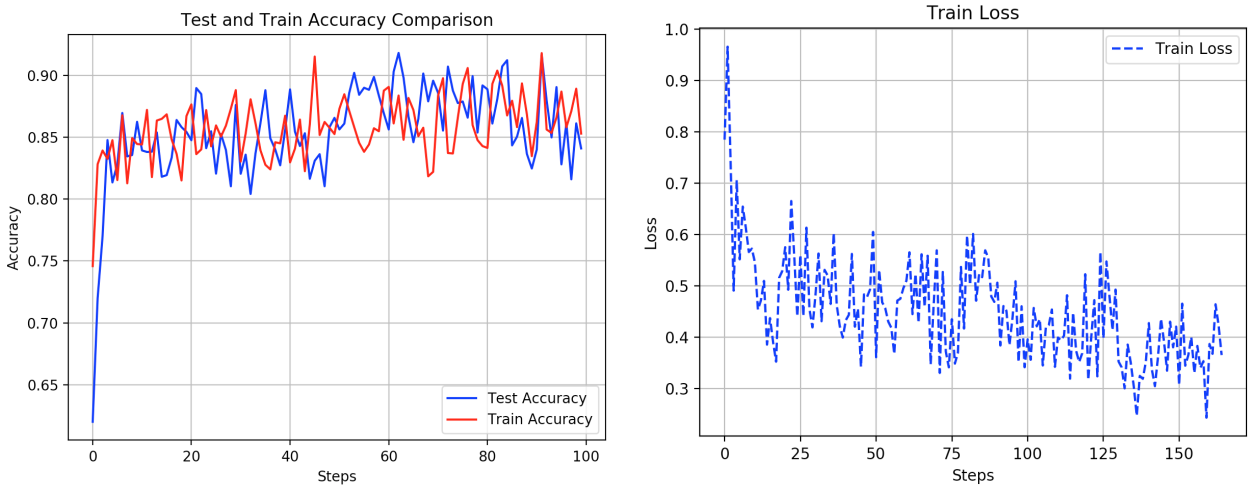

Figure 7: Segmentation accuracy and loss

6 Conclusion and Future Work

In this project, we proposed a 3D object segmentation framework and complete model calibration approaches. A model that enables the classification of objects in street scenes was also presented. We also applied the proposed model to real world data sets to evaluate the performance of our framework. It was shown by numerical experiments that our model could be used to perform classification/segmentation tasks for 3D objects and achieve a high accuracy.

This project can be extended in a few directions. The first thing we can try to see if it will make any difference is to replace dilated convolution with the idea from PointNet++ [7]. PointNet uses a single max pooling operation to aggregate the whole point set, we use dilated convolution to improve on this idea. PointNet++ uses a hierarchical grouping of points to progressively abstract a larger local regions.

Another we can try is to change the segmentation network into a two-step approach or multi-step approach. The segmentation network first predicts the maximum possible objects existing in the scene, and this information is concatenated into the object segmentation part. Instead, we could try to train the first branch, fixed the second branch. After it's stable enough, we'll then add the network and train the second branch. After loss reducing to some threshold, resume the training for the whole network.

The way we accuracy for segmentation can also be improved. Our accuracy is based on the assumption that the majority of points have the same location as the label. This assumption is not guaranteed to be correct. In the future we'll try to correlate the predicted output with their positions first, and then we'll compare the outputs with the labels.

References

[1] Y. Amit. *2D object detection and recognition: Models, algorithms, and networks*. MIT Press, 2002.

[2] L. Bell. 10 benefits of self-driving cars, 2019.

[3] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[4] P. L. Liu. Lifting 2d object detection to 3d in autonomous driving, 2019.

[5] T. S. Paparaju. Lyft competition : Understanding the data, 2019.

[6] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[7] C. R. Qi, L. Yi, H. Su, and L. 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.