School of Electronic Engineering and Computer Science

MSc Big Data Science
Project Report 2018

3D Point Cloud Data Classification

Po-Kai Chen

August 2018

Disclaimer

This report, with any accompanying documentation and/or implementation, is submitted as part requirement for the degree of MSc in Big Data Science at the University of London.

It is the product of my own labour except where indicated in the text.

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Miles Hansard, for guiding me on the project, and providing me GPU-resource, which help me a lot on deep learning task. Without his help, I believe it would take me much more time on finding starting point.

In addition, I would also like to acknowledge Mr. Haris Krikelis for helping me with the problem I met when I tried to remotely access GPU server.

In final, I wish to give special thanks to my beloved family, who always support and encourage me.

Abstract

Although deep learning has been widely discussed in image processing and has achieved excellent results, 3D point cloud data deep learning technique is relatively immature since it is not universal and image-related deep learning methods cannot be applied directly due to the way how data is recorded. However, because the demands for automatic driving, river survey, etc. have increased significantly in recent years, and point cloud data has some characteristics suitable for these tasks, it becomes more important.

In this paper, we use ModelNet40 (Wu et al., 2015) data provided by Princeton University to explore two of the best-performing point cloud deep learning classification methods, PointNet (Qi et al., 2017b), and PointCNN (Li et al., 2018). We train the models and evaluate their robustness with different kinds of noise data, and data with missing value. In addition, we also modify the structure of PointCNN to improve its performance.

Table of Contents

Table of Figures

Table of Algorithm

Table of Table

1. Introduction

1.1. Background

3D point cloud data is a set of data points which is represented by its three coordinates (x, y, z) in space (Qi et al. 2017b), some also include colour information (R, G, B). One of the well-known representatives is Light Detection and Ranging (Lidar) data (Reutebuch et al., 2005), which is collected by illuminating pulsed laser light at a target area and calculating the distance via the difference of return time that laser back to sensor.

Deep learning (LeCun et al., 2015) is a subfield of machine learning which can learn data representations and features itself. It has already a big success on the tasks such as image classification and segmentation.

1.2. Motivation

Though deep learning is very successful in the field of image and video, there are some limitations on these kinds of data. For example, RGB-video cannot detect the range and the distance of the objects. Furthermore, videos will be influenced by some condition such as bright sunlight, pitch darkness, rain, which Lidar would not; hence, Lidar can perform equally well day and night.

Thanks to the advantages of Lidar, it can be used on autonomous vehicles, agriculture, river survey etc. It can have many usages and be able to improve the life of human if point cloud data can be handled properly.

1.3. External Libraries

TensorFlow (Abadi et al., 2016), consists of tensor and flow, is developed by the Google Brain team as an open-source software library for dataflow programming. In TensorFlow, a model is represented as a data flow graph, which contains a set of nodes called operations. Each operation takes a tensor as input and outputs a tensor, where tensor is the representation of data and flows between operations.

1.4. Data Introduction

The data can be accessed on the Princeton ModelNet website, the link is shown as below: [online] http://modelnet.cs.princeton.edu/

The training data is point cloud of ModelNet40 models (Wu et al., 2015) made up of 40 objects categories in HDF5 files, with dimensions (9840, 2048, 3), which is separated into five files, where the first four with (2048, 2048, 3), and the last one with (1648, 2048, 3). That is, it contains 9840 objects, each object is made up by 2048 points with coordinates (x, y, z), and belongs to one of the forty categories. 2468 objects are in the testing data, and is separated into two files, containing 2048, and 420 respectively.

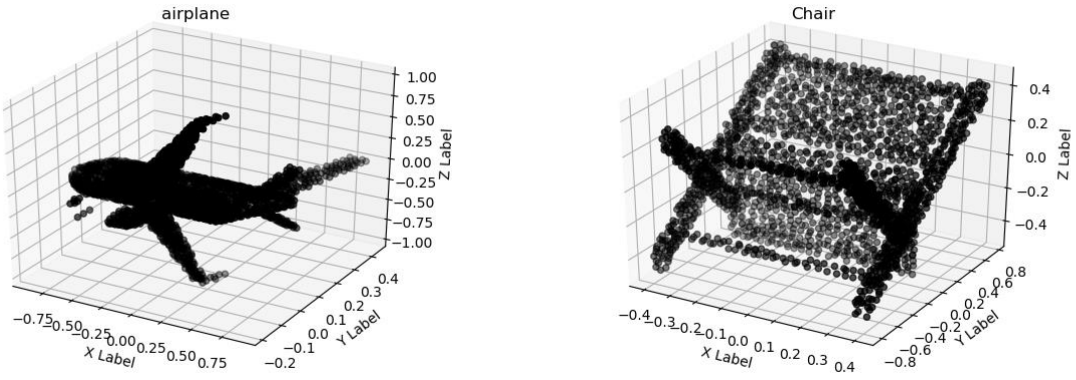
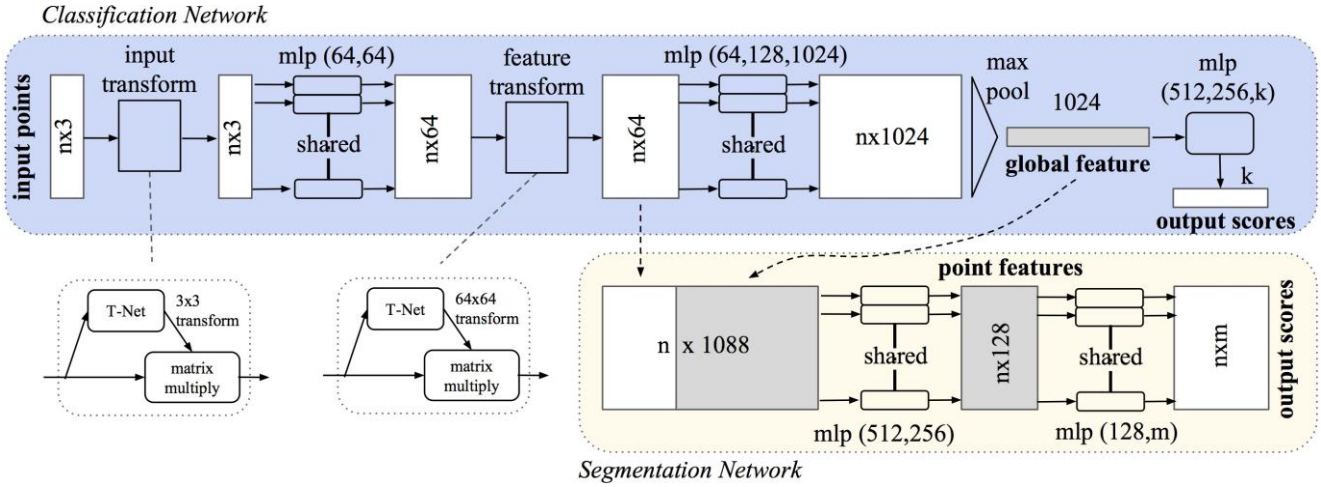


Figure 1 Visualization of ModelNet40 Data

Figure 1 is the visualization of two sample in dataset by using python library ‘matplotlib.pyplot’. It allows users to rotate and view the data from different angle.

2. Literature Review

2.1. PointNet



C. R. Qi et al, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," 2016.

Figure 2 PointNet Network

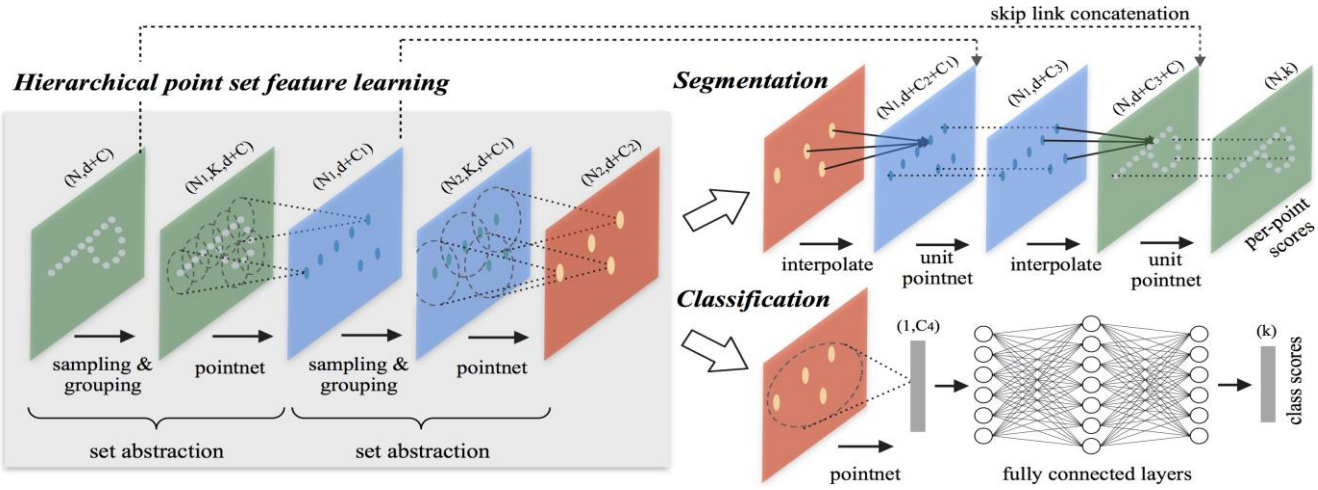
Unlike images and voxel arrays, Convolution Neural Network (Krizhevsky et al. 2012) cannot be applied directly to point cloud data because it has an irregular format. In other words, the former two kinds of data are recorded by pixel arrays and volumetric grids, whose data is arranged in order, so they can extract the features from inputs by moving filters horizontally and vertically, but the latter’s is unordered.

Although there are some methodologies such as Volumetric CNNs and Multiview CNNs (Qi et al., 2016), to apply CNNs to point cloud data. However, the former technique makes data become unnecessarily voluminous, which requires far more computing resource and time, and the latter is hard to extend to other 3D tasks.

Unordered point sets are consumed by Pointnet (Qi et al., 2017b) directly as inputs. T-net, the spatial transformer, canonicalize the data before fed into network, followed

by MLP, which stand for Multilayer perceptron, extracting feature from each data point. Finally, network aggregates all information from each point and gets global feature by max pooling and then classify the input through fully connected layer.

2.2. PointNet++



C. R. Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," 2017.

Figure 3 PointNet++ Network

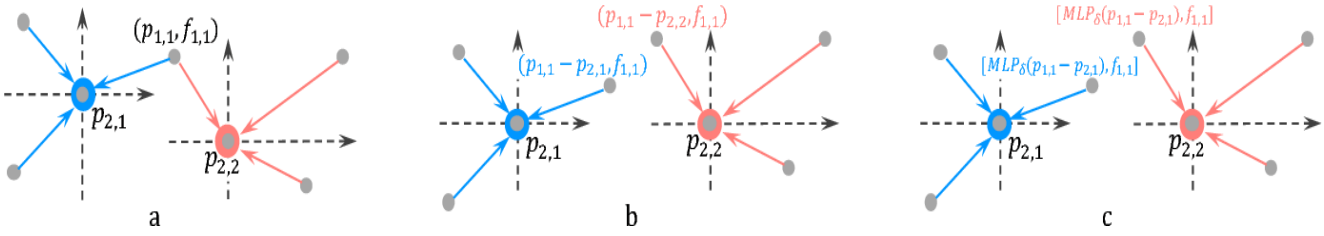
Though PointNet (Qi et al., 2017b) is intuitive and effective, most of the operations are designed for single point and the only function that considers whole points is max pooling, meaning that network considers only the global features but no local one, which is a very important element for Convolutional Neural Network.

Thus, PointNet++ (Qi et al., 2017c) is designed to learn local features by including hierarchical architecture, abstraction levels, which is composed of sampling, grouping, and pointnet layer. Sampling layer selects a subset of input by using farthest point sampling (FPS), and grouping layer chooses the K nearest neighbours of each centroid point that chosen from previous layer. At the end, set abstraction applies mini-PointNet to extract local features and passes them to the next abstraction level.

The concept of PointNet++ is similar to CNN, learning local features at lower level, and outputs the information as input to the next level, until obtaining features of the whole point set.

2.3. PointCNN

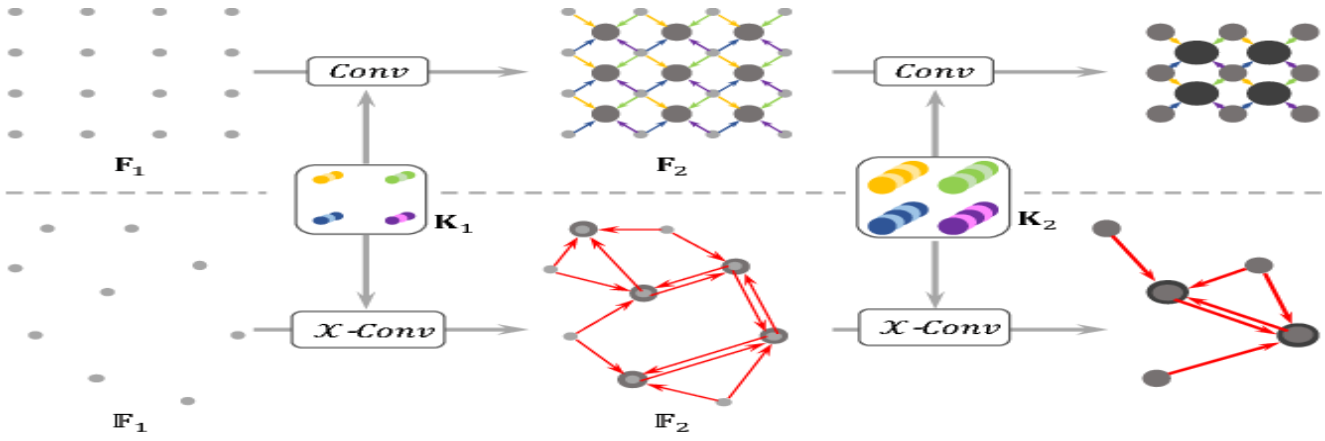
Compare to PointNet++, PointCNN (Li et al., 2018) not only includes the hierarchical structure, but also considers the importance of regularity and order of the data.



[3] Y. Li et al, "PointCNN," 2018.

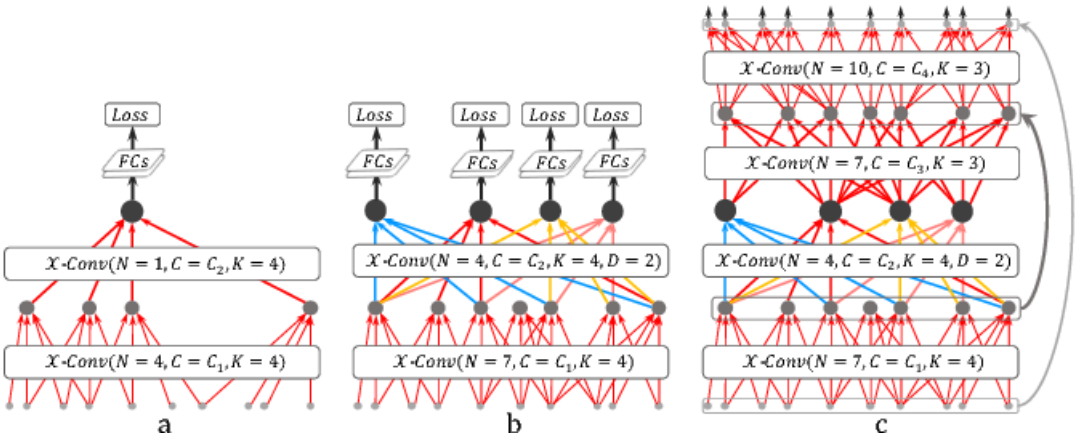
Figure 4 Process for Converting Point Coordinate

X-Conv Operator, the key of PointCNN, first denotes representative points and gets their K neighbours. Since X-Conv extracts features from local regions, PointCNN introduces relative position system, which transforming absolute position into local coordinate to representative point. For example, a point $p_{1,1}$ in figure 4(a), has different relative position to centroid points $p_{2,1}$ and $p_{2,2}$ in figure 4(b).



Y. Li et al, "PointCNN," 2018.

Figure 5 Hierarchical Convolution Structure



Y. Li et al, "PointCNN," 2018.

Figure 6 PointCNN Architecture for Classification (a and b) and Segmentation (c)

After that, X-transformation matrix is learnt to permute the points into latent potentially canonical order, making it able to implement image neural network.

At the end of X-Conv, convolution is applied to and the output will become input of the next X-Conv.

As figure 5 and figure 6 shown, X-Conv aggregates information from neighbours to representative points as Convolution does, making the number of points less and less but each contains more and more information. Besides, the output representative point number, feature dimensionality, and the number of neighbours that each representative point taken into consideration can also be decided, like filter size, and channel in CNN.

3. Implementation

3.1. Introduction of Point Cloud Classification

As general deep learning classification task, this paper takes ModelNet40 (Wu et al., 2015) data as input to train a deep learning network that can perform 3D object classification.

```
chosen_data = chosen_data.reshape((1,chosen_dataset.shape[1],chosen_dataset.shape[2]))
chosen_data_reduce = chosen_data[:,0:NUM_POINT,:]
with tf.Graph().as_default():
    evaluate(chosen_data_eva = chosen_data_reduce, num_votes=1)
```

```
WARNING:tensorflow:Variable += will be deprecated. Use variable.assign_add if you want assignment to the variable value or 'x = x + y' if you want a new python Tensor object.
INFO:tensorflow:Summary name classify loss is illegal; using classify_loss instead.
INFO:tensorflow:Summary name mat loss is illegal; using mat_loss instead.
INFO:tensorflow:Restoring parameters from log/model.ckpt
Model restored.
True label of the input data is airplane
Input data is classified as: airplane
```

Figure 7 Demonstration of Implementation

Figure 7 takes airplane data in Figure 1 as example. Date is fed as an input to the model, and network will classify it as one of the forty categories.

Apart from simple input classification task, point cloud classification technique also works as a crucial element to others task, like the role of image classification in image processing. Frustum PointNet (Qi et al., 2017a) and VoxelNet (Zhou & Tuzel, 2017) are both good illustrations.

Frustum PointNet uses both RGB-D data and point cloud data as input and performs object detection by applying PointNet on 3D data. VoxelNet, a state-of-the-art 3D object detection method developed by Apple, also includes the similar structures as PointNet.

Although 3D data classification can be implemented on many places, this paper focuses on classification itself.

3.2. PointNet

3.2.1. Data Augmentation

Firstly, shuffle the order of data randomly, and 1024 points is chosen uniformly from 2048 points in each object. Then every 32 objects are bound as a batch to be propagated through the network. Finally, the object is randomly rotated and position of each points is jittered by noise.

3.2.2. Network Structure

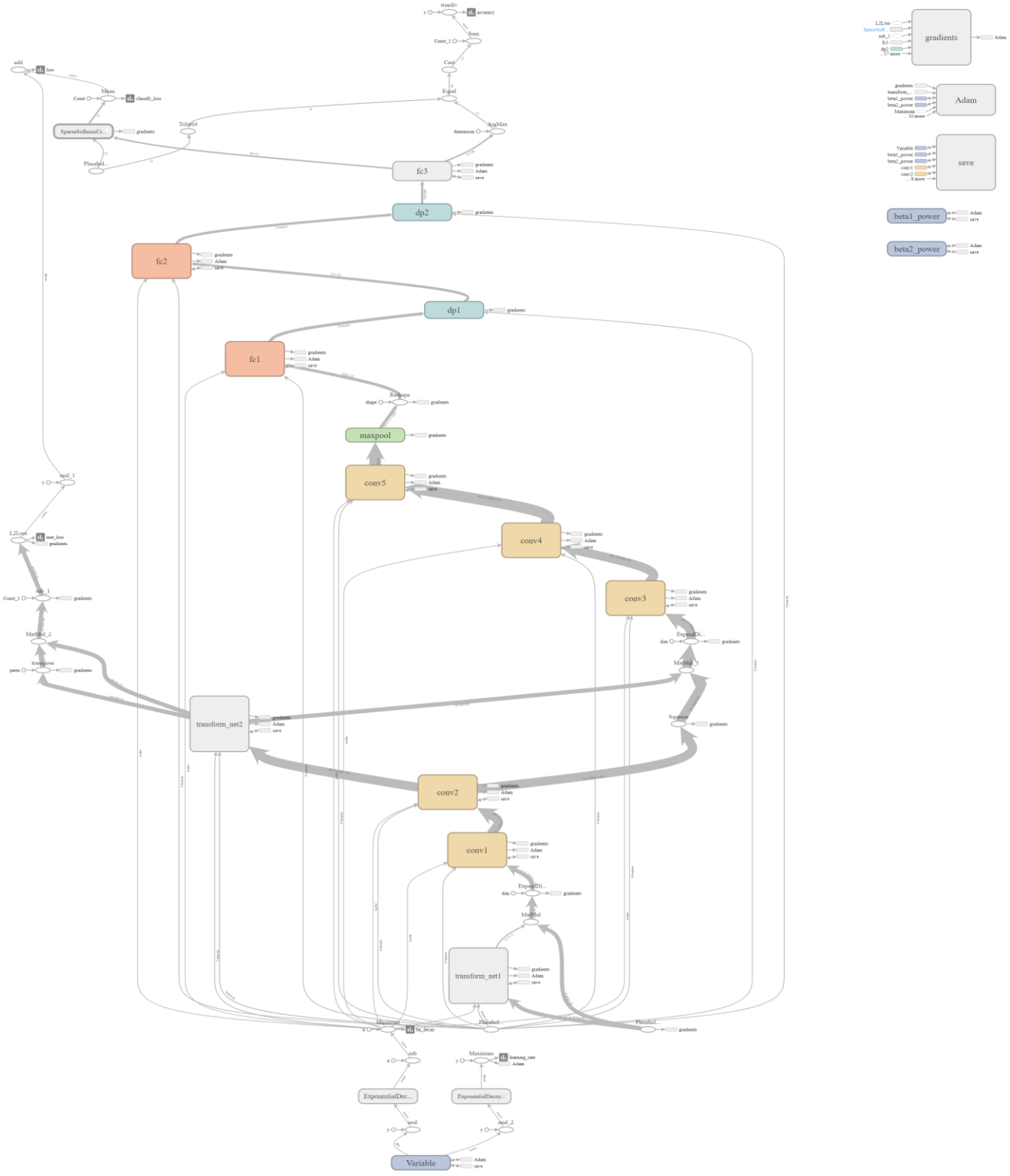


Figure 8 Visualization of the PointNet Network Structure

Figure 8 visualizes the structure of the network by “tensorboard”, where transform_net, conv, fc, and dp, stand for transformer network, convolution layer, fully connected layer, and dropout respectively.

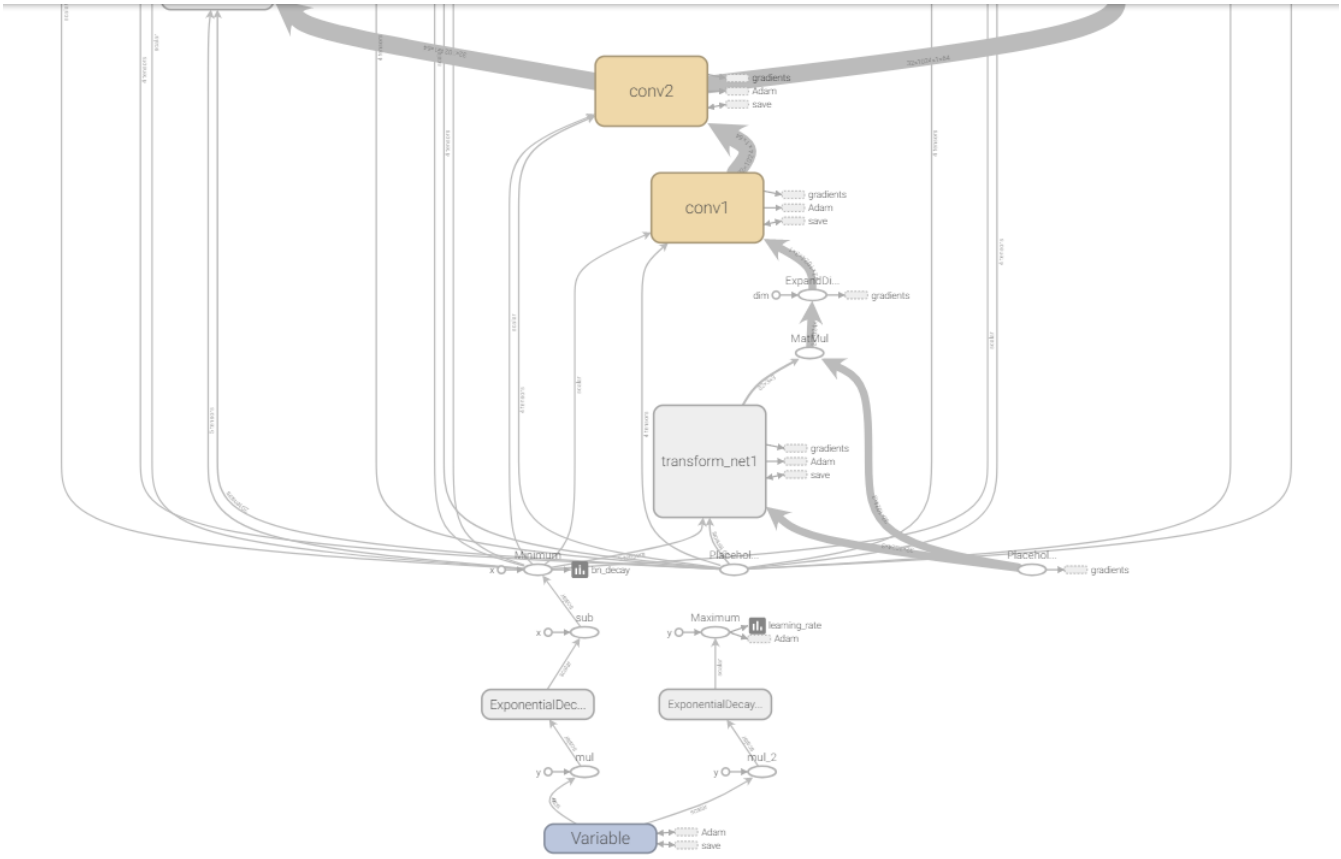


Figure 9 Visualization of the PointNet Network Structure Part 1

Figure 9 shows the first part of the network. Training data multiply with transform matrix, which will be discussed later, and go through two convolution layers. The first layer has 64 kernels with size (1, 3, 1), and there are also 64 filters but with height, width, and depth (1, 1, 64) for the second one.

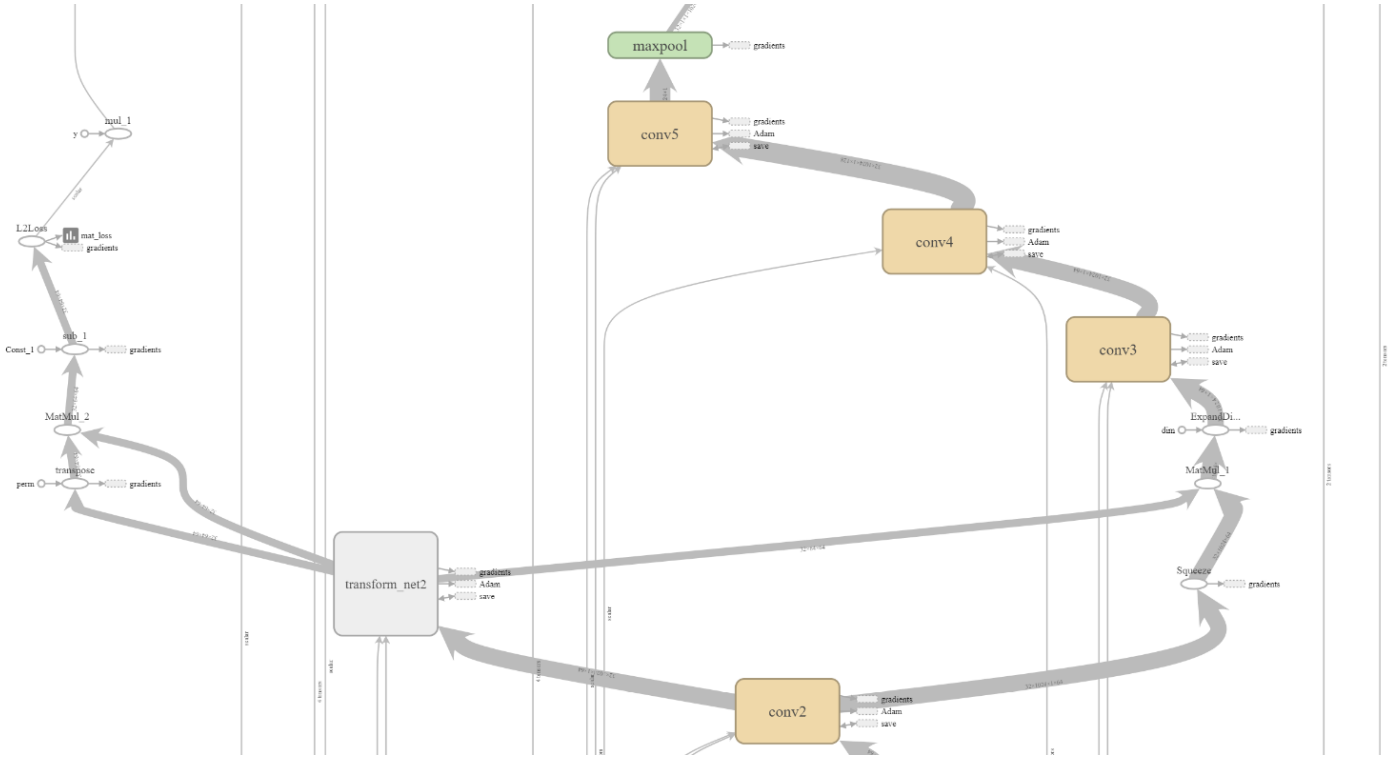


Figure 10 Visualization of the PointNet Network Structure Part 2

After passing two convolution layers, output of conv2 multiplies again with transform matrix. Next, three convolution layers with 64, 128, 1024 kernels respectively (all with height, width (1, 1)), are implemented. Following is a maxpool layer that takes input with dimension (n, 1, 1024), where n is the number of points, and generate global features with dimension 1024.

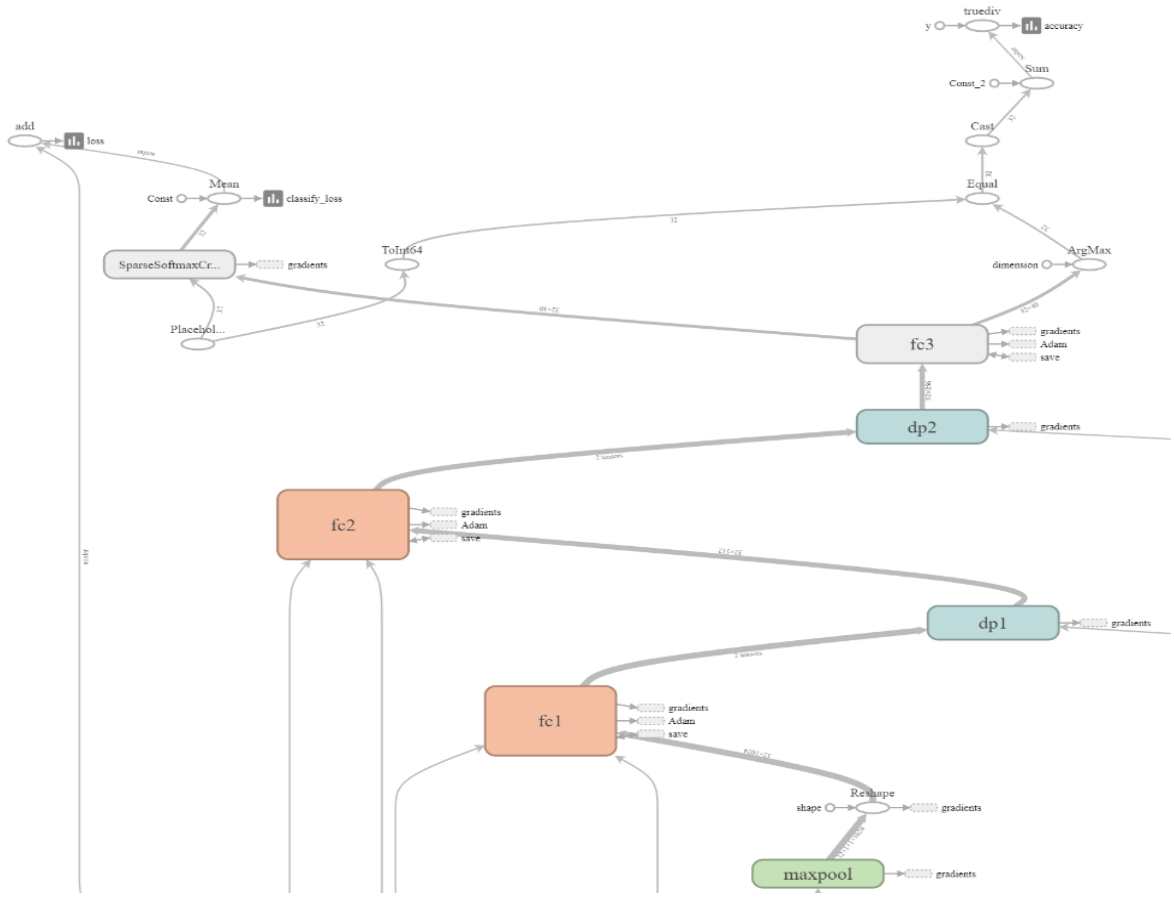


Figure 11 Visualization of the PointNet Network Structure Part 3

Lastly, global features pass through two fully connected layer followed by dropout and one fully connected layer without. The loss to be optimized, is observed by computing sparse softmax cross entropy between logits and labels of 40 categories.

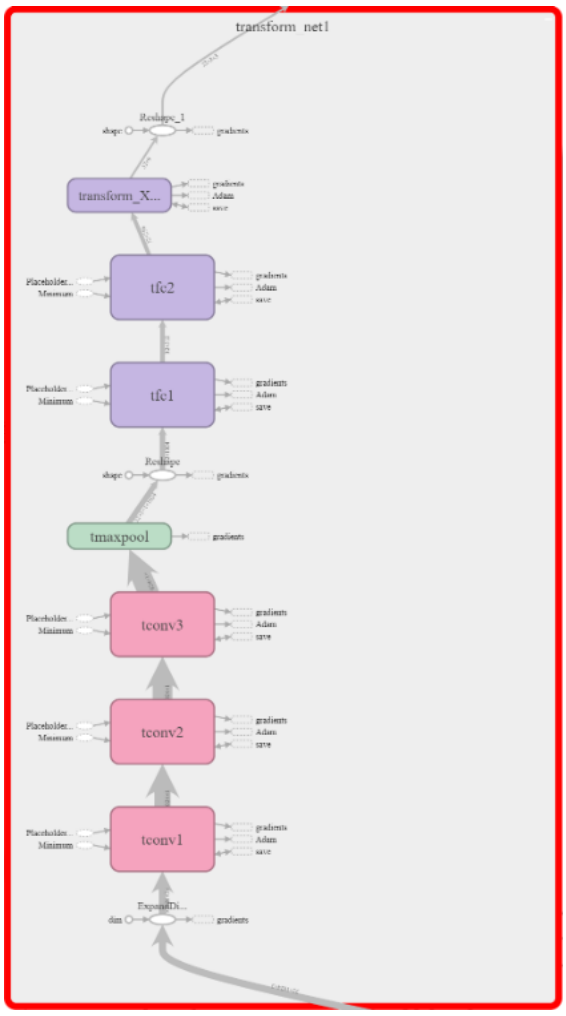


Figure 12 Input Transformer

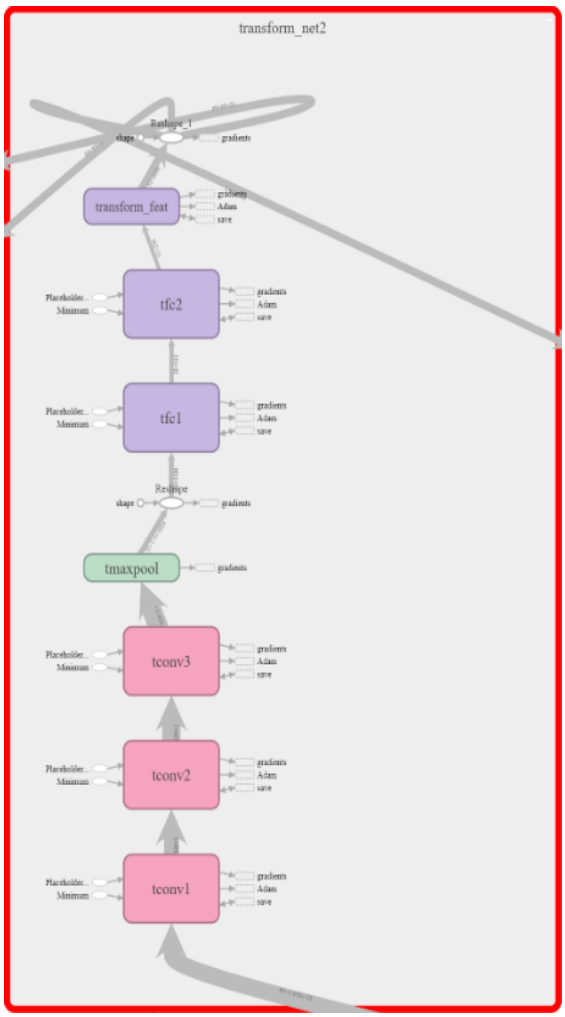


Figure 13 Feature Transformer

In Figure 12, T-Net, transform_net1, learns B (size per batch) 3x3 transform matrix, by 3 convolution layers, with 64 * (1, 3, 1), 128 * (1, 1, 64), 1024 * (1, 1, 128) kernels respectively, followed by one maxpool, and two fully connected layers. At the end of transform_net1, weights with dimension (256, 3x3) are multiplied with the output from fully_connected_layer_2 who has dimension (B, 256), and generates B * (3x3) transform matrix.

Figure 13 shows that the second T-net (B * (64x64) matrix), transform_net2, share the same structures as the first one, but with different size of kernels and matrix it learnt.

3.2.3. Result

The loss, accuracy during training, and parameters that network learnt, are all saved in “log” directory via TensorFlow function “save”.

To see how well is the model learnt, the paper loads in the statistics and plots them by ‘matplotlib.pyplot’ again.

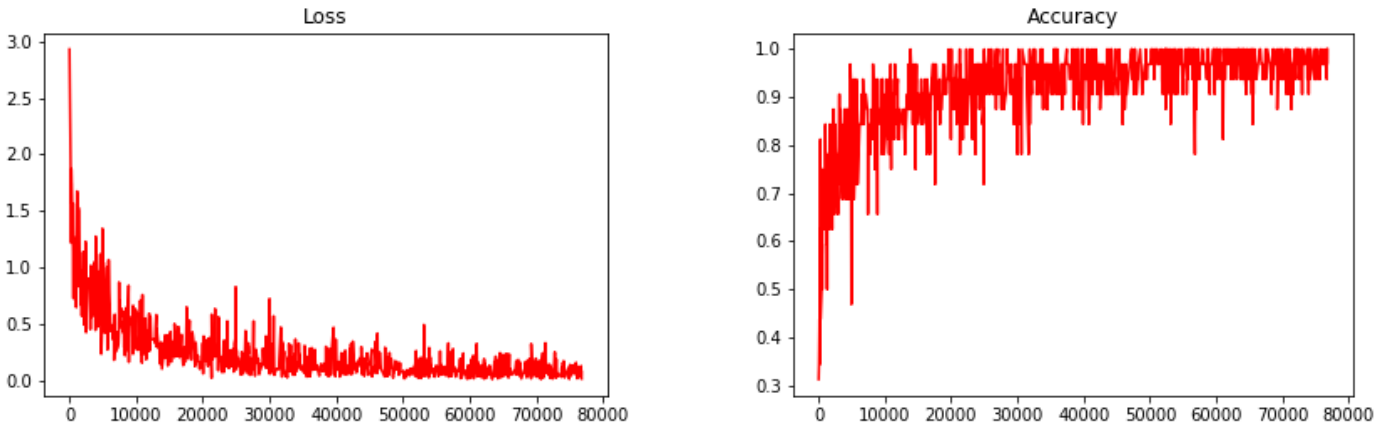


Figure 14 Loss and Accuracy of PointNet during Training

Since 250 is set as the number of epochs, and the training data is split into 308 batches where each contains 32 objects except the last batch, there are about 77000 records. As the Figure 14 shown, the value of loss and accuracy fluctuate around a certain percentage after about 130 epochs.

Apart from viewing the statistics during training, the paper also reads in the model and feeds testing data in. Below is the mean loss, overall accuracy, average class accuracy, and some example of class accuracy. Due to the imbalance number of objects in each category, accuracy and average class accuracy (avg class acc) are slightly different.

```
eval mean loss: 0.514813
eval accuracy: 0.886143
eval avg class acc: 0.860860
  airplane: 1.000
   bathtub: 0.840
       bed: 0.980
     bench: 0.650

       …

      vase:  0.820
  wardrobe: 0.650
      xbox: 0.800
```

Furthermore, wrongly classified point clouds are visualized by rendering them into three-view images. Figure 15, 16 are two of examples.

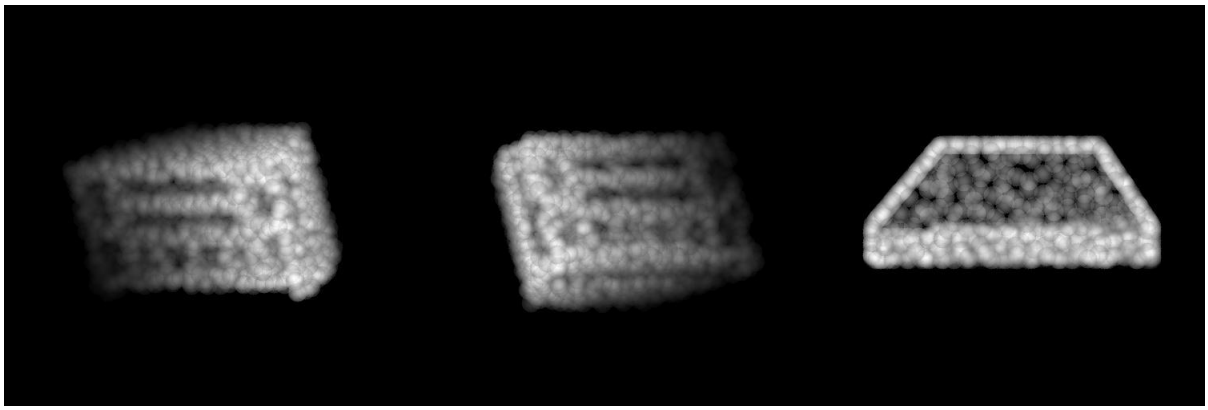


Figure 15 TV-Stand Misclassified as Radio

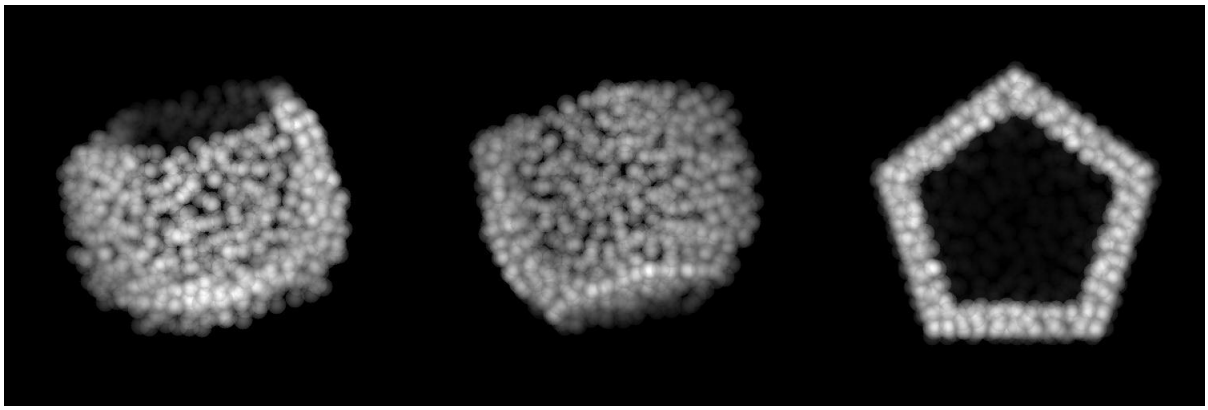


Figure 16 Vase Misclassified as Cup

3.2.4. Robustness Test

Since the point cloud data from real life may not be as perfect as the data such as ModelNet40 (Wu et al., 2015), this paper evaluates the model by adding some random noise data, or structured data that extract from other objects to imitate the situation that may encounter.

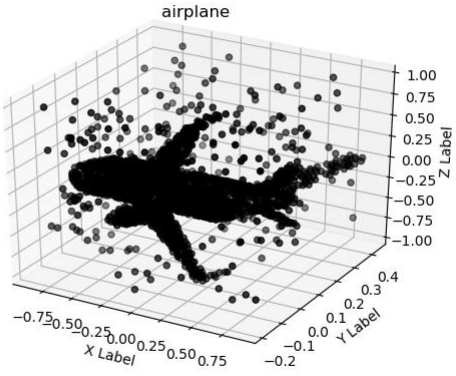


Figure 17 Random Noise Data

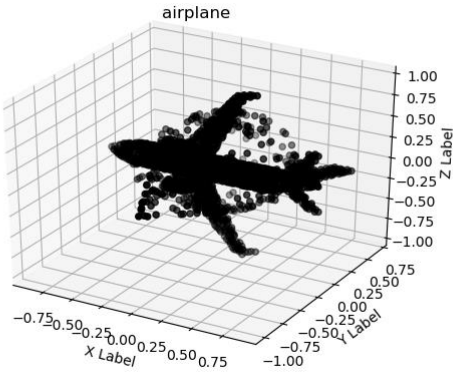


Figure 18 Structured Noise Data

Continue with the first data in Figure 1, this paper visualizes the same point cloud data with random noise, and structured noise as example.

In Figure 17, 15% of data points are replaced by random points generated based on the maximum and minimum value of x-axis, y-axis, and z-axis.

15% of data in Figure 18 is also replaced by point cloud chosen from one of other objects randomly.

```
####### current_data = random_noise_data(current_data, 15) #######
eval mean loss: 2.183289
eval accuracy: 0.570502
eval avg class acc: 0.516442
```

```
####### current_data = structured_noise_data(current_data, 15) #######
eval mean loss: 5.009438
eval accuracy: 0.392220
eval avg class acc: 0.343628
```

Clearly, accuracy of the model drops dramatically no matter which kinds of noise data is added. It is because what PointNet (Qi et al., 2017b) learnt are all global features, when the noise data shows up in testing data that is not seen while training, it cannot classify them very well.

Except from evaluation, noise data is also used in training the model to see how the model performs.

Train with 15% random noise	Test with original data	Test with 15% random noise	Test with 15% structured noise
eval mean loss:	0.914696	0.600877	5.07198
eval accuracy:	0.799433	0.841556	0.410859
eval avg class acc:	0.756023	0.828860	0.366407

Table 1 Statistics of PointNet Trained with 15% Noise Data

Train with 15% Structured noise	Test with original data	Test with 15% random noise	Test with 15% structured noise
eval mean loss:	0.648329	0.752531	0.700067
eval accuracy:	0.814019	0.786872	0.787682
eval avg class acc:	0.761593	0.719424	0.729477

Table 2 Statistics of PointNet Trained with 15% Structured Data

Table 1 and 2 show the statistics of PointNet trained with different kinds of data. Interestingly, model trained with 15% random noise and tested with 15% random noise still gets a high accuracy that is far better than the model trained with no noise data. The same situation can also be observed in PointNet trained with 15% structured noise, with slightly lower accuracy.

As for testing the model with original data, both of the models that trained with noise cannot classify as well as the one trained without noise.

Lastly, the paper tests the robustness to missing value by evaluating model with data only about one eighth points remain.

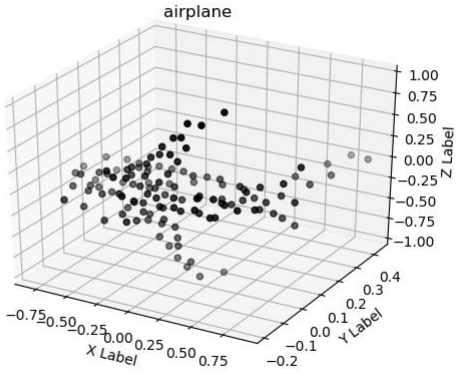


Figure 19 Point Cloud with 128 Points Remain

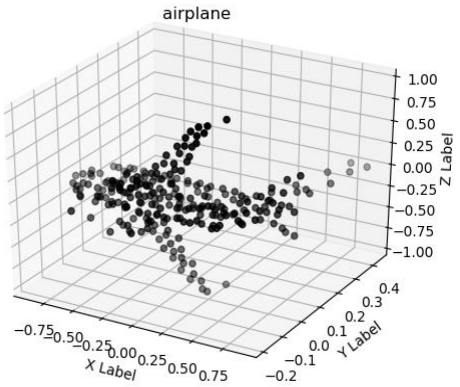


Figure 20 Point Cloud with 256 Points Remain

Figure 19, 20 are illustrations of airplane data shown in Figure 1 with only 128, and 256 points remain.

```
#### 128 ####
eval mean loss: 1.379814
eval accuracy: 0.702593
eval avg class acc: 0.708547

#### 256 ####
eval mean loss: 0.669553
eval accuracy: 0.834279
eval avg class acc: 0.818674
```

Above are statistics while model is evaluated with 128, 256 points. Even with only one eighth points, accuracy still can reach 70%, and with one fourth, accuracy only drops by 5%.

3.3. PointCNN

3.3.1. Data Augmentation

Same as what is done in PointNet (Qi et al., 2017b), point cloud are shuffled and jittered randomly. However, 128 is used as batch size in PointCNN (Li et al., 2018).

Feedable iterator technique is used to split the data into batch and passes them when they are needed.

3.3.2. X-Conv Operator

The steps of X-Conv Operator are must-know knowledge before viewing the whole structure of PointCNN ; hence, the paper briefly introduces them here.

Input : $\mathbf{K}, p, \mathbf{P}, \mathbf{F}$
Output: $\mathbf{F}_p$ ▷ Features "projected", or "aggregated", to 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 dim. 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 $\mathbf{F}_\mathcal{X}$

[3] Y. Li et al, "PointCNN," 2018.

Algorithm 1 X-Conv Operator

Input:

K: number of neighbours

p: representative point

P: absolute position

F: Feature

Output:

Fp: output feature of X-Conv, also the input F for the next X-Conv

As CNNs (Krizhevsky et al. 2012), X-Conv is applied on local regions of point cloud and get a higher-level representation points containing more information. Because output features and representative points should be associated, neighbours of these points and their features are also taken as part of input.

At step 1, for each representative point p, K nearest neighbourhood points are chosen and transformed into relative coordinates. Then each point recorded in relative position is lifted individually into C_δ dimensions, just like MLP extracts features from single point in PointNet (Qi et al., 2017b). Step 3 concatenates features from step 2, and features from last X-Conv, where only F_δ are used during the first X-Conv layer since F is empty. At the next step, K x K matrix is learnt by applying MLP on entire neighbours' coordinates, then the matrix weights and permutes F* into latent potentially canonical order, making it suitable for applying convolution K (K × (C1 + C_δ) × C2 tensor).

In paper, PointCNN (Li et al. 2018), algorithm 1 is summarized more concisely as,

$$F_p = X{-}Conv(K,p, P, F) = Conv(K, MLP(P - p) \times [MLP_\delta (P - p), F])$$

3.3.3. Network Structure

Since the structure of the PointCNN (Li et al., 2018) network is stacked up by several X-Conv Operator, it becomes too massive to be shown in one page. Hence, this paper plots a simplified structure and introduces the detail of them in the following paper.

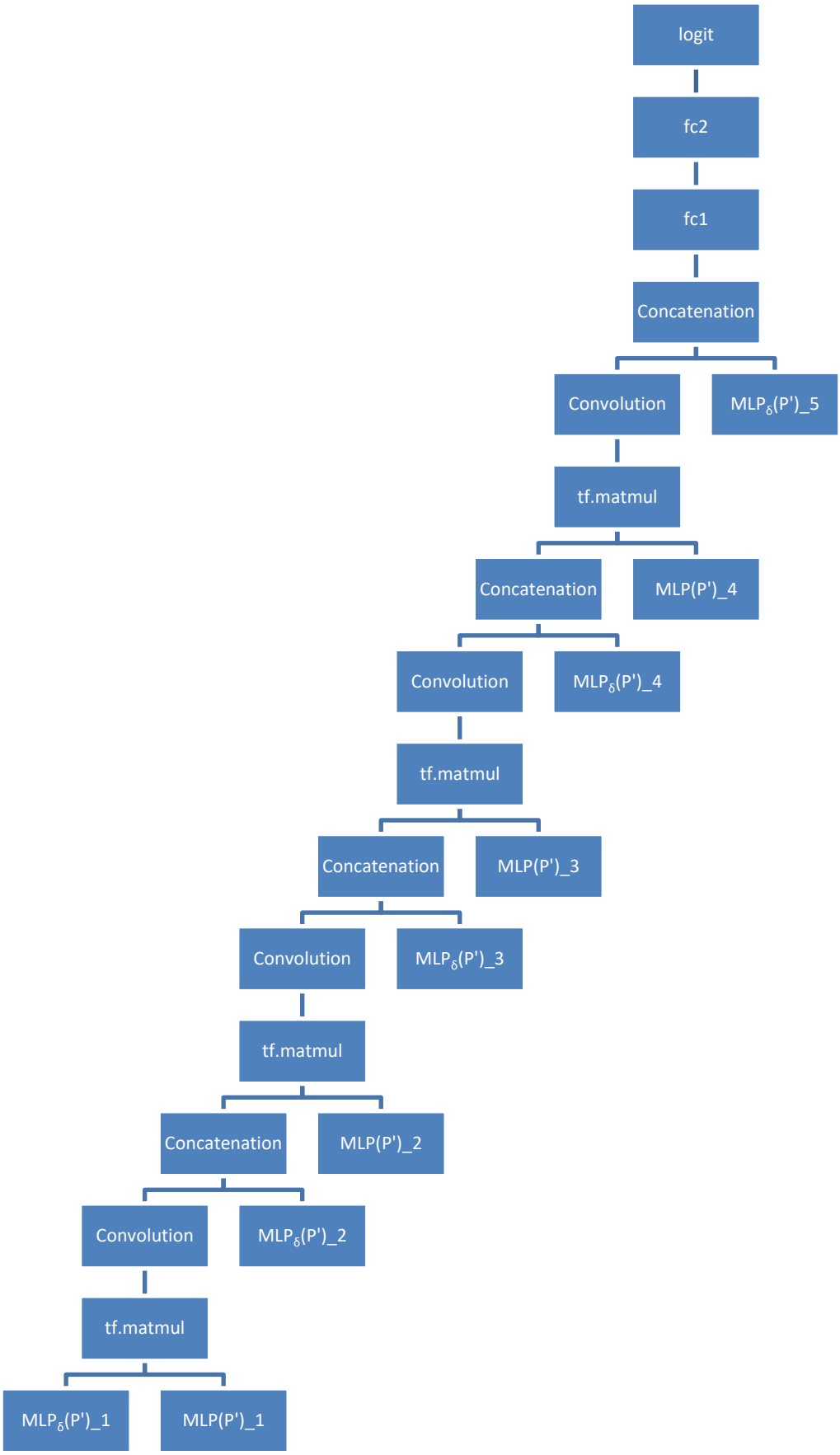


Figure 21 Visualization of the PointCNN Network Structure

In Figure 21, MLP$_\delta$ (P') and MLP(P') stand for step 2 and 4 in algorithm 1. "tf.matmul" is an operation that multiplies two matrixes and "concatenation" concatenates two matrixes.

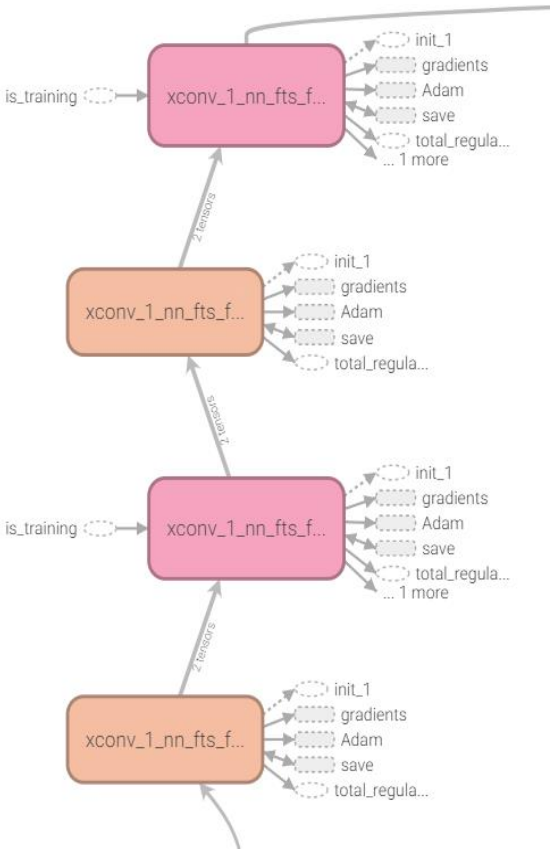


Figure 22 Features Extraction from Individual Point

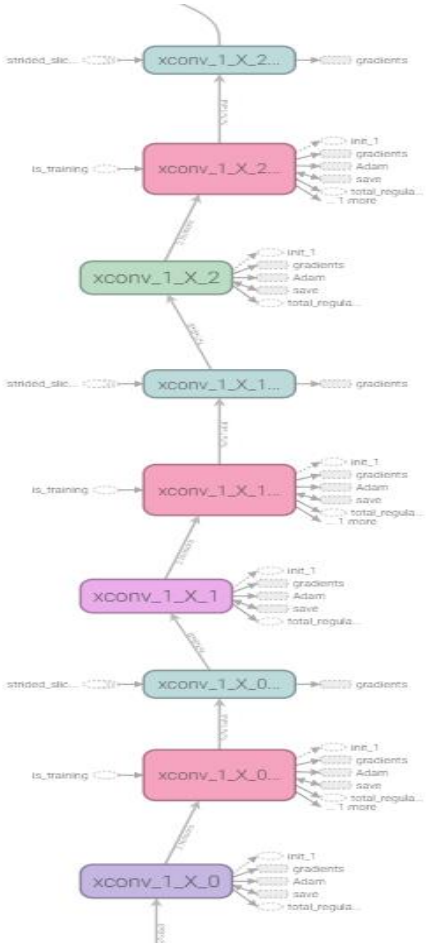


Figure 23 Process for Learning X-transformation Matrix

Structure of MLP$_\delta$ (P') is shown in Figure 22. It is composed of two densely-connected layers (fully connected layer) both followed by Batch Normalization (Ioffe et al. 2015), which normalizes each training mini-batch, and speeds up next layer's training. Furthermore, it also has the function of regularizer.

There are five MLP$_\delta$ (P') in PointCNN network, marked as MLP$_\delta$ (P')_1 to MLP$_\delta$ (P')_5 in Figure 21, where the last one works as the point-wise global features extraction in PointNet (Qi et al., 2017b).

During the first layer in Figure 23, K * K kernels with size (1, K, 3) are learnt to apply convolution on input data with dimension (N, P, K, 3), where N stands for the size of batch, P is the number of representative points, and K is the number of neighbours; hence, the output of convolution layer has dimension (N, P, 1, K * K). Next, the output goes through Batch Normalization and then is reshaped into (N, P, K, K).

Depthwise convolution layer (Chollet, 2017), the second layer, applies (1, K) kernel separately on every single channel of input, which can reduce the parameters be trained dramatically. Following are Batch Normalization and operation that reshapes it into (N, P, K, K) again.

Finally, the last layer shares the same structure as the second one, and output N * P (K x K) matrixes. That is, for each object in the batch with size N, P matrixes with size (K x K) are trained.

At the next step, the features extracted in Figure 22 are multiplied by matrixes learnt in Figure 23 with in operation “tf.matmul” to permute the points into latent potentially canonical order. Then depthwise separable convolution layer, the combination of depthwise convolution layer and pointwise convolution (also known as 1 x 1 convolution that mixes channels get from depthwise), extracts the features and pasts it to next X-Conv Operator.

The operations above are called as X-Conv Operator, and the output will be concatenated with MLP_{δ} (P')_2 and multiplied by X-transformation Matrix again.

PointCNN stacks four X-Conv to learn features hierarchically and concatenates the output with MLP_{δ}(P')_5 to include the features learnt directly from original point cloud. Sparse softmax cross entropy are also used to optimize in PointCNN.

3.3.4. Result

Since the number of objects, batch size, and the number of epochs is 9840, 128, and 1024 respectively, the model is trained 9840 / 128 * 1024 = 78848 times each with a batch of objects. Evaluation is executed every 500 steps, so there are 158 records.

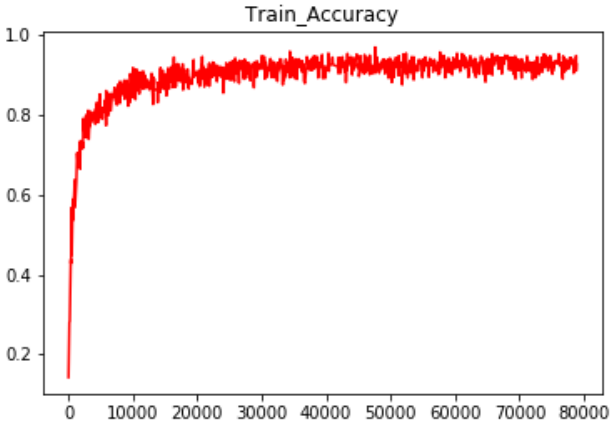


Figure 24 Accuracy of PointCNN during Training

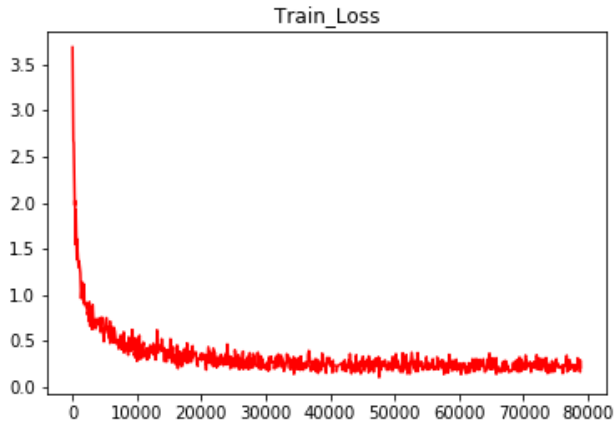


Figure 25 Loss of PointCNN during Training

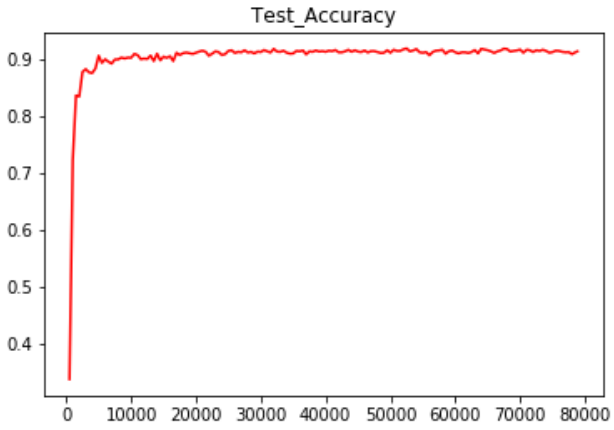


Figure 26 Evaluation of PointCNN Classification Accuracy

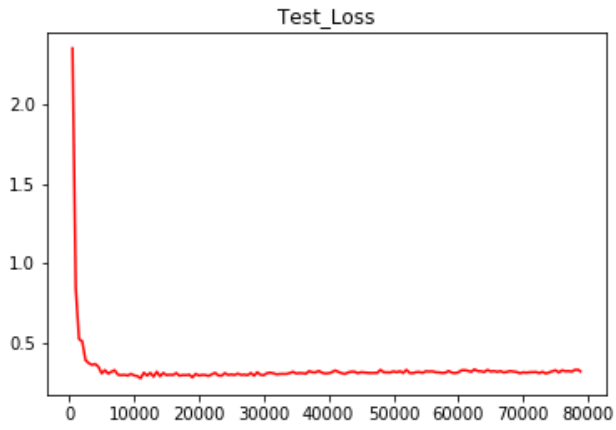


Figure 27 Evaluation of PointCNN Classification Loss

As the information shown in Figure 24 to Figure 27, accuracy and loss all converge after around 35,000, which is about 450 epochs. Except that, accuracy in training and evaluation both reach higher than 90%, 2% better than that in PointNet (Qi et al., 2017b).

3.3.5. Attempts with Different Network Structure

Structure 1:

As X-transformation matrixes transform the data into potentially canonical order that can apply image neural network techniques to, this paper replaces the part of Convolution (separable convolution) in Figure 21 with different structures. Here the paper compares all the models trained with 500 epochs, where the accuracy of the original structure is 91.53%, and 87.47% for average class accuracy.

Structure 2:

In the first place, this paper replaces depthwise separable convolution layer with simply convolution layer. The accuracy and average class accuracy keep at similar levels, 91.69%, and 87.42%; however, the number of parameters to be trained increases substantially from 599,340 to 2,261,340, almost four times more than using separable convolution layer.

Structure 3:

Next, refers to Inception Model (Szegedy et al. 2015) includes 1 x 1 before applying convolution to reduce the parameter to train, the structure is also employed here to speed up training time. The quantity of parameters here are reduced to 549084, and accuracy becomes 91.25%.

Structure 4:

The advantages of Inception Model are not only using 1 x 1 convolution, but also concatenates convolution layers with different kernel size to learn feature with different views. Hence, kernels with height and width (1, K), (1, 0.75 * K), (1, 0.5 * K) are applied, and the last two are followed by max-pool to gather information. 91.05%

is observed as accuracy of this structure with 505764 parameters to be trained, 15% less than the original one.

Structure 5:

Finally, concept of The All Convolution Net (Springenberg et al. 2014), which replaces the max-pool in CNNs, is also used in this paper. The max-pool after convolution layers are replaced with convolution layers which have kernel with height and width (1, 0.25 * K + 1), (1, 0.5 * K + 1) respectively, and accuracy becomes 92.03%.

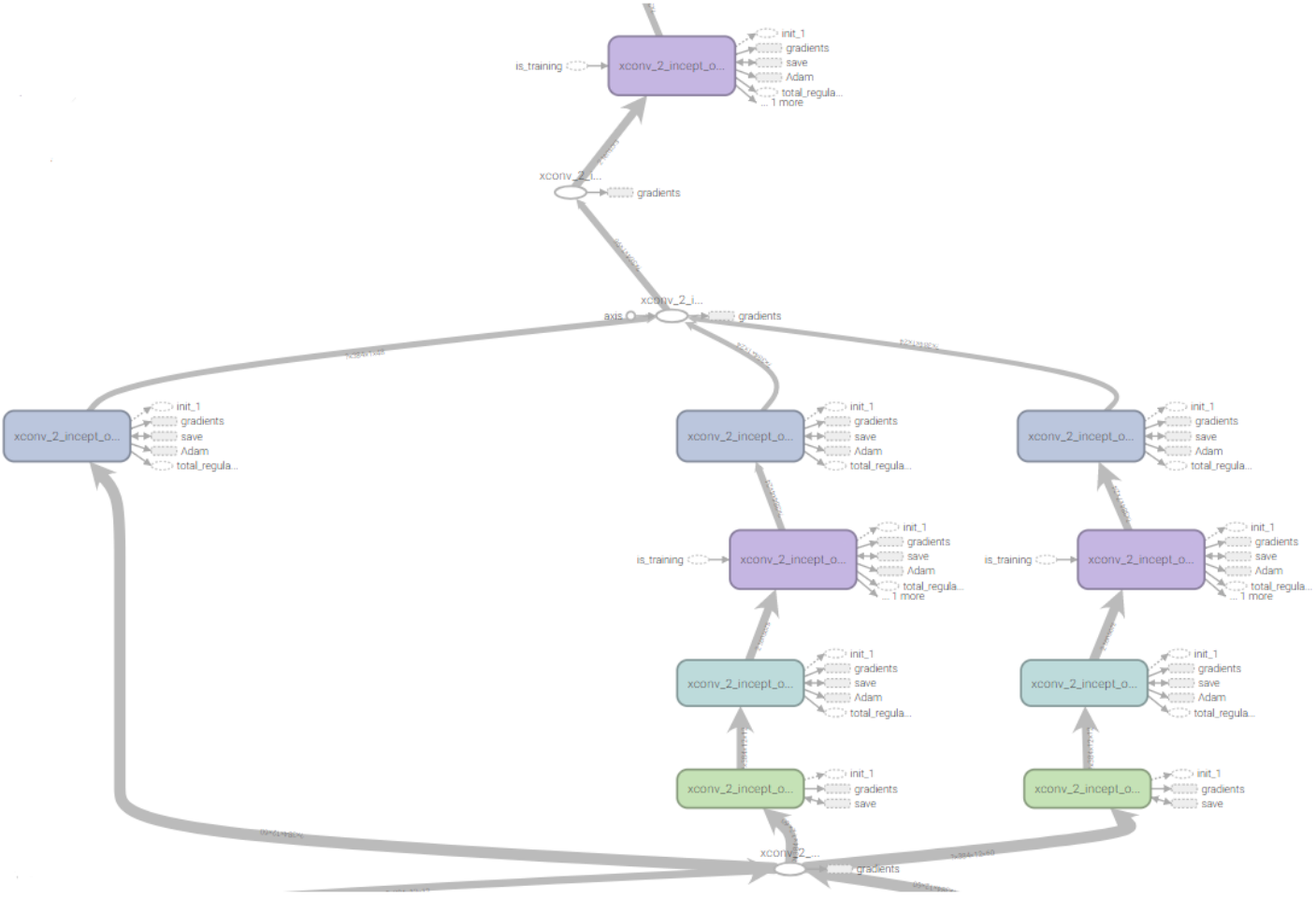


Figure 28 Inception Structure in PointCNN

Figure 28 shows the structure of Inception Model that replaces depthwise separable convolution layer in original PointCNN (Li et al., 2018) network.

On the left side of the figure is depthwise separable convolution layer with kernel height and width (1, K). The structures in the middle and right side of figure are 1 x 1 convolution layer followed by depthwise separable convolution, Batch Normalization, and one more depthwise separable convolution. The results from the three layers are then concatenated together before going through Batch Normalization (Ioffe et al. 2015) and pass to the next X-Conv Operator.

At the end of this section, this paper makes a table to compare each structure with the number of parameters to be train and accuracy.

	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Number of Parameters	599,340	2,261,340	549084	505764	560244
Accuracy	91.53	91.69	91.25	91.05	92.03
Avg. Class Accuracy	87.47	87.42	87.03	86.95	87.84

Table 3 Comparison between Different PointCNN Structures

3.3.6. Robustness Test

With 15% points replaced by random noise, PointCNN (Li et al., 2018) with inception structure still maintains 89.59% accuracy. If the model is trained with data with random noise, accuracy can even reach 90.88%.

Besides, network also performs very well with 87.03% accuracy on 15% structured noise data.

As for missing point, PointCNN still can classify 87.12% objects with only 128 point accurately.

4. Conclusion

Here the paper compares PointNet (Qi et al., 2017b) with PointCNN (Inception Model Structure).

	Original Data	15% Random Noise Data	15% Structured Noise Data	Missing Point (128 points)
PointNet	88.61	57.05	39.22	70.26
PointCNN	92.03	89.59	87.03	87.12

Table 4 Comparison between PointNet and PointCNN

As shown in table 4, it is obvious that PointCNN performs better than PointNet on no matter what kinds of data is used for testing.

Apart from normal models, here also compares models trained with 15% random noise data.

	Original Data	15% Random Noise Data
PointNet	79.94	84.16
PointCNN	91.07	90.88

Table 5 Comparison between PointNet and PointCNN Trained with 15% Random Noise

Not surprisingly, PointCNN outperforms PointNet on both testing data.

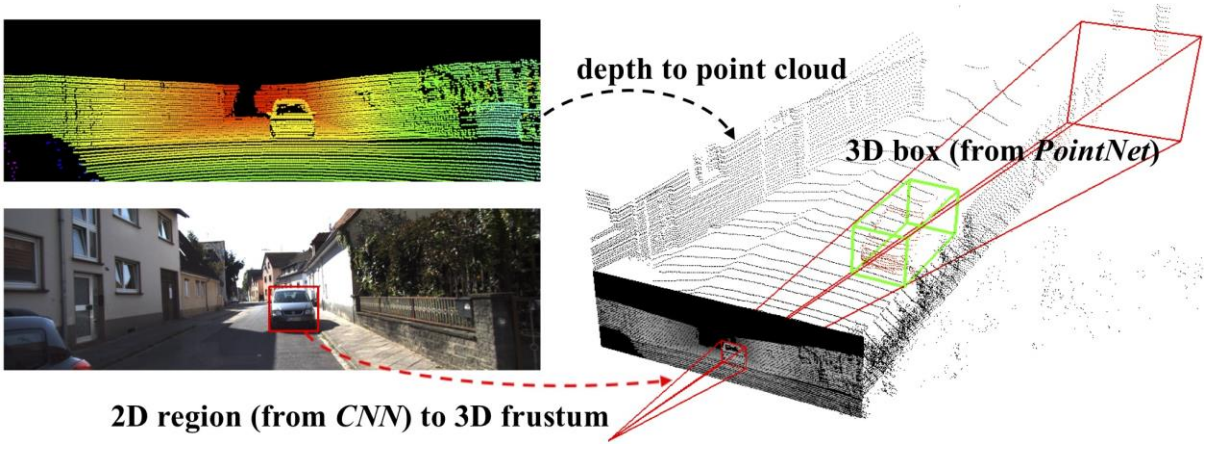
According to the statistics shown above, though PointCNN has a longer training time than PointNet, its accuracy and robustness are much better than those of PointNet.

As for structure of PointCNN, including Inception Model Layer can improve accuracy slightly and reduce the number of parameters be trained. Hence, this paper believes that it is a better choice than others for 3D Point Cloud Classification tasks.

5. Future Works

3D Point Cloud Classification as mention above, is indispensable part of object detection and object recognition, which are very practical technique for autonomous vehicle, agricultural application system, river survey, etc.

Therefore, the further step of this paper is to train PointCNN with different point cloud data and apply it on object detection task as what Frustum PointNet (Qi et al., 2017a) shown in Figure 29 does.



C. R. Qi et al, "Frustum PointNets for 3D Object Detection from RGB-D Data," 2017.

Figure 29 Frustum PointNets 3D Object Detection Pipeline

6. References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., (2016), November. Tensorflow: a system for large-scale machine learning. *In OSDI (Vol. 16, pp. 265-283).*

Chollet, F., (2017). Xception: Deep learning with depthwise separable convolutions. *arXiv preprint, pp.1610-02357.*

GitHub. (2018). charlesq34/pointnet. [online] Available at: https://github.com/charlesq34/pointnet [Accessed 22 Aug. 2018].

GitHub. (2018). yangyanli/PointCNN. [online] Available at: https://github.com/yangyanli/PointCNN [Accessed 22 Aug. 2018].

Ioffe, S. and Szegedy, C., (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167.*

Krizhevsky, A., Sutskever, I. and Hinton, G.E., (2012). Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems (pp. 1097-1105).*

LeCun, Y., Bengio, Y. and Hinton, G., (2015). Deep learning. *nature, 521(7553), p.436.*

Li, Y., Bu, R., Sun, M. and Chen, B., (2018). PointCNN. *arXiv preprint arXiv:1801.07791.*

Qi, C.R., Su, H., Nießner, M., Dai, A., Yan, M. and Guibas, L.J., (2016). Volumetric and multi-view cnns for object classification on 3d data. *In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5648-5656).*

Qi, C.R., Liu, W., Wu, C., Su, H. and Guibas, L.J., (2017a). Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488.*

Qi, C.R., Su, H., Mo, K. and Guibas, L.J., (2017b). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 1(2), p.4.*

Qi, C.R., Yi, L., Su, H. and Guibas, L.J., (2017c). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *In Advances in Neural Information Processing Systems(pp. 5099-5108).*

Reutebuch, S.E., Andersen, H.E. and McGaughey, R.J., (2005). Light detection and ranging (LIDAR): an emerging tool for multiple resource inventory. *Journal of Forestry, 103(6), pp.286-292.*

Springenberg, J.T., Dosovitskiy, A., Brox, T. and Riedmiller, M., (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806.*

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., (2015). Going deeper with convolutions. *In Proceedings of the IEEE conference on computer vision and pattern recognition(pp. 1-9).*

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X. and Xiao, J., (2015). 3d shapenets: A deep representation for volumetric shapes. *In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1912-1920).*

Zhou, Y. and Tuzel, O., (2017). Voxelnet: End-to-end learning for point cloud based 3d object detection. *arXiv preprint arXiv:1711.06396.*