PointNet For Point Cloud Classification

Weijia Chen weijiac
Zhaohan Lu zhaohan
Yuping Wang ypw
Xuran Zhao xuranzh

Abstract

Point cloud classification is a special problem in computer vision. Unlike 2D images, point clouds consist of unordered and scattered 3D points. In this work, we implement a Neural Network based point cloud classifier, PointNet [3], which directly takes in point cloud as input and produce a label as output. Experiments are proposed and implemented on the model and input data, such as the removal of spatial transform network and the inclusion of the farthest point sample layer. The model is trained and tested with resampled ModelNet40 [8] dataset with and without normal vectors. The performance of the models is compared and analyzed for a better understanding of the approach and finding the direction of future improvements.

1. Introduction

Robots for localization and navigation tasks nowadays, such as autonomous vehicles, are typically equipped with both LIDAR and RGB cameras. While the RGB camera provides abundant information about the scene and the 2D image pipelines are quite mature now, it is heavily affected by lighting. In contrast, information from LIDAR is invariant during the daytime or at night. Therefore, utilizing point clouds from LIDAR for object detection and classification would be more effective under different lighting conditions. In this project, we implement and analyze PointNet [4], a neural network model that can effectively extract the features from the irregular-format point clouds and classify them accordingly. Before this work, typical convolutional solutions for classification in 3D space is to voxelize the space, then apply convolution in a similar manner as for 2D images. However, this approach is constrained by the resolution issue rising from the discontinuity of the voxel representation. Furthermore, applying convolution on point clouds results in high computation cost, since the point cloud obtained from LIDAR usually only distribute along the convex hulls of the objects.

The problem when processing point clouds directly is the unorderness of how the points are stored. The key idea in PointNet is to process independent raw points $P \in (x, y, z)$ through multi-layer perceptrons (MLP) and affine transformations, and then applying a symmetric function, max pooling, on the high dimensional features. The networks output classification labels in the end.

The goal of our work is to implement the PointNet for classification tasks and analyze its performance with modified parameters and structures. Starting from a default network, we experiment on how different network configurations affect the classification results. We compare the network's performance when (1) feeding the network randomly sampled points or Farthest sampled points (FPS), (2) using with and without the Spatial Transformation Networks (STN), (3) training on just point coordinates and training alongside the normal from mesh vectors, (4) training on data with or without random noise. The critical subset of points that the network learns is also visualized.

2. Related Work

Several attempts have been made for object classification on 3D data. Before trying to tackle this problem with CNN, the traditional method for 3D data classification is to extract some hand-crafted features from the point cloud or mesh and then apply off-the-shelf classifiers such as SVM or Softmax.

In 2.5D CNNs [5], RGBD data is used, with depth channel information treated as an additional channel. Though straightforward, this approach does not fully utilize the geometric information.

In multi-view CNNs [6], several viewpoints are set up to capture the rendered 2D views of the objects, which are then inputted into a well-engineered VGG-M network for classification.

In volumetric CNNs such as Voxnet [2] and 3D ShapeNets [8], the 3D objects are converted to fixed-sized volumetric occupancy grid representation, which is used as the input to the CNN. In Voxnet, several rotated copies are created for both training and testing to concur with the issue of rotational variance. In 2016, a CNN combining multi-view approach and volumetric approach is developed [4]. However, one deficiency in such an approach is the constraint of the resolution caused by the occupancy grid representation.

There have also been approaches exploring the CNN paradigm to manifolds [1]. However, such methods would not work on non-manifold artificial objects such as furniture.

Recently, EdgeConv [7] is proposed for point cloud classification and segmentation tasks. Instead of using the points directly, it utilizes edge features for obtaining translation-invariant properties.

3. Method

Point clouds are essentially an unordered data set, which means that the ordering of points does not affect its representation of the overall shape in geometry. One of the challenges is same feature should be extracted regardless of the ordering of points input. Correspondingly, the symmetric function, max pooling, is utilized after obtaining the features in latent dimension to remove the influence from the unorderness.

Another challenge when dealing with the point clouds is that the model may undergo some rigid transformation so that the data is not suitable for us to take it directly as input. The transformation layer composed of a spacial transform network before the multi-layer perceptron can transform the input into the desired form of the next layer. The first transformation layer aims align the object in the Cartesian coordinates for classification or segmentation, while the second one aims to align the extracted 64-dimensional features from the last layer. As a result, PointNet is rotation-invariant.

We implemented our model following the descriptions in the original paper. The general structure can be seen from Figure 1 (assume a batch size of 4 and 50 points are selected from each point cloud). From left to right, each layer, with respect to one training sample, is as follows (assume we select n points for each point cloud and we have 40 classes/categories):

- Input layer with size n x 3, meaning n points and each of them has x, y, z coordinates
- Spatial Transformer Network of input size 3 (Note this network is omitted in the figure because this transformation network is optional and open to different implementation)
  - 1D Convolution to expand the number of features from 3 to 64, with Batch Norm and Relu
  - 1D Convolution to expand the number of features from 64 to 128, with Batch Norm and Relu
  - 1D Convolution to expand the number of features from 128 to 1024, with Batch Norm and Relu
  - Max Pooling
  - Linear Transformation to reduce the number of features from 1024 to 512, with Batch Norm and Relu
  - Linear Transformation to reduce the number of features from 512 to 256, with Batch Norm and Relu
  - Linear Transformation to reduce the number of features from 256 to 9
  - The resulting vector is then reshaped into a 3 x 3 matrix to represent a rigid rotation
  - The transformation matrix is then applied to each point
- 1D Convolution to expand the number of features from 3 to 64, with Batch Norm
- Spatial Transformer Network of input size 64 (Note this network is omitted in the figure because this transformation network is optional and open to different implementation)
  - 1D Convolution of size from 64 to 64, with Batch Norm and Relu
  - 1D Convolution to expand the number of features from 64 to 128, with Batch Norm and Relu
  - 1D Convolution to expand the number of features from 128 to 1024, with Batch Norm and Relu
  - Max Pooling
  - Linear Transformation to reduce the number of features from 1024 to 512, with Batch Norm and Relu
  - Linear Transformation to reduce the number of features from 512 to 256, with Batch Norm and Relu
  - Linear Transformation to reduce the number of features from 256 to 64
  - The resulting vector is then reshaped into a 64 x 64 matrix to represent a rigid rotation
  - The transformation matrix is then applied to each point
- 1D Convolution to expand the number of features from 64 to 64, with Batch Norm and Relu
- 1D Convolution to expand the number of features from 64 to 128, with Batch Norm and Relu
- 1D Convolution to expand the number of features from 128 to 1024, with Batch Norm and Relu
- Linear Transformation to reduce the number of features from 1024 to 512, with Batch Norm and Relu

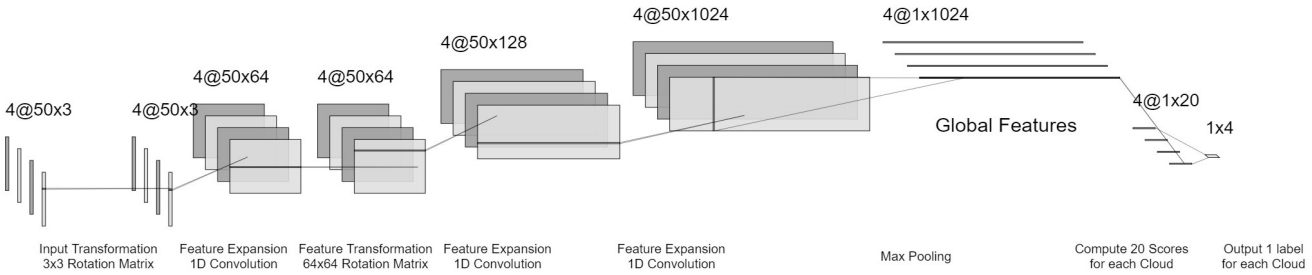


Figure 1: This is the general structure.

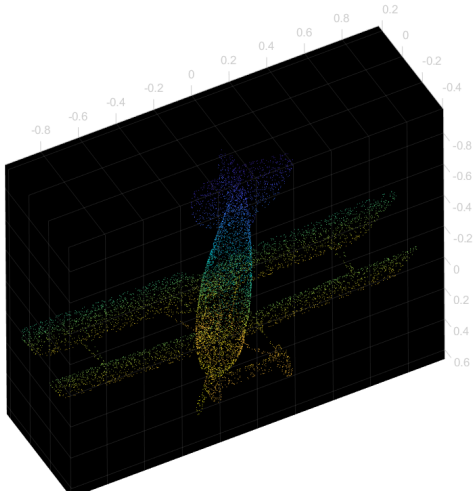


Figure 2: Point Cloud for an airplane

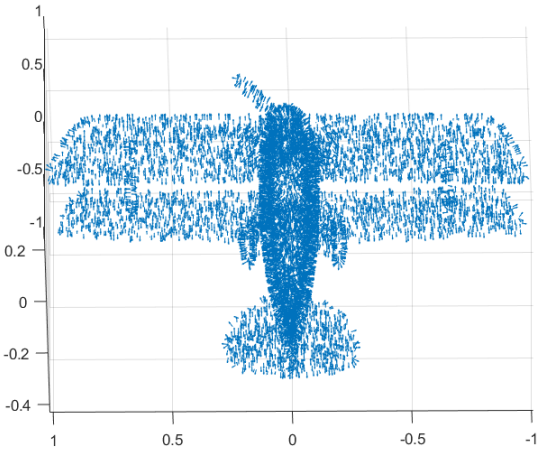


Figure 3: Point Cloud for an airplane with normal vectors

- Linear Transformation to reduce the number of features from 512 to 256
- Dropout with ratio 0.4
- Batch Norm and Relu
- Linear Transformation to reduce the number of features from 256 to 40
- Softmax
- Extract class label by finding the index corresponding to the max value

We compute softmax classification loss between the prediction and the ground truth.

4. Experiments

4.1. Dataset

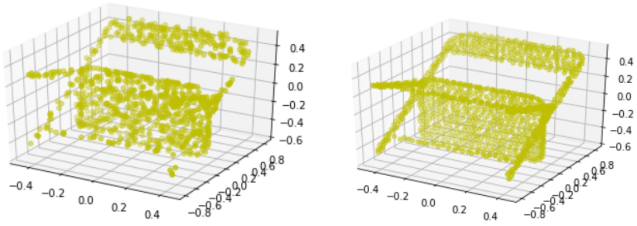
We use the randomly sampled version of ModelNet40 to test our model. This dataset is prepared by the PointNet authors, and it consists of 40 categories of point clouds. Each point cloud is described by 10000 data points with each point has its x, y, z coordinates and the corresponding normal from mesh vectors. Since the points are ordered randomly in the point cloud, arbitrarily choosing a subset of them will still preserve the structure of the object. To accomplish our classification task, we use the x, y, z coordinates of the first 1024 points in each point cloud. In the later discussion, we will include an experiment where we, in addition to the x, y, z coordinates, also feed the normal channels into our network.

4.2. Execution Details

We trained our model on a GTX 1080 GPU. We choose our batch size as 60 and we choose 1024 points per a given point cloud because we find this configuration gives a reasonable result of greater than 70% test accuracy under 7 hours while also avoid using more memory the GPU can provide. In all of our experiments, we train our model for 200 epochs. We use a learning rate of 0.001, just like the original paper authors do. This configuration would be the default model. Based on it, different modifications have been implemented and analysed.

4.3. Farthest Point Sample layer(FPS)

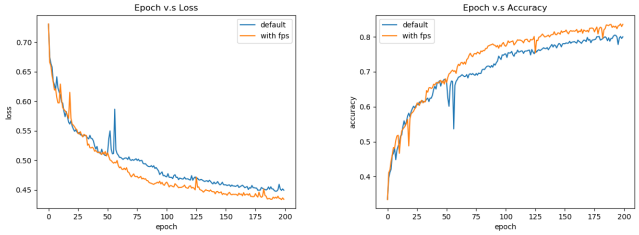
Compared with the random sampling, FPS would extract the points that could better represent the shape of the point cloud. In this algorithm, a seed point is randomly chosen as the initialization in the group of sampling points. Then, for every other point, its distance to the result set is calculated, defined as the distance from the point to its closest point in the result set. The farthest point is added to the samples, and this step is repeated until we obtain enough sample points. As shown in Figure 4, selecting 1024 points from 10000 points using Farthest Point Sampling is a better representation without visual occlusion of areas compared with the randomly sampling method.



(a) Randomly Sampling (b) Farthest Point Sampling

Figure 4: Farthest Point Sampling compared with randomly sampling

Figure 5 shows the performance difference between the default model and the FPS model. The modification has certainly boosted the performance, but to a limited extent with the sacrifice of much longer training time. The training of the FPS model took 4.8 times that of the default model, though the time should be shorter if the execution of this part is also migrated to GPS.



(a) Loss v.s. epoch (b) Accuracy v.s. epoch

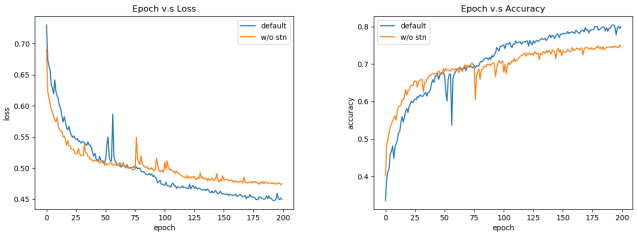
Figure 5: Performance comparison with or without FPS

4.4. Spatial Transform Network(STN)

Due to the invariance properties of the point set, two alignment networks are inserted in PointNet at the initial stage and after the 64-dim data is obtained. This is to ensure the classification would be invariant to rigid spatial transforms such as rotation and transformation. 3×3 and 64×64 matrices are generated by the networks respectively, which are them applied on the points and features to align them. These two mini-networks are inserted into the model and trained together.

To make convergence easier, random noise is added to the identity matrix as the initialization. If we take the identity matrix itself as output directly, it is difficult to optimize the spatial transform model, because there is no orientation information of the ground truth and there are specific requirements for a rotation matrix. In the original paper, the author are also including a regularization term in the final loss function, with a weighting parameter of 0.01. This loss inclusion is for constraining the rotational matrix as an orthogonal matrix, which is more stable mathematically. However, we have shown that even without this regularization loss, the inclusion of STN has improved the performance of the networks.

It could be observed that with the spatial transform networks, the training is more stable with higher accuracy and lower loss reached, as shown in Figure 6:



(a) Loss v.s. epoch (b) Accuracy v.s. epoch

Figure 6: Performance comparison with or without spatial transform networks

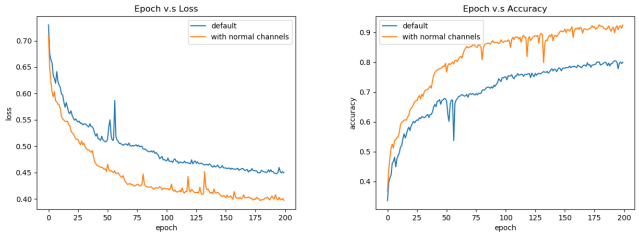
4.5. Normal Channels

So far, the inputs to the PointNet are the 3D-coordinates of the sample points. While relatively effective, they contain only local features of the shape since the networks are operated on each point separately. To include the global features in the classification process, the networks are trained with normal included as additional dimensions. The structure of the networks remained to be the same, except for the first alignment mini-network changed to be 6×6 instead.

As shown in Figure 8, the performance of the networks has further improved due to the inclusion of normal information, as expected.

4.6. Noise Input

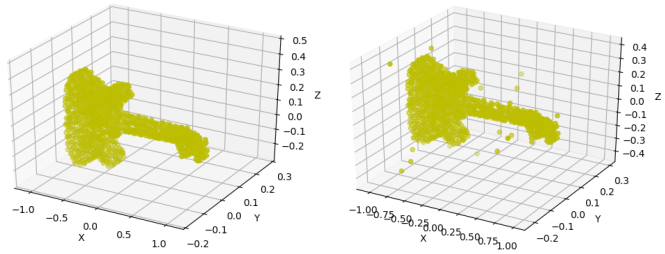
Since PointNet is processing independent point, the noise included in the data should have considerate influence



(a) Loss v.s. epoch (b) Accuracy v.s. epoch

Figure 7: Performance comparison with or without normal information

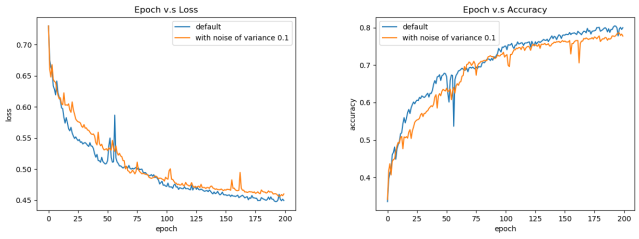
in the network training and test. In the experiment, we have applied 3% noise $n \sim N(0, 0.1)$ onto the sampling points, and analysed how that would influence the PointNet training and testing.



(a) Sampling Points without noise (b) Sampling Points with noise

Figure 8: Point clouds without noise and with 3% noise $n \sim N(0, 0.1)$

It appeared that the inclusion of a small amount of noise in the data does not have a critical influence for the training process, resulting in only a slight drop in performance in training.



(a) Loss v.s. epoch (b) Accuracy v.s. epoch

Figure 9: Performance comparison with or without noise applied

4.7. Critical Subset

The critical point set contains those points which contributed to the global feature obtained by max pooling. As shown in Figure 10, the red points are critical points and the yellow ones are 1024 sampling points. Generally, these points are distributed along the contour and vertex of the object.

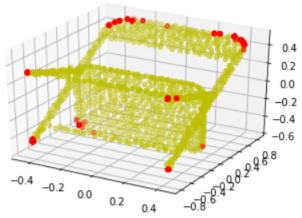


Figure 10: critical point set

4.8. Evaluation

The model is tested with the test partition of ModelNet40. The test accuracy is the ratio of correct predictions over ground truth labels. Results are listed in Table 1.

Table 1: Test Accuracy for different modifications in the model

Model	Test Accuracy (%)
Default	70.09
FPS	74.37
Without STN	67.82
Normal Included	84.58
Noise Included	49.29

From the testing results, it could be concluded that the model with normal as additional input reaches the highest accuracy. This is due to the inclusion of global features, though it could be observed from the learning curve that it is less stable than the default model. Following that, the test accuracy of the default model is 74.34%. When spatial transform networks are removed, the test accuracy dropped to 67.82%. Though it appeared to be converging faster initially compared with the default model, the converged performance falls behind. This disparity in the training and final test accuracy indicating the effectiveness of the alignment networks and the influence of the transformation invariance feature of point cloud representation in classification. When including noise in the data, the test accuracy dropped to the lowest, only 49.28%. Hence for noisy 3D data, it is suggested to preprocess them through some noise removal procedure before feeding into PointNet.

5. Conclusion

In this project, we successfully implement a deep neural network, PointNet, which directly takes point clouds as input. The output of the max-pooling layer, object's global feature, is fed into the classification layer and we obtain a satisfying result. We proposed the modifications with the sampling layer (FPS), spatial transform network (STN) and the normal channels, and made comparisons with the original network. The network converges faster during training with those modifications and they also have higher accuracy according to the evaluation process. However, the model does not use the information of the surrounding points. The noise in the input points could have a great impact on the results. Future work may include considering a larger convolution kernel in the deep neural network and implementing our work as a local learner. Adopting a reasonable grouping method may also increase the robustness of the model and bring us more information about the point cloud data.

6. Acknowledgement

The authors gratefully acknowledge the support of Professor Andrew Owens, Haozhu Wang, Anthony Liang and Bingqi Sun, and all the staffs of Winter 2020 EECS 504 at the University of Michigan.

References

[1] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

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

[3] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas 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.

[4] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[5] Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Y Ng. Convolutional-recursive deep learning for 3d object classification. In *Advances in neural information processing systems*, pages 656–664, 2012.

[6] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

[7] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[8] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.