# Point Cloud Classification with PointNet

**Shayan Hoshyari** **Zicong Fan** **Dylan Green**

## 1 Introduction

Three-dimensional geometric models are widely used in many fields such as computer graphics, computer-aided design (CAD), and multimedia. Given the recent popularity and unprecedented success of deep learning methods for applications in image processing and understanding, a number of novel architectures have been proposed for the tasks of classification and segmentation of 3-D geometry data. In this report we briefly discuss the existing literature on this topic and discuss our implementation of the PointNet architecture for classification using PyTorch. We also provide the custom traced MNIST dataset, which is based on MNIST, but is represented in point clouds. Finally, we validate our implementation using the traced MNIST and ModelNet10 datasets.

## 2 Related Work

### 2.1 Multi-View Based and Voxel Based Methods

Most of the early approaches for reasoning about 3D geometry leverage Multi-View and Voxel-Based neural network architectures. Multi-View Based Methods are based on the observation that a 3-D model can be rendered as one or multiple 2-D images, which can be processed through conventional Convolution Neural Networks (CNNs) [13]. An obvious strength of this approach is that it leverages existing tools in the image processing community and allows the use of pre-trained models. However, this approach often comes at the cost of many hyperparameters (e.g., the camera placement procedure).

Voxel-Based Methods focus on performing convolutions on a spatial grid. For example, Wu et al. [16] used a uniform $30 \times 30 \times 30$ grid to represent the input data and built a generative model (a Convolutional Deep Belief Network (CDBN)) which supports a variety of applications such as shape detection, shape completion, and next best view prediction. Nevertheless, training this model is difficult and the input voxel resolution is insufficient to capture complex object shapes. Wang et al. [15] and Riegler et al. [10] relaxed the computational cost of uniform spatial grids by representing the input field with octrees. Unfortunately, implementing such octree based methods using standard neural network libraries (e.g., PyTorch [7]) is non-trivial. Several works have explored alternative methods to voxel-based convolutions. Li et al. [6] propose a field probing scheme, which converts the input shape into a distance field and samples this field at several positions. The positions and weights for each filter are then learned during training. The output units of the probes are then passed to a fully connected network to predict class probabilities. This method is more memory efficient and easier to implement compared to the uniform grid based convolution schemes.

### 2.2 Point Cloud Based Architectures

Qi et al. [8] introduces a network architecture that can directly process point clouds. The input to the network is the 3D position (and, optionally, additional information such as normals or colours) of a variable number of points. The main challenge, in comparison to a conventional CNN, is that the output should be invariant under rigid body transformations or permutations of the input points. They introduce the PointNet architecture which satisfies the order invariance property and theoretically prove that it can approximate any order invariant function given enough hidden units. The proposed network architecture achieves results on par with state-of-the-art methods. Since sampling points from the surface of a 3D model leads to much less data compared to multi-view renders or uniform volumetric sampling, the proposed network is more efficient ($\sim 10-100\times$) and has fewer parameters ($\sim 10\times$) compared to its voxel-based and multi-view-based predecessors. The sampling of point clouds can also be done adaptively for further optimization. Nevertheless, the neighborhood of each point is completely ignored in the network until the very last max-pooling layer. To resolve this issue, Qi et al. [9] introduced the Pointnet++ network which partitions points into smaller clusters and performs PointNet operations on each cluster to mimic a convolution effect. As a result, PointNet++ provides an improvement in performance but suffers from computational cost and contains more parameters.

Apart from the PointNet architecture, there are other approaches available for point cloud data. Klokov and Lempitsky [5] introduce a novel convolution and pooling like operation based on $k$-d trees. Atzmon et al. [1] define convolution and pooling operations on point clouds by leveraging radial basis functions (by heavily relying on sparse matrix operations), and theoretically justify the robustness of their approach to noisy and sparse point-clouds.

### 2.3 Learning on Graphs

Due to the trending popularity of graph-based neural networks in the deep learning community (e.g., semi-supervised document classification [4], image scene graph generation [17] and visual question answering [14]), there is also interest in exploiting graph structures for applications involving point clouds. For example, KCNet was introduced by Shen et al. [12] to improve the computational efficiency of PointNet++. Specifically, they provide two operations for local point cloud structures. One operation is to define a convolution kernel for point clouds to capture the local geometric structure. The other operation utilizes feature aggregation on a $k$-nearest-neighbor graph based on 3D positions. As a result, KCNet achieves comparable accuracy in point cloud classification and parts segmentation tasks while maintaining a number of parameters and inference speed similar to that of PointNet.

## 3 Methodology

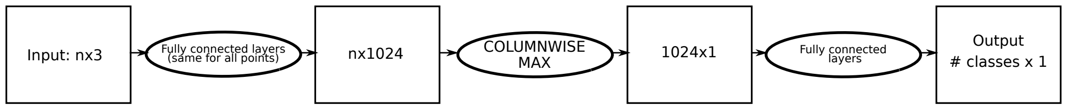

Figure 1: **The vanilla PointNet architecture:** the vanilla PointNet processes $n$ input points independently through a sequence of fully-connected (FC) layers with ReLU activation; a max pooling procedure is then applied to aggregate the $n$ points with 1024 dimensional features; finally, with another sequence of FC layers with ReLU activation, the pooled features are mapped into the label space for classification.

In our classification task, we take a point cloud as input, and wish to output the class that it belongs to. We decided to implement the vanilla PointNet architecture [8] to solve this problem, due to its simplicity and promising performance. For a set of $n$ points $\{P_i \mid i = 1, \ldots, n\}$ (where $P_i$ contains either 2D or 3D coordinates plus extra features such as colour and normal), each point is first independently lifted to a higher dimensional feature space through a sequence of fully-connected (FC) layers with rectified linear unit (ReLU) activation. Then, a max pooling procedure is applied to combine the high dimensional information of the individual points. Finally, the pooled features are passed through a second series of FC layers whose output can be used in conjunction with the softmax function to produce per-class probabilities. This procedure is schematically illustrated in 1. Note that individual points interact with one another only in the max pooling layer.

The full PointNet architecture [8] includes additional heuristic layers to further improve the performance. These layers are mostly responsible for making the network invariant under rigid transformations of the input 3-D model. We have not considered these additional layers for a few reasons: the time constraints of a course project are restrictive, the datasets that we have studied (see Section 4) include oriented shapes, and finally the ablation studies of Qi et al. [8] show that the vanilla version of the network already achieves acceptable results.

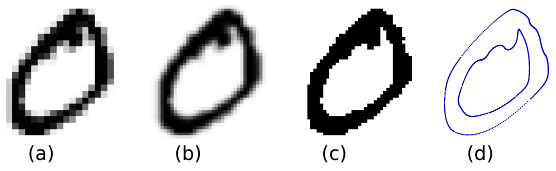

Figure 2: **Creating the traced MNIST dataset**: (a) Sample from the MNIST dataset, (b) scaling by linear interpolation, (c) thresholding, (d) tracing with the Potrace library [11]

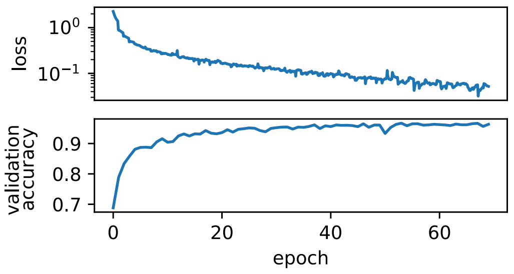

Figure 3: **Loss and validation accuracy over epochs for training on the Traced MNIST dataset**

## 4 Experiments

We have implemented the vanilla pointnet architecture using the PyTorch library [7], and validated it on the traced MNIST and ModelNet10 [16] datasets. The traced MNIST dataset was created as part of this work (see Section 4.1 for details) as a variant of its predecessor, MNIST [2], by shifting the data representation from pixels to point clouds.

### 4.1 Traced MNIST Dataset

**Dataset Overview:** The traced MNIST dataset consists of 48k training samples, 12k validation samples, and 10k test samples. Each sample contains a point cloud that represents "the outline" (see Fig. 2d) of a hand-written digit. We have created these point clouds by processing the original images in the MNIST datasets, following the Potrace [11] bitmap tracing pipeline (see Figure 2). Each digit image is thresholded to black and white and then passed to the Potrace library so that its outline is traced as smooth Bezier curves. Then, points are sampled (with a distance of 0.1 pixel) from the Bezier curves to construct the sample point clouds. To construct smoother outlines and leverage the information hidden in the anti-aliased pixels, we first scale the MNIST images(2x) with linear interpolation before thresholding them. To automate this process, we wrote a Python binding for the Potrace C library, and then processed the whole MNIST dataset via a Python script.

**Training and Evaluation:** For the first FC network in the vanilla PointNet architecture (Figure 1) we have used layer sizes of 2, 64, 128, and 1024 for the input, hidden, and output units, respectively. The second FC network layer sizes are 1024, 512, 256, and 10 for the input, hidden, and output units, respectively. Note that the number of units in the first network (2) and the number of output units in the second network (10) are dictated by the space dimensions and the number of classes, respectively. The number of layers and the number of hidden units have the same values as the original work of Qi et al. [8]. We experimented with using smaller numbers of layers or hidden units, but could not acheive competing results. For training, we used the Adam optimizer [3] with a learning rate of $10^{-3}$ and a batch size of 300 for 70 effective passes over data (epochs). The training loss and validation accuracy are shown in Figure 3. We evaluated our model on the test set and achieved a test accuracy of 96.5% (cf. the accuracy of 99% achieved by the methods that use inputs represented by pixels [2]). The accuracies and precisions per class are shown in Table 1. The table shows that our model achieves above 90% both in accuracy and precision across all categories in the Traced MNIST digit classification task.

Table 1: **Precision and recall for each class in Traced MNIST test set**

| category | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| precision | 98.9 | 99.2 | 96.0 | 93.0 | 97.7 | 98.7 | 98.8 | 97.1 | 91.4 | 94.9 |
| recall | 97.2 | 98.6 | 96.5 | 97.1 | 97.1 | 93.3 | 96.7 | 95.8 | 97.5 | 94.7 |

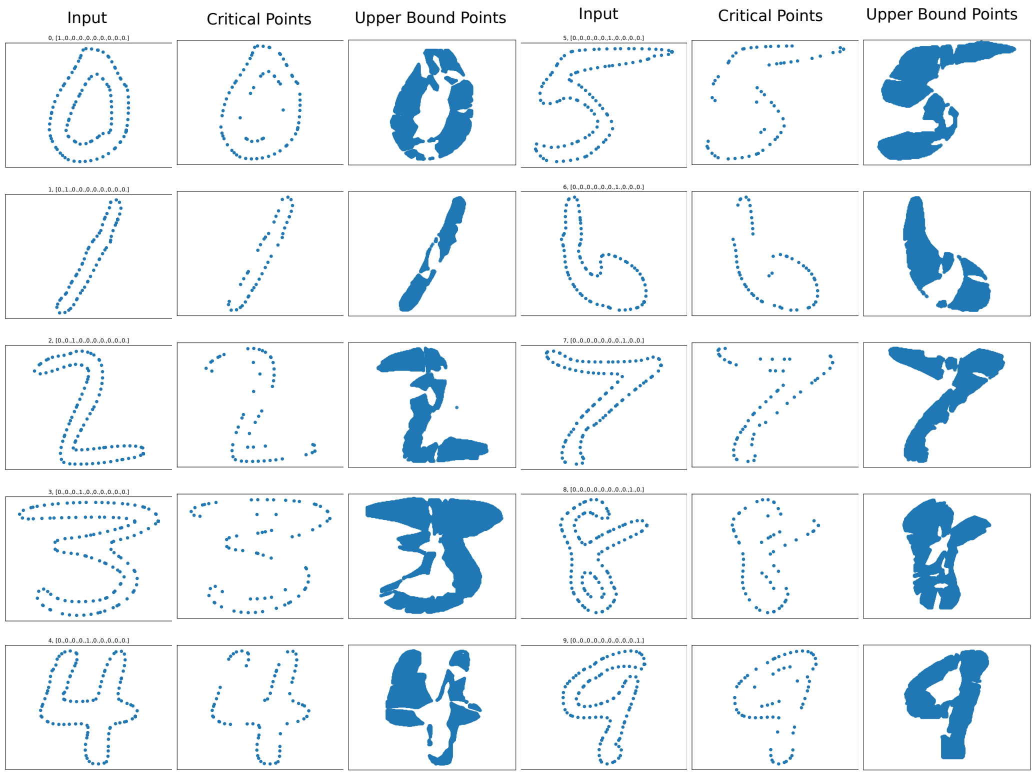

Figure 4: **Examples of correctly classified cases in Traced MNIST test set**. The ground truth label and the predicted probabilities over the categories from 0 to 9 of the classifier are shown above the input.

**Qualitative Analysis**: Figure 4 provides some successful predictions from our model. Each example is provided with an input (along with its ground truth label and softmax probability over the categories from 0 to 9). On the other hand, Figure 5 shows the failed predictions. The failure cases are separated to the left and right regions. The left region contains examples that are difficult for a human to classify; the right region contains the ones that are easier for a human to classify.

Additionally, following the visualization procedure of Qi et al. [8], we studied what the model has learned by visualizing the critical and the upper bound point-sets of an input shape (see Figure 4). The critical points are the set of points that contribute to the max pooling layer. In other words, it reflects a small set of points that dictate the output of the network. The upper bound point-set is the set of all points in space whose addition to the corresponding sample will not change the values in the max pooled units. Intuitively, one would expect the upper-bound points to be the interior of the model, as adding points to the interior of the shape should not change its class. Figure 4 validates that this is indeed the case.

### 4.2 ModelNet10 Dataset

**Dataset Overview:** To further examine our implemented network, we evaluated it on the ModelNet10 [16] dataset. The dataset contains ten categories of 3D CAD models of common real-life objects. Figure 6 shows the 10 categories along with some examples of the training set. ModelNet10 contains surface triangle meshes. To sample point clouds from the meshes (see Figure 9) we incorporate the following procedure. First, we sample a triangle from the mesh where the sampling

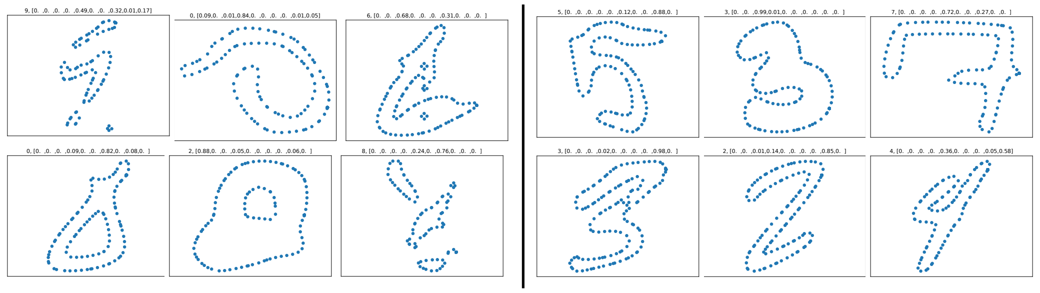

Figure 5: **Examples of wrongly classified cases in Traced MNIST test set**. The ground truth label and the predicted probabilities ranging from the category 0 to the category 9 are shown above the input. On the right we showcase which are trivial for a human to classify, whereas, on the left, we show more ambiguous cases.

Table 2: **Precision and recall for each class in ModelNet10 test set.** The model was both trained and tested on 1024-sized point clouds.

| category | toilet | table | sofa | night stand | monitor | dresser | desk | chair | bed | bathtub |
|---|---|---|---|---|---|---|---|---|---|---|
| precision | 97.2 | 85.7 | 98.0 | 84.3 | 95.1 | 70.3 | 72.4 | 99.0 | 100.0 | 84.2 |
| recall | 100.0 | 78.0 | 97.0 | 68.6 | 98.0 | 90.7 | 73.3 | 100.0 | 86.0 | 96.0 |

probability of each triangle corresponds to its area. Once a triangle is chosen, we uniformly select a point from the the triangle. By repeating the procedure $n$ times (triangles are allowed to be picked more than once) we obtain a point-cloud with $n$ points.

**Training and Evaluation:** Here, we have once again chosen the hyper-parameters (i.e., number of layers, hidden unit sizes, and point-cloud size) based on the the work of Qi et al. [8]. We use the same model (except for changing the input unit size to 3 in accordance to the space dimensions) and training procedure as in the previous experiment. Also, we use point cloud sizes of $1024$. Although the model can handle arbitrary sizes, a fixed size amongst all training samples is necessary for training in batches. We obtained an average test accuracy of $88.6\%$ (cf. the accuracy of $89.2\%$ for the full PointNet architecture on the bigger dataset of ModelNet50 [8]). Using $10\%$ of the training data as our validation set and a batch size of $50$, the training loss and validation accuracies over epochs are provided in Figure 8. The precision and recall over each category are shown in Table 2. Not surprisingly, classes with distinct shapes (e.g., toilets, sofas, and monitors) are identified with high precision and recall values, whereas the classifier has struggled to distinguish classes with similar shapes (e.g., desks and tables).

One practical question is the effect of point cloud size on accuracy. We conducted a study where we trained and tested our model on various point cloud sizes. Figure 9 shows the test accuracy obtained via different combinations of point cloud sizes. It seems that the model performs equally well for point cloud sizes of $1024$ and $4096$. Strangely, the accuracy drops for the model trained on a point cloud size of $2048$. We suspect that this might be due to an artifact in our training procedure, where we have selected the model parameters at the last iterate of the Adam optimizer, rather than selecting the model paramters that result in the smallest loss value. Not surprisingly, smaller point cloud sizes of $256$ and $512$ perform relatively poorly. This study can explain why Qi et al. [8] have selected the size of $1024$ for their sampled point clouds, as it offers the best trade-off between speed and accuracy.

## 5 Conclusion

In this work, we implemented the vanilla PointNet architecture [8] for the task of point cloud classification. We introduced the traced MNIST dataset and used it to validate our implementation. We further validated our implementation on the ModelNet10 [16] dataset. We also performed studies to gain further insight into what the network has learned, and the effect of point cloud density on the accuracy of the network.

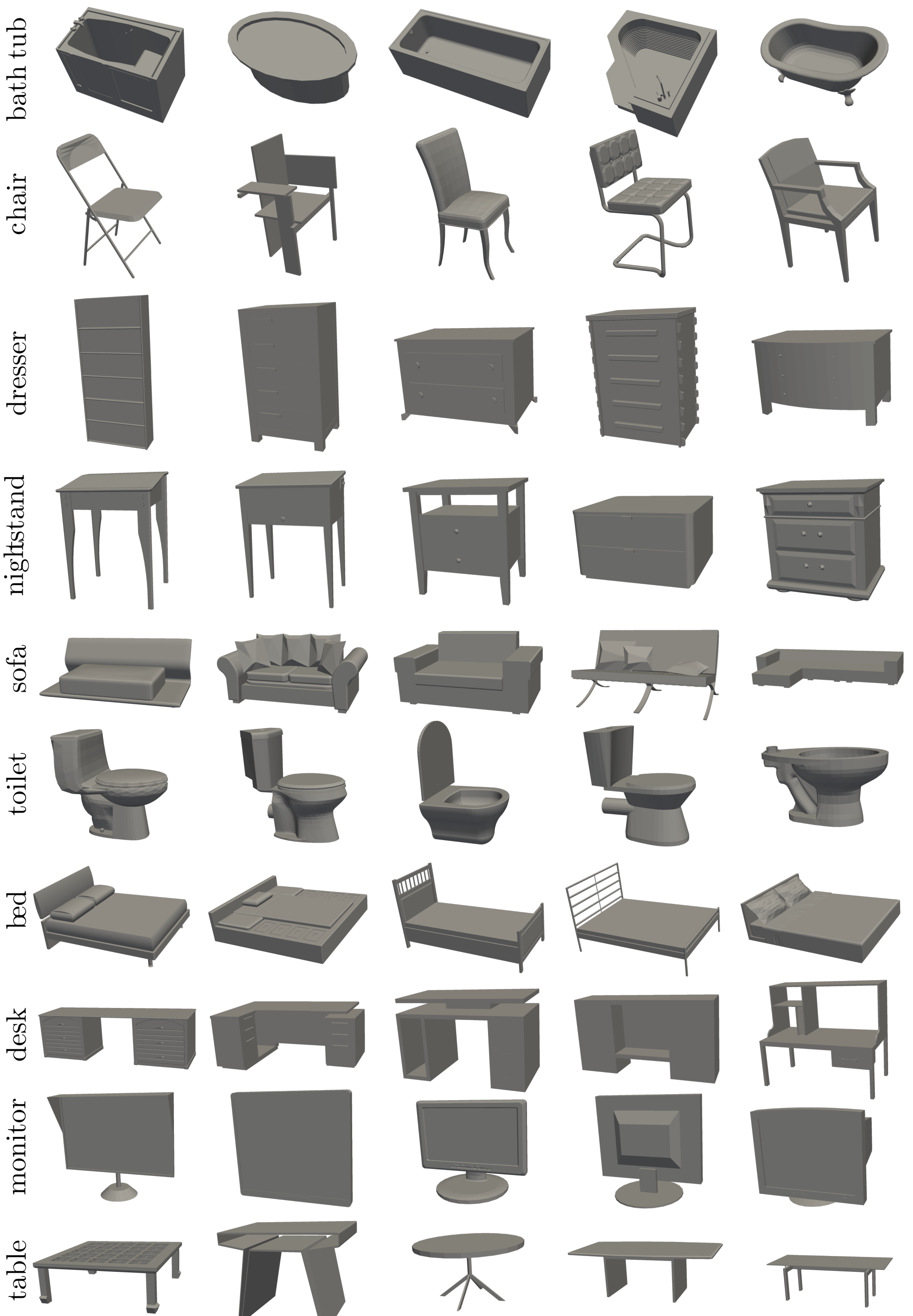

Figure 6: **Examples of samples in the ModelNet10 training data**

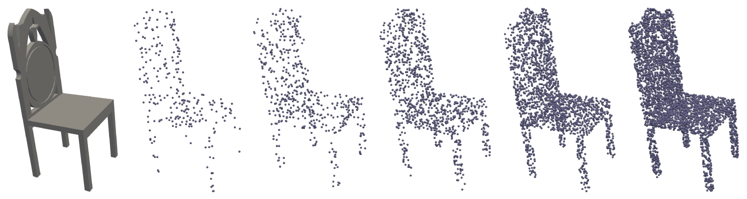

Figure 7: **Point cloud sampling**: input mesh, and point-clouds sampled with $256, 512, 1024, 2048,$ and $4096$ points

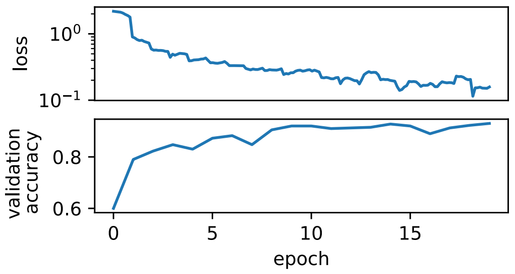

Figure 8: **Loss and validation accuracy over epochs for training on the ModelNet10 dataset**. The model was both trained and validated on 1024-sized point clouds.

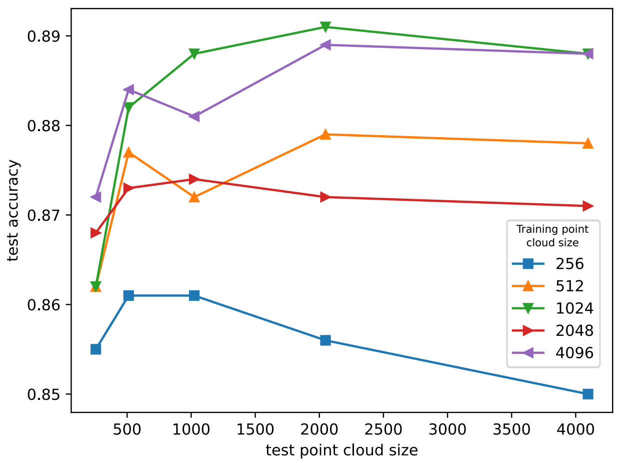

Figure 9: **Effect of various point cloud sizes at training and testing**

## References

[1] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *ACM TOG (SIGGRAPH Proc.)*, 2018.

[2] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *NeurIPS*, 2016.

[5] R. Klokov and V. Lempitsky. Escape from cells: deep kd-networks for the recognition of 3D point cloud models. In *ICCV*, 2017.

[6] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. FPNN: Field probing neural networks for 3D data. In *NeurIPS*, 2016.

[7] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[8] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. *CVPR*, 2017.

[9] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017.

[10] G. Riegler, A. Osman Ulusoy, and A. Geiger. Octnet: Learning deep 3D representations at high resolutions. In *CVPR*, 2017.

[11] P. Selinger. potrace: Transforming bitmaps into vector graphics. *http://potrace.sourceforge.net*, 2012.

[12] Y. Shen, C. Feng, Y. Yang, and D. Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *CVPR*, 2018.

[13] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. *ICCV*, 2015.

[14] D. Teney, L. Liu, and A. van den Hengel. Graph-structured representations for visual question answering. In *CVPR*, 2017.

[15] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM TOG (SIGGRAPH Proc.)*, 2017.

[16] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D Shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[17] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh. Graph r-cnn for scene graph generation. In *ECCV*, 2018.