# An Exploration of Object Classification on Point Clouds Through Deep Learning

Joshua Negreanu
Oregon State University
negreanj@oregonstate.edu

Jose Sanchez
Oregon State University
sanchej7@oregonstate.edu

Carson Nitta
Oregon State University
nittac@oregonstate.edu

## Abstract

*We discuss our final project in the applications of deep learning for point clouds. Point clouds as a data structure are discussed, followed by well-established deep learning methods for the classification and segmentation of point clouds. Among the three main ones, point-based models are the focus of this paper. The two models discussed in detail are PointNet and Point Transformer, both order-invariant deep networks designed to extract features from raw point clouds. This project overviews our implementation, training, and testing of both models on the widely-used ModelNet10 and ModelNet40 datasets. We note our observations and learning outcomes in the conclusion and discuss future work of interest.*

## 1. Introduction

With the utility of deep learning in the understanding of multi-dimensional data, such as images, it is a natural progression to approach 3-dimensional data with the tools of deep learning as well. However, unlike an image, which is a discrete grid of pixels, each with three separate color channels, 3D shapes can be expressed in a variety of ways, and live in a continuous space. Primarily, we are interested in real-world applications, meaning that deep learning models must be well suited for the exact data that 3D imaging tools provide.

### 1.1. Point Clouds

With these interests in mind, we discuss the concept of point clouds. A point cloud $P$ is a set of points $\{p_i\}$, where each point is attributed a 3D spatial coordinate, $(x, y, z) \in \mathbb{R}^3$, and potentially other features such as color, opacity, intensity, reflectivity, etc.

Note that because $P$ is a set, $P$ does not structurally change if points $p_i$ are reordered. Therefore, the same point cloud is represented regardless of ordering. Models that expect a point cloud must therefore be order-invariant.

LiDAR (Light Detection and Ranging) is a technology that allows the construction of point clouds for real-world objects. Understanding such point clouds can have incredible advantages in the areas of autonomous driving, robotics, and surveillance.

## 2. Related Works

Two common deep learning problems pertaining to point clouds are object classification and point cloud segmentation (which can consist of semantic, part, or instance segmentation).

Point clouds have seen the application of three main distinct types of deep learning approaches [3]. These will be discussed in brevity, and the last will ultimately be the focus of this project.

### 2.1. Multiview-Based

The first approach is a multiview-based method. This involves the separation of the 3D scene into multiple views, usually planar slices, and the extraction of features view-wise. These features are then combined to create a global feature of understanding. Often, points can be projected onto planes and discretized, allowing for CNN models to extract features.

### 2.2. Volumetric-Based

The second approach is a volumetric-based method. This involves the conversion of an object or scene into voxels, the 3D analog of pixels. 3D-CNNs, which utilize a cube kernel, can then extract information similar to a 2D-CNN on an image. Though sparse convolutions can help improve the efficiency of such a model, ultimately the conversion of a point cloud to its voxel-representation is computationally- and time-intensive.

### 2.3. Point-Based

The third approach is a point-based method. Such methods take in a point cloud as the raw set of points. Layers of the model are then shared among all points, and functions of the point cloud $P$ must be symmetric, meaning that order of inputs does not change the output. Often, and as will be shown in sections **2** and **3**, average and max are functions used to produce global features from a collection of points.

In this project, we focus on point-based deep learning on point clouds. Specifically, we compare an early order-invariant point-based deep learning model, PointNet [2], with a more recent point-based model, Point Transformer [6]. The latter utilizes self-attention [4], a development which has caused the transformer architecture to inject itself into so many different deep learning problems and with consistently impressive results.

We implement both models in PyTorch [1], noting a simpler implementation of the Point Transformer due to time constraints, and compare both models' performances on the Princeton ModelNet10 and ModelNet40 point cloud classification datasets [5].

## 3. PointNet

We replicate the architecture of PointNet. The model utilizes a backbone that can produce point features and a global feature. The global feature is passed through a multilayer perceptron for classification. The concatenation of point and global features is utilized for point segmentation.

The key component of PointNet is the T-Net, a mini point network that computes a linear operator for points within a space.

### 3.1. T-Net

The T-Net acts as a sub-network to the PointNet. The purpose is to propose a linear operator for a collection of points within a $d_f$ dimensional space. The T-Net applies a shared multilayer perceptron, followed by a global max pooling layer, which acts as a symmetric function. This global feature is passed through a linear layer, and the resulting vector of $d_f \times d_f$ length is reformatted into the shape of a square $d_f \times d_f$ matrix.

The same points are then multiplied by the learned linear operator into the same $d_f$ feature space.

### 3.2. Loss Function

For training PointNet, a special loss function is utilized. For classification, the baseline is cross-entropy loss. We want the second T-Net to act as a reordering of points into a canonical shape. However, because the second T-Net operator is a $64 \times 64$ matrix, it is highly unlikely that the T-Net will learn this properly. Therefore, PointNet attempts to ensure that this matrix is learned to be orthogonal, meaning columns and rows are orthonormal vectors. We calculate the loss of PointNet as:

$$L = L_{CE} + \lambda \|I - AA^T\|_2^2 \quad (1)$$

$L_{CE}$ is normal cross-entropy loss, $\lambda$ is the regularization weight, $A$ is the linear operator proposed by the T-Net, and $I$ is the identity. This regularizing term forces the model to make the linear operator closer to orthogonal, preserving data. It is also initialized as the identity and the T-Net learns to make modifications to it through training.

[1] https://github.com/joses2017smjh/PointCloudclassification

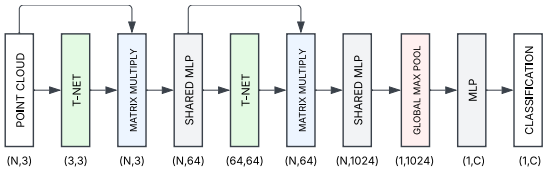


Figure 1. PointNet classifier architecture.

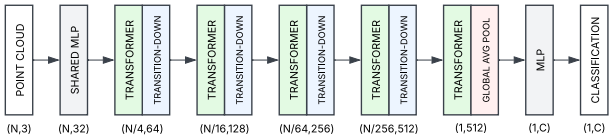


Figure 2. Point Transformer classifier architecture.

## 4. Point Transformer

Point Transformer [6] adopts the popular transformer architecture, employing self attention mechanisms between individual points. Point Transformer consists of an encoder and decoder. The encoder is utilized to create a global feature of the point cloud, which is directly handled by a multilayer perceptron for classification. The decoder is used for point cloud segmentation. We implement the Point Transformer decoder as a backbone and utilize a simple classification head. In this paper, we discuss our specific implementation, which, among other changes, uses scalar attention as opposed to vector attention as in the original paper.

The encoder utilizes two types of blocks: transformer blocks and transition-down blocks.

### 4.1. Transformer Block

The transformer block consists of an initial linear layer followed by four separate attention heads. We originally tried eight headed attention, but found this model to be too complex for training and prone to significant overfitting. Attended features from each head are then concatenated and ran through two more linear layers to transform to the original feature space. A residual is then added.

Each head performs point-wise attention. All points are ran through three separate linear operators, $q$, $k$, and $v$, resulting in the queries, keys, and values respectively.

Additionally, positional encoding is handled by a multilayer-perceptron function $\phi$. Because points are already positioned relative to each other in 3D space, the spatial coordinates $(x, y, z)$ provide a natural choice for the basis of positional encoding.

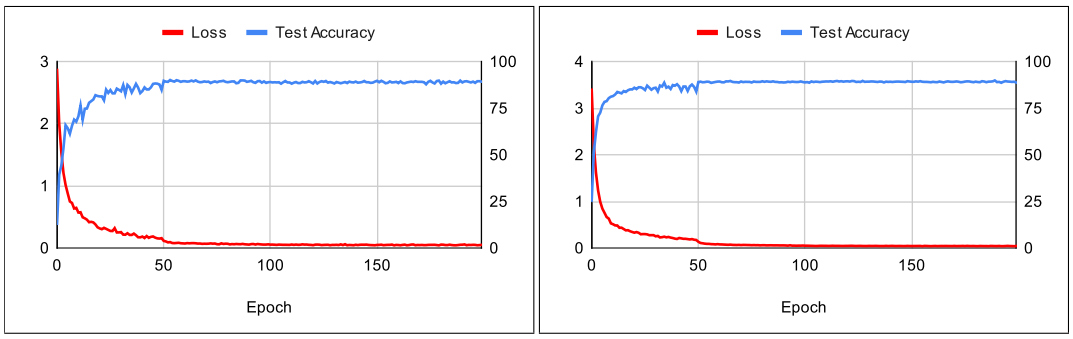


Figure 3. PointNet training on ModelNet10 (left) and ModelNet40 (right).

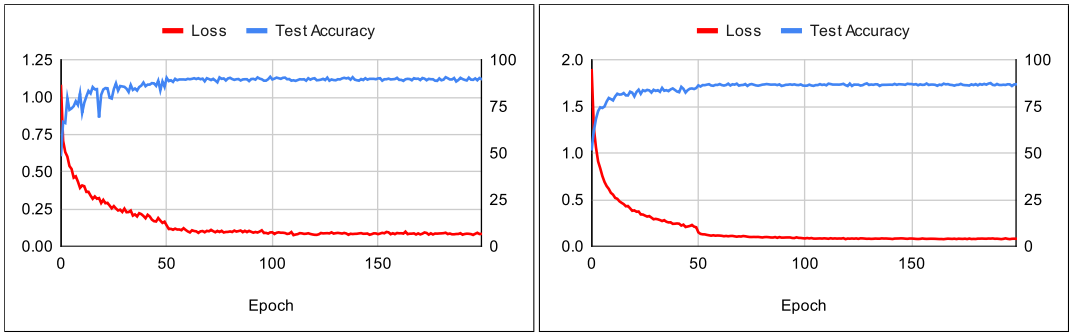


Figure 4. Point Transformer training on ModelNet10 (left) and ModelNet40 (right).

Therefore, a fully attended feature $\hat{x_i}$ for a point feature $x_i$ is calculated given all points' features $x_j$ and coordinates $p_j$:

$$\hat{x_i} = \sum_j \sigma \left( \frac{q(x_i + \phi(p_i))k(x_j + \phi(p_j))^T}{\sqrt{d_f}} \right) v(x_j + \phi(p_j)) \tag{2}$$

Note that $\sigma$ is a row-wise softmax and $d_f$ is the dimension of the feature vector $x_i$. $q$, $k$, and $v$ do not change the dimensionality. We denote the attended feature of the first head as $\hat{x_i}^{(1)}$. Each head has such an internal process, and we construct the following concatenated attended feature:

$$\hat{x_i}^* = [\hat{x_i}^{(1)}\ \hat{x_i}^{(2)}\ \hat{x_i}^{(3)}\ \hat{x_i}^{(4)}] \tag{3}$$

This is then ran through two more linear layers to transform into the original feature space, and then added to a residual.

### 4.2. Transition-Down Block

The transition-down block simultaneously decreases the number of points by a factor of 4 and increases the feature dimensionality by a factor of 2. This allows a select few points to convey localized point cloud information with an increased dimensionality. Points are randomly sampled from the point cloud and ran through a shared multilayer perceptron to increase the feature space. Note that using random sampling works well when the points are already evenly distributed. This is not the case in much real-world LiDAR data; on the contrary, points are often dense in certian regions and sparse in others. A better, and future, implementation would use grid-sampling, allowing for an equal representation of the entire point cloud.

## 5. Experiments

We loaded both datasets with a point sampling pre-transform of 1024 points. All points are then normalized.

| Dataset | Training Samples | Testing Samples |
|---|---|---|
| ModelNet10 | 3991 | 908 |
| ModelNet40 | 9843 | 2468 |

Table 1. Training and testing samples for ModelNet10 and ModelNet40.

### 5.1. Hyperparameter Tuning

An initial experiment was performed for both models to search for optimal hyperparameters. Each model was trained for 25 epochs on each dataset with different combinations of learning rate, momentum, weight decay, and (in the case of PointNet) regularization weight.

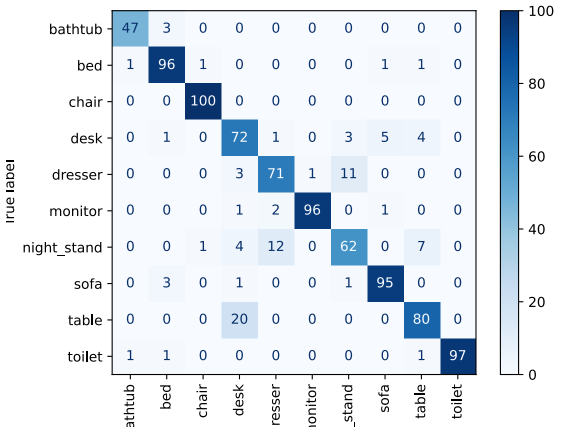


Figure 5. PointNet confusion matrix on ModelNet10.

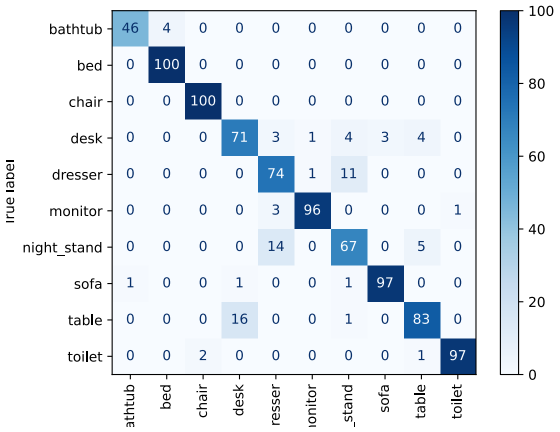


Figure 6. Point Transformer confusion matrix on ModelNet10.

| Architecture | ModelNet10 | ModelNet40 |
|---|---|---|
| PointNet | 90.97% | **89.95%** |
| Point Transformer | **92.4%** | 87.6% |

Table 2. Test accuracy performance of PointNet and Point Transformer on ModelNet10 and ModelNet40.

Our first set of experiments was done on Modelnet10 with stochastic gradient descent using a wide variety of testing parameters and a consistent batch size of 64. We tested learning rate values of $\{0.001, 0.01, 0.1\}$, momentum values of $\{0.8, 0.9\}$, weight decay values of $\{0.0, 1 \times 10^{-5}\}$, and regularization weights of $\{0.0001, 0.001\}$. This resulted in 24 total configurations. When adapting these experiments to Modelnet40, we excluded the learning rate of .0001 due to its poor results and slow convergence. This created only 12 different experiments which provided essential insights on tuning the model later on.

### 5.2. Training

For our final sets of experiments, we trained each model with stochastic gradient descent for 200 epochs with a batch size of 64, reducing the learning rate by a factor of 10 every 50 epochs. PointNet was trained on ModelNet10 and ModelNet40 with an initial learning rate of 0.1, momentum of 0.9, weight decay of 0, and regularization weight of 0.001. Point Transformer was trained on ModelNet10 and ModelNet40 with an initial learning rate of 0.01, momentum of 0.8, and weight decay of 0.0001.

We noticed that Point Transformer quite often tended to overfit, especially during these longer training sessions. This required the usage of weight decay to improve testing accuracy. However, improvements can certainly be made in regards to training our transformer model, as will be discussed.

## 6. Results

We note key results of both models in regards to both classification datasets. Our implementation of Point Transformer performed better than PointNet on ModelNet10, but performed worse on ModelNet40.

The latter result was a surprise for us, but in line with our difficult experience in properly training the transformer. We discuss the implications of this performance in sections **7** and **8**.

We also analyze the robustness of our models given different point cloud sizes. Training was done all on 1024-point clouds. We therefore test clouds of 256, 512, 1024, 2048, and 4096 points to determine if the models can still correctly classify with a reduced sample size. We also determine if more points leads to better performance.

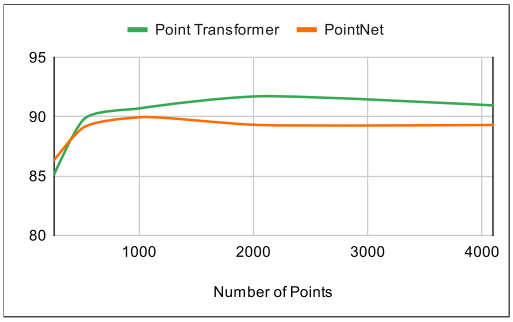


Figure 7. PointNet and Point Transformer performance on varying point cloud sizes on ModelNet10.

Point Transformer does worse with few points. In fact, Point Transformer can only operate with 512 points minimum, as down-sampling through transition-down blocks reduces the number of points by a factor of 512. PointNet,

however, can handle any number of points. However, with enough points, Point Transformer performs better. With more points than the training data, Point Transformer's attention mechanisms are able to take advantage of the extra information, improving performance even more. PointNet, however, does not utilize the extra data and drops in performance.

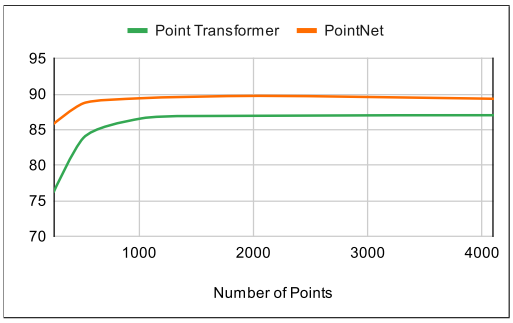


Figure 8. PointNet and Point Transformer performance on varying point cloud sizes on ModelNet40.

Because PointNet performed better than Point Transformer on ModelNet40, it is natural to see that PointNet performs better on all point cloud sizes. However, an interesting trend to note is that PointNet does not always perform better with denser point clouds, whereas Point Transformer incrementally yet consistently improves in performance given more data. If trained more carefully, such a model should be able to improve performance given better data.

This is a key feature of the transformer model; connections through attention are more fruitful given more data, instead of overloading the model.

## 7. Conclusion

The purpose of this project was to learn how deep learning is applied to point clouds. We studied two networks, PointNet and Point Transformer, and implemented each from scratch to gain a deeper understanding of their mechanisms. We also aimed to compare the two on the ModelNet10 and ModelNet40 benchmarks.

We conclude our paper with a discussion of the results obtained.

### 7.1. Discussion

We expected both models to perform well on ModelNet10, as it is a relatively simple dataset with only 10 classes. We also expected the Point Transformer to perform better than PointNet, even with our simplified implementation. This is what we saw in the results in Table 1.

However, PointNet performed better than Point Transformer on ModelNet40, a more complex classification dataset. We hypothesize why it is our implementation was unable to meet the same performance level.

Because Point Transformer is so adaptable through the attention mechanism, it can easily overfit. Our original design had eight heads per transformer block. However, we reduced the number to four. Reducing model complexity helped stabilize training, but it was still challenging. In the end, a high-performance version of the Point Transformer was never achieved for ModelNet40, though we are sure it is capable.

This may be primarily due to a limitation in our implementation. In transition-down blocks, we simply random sample points and pass them through a shared multilayer perceptron to increase feature dimensionality. However, the paper samples the KNN of a point and essentially passes them through a mini PointNet, incorporating spacial detail of surrounding points into the sampled point. This information is lost in our model; relationships are only attained through attention mechanisms. And this works well, but can be improved through a more robust transition-down layer.

This may also explain why our Point Transformer performs poorly on a smaller point cloud, because downsampling looses too much information.

## 8. Future Work

We discuss future work in this area pertaining to deep learning on point clouds. The time window for this project was limited, but we plan on continuing our interests in the area.

### 8.1. Segmentation

A hope we had for this project was to test out point cloud segmentation. Specifically, we planned on training and testing PointNet and Point Transformer on ShapeNet [1], a part segmentation point cloud dataset with 16 object classes and 50 part classes. However, we recieved access to ShapeNet too late into the term, and due to time constraints we were unable to apply the same techniques and rigor to the implementation, training, and testing of point cloud classification. Nonetheless, we did study the methods in which segmentation is handled by both architectures, and plan on implementing and testing them at a later time.

### 8.2. Developing a Unique Architecture

Another future goal would be to propose a novel architecture for deep learning on point clouds. Most likely inspired by the transformer model, as it is a keystone in the current deep learning paradigm, we would like to construct a light-weight and efficient model that can perform classification and segmentation well. We have laid the groundwork in our understanding for such a project, and will pursue it in the future.

## References


[1] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

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

[3] S. Sarker, P. Sarker, G. Stone, R. Gorman, A. Tavakkoli, G. Bebis, and J. Sattarvand. A comprehensive overview of deep learning techniques for 3d point cloud classification and semantic segmentation. *Machine Vision and Applications*, 35(4), May 2024.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[5] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. 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.

[6] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.