A Comparative Study Using the ModelNet Dataset for Point Cloud Classification Using Different Machine Learning And Deep Learning Models

Lalith Teja Nagidi-lalith@udel.edu
Masters in Data Science
University of Delaware

Dhana Kankanala-dhanak@udel.edu
Masters in Data Science
University of Delaware

Kishore Madithati-kmaditha@udel.edu
Masters in Data Science
University of Delaware

Ravi Chigurupati-ravich@udel.edu
Masters in Data Science
University of Delaware

Harshitha Chengalraya-harshith@udel.edu
Masters in Data Science
University of Delaware

Ashish Mulaka-ashishr@udel.edu
Masters in Data Science
University of Delaware

***Abstract*—Point cloud is a set of data points in 3D space that represents the shape or surface of a physical object, and we took modelnet dataset which contains CAD models from the 40 different categories which are most commonly used objects like bed, chair, desk, dresser, sofa.etc. and we used different Machine learning/ Deep Learning techniques like Convolutional neural networks, MLP classifier, Random Forest classifier and Pointnet to analyze how these models are performing on this kind of dataset and we also perform this by varying number of samples in order to understand how well the models are predicting by reducing or increasing the number of samples.**

I. Introduction

Point cloud data is widely used to represent data in diverse fields such as augmented reality, computer vision, and robotics. It allow the representation of objects or scenes in a three-dimensional space through a set of data points. Researchers and practitioners have investigated a variety of applications, such as object recognition, scene interpretation, and shape analysis, by utilizing the information included in these point clouds.

In this project, our objective is to conduct a comparative study of different machine learning and deep learning models for point cloud classification. Mainly, we will be working with the ModelNet dataset, which comprises a collection of CAD (Computer-Aided Design) models from 40 distinct categories. These categories encompass a wide range of commonly encountered objects, such as chairs, beds, desks, and sofas. By training and testing various machine learning and deep learning models on this dataset, we can gain strengths, weaknesses, and overall performance when it comes to point cloud classification tasks.

After getting results from various models, we are interested to identify the most suitable models for point cloud classification, by considering factors such as accuracy, computational efficiency, and robustness. The findings of this project help in fields like augmented reality, computer vision, and robotics, where accurate and efficient point cloud classification is crucial for tasks ranging from object recognition in real-world environments to autonomous navigation.

II. Related work

The paper titled "Multi-view Convolutional Neural Networks for 3D Shape Recognition" by H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller. This team perform 2D views can be highly informative for 3D shape recognition and is amenable to emerging CNN architectures and their derivatives with an accuracy of 68.26%.

In the paper titled "Deep Learning with Sets and Point Clouds" by Siamak Ravanbakhsh, Jeff Schneider, Barnabas Poczos. This team mainly focus to demonstrate the usefulness of layer in set-outlier detection as well as semi-supervised learning with clustering side-information.

In the paper titled "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation" by Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas. This team model provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing.

In the paper titled "PointNet: A 3D Convolutional Neural Network for real-time object class recognition" by A. Garcia-Garcia, F. Gomez-Donoso†, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, J. Azorin-Lopez. This team model work, to propose PointNet, a new approach inspired by VoxNet and 3D ShapeNets, as an improvement over the existing methods by using density occupancy grids representations for the input data, and integrating them into a supervised Convolutional Neural Network architecture. An extensive experimentation was carried out, using ModelNet - a large-scale 3D CAD models dataset - to train and test the system, to prove that this approach is on par with state-of-the-art methods in terms of accuracy while being able to perform recognition under real-time constraints.

III. Data and Methods

We conducted an analysis on the ModelNet40 dataset and selected a small sample of 10 objects for our model development and testing to reduce the complexity of data and get more accuracy results to distinguish each object. The Table-1 below displays the 10 objects we included in our study, along with the respective sizes of the training and testing datasets. The Fig-1 below the table displays shapes of objects we considered.

TABLE I
Data Set Details

Object Name	***Train size***	***Test size***
Airplane	626	100
Bed	515	100
Bookshelf	572	100
Bottle	335	100
Chair	889	100
Mantel	284	100
Moniter	465	100
Sofa	680	100
Table	392	100
Toilet	344	100

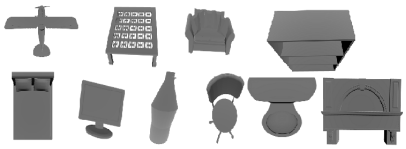

Fig. 1. Objects used in model

The dataset consists of object files in the (.off) format, which we load using the trimesh library. While other Python libraries like Open3D, PyMesh, and Blender can also load files for PointNet, trimesh offers distinct advantages such as being lightweight and focused, fast and efficient, supporting a wide range of file formats, and providing a simplified API.
Visualization plays a crucial role in understanding and analyzing the outcomes of PointNet models. PointNet is specifically designed to process unordered point clouds, where each point represents a 3D coordinate in space. To facilitate loading the mesh dataset, we utilize the trimesh function. From this dataset, we extract individual samples of 512, 1024, and 2048 points using the .sample() function, which randomly selects items along the object's axis. Our training set consists of 5102 points, while the total test set comprises 1000 points. The below Fig-2 depicts the visualization of a Airplane with different sample sizes, specifically 512, 1024, and 2048. The Fig-2 clearly illustrates that the sample size of 512 contains fewer points compared to 1024, and similarly, the sample size of 1024 has fewer points compared to 2048.
We employed various methods, including Random Forest Classifier, MLP Classifier (Multilayer Perceptron), CNN (Convolutional Neural Networks), and PointNet, for our model building process.

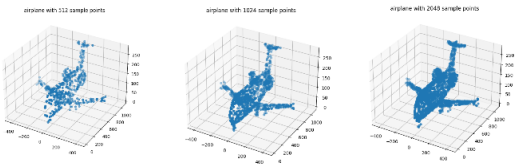

Fig. 2. Visualization of objects with different sample points

A. Random Forest Classifier

A Random Forest is an ensemble of a well-known algorithm. The goal is a set of de-correlated decision trees that produce better results than a single decision tree. To classify a new instance, each decision tree in the ensemble is given a vote, and the class with the target vote is selected as the prediction for the new instance. Below Fig-3 explains the architecture of Random Forest.

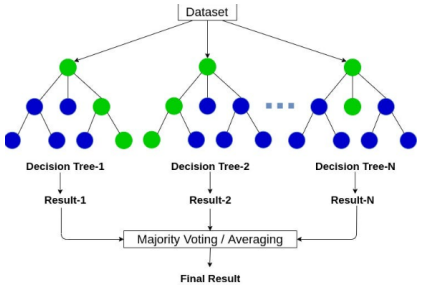

Fig. 3. Random Forest Architecture

Point cloud contains large number of data points in 3D, Random forest classifier can handle high dimensional data effectively. We have flattened the data points in 2D and achieved 76% accuracy. From the results which we obtained by using a Random forest classifier, we also plotted the Confusion Matrix of the Random Forest Classification Model. The confusion matrix presented below provides insights into the accuracy of predicted labels compared to the actual labels. For instance, the model exhibited confusion between the labels "bed" and "sofa" due to their similarities. Notably, as the number of points increased from 512 to 1024, more data points were considered, resulting in a higher occurrence of the model predicting "bed" as "sofa".

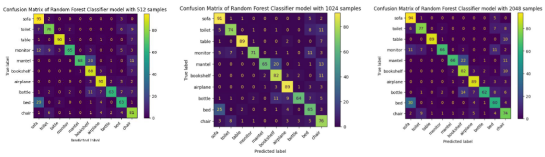

Fig. 4. Confusion Matrix of Random Forest Classification Model

B. Multi-Layer Perceptron

Multi-Layer Perceptron classifier is a feedforward artificial neural network used for supervised learning tasks, mainly for

classification problems. It consists of multiple layers of interconnected nodes known as neurons that mimic the structure of neurons in the human brain.

Basic structure of an Multi-Layer Perceptron classifier consists of three types of layers. Input Layer: This layer receives the input data, which could be a set of features or attributes associated with the problem at hand. Hidden Layers: These are intermediate layers between the input and output layers. Output Layer: This layer produces the final output of the MLP classifier. The functionality of an MLP classifier involves two main steps One is forward propagation In this step, the input data is fed into the MLP, and the computations flow from the input layer through the hidden layers to the output layer. Each node in the network calculates a weighted sum of its inputs, applies an activation function to the sum, and passes the result to the nodes in the next layer. Second is backpropagation, after the forward propagation, the output of the MLP is compared with the true labels of the training data to determine the prediction error. The error is then propagated backward through the network, adjusting the weights and biases of the connections between the nodes. This adjustment is performed using an optimization algorithm, such as gradient descent, to minimize the error and improve the model's performance. MLPClassifier(hidden_layer_sizes=(100, 100), max_iter1000, random_state=3).

C. Convolutional neural networks

Convolutional Neural Networks is a type of Artificial Neural Network, which is useful for finding patterns in images to recognize objects, classes, and categories. A neural network consists of neurons interconnected among one another, receiving information from previous layers and passing it to the subsequent layers. The output from the last layer is generally the output of the model.

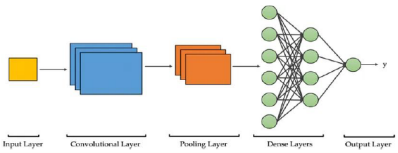

Fig. 5. Convolutional neural networks Architecture

Here, the input data is reshaped to satisfies the requirement of the Conv2D layers. As, we used Conv2D layers as the convolutional layers in this model to process the point cloud data. Initially, we used 2 Conv2D layers with the filters 32 and 64 each, along with the kernel size of (3x3) matrix and ReLU as an activation function. Then, we did the max pooling with pool size (2x2) matrix. Again, we have taken 2 Conv2D layers with the previous specifications but this time without the pooling layer. Then repeated the process with 2 more Conv2D layers with 126 and 264 filters each, along with the kernel size of (5x5) matrix and ReLU as an activation function. Finally, we used a Global max pooling layer. Subsequently, 2 Dense layers with 128 units and ReLU as an activation function for the first Dense layer and 10 units with the softmax as an activation function for the second Dense layer. We compiled the model using 'Adam' optimizer, 'sparse_categorical_crossentropy' as the loss function, and Accuracy as the metrics. Then we trained the model using the input dataset and 30 iterations with a batch size of 32. We have taken confusion matrix to analyze the performance of the model. As we can see the CNN model predicted very well at 512 data points. But with the increase in the data points from 512 to 1024 and then to 2048, the model is confusing on the objects, for instance bookshelf and mantle, bookshelf and bottle. Below Fig-7 explains about CNN Model summary.

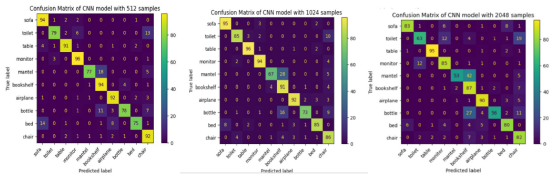

Fig. 6. Confusion Matrix of Convolutional Neural Network Model

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 1024, 3, 1)	0
conv2d_6 (Conv2D)	(None, 1024, 3, 32)	320
conv2d_7 (Conv2D)	(None, 1024, 3, 64)	18496
max_pooling2d_1 (MaxPooling 2D)	(None, 512, 1, 64)	0
conv2d_8 (Conv2D)	(None, 512, 1, 32)	18464
conv2d_9 (Conv2D)	(None, 512, 1, 64)	18496
conv2d_10 (Conv2D)	(None, 512, 1, 128)	204928
conv2d_11 (Conv2D)	(None, 512, 1, 256)	819456
global_max_pooling2d_1 (Glo balMaxPooling2D)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 10)	1290

Total params: 1,114,346
Trainable params: 1,114,346
Non-trainable params: 0

Fig. 7. CNN Model summary

D. PointNet

PointNet is a deep learning architecture created for analyzing and categorizing point cloud data. Without requiring any manual feature extraction or alignment, it can handle unordered and erratically sampled point clouds. In this project we are using pointNet for classification of objects. The following are the primary elements of the PointNet architecture:

- Input Point Cloud: PointNet receives a point cloud as input, which is a collection of 3D points.
- PointNet Encoder: The input point cloud is processed by the PointNet encoder, which extracts local and global characteristics.

- Global Feature Extraction: High-level details regarding the structure and geometry of the item in the point cloud are encoded in this global feature vector.

The primary advantage of PointNet is its capacity to manage unordered point clouds by utilizing shared MLP layers that each point can independently control. Because of this, PointNet permutation is invariant—that is, it yields the same results regardless of the arrangement of the points.

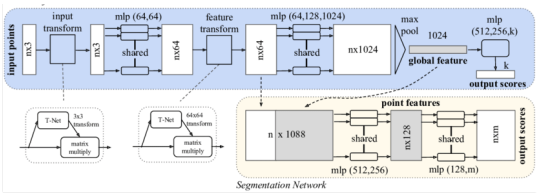

Fig. 8. PointNet Architecture

The TnetLayer function, a crucial part of PointNet that learns the transformation matrices to align and normalize the input point clouds, is defined in the provided code.

The use of pointNet in this project allows for direct classification without transforming data into other representations. It can handle point clouds with varying number of points, making it suitable for datasets like ModelNet,where the number of points may vary across different CAD models. Below Fig-9,10,11 explains about pointNet summary.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 1024, 3)]	0	[]
conv1d_22 (Conv1D)	(None, 1024, 64)	256	['input_3[0][0]']
batch_normalization_34 (BatchNormalization)	(None, 1024, 64)	256	['conv1d_22[0][0]']
activation_34 (Activation)	(None, 1024, 64)	0	['batch_normalization_34[0][0]']
conv1d_23 (Conv1D)	(None, 1024, 64)	4160	['activation_34[0][0]']
batch_normalization_35 (BatchNormalization)	(None, 1024, 64)	256	['conv1d_23[0][0]']
activation_35 (Activation)	(None, 1024, 64)	0	['batch_normalization_35[0][0]']
conv1d_24 (Conv1D)	(None, 1024, 256)	16640	['activation_35[0][0]']
batch_normalization_36 (BatchNormalization)	(None, 1024, 256)	1024	['conv1d_24[0][0]']
activation_36 (Activation)	(None, 1024, 256)	0	['batch_normalization_36[0][0]']
conv1d_25 (Conv1D)	(None, 1024, 512)	131584	['activation_36[0][0]']
batch_normalization_37 (BatchNormalization)	(None, 1024, 512)	2048	['conv1d_25[0][0]']
activation_37 (Activation)	(None, 1024, 512)	0	['batch_normalization_37[0][0]']
global_max_pooling1d_6 (GlobalMaxPooling1D)	(None, 512)	0	['activation_37[0][0]']
dense_22 (Dense)	(None, 512)	262656	['global_max_pooling1d_6[0][0]']
batch_normalization_38 (BatchNormalization)	(None, 512)	2048	['dense_22[0][0]']
activation_38 (Activation)	(None, 512)	0	['batch_normalization_38[0][0]']
dense_23 (Dense)	(None, 256)	131328	['activation_38[0][0]']
batch_normalization_39 (BatchNormalization)	(None, 256)	1024	['dense_23[0][0]']
activation_39 (Activation)	(None, 256)	0	['batch_normalization_39[0][0]']
dense_24 (Dense)	(None, 9)	2313	['activation_39[0][0]']
reshape_6 (Reshape)	(None, 3, 3)	0	['dense_24[0][0]']
dot_4 (Dot)	(None, 1024, 3)	0	['input_3[0][0]', 'reshape_6[0][0]']
conv1d_26 (Conv1D)	(None, 1024, 64)	256	['dot_4[0][0]']

Fig. 9. pointNet summary

batch_normalization_40 (BatchNormalization)	(None, 1024, 64)	256	['conv1d_26[0][0]']
activation_40 (Activation)	(None, 1024, 64)	0	['batch_normalization_40[0][0]']
conv1d_27 (Conv1D)	(None, 1024, 64)	4160	['activation_40[0][0]']
batch_normalization_41 (BatchNormalization)	(None, 1024, 64)	256	['conv1d_27[0][0]']
activation_41 (Activation)	(None, 1024, 64)	0	['batch_normalization_41[0][0]']
conv1d_28 (Conv1D)	(None, 1024, 64)	4160	['activation_41[0][0]']
batch_normalization_42 (BatchNormalization)	(None, 1024, 64)	256	['conv1d_28[0][0]']
activation_42 (Activation)	(None, 1024, 64)	0	['batch_normalization_42[0][0]']
conv1d_29 (Conv1D)	(None, 1024, 64)	4160	['activation_42[0][0]']
batch_normalization_43 (BatchNormalization)	(None, 1024, 64)	256	['conv1d_29[0][0]']
activation_43 (Activation)	(None, 1024, 64)	0	['batch_normalization_43[0][0]']
conv1d_30 (Conv1D)	(None, 1024, 256)	16640	['activation_43[0][0]']
batch_normalization_44 (BatchNormalization)	(None, 1024, 256)	1024	['conv1d_30[0][0]']
activation_44 (Activation)	(None, 1024, 256)	0	['batch_normalization_44[0][0]']
conv1d_31 (Conv1D)	(None, 1024, 512)	131584	['activation_44[0][0]']
batch_normalization_45 (BatchNormalization)	(None, 1024, 512)	2048	['conv1d_31[0][0]']
activation_45 (Activation)	(None, 1024, 512)	0	['batch_normalization_45[0][0]']
global_max_pooling1d_7 (GlobalMaxPooling1D)	(None, 512)	0	['activation_45[0][0]']
dense_25 (Dense)	(None, 512)	262656	['global_max_pooling1d_7[0][0]']
batch_normalization_46 (BatchNormalization)	(None, 512)	2048	['dense_25[0][0]']
activation_46 (Activation)	(None, 512)	0	['batch_normalization_46[0][0]']
dense_26 (Dense)	(None, 256)	131328	['activation_46[0][0]']
batch_normalization_47 (BatchNormalization)	(None, 256)	1024	['dense_26[0][0]']
activation_47 (Activation)	(None, 256)	0	['batch_normalization_47[0][0]']
dense_27 (Dense)	(None, 4096)	1052672	['activation_47[0][0]']

Fig. 10. pointNet summary

IV. Experiments

We clearly explained detailing of our experiment setups to evaluate our methods clearly above and we achieved below results.

The resulted model for pointNet is compiled, using the sparse_categorical_accuracy as metrics and loss as sparse_categorical_crossentropy. The Adam optimizer is used with a learning rate of 0.001.The model is then trained using fit function and 20 epochs. The code is broken down as follows: The PointNet architecture is used for processing point cloud data. It consists of two main components: the T-Net and the shared MLP network.

The T-Net is responsible for transforming the input features into a canonical representation. It utilizes three convolutional layers with ReLU activation (with dimensions 64, 128, and 1024) to extract local features and learn a transformation matrix. This matrix is then multiplied with the input point cloud to transform it.

The shared MLP network is applied to each point in the transformed point cloud for each point in point cloud. This network consists of two convolutional layers (with dimensions 64 and 64) to extract local features. The resulting matrix has dimensions of nx64.

Afterward, another T-Net layer is used to further transform the features. This is followed by an MLP network with three convolutional layers (with dimensions 64, 128, and 1024) and ReLU activation to extract global features. The resulting matrix has dimensions of nx1024.

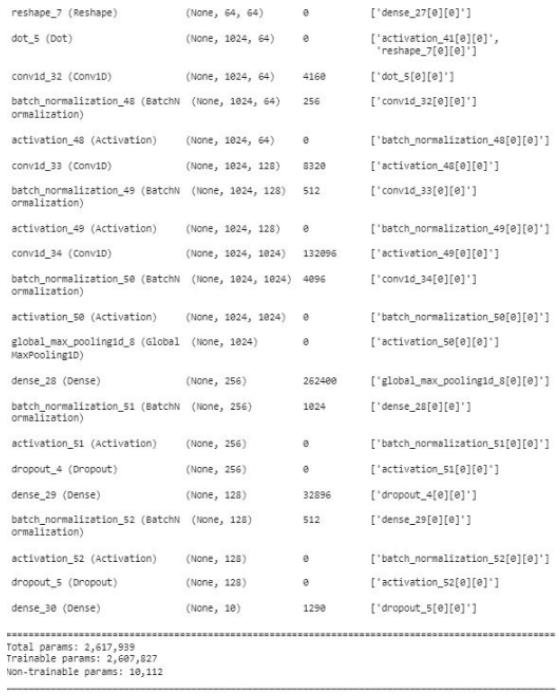

Fig. 11. pointNet summary

A Maxpooling layer is then applied to obtain a global feature vector. This vector is fed into fully connected layers for classification or other downstream tasks. In the case of object classification, a softmax layer is used to predict class probabilities for the input point cloud.

Overall, PointNet ensures consistent extraction of local features and spatial transformations across the entire point cloud by sharing the same MLP layers. Orthogonal regularization is employed to improve the orthogonality among the weight vectors of the neural network, reducing redundancy and enhancing overall performance.

The table below represents the performance of different models using various sample sizes. Each row corresponds to a specific model, while the columns represent the sample sizes: 512, 1024, and 2048.

TABLE II
RESULTS

Model	*512 samples*	*1024 samples*	*2048 samples*
MLP Classifier	24%	25.4%	22.6%
CNN	88.8%	74.2%	72.50%
PointNet	72.1%	74.9%	72.50%
Random Forest Classifier	77.2%	76.1%	75.7%

- Random Forest Classifier achieved accuracies of 77.2%, 76.1%, and 75.7% for sample sizes 512, 1024, and 2048, respectively.
- MLP Classifier attained accuracies of 24%, 25.4%, and 22.6% for the corresponding sample sizes.
- CNN obtained accuracies of 88.8%, 74.2%, and 72.5% for the sample sizes mentioned.
- PointNet achieved accuracies of 72.1%, 74.9%, and 72.0% for the respective sample sizes.

Compared to all the methods MLP got less accuracy this is because MLP Struggles to handle datasets with higher dimentinality and also MLP models are prone to overfitting when number of samples is limited and we got CNN with higher accuracy for 512 sample size.

V. Discussions and Conclusions

When we are doing predictions on pointNet model if the sample size increases for example from 512 to 1024 or 2048 the accuracy of the prediction power will reduce because of potential overfitting. Overfitting occurs when a model performs very well on training data and fails to generalize well on new unseen data. This may also happen when the training data is noisy or biased for example the number of samples increases the complexity of the model also increasing more sample means more input features. If model capacity is not adjusted it can lead to overfitting. The model might start memorizing the training samples instead of learning general patterns. Therefore the higher number of samples may introduce more noise or outliers into the data. We used the Regularization technique means applying regularization methods such as dropout to prevent overfitting and data augmentation means applying various transformations such as rotating images to achieve better results.

References

[1] H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. ICCV2015.
[2] Siamak Ravanbakhsh, Jeff Schneider, Barnabas Poczos. Deep Learning with sets and point clouds.
[3] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. CVPR 2017.
[4] A. Garcia-Garcia, F. Gomez-Donoso†, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, J. Azorin-Lopez. PointNet: A 3D Convolutional Neural Network for Real-Time Object Class Recognition.
[5] Shuaifeng Zhi, Yongxiang Liu, Xiang Li, Yulan Guo Towards real-time 3D object recognition: A lightweight volumetric CNN framework using multitask learning Computers and Graphics (Elsevier)
[6] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, Leonidas Guibas. Learning Representations and Generative Models for 3D Point Clouds, arXiv 2017.