DEEP LEARNING

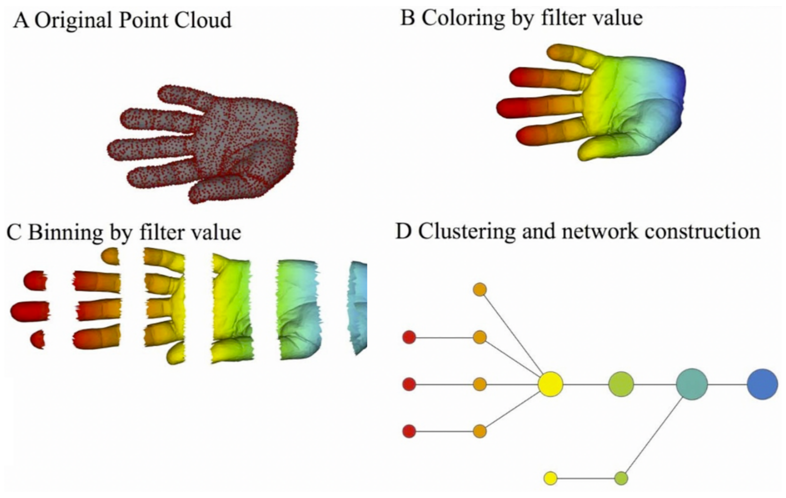
3D Object Classification using Graph Neural Networks

1 - Introduction

In this project, ModelNet10 dataset is used, which consists of 3D CAD models from 10 categories. The models are represented as .off files, which were converted into graphs. These graphs were then inputted into a Graph Neural Network (GNN) created using the PyTorch Geometric library. The GNN has two hidden convolution layers and a global average pooling layer, which classifies the object.

2 - Algorithm To Convert Point Cloud into Graphs

The conversion process involves several steps. First, the point cloud is filtered using one of the axis values. Then, binning is performed with the filtered values with overlap, and each bin is clustered. Each cluster is represented as a node, and if two clusters intersect, there is an edge between them.


A Original Point Cloud
B Coloring by filter value
C Binning by filter value
D Clustering and network construction

Images from [Carlsson et al, 2013]

2.1 ModelNet10 Dataset into Graphs

The conversion process begins with the extraction of vertices and faces from the .off file. For each face, all distances between the pair of points are measured. If that distance exceeds a certain threshold, it adds n - 1 points between those points. The vertices extracted and vertices added by processing each face are concatenated.

After the vertices are filtered using an axis (x, y, or z), binning is done with overlap using the filtered values. DBSCAN is then used to create clusters. Each cluster is converted into a node with 7 features: mean of x, y, z, standard deviation of x, y, z, and density (i.e., the number of points in the cluster divided by the total points). Two nodes share an edge if their clusters intersect with each other.

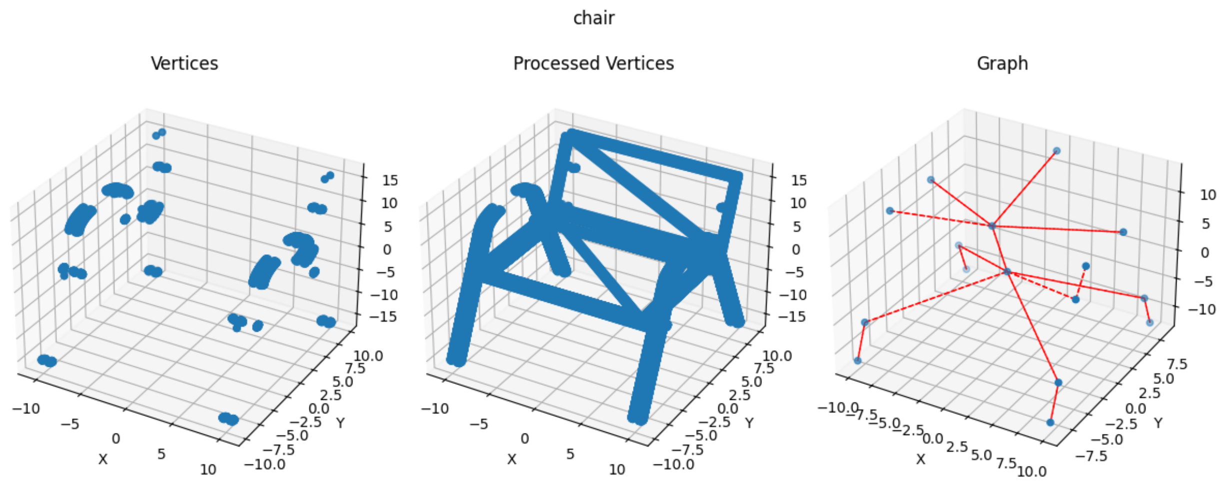
3 - Conversation and Training Process

3.1 The Conversation Process

In the conversion process follows a specific set of steps to transform the 3D models into graph representations:

1. Point Addition: For each face in the 3D model, it measures the distance between each pair of points. If the distance exceeds 10, it adds 99 points between those points. This step helped us capture more details from the 3D model.
2. Vertex Filtering: I filtered the vertices using the z-axis. This filtering step was crucial in reducing the complexity of the 3D model while retaining the essential features.
3. Binning: I performed binning on the filtered vertices using 7 bins. An overlap of 20% was allowed between the bins to ensure that it did not miss any important features that could fall on the edges of the bins.

4. Clustering: I used the DBSCAN clustering algorithm with an eps value of 1 and a minimum points requirement of 5. This step grouped the vertices into clusters, which were then represented as nodes in the graph.



3.2 The Training Process

The training process involved several steps and considerations:

1. Model Configuration: The Graph Neural Network (GNN) was configured with two hidden layers, each of size 64. This configuration was chosen to balance the model's capacity to learn complex patterns in the data and the computational efficiency.
2. Loss Function and Optimizer: The Cross Entropy loss function was used to quantify the model's performance. The Adam optimizer was chosen for its efficiency and effectiveness in updating the model's weights during backpropagation.
3. Training Loop: The model was trained with a batch size of 16 and a learning rate of 0.001. These hyperparameters were selected to ensure stable and efficient learning.
4. Results: After 100 epochs, the GNN achieved an accuracy of 81% on the test dataset. This result demonstrates the effectiveness of the model in classifying 3D objects.

Section 4: Future Works

The promising results from this project open up several avenues for future work. Here are some areas to explore:

Hyperparameter Tuning

While the current model configuration and training process have yielded satisfactory results, there is always room for improvement. Conducting extensive hyperparameter tuning on both the model and the graph conversion process. This includes but is not limited to:

- Model Hyperparameters: The variation of different configurations of the Graph Neural Network, such as varying the number of hidden layers and their sizes. Exploring These deepers networks with regularization terms such as L1, L2 weight decay, batch normalization and dropouts to optimize the training process.
- Graph Conversion Parameters: The process of converting 3D models into graphs involves several parameters, such as the number of bins, the overlap percentage, and the number of points to add. Experimenting with different values for these parameters to see might improve the quality of the graphs and, consequently, the performance of the model.

Experimenting with Different Clustering Algorithms

I used DBSCAN for creating clusters in the graph conversion process. However, there are many other clustering algorithms available, each with its own strengths and weaknesses. Experimenting with different clustering algorithms and tuning their parameters can yield better results.

Experimenting with Different Filter Functions

The filter function plays a crucial role in the graph conversion process. I currently filtered vertices using an axis (x, y, or z). However, there might be other, more effective ways to filter these vertices. Experimenting with different filter functions to create better graph representations of the objects.

By exploring these areas, further improve the performance of our 3D object classification model and contribute to the advancement of Graph Neural Networks in the field of 3D object recognition.

5 - Conclusion

In conclusion, this project demonstrated the potential of Graph Neural Networks (GNNs) in the field of 3D object classification. By converting 3D models from the ModelNet10 dataset into graph representations and training a GNN on these graphs, it achieved an accuracy of 81% on the test dataset.

The conversion process, which involved filtering vertices, binning, and clustering, played a crucial role in the success of the project. The use of DBSCAN for clustering and the addition of points based on distance thresholds proved to be effective strategies.

However, there is always room for improvement. Future work will focus on hyperparameter tuning, experimenting with different clustering algorithms, and exploring different filter functions. By continuing to refine and optimize our approach, we might push the boundaries of what is possible with GNNs in 3D object classification.

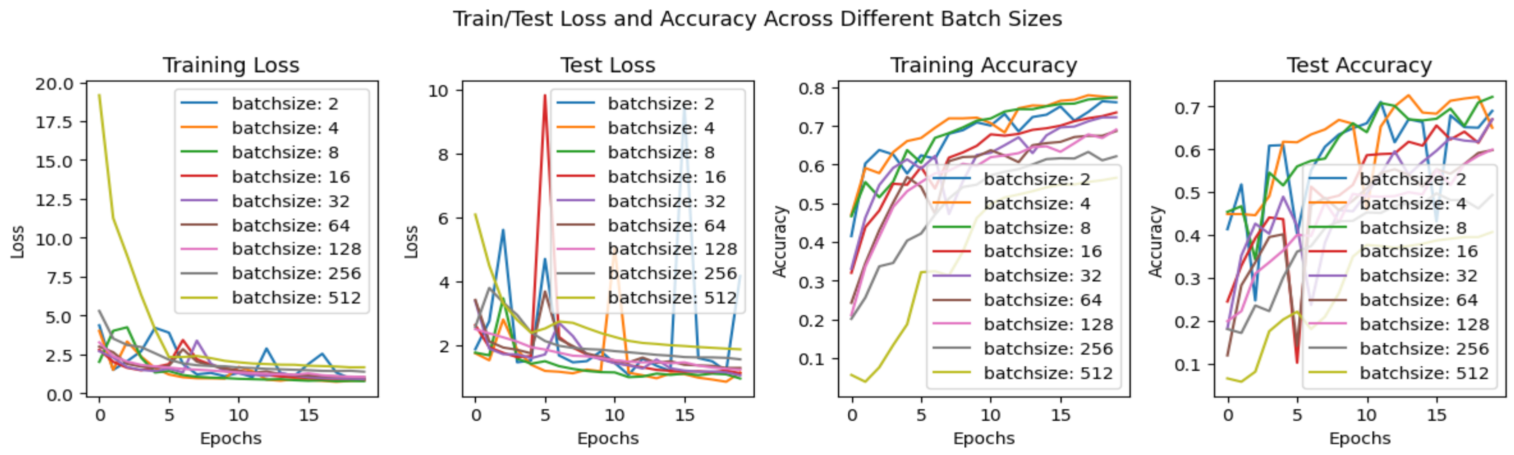
This project serves as a testament to the power of GNNs and their potential applications in various fields. As I continue to explore and innovate, I look forward to uncovering new possibilities and contributing to the advancement of this exciting area of research.

Appendix: Further Experiments

The results presented above were obtained from the following experiments that I conducted:

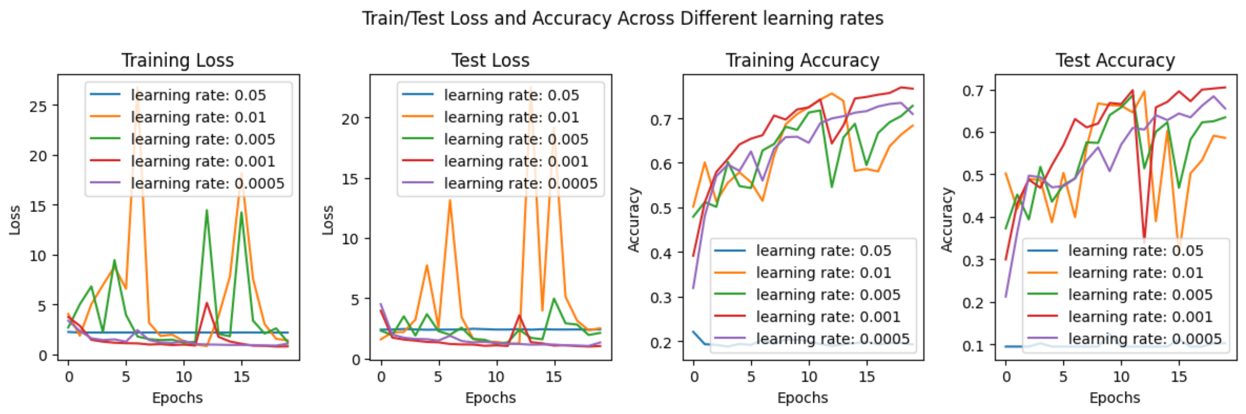
Experiment 1: Different Batch Sizes

I experimented with different batch sizes, including 2, 4, 8, 16, 32, 64, 128, 256, and 512. I found that the smallest batch size worked well. Since the model wasn't complex, it wasn't too time-consuming. The figure below shows four graphs of Training Loss, Test Loss, Training Accuracy, and Test Accuracy across 20 epochs with a learning rate of 0.001.



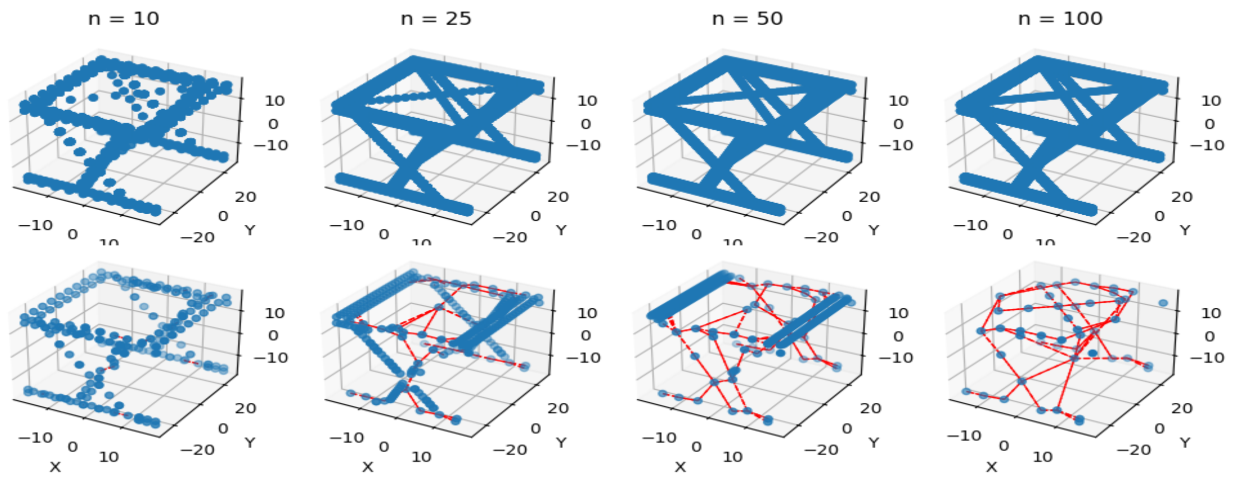
Experiment 2: Different Learning Rates

I also experimented with different learning rates to find the ideal rate. The rates tested were 0.05, 0.01, 0.005, 0.001, and 0.0005, with a batch size of 16. The figure below shows four graphs of Training Loss, Test Loss, Training Accuracy, and Test Accuracy across 20 epochs.



Experiment 3: Number of Points Added

I also experimented with adding different numbers of points during the conversion process. Which leads to the following observation that adding more points (n > 50) during the conversion process leads to smaller connected graphs. This observation suggests that the number of points added can significantly impact the structure of the resulting graph and, consequently, the performance of the Graph Neural Network. The plot below shows different graphs created with different amounts of



points added.

Experiment 4: Different Filter Axes

These experiments also revealed that filtering vertices using the z-axis leads to slightly better results than using the y and x-axes. This finding indicates that the choice of filter axis can influence the quality of the graph representations and the effectiveness of the GNN. I plan to continue experimenting with different filter axes to optimize the graph conversion process.