Final Project Report
Introduction to Geometric Deep Learning

Anya Sukachev and Sergiy Horef

October 6, 2024

1 Task 1: Point Cloud Classification

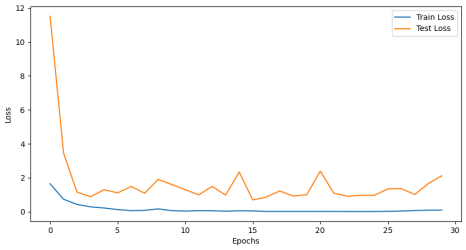
Disclaimer: we have tried as hard as we could to have reproducible results, however, there are small things that torch uses for which we are not able to set the seed.
All of the results we show here are reproducible up to 0.05 and show small fluctuations to either side.

1.1 Method and Architecture choice

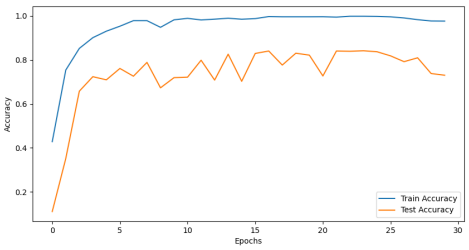
We have started by trying to implement the PointNet++ model, the link to which was provided in the task pdf file. We had to make some changes in the code provided there, as some parts plaindly didn't work, and some worked in an incorrect way (for example, the accuracy calculation returned results higher than 1).

When we had a working model, we have run a wandb hyperparameter optimization, and have found that the best model had these paramaters: sample of 512 points for each cloud and batches of size 128. (Other parameters can be found in the 'q1_code.py' file.)

After 30 epochs, this model was achieving the following:
Train loss: **0.0149**, Train accuracy: **0.9947**
Test loss: **1.1218**, Test accuracy: **0.8271**
This test accuracy result is approximately the best that this model can achieve.
Here are the graphs of the loss and accuracy on both train and tests:



(a) Loss of PointNet++



(b) Accuracy of PointNet++

Figure 1: Loss and Accuracy of PointNet++

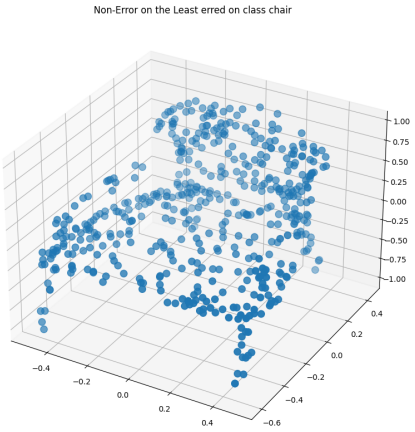
We also provide the number of errors per each class, which will be useful to compare this model against our final choice of GBNet.

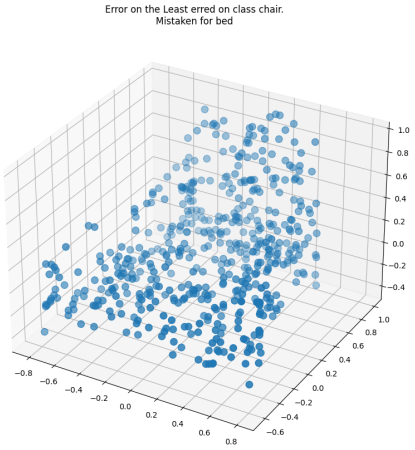
class	bathtub	bed	chair	desk	dresser
num of errors	19	11	1	41	19
class	monitor	night stand	sofa	table	toilet
num of errors	8	44	6	3	5

Table 1: Number of errors per each class in PointNet++

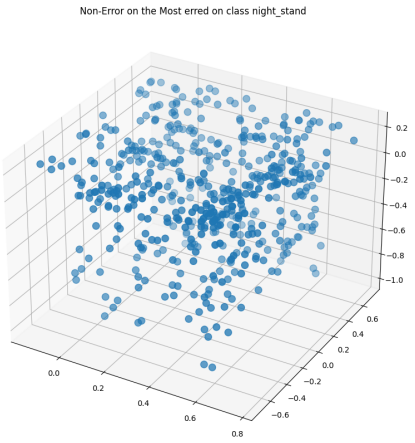
Least erred class - **chair**. Most erred class - **night stand**.

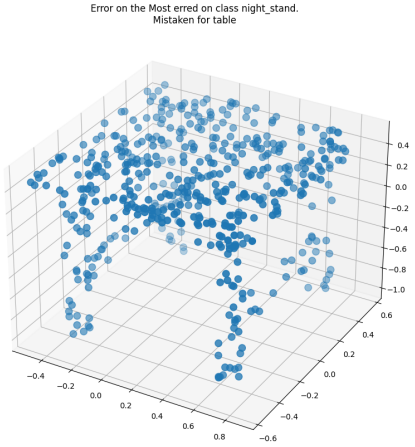
Examples of point clouds that our model errer/did not err upon:





(a) Correctly classified on the least erred class (b) Incorrectly classified on the least erred class





(c) Correctly classified on the most erred class (d) Incorrectly classified on the most erred class

Figure 2: Examples of correctly and incorrectly classified point clouds for the least and most erred classes. For PointNet++

These results are not great, so we have search for an additional model to try.

We have found a paper presenting a more complex approach than that of PointNet++ called GBNet (Geometric Back-projection Network) - paper link. Creators of the paper have also provided the implementation of the model itself, which we have fit to work on our task - original github link.

As far as we understand from a brief familiarization with the paper - the model operates by enriching the geometric information of points in low-level 3D space explicitly while also applying CNN-based structures to learn local geometric context implicitly.
At the low level, the model uses the Geometric Point Descriptor module to enhance the representation of scattered point clouds. This module operates by identifying explicit geometric relations among points. For each point p_i in the point cloud, the model searches for its two nearest neighbors p_{j1} and p_{j2}, and uses these points to form a triangle. This triangle helps in estimating various geometric properties such as edges and normal vectors.
In the high-level space, the model incorporates the Attentional Back-projection Edge Features Module (ABEM). This module is designed to refine the feature learning process by integrating an error-correcting feedback mechanism. The ABEM captures local geometric context and projects it back into the network, allowing for the correction and enhancement of the feature extraction process.

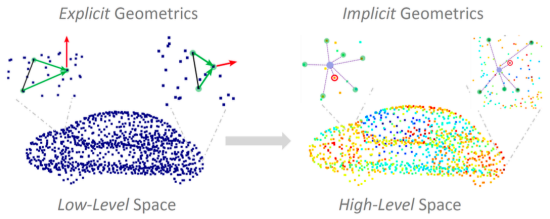


Figure 3: Enriching geometric features. In low-level space, we explicitly estimate geometric items as prior knowledge for the network e.g., edges (green vectors), normals (red vectors). In high-level space, we aggregate neighbors (green points) to implicitly capture both prominent (red points) and fine-grained geometric features (purple points). (Fig 1 in the paper.)

Here is an image of the full architecture:

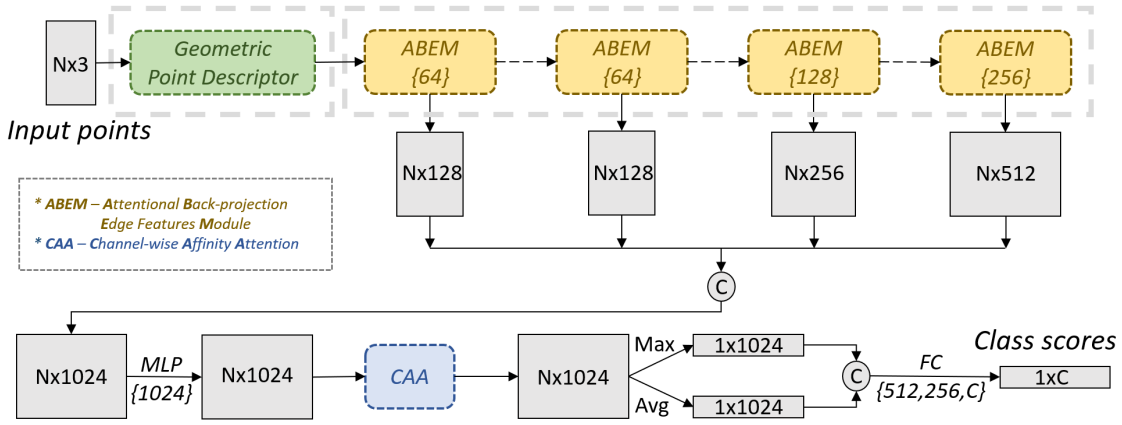
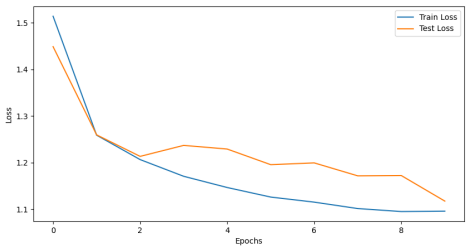


Figure 4: The Geometric Point Descriptor offers more low-level geometric clues for subsequent high-level geometric feature learning in cascaded ABEMs, representing the point features in multiple scales of embedding space by aggregating local context. The CAA module refines the learned feature map to avoid channel-wise redundancy. Finally, we use the concatenation of max-pooling and average-pooling results, as well as fully connected layers to regress the class scores. (Figure 2 in the paper)

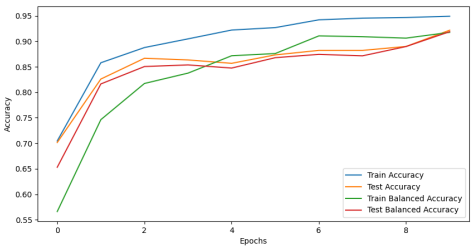
For this model we had a very constrained set of possible hyperparameters, due to the limited size of the memore available on the GPU. We have started with the parameters chosen for PointNet++, and gradually lowered the batch size, until we got something that worked. The final parameters are: sample of 512 points for each cloud and batches of size 32.

We have run this model for 10 epoch in total, and it produces much better results:
Train loss: **1.0959**, Train accuracy: **0.9491**, Train Balanced Accuracy: **0.9176**
Test loss: **1.1174**, Test accuracy: **0.9218**, Test Balanced Accuracy: **0.9189**
In this model we preset also the balanced accuracy measure, which presents an average accuracy for all classes (that is, it accounts for the fact that not all classes are equally presented in the test set.)

Here are the graphs of the loss and accuracy on both train and tests:



(a) Loss of GBNet



(b) Accuracy of GBNet

Figure 5: Loss and Accuracy of GBNet

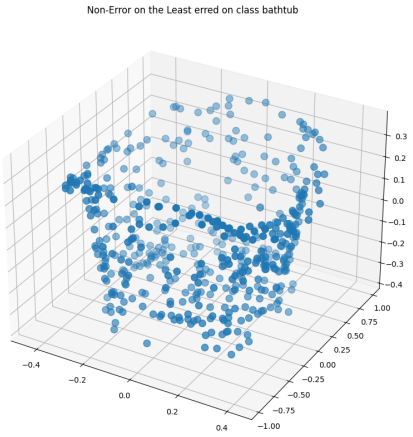
We also provide the number of errors per each class:

class	bathtub	bed	chair	desk	dresser
num of errors	0	2	0	24	19
class	monitor	night stand	sofa	table	toilet
num of errors	2	18	3	5	1

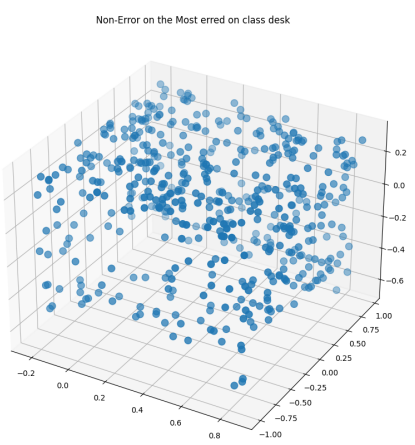
Table 2: Number of errors per each class in GBNet

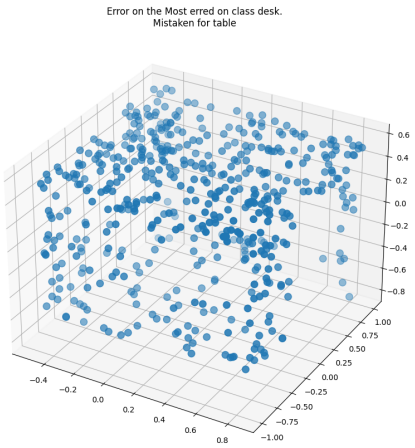
Least erred class - **bathtub**. Most erred class - **desk**.

Examples of point clouds that our model errer/did not err upon:



(a) Correctly classified on the least erred class





(b) Correctly classified on the most erred class (c) Incorrectly classified on the most erred class

Figure 6: Examples of correctly and incorrectly classified point clouds for the least and most erred classes. For GBNet

We can see that our model struggles the most with the desk, dresser and night stand classes. (In the figure above, we can see an example of a misclassification - cloud of a desk is mistaken for a table.)
This is due to the simple fact that at this resolution (512 points) all of them look very much alike. Additionally, table and desk name a very similar set of objects, and are therefore hard to distinguish.

On the other hand, bathtub is quite different from everything else, and therefore can be easily classified by our model. (It also has much smoother angles, which is one of the things explicitly accounted for in the GBNet.)

2 Task 2: Graph Classification

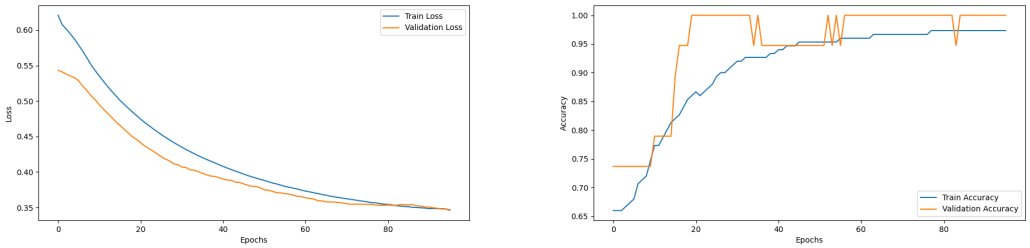
2.1 Method and Architecture choice

We have started by trying the GAT model we have seen in class.
In order to use this model for graph classification, we have added a readout function (function that takes embeddings of all nodes and creates a unified embedding based on some aggreagation function), and a fully connnected layer which takes the result produced by the readout function and returns a list of probabilities of each class.

We have created a variable implementation, such that we could dinamically define the number of GNN layers, number of attention head, readout aggregation function, and the hidden dimention of the fully connected network.

Because from our initial runs (with some randomly chosen parameters) we were already able to get to a validation accuracy of 100%, we did not feel the need to try a different model altogether, and instead used wandb to make hyperparameter optimization.

All variables we have used in the optimization can be found in q2_wandb.py, but in total we have looked at 1800 different combinations of parameters. Later, we chose only the parameter sets which resulted in validation accuracy of 100% and all of those we sorted by the minimal validation loss, and overall stability.
Our final model has the following results on the training and validation sets after 96 epochs of training:
Train loss: **0.3468**, Train accuracy: **0.9733**
Validation loss: **0.3457**, Validation accuracy: **1.0000**



(a) Loss on Train and Validation (b) Accuracy on Train and Validation

Figure 7: Plots of the performance of our final model for Task 2

2.2 Discussion of Results

We have a very small dataset, which includes only 150 training points, 19 validation and 19 testing points.
It is relatively easy to get a high accuracy on 19 points in total, and the complexity of the model is expressive enough to achive an almost perfect accuracy on train. On the contrary, we can see small jumps on the graph of the validation accuracy, which result from the fact that even a single mistake is equivalent to around 5.23% in accuracy.

One thing which we would like to point out in addition to everything that we have said before - the training and validation sets have an unequal number of 0s and 1s, with ration of 1s to 0s as approximately 2:1. Therefore we have manually set the weight in the loss function we have used to give twice more weight to the 0 label.