PROJECT REPORT : 3D POINT CLOUD CLASSIFICATION

Chupin Clément
Université Clermont-Auvergne
M2 PAR

Hubert Villeneuve
Université Clermont-Auvergne
M2 PAR

March 11, 2023

1 Introduction

A point cloud is a kind of data set composed of spatial coordinates for multiple points that can be obtained via 3D laser scanning. Typically, point clouds will represent scenes or specific objects. While it may be obvious what a point cloud represents to the human eye, classification through computing is a complex task. As such, it is a good application for artificial intelligence.

The goal of this project is to classify various objects formed by 3D point clouds using neural networks. Essentially, by feeding a labelled data set to a neural network, it attempts to find features representing each class so that it can generalize to an unlabelled data set. One network studied in this project is PointNet. While it has the capacity to separate a single point cloud in various segments, as seen in figure 1.1, only the general classification will be used.

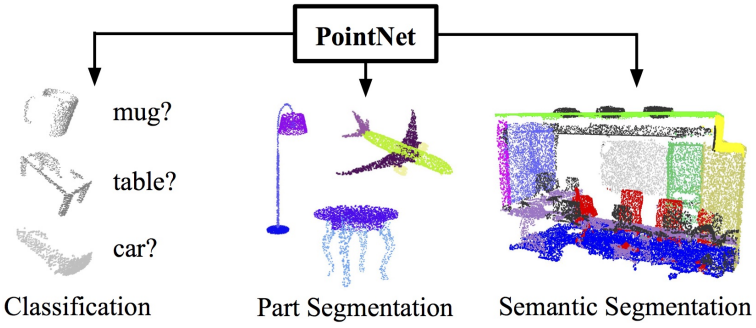

Figure 1.1: Explanation of classification, segmentation [1]

In this report, section 2 makes an initial exploration of neural networks. Classification will be detailed in sections 3, using a basic MLP network, and section 4, using PointNet. Data augmentation techniques will also be discussed in this section.

2 MLP for 2D Data

There exists online tools to get an initial grasp on the workings of a neural network and TensorFlow offers one of them [2]. This tool simulates a multi layered perceptron (MLP) network for binary classification. The two types of data points are represented by orange and blue colors, and the background color shows the classification done by the network. Ideally, the data points and background colors should match. Various options can be modified to see the effects on the network behavior. Some of the options used include a learning rate of 0.001, ReLU activation functions and two hidden layers of eight neurons each.

As seen in figures 2.1 and 2.2, convergence is reached relatively quickly and the data is overall well classified. For the spiral dataset in figure 2.3, more neurons are used since convergence is not as quick. Even then, the accuracy is not ideal for the dataset. This can be explained by the fact that a clean decision boundary in the shape of a spiral is not as simple to approximate from the provided features.

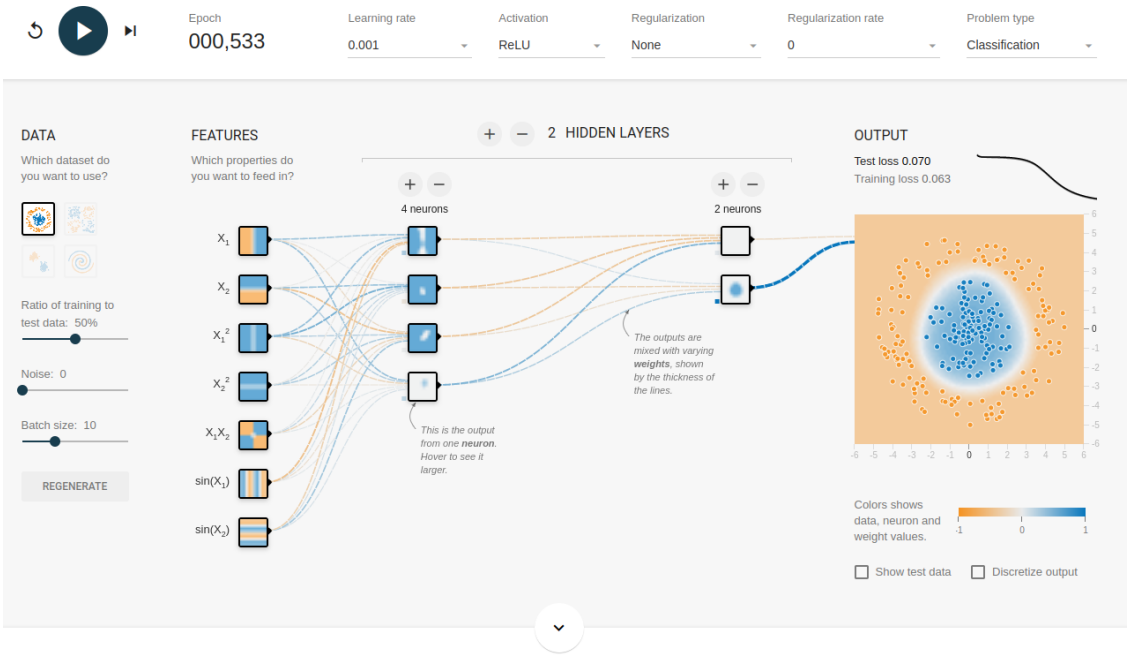

Figure 2.1: TensorFlow tool for concentric data

Using a cloned repository of the TensorFlow tool that has been modified to enable more options [3], a sine activation function can now be used [4]. Using the spiral dataset, the network differentiates the two classes much better and with fewer neurons as seen in figure 2.4. In any case, classical uses of neural network for binary classification rarely have dataset in spiral shapes, so this problem is purely academic.

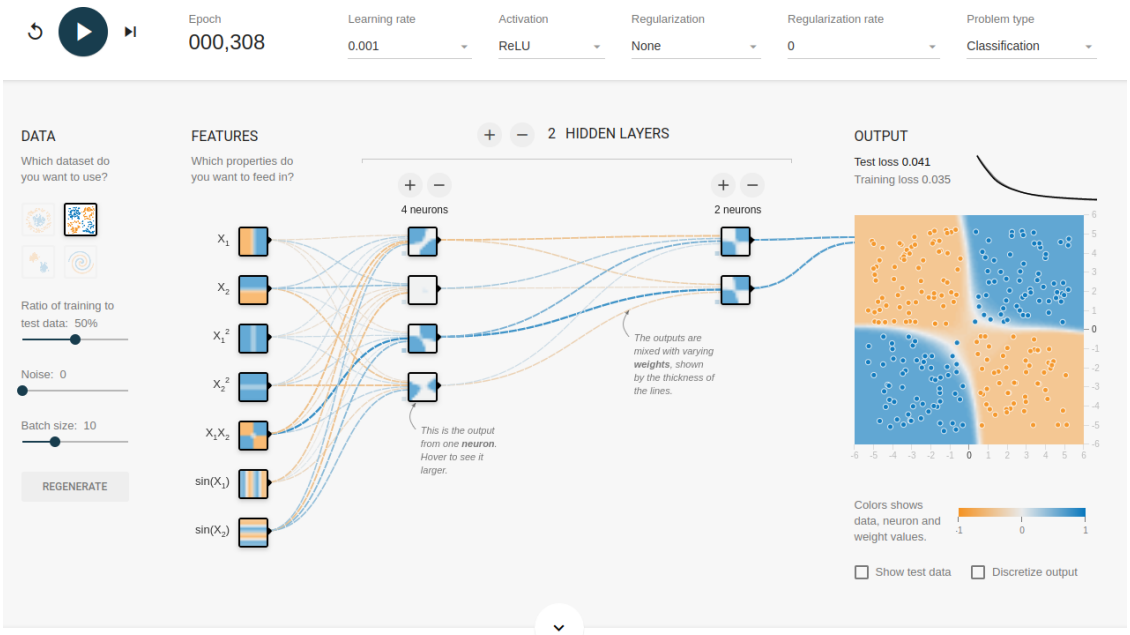

Figure 2.2: TensorFlow tool for 2x2 data

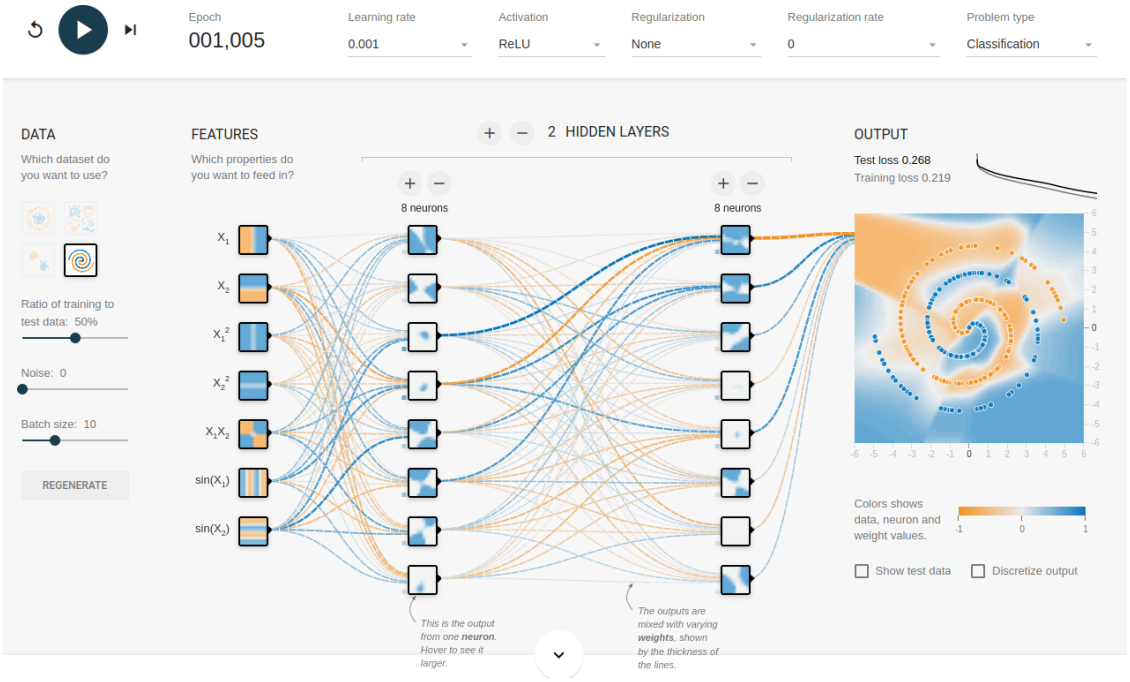

Figure 2.3: TensorFlow tool for spiral data

3 PointMLP

Moving on to the 3D point cloud object classification problem, ModelNet10 is the studied dataset, used over ModelNet40 for being simpler. It contains 3991 3D point clouds for training and 908

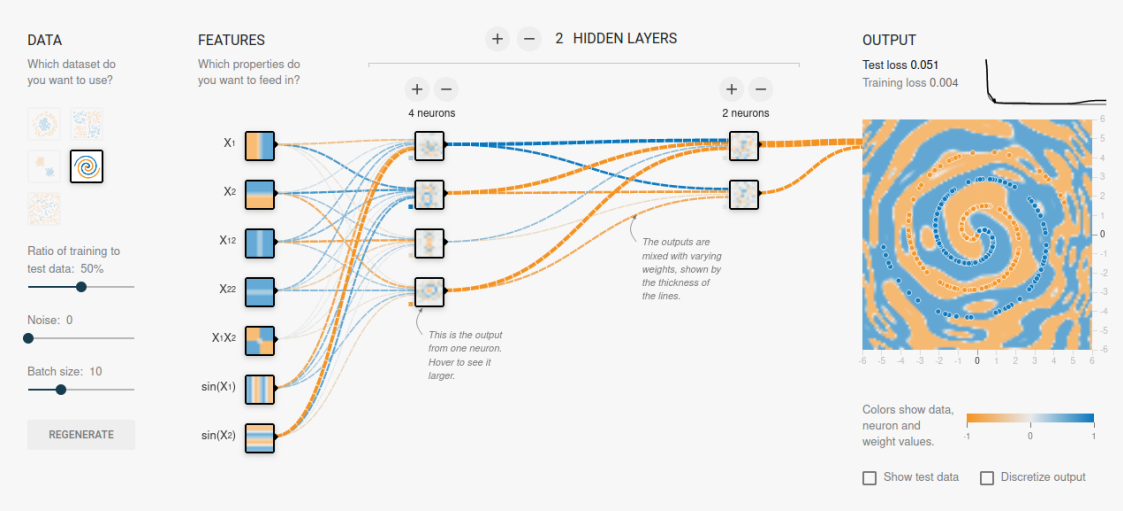

Figure 2.4: Modified TensorFlow tool for spiral data

for testing. Figure 3.1 shows a point cloud example representing a chair from the dataset. Other examples can be found in appendix B.

Figure 3.1: ModelNet10 3D point cloud example

The neural network to which ModelNet10 will be fed is a 3-layered MLP network: PointMLP. The general parameters used for this exercise are found in table 3.1. The number of epochs was limited to 50 to reduce the computation time. In addition, the loss function used is CrossEntropyLoss for being good at classification with multiple classes.

After 50 epochs, the classification accuracy on figure 3.2 is 43.8% for the training dataset and 22.1% for the testing dataset. This difference can be explained by the fact that the network manages to find a solution for the training, but has a difficult time generalizing this solution for the testing.

Table 3.1: General parameters

Parameter	Value
Epochs	50
NbPoints per point cloud	1024
Dataset	ModelNet10
Train size	3991
Test size	908
Learning rate	0.001
Loss function	CrossEntropyLoss

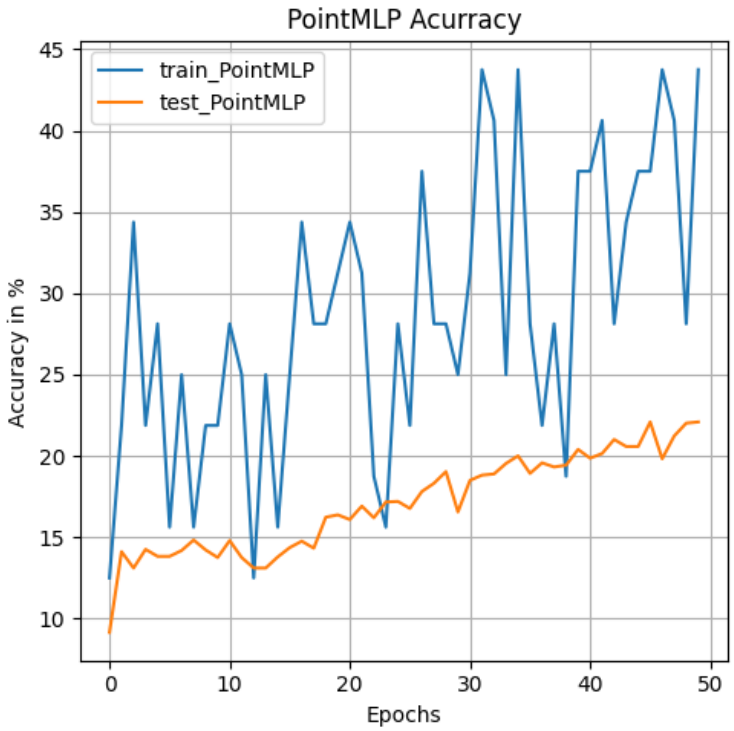

Figure 3.2: Accuracy curve of the PointMLP on train and test dataset

The accuracy values are quite low and most likely due to the simplistic nature of the network. While using weight dropout helps to better generalize and reduce overtraining, flattening the 1024 points at the start is a questionable choice. The result of the flatten operation is a column vector filled with the XYZ coordinates for each point while not being labeled as such. It is unclear whether or not the coordinates lose their meaning in this process.

4 PointNet

For this exercise, the PointNet network is used to classify the ModelNet10 dataset. This network is more complex than PointMLP and its architecture is shown in figure 4.1. This exercise is split in three parts. The first and second use PointNet without and with the T-Net feature. The third part is to test the effects of data augmentation on the ModelNet10 dataset.

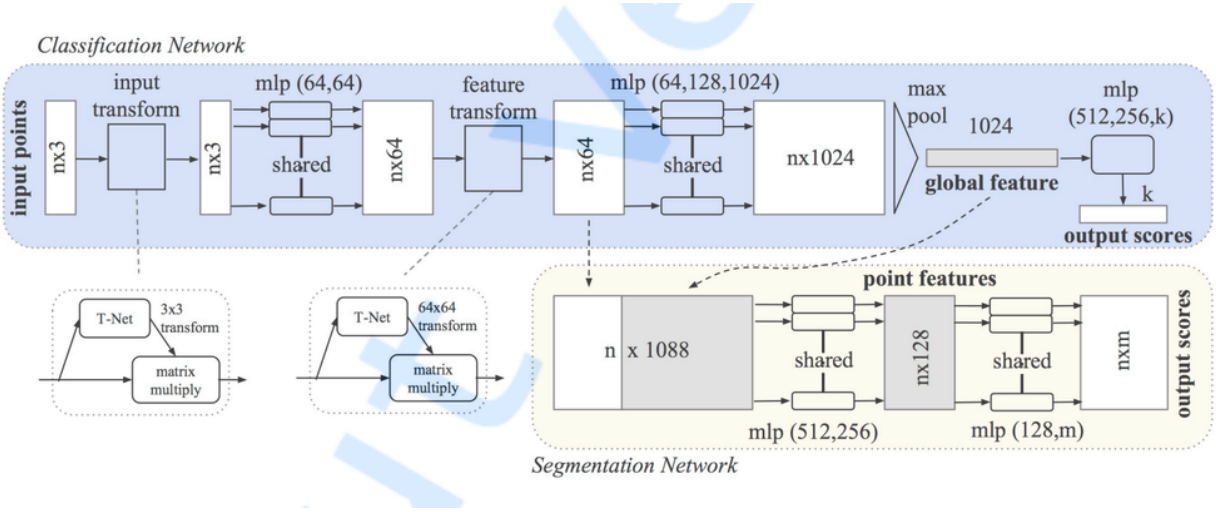

Figure 4.1: PointNet Architecture

4.1 Basic version of PointNet

The first step is to try a basic version of PointNet, one that doesn't use the T-Net feature. The result of this network testing and training can be found in figure 4.2. The training accuracy is 96.9% and the testing accuracy, 88.6%. The difference between these values has the same explanation than in section 2. These values are much higher than for the PointMLP network as the structure is more complex, allowing for better classification capacities. The network starts by extracting features from the raw data in order to treat them. The different point cloud objects activate different patterns for the neural network's understanding.

The testing and training accuracies for each object are plotted in figures 4.3 and 4.4, and the values are found in table 4.1. It can be observed that some objects are more difficult to classify than others. This could be solved by attempting to fine tune the loss function.

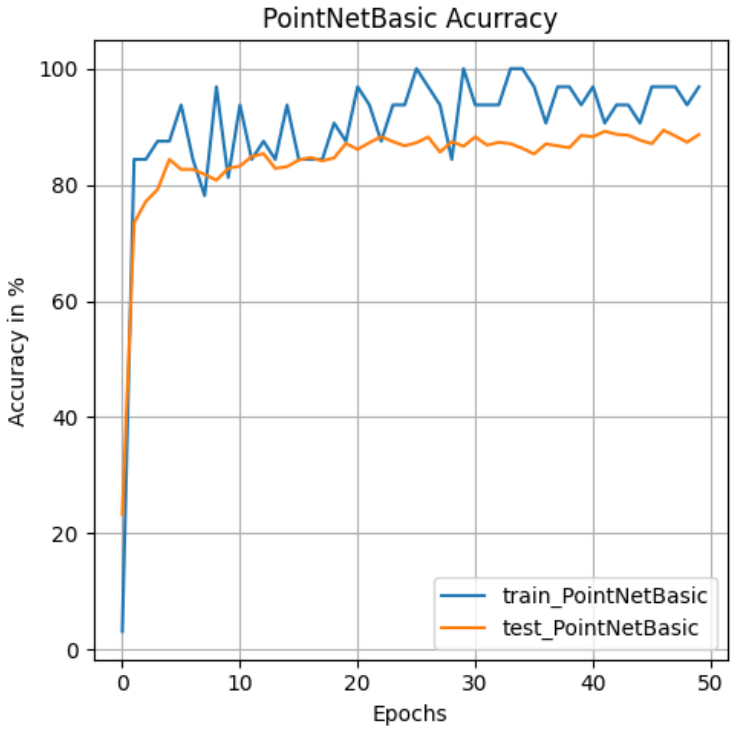

Figure 4.2: Accuracy curve of the PointNetBasic on train and test dataset

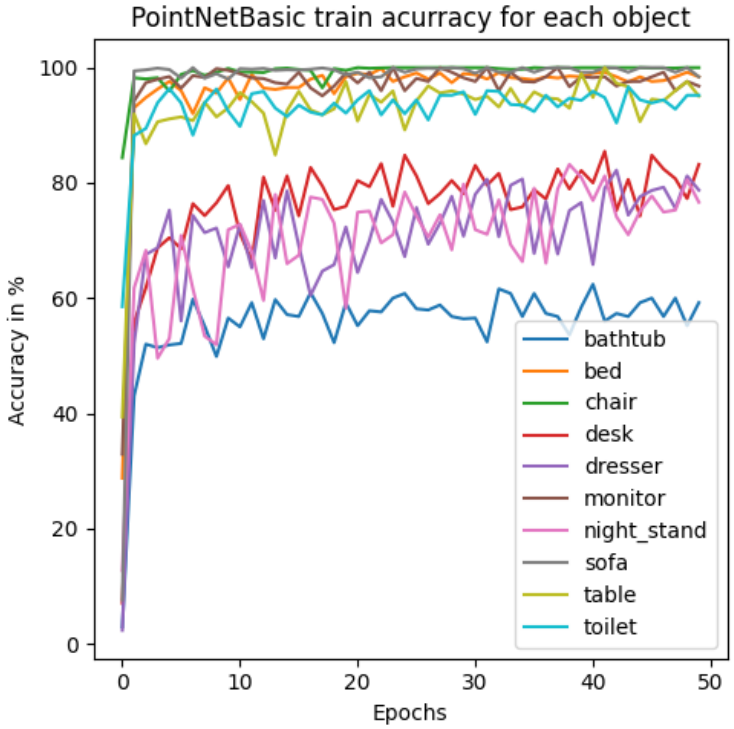

Figure 4.3: Training accuracy curves for all objects

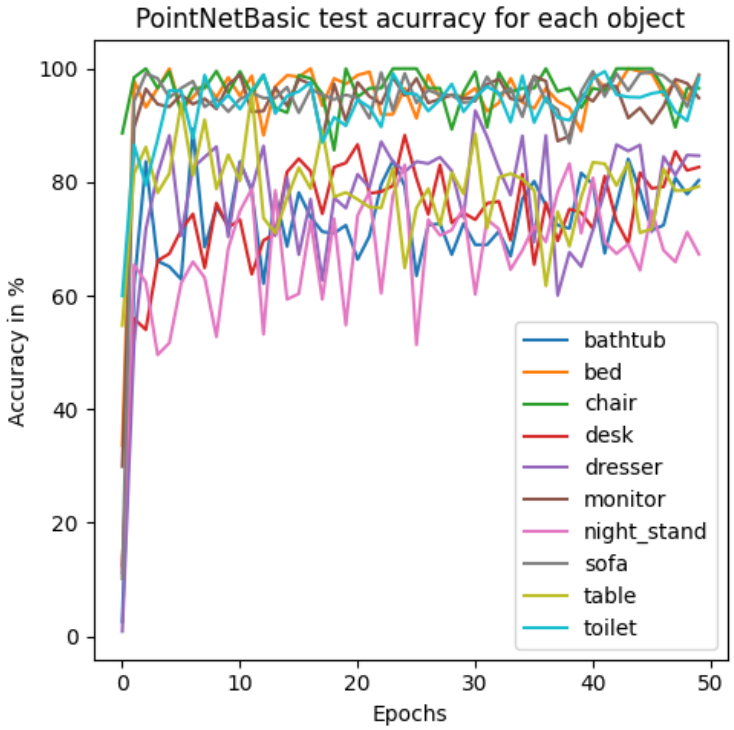

Figure 4.4: Testing accuracy curves for all objects

Table 4.1: Detection of each object

Object	Accuracy train	Accuracy test
bathtub	59.2	80.3
bed	98.4	98.8
chair	100	96.6
desk	83.2	82.7
dresser	78.7	84.7
monitor	96.8	94.8
nightstand	76.6	67.3
sofa	98.4	98.9
table	95.1	79.3
toilets	95.2	98.3

4.2 PointNet with T-Net network

T-Net is an addition that takes the raw point cloud as input and outputs 3x3 matrices that are analogous to rotation matrices. The purpose of this is to find a better alignment for the point cloud

objects so that they become more understandable for the neural network. For example, if one point cloud corresponds to an upside-down chair, T-Net would realign it by placing the four legs on the bottom.

The training accuracy is 93.8% and the testing accuracy, 85.5%, after 50 epochs as seen in figure 4.5. The results are superimposed to those of the network minus T-Net on figure 4.6. While the results are overall very similar, PointNet with T-Net has a slower start. Indeed, this network has the goal of finding a realignment for the raw point cloud, which takes time, in addition to classifying them. Then, if the resulting accuracy values for both networks are similar, logic dictates that T-Net has little impact on the point cloud. This can be explained by the fact that the alignment of the raw point cloud is already good enough for the network to understand.

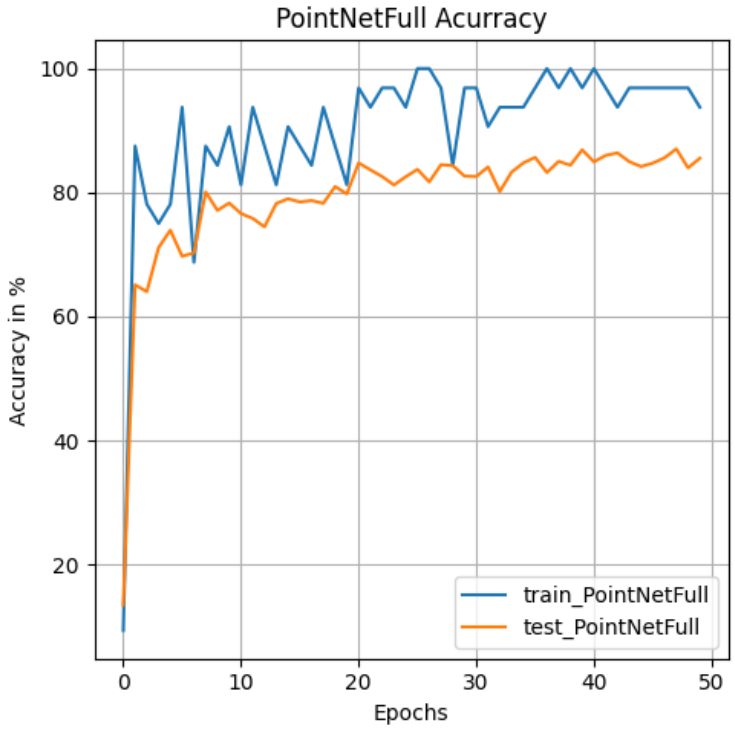

Figure 4.5: Accuracy curve of the PointNetFull on train and test dataset

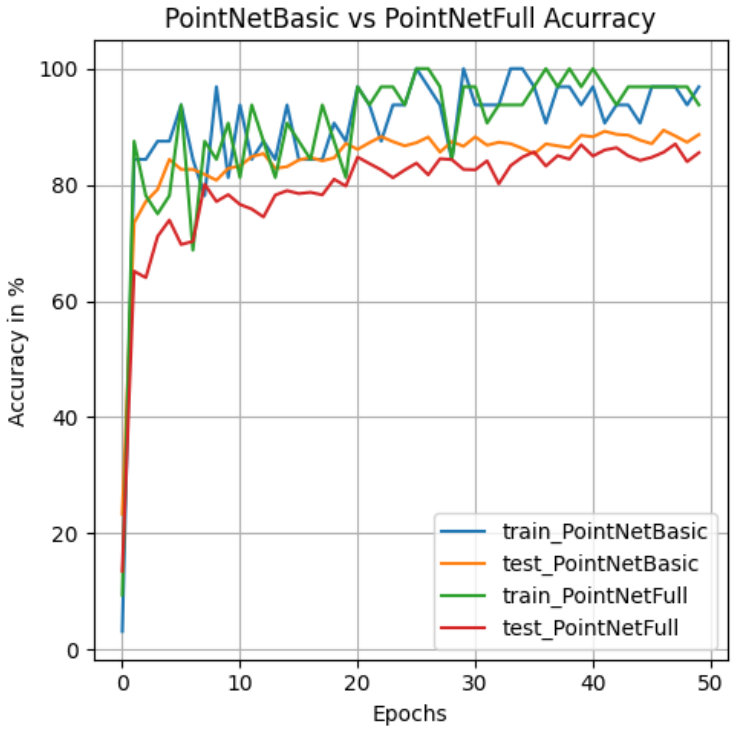

Figure 4.6: Accuracy curve on train and test dataset of PointNetFull vs PointNetBasic

4.3 Data augmentation for 3D data

If T-Net attempts to realign the point clouds to get better classification results, perhaps modifying them in other ways, through data augmentation, can improve the network's performance. For this exercise, two methods are attempted: dimension reduction and voxelization.

4.3.1 Dimension reduction

From the discussion on using T-Net, one hypothesis that arises is that the point clouds are well aligned and centered with respect to the vertical Z-axis. In this case, maybe the normal distance of each point with respect to the Z-axis is one of the features the network is looking for? By doing so, a dimension reduction in induced by having only the normal distance and the Z-coordinate. The transformation applied to the XYZ-coordinates is detailed in equation 1:

$$DimRedux(x, y, z) = \begin{cases} x = \sqrt{(x)^2 + (y)^2} \\ y = 0 \\ z = z \end{cases} \tag{1}$$

If one applies this transformation to a 3D point cloud, the output can be seen in figure 4.7. Here, a 3D point cloud representing a computer screen is reduced to the 2D scatter plot. The result of this

transformation, added to the basic ModelNet, can be found in figure 4.8. Once again, the accuracy is very similar. Only the testing accuracy for the transformation applied is lower. One potential explanation is that the normal distance to the Z-axis is not a major feature. The transformation applied even has a negative impact in this case.

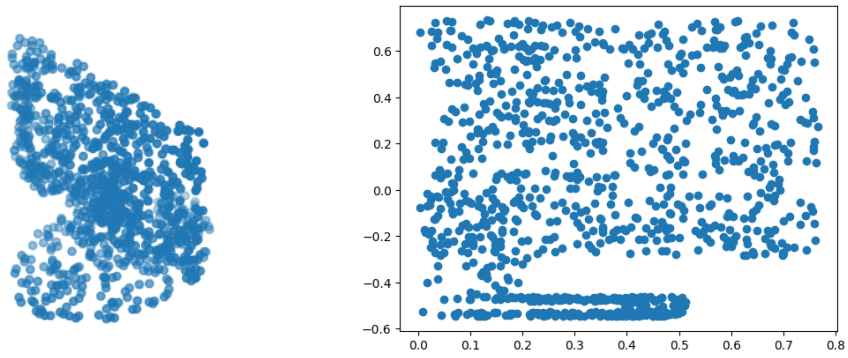

Figure 4.7: Raw and augmented point cloud

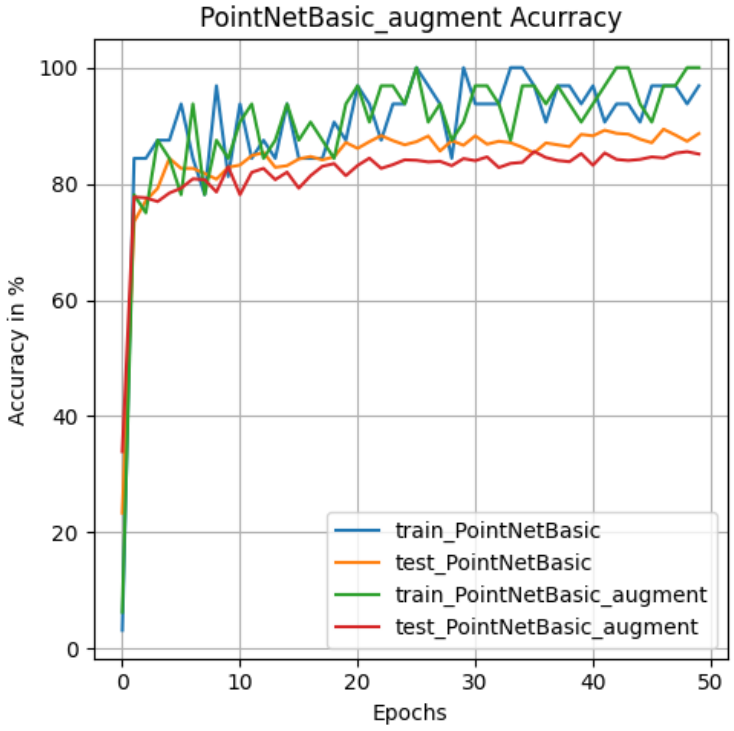

Figure 4.8: Accuracy curve of the PointNetBasic on train and test, for the basic and the augmented dataset by dimension reducing

4.3.2 Voxelization

The second operation is to perform a voxelization on the point clouds in order to reduce the number of points involved and attempt to getting a faster and more efficient network. Voxelization implies that the points are placed in voxels. For each voxel, the points centroid is kept as a new point and all centroids form the new point cloud. It is important to note that to keep the same number of inputs to the network, the coordinates of the 1024 points are modified rather than deleting the points. Since a max pooling operation is used on ModelNet, having multiple points with the same coordinates shouldn't have an impact. The actual operation performed to voxelize is detailed in equation 2. Figure 4.9 shows the result of the voxelization to a 3D point cloud representing a chair.

$$Voxelize(x, y, z) = \begin{cases} x = round(x * Vsize)/Vsize \\ y = round(y * Vsize)/Vsize \\ z = round(z * Vsize)/Vsize \end{cases} \tag{2}$$

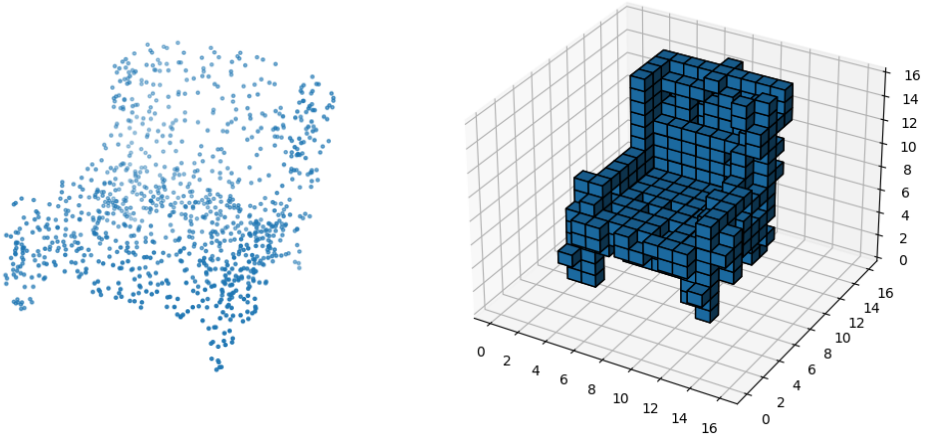

Figure 4.9: Raw and voxelized point cloud

The accuracy comparison for the voxelization operation and the basic ModelNet network can be found in figure 4.10. The accuracy for both training is the same. Testing accuracy is visibly lower for the voxelized dataset. It could be that reducing the object resolution makes it slightly less recognizable to PointNet. In fact, if even larger voxels are used, the accuracy becomes even lower as shown in figure 4.11.

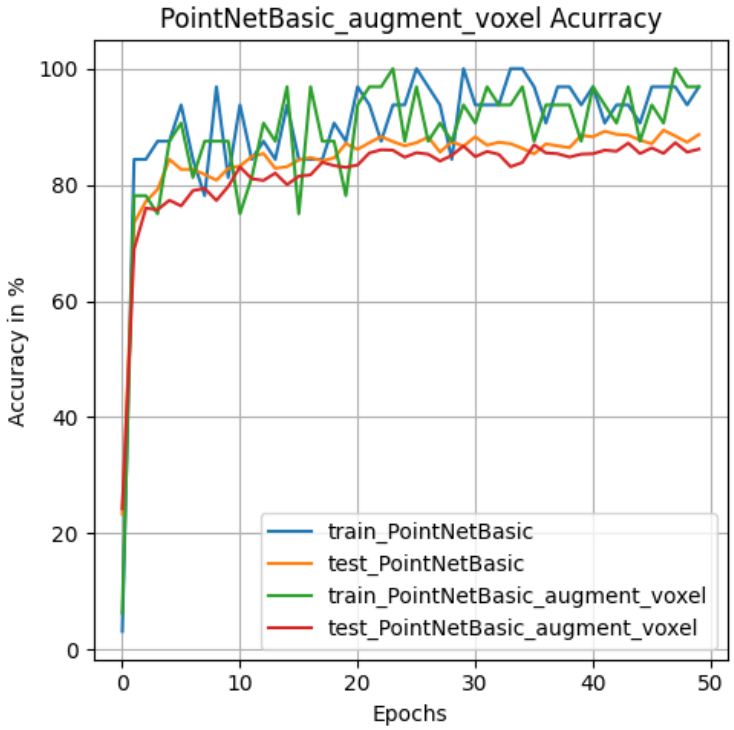

Figure 4.10: Accuracy curve of the PointNetBasic on train and test, for the basic and the augmented dataset by voxelization

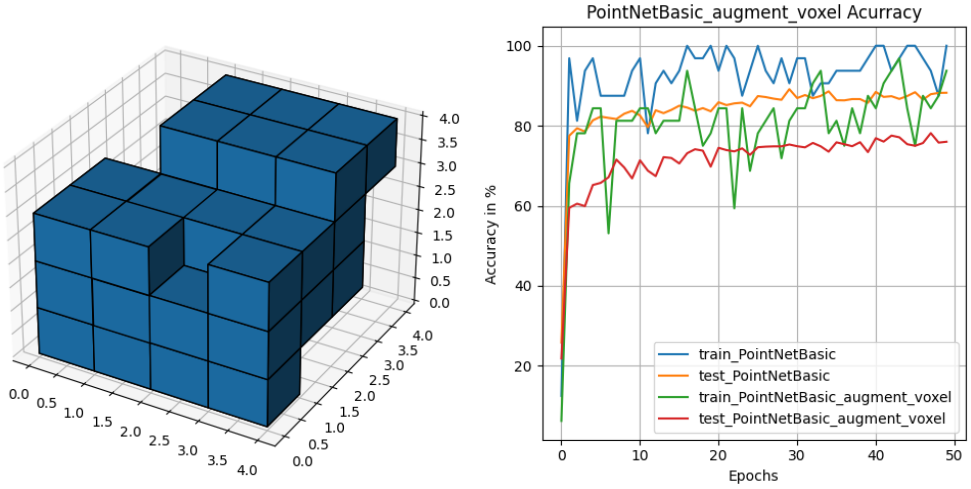

Figure 4.11: Larger voxels and corresponding accuracy

5 Conclusion

Classifying 3D point clouds using artificial intelligence was detailed in this report through three sections: understanding neural networks, classifying using a basic MLP network and classying

using the more complex PointNet network. The results showed that the basic version of PointNet was better at classification than the basic MLP network. Furthermore, that version of PointNet did not fare any better with using T-Net, likely due to how the data set is made, nor with using data augmentation.

Further perspectives arise from using complex PointNet. One could add the surface normals to the points, in addition to their coordinates, as additional information that could help classification accuracy for the network. Another data augmentation operation could be to flatten each point cloud on a plane representing the main axis of symmetry.

References

[1] Charles Ruizhongtai Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *CoRR* abs/1612.00593 (2016). arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

[2] Daniel Smilkov and Shan Carter. *Tensorflow — Neural Network Playground*. URL: http://playground.tensorflow.org (visited on 01/14/2023).

[3] David Cato. *Neural Network Playground*. URL: https://dcato98.github.io/playground (visited on 01/14/2023).

[4] Vincent Sitzmann et al. "Implicit Neural Representations with Periodic Activation Functions". In: *Proc. NeurIPS*. 2020.

Appendices

A Network codes

```
PointMLP(
  (fc_1): Linear(in_features=3072, out_features=512, bias=True)
  (bn_1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_2): Linear(in_features=512, out_features=256, bias=True)
  (dropout_1): Dropout(p=0.3, inplace=False)
  (bn_2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_3): Linear(in_features=256, out_features=1024, bias=True)
  (bn_3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

Figure A.1: PointMLP

```
PointNetBasic(
  (fc_1): Conv1d(3, 32, kernel_size=(1,), stride=(1,))
  (bn_1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_2): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_3): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_4): Conv1d(32, 128, kernel_size=(1,), stride=(1,))
  (bn_4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_5): Conv1d(128, 512, kernel_size=(1,), stride=(1,))
  (bn_5): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (mp): MaxPool1d(kernel_size=512, stride=512, padding=0, dilation=1, ceil_mode=False)
  (fc_6): Linear(in_features=1024, out_features=256, bias=True)
  (bn_6): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_7): Linear(in_features=256, out_features=128, bias=True)
  (dropout_1): Dropout(p=0.3, inplace=False)
  (bn_7): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_8): Linear(in_features=128, out_features=10, bias=True)
)
```

Figure A.2: PointNet Basic

```
PointNetFull(
  (input_transform_1): InputTransform(
    (t_net): Tnet(
      (fc_1): Conv1d(3, 32, kernel_size=(1,), stride=(1,))
      (bn_1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (fc_2): Conv1d(32, 64, kernel_size=(1,), stride=(1,))
      (bn_2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (fc_3): Conv1d(64, 256, kernel_size=(1,), stride=(1,))
      (bn_3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (mp): MaxPool1d(kernel_size=256, stride=256, padding=0, dilation=1, ceil_mode=False)
      (fc_4): Linear(in_features=1024, out_features=128, bias=True)
      (bn_4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (fc_5): Linear(in_features=128, out_features=64, bias=True)
      (bn_5): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (fc_6): Linear(in_features=64, out_features=9, bias=True)
    )
  )
  (fc_1): Conv1d(3, 32, kernel_size=(1,), stride=(1,))
  (bn_1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_2): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_3): Conv1d(32, 32, kernel_size=(1,), stride=(1,))
  (bn_3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_4): Conv1d(32, 128, kernel_size=(1,), stride=(1,))
  (bn_4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_5): Conv1d(128, 512, kernel_size=(1,), stride=(1,))
  (bn_5): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (mp): MaxPool1d(kernel_size=512, stride=512, padding=0, dilation=1, ceil_mode=False)
  (fc_6): Linear(in_features=1024, out_features=256, bias=True)
  (bn_6): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_7): Linear(in_features=256, out_features=128, bias=True)
  (dropout_1): Dropout(p=0.3, inplace=False)
  (bn_7): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc_8): Linear(in_features=128, out_features=10, bias=True)
)
```

Figure A.3: PointNet Full

B ModelNet10 objects

Table B.1: Number of each object

Object	Number in train dataset	Number in test dataset
bathtub	106	50
bed	515	100
chair	889	100
desk	200	86
dresser	200	86
monitor	465	100
nightstand	200	86
sofa	680	100
table	392	100
toilets	344	100

Figure B.1: All objects