Neural Networks and Deep Learning Project 3

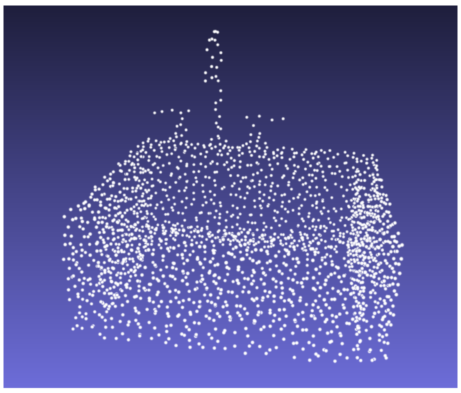
(Cheng Zhong 16307110259)

1 Save 3D Point Cloud

In this task, we are required to save the point cloud data into a *obj* file.
From the Wikipedia article, the *obj* file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary.
The storage of obj files needs to follow a certain format, such as

```
# List of geometric vertices, with (x, y, z [,w]) coordinates, w is ...
    optional and defaults to 1.0.
v 0.123 0.234 0.345 1.0
v ...
...
```

and using the visualization tool **MeshLab** , we can visualize the *obj* file as follows.


(a) Wash Basin

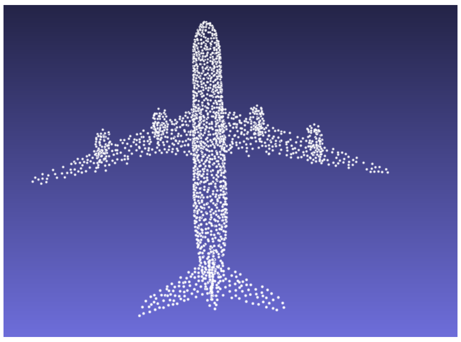

(b) Plane

Figure 1: Pointcloud visualization

2 Training and Testing

The point cloud has the following characteristics.
First, geometrically, the order of points does not affect its representation of the overall shape in space. For example, the same point cloud can be represented by two completely different matrices. We hope that regardless of the order of the point cloud, the model can get the same features from it.
Second, after a point cloud undergoes a rigid transformation in space, such as rotation or translation, the coordinates of the point cloud will change. Sometimes, in order to facilitate the processing of the point cloud (such as classification, etc.), we need to rotate it to the front or side. At this time, the point cloud data (N*3 matrix) obtained by the model will be completely different. We hope that the transformation of the point cloud will not affect the results of the model.

Traditional processing methods for point cloud have two major categories:
Project point cloud data onto a two-dimensional plane. This method does not directly process the three-dimensional point cloud data, but first projects the point cloud to some specific angles of view and then processes them, e.g. the forward view angle and the bird's eye view angle. By combining these data from different perspectives, we can finish the cognitive task of point cloud data. The most representative algorithms on this field are MV3D and AVOD.
Divide point cloud data into voxels with spatial dependencies. This method divides the three-dimensional space to introduce spatial dependence to the point cloud data, and then uses 3D convolution and other methods for processing. The accuracy of this method depends on the fineness of the three-dimensional space segmentation, and the computational complexity of 3D convolution is relatively high.
In this project, I will test the **cls_3d**, **cls_3d with BatchNorm** and **pointnet** algorithm.

2.1 cls_3d

cls_3d is a simple network consisting of 3 convolutional layers, a max-pooling layer and a fully connected layer, using ReLU as the activation function. The network structure, accuracy curve and loss curve is as follows. We can reach the accuracy of **0.7385** on validation set and **0.7065** in test set.

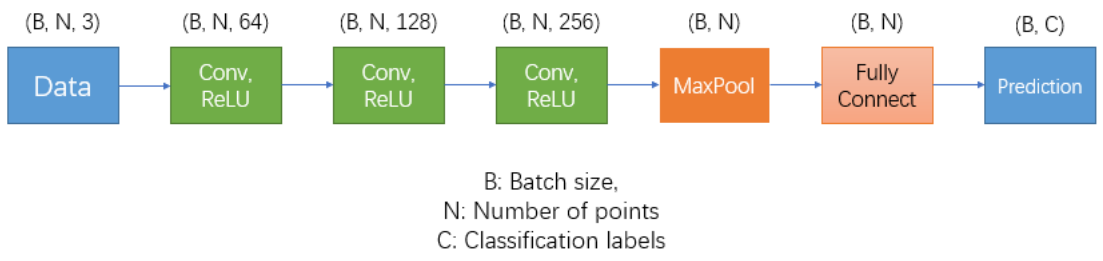
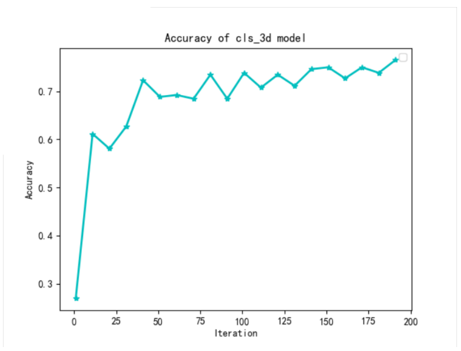
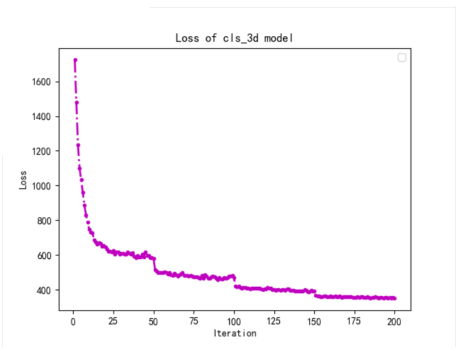


Figure 2: Structure of cls_3d



(a) Accuracy on validation set



(b) Loss

Figure 3: Training process of cls_3d

2.2 PointNet

PointNet and PointNet++ networks were proposed by Charles R. Qi in 2017 and published at CVPR 2016 and NIPS 2017[1]. Different from the traditional network, pointnet can learn a DxD rotation matrix that is most conducive to the classification or segmentation of the network by learning the posture information of the point cloud itself. At the same time, when the network performs a certain degree of feature extraction on each point, maxpooling layer can extract the global feature of pointcloud. The network structure, accuracy curve and loss curve is as follows. We reproduced the code in the paper and adjusted it for the data set, and it can reach the accuracy of **0.8815** on validation set and **0.8511** in test set.

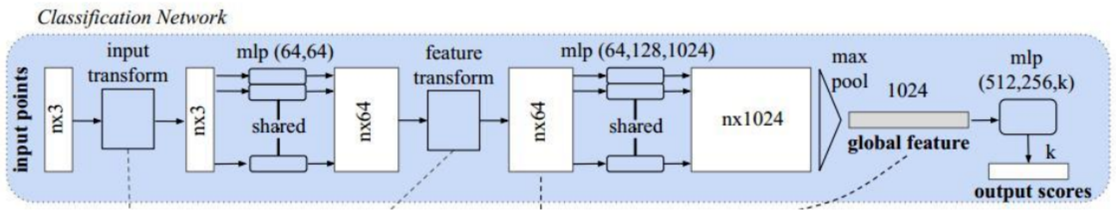
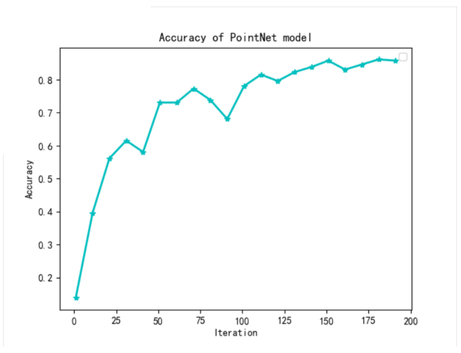
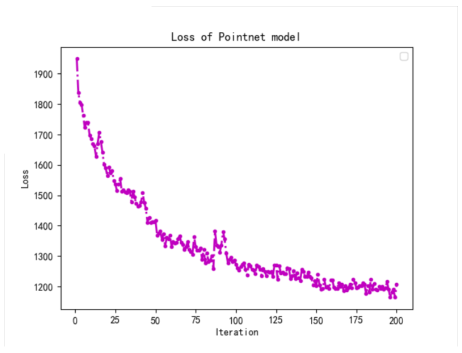


Figure 4: Structure of Pointnet



(a) Accuracy on validation set

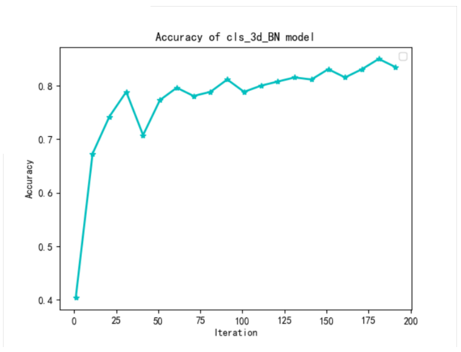


(b) Loss

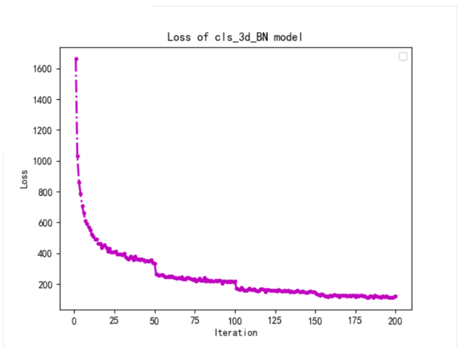
Figure 5: Training process of pointnet

2.3 Cls_BN

Pointnet achieved better performance on the test set, but the price is more parameters and training time. So, I try to add a BatchNorm layer (like what pointnet does) after each convolution layer in cls network to reduce the parameters while maintaining the correct rate. The accuracy curve and loss curve is as follows. We can reach the accuracy of **0.8500** on validation set and **0.8171** in test set.



(a) Accuracy on validation set



(b) Loss

Figure 6: Training process of cls_3d_BN

2.4 Conclusion

Here is the hyper-parameter I used in three experiences above. After comparing the three models, I think the cls_3d_BN model can achieve relatively better results after using fewer parameters.
Of course, if we make more detailed adjustments to the hyper parameters and do some data augmentation, it may bring better results, but this is outside the scope of our inquiry. As can be seen from the performance, without other feature information, point-net's performance isn't better than simple neural network.

Table 1: Hyper-parameters

Batchsize	Stepsize	Epoch	Epoch to decay
4	10^{-2}	200	50
Optimizer	Loss	Activation	
Adam(0.5, 0.999)	CrossEntropyLoss	ReLU	

Table 2: Comparison of two network

Network	Accuracy on valid	Accuracy on test	parameter	Training time
Cls_3d	0.7385	0.7065	51880	10min for 200 Epoch
Cls_3d_BN	0.8500	0.8171	52776	10min for 200 Epoch
Pointnet	0.8815	0.8511	1614129	30min for 200 Epoch

3. Q&A

3.1 Colored .obj file

If we want to store the color data of each vertex in the obj file, the only thing we need to do is to add the corresponding RGB value (0-1) after the coordinates. e.g.

```
# List of geometric vertices, with (x, y, z, R, G, B) coordinates, w is ...
    optional and defaults to 1.0.
v 0.123 0.234 0.345 R G B
v ...
...
```

The visualization result is:

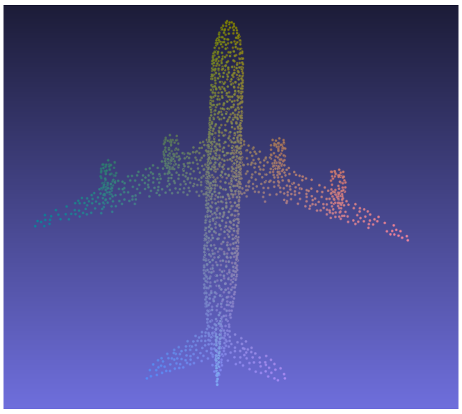


Figure 7: Colored plane

3.2 Differences between neural networks for point clouds and for ordinary images

Here are the differences between pointcloud and ordinary image: pointcloud is disorder, sparseness and has limited information. The difference between the two method is mainly the way of extracting data features
Therefore, when processing images with neural networks, we can use normalized format processing methods to extract the features, such as convolution kernels with shared weights, while we need to design a function independent of the order of the input data for point cloud.
In general, researchers use the common symmetric function in deep neural networks: **Max Pooling** to solve the disorder problem, and share the network parameters to deal with changes in input dimensions.

3.3 Transform the coordinates in the three-dimensional space to correspond to the pixels of the image

The essence of this question is how to find the corresponding parts in the 3D model and the 2D image. In actual data, this feature is often more obvious, for example color, obvious boundaries can be used as matching features between the two. As for point cloud data that only contains location information, we need to start with spatial information.
From the *01-camera-models.pdf*, the transformation of the image can be achieved by multiplying its coordinates with an affine matrix. The shallow features of the image (such as lines, relative position information, etc.) do not change after the affine transformation.
In my opinion, solving this problem requires three steps: Extract the features of the image and the space model; find the best matching transformation between the two through the affine matrix and the projection matrix; determine the coordinate correspondence relationship by synthesizing the above information.
When the dataset is large, we can extract image features through convolution and pooling and match through the fully connected layer, and then match the corresponding part of the model with the image by calculating the receptive field. Through deep learning, we can find the best mapping.
But if we only have one image and one model, We can only use multi-angle projection, and then use the gradient to detect the contour of the projection and the picture, calculate

the affine matrix of the projection and the picture and calculate the accuracy, and select the transformation method with the highest accuracy.

3.4 color a 3D model utilizing the information of single image

If the coloring of the model has a general pattern (for example, the doors of car are the same color as the roof), we can calculate the a prior probability through a prior network to infer the color, otherwise we can only calculate the color of the corresponding part from image.
From 3.3,we have obtained the affine and projection matrix relationship for the model that best matches the picture. Then we can map the corresponding part of the model to the pixels of the image through transformation. For the part mapped between the pixels, we can calculate the RGB by interpolation. In this way, we can color the model utilizing the information of single image.

References

[1] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.