Report of Project3

Jiwen Zhang
16307110435

1 INTRODUCTION

Point cloud, a point set defined in 3D metric space, is one of the most important data format for 3D representation. Its gaining increased popularity as a result of increased availability of acquisition devices, such as LiDAR, as well as increased application in areas such as robotics, autonomous driving, augmented and virtual reality. Since 3D data provides rich information about the full geometry of 3D objects, it has great potential to achieve impressive results in many 3D vision tasks like classification, segmentation and detection. Many state-of-the-art models have been proposed.

In this project, we will first extract a model from those 3D points and save it as an obj file so as to better understand the storage format of a 3D point cloud. Then we will train a neural network on 3D classification task. Finally, we will answer the questions proposed by TA and also give our own thinkings.

2 SAVE 3D POINT CLOUD

OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. It is a simple data-format that represents 3D geometry alone.

A vertex can be specified in a line starting with the letter v, which is followed by $(x, y, z[, w])$ coordinates. w is optional and defaults to 1.0. For example:

```
# 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 ...
```

If we want to store the color data of each vertex in the obj file (**Question 1**), data should be put like this:

```
# List of geometric vertices
# with (x, y, z, r, g, b) coordinates
# r, g, b should lie in [0, 1]
v 0.123 0.234 0.345 0 0.7 0
v ...
```

The visualization of our 3D point cloud is shown in Fig. 2.

3 3D CLASSIFICATION

3.1 Neural Network for Point Clouds

Applying deep learning on 3D point cloud data is hugely different from applying it on 2D images (**Question 2**). It comes with many challenges [1] that won't appear in 2D image tasks. These challenges can be categorized into the following:

- **Irregularity**: Point cloud data is irregular, that is, the points are not evenly sampled across the different regions of an object/scene, so some regions could have dense points while others sparse points.
- **Unstructured**: Point cloud data is not on a regular grid. Each point is scanned independently and its distance to neighboring points is not always fixed, in contrast, pixels in images are represented on a 2 dimension grid, and spacing between two adjacent pixels is always fixed.
- **Unorderdness**: Point cloud of a scene is the set of points(usually represented by (x, y, z)) obtained around the objects in the scene and are usually stored as a list in a file. As a set, the order in which the points are stored does not change the scene represented.

In short, those challenges, especially unorderdness, construct the barrier of directly applying Convolutional Neural Networks(CNNs) on 3D point cloud. As we all know, the success of CNNs credits to the convolution operation, which enables learning on local regions in a hierachical manner as the network gets deeper. However, this manner requires ordered grids of input which point cloud data lacks.

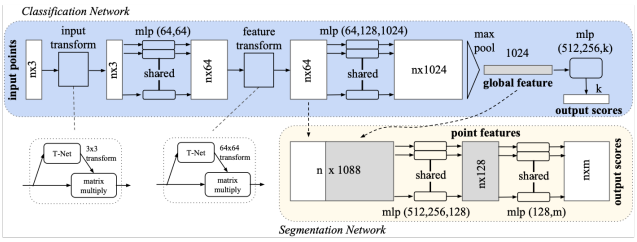

Figure 1: PointNet

3.2 PointNet and PointNet++

So how can a machine learning algorithm not depend on the order of its input? PointNet [2] gives the answer. Since the essential problem in machine learning is function approximation, the key to resolve the unorderdness is to find a CNN that can approximate a function which is invariant to the order of 3D points. PointNet did it (see Fig. 1) by using the following function

$$f(x, y, z) = max(x, y, z)$$

which is symmetric on both x, y, and z. This corresponds to the max-pooling layer in PointNet. For each tiny group of points, after a few transformations there is a maxpooling operation that combines local features with invariability. Charles R. Qi and Hao Su et al. have also proved that PointNet's network structure can fit any continuous set function. This laid the foundation for future works.

However, PointNet is not perfect. By design PointNet can not capture the local structures as the pooling operation used by PointNet is on global features, thus limiting its generalizability to complex scenes.

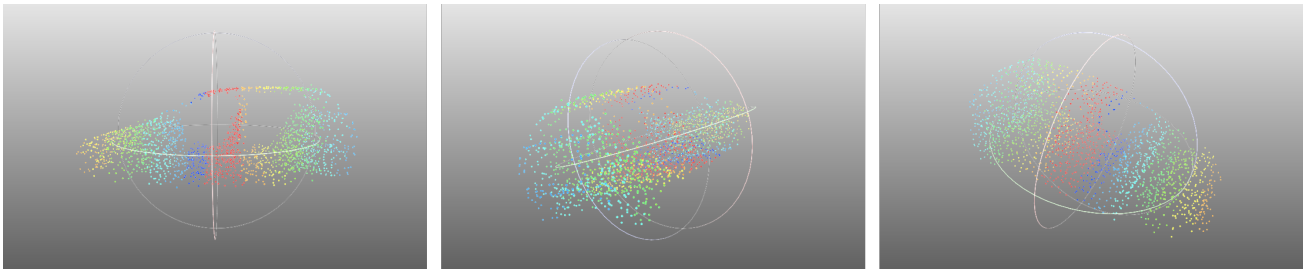

Figure 2: 3D Point Cloud

PointNet++ [3] introduces a hierarchical neural network that applies PointNet recursively on a nested partitioning of the input point set. As shown in Fig. 3, PointNet++ is composed of three core layers:

- **Sampling layer**: Use iterative farthest point sampling (FPS) to choose a subset of points $x_{i_1}, x_{i_2}, ..., x_{i_m}$ such that x_{i_j} is the most distant point from the set $x_{i_1}, ..., x_{i_{j-1}}$ with regard to the rest points. The input of this layer is N points and it returns N' indexes of selected points.
- **Grouping layer**: The input of this layer is a point set of size $N \times (d + C)$ with N' sampled centroid indexes. Given the coordinate of a centroid $n' \in \{1, 2, ..., N'\}$, define its neighbourhood as a sphere with given radius r. Local region features are extracted by randomly sampling K points in the neighborhood of centroid points. Thus, the output are groups of point sets of size $N' \times K \times (d + C)$.
- **PointNet layer**: Also called feature learning layer. It is used to extract local features abstracted by centroids and s the centroid's neighborhood.

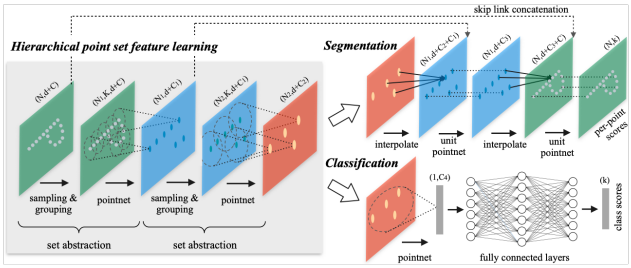

Figure 3: PointNet++

Unlike picture data distributed on a regular pixel grid with uniform density, the distribution of point cloud data in space is irregular and uneven. Although PointNet can be used to extract features locally from each point cloud, the uniformity of the point cloud in each locality is inconsistent, which may cause the learned PointNet to fail to extract good local features. For example, the farther away the LiDAR data usually becomes sparse, so in sparse places, a larger scale range should be considered to extract features. To this end, [3] proposed two grouping strategies to ensure better feature extraction, i.e. multi-scale grouping(MSG) and multi-resolution grouping(MRG).

Multi-scale grouping(MSG) is a simple but effective way to capture multi-scale patterns. It applies grouping layers with different scales followed by PointNets to extract features of each scale. Then features at different scales are then concatenated to form a multi-scale feature. However, because features need to be extracted for each scale, the increase in the amount of calculation is also very significant.

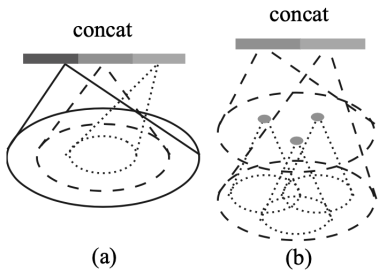

Figure 4: (a) Multi-Scale Grouping; (b) Multi-Resolution Grouping

Multi-resolution grouping(MRG) is thus proposed as an alternative approach to avoid such expensive computation. As shown in Fig. 4, the feature vector of some layer L_i is a concatenation of two vectors. The first part is extracted from the previous layer L_{i-1} through the PointNet, and the second part is obtained by directly extracting features from all raw points in in the local region through the PointNet. This makes PointNet++ have the following local pattern recognition capabilities: (a) when the point cloud density near the downsampling candidate points is large, the local features of the downsampling candidate point can be effectively extracted through the first part; (b) when the point cloud near the downsampling candidate points is sparse, then the second part can be used to assist the feature vector to obtain more reliable local features.

3.3 Implementation Deatils

In this project, we implement the PointNet++ on classification task with multi-scale grouping. To better illustrate the performance of our model, we use the sample code provided by TA as the baseline. With no data argumentation and the same parameter setting, the result on testset is shown in Tab. 1. Note that the hyperparamteter settings for both PointNet++ and baseline are exactly the same (see Tab. 2).

In fact, what we do find is that with $lr = 0.001, batch\_size = 16$, PointNet++ can quickly fit the dataset and gives a nice testing accuracy. As seen in Fig. 5, the loss function of PointNet++ is very

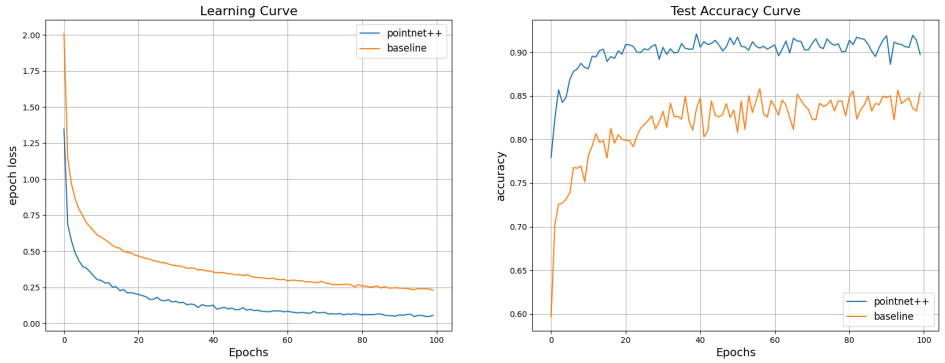

Figure 5: Learning Curve and Accuracy

Table 1: Network Performance

Accuray	After 10 epoch	After 20 epoch	Best
Baseline	0.7512	0.8002	0.8582
PointNet++	**0.8825**	**0.8979**	**0.9210**

smooth and is nearly non-decreasing in the last few epochs, indicating that the model fits well.

Table 2: Hyperparameter Settings

Batch Size	Learning Rate	Optimizer	Loss Function
16	0.001	Adam	Cross Entropy

To better explore the learning ability of PointNet++, several data argumentation method are used.

- **Random Input Dropout**: As described in [3], we randomly dropping out input points with a randomized probability for each instance. For each instance in the trainset, a dropout ratio θ is uniformly sampled from $[0, 0.95]$. Then for a specific point, it has a probability p of being dropped. During test, we just keep all available points.
- **Rotation**: To rotate an object along a given axes, say z, the rotation matrix is

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  For each instance in the trainset, it is randomly rotated along z axes with probability 0.5. The degree of rotation, γ, is also randomly sampled from $[0, 360]$. During test, we just keep all available points.

As we can seed from Tab. 3, data argumentation does improve the network performance slightly.

Table 3: Data Argumentation

Accuray	w.o. Data Arg	w.t. Data Arg
Baseline	0.8582	**0.8578**
PointNet++	0.9210	**0.9257**

4 BONUS

Question 3: Suppose you have an image of a car and its corresponding 3D model in the world coordinate system, how do you transform the coordinates in the three-dimensional space to correspond to the pixels of the image, i.e. pixel coordinate system?

My Answer: As described in `01-camera-models.pdf`, given a pinhole O and a corresponding coordinate system, a focal length f, point $P(x, y, z)$ in on a 3D object, the projection of P, say P', on retinal plane is

$$P' = [x', y']^T = \left[f \cdot \frac{x}{z}, f \cdot \frac{y}{z}\right]^T$$

However, at this time we need to find the mapping from a 3D object to its 2D projection, which means that we have no idea about the where the pinhole O is, what the coordinate system should be and what is the length of f.

It is proved that the world coordinate system can be represented by a 4×4 matrix.

$$M = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{matrix} \rightarrow & x-axis \\ \rightarrow & y-axis \\ \rightarrow & z-axis \\ \rightarrow & translation \end{matrix}$$

See the example in Fig. 6. The coordinates from Fig. 6, are, in purple, the position, and in red, green and blue, the coordinates of the x-, y- and z-axis of an arbitrary coordinate system (which are all defined with respect to the world coordinate system). Note that the axes making up this arbitrary coordinates system are unit vectors.

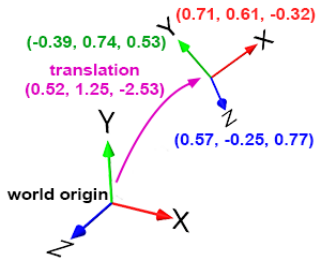

Figure 6: Coordinates systems translation and axes coordinates are defined with respect to the world coordinate system (a right-handed coordinate system is used).

As long as we have a point $P(x, y, z)$ defined in a fixed coordinate system, we can quickly compute $P'(x', y', z')$ in another coordinate system by

$$\begin{bmatrix} P' \\ 1 \end{bmatrix} = M * \begin{bmatrix} P \\ 1 \end{bmatrix}$$

Then we can use a naive gird search by looping over space angle and focal length to find the best fitted coordinate system with pinhole O and f.

5 OPEN QUESTION

Question 4: If you want to design a framework to learn to color a 3D model utilizing the information of single image, what do you think?

My Answer: From Question 3, we have the transformation matrix describing how to project a 3D object into a 2D image, thus gives the clue to color a 3D model based on a single image information. However, there must be a assumption: the coloring of the model has a pattern, or the coloring is deducible. For example, clothing is usually symmetrical and thus we can paint a skit even though we only see the front of the it. A generating model could be employed to catch the joint distribution of color co-occurrence with image information.

If the coloring does not have any pattern, we can only paint the corresponding part of the 3D model from image.

REFERENCES

[1] Saifullahi Aminu Bello, Shangshu Yu, and Cheng Wang. 2020. Review: deep learning on 3D point clouds. *Remote. Sens.* 12 (2020), 1729.

[2] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), 77–85.

[3] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*.