

## Project-3 of “Neural Network and Deep Learning”

### 1. Save 3D Point Cloud

The data used in this experiment is stored in the form of HDF5 files, in order to have a deeper understanding of the 3D point cloud data, in this part, we try to take a 3D model from the dataset and save it.

#### 1.1 Create a dataset loader using the pytorch dataset provided in the sample code

```
1. root = 'D:\\modelnet40_ply_hdf5_2048'
2.
3. train_data_list = 'train_files.txt'
4. test_data_list = 'test_files.txt'
5. train_dataset = ModelNetDataset(root, train_data_list)
6. test_dataset = ModelNetDataset(root, test_data_list)
7.
8. batch_size = 1
9.
10. train_loader = DataLoader(train_dataset, batch_size = batch_size, shuffle = True, num_workers = 0)
11. test_loader = DataLoader(test_dataset, batch_size = batch_size, shuffle = True, num_workers = 0)
```

#### 1.2 Read the data with the dataset loader and take out one of the models (2048 ×3)

```
1. it = iter(train_loader)
2. batch = next(it)
3. print(batch)
4. points= batch['points']
5.
```

```
6. points = points.squeeze(0).permute(1,0)
7. points_ls = points.numpy()
```

```
{'points': tensor([[[ 0.0855, -0.0679, -0.2667, ...,
0.0139, 0.2947, -0.0762],
[-0.3087, 0.5406, -0.2915, ..., -0.6664,
-0.6770, 0.3341],
[-0.1598, 0.1991, -0.3241, ..., 0.1305,
-0.0528, 0.1660]]]), 'label': tensor([[8]],
dtype=torch.uint8)}
```

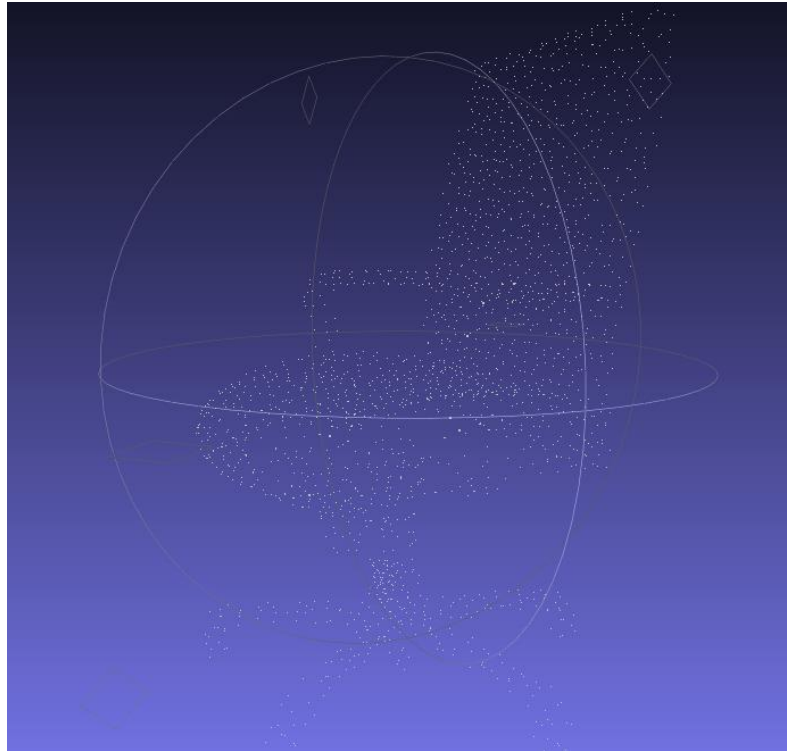
### batch 输出结果

## 1.3 Save the model as the format of obj file using numpy (Note: You need to first understand the storage structure of the obj file from the above materials)

```
1. f = open('pointcloud.obj', "w",encoding='UTF-8')
2. for point in points_ls:
3.     f.write('v' + ' ' + str(point[0]) + ' ' + str(point[1]) + ' ' + str(point[2]))
4.     f.write("\n")
5. f.close()
```

## 1.4 show the 3D picture

As can be seen from the above figure, the entity corresponding to the batch output label is the chair. Open with Meshlab software to obtain Figure,



**chair**

## 2. Training and Testing

To prevent the model from overfitting, I added a dropout layer before the final output layer. At the same time, it is well known that BN is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs, so in order to get a better model effect, I perform BN operations on the output variables after each convolution. Last but not least, I deepened the depth of the convolutional neural network, hoping to get a better model effect.

```
1. class cls_3d(nn.Module):  
2.     def __init__(self):  
3.         super().__init__()  
4.         self.conv1 = nn.Conv1d(3, 64, 1)  
5.         self.batch1 = nn.BatchNorm1d(64)  
6.         self.conv2 = nn.Conv1d(64, 128, 1)
```

```

7.     self.batch2 = nn.BatchNorm1d(128)
8.     self.conv3 = nn.Conv1d(128, 256, 1)
9.     self.batch3 = nn.BatchNorm1d(256)
10.    self.conv4 = nn.Conv1d(256, 512, 1)
11.    self.batch4 = nn.BatchNorm1d(512)
12.    self.fc1 = nn.Linear(512, 256)
13.    self.dropout = nn.Dropout(p=0.2)
14.    self.fc2 = nn.Linear(256, 40)
15.    self.relu = nn.ReLU()
16.
17.    def forward(self, x):
18.        x = self.relu(self.conv1(x))
19.        x = self.batch1(x)
20.        x = self.relu(self.conv2(x))
21.        x = self.batch2(x)
22.        x = self.relu(self.conv3(x))
23.        x = self.batch3(x)
24.        x = self.relu(self.conv4(x))
25.        x = self.batch4(x)
26.        x = torch.max(x, 2, keepdim=True)[0]
27.        x = x.view(x.size(0), -1)
28.
29.        x = self.fc1(x)
30.        x = self.dropout(x)
31.        x = self.fc2(x)
32.
33.    return x

```

The experimental result is as follows,

Model	total train epoch	learning rate	batch size	test Loss	test Acc
cls_3d	50	1e-3	32	0.511	84.8%
	100			0.486	86.8%

	150			0.581	84.8%
--	-----	--	--	-------	-------

Therefore, the optimal epoch = 100.

Model	total train epoch	learning rate	batch size	test Loss	test Acc
cls_3d	100	1e-3	32	0.486	86.8%
		2e-3		0.537	85.9%
		1e-4		0.525	83.6%

Therefore, the optimal learning rate = 1e-3.

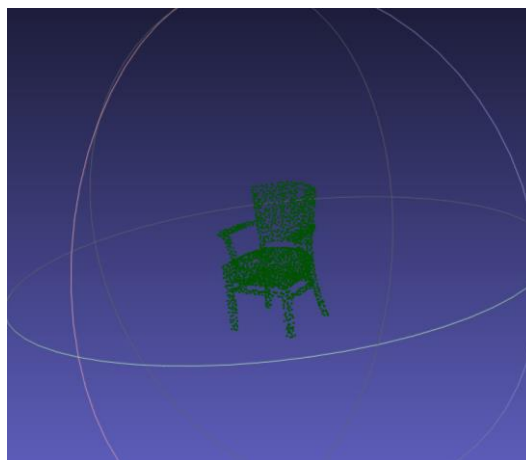
Model	total train epoch	learning rate	batch size	test Loss	test Acc
cls_3d	100	1e-3	32	0.486	86.8%
			1	11.76	16.5%
			64	0.553	84.7%

Therefore, the optimal batch size = 32. And the best accuracy of test is 86.8%.

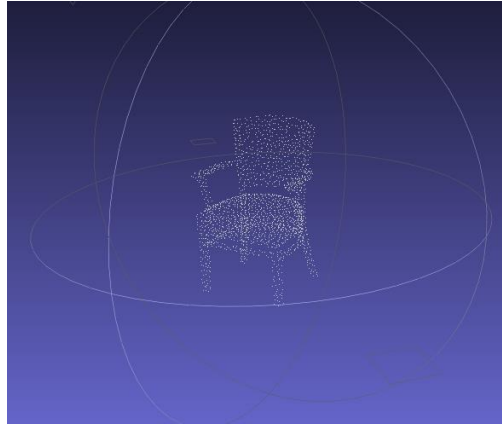
### 3. Question

**3.1 In what form is the obj file storing vertex color data? If in addition to the vertex coordinates (x, y, z), you also want to store the color data of each vertex in the obj file, what should you do? (If you can achieve it, please submit the final obj file or 3D model screenshot)**

A vertex can be specified in a line starting with the letter v. That is followed by (x, y, z [, w]) coordinates. W is optional and defaults to 1.0. Some applications support vertex colors, by putting red, green and blue values after x, y and z. The color values range from 0 to 1. I chose green as the color, and chose the “chair” for easy comparison to view the results. The result is as follows,



pointcloud\_color.obj — chair



pointcloud\_without\_color.obj — chair

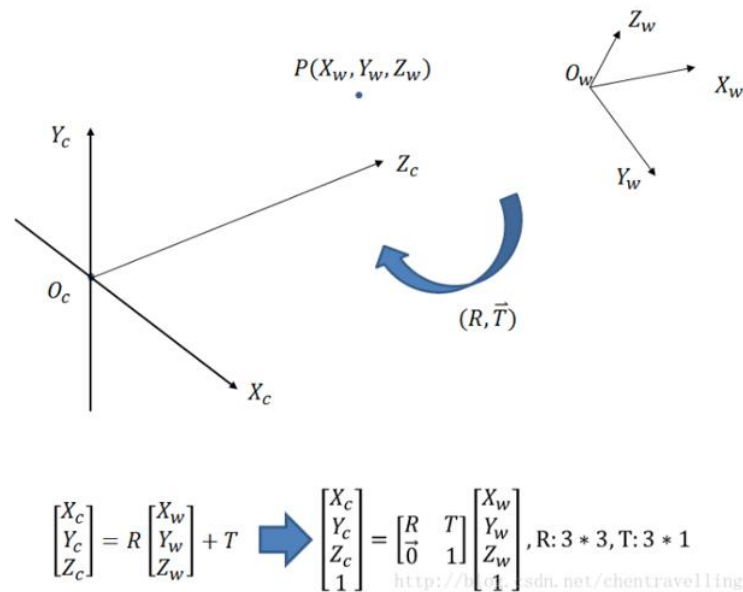
### 3.2 What are the differences between neural networks for point clouds and for ordinary images?

Point clouds input to the neural network is point coordinates or vector representation of points, while ordinary images input to the neural network are the width and height of the image and pixels. For point clouds, one-dimensional convolution is more suitable, while for ordinary images, two-dimensional convolution is more suitable.

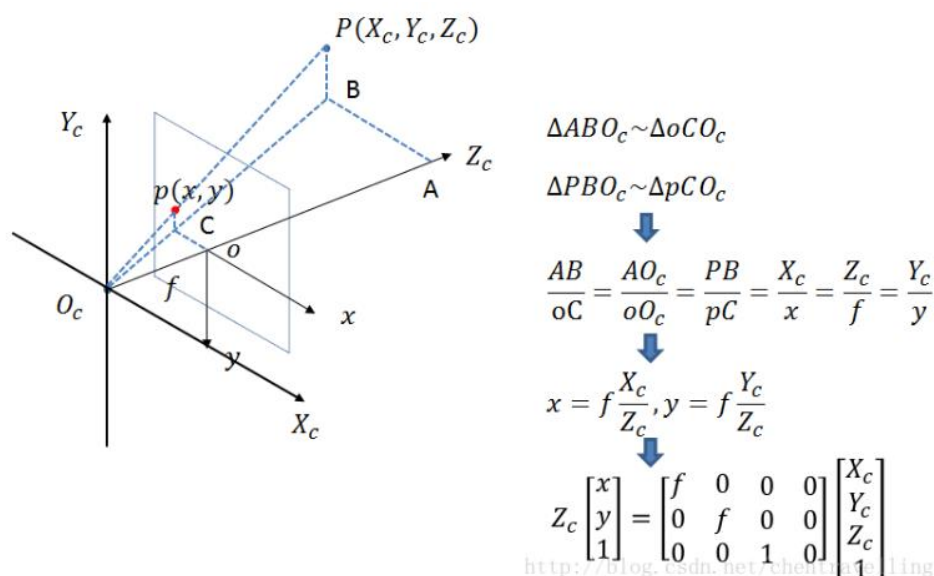
### 3.3 (Bonus) Suppose you have an image of a car and its corresponding 3D model in the world coordinate system, how do you transform the coordinates in the three-dimensional space to correspond to the pixels of the image, i.e. pixel coordinate system (refer to 01-camera-models.pdf)?

First, you need to transform **the world coordinate system** to the **camera coordinate system**. This change belongs to rigid body

transformation, that is, the object does not deform and only needs to rotate and translate. The transformation diagram and formula are shown in the figure below.



Next, you need to convert **the camera coordinate system** to **the image coordinate system**. This transformation belongs to the perspective projection relationship and is a process from 3D to 2D. The transformation diagram and formula are shown in the figure below.

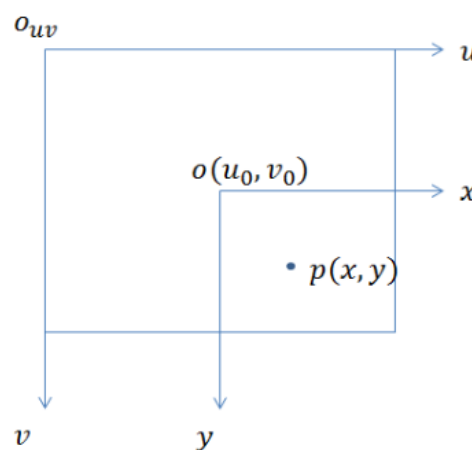




At this time, the unit of the projection point  $p$  is still mm, not a pixel, and needs to be further converted to a pixel coordinate system.

Finally, **the image coordinate system** needs to be converted into **the pixel coordinate system**.

The pixel coordinate system and the image coordinate system are on the imaging plane, but their origins and measurement units are different. The origin of the image coordinate system is the intersection of the optical axis of the camera and the imaging plane, which is usually the midpoint or principal point of the imaging plane. The unit of the image coordinate system is mm, which is a physical unit, and the unit of the pixel coordinate system is pixel. We usually describe that a pixel is a few rows and columns. So the conversion between the two is as follows, where  $dx$  and  $dy$  indicate how many mm each column and each row represent, ie  $1\text{pixel}=dx\text{ mm}$



$$\begin{cases} u = \frac{x}{dx} + u_0 \\ v = \frac{y}{dy} + v_0 \end{cases}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

<http://blog.csdn.net/chentravell>

In summary, the formula for converting the world coordinate system into a pixel coordinate system is as follows

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Camera internal parameters}} \underbrace{\begin{bmatrix} R & T \\ \vec{0} & 1 \end{bmatrix}}_{\text{Camera external parameters}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

http:// net . . . lling

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

According to the obj file of the 3D picture, obtain the color information and input it to the neural network. The real label of the picture is the category of each color, so the original problem can be transformed into a multi-label problem (One picture can have multiple colors). We can get the result by training the neural network.