

Lecture Series for Computer Vision

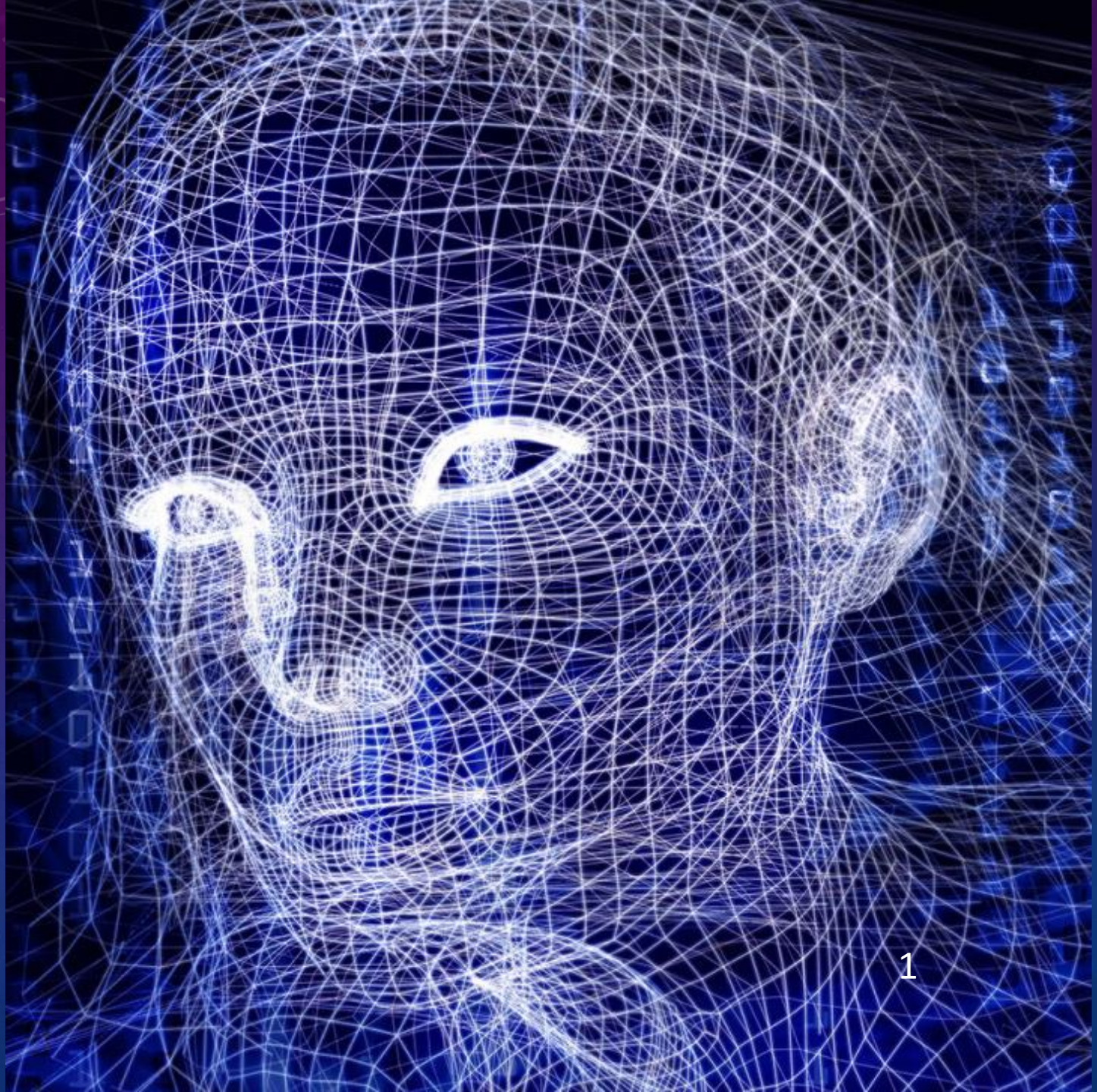
Chapter 6

GAN for 3D Reconstruction

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science
and Technology





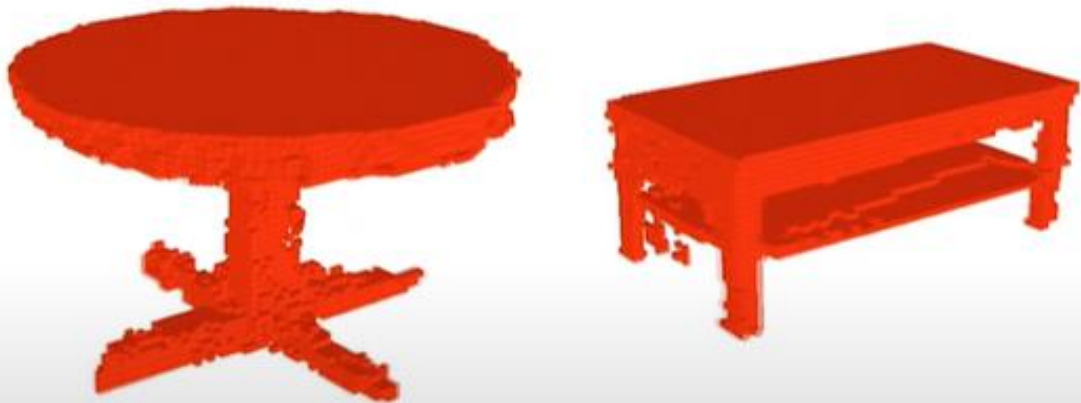
GAN-based 3D Reconstruction

AI Makes 3D Models From Photos (3D-GAN)

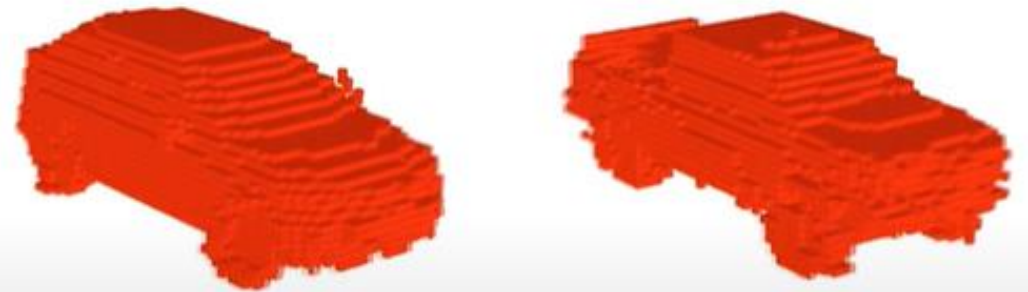
Source: [Wu and Zhang et al. 2017]

Randomly Sampled Shapes

Tables



Cars



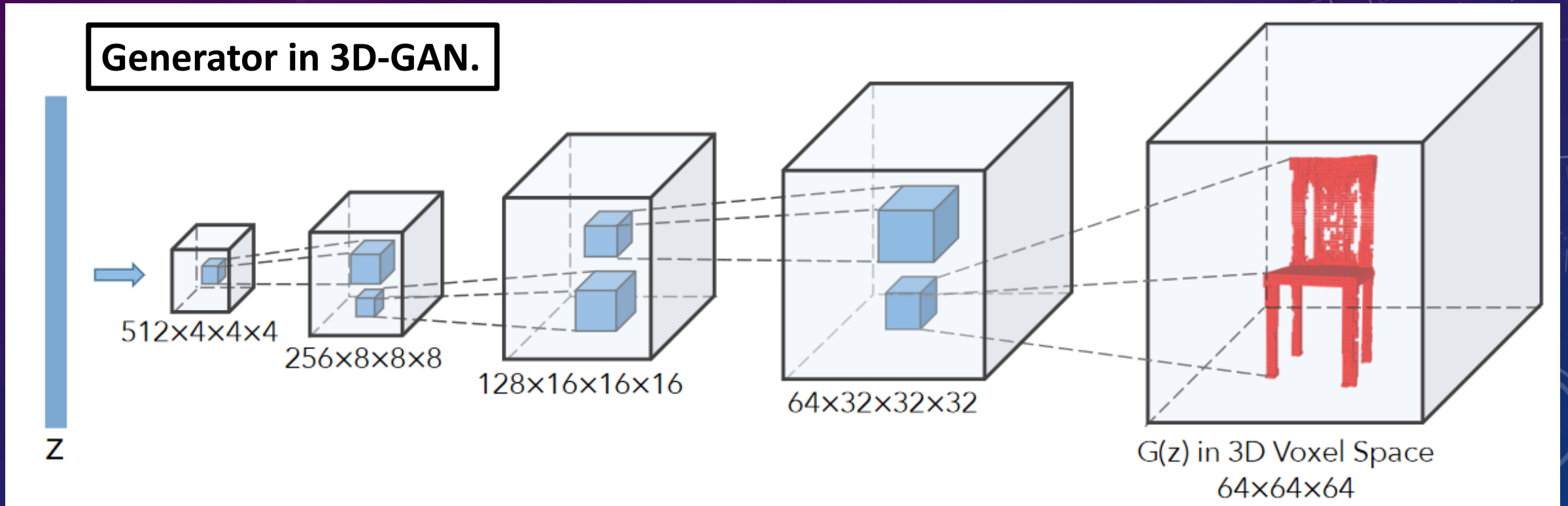
This Neural Network Creates 3D Objects From Your Photos



<https://youtu.be/548sCh0mMRc> [00:00 - 04:48]

Example 6.1 – 3D Chair Generation by 3D-GAN

(Wu and Zhang et al., 2017)



The generator in 3D-GAN. The discriminator mostly mirrors the generator.

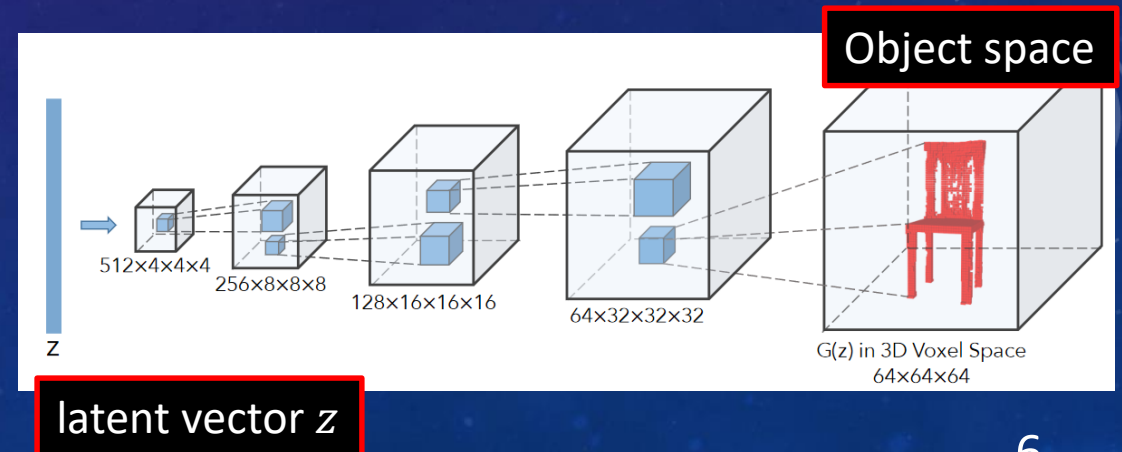
Example 6.1 – 3D Chair Generation by 3D-GAN

(Wu and Zhang et al., 2017)

Our results ($64 \times 64 \times 64$)



The generator in 3D-GAN aims to generate the novel objects by sampling a latent vector z and mapping it to the object space.



Example 6.1 – Structure of the Codes

Please download the 3D-GAN codes from the Moodle “Exercise6-1_3D-GAN.zip”

The code structure can be visualized as follows:

|-- 3D_GAN

|-- main.py

|-- train.py

|-- utils.py

|-- ... etc.

|-- input

|-- chair

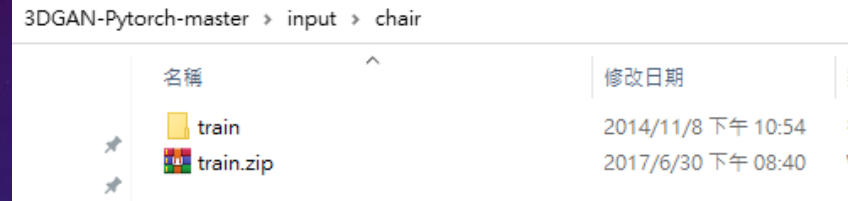
|-- train.zip (Please unzip it!)

3DGAN-Pytorch-master				
	名稱	修改日期	類型	大小
	3D_GAN	2021/5/17 上午 10:48	檔案資料夾	
	input	2017/6/30 下午 08:40	檔案資料夾	
	output	2021/5/17 上午 10:49	檔案資料夾	
	README.md	2017/6/30 下午 08:40	Markdown 來源...	3 KB

3DGAN-Pytorch-master > 3D_GAN				
	名稱	修改日期	類型	大小
	.idea	2021/5/17 上午 10:48	檔案資料夾	
	__pycache__	2021/5/17 上午 10:48	檔案資料夾	
	floyd_requirements.txt	2017/6/30 下午 08:40	文字文件	1 KB
	lr_sh.py	2017/6/30 下午 08:40	JetBrains PyChar...	12 KB
	main.py	2021/5/17 上午 10:47	JetBrains PyChar...	4 KB
d_illum_S	model.py	2017/6/30 下午 08:40	JetBrains PyChar...	5 KB
	train.py	2021/5/17 上午 10:48	JetBrains PyChar...	6 KB
	utils.py	2017/6/30 下午 08:40	JetBrains PyChar...	5 KB

Example 6.1 – How to train 3D-GAN

- Build the Pytorch environment on Anaconda, which is the same as described in “CV_Ch4_Example 4.6 CycleGAN”
- Prepare the data by unzipping the train.zip file in “/input/chair/train.zip”



- Train the model with [train.py](#) in “/3D_GAN/train.py” :

```
C:\WINDOWS\system32> cd 3DGAN-Pytorch-master/3D_GAN
```

```
C:\WINDOWS\system32\ 3DGAN-Pytorch-master/3D_GAN > python train.py
```

- If you see the following statements, the model is successfully being trained!

```
Anaconda Prompt (anaconda3) - python main.py
Epoch-5, Step-11, Discriminator Loss:1.2288357019424438, Generator Loss:1.3058568239212036
Epoch-5, Step-12, Discriminator Loss:1.3398091793060303, Generator Loss:2.0633797645568848
Epoch-5, Step-13, Discriminator Loss:0.965429961681366, Generator Loss:0.8995237350463867
Epoch-5, Step-14, Discriminator Loss:1.682672381401062, Generator Loss:2.442201614379883
Epoch-5, Step-15, Discriminator Loss:1.4248782396316528, Generator Loss:2.051032066345215
Epoch-5, Step-16, Discriminator Loss:1.0689854621887207, Generator Loss:1.3518543243408203
Epoch-5, Step-17, Discriminator Loss:1.3359825611114502, Generator Loss:1.4505648612976074
Epoch-5, Step-18, Discriminator Loss:0.9492653012275696, Generator Loss:0.8005374073982239
Epoch-5, Step-19, Discriminator Loss:1.4918588399887085, Generator Loss:2.0125999450683594
```


Example 6.1 – (model.py) Generator

```
class _G(torch.nn.Module):
    def __init__(self, args):
        super(_G, self).__init__()
        self.args = args
        self.cube_len = args.cube_len

        padd = (0, 0, 0)
        if self.cube_len == 32:
            padd = (1, 1, 1)

        self.layer1 = torch.nn.Sequential(
            torch.nn.ConvTranspose3d(self.args.z_size, self.cube_len*8, kernel_size=4, stride=2, bias=args.bias, padding=padd),
            torch.nn.BatchNorm3d(self.cube_len*8),
            torch.nn.ReLU()
        )
        self.layer2 = torch.nn.Sequential(
            torch.nn.ConvTranspose3d(self.cube_len*8, self.cube_len*4, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len*4),
            torch.nn.ReLU()
        )
        self.layer3 = torch.nn.Sequential(
            torch.nn.ConvTranspose3d(self.cube_len*4, self.cube_len*2, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len*2),
            torch.nn.ReLU()
        )
        self.layer4 = torch.nn.Sequential(
            torch.nn.ConvTranspose3d(self.cube_len*2, self.cube_len, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len),
            torch.nn.ReLU()
        )
```

Employ “ConvTranspose3d” to perform 3D transposed convolution

Example 6.1 – (model.py) Discriminator

```
class _D(torch.nn.Module):
    def __init__(self, args):
        super(_D, self).__init__()
        self.args = args
        self.cube_len = args.cube_len

        padd = (0,0,0)
        if self.cube_len == 32:
            padd = (1,1,1)

        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv3d(1, self.cube_len, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len),
            torch.nn.LeakyReLU(self.args.leak_value)
        )
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv3d(self.cube_len, self.cube_len*2, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len*2),
            torch.nn.LeakyReLU(self.args.leak_value)
        )
        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv3d(self.cube_len*2, self.cube_len*4, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len*4),
            torch.nn.LeakyReLU(self.args.leak_value)
        )
        self.layer4 = torch.nn.Sequential(
            torch.nn.Conv3d(self.cube_len*4, self.cube_len*8, kernel_size=4, stride=2, bias=args.bias, padding=(1, 1, 1)),
            torch.nn.BatchNorm3d(self.cube_len*8),
            torch.nn.LeakyReLU(self.args.leak_value)
        )
```

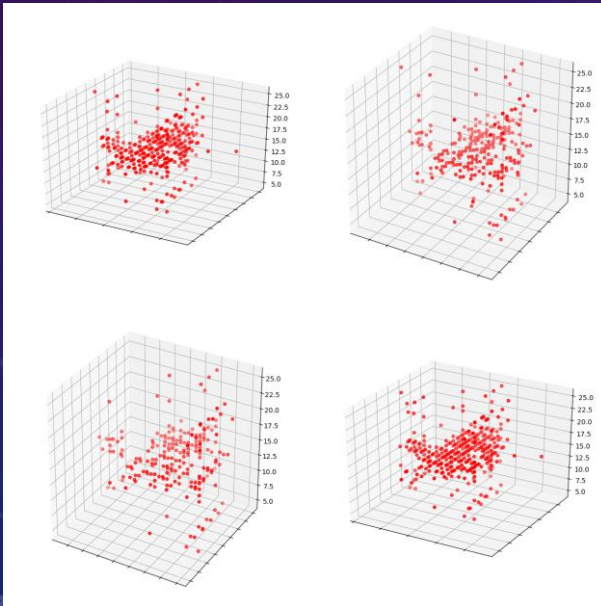
The network structure of the discriminator mostly mirrors the generator

Use “Conv3d” and “BatchNorm3d” since the input data are 3-dimentional data

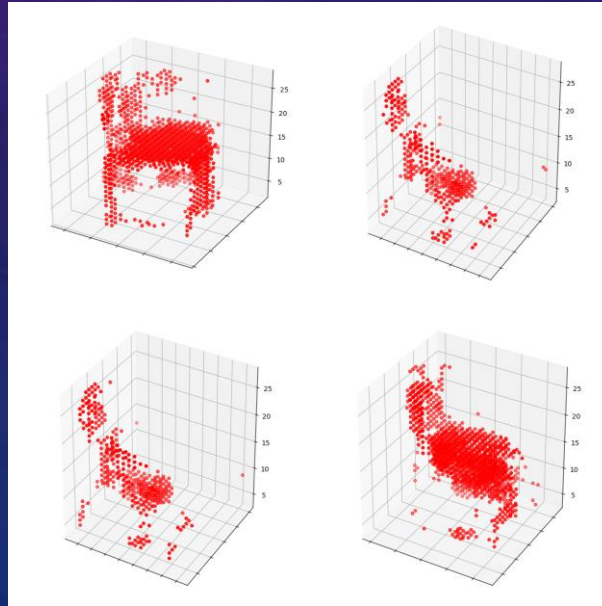
Example 6.1 – Result

- The generated 3D objects and the model will be saved in “/output/image”
- The tensorboard log file will be saved in “/output/log”, and you can use `tensorboard --logdir=./` to visualize the outputs during training.

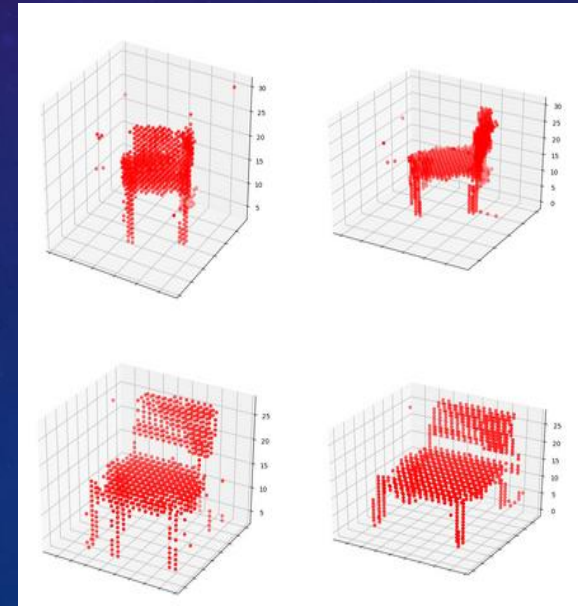
Epoch-0



Epoch-200



Epoch-1000



Exercise 6.1 – 3D-GAN

As the training process takes too long (around 1,000 epochs), please download the pre-trained model on the Moodle, and use it as your pre-trained weights to answer the following question.

1. Use the pre-trained model to generate 10 different 3D-object data by the random noise.
2. Re-trained the model with WGAN-GP, the WGAN-GP's code can be found in "CV_Ch4_Example 4.5 DC-GAN". You only need to train 500 steps, and show what can be observed in the generated data.