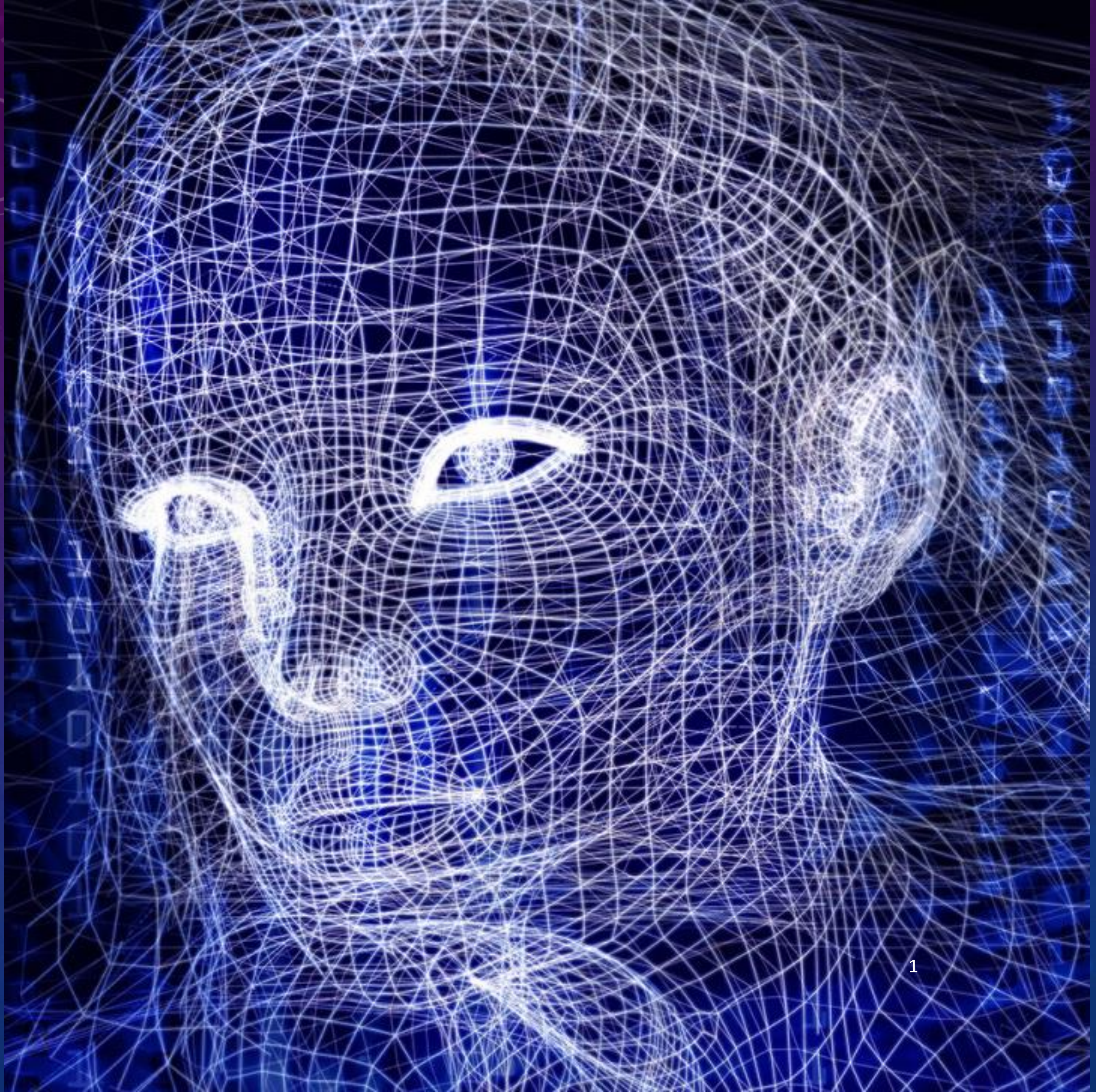
COMPUTER VISION AND ITS APPLICATIONS

CH 4_EXERCISE

徐繼聖

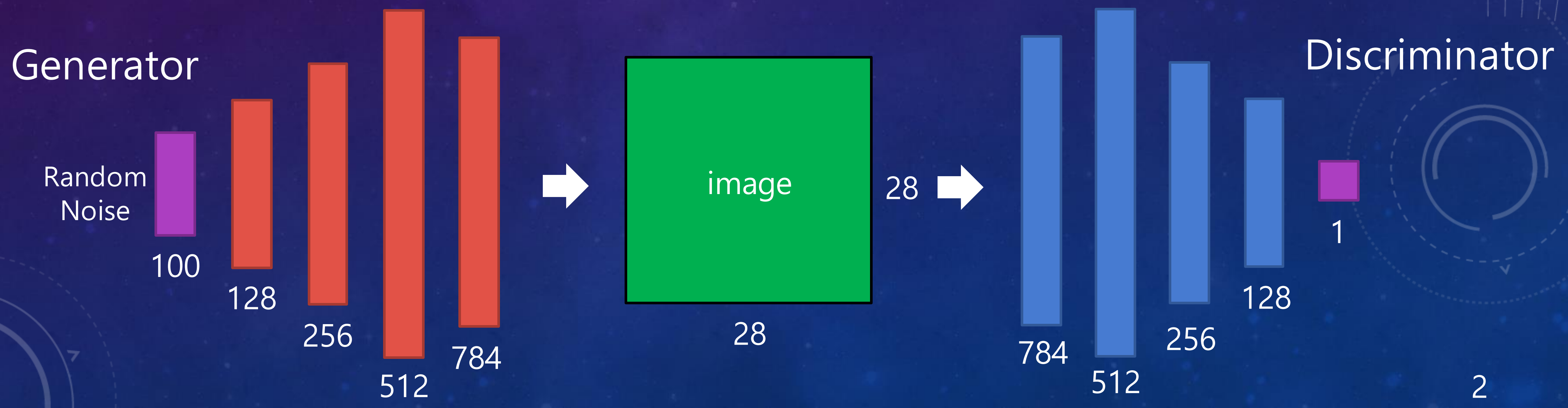
Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



Example 4-1: Generative Adversarial Network

- Please download the "exercise4.1_ GAN.ipynb" on Moodle.
- Upload the "exercise4.1_ GAN.ipynb" to the Google Colab.
- Follow the sample code to understand the data flow of the generative adversarial network.
  - ➢ The generator aims to generate a novel image from a noise input.
  - ➢ The discriminator aims to distinguish the generated image from the real one.



Example 4-1: Generative Adversarial Network

Define the hyper-parameter and load the training data

```
train_epoch = 50
batch_size = 64
noise_size = 100
lr = 2e-4

img_transform = transforms.Compose([
transforms.ToTensor(), transforms.Normalize([0.5], [0.5])])

dataset = torchvision.datasets. FashionMNIST(root='./data', train=True, download=True,
transform=img_transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=False)
```

Define the hyperparameter

Download the FashionMNIST dataset to the folder './data'

Example 4-1: Generative Adversarial Network

Define the generator and discriminator

```
class generator(nn.Module):
    def __init__(self, input_size=100, n_class = 28*28):
        super(generator, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.fc2 = nn.Linear(self.fc1.out_features, 512)
        self.fc3 = nn.Linear(self.fc2.out_features, 1024)
        self.fc4 = nn.Linear(self.fc3.out_features, n_class)
        self.tanh = nn.Tanh()

    def forward(self, input):
        x = F.leaky_relu(self.fc1(input), 0.2)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = self.tanh(self.fc4(x))
        return x
```

Network Structure of the Generator

```
class discriminator(nn.Module):
    def __init__(self, input_size=28*28, n_class=1):
        super(discriminator, self).__init__()
        self.fc1 = nn.Linear(input_size, 1024)
        self.fc2 = nn.Linear(self.fc1.out_features, 512)
        self.fc3 = nn.Linear(self.fc2.out_features, 256)
        self.fc4 = nn.Linear(self.fc3.out_features, n_class)
        self.sigmoid = nn.Sigmoid()
    def forward(self, input):
        x = F.leaky_relu(self.fc1(input), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = F.dropout(x, 0.3)
        x = self.sigmoid(self.fc4(x))
        return x
```

Network structure of the discriminator

Example 4-1: Generative Adversarial Network

Define the loss function

```
G = generator(input_size=noise_size, n_class=28*28)
D = discriminator(input_size=28*28, n_class=1)
```

Build a model

```
if torch.cuda.is_available():
    G.cuda()
    D.cuda()

print(G)
print(D)


BCE_loss = nn.BCELoss()
```

Use “binary cross-entropy” as loss function

```
D_optimizer = torch.optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
```

Example 4-1: Generative Adversarial Network

Train the discriminator to determine whether a sample is from the real/fake distribution.

```
# train discriminator D
    D.zero_grad()
    x_ = x_.view(-1, 28 * 28)
    mini_batch = x_.size()[0]
    y_real_ = torch.ones(mini_batch)
    y_fake_ = torch.zeros(mini_batch)
    x_, y_real_, y_fake_ = Variable(x_),
Variable(y_real_), Variable(y_fake_)
    if torch.cuda.is_available():
        x_ = x_.cuda()
        y_real_ = y_real_.cuda()
        y_fake_ = y_fake_.cuda()
    D_result = D(x_)
    D_real_loss = BCE_loss(D_result, y_real_)
    D_real_score = D_result


    z_ = torch.randn((mini_batch, noise_size))
    z_ = Variable(z_)
    if torch.cuda.is_available():
        z_ = z_.cuda()
    G_result = G(z_)

    D_result = D(G_result)
    D_fake_loss = BCE_loss(D_result, y_fake_)
    D_fake_score = D_result

    D_train_loss = D_real_loss + D_fake_loss

    D_train_loss.backward()
    D_optimizer.step()
```

Flatten the input images

Generate the noises

Generate the image from the random noise

Note that, during training discriminator,
Real input with Label 1
Generated input with Label 0

Example 4-1: Generative Adversarial Network

Start to training the "generator" G

```
# train generator G
        G.zero_grad()

        z_ = torch.randn((mini_batch, noise_size))   ← Generate the noises
        y_ = torch.ones(mini_batch)

        z_, y_ = Variable(z_), Variable(y_)
        if torch.cuda.is_available():
            z_ = z_.cuda()
            y_ = y_.cuda()
        G_result = G(z_)   ← Generate the images by the noises
        D_result = D(G_result)
        G_train_loss = BCE_loss(D_result, y_)   ← Note that, during training generator,
                                                   **Generated input with Label 1
        G_train_loss.backward()
        G_optimizer.step()
```

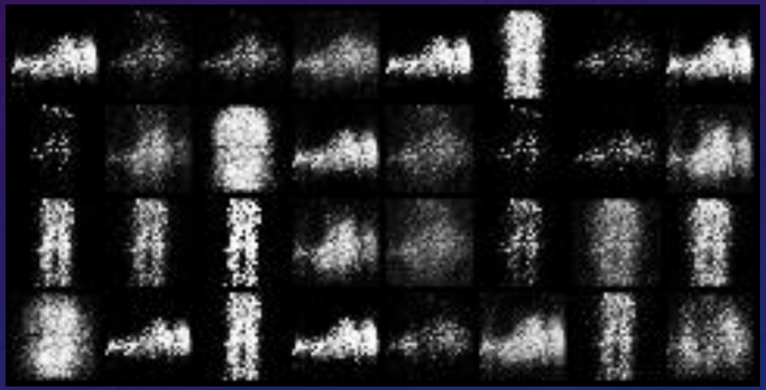
Example 4-1: Generative Adversarial Network

```
if epoch % 1 == 0:
        pic = to_img(G_result.cpu().data)
        save_image(pic, './gan_img/output_{}.png'.format(epoch))
```

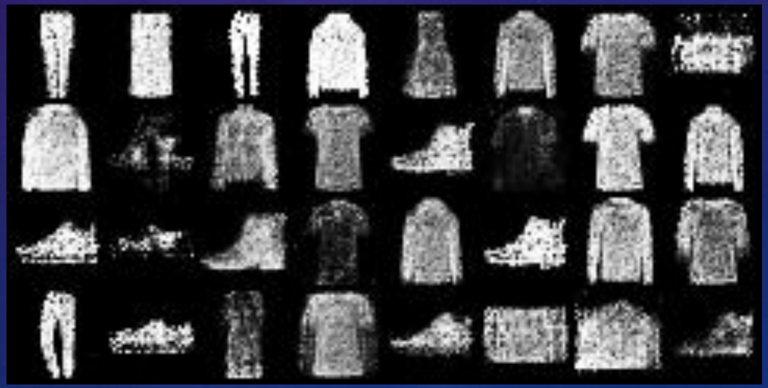
Save the output images

- Result:

Epoch 1 :



Epoch 30 :



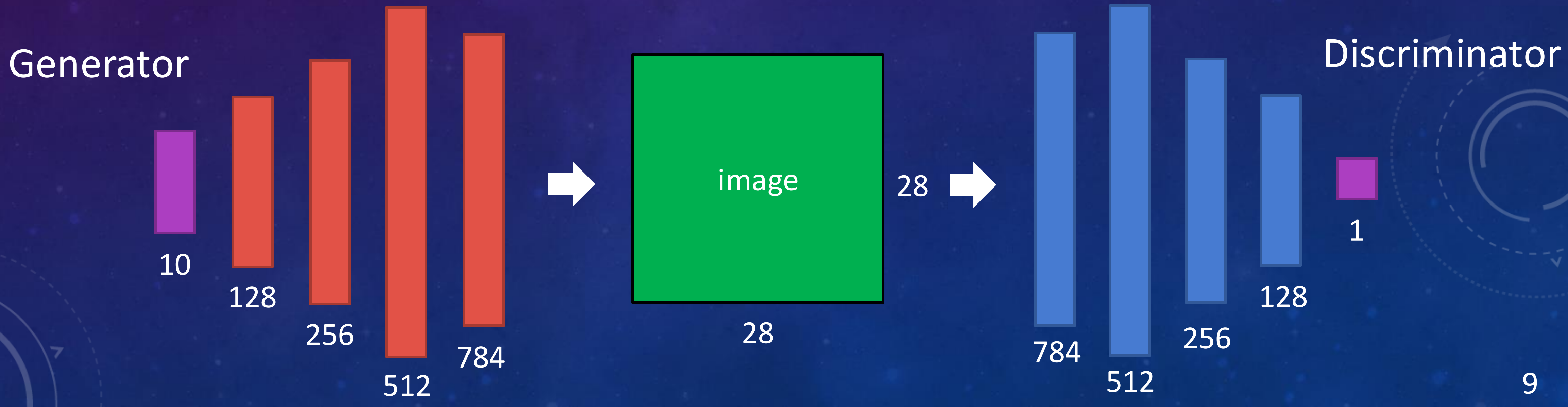
Epoch 50 :

~30 mins

Exercise 4-1: Generative Adversarial Network

- Please download the "exercise4.1_ GAN.ipynb" on Moodle.
    1. Train the GAN and compare the images reconstructed from different numbers of epochs.
    2. Change the learning rate from 0.0002 to 0.002 and compare the images.
    3. Change the generator and the discriminator structures to the below architecture and compare the differences of the results.

Please copy your results and code and paste to a MS Word, then upload to Moodle.



Example 4-2: Generate faces from StyleGAN

- Please download the "exercise4.2_ StyleGAN.ipynb" on Moodle.
- Upload the "exercise4.2_ StyleGAN.ipynb" to the Google Colab.
- Follow the sample code to understand the data flow of the StyleGAN.
    - Generator will generate the faces from random noises.
    - Use two random noises to generate the mixing-style image.

Example 4-2: Generate faces from StyleGAN

```
!wget https://nnabla.org/pretrained-models/nnabla-examples/GANs/stylegan2/styleGAN2_G_params.h5
from generate import *
from IPython.display import Image, display
ctx = get_extension_context("cudnn")
nn.set_default_context(ctx)

num_layers = 18
output_dir = 'results'

nn.load_parameters("styleGAN2_G_params.h5")
```

Get the stylegan pretrained weight

Example 4-2: Generate faces from StyleGAN

```
#@markdown Choose the seed for noise input **z**. (This drastically changes the result)
latent_seed = 217    #@param {type: "slider", min: 0, max: 1000, step:1}

#@markdown Choose the value for truncation trick.
truncation_psi = 0.5    #@param {type: "slider", min: 0.0, max: 1.0, step: 0.01}

#@markdown Choose the seed for stochasticity input.  (This slightly changes the result)
noise_seed = 500    #@param {type: "slider", min: 0, max: 1000, step:1}

#@markdown Number of images to generate
batch_size = 1    #@param {type: "slider", min: 0, max: 20, step:1}
```

Choose the seed for noises, make the noise fixed each time.

Example 4-2: Generate faces from StyleGAN

```
rnd = np.random.RandomState(latent_seed)
z = rnd.randn(batch_size, 512)

nn.set_auto_forward(True)

style_noise = nn.NdArray.from_numpy_array(z)
style_noises = [style_noise for _ in range(2)]

rgb_output = generate(batch_size, style_noises, noise_seed, mix_after=7, truncation_psi=truncation_psi)

images = convert_images_to_uint8(rgb_output, drange=[-1, 1])

# Display all the images
for i in range(batch_size):
    filename = f'seed{latent_seed}_{i}.png'
    imsave(filename, images[i], channel_first=True)
    display(Image(filename, width=512, height=512))
```

Generate the noises as the style.

Generate the facial image by given noise

Output

Example 4-2: Generate faces from StyleGAN

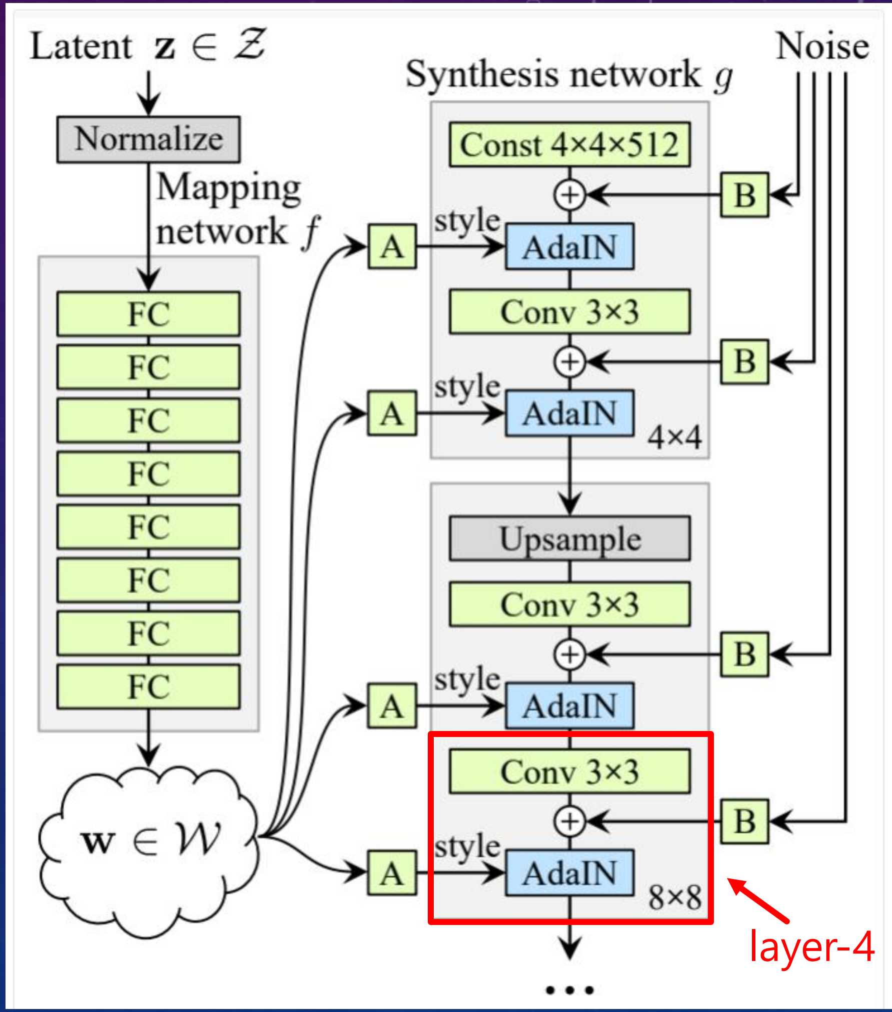
- Make mixing-style image

```
#@title  StyleGAN2  style
#@markdown  Choose  seed
latent_seed  =  819      #@p

#@markdown  Choose  seed
latent_seed2  =  117    #@

#@markdown  Choose  from
mix_after  =  4      #@para

#@markdown  Choose  seed
noise_seed  =  500     #@p
```

The noise seed for Style-A

The noise seed for Style-B

The layer of mixing,
Style-A before layer-4,
Style-B after layer-4



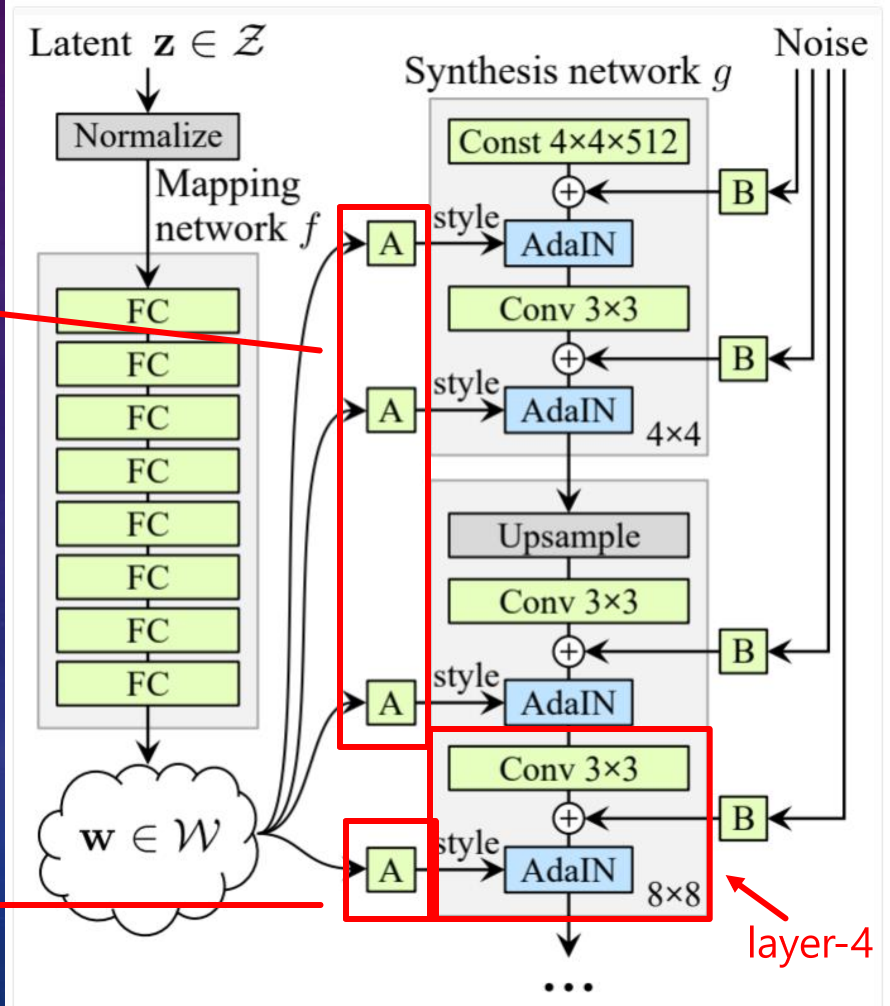
Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

```
#@title StyleGAN2 style
#@markdown Choose seed
latent_seed = 819    #@p

#@markdown Choose seed
latent_seed2 = 117    #@

#@markdown Choose from
mix_after = 4     #@paran

#@markdown Choose seed
noise_seed = 500    #@pa
```

Style-A

Style-B



Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

```
#@markdown Choose seed
noise_seed = 500     #@p

#@markdown Choose the
truncation_psi = 0.7

#@markdown Number of i
batch_size_A = 2     #@p

#@markdown Number of i
batch_size_B = 4     #@p
```

The number of style-A images, it will generate the two noises from style-A random seed, 819.

The number of style-B images, it will generate the four noises from style-B random seed, 117.

Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

```
rnd1 = np.random.RandomState(latent_seed)
z1 = nn.NdArray.from_numpy_array(rnd1.randn(batch_size_A, 512))

rnd2 = np.random.RandomState(latent_seed2)
z2 = nn.NdArray.from_numpy_array(rnd2.randn(batch_size_B, 512))


nn.set_auto_forward(True)
```

Use the seed to get the fixed noises

Generate the 512-dimension noises.

Use the seed to get the fixed noises

Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

Use for loop to mix the two styles and generate the mixing images

[Style-A noise, Style-B noise]

```
mix_image_stacks = []
for i in range(batch_size_A):
    image_column = []
    for j in range(batch_size_B):
        style_noises = [F.reshape(z1[i], (1, 512)), F.reshape(z2[j], (1, 512))]
        rgb_output = generate(1, style_noises, noise_seed, mix_after, truncation_psi)
        image_column.append(convert_images_to_uint8(rgb_output, drange=[-1, 1])[0])
    image_column = np.concatenate([image for image in image_column], axis=2)
    mix_image_stacks.append(image_column)
mix_image_stacks = np.concatenate([image for image in mix_image_stacks], axis=1)
```

Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

Generate the style-A image

```
style_noises= [z1, z1]
rgb_output = generate(batch_size_A, style_noises, noise_seed, mix_after, truncation_psi)
image_A = convert_images_to_uint8(rgb_output, drange=[-1, 1])
image_A = np.concatenate([image for image in image_A], axis=1)
```

style-A noise

```
style_noises = [z2, z2]
rgb_output = generate(batch_size_B, style_noises, noise_seed, mix_after, truncation_psi)
image_B = convert_images_to_uint8(rgb_output, drange=[-1, 1])
image_B = np.concatenate([image for image in image_B], axis=2)
```

style-B noise

Generate the style-B image

Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

```
top_image = 255 * np.ones(rgb_output[0].shape).astype(np.uint8)

top_image = np.concatenate((top_image, image_B), axis=2)
grid_image = np.concatenate((image_A, mix_image_stacks), axis=2)
grid_image = np.concatenate((top_image, grid_image), axis=1)

imsave("grid.png", grid_image, channel_first=True)
display(Image("grid.png", width=256*(batch_size_B+1), height=256*(batch_size_A+1)))
```

Make the grid images for visualization

Example 4-2: Generate faces from StyleGAN

- Make mixing-style image

Style-B, seed 117
mixing layer: 4

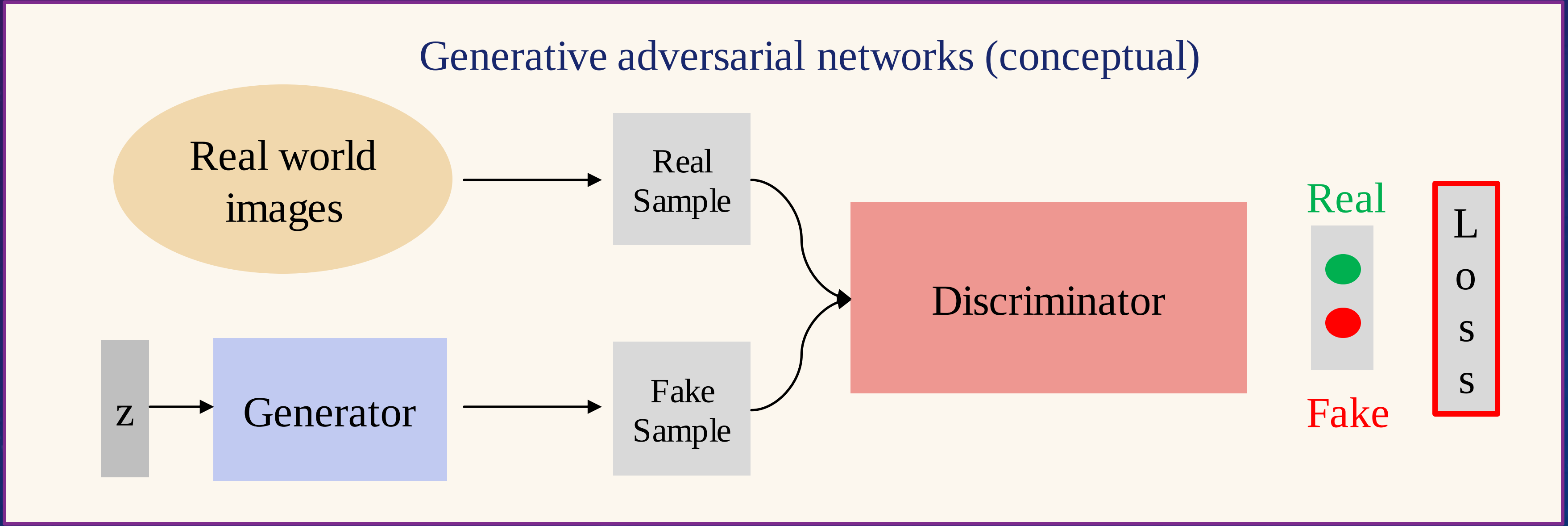
Style-A, seed 819
mixing layer: 4

Exercise 4-2: Generate faces from StyleGAN

- Please download the "exercise4.2_ StyleGAN.ipynb" on Moodle.
- Upload the "exercise4.2_ StyleGAN.ipynb" to the Google Colab.
- Follow the sample code to understand the data flow of the StyleGAN.
    - Adjust the seed number to generate the different image and record the seed number
    - Use mixing method to generate the images
    - Adjust the mixing layer to generate the synthesized image and observe the change along the layer number.

Example 4-3: Adversarial Loss Function

- Please download the "exercise4.3_ GAN.ipynb" on Moodle.
- Upload the "exercise4.3_ GAN.ipynb" to the Google Colab.
- Follow the sample code to understand the data flow of the generative adversarial network.
    - ➢ The generator aims to generate a novel image from a noise input.
    - ➢ The discriminator aims to distinguish the generated image from the real one.
    - ➢ Observe the trajectories of the generator loss and discriminator loss



Example 4-3: Adversarial Loss Function

```
train_epoch = 50
batch_size = 64
noise_size = 100
lr = 2e-4
loss_type = 'non-saturated' #'non-saturated','saturated','wgan','wgan-gp

img_transform = transforms.Compose([
transforms.ToTensor(), transforms.Normalize([0.5], [0.5])])

dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=img_transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=False)
```

Loss type

Example 4-3: Adversarial Loss Function

- We use the structure of previous exercise as the model for training. Refer to Example 4-1

```
class generator(nn.Module):
    # initializers
    def __init__(self, input_size=100, n_class = 28*28):
        super(generator, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.fc2 = nn.Linear(self.fc1.out_features, 512)
        self.fc3 = nn.Linear(self.fc2.out_features, 1024)
        self.fc4 = nn.Linear(self.fc3.out_features, n_class)
        self.tanh = nn.Tanh()

    # forward method
    def forward(self, input):
        x = F.leaky_relu(self.fc1(input), 0.2)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = self.tanh(self.fc4(x))

        return x
```

```
class discriminator(nn.Module):
    # initializers
    def __init__(self, input_size=28*28, n_class=1):
        super(discriminator, self).__init__()
        self.fc1 = nn.Linear(input_size, 1024)
        self.fc2 = nn.Linear(self.fc1.out_features, 512)
        self.fc3 = nn.Linear(self.fc2.out_features, 256)
        self.fc4 = nn.Linear(self.fc3.out_features, n_class)
        self.sigmoid = nn.Sigmoid()

    # forward method
    def forward(self, input):
        x = F.leaky_relu(self.fc1(input), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc2(x), 0.2)
        x = F.dropout(x, 0.3)
        x = F.leaky_relu(self.fc3(x), 0.2)
        x = F.dropout(x, 0.3)
        x = self.sigmoid(self.fc4(x))

        return x
```

Example 4-3: Adversarial **Loss Function**

Loss part:

```
##define the loss function

####saturated loss
def saturated_loss(DG_score):
    G_loss = torch.mean(torch.log((1-DG_score) + 1e-8))
    return G_loss

###non-saturated
BCE_loss = nn.BCELoss()
def non_saturated_loss(D_result, y_):
    G_loss = BCE_loss(D_result,y_)
    return G_loss

####wgan
def wgan(D_real,D_fake):
    loss_D = -torch.mean(D_real) + torch.mean(D_fake)
    loss_G = -torch.mean(D_fake)
    return loss_D,loss_G
```

Saturated loss

Non-saturated loss

WGAN loss

Example 4-3: Adversarial **Loss Function**

Loss part:

```
def  wgan_gp(D,  interpolates,  D_real,  D_fake,  flag):
    if  flag  ==  'D':
        loss  =  -torch.mean(D_real)  +  torch.mean(D_fake)
        #  wgan  =  loss.detach()
        wgan  =  loss

        x  =  D(interpolates)
        #  print(x.shape)
        gradients  =  autograd.grad(outputs=x.sum(),  inputs=interpolates,
                                    create_graph=True)[0]

        gradients  =  gradients.view(gradients.size(0),  -1)
        gradient_penalty  =  ((gradients.norm(2,  dim=1)  -  1)  **  2).mean()

        loss  +=  10*gradient_penalty

        return  loss, wgan, (gradients.norm(2,  dim=1)).mean()
    if  flag  ==  'G':

        loss  =  -D_fake.mean()

        return  loss
```

Wasserstein distance

Find the discriminator's gradients

Gradient penalty

Example 4-3: Adversarial Loss Function

training part, using non-saturated loss:

```
D_result_real = D(x_)
D_real_score = D_result_real

z_ = torch.randn((mini_batch, noise_size))
z_ = Variable(z_)
if torch.cuda.is_available():
    z_ = z_.cuda()

G_result = G(z_)
D_result_fake = D(G_result)
D_fake_score = D_result_fake
```

Generate the image from the random noise.

```
D_real_loss = BCE_loss(D_result_real, y_real_)
D_fake_loss = BCE_loss(D_result_fake, y_fake_)
D_train_loss = D_real_loss + D_fake_loss
```

Compute the non-saturated GAN loss for updating the discriminator

```
D_train_loss.backward()
D_optimizer.step()
D_losses.append(D_train_loss.data)
```

Example 4-3: Adversarial Loss Function

Remember to clear the gradient first after updating the discriminator

```
G.zero_grad()
```

```
z_ = torch.randn((mini_batch, noise_size))
y_ = torch.ones((mini_batch,1))

z_, y_ = Variable(z_), Variable(y_)
if torch.cuda.is_available():
    z_ = z_.cuda()
    y_ = y_.cuda()

G_result = G(z_)
D_result_fake = D(G_result)
```

```
D_fake_loss = BCE_loss(D_result_fake, y_real_)
G_train_loss = D_fake_loss
```

Compute the non-saturated GAN loss for updating the generator

```
G_train_loss.backward()
G_optimizer.step()

G_losses.append(G_train_loss.data)

writer.add_scalar("D_Loss/train", D_train_loss, step)
writer.add_scalar("G_Loss/train", G_train_loss, step)
```

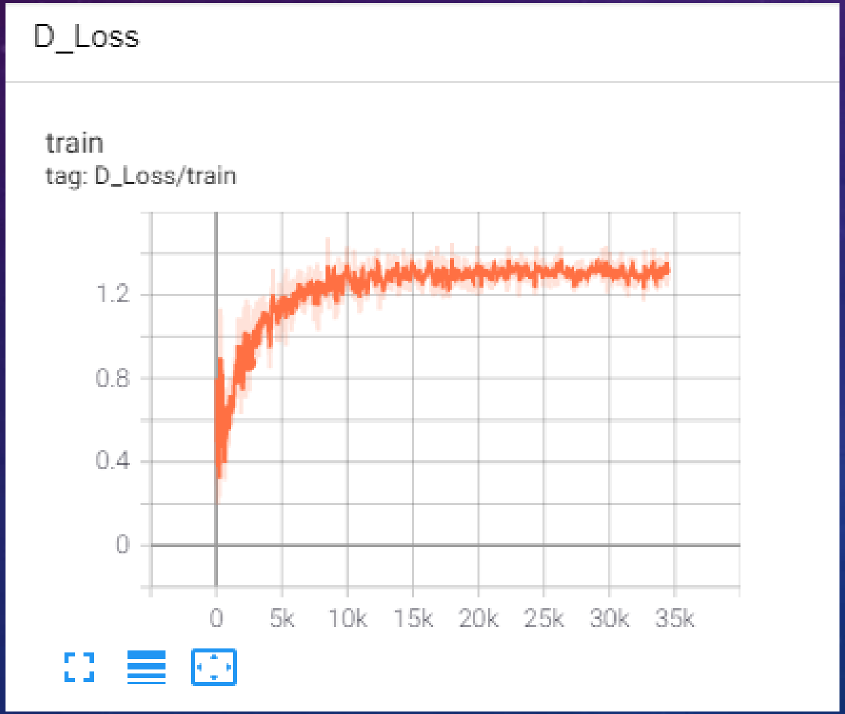
Use tensorboard to record the loss

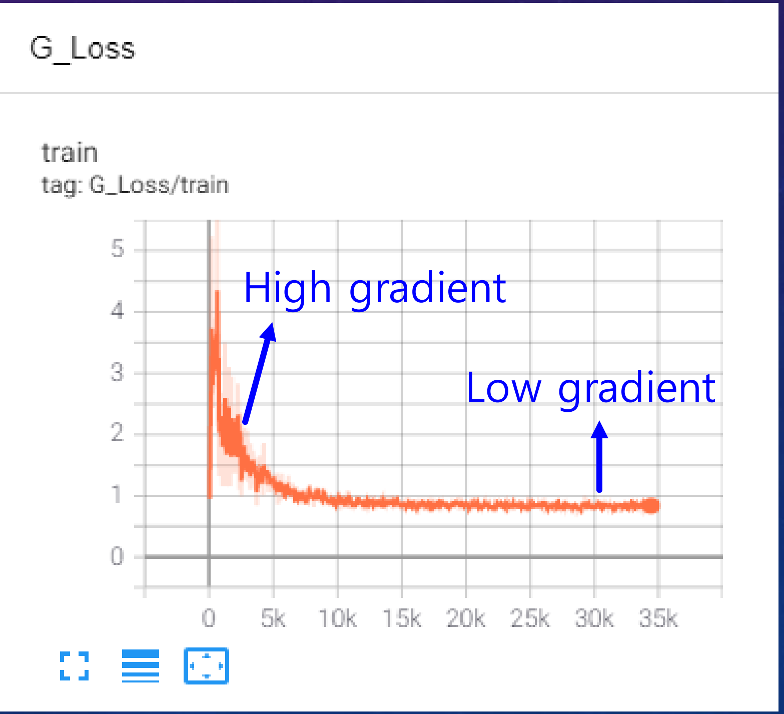
Example 4-3: Adversarial Loss Function

```
writer.close()
!pip install tensorboard
%load_ext tensorboard
%tensorboard --logdir=runs
```

Show the tensorboard on the Colab.

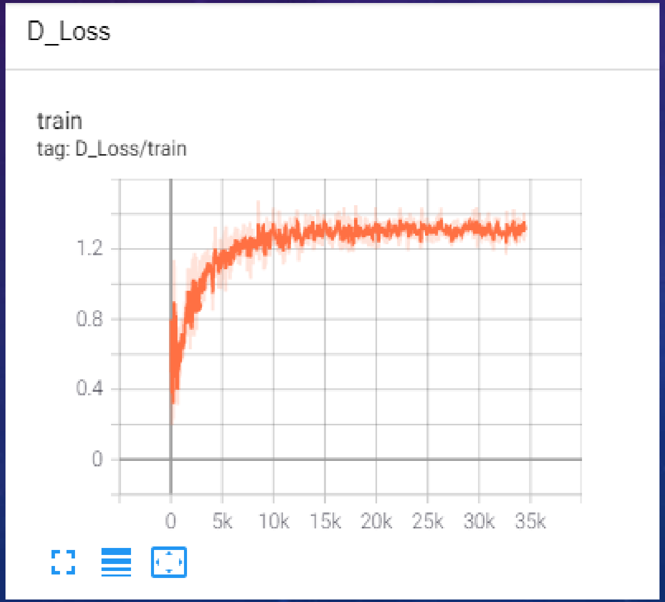
Non-saturated loss trajectory

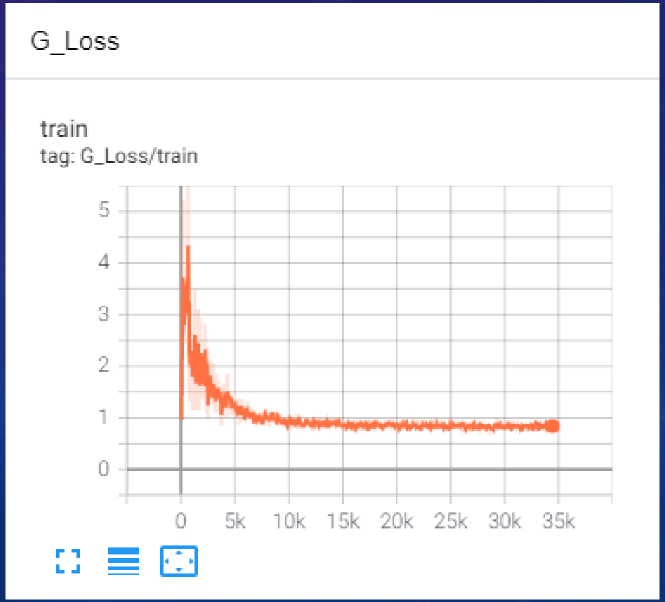




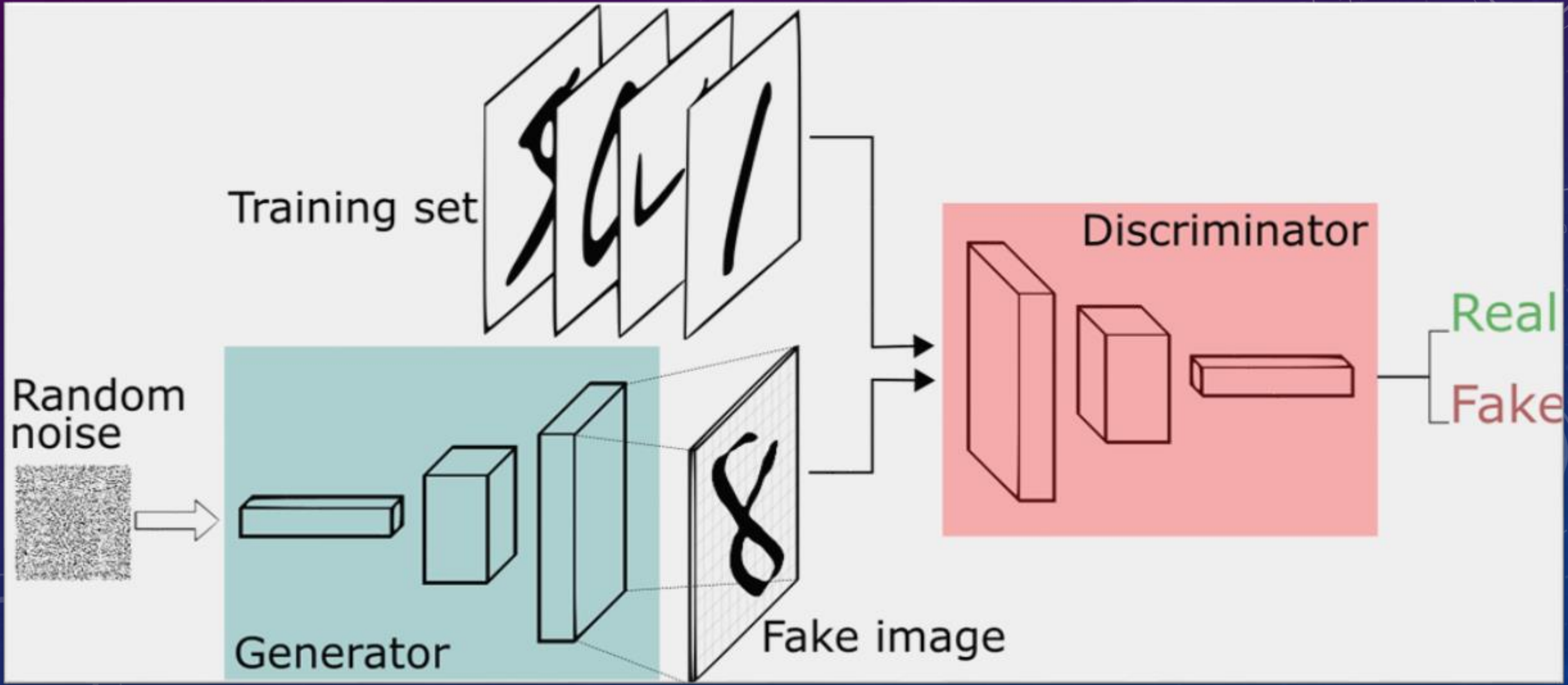
Exercise 4-3: Adversarial **Loss Function**

- Please download the “exercise4.3_ GAN.ipynb” on Moodle.
- Upload the “exercise4.3_ GAN.ipynb” to the Google Colab.
- Follow the sample code to understand the data flow of the generative adversarial network.
  - ➢ Compare the training results by using “wgan”, “wgan-gp”, “saturated loss”, and “non-saturated loss”.
  - ➢ Observe the trajectories of the generator loss and discriminator loss





Let's Try DCGAN By Ourselves: Example 6 Simple GAN Training On Fashion-MNIST



Hyper Parameters and training Data

```
img_size = 32
train_epoch = 5
batch_size = 128
noise_size = 100
lr = 2e-4
experiment_name= 'KL_Loss_Batch_Norm'

experiment_path= './gan_img/{}'.format(experiment_name)

if not os.path.exists(experiment_path):
    os.mkdir(experiment_path)

img_transform = transforms.Compose([
transforms.Resize(img_size),  transforms.ToTensor(), transforms.Normalize([0.5],  [0.5])])

dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True,
transform=img_transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=False)
```

Define the hyperparameters

Give your experiment a describing name

Download the Mnist dataset to the folder './data'

Function for initialization of weights as a Gaussian distribution with mean and standard deviation

```
def normal_init(m, mean, std):
    if isinstance(m, nn.ConvTranspose2d) or isinstance(m,
nn.Conv2d):
        m.weight.data.normal_(mean, std)
        m.bias.data.zero_()

fixed_z_ = torch.randn((8 * 8, noise_size)).view(-1,
noise_size, 1, 1)    # fixed noise
fixed_z_ = Variable(fixed_z_.cuda())
```

Signature of the function, that will change the shape of the distribution

You can later change the mean and std to play around with the initialization of your weights

Color code for network architecture

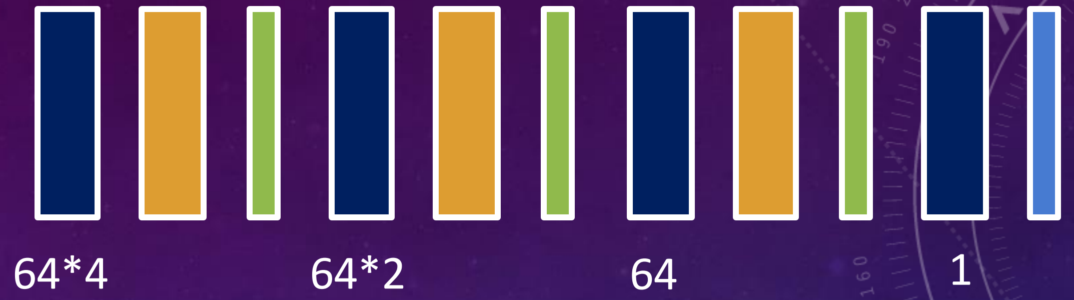
Layer

- Convolutional Layer
- Deconvolutional Layer
- BatchNorm Layer

Activation Function

- ReLU
- Leaky ReLU
- tanh
- sigmoid

DCGAN - Generator




```
class generator(nn.Module):
    # initializers
    def __init__(self, input_size = 100, d=64):
        super(generator, self).__init__()
        self.deconv1 = nn.ConvTranspose2d(input_size, d*4,
4, 1, 0)
        self.deconv1_bn = nn.BatchNorm2d(d*4)
        self.deconv2 = nn.ConvTranspose2d(d*4, d*2, 4, 2, 1)
        self.deconv2_bn = nn.BatchNorm2d(d*2)
        self.deconv3 = nn.ConvTranspose2d(d*2, d, 4, 2, 1)
        self.deconv3_bn = nn.BatchNorm2d(d)
        self.deconv4 = nn.ConvTranspose2d(d, 1, 4, 2, 1)
```

Define the architecture with nn.ConvTranspose2d

```
torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1,
padding=0, output_padding=0, groups=1, bias=True, dilation=1,
padding_mode='zeros')
```

```
# weight_init
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

# forward method
    def forward(self, input):
        x = F.relu(self.deconv1_bn(self.deconv1(input)))
        x = F.relu(self.deconv2_bn(self.deconv2(x)))
        x = F.relu(self.deconv3_bn(self.deconv3(x)))
        x = F.tanh(self.deconv4(x))

        return x
```

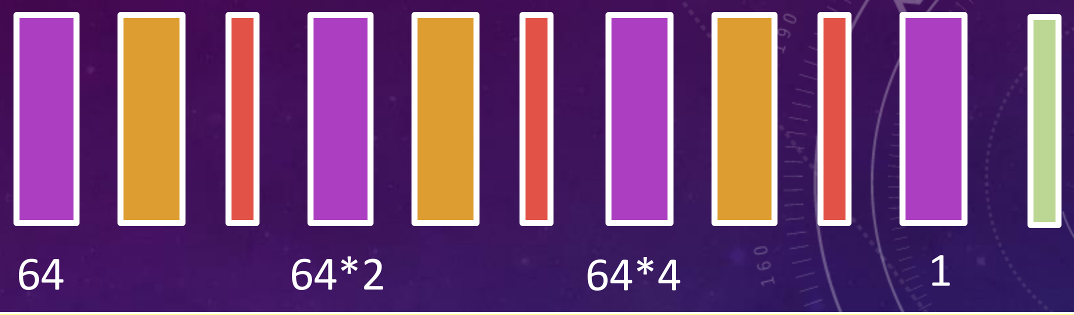
Forward pass

```
torch.nn.ReLU(inplace=False)
```

Applies the rectified linear unit function element-wise:

$$\text{ReLU}(x) = \max(0, x)$$

DCGAN - Discriminator




```
class discriminator(nn.Module):
    # initializers
    def __init__(self, d=64):
        super(discriminator, self).__init__()
        self.conv1 = nn.Conv2d(1, d, 4, 2, 1)
        self.conv1_bn = nn.BatchNorm2d(d)
        self.conv2 = nn.Conv2d(d, d*2, 4, 2, 1)
        self.conv2_bn = nn.BatchNorm2d(d*2)
        self.conv3 = nn.Conv2d(d*2, d*4, 4, 2, 1)
        self.conv3_bn = nn.BatchNorm2d(d*4)
        self.conv4 = nn.Conv2d(d*4, 1, 4, 1, 0)
```

Define the discriminator with nn.Conv2d

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros')
```

```
# weight_init
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

# forward method
    def forward(self, input):
        x = F.leaky_relu(self.conv1_bn(self.conv1(input)), 0.2)
        x = F.leaky_relu(self.conv2_bn(self.conv2(x)), 0.2)
        x = F.leaky_relu(self.conv3_bn(self.conv3(x)), 0.2)
        x = F.sigmoid(self.conv4(x))

        return x
```

Comparison generator and discriminator

DCGAN- Weight initialization, loss and optimizer

```
G = generator(input_size = noise_size)
D = discriminator()
```

Build the instance of our classes generator and discriminator model

```
G.weight_init(mean=0.0, std=0.02)
D.weight_init(mean=0.0, std=0.02)
```

Weight initialization

```
if torch.cuda.is_available():
    G.cuda()
    D.cuda()

print(G)
print(D)

BCE_loss = nn.BCELoss()
```

"binary cross-entropy" as loss function

```
# D_optimizer = torch.optim.Adagrad(D.parameters(), lr=lr)
# G_optimizer = torch.optim.Adagrad(G.parameters(), lr=lr)

# D_optimizer = torch.optim.SGD(D.parameters(), lr=lr,
momentum=0.9)
# G_optimizer = torch.optim.SGD(G.parameters(), lr=lr,
momentum=0.9)

D_optimizer = torch.optim.Adam(D.parameters(), lr=lr,
betas=(0.5, 0.999))
G_optimizer = torch.optim.Adam(G.parameters(), lr=lr,
betas=(0.5, 0.999))
```

Above are 3 choices for the optimizer, default is Adam

DCGAN – Train the "discriminator D"

Start to training the "discriminator" D

```
# train discriminator D
     D.zero_grad()
    mini_batch = x_.size()[0]

    y_real_ = torch.ones(mini_batch)
    y_fake_ = torch.zeros(mini_batch)

    x_, y_real_, y_fake_ = Variable(x_),
     Variable(y_real_), Variable(y_fake_)

    if torch.cuda.is_available():
      x_ = x_.cuda()
      y_real_ = y_real_.cuda()
      y_fake_ = y_fake_.cuda()
```

Discriminate if real images are real or fake

```
    D_result = D(x_).squeeze()
    D_real_loss = BCE_loss(D_result, y_real_)
    D_real_score = D_result
```

```
    z_ = torch.randn((mini_batch, noise_size)).view(-
                1, noise_size, 1, 1)
    z_ = Variable(z_)
    if torch.cuda.is_available():
      z_ = z_.cuda()
```

Generate the noises

```
    G_result = G(z_)
```

Generate the images from the noises

```
    D_result = D(G_result).squeeze()
    D_fake_loss = BCE_loss(D_result, y_fake_)
    D_fake_score = D_result.data.mean()

    D_train_loss = D_real_loss + D_fake_loss

    D_train_loss.backward()
    D_optimizer.step()
```

DCGAN Train the Generator

```
# train generator G
    G.zero_grad()
```

Generate the noises

```
    z_ = torch.randn((mini_batch, noise_size)).view(-
    1, noise_size, 1, 1)


    z_ = Variable(z_)
    if torch.cuda.is_available():
        z_ = z_.cuda()

    G_result = G(z_)
```

Generate the images from the noises

```
    D_result = D(G_result).squeeze()
```

Discriminate the fake images are real or fake

```
    G_train_loss = BCE_loss(D_result, y_real_)
    G_train_loss.backward()
    G_optimizer.step()
```

DCGAN Results

```
if epoch % 1 == 0:
    save_path = './gan_img/output_{}.png'.format(epoch)
    show_result((epoch+1), save=True, isFix=True)
```

Save the output images

- Result:

Epoch 1 :

Epoch 10 :

Epoch 30 :

~20 mins

Exercise 4-4: DC-GAN

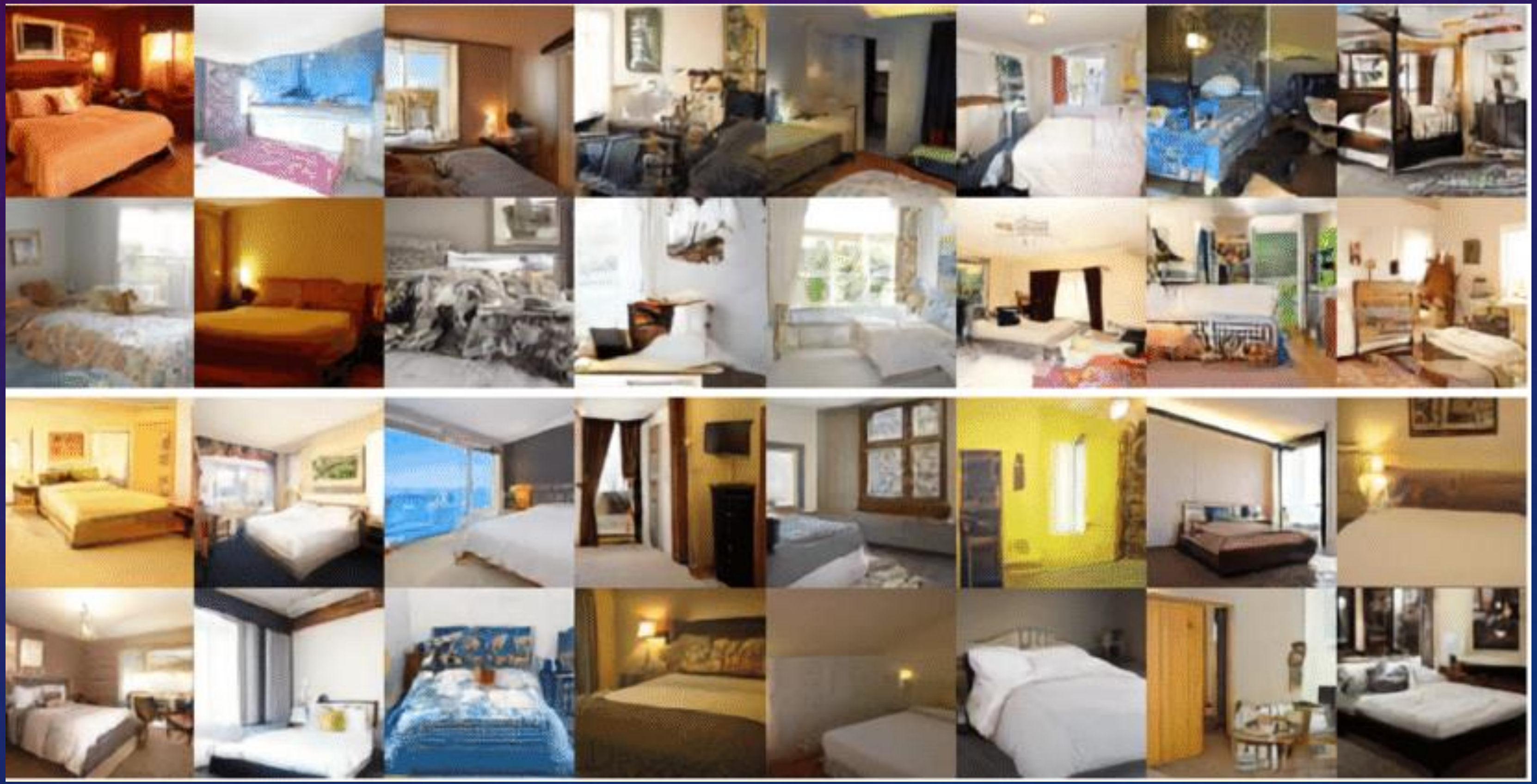
- Please download the “exercise4.4_ DCGAN.ipynb” on Moodle.
- Upload the “exercise4.4_ DCGAN.ipynb” to the Google Colab.
- Follow the sample code to understand the data flow of the generative adversarial network.
  - ➢ Compare the architecture and results with the exercise4-2.
  - ➢ Try to implement the gan loss function from exercise4-3.
  - ➢ Compare the training results by using “wgan-gp”.
  - ➢ given the trajectories of the generator loss and discriminator loss, please write down your description

Face Dataset - CelebA

- CelebA is a dataset of 202,599 face images with 10,177 celebrity identities, which provides 5 face key points (facial landmarks) coordinates and 40 facial attributes

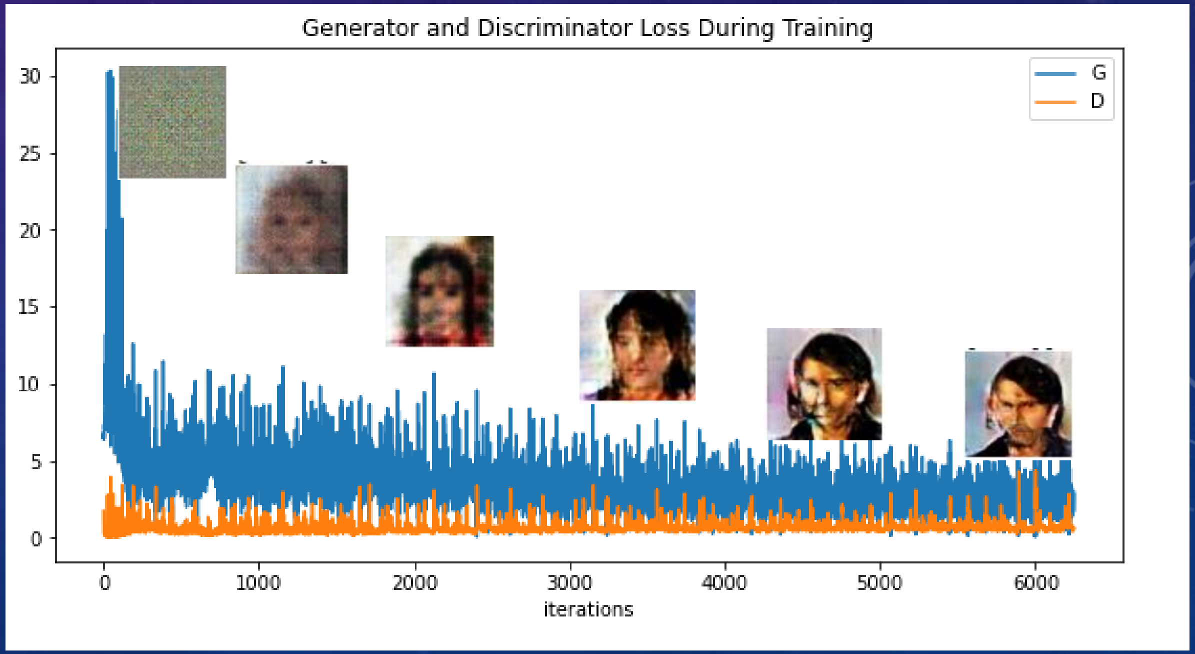
Bedroom Dataset - LSUN

- It contains around one million labeled images for each of 10 scene categories and 20 object categories

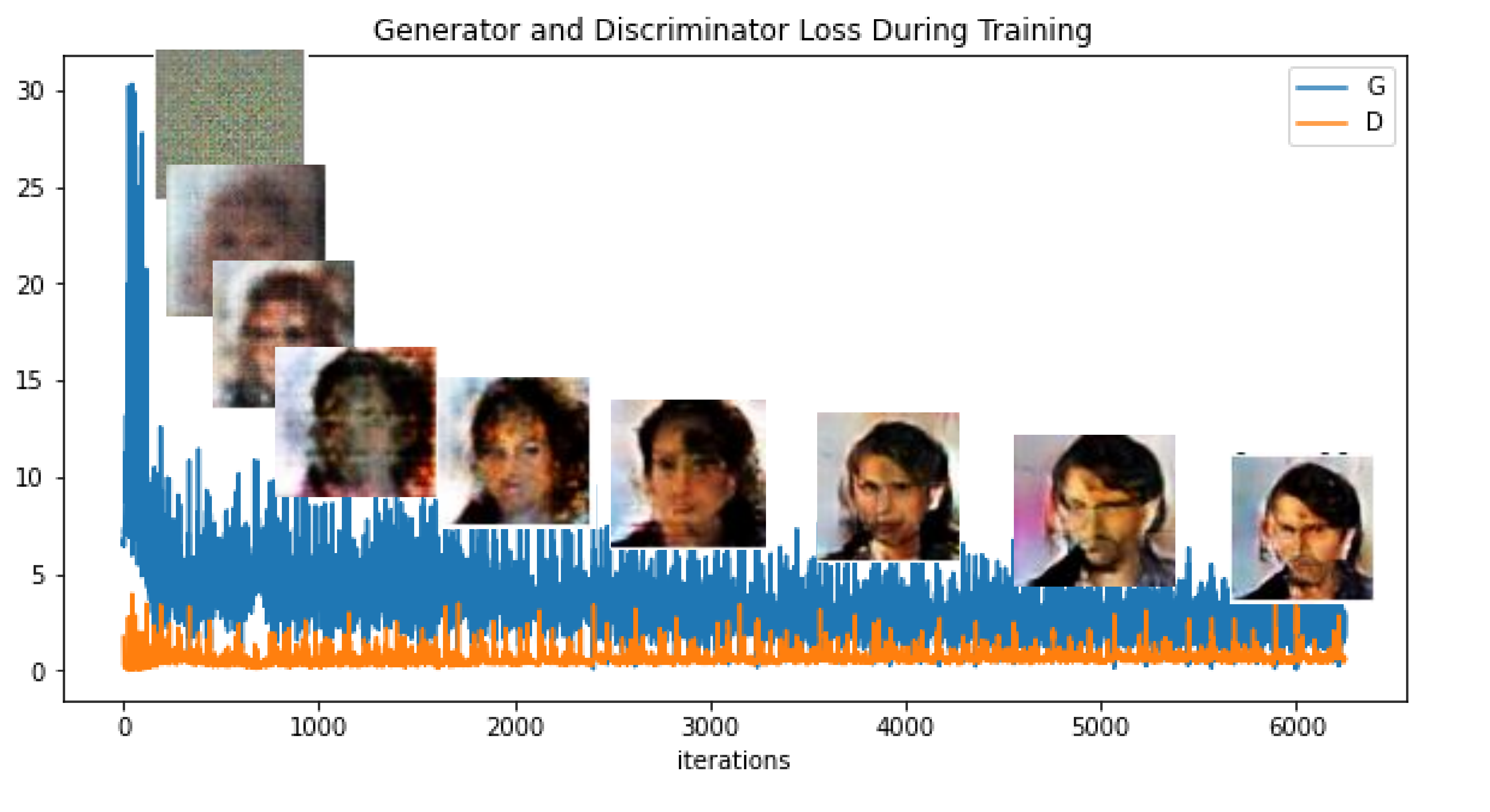


Example 4-5: DC-GAN - Face

- Please download the "exercise4.5_CelebA_DCGAN.ipynb" on Moodle.
- Upload the "exercise4.5_CelebA_DCGAN.ipynb" to the Google Colab.
- Follow the sample code to understand the data flow of the generative adversarial network.
  - Download the CelebA dataset given the code on the Colab.
  - Define the structure of generator and discriminator
  - Define the loss and optimizer
  - Train the DC-GAN model for celebA
  - Use the fixed noise to observe each generated image during training process

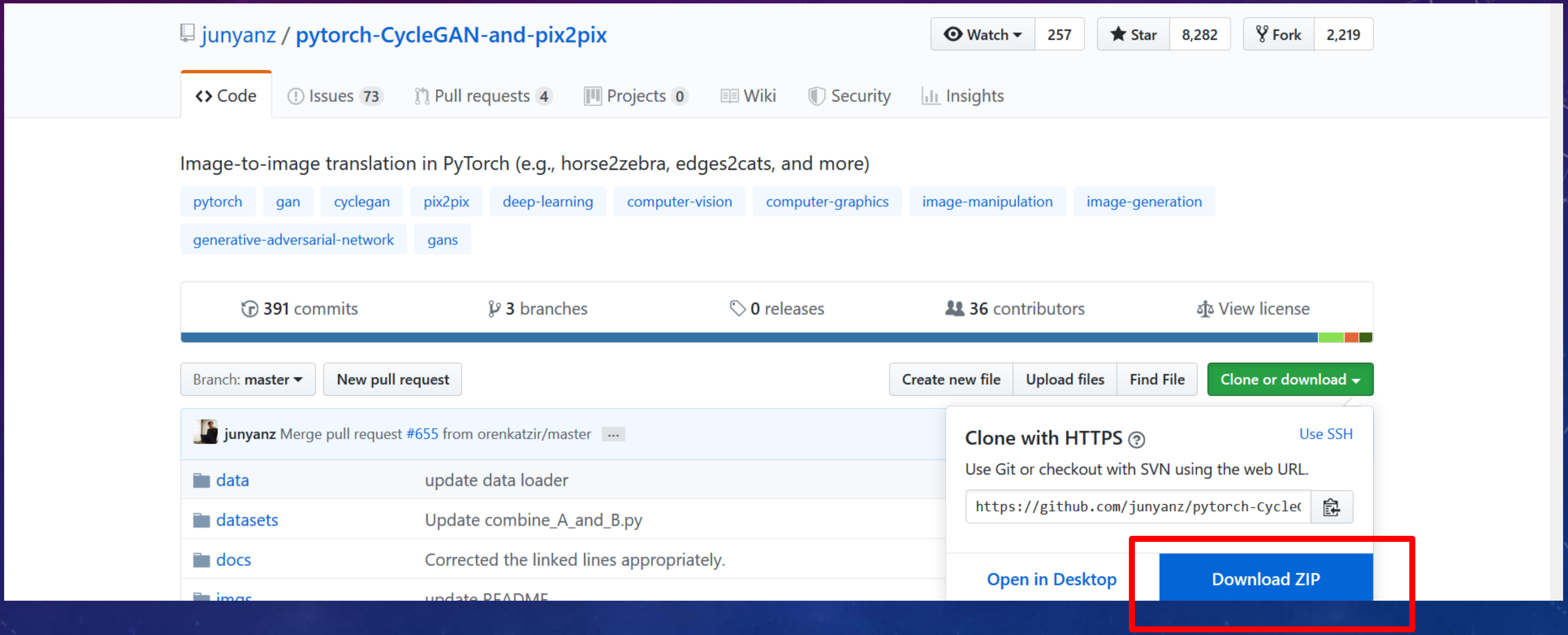


Example 4-5: DC-GAN - Face



Example 4-5: DC-GAN - Face

CycleGAN



1. Please visit https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

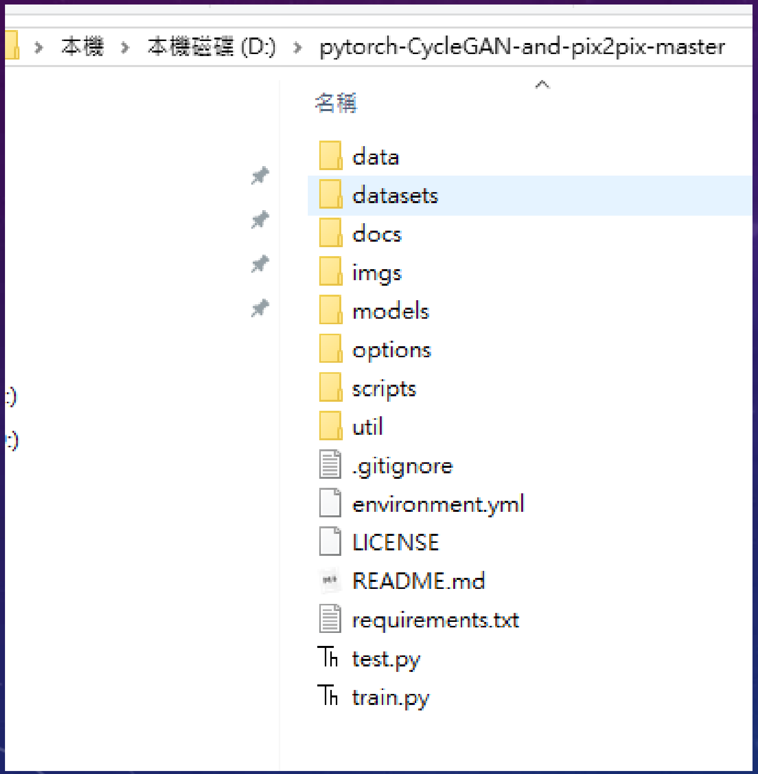
2. Download the zip file of the codes

Exercise 4-5: DC-GAN - LSUN

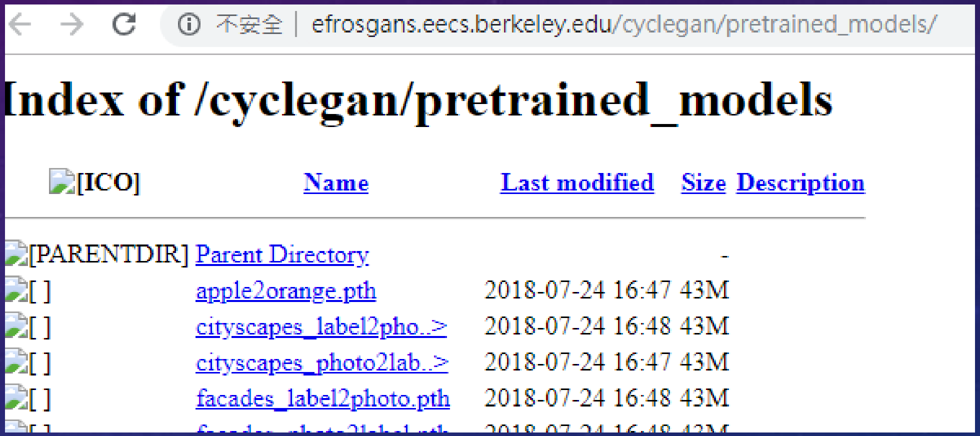
- Please download the "exercise4.5_DCGAN.ipynb" on Moodle.
- Upload the "exercise4.5_DCGAN.ipynb" to the Google Colab.
- Follow the sample code to understand the data flow of the generative adversarial network.
  - ➢ Download the LSUN dataset given the code on the Colab.
  - ➢ Define the structure of generator and discriminator which are same as exercise 4-4.
  - ➢ Define the loss and optimizer which are same as exercise 4-4.
  - ➢ Train the DC-GAN model on LSUN dataset.
  - ➢ Use the fixed noise to observe each generated image during training process.

CycleGAN

3.Unzip your code and put it in the directory desired.



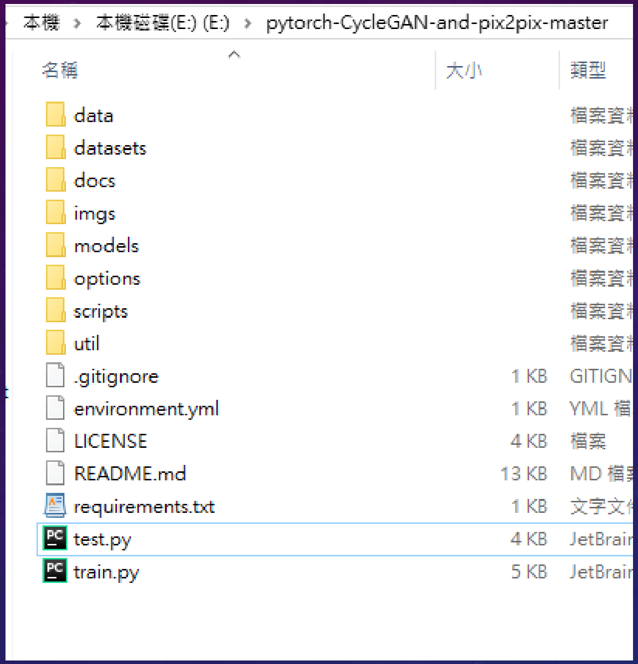
4. Visit http://efrosgans.eecs.berkeley.edu/cyclegan/pretrained_models/ for pretrained models



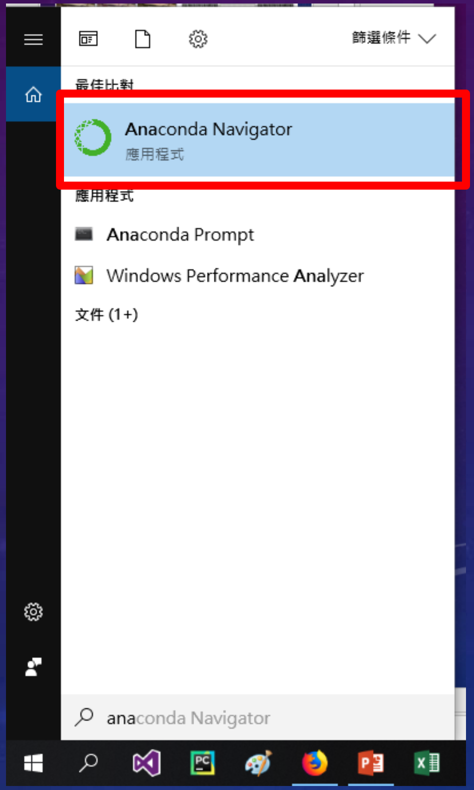
Let’s take appe2orange for example.

CycleGAN

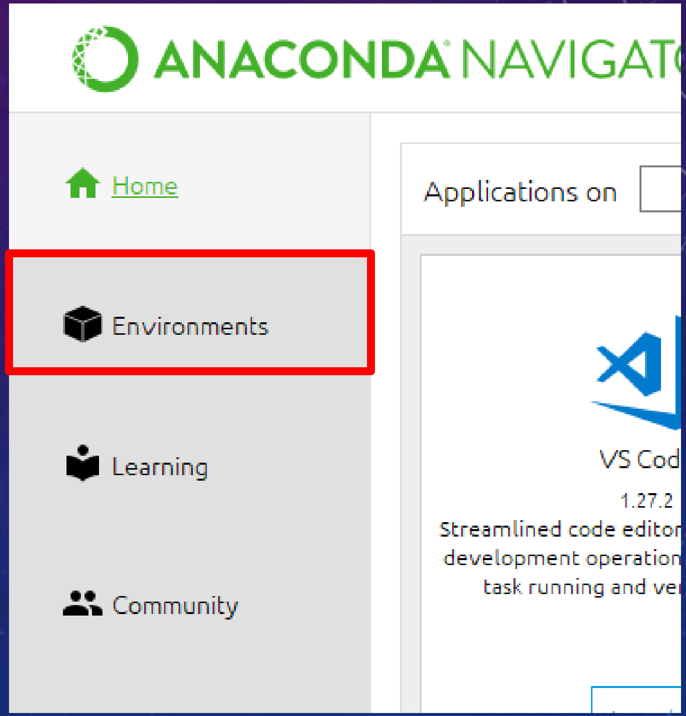
1. Extract your files to D:/



2. Open your Anaconda



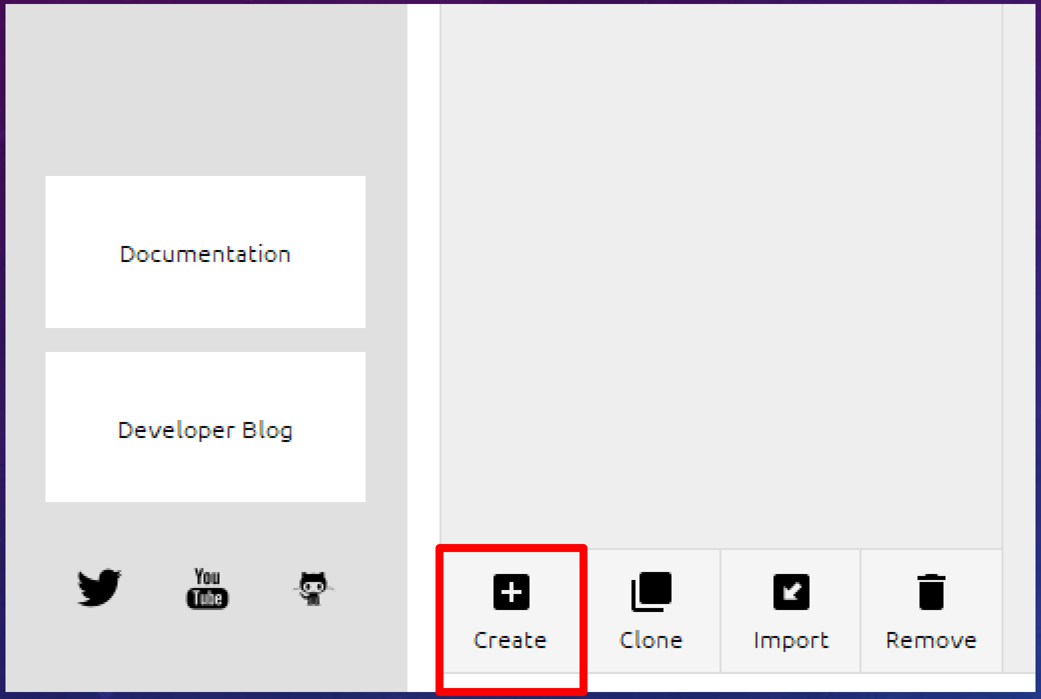
3. Click Environment



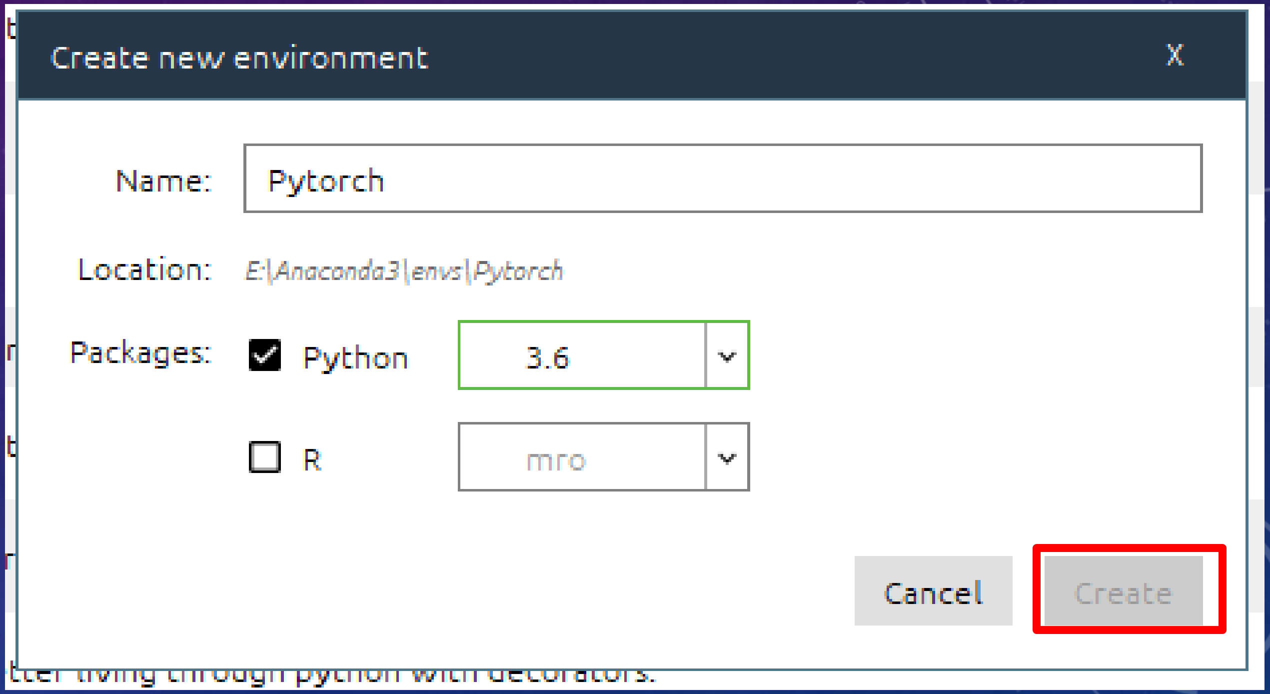
http://efrosgans.eecs.berkeley.edu/cyclegan/pretrained_models/

CycleGAN

4.Create a new environment.



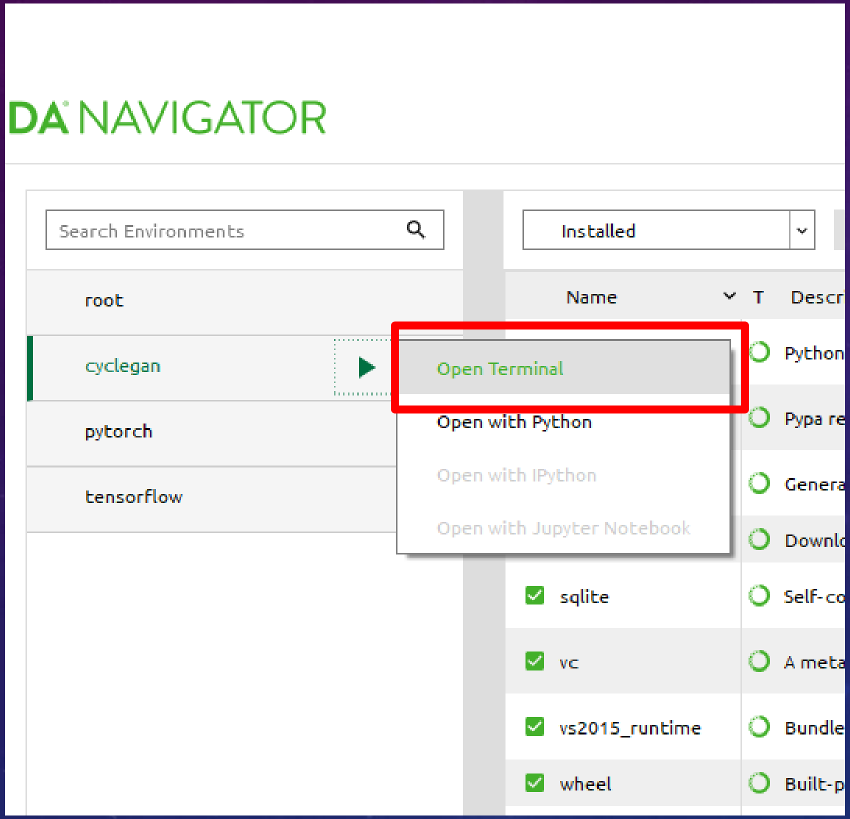
click



click

CycleGAN

5. After building steps, open your terminal.



6. Enter the folder where you put the codes

```
(C:\Anaconda3\envs\cyclegan) C:\Users\c_user>D:

(C:\Anaconda3\envs\cyclegan) D:\>cd pytorch-CycleGAN-and-pix2pix-master

(C:\Anaconda3\envs\cyclegan) D:\pytorch-CycleGAN-and-pix2pix-master>
```

7. Enter the command

"conda install pytorch=0.4.1 cuda80 –c pytorch"

8. Enter "y" for the permission

```
Proceed ([y]/n)? y
```

CycleGAN

Scipy : Please install the version of 1.2.1

```
pytorch-CycleGAN-and-pix2pix-master>pip install scipy==1.2.1
```

And other packages

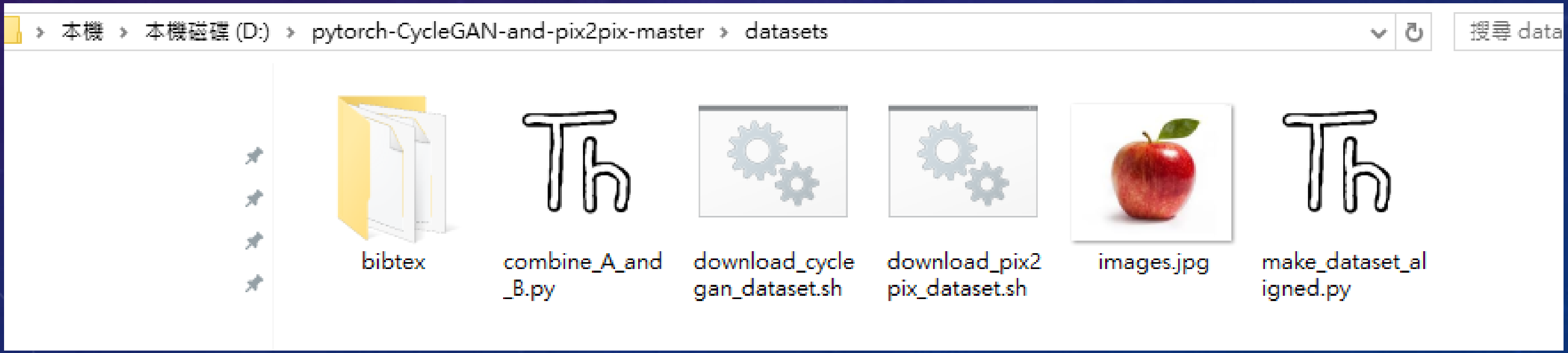
```
(C:\Anaconda3\envs\cyclegan) D:\pytorch-CycleGAN-and-pix2pix-master>pip install dominate pillow
```

```
(C:\Anaconda3\envs\cyclegan) D:\pytorch-CycleGAN-and-pix2pix-master>pip install torchvision
```

CycleGAN

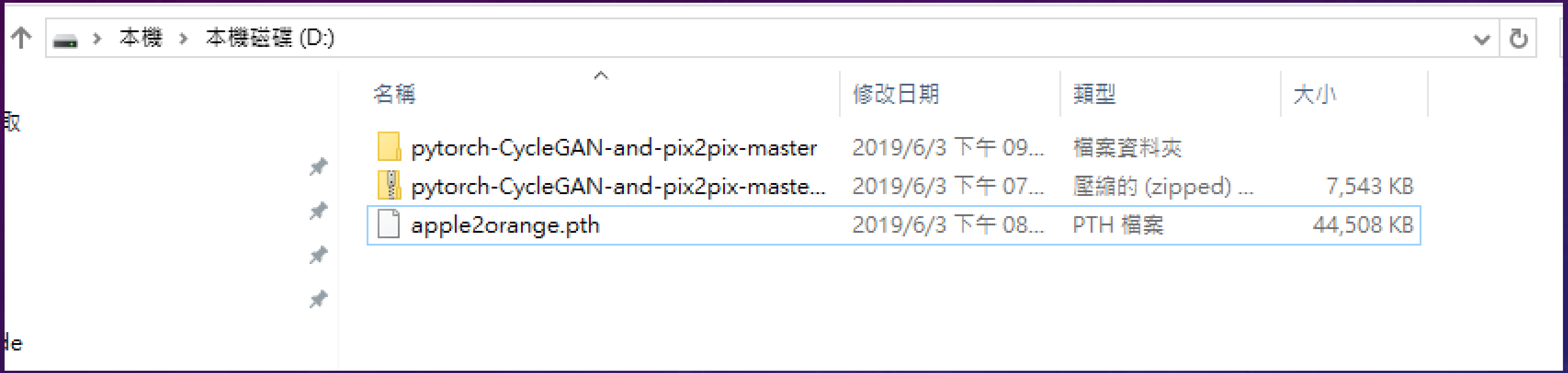
We will take “apple2orange” for example.

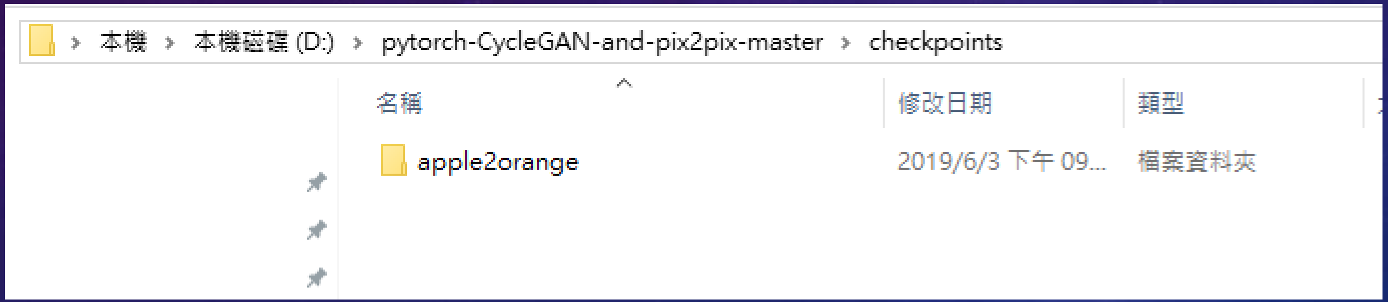
Please put your picture here



The code will find the “.jpg” file by itself.

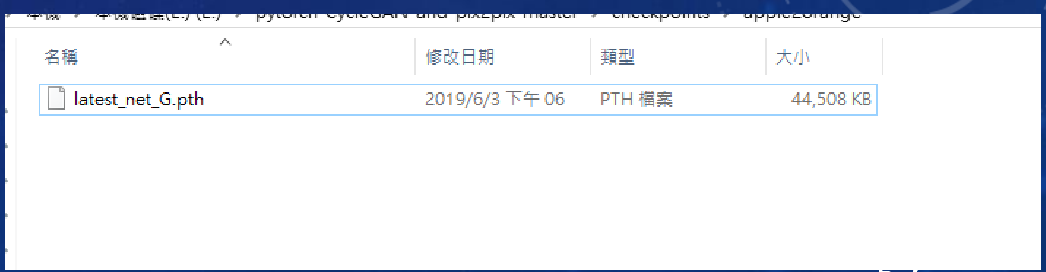
CycleGAN





1. Make the directory in <CycleGAN dir>/checkpoints/apple2orange
2. Copy it and paste it to <CycleGAN dir>/checkpoints/apple2orange/

Then , rename it as “latest_net_G.pth”



CycleGAN

- Enter the command <python test.py --dataroot datasets --model test --name apple2orange --no_dropout>

```
>python test.py --dataroot datasets --model test --name apple2orange --no_dropout --gpu_ids -1
```

And you will get your results in the folder (pytorch-CycleGAN-and-pix2pix-master\results\apple2orange\test_latest\images)

Exercise 4-6: CycleGAN

- Please download the pre-trained model from the following link http://efrosgans.eecs.berkeley.edu/cyclegan/pretrained_models/
- Take arbitrary **face** image as input, please use the monet2photo.pth, style_vangogh.pth and stytle_cezanne.pth pretrained models, and add Monet/Vangogh/Cezanne-style to the input image.
- Utilize the same image as input, and use orange2apple.pth as the pretrained model to transfer the style of the input image.
- Write down what you observe in the docx file.