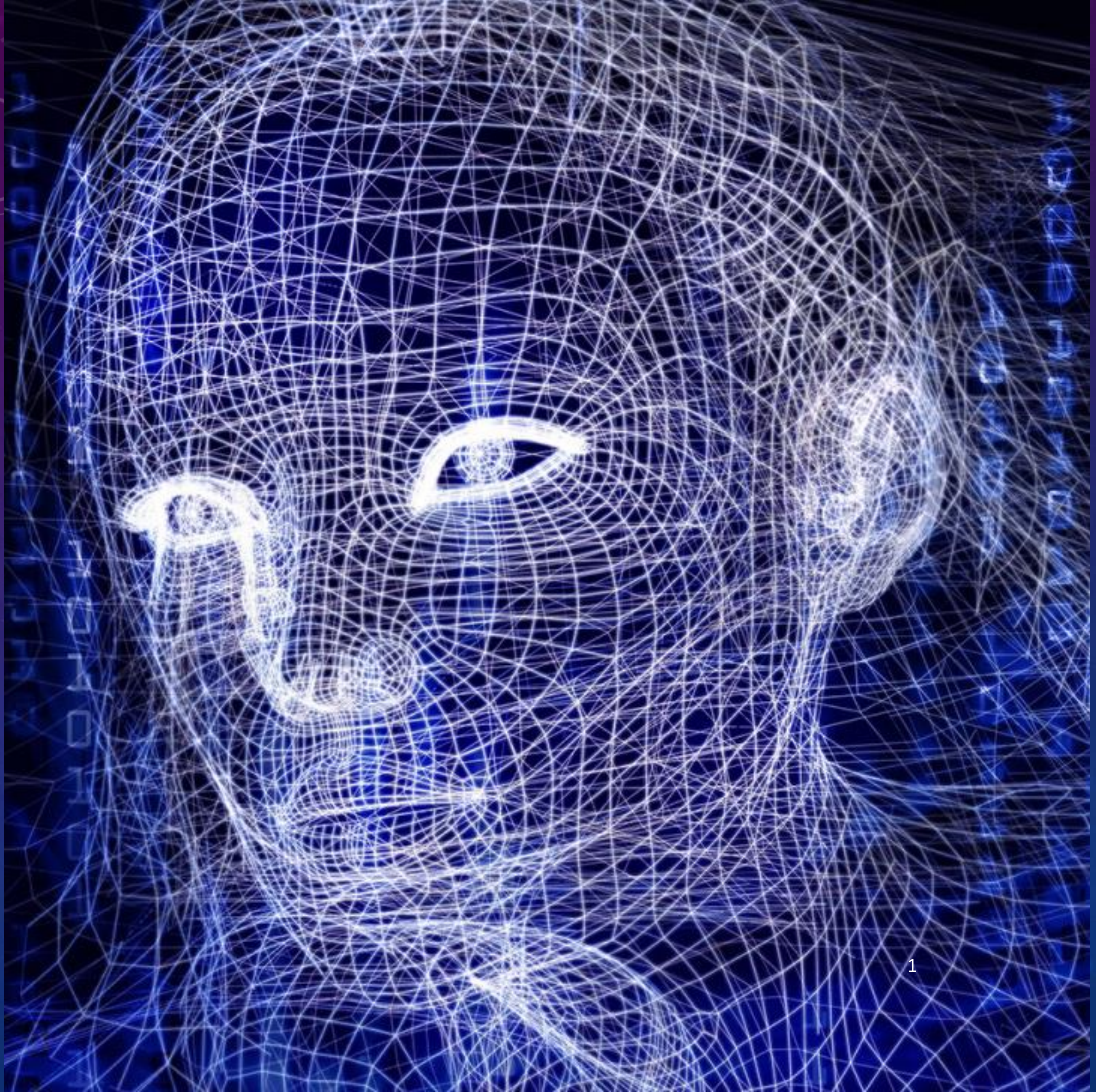
COMPUTER VISION AND ITS APPLICATIONS

CH 4_EXERCISE

徐繼聖

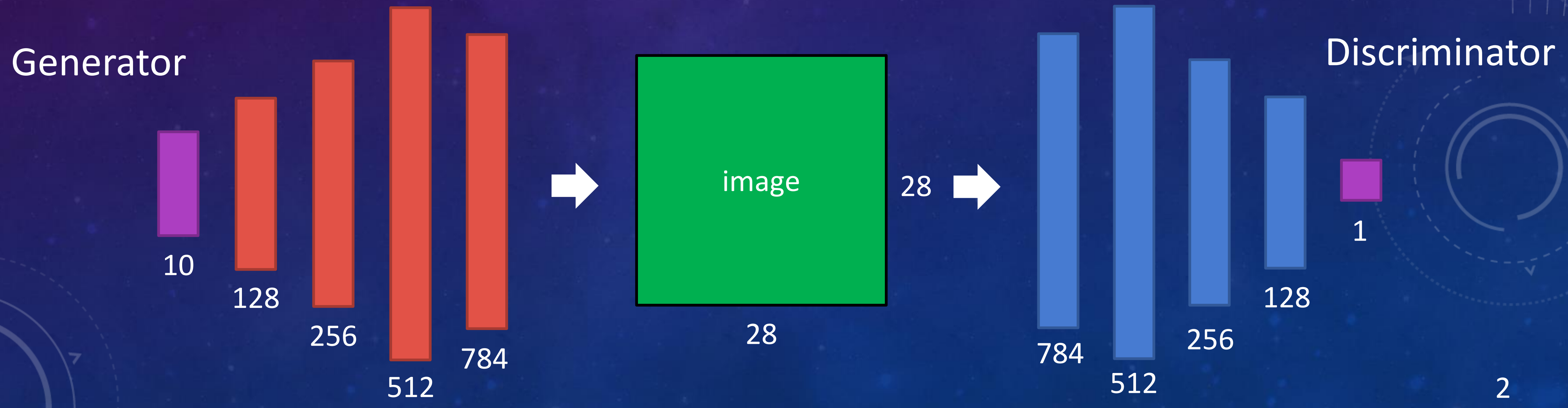
Gee-Sern Jison Hsu

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.MNIST(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 Mnist 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
```

Define 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
```

Define 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)

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

print(G)
print(D)


BCE_loss = nn.BCELoss()

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))
```

Build a model

Use “binary cross-entropy” as loss function

Example 4-1: Generative Adversarial Network

Start to training the “discriminator” D

Generate the noises

Flatten the real images

```
# 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
```

Discriminate the real images are real or fake

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

Generate the images by the noises

```
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()
```

Example 4-1: Generative Adversarial Network

Start to training the “generator” G

```
# train generator G
    G.zero_grad()
                                    Generate the noises
    z_ = torch.randn((mini_batch, noise_size))
    y_ = torch.ones(mini_batch)

    z_, y_ = Variable(z_), Variable(y_)
    if torch.cuda.is_available():
        z_ = z_.cuda()
        y_ = y_.cuda()              Generate the images by the noises
    G_result = G(z_)
    D_result = D(G_result)             Discriminate the fake images are real or fake
    G_train_loss = BCE_loss(D_result, y_)
    G_train_loss.backward()
    G_optimizer.step()
```

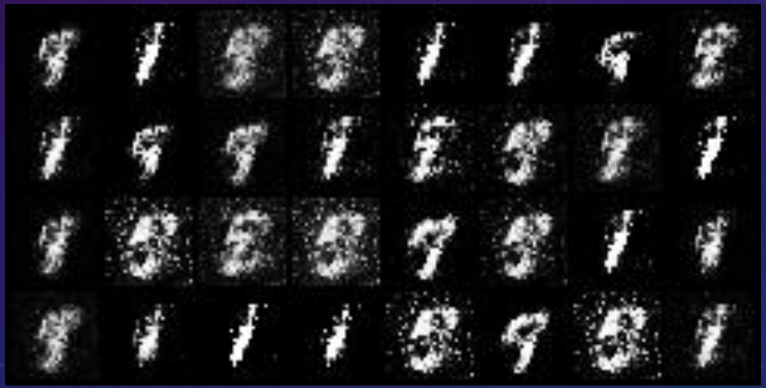
Example 4-1: Generative Adversarial Network

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

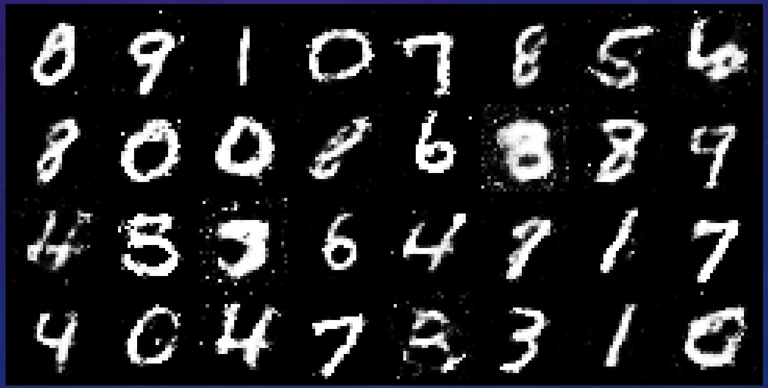
Save the output images

- Result:

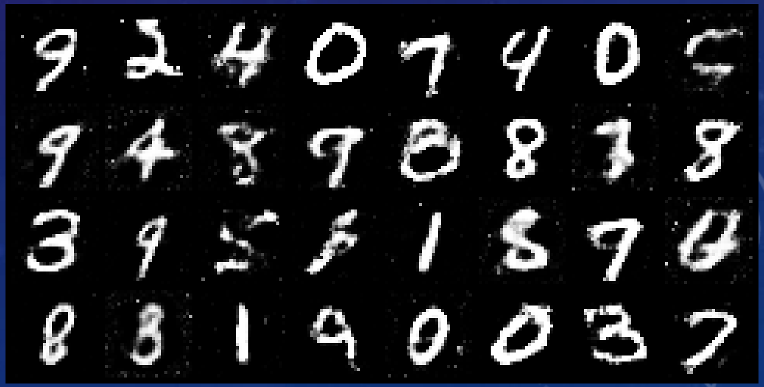
Epoch 1 :



Epoch 30 :



Epoch 50 :



~30 mins

Excercise 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 discriminator 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.