GNN PyTorch MNIST 實作

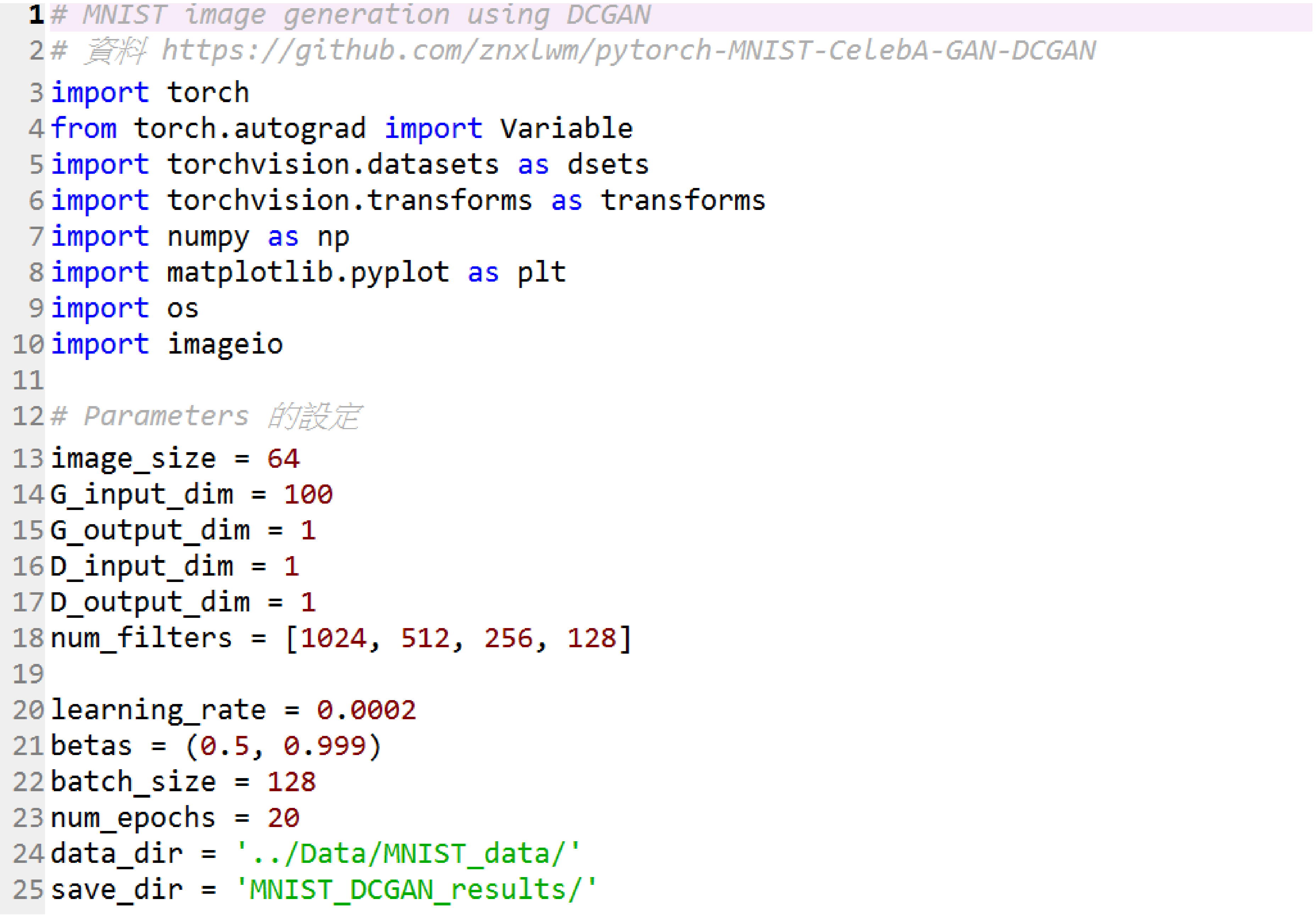
金門大學資工系 馮玄明整理

實作專案 GAN 實作 MNIST

- 對象:大學與研究所初學者
- 目的:學習 Python 設計GAN完成數位手寫影像生成的應用
- 來源 https://github.com/znxlwm/pytorch-MNIST-CelebA-GAN-DCGAN

09 GAN 實作 MNIST

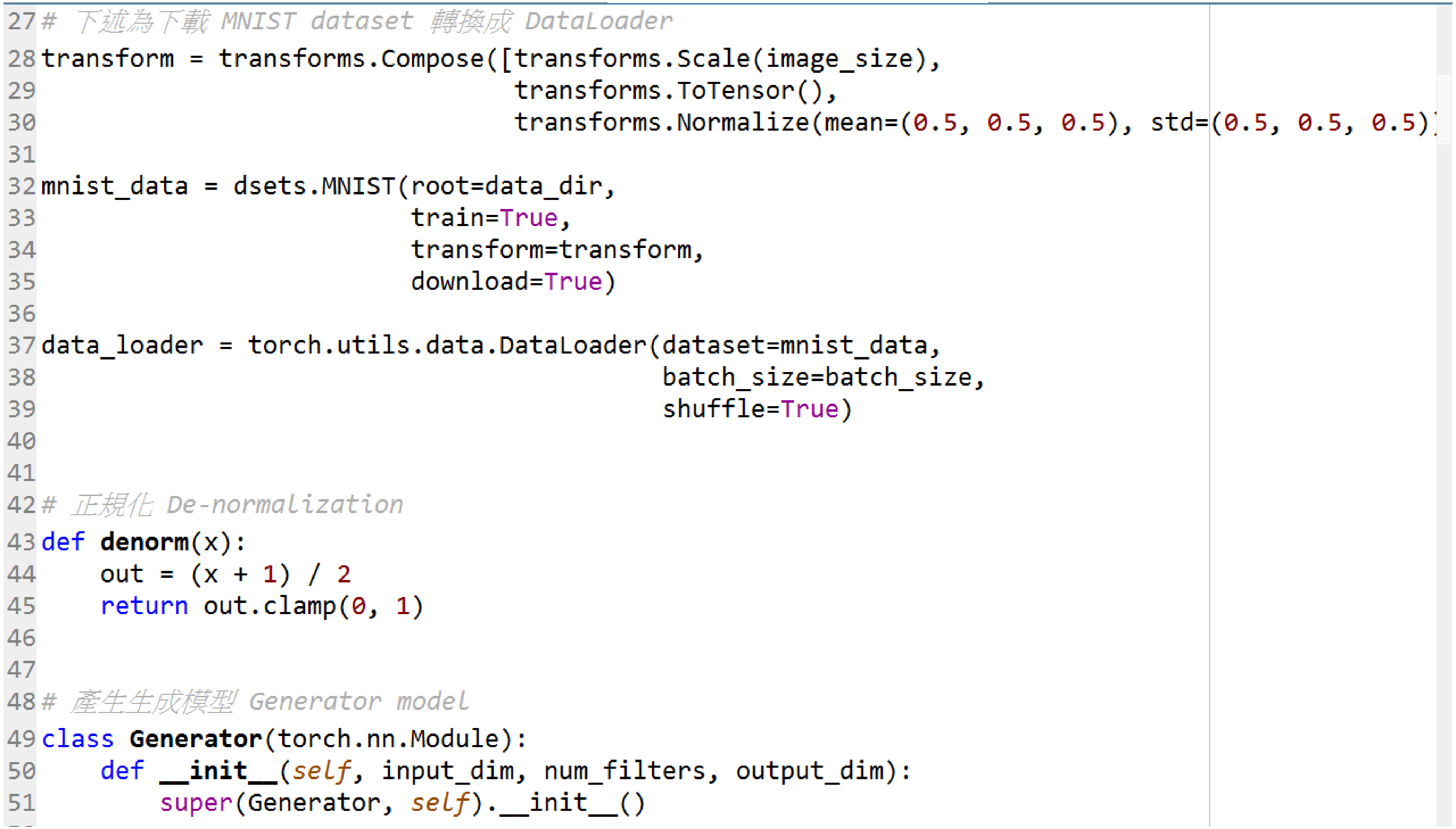
程式碼解說1



```python
# MNIST image generation using DCGAN
# 資料 https://github.com/znxlwm/pytorch-MNIST-CelebA-GAN-DCGAN
import torch
from torch.autograd import Variable
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
import os
import imageio

# Parameters 的設定
image_size = 64
G_input_dim = 100
G_output_dim = 1
D_input_dim = 1
D_output_dim = 1
num_filters = [1024, 512, 256, 128]

learning_rate = 0.0002
betas = (0.5, 0.999)
batch_size = 128
num_epochs = 20
data_dir = '../Data/MNIST_data/'
save_dir = 'MNIST_DCGAN_results/'
```

09 GAN 實作 MNIST

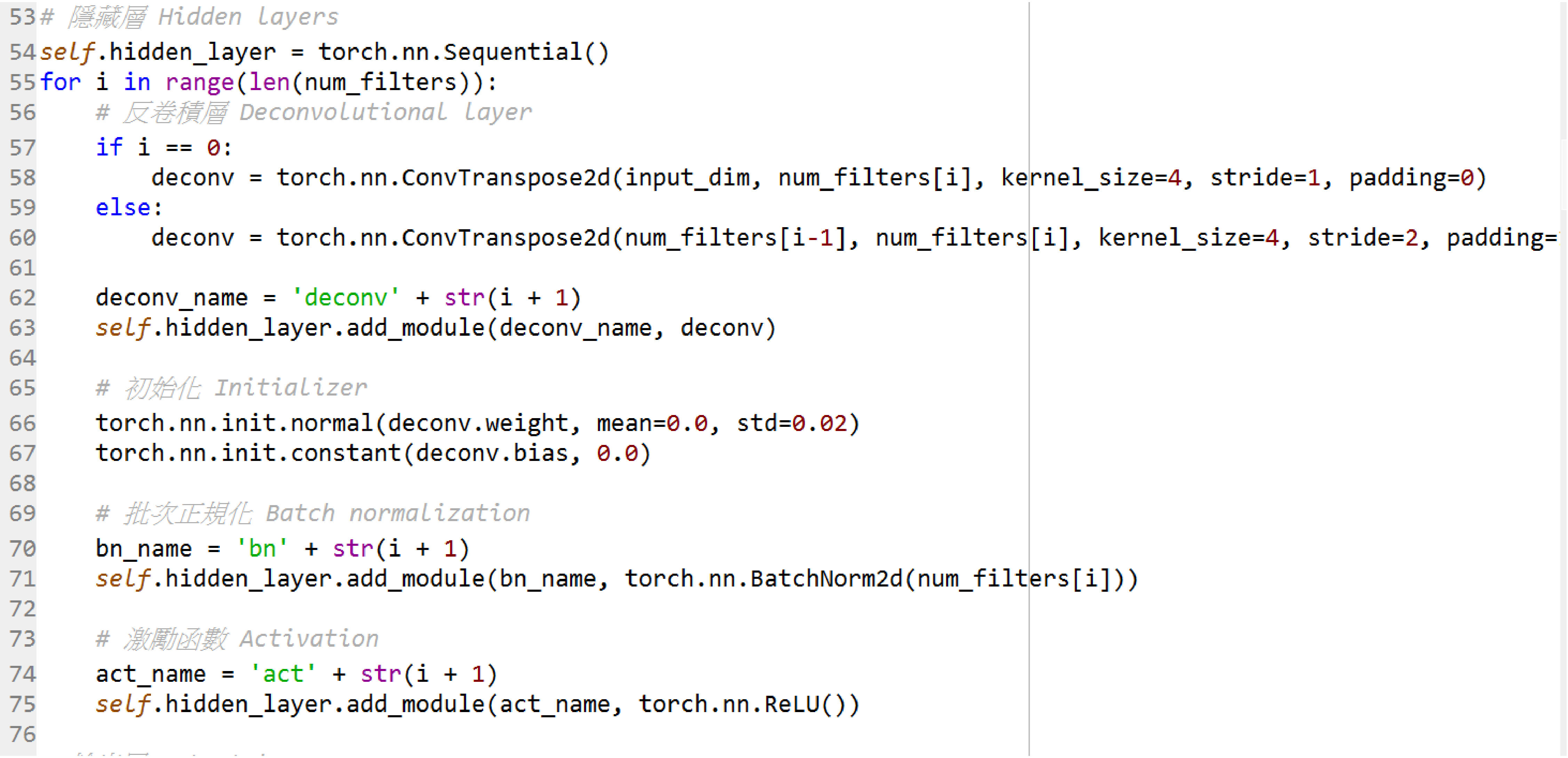
程式碼解說2



```python
# 下述為下載 MNIST dataset 轉換成 DataLoader
transform = transforms.Compose([transforms.Scale(image_size),
                                transforms.ToTensor(),
                                transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)

mnist_data = dsets.MNIST(root=data_dir,
                         train=True,
                         transform=transform,
                         download=True)

data_loader = torch.utils.data.DataLoader(dataset=mnist_data,
                                          batch_size=batch_size,
                                          shuffle=True)


# 正規化 De-normalization
def denorm(x):
    out = (x + 1) / 2
    return out.clamp(0, 1)


# 產生生成模型 Generator model
class Generator(torch.nn.Module):
    def __init__(self, input_dim, num_filters, output_dim):
        super(Generator, self).__init__()
```

09 GAN 實作 MNIST

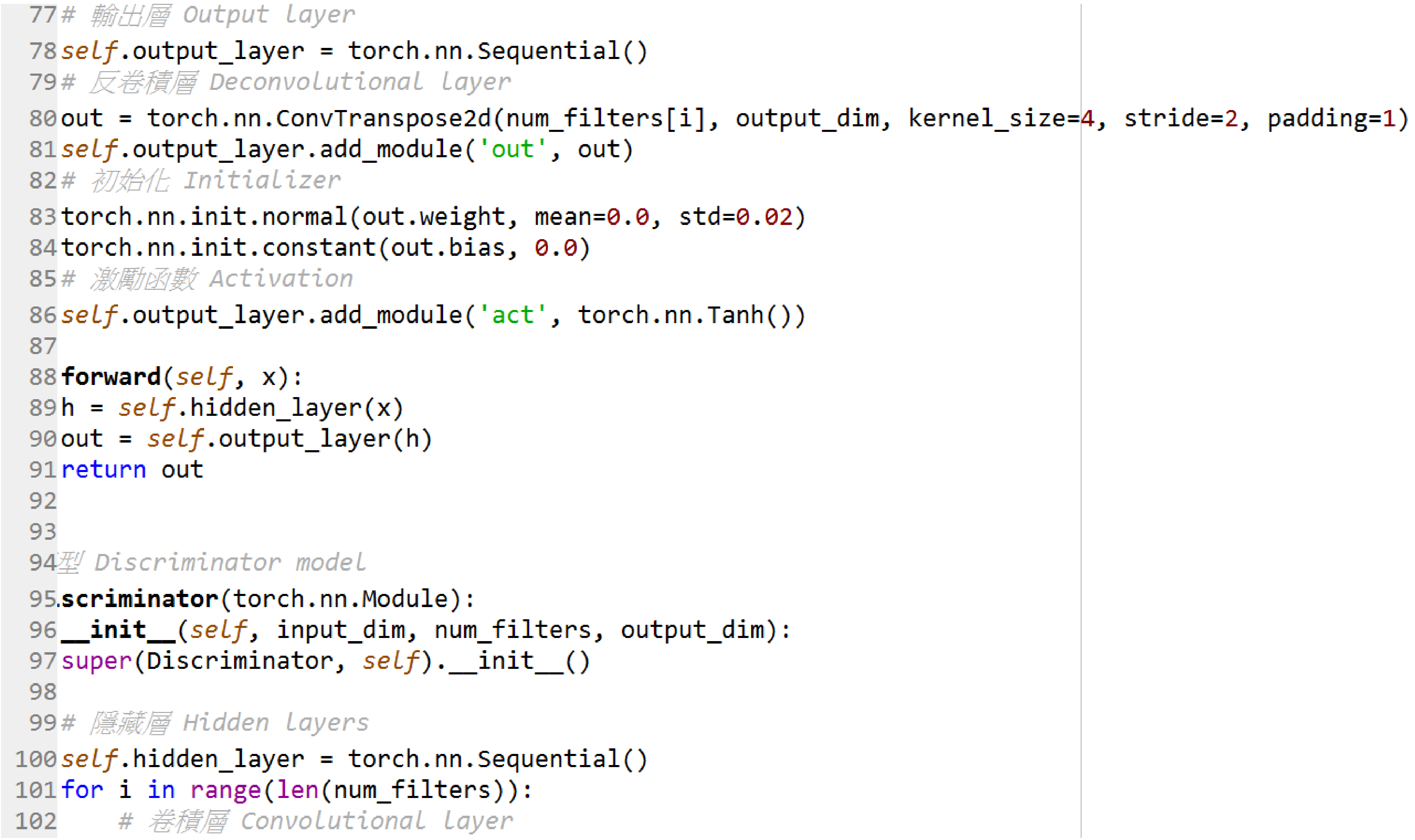
程式碼解說3



```
# 隱藏層 Hidden layers
self.hidden_layer = torch.nn.Sequential()
for i in range(len(num_filters)):
    # 反卷積層 Deconvolutional layer
    if i == 0:
        deconv = torch.nn.ConvTranspose2d(input_dim, num_filters[i], kernel_size=4, stride=1, padding=0)
    else:
        deconv = torch.nn.ConvTranspose2d(num_filters[i-1], num_filters[i], kernel_size=4, stride=2, padding=

    deconv_name = 'deconv' + str(i + 1)
    self.hidden_layer.add_module(deconv_name, deconv)

    # 初始化 Initializer
    torch.nn.init.normal(deconv.weight, mean=0.0, std=0.02)
    torch.nn.init.constant(deconv.bias, 0.0)

    # 批次正規化 Batch normalization
    bn_name = 'bn' + str(i + 1)
    self.hidden_layer.add_module(bn_name, torch.nn.BatchNorm2d(num_filters[i]))

    # 激勵函數 Activation
    act_name = 'act' + str(i + 1)
    self.hidden_layer.add_module(act_name, torch.nn.ReLU())

```

09 GAN 實作 MNIST

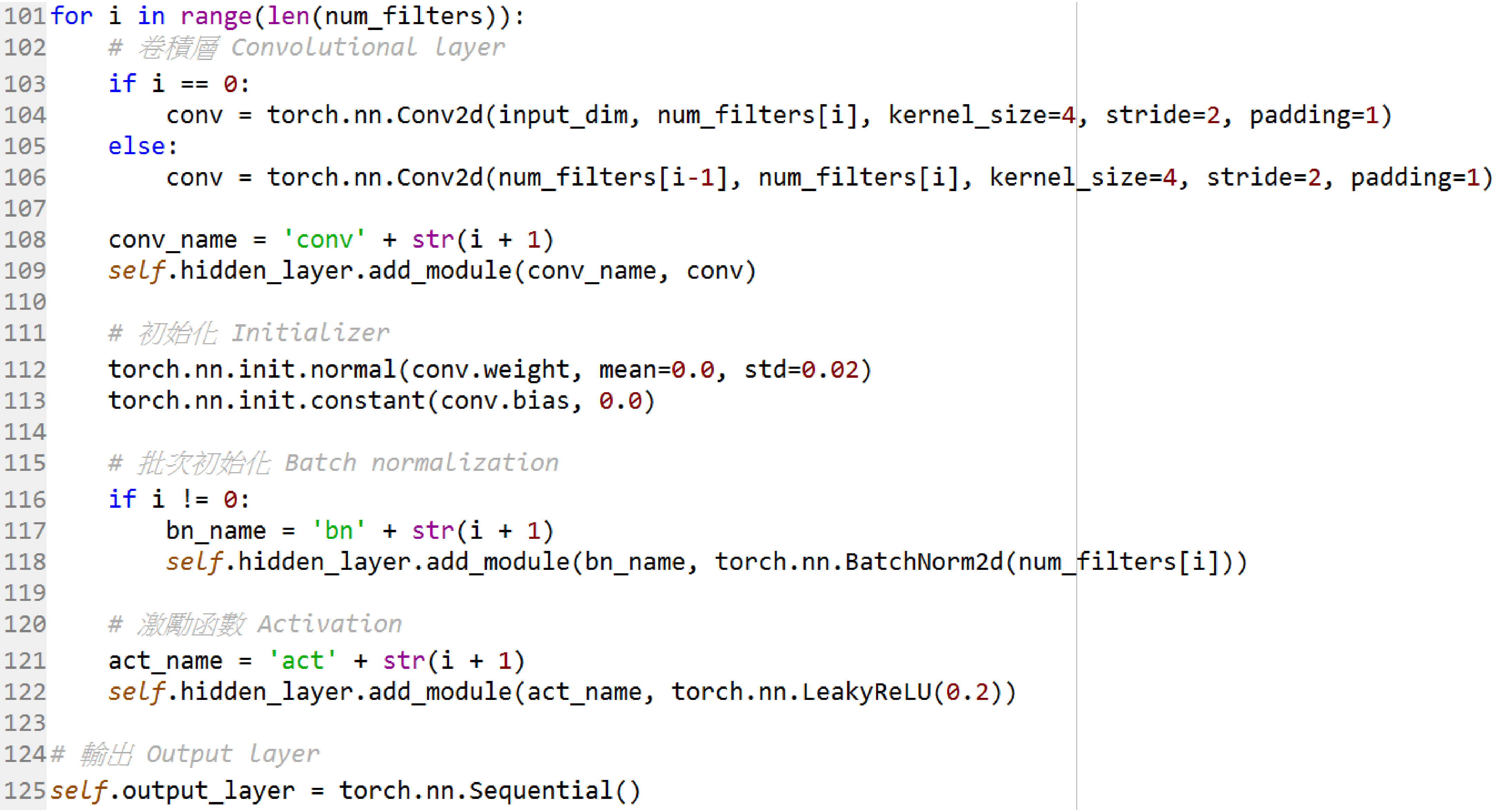
程式碼解說4



```
# 輸出層 Output layer
self.output_layer = torch.nn.Sequential()
# 反卷積層 Deconvolutional layer
out = torch.nn.ConvTranspose2d(num_filters[i], output_dim, kernel_size=4, stride=2, padding=1)
self.output_layer.add_module('out', out)
# 初始化 Initializer
torch.nn.init.normal(out.weight, mean=0.0, std=0.02)
torch.nn.init.constant(out.bias, 0.0)
# 激勵函數 Activation
self.output_layer.add_module('act', torch.nn.Tanh())

forward(self, x):
h = self.hidden_layer(x)
out = self.output_layer(h)
return out


型 Discriminator model
scriminator(torch.nn.Module):
__init__(self, input_dim, num_filters, output_dim):
super(Discriminator, self).__init__()

# 隱藏層 Hidden layers
self.hidden_layer = torch.nn.Sequential()
for i in range(len(num_filters)):
    # 卷積層 Convolutional layer
```

09 GAN 實作 MNIST

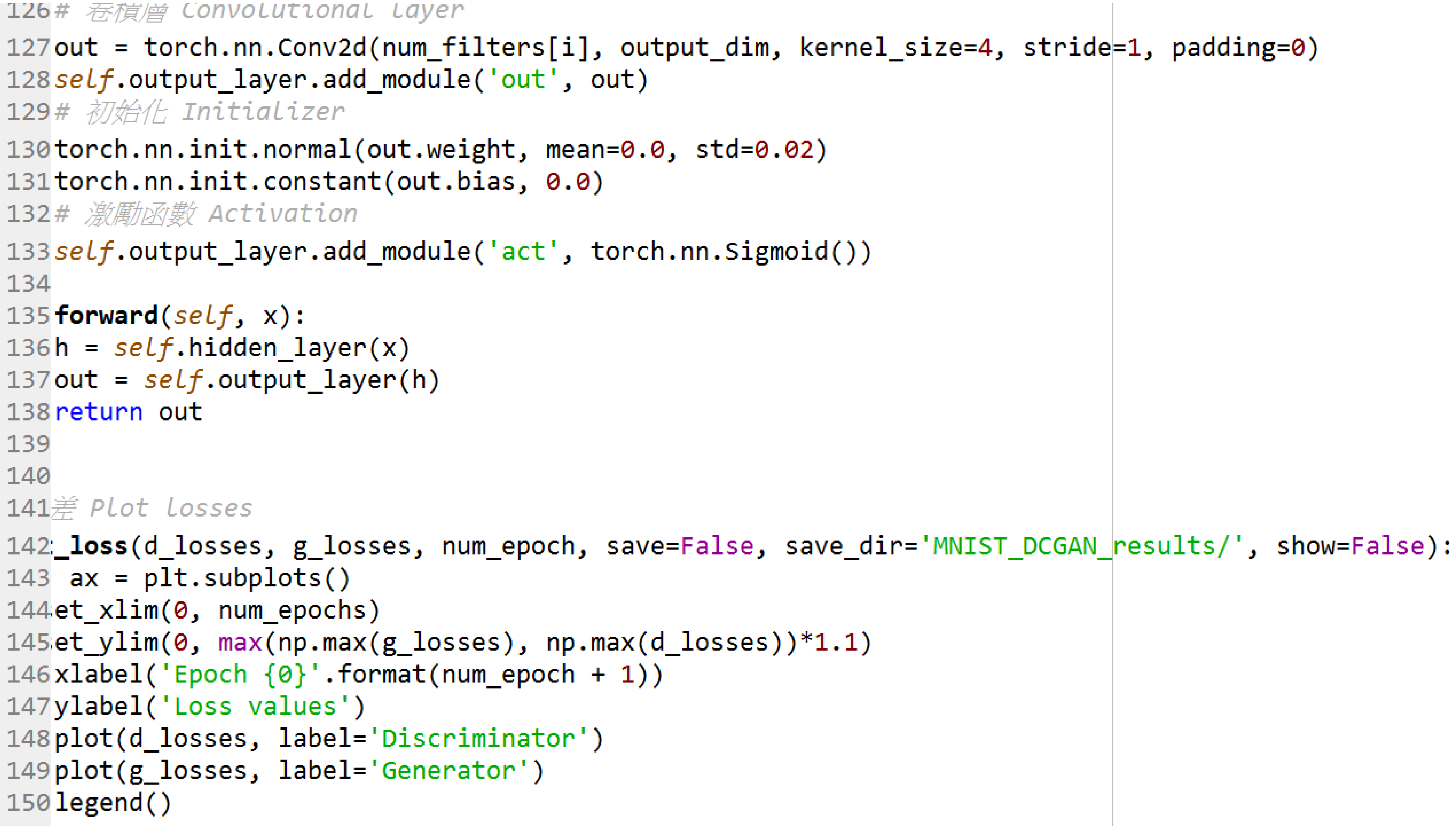
程式碼解說5



```
for i in range(len(num_filters)):
    # 卷積層 Convolutional layer
    if i == 0:
        conv = torch.nn.Conv2d(input_dim, num_filters[i], kernel_size=4, stride=2, padding=1)
    else:
        conv = torch.nn.Conv2d(num_filters[i-1], num_filters[i], kernel_size=4, stride=2, padding=1)

    conv_name = 'conv' + str(i + 1)
    self.hidden_layer.add_module(conv_name, conv)

    # 初始化 Initializer
    torch.nn.init.normal(conv.weight, mean=0.0, std=0.02)
    torch.nn.init.constant(conv.bias, 0.0)

    # 批次初始化 Batch normalization
    if i != 0:
        bn_name = 'bn' + str(i + 1)
        self.hidden_layer.add_module(bn_name, torch.nn.BatchNorm2d(num_filters[i]))

    # 激勵函數 Activation
    act_name = 'act' + str(i + 1)
    self.hidden_layer.add_module(act_name, torch.nn.LeakyReLU(0.2))

# 輸出 Output layer
self.output_layer = torch.nn.Sequential()
```

09 GAN 實作 MNIST

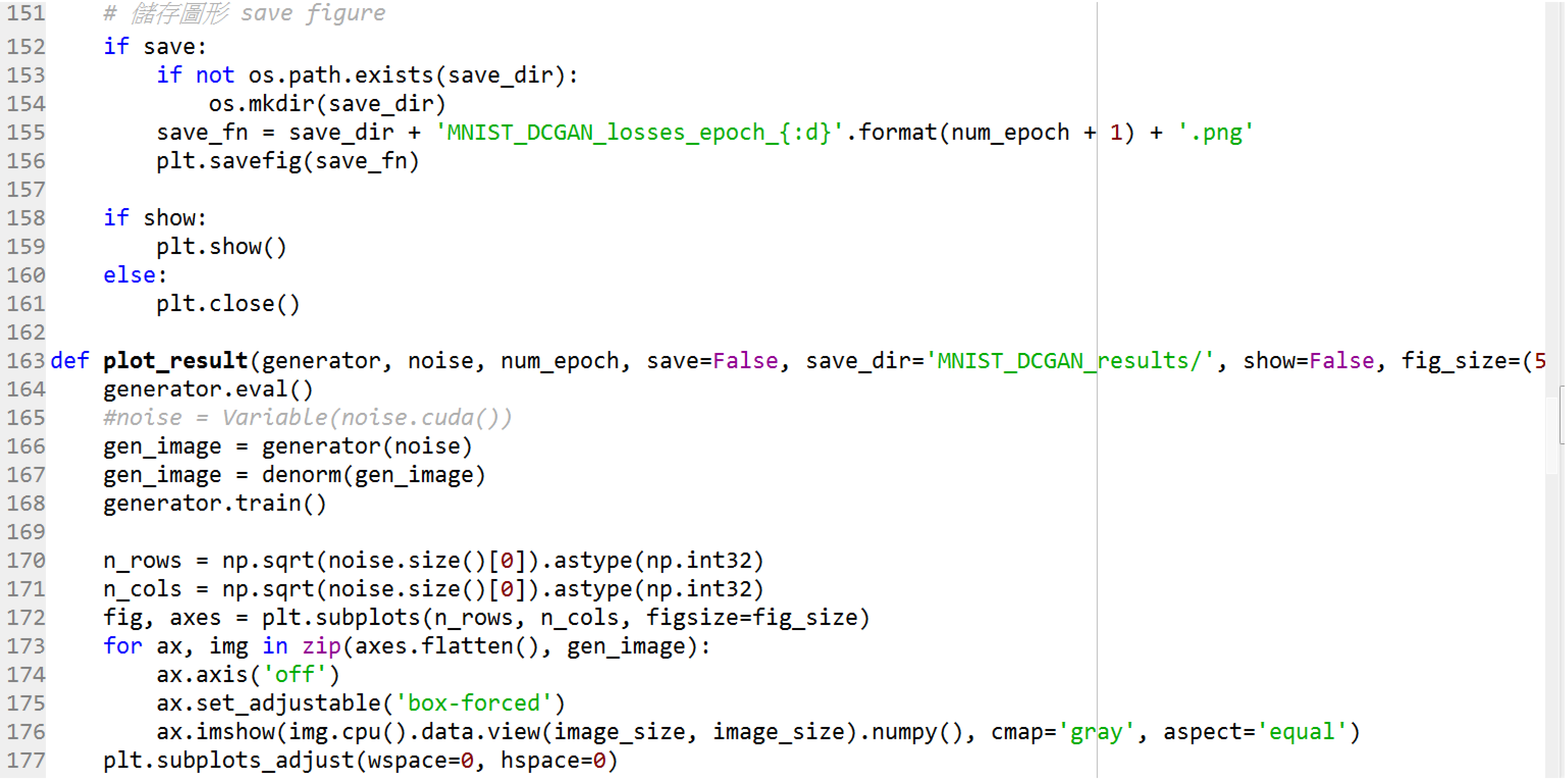
程式碼解說 6



```
# 卷積層 Convolutional layer
out = torch.nn.Conv2d(num_filters[i], output_dim, kernel_size=4, stride=1, padding=0)
self.output_layer.add_module('out', out)
# 初始化 Initializer
torch.nn.init.normal(out.weight, mean=0.0, std=0.02)
torch.nn.init.constant(out.bias, 0.0)
# 激勵函數 Activation
self.output_layer.add_module('act', torch.nn.Sigmoid())

forward(self, x):
h = self.hidden_layer(x)
out = self.output_layer(h)
return out


差 Plot losses
_loss(d_losses, g_losses, num_epoch, save=False, save_dir='MNIST_DCGAN_results/', show=False):
 ax = plt.subplots()
et_xlim(0, num_epochs)
et_ylim(0, max(np.max(g_losses), np.max(d_losses))*1.1)
xlabel('Epoch {0}'.format(num_epoch + 1))
ylabel('Loss values')
plot(d_losses, label='Discriminator')
plot(g_losses, label='Generator')
legend()
```

09 GAN 實作 MNIST

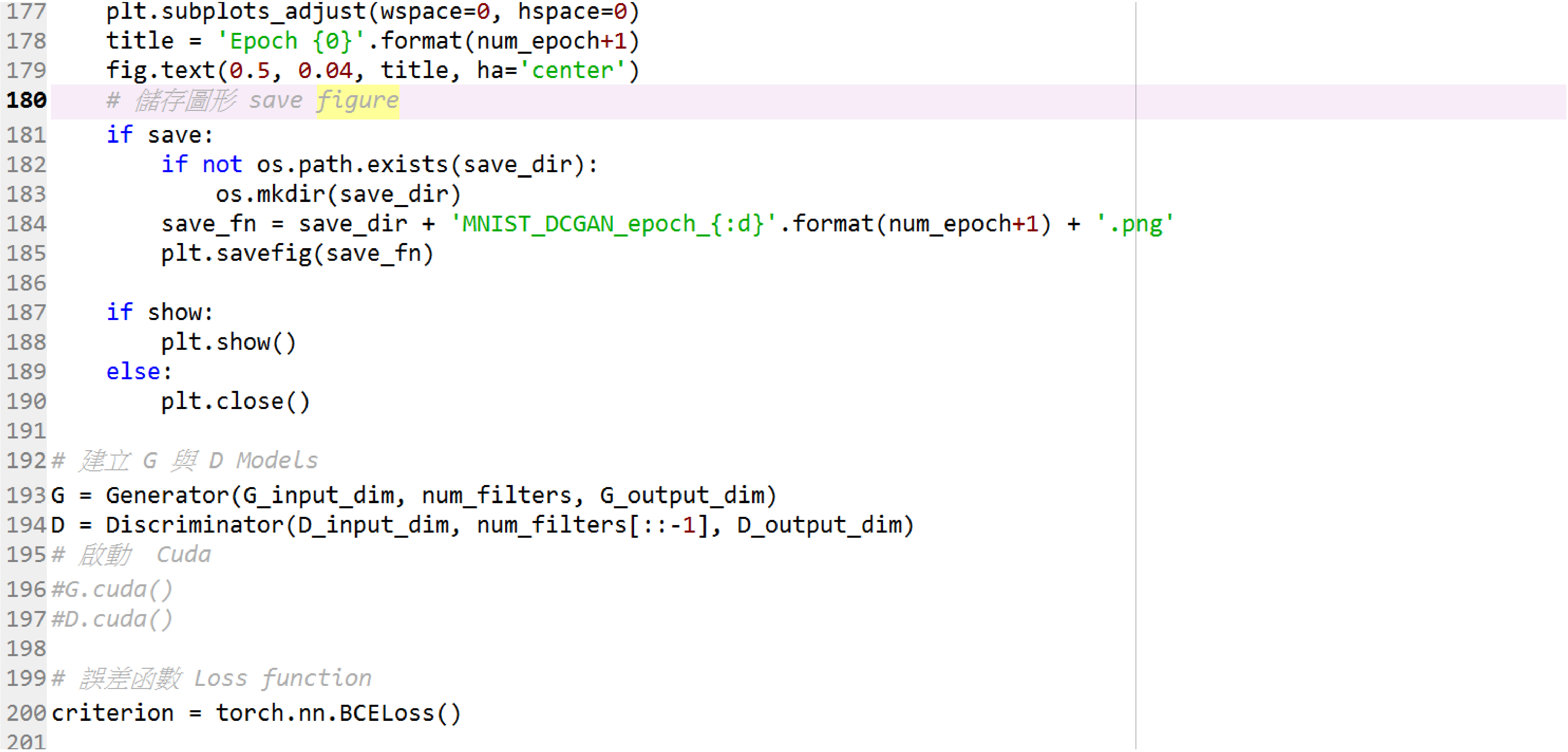
程式碼解說 7



```
    # 儲存圖形 save figure
    if save:
        if not os.path.exists(save_dir):
            os.mkdir(save_dir)
        save_fn = save_dir + 'MNIST_DCGAN_losses_epoch_{:d}'.format(num_epoch + 1) + '.png'
        plt.savefig(save_fn)

    if show:
        plt.show()
    else:
        plt.close()

def plot_result(generator, noise, num_epoch, save=False, save_dir='MNIST_DCGAN_results/', show=False, fig_size=(5
    generator.eval()
    #noise = Variable(noise.cuda())
    gen_image = generator(noise)
    gen_image = denorm(gen_image)
    generator.train()

    n_rows = np.sqrt(noise.size()[0]).astype(np.int32)
    n_cols = np.sqrt(noise.size()[0]).astype(np.int32)
    fig, axes = plt.subplots(n_rows, n_cols, figsize=fig_size)
    for ax, img in zip(axes.flatten(), gen_image):
        ax.axis('off')
        ax.set_adjustable('box-forced')
        ax.imshow(img.cpu().data.view(image_size, image_size).numpy(), cmap='gray', aspect='equal')
    plt.subplots_adjust(wspace=0, hspace=0)
```

09 GAN 實作 MNIST

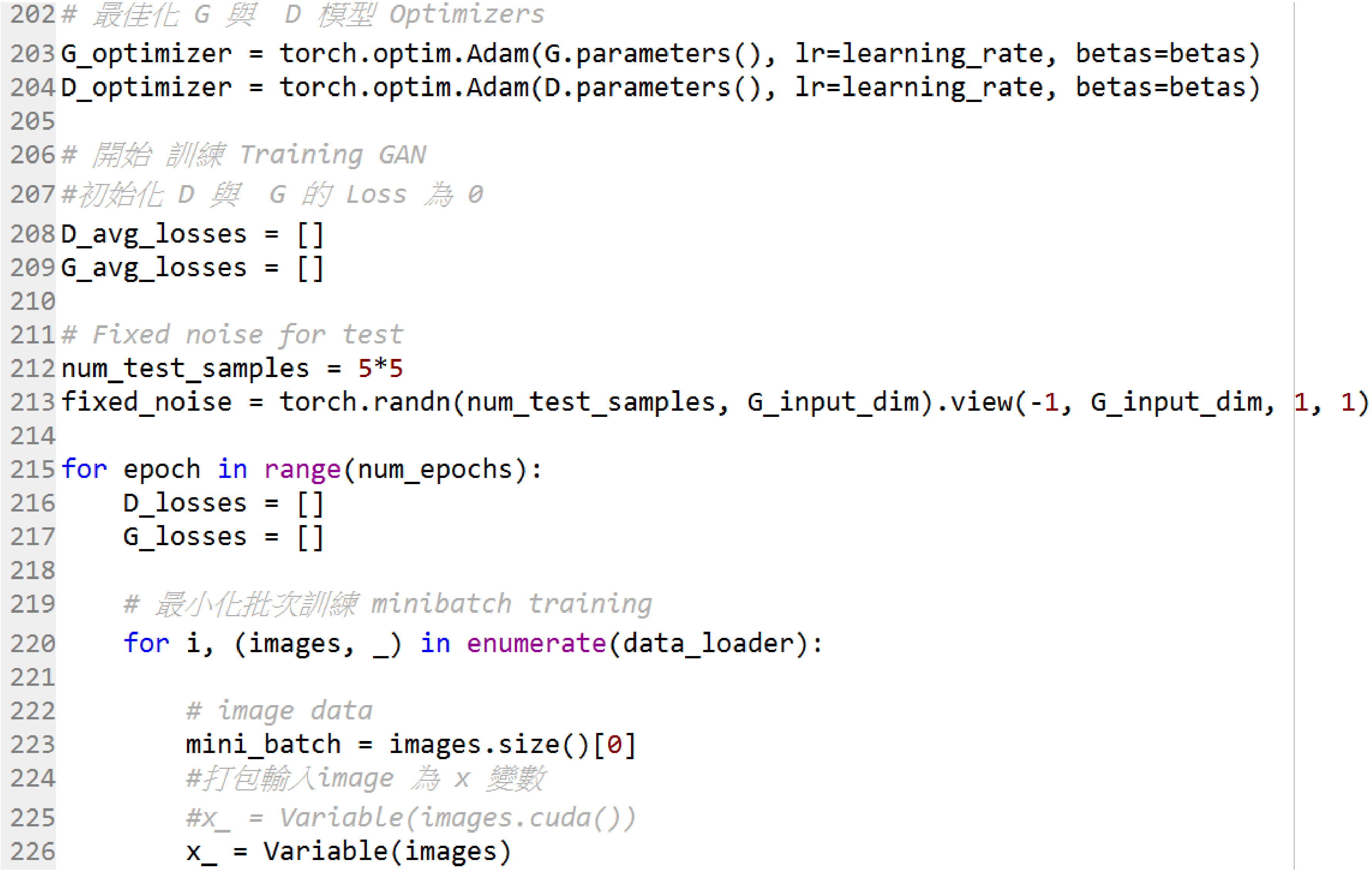
程式碼解說8



```
    plt.subplots_adjust(wspace=0, hspace=0)
    title = 'Epoch {0}'.format(num_epoch+1)
    fig.text(0.5, 0.04, title, ha='center')
    # 儲存圖形 save figure
    if save:
        if not os.path.exists(save_dir):
            os.mkdir(save_dir)
        save_fn = save_dir + 'MNIST_DCGAN_epoch_{:d}'.format(num_epoch+1) + '.png'
        plt.savefig(save_fn)

    if show:
        plt.show()
    else:
        plt.close()

# 建立 G 與 D Models
G = Generator(G_input_dim, num_filters, G_output_dim)
D = Discriminator(D_input_dim, num_filters[::-1], D_output_dim)
# 啟動  Cuda
#G.cuda()
#D.cuda()

# 誤差函數 Loss function
criterion = torch.nn.BCELoss()

```

09 GAN 實作 MNIST

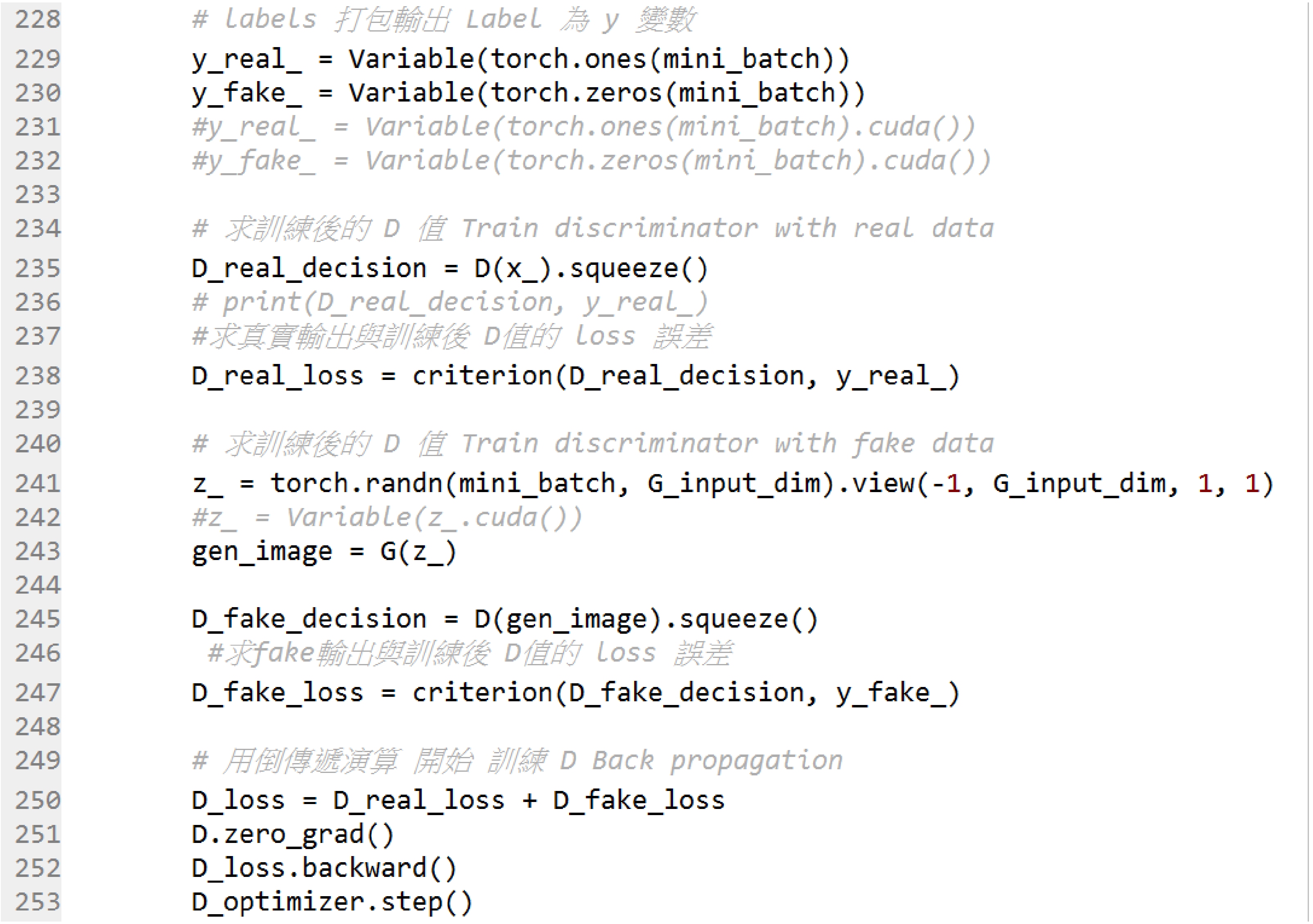
程式碼解說9



```python
# 最佳化 G 與  D 模型 Optimizers
G_optimizer = torch.optim.Adam(G.parameters(), lr=learning_rate, betas=betas)
D_optimizer = torch.optim.Adam(D.parameters(), lr=learning_rate, betas=betas)

# 開始 訓練 Training GAN
#初始化 D 與  G 的 Loss 為 0
D_avg_losses = []
G_avg_losses = []

# Fixed noise for test
num_test_samples = 5*5
fixed_noise = torch.randn(num_test_samples, G_input_dim).view(-1, G_input_dim, 1, 1)

for epoch in range(num_epochs):
    D_losses = []
    G_losses = []

    # 最小化批次訓練 minibatch training
    for i, (images, _) in enumerate(data_loader):

        # image data
        mini_batch = images.size()[0]
        #打包輸入image 為 x 變數
        #x_ = Variable(images.cuda())
        x_ = Variable(images)
```

09 GAN 實作 MNIST

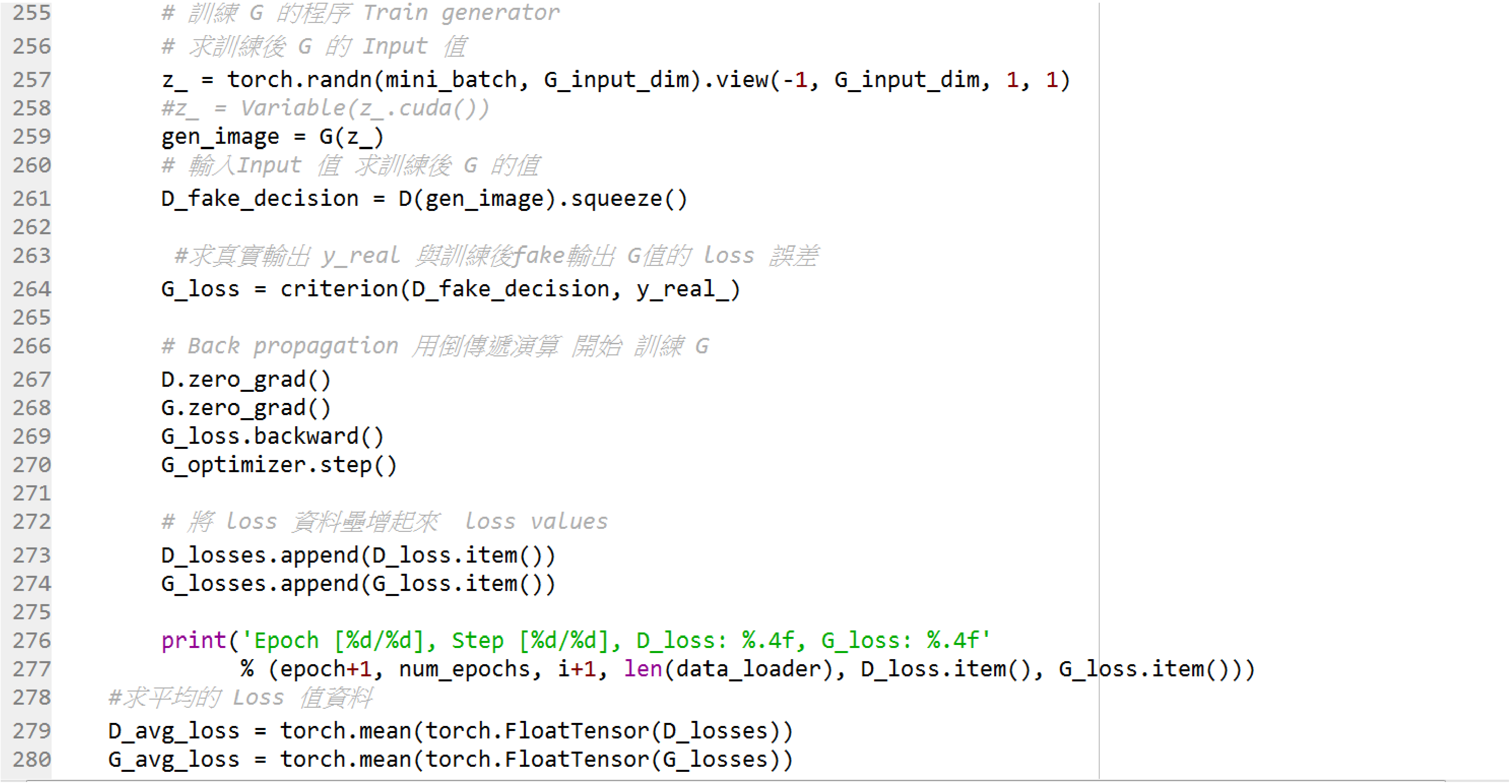
程式碼解說10



```
        # labels 打包輸出 Label 為 y 變數
        y_real_ = Variable(torch.ones(mini_batch))
        y_fake_ = Variable(torch.zeros(mini_batch))
        #y_real_ = Variable(torch.ones(mini_batch).cuda())
        #y_fake_ = Variable(torch.zeros(mini_batch).cuda())

        # 求訓練後的 D 值 Train discriminator with real data
        D_real_decision = D(x_).squeeze()
        # print(D_real_decision, y_real_)
        #求真實輸出與訓練後 D值的 loss 誤差
        D_real_loss = criterion(D_real_decision, y_real_)

        # 求訓練後的 D 值 Train discriminator with fake data
        z_ = torch.randn(mini_batch, G_input_dim).view(-1, G_input_dim, 1, 1)
        #z_ = Variable(z_.cuda())
        gen_image = G(z_)

        D_fake_decision = D(gen_image).squeeze()
         #求fake輸出與訓練後 D值的 loss 誤差
        D_fake_loss = criterion(D_fake_decision, y_fake_)

        # 用倒傳遞演算 開始 訓練 D Back propagation
        D_loss = D_real_loss + D_fake_loss
        D.zero_grad()
        D_loss.backward()
        D_optimizer.step()
```

09 GAN 實作 MNIST

程式碼解說11

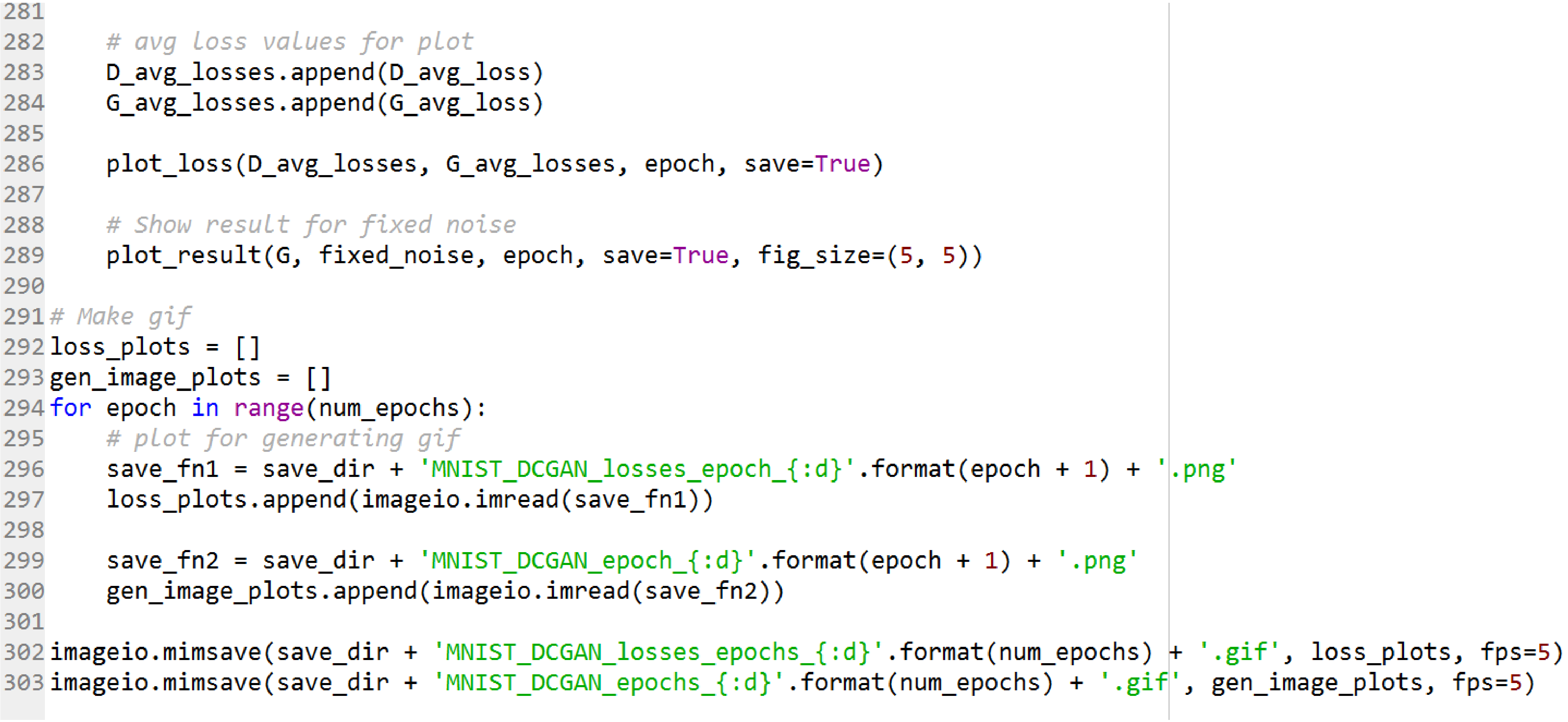


```
        # 訓練 G 的程序 Train generator
        # 求訓練後 G 的 Input 值
        z_ = torch.randn(mini_batch, G_input_dim).view(-1, G_input_dim, 1, 1)
        #z_ = Variable(z_.cuda())
        gen_image = G(z_)
        # 輸入Input 值 求訓練後 G 的值
        D_fake_decision = D(gen_image).squeeze()

         #求真實輸出 y_real 與訓練後fake輸出 G值的 loss 誤差
        G_loss = criterion(D_fake_decision, y_real_)

        # Back propagation 用倒傳遞演算 開始 訓練 G
        D.zero_grad()
        G.zero_grad()
        G_loss.backward()
        G_optimizer.step()

        # 將 loss 資料壘增起來  loss values
        D_losses.append(D_loss.item())
        G_losses.append(G_loss.item())

        print('Epoch [%d/%d], Step [%d/%d], D_loss: %.4f, G_loss: %.4f'
              % (epoch+1, num_epochs, i+1, len(data_loader), D_loss.item(), G_loss.item()))
    #求平均的 Loss 值資料
    D_avg_loss = torch.mean(torch.FloatTensor(D_losses))
    G_avg_loss = torch.mean(torch.FloatTensor(G_losses))
```

09

GAN 實作 MNIST

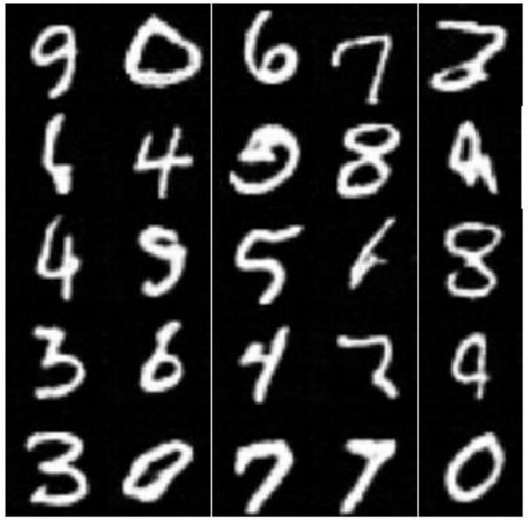
程式碼解說12



```
    # avg loss values for plot
    D_avg_losses.append(D_avg_loss)
    G_avg_losses.append(G_avg_loss)

    plot_loss(D_avg_losses, G_avg_losses, epoch, save=True)

    # Show result for fixed noise
    plot_result(G, fixed_noise, epoch, save=True, fig_size=(5, 5))

# Make gif
loss_plots = []
gen_image_plots = []
for epoch in range(num_epochs):
    # plot for generating gif
    save_fn1 = save_dir + 'MNIST_DCGAN_losses_epoch_{:d}'.format(epoch + 1) + '.png'
    loss_plots.append(imageio.imread(save_fn1))

    save_fn2 = save_dir + 'MNIST_DCGAN_epoch_{:d}'.format(epoch + 1) + '.png'
    gen_image_plots.append(imageio.imread(save_fn2))

imageio.mimsave(save_dir + 'MNIST_DCGAN_losses_epochs_{:d}'.format(num_epochs) + '.gif', loss_plots, fps=5)
imageio.mimsave(save_dir + 'MNIST_DCGAN_epochs_{:d}'.format(num_epochs) + '.gif', gen_image_plots, fps=5)
```

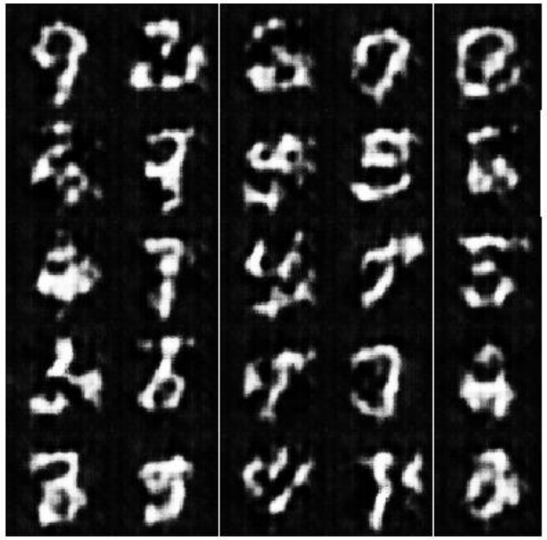
06 GAN 實作 MNIST

程式輸出



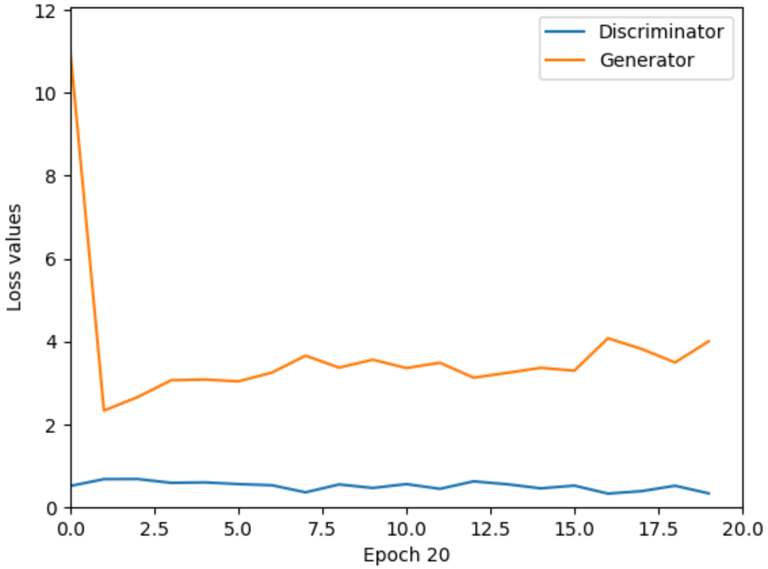
Epoch 20

訓練用資料



Epoch 1

產生的結果



20次效能分析響應圖

thank you!

謝謝聆聽

THANK YOU FOR YOUR ATTENTION