DLCV 2022 Fall

郭思言 B08508002 醫工四

HW2

Part 1

1. Print the architecture of the method a and b:

   (1) Method a: DC-GAN

```
Discriminator
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           3,136
            Conv2d-2          [-1, 128, 16, 16]         131,072
       BatchNorm2d-3          [-1, 128, 16, 16]             256
            Conv2d-4            [-1, 256, 8, 8]         524,288
       BatchNorm2d-5            [-1, 256, 8, 8]             512
            Conv2d-6            [-1, 512, 4, 4]       2,097,152
       BatchNorm2d-7            [-1, 512, 4, 4]           1,024
            Conv2d-8              [-1, 1, 1, 1]           8,193
================================================================
Total params: 2,765,633
Trainable params: 2,765,633
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 1.38
Params size (MB): 10.55
Estimated Total Size (MB): 11.97
----------------------------------------------------------------
```

```
Generator
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 512, 4, 4]         819,200
       BatchNorm2d-2            [-1, 512, 4, 4]           1,024
   ConvTranspose2d-3            [-1, 256, 8, 8]       2,097,152
       BatchNorm2d-4            [-1, 256, 8, 8]             512
   ConvTranspose2d-5          [-1, 128, 16, 16]         524,288
       BatchNorm2d-6          [-1, 128, 16, 16]             256
   ConvTranspose2d-7           [-1, 64, 32, 32]         131,072
       BatchNorm2d-8           [-1, 64, 32, 32]             128
   ConvTranspose2d-9            [-1, 3, 64, 64]           3,075
================================================================
Total params: 3,576,707
Trainable params: 3,576,707
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 1.97
Params size (MB): 13.64
Estimated Total Size (MB): 15.61
----------------------------------------------------------------
```

   In method a, I use Leaky ReLU as activation function for the discriminator and ReLU for generator.

   (2) Method b: DC-GAN with modification (architecture, loss, and training session)

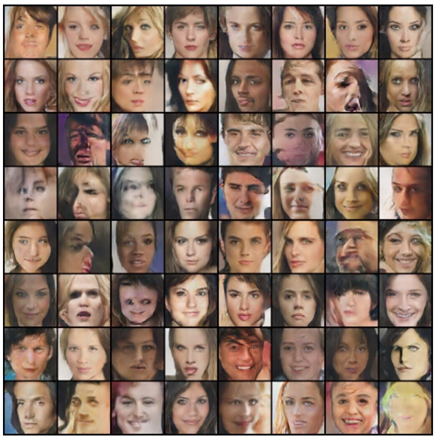
```
Discriminator
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           3,136
           Dropout-2           [-1, 64, 32, 32]               0
            Conv2d-3          [-1, 128, 16, 16]         131,072
       BatchNorm2d-4          [-1, 128, 16, 16]             256
           Dropout-5          [-1, 128, 16, 16]               0
            Conv2d-6            [-1, 256, 8, 8]         524,288
       BatchNorm2d-7            [-1, 256, 8, 8]             512
           Dropout-8            [-1, 256, 8, 8]               0
            Conv2d-9            [-1, 512, 4, 4]       2,097,152
      BatchNorm2d-10            [-1, 512, 4, 4]           1,024
          Dropout-11            [-1, 512, 4, 4]               0
           Conv2d-12              [-1, 1, 1, 1]           8,193
================================================================
Total params: 2,765,633
Trainable params: 2,765,633
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 2.31
Params size (MB): 10.55
Estimated Total Size (MB): 12.91
----------------------------------------------------------------
```

```
Generator
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 512, 4, 4]         819,200
       BatchNorm2d-2            [-1, 512, 4, 4]           1,024
   ConvTranspose2d-3            [-1, 256, 8, 8]       2,097,152
       BatchNorm2d-4            [-1, 256, 8, 8]             512
   ConvTranspose2d-5          [-1, 128, 16, 16]         524,288
       BatchNorm2d-6          [-1, 128, 16, 16]             256
   ConvTranspose2d-7           [-1, 64, 32, 32]         131,072
       BatchNorm2d-8           [-1, 64, 32, 32]             128
   ConvTranspose2d-9            [-1, 3, 64, 64]           3,075
================================================================
Total params: 3,576,707
Trainable params: 3,576,707
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 1.97
Params size (MB): 13.64
Estimated Total Size (MB): 15.61
----------------------------------------------------------------
```

   In method b, I use Leaky ReLU as activation function for both the discriminator and the generator.

2. Please show generated images of both method A and B then discuss the difference between method A and B.

   (1) Method a and mothod b:

(2) Discussion:

左邊的為沒有經過調整的一般的 DCGAN, 可以看出其實有很多是不太正常的人臉，雖然訓練了接近 300 個 epoch，discriminator 與 generator 並沒有達到真正 adversarial training 的目的。

而右邊的為經過 modify 後的 DCGAN，可以看出雖然仍有許多照片我們一眼就可以看出是生成的，但是正常的人臉比例更多，而且正常的人臉相較左邊的更有光澤的感覺（左邊的較模糊)。

(3) Please discuss what you’ve observed and learned from implementing GAN:

原先 train method b 時我使用的是 wgan 以及嘗試使用 wgan-gp，結果不知道是什麼問題感覺不像文獻上寫的較 dcgan 有較好的成效，感覺雖然可以生出蠻高解析度的圖片，但是 fid score 以及 face recognition 數值都和一般 dcgan 差不多。另外，也因為參數的部分大多固定沒有太多地方可多做調整，後來決定回歸 dcgan 來做 modify。我幾乎使用了 gan train tips 的所有方法：包括增加了使用 soft and noisy label, random flip labels, 使用 leaky relu, dropout, 等等才成功到達 baseline 的標準，非常的難 train。此外難 train 的部分在於 train 不好也不知道問題出在哪個部分。

Part 2:

Reference: https://github.com/TeaPearce/Conditional_Diffusion_MNIST

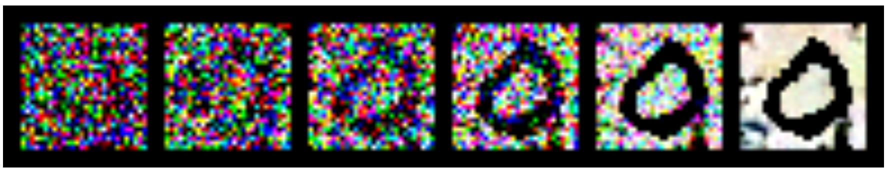
1. Print model architecture:

```
DDPM(
  (nn_model): ContextUnet(
    (init_conv): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(3, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate=none)
      )
      (conv2): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate=none)
      )
    )
    (down1): UnetDown(
      (model): Sequential(
        (0): ResidualConvBlock(
          (conv1): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
          (conv2): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
        )
        (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (down2): UnetDown(
      (model): Sequential(
        (0): ResidualConvBlock(
          (conv1): Sequential(
            (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
          (conv2): Sequential(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
        )
        (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (to_vec): Sequential(
      (0): AvgPool2d(kernel_size=7, stride=7, padding=0)
      (1): GELU(approximate=none)
    )
    (timeembed1): EmbedFC(
      (model): Sequential(
        (0): Linear(in_features=1, out_features=512, bias=True)
        (1): GELU(approximate=none)
        (2): Linear(in_features=512, out_features=512, bias=True)
      )
    )
    (timeembed2): EmbedFC(
      (model): Sequential(
        (0): Linear(in_features=1, out_features=256, bias=True)
        (1): GELU(approximate=none)
        (2): Linear(in_features=256, out_features=256, bias=True)
      )
    )
    (contextembed1): EmbedFC(
      (model): Sequential(
        (0): Linear(in_features=10, out_features=512, bias=True)
        (1): GELU(approximate=none)
        (2): Linear(in_features=512, out_features=512, bias=True)
      )
    )
    (contextembed2): EmbedFC(
      (model): Sequential(
        (0): Linear(in_features=10, out_features=256, bias=True)
        (1): GELU(approximate=none)
        (2): Linear(in_features=256, out_features=256, bias=True)
      )
    )
    (up0): Sequential(
      (0): ConvTranspose2d(512, 512, kernel_size=(7, 7), stride=(7, 7))
      (1): GroupNorm(8, 512, eps=1e-05, affine=True)
      (2): ReLU()
    )
    (up1): UnetUp(
      (model): Sequential(
        (0): ConvTranspose2d(1024, 256, kernel_size=(2, 2), stride=(2, 2))
        (1): ResidualConvBlock(
          (conv1): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
          (conv2): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
        )
        (2): ResidualConvBlock(
          (conv1): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
          (conv2): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate=none)
          )
        )
      )
    )
    (up2): UnetUp(
      (model): Sequential(
        (0): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
        (1): ResidualConvBlock(
          (conv1): Sequential(
```

```
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate=none)
        )
        (conv2): Sequential(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate=none)
        )
      )
      (2): ResidualConvBlock(
        (conv1): Sequential(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate=none)
        )
        (conv2): Sequential(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate=none)
        )
      )
    )
  )
  (out): Sequential(
    (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): GroupNorm(8, 256, eps=1e-05, affine=True)
    (2): ReLU()
    (3): Conv2d(256, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
(loss_mse): MSELoss()
```

2. Show 10 generated images each for digit 0-9:

3. Visualize the first 0 in reverse process:



/ T: 0 / T: 80 / T: 160 / T: 240 / T: 320 / T: 400

4. Dicussion:

Diffusion model training 可以相較其他 generative model 的方式收斂的快，以我這次為例我只利用 28 個 epoch 及達到 90 percent 的 accuracy。而整個 training 過程也只需要僅 30 至 60 分鐘。而我認為這個 diffusion model 反而

是在 sampling 的過程花費較多時間，要生出一張新的照片必須要經過所有的 timestep，去除多次的 noise 後才能有最終的照片。

Part 3:

Reference: https://github.com/fungtion/DANN

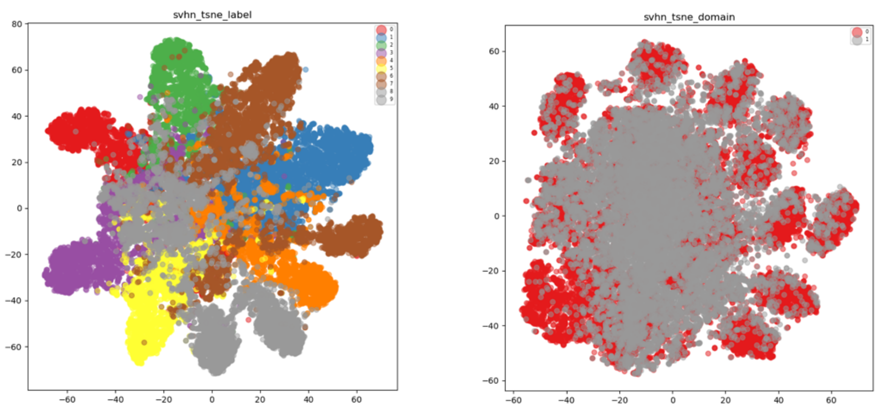
1. Create and fill the table:

	MNIST-M → SVHN	MNIST-M → USPS
TRAINED ON SOURCE	0.292189	0.745296
ADAPTATION (DANN)	0.470825	0.880376
TRAINED ON TARGET	0.919053	0.989247

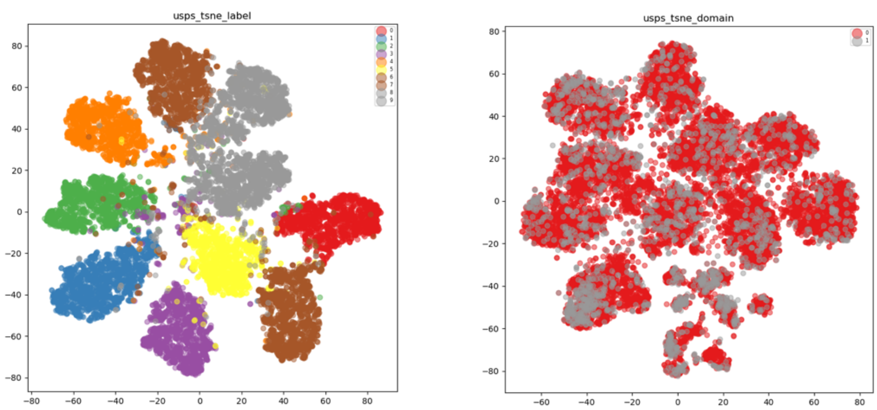
```
Accuracy of the svhn dataset: 0.292189
Accuracy of the svhn dataset: 0.470825
Accuracy of the svhn dataset: 0.919053
Accuracy of the usps dataset: 0.745296
Accuracy of the usps dataset: 0.880376
Accuracy of the usps dataset: 0.989247
```

2. Visualize latent space:

(1) Mnist-m -> Svhn:



(2) Mnist-m -> Usps:



3. Describe implementation details and discussion:

   Dann 的 training 包含 source 以及 target 兩筆不同的 dataset，因此這兩筆 dataset 的相似或是複雜程度會造成最終訓練成效好壞非常大的影響。可以由 mnist-m->svhn 和 mnist-m->usps 可看出：usps 為較簡單的黑白數字資料，由相較複雜的 mnist-m 為 source 可以輕易的達到不錯的 accuracy; 相對的 schn 為自然界中包的彩色數字資料，因此由同樣的 mnist-m 為 source 就相對的比較沒有那麼高的 accuracy，但是仍相較只在 source 上面 train 有更好的結果。