HW2

Done	☐
Due date	@October 31, 2022
Subject	DLCV

Part 1

1. Please print the model architecture of method A and B.

Model A:

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Model B:

```
MyGenerator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
MyDiscriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
```

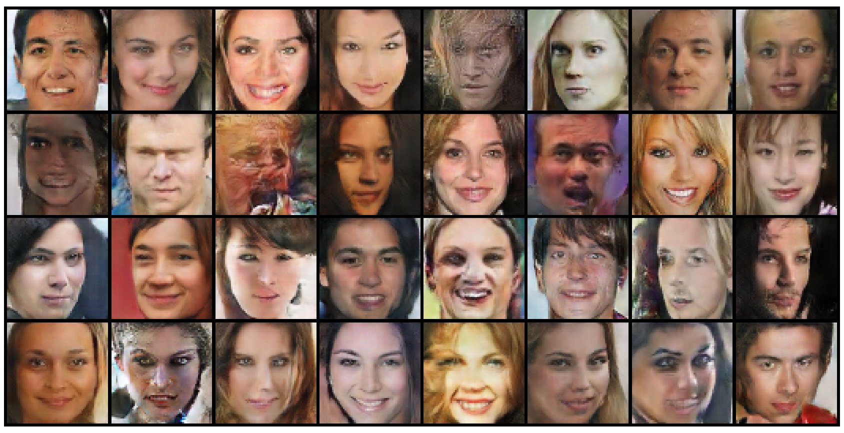
```
  )
)
```

Please show the first 32 generated images of both method A and B then discuss the difference between method A and B.

產生的圖片如下，在 B model 中我嘗試 **GAN tranning tips** 中提到對 Discriminator 訓練的技巧，就是讓 Label 不全然為 1 和 0 ，而是加入些許雜訊。像是 true 的 lable 就可能為 1.1, 0.9, 1.23… ，而 False 也相同。

最後這個技巧讓我的 DCGAN 成功過 strong baseline

Model A



Model B

Please discuss what you’ve observed and learned from implementing GAN.

在這次作業中可以完全體會 **No pain, No GAN** 這句話，GAN 訓練的難度真的比一般的 model 高很多，甚至有些運氣成份，最麻煩的是可能會遇到 mode collapse 的問題，所以最好每隔幾個 epoch 就把 model 存起來備份一次，以免 mode collapse讓前面的訓練白費。

Part 2

1. Please print your model architecture and describe your implementation details.

模型的架構如下，我訓練的參數為 **learning rate 3e-4** 並且使用 **Adam** 為 optimizer 還有 **t**= 400

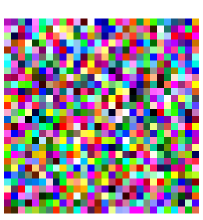
```
DDPM(
  (nn_model): ContextUnet(
    (init_conv): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(3, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate=none)
      )
      (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()
```

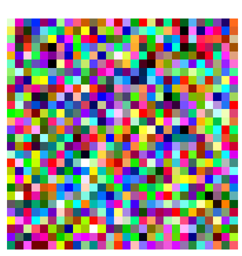
```
        (3): Conv2d(256, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (loss_mse): MSELoss()
  )
```

2. Please show 10 generated images for each digit (0-9)

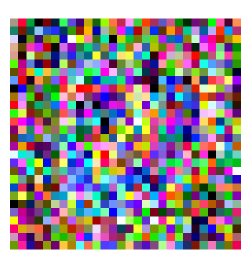
3. Visualize total six images in the reverse process of the first “0”



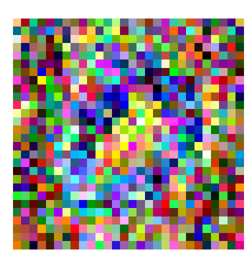
t = 0



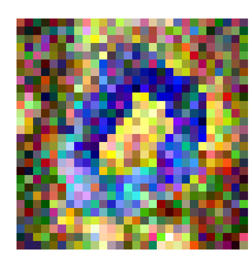
t = 80



t = 160



t = 240



t = 320

t = 400

4. Please discuss what you’ve observed and learned from implementing conditional diffusion model.

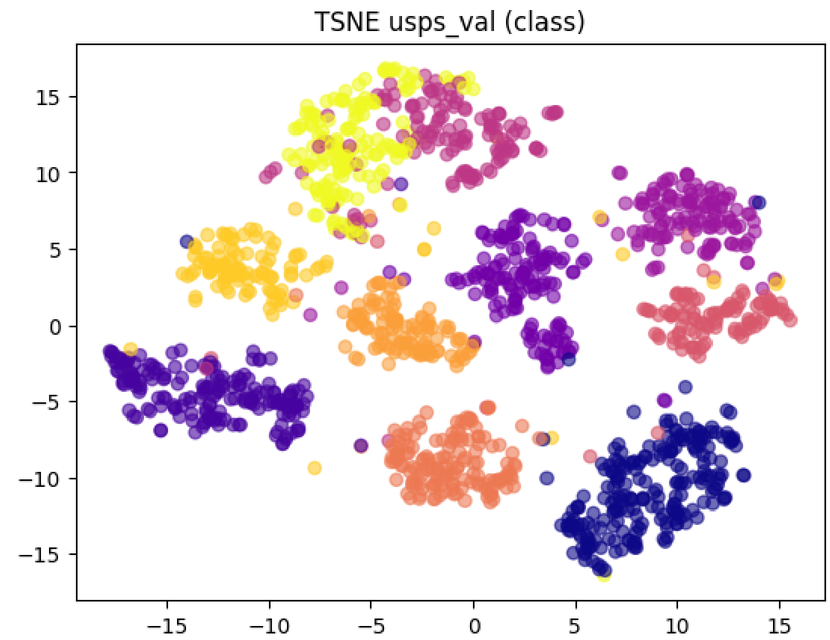
寫過 part 1 再寫 part 2 可以了解不同的照片生成方法，一個是從 generator 跟 discriminator 之間來回對抗，這個則是透過還原雜訊的方式做訓練，不得不佩服想出這些方法的人，實做才知道看似簡單的步驟背後的數學推導可是很複雜的。

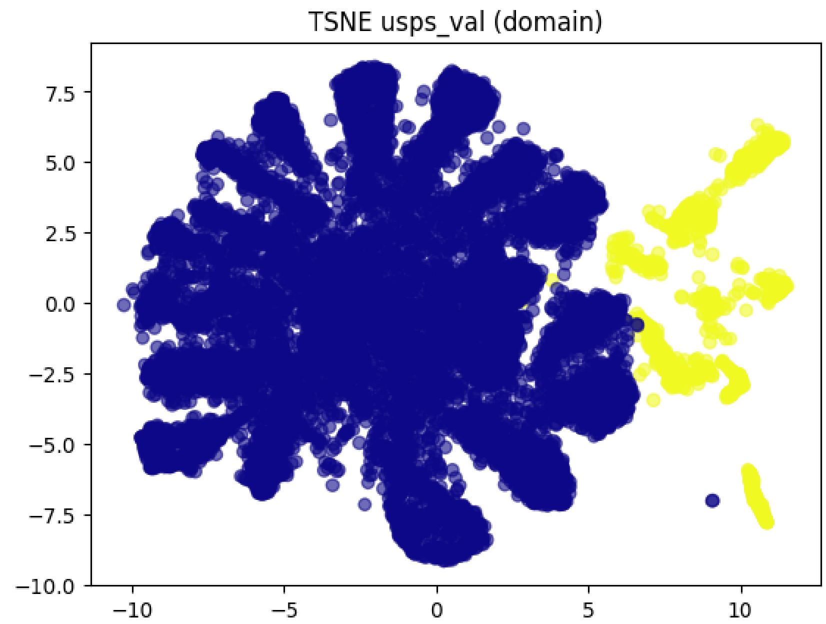
Part 3

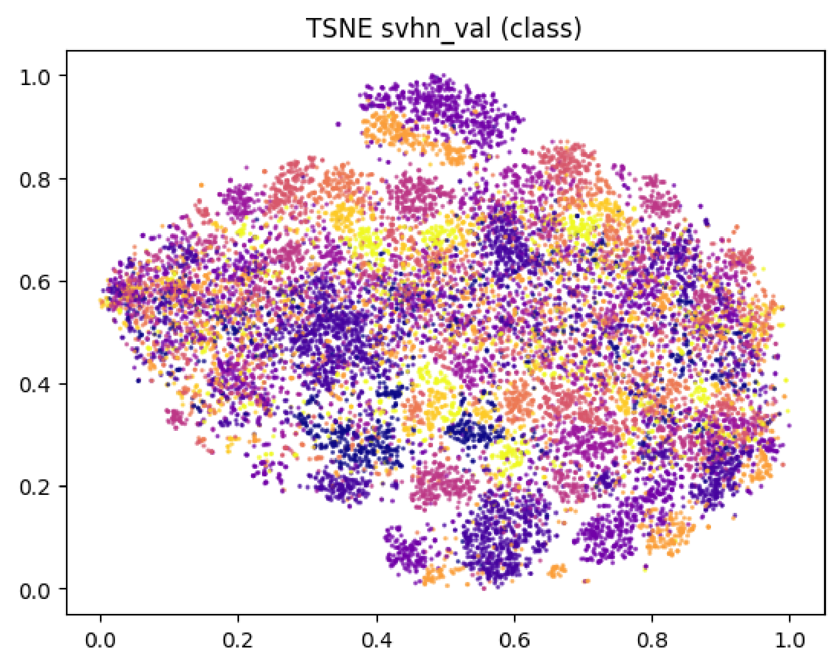
1. Please create and fill the table with the following format

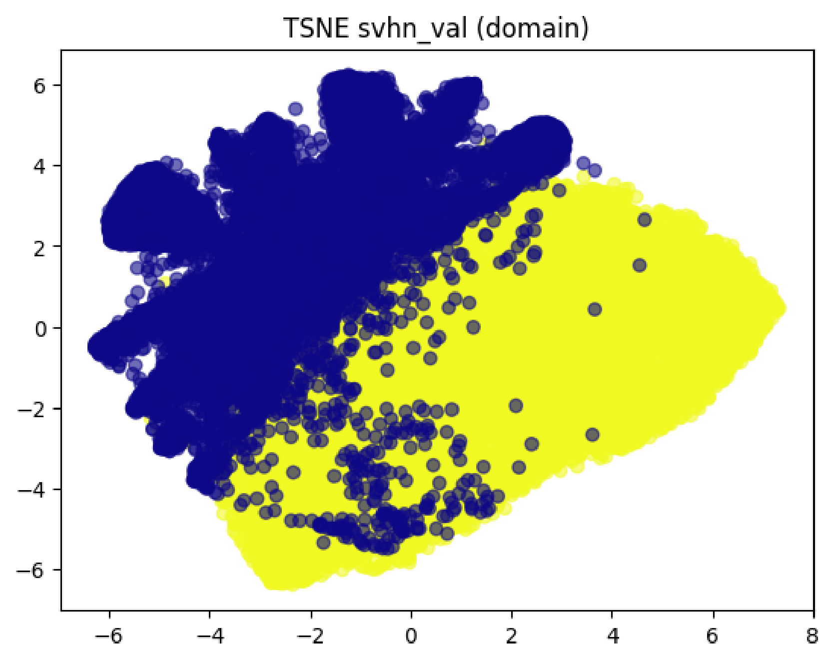
	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	0.19	0.12
Adaptation (DANN)	**0.4881**	**0.7628**
Trained on target	0.91	0.98

2. Please visualize the latent space of DANN by mapping the *validation* images to 2D space with t-SNE.









3. Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.

在這次作業中我們實做了遷移學習，其中運作原理很類似 GAN，都是想辦法混淆 discriminator，但實做後才發現這方法沒有想像中容易，要得到好的 val score 除了調整 lr batch_size 等常見參數，還有 discriminator loss 佔總 loss 的比例要微調。從 tsne 圖中可以發現 features 也沒有到很混雜。所以感覺這個模型還有很大進步空間。