DLCV Homework 3

r09944003 網媒所碩一 陳竣宇

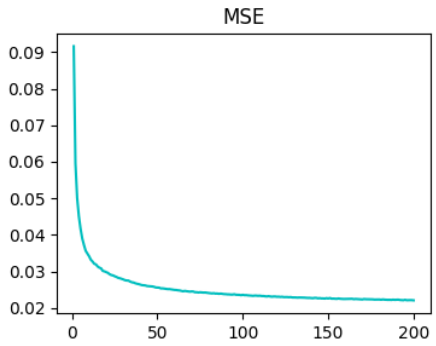
Problem 1: VAE

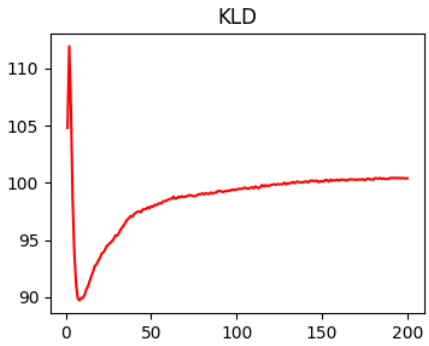
1. ◦ Network architecture

```
VAE(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (1): Sequential(
      (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (fc_mu): Linear(in_features=2048, out_features=1024, bias=True)
  (fc_var): Linear(in_features=2048, out_features=1024, bias=True)
  (decoder_input): Linear(in_features=1024, out_features=2048, bias=True)
  (decoder): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (1): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (2): Sequential(
      (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (3): Sequential(
      (0): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (final_layer): Sequential(
    (0): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): Conv2d(32, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): Tanh()
  )
)
```

- Implementation details
  - Training epoch: 200
  - Batch size: 128
  - Optimizer: AdamW (learning rate = 0.001)
  - No data augmentation
  - Data normalization to (-1, 1)
  - Lambda_KL: 0.0001

2. **Learning curve**



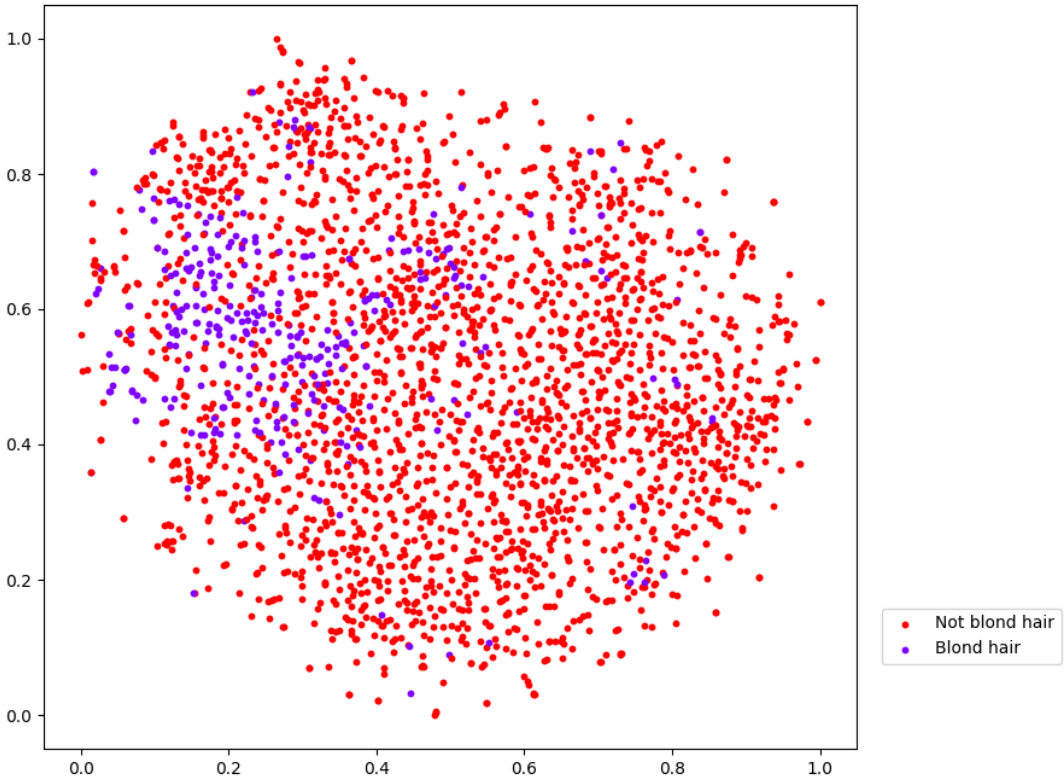


3.

Testing Image										
Recon. Image										
MSE	0.035750	0.036140	0.011202	0.012298	0.043287	0.016112	0.019895	0.018789	0.018627	0.027034

4.

5. For this problem, I choose **Blond_Hair** as the attribute.



6. Discussion
    - 在對testing image做reconstruction時，我發現原本的input丟入訓練好的VAE model後decoder所output的圖會和原圖有些微的差距，甚至會感覺變了一個人。我認為是因為在原先的Autoencoder model中因為要minimize reconstuction error因此會讓重建後的圖去靠近原圖，而VAE除了minimize reconstuction error之外還會讓 $p(Z|X)$ 去接近normal distribution，防止noise為0的同時保證了model的生成能力。
    - 從learning curve可以觀察到，在一開始decoder未經訓練時model會去降低noise來讓reconstruction loss快速下降，因此KLD loss會先呈現一個上升的趨勢，當decoder快速收斂後model就會讓noise增加來提高generation能力，因此KLD loss快速下降。
    - 從tSNE的結果來分析，我們可以看出不是金髮的圖像基本呈現平均分散，而有金髮的圖像可以推論他們在高維空間應該會比較接近，而使得tSNE降維後的結果在去除一些outlier後有聚集的趨勢。

Problem 2: GAN

1.
    - Network architecture

```
Generator(
  (l1): Sequential(
    (0): Linear(in_features=100, out_features=8192, bias=False)
    (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Dropout(p=0.5, inplace=False)
  )
  (l2_5): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout(p=0.5, inplace=False)
    )
    (1): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout(p=0.5, inplace=False)
    )
    (2): Sequential(
      (0): ConvTranspose2d(128, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout(p=0.5, inplace=False)
    )
    (3): ConvTranspose2d(64, 3, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1))
    (4): Tanh()
  )
)
Discriminator(
  (ls): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout(p=0.5, inplace=False)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout(p=0.5, inplace=False)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout(p=0.5, inplace=False)
    )
    (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
  )
)
```

- Implementation details
  - WGAN (weight clipping = 0.01)
  - Training epoch: 200
  - Batch size: 64
  - Optimizer: Both generator and discriminator are **RMSprop** (learning rate = 0.00005)
  - No data augmentation
  - Data normalization to (-1, 1)

2.

3.

- 在這一題我一開始選擇先實作最基本的DCGAN，在前幾個epoch之後其實就能觀察到生成的data大致上已經具有人臉的雛形，然而在過了數10個epoch後生成圖像的品質不見好轉，並且有機會出現mode collapse的情況。
- 後來我在generator和discriminator分別加上Dropout layer來減緩mode collapse，並且透過更改model、optimizer以及相關訓練過程來增加model的生成能力，達到最後的結果。

4.

- 在觀察兩種model生成的影像後，我發現VAE的好處是可以透過encoder和decoder來比較原本圖像和reconstruct圖像的差距，藉此來幫助評估model的好壞。
- 而GAN生成的影像則比VAE來得更清晰一些，我認為這是adversarial network所帶來的益處。

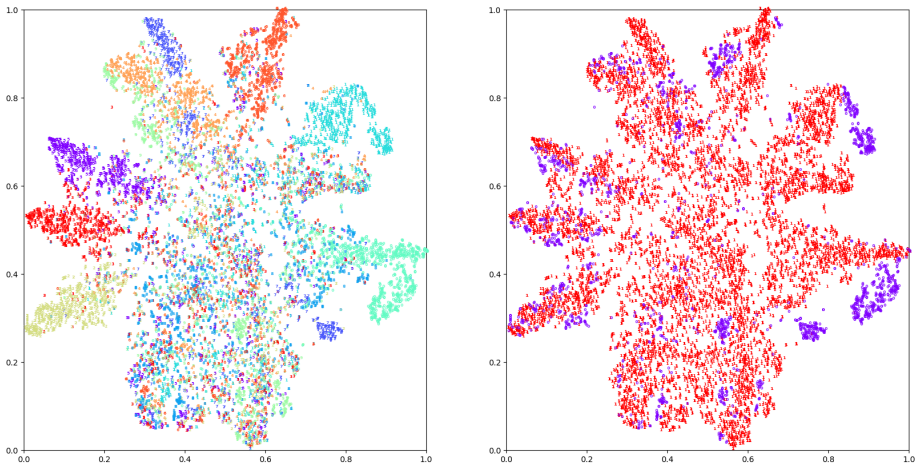
Problem 3: DANN

1. (1, 2, 3)

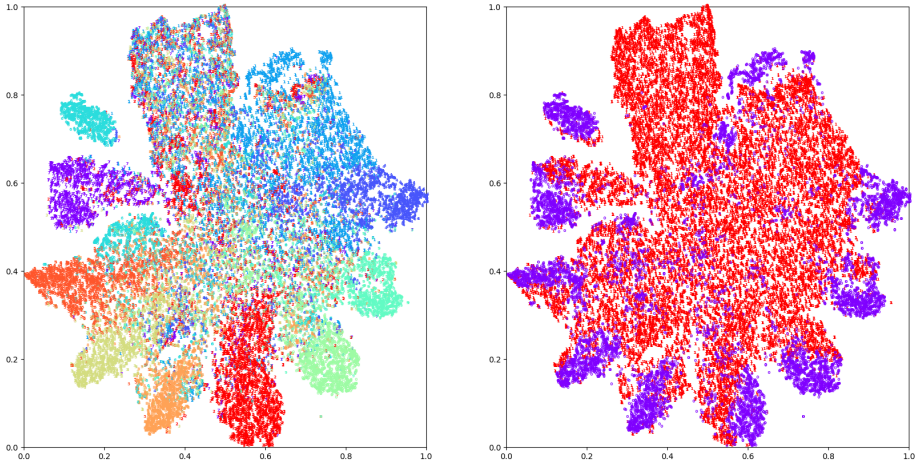
	USPS --> MNIST-M	**MNIST-M --> SVHN**	**SVHN --> USPS**
Trained on source	0.2666	0.2725	0.6297
Adaption (DANN)	0.4484	0.4372	0.6741
Trained on target	0.9725	0.9154	0.9636

2. (4)

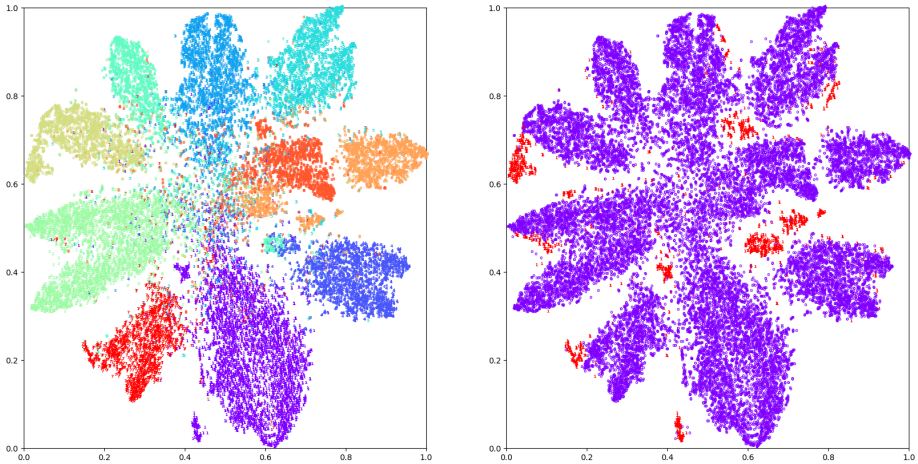
- 每張圖左邊的子圖每個顏色代表不同的digit class，右邊則是不同的domain
- USPS --> MNIST-M



- MNIST-M --> SVHN



- SVHN --> USPS



1. (5)

- Network architecture

```
FeatureExtractor(
  (conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout2d(p=0.5, inplace=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Dropout2d(p=0.5, inplace=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): LeakyReLU(negative_slope=0.2)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Dropout2d(p=0.5, inplace=False)
    (15): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): LeakyReLU(negative_slope=0.2)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Dropout2d(p=0.5, inplace=False)
    (20): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (22): LeakyReLU(negative_slope=0.2)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Dropout2d(p=0.5, inplace=False)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (27): LeakyReLU(negative_slope=0.2)
  )
)
LabelPredictor(
  (layer): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Linear(in_features=512, out_features=512, bias=True)
    (4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2)
    (6): Linear(in_features=512, out_features=10, bias=True)
  )
)
DomainClassifier(
  (layer): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Linear(in_features=512, out_features=512, bias=True)
    (4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2)
    (6): Linear(in_features=512, out_features=512, bias=True)
    (7): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2)
    (9): Linear(in_features=512, out_features=512, bias=True)
    (10): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.2)
    (12): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

- Implementation details
  - Training epoch: 200
  - Batch size: 128
  - Optimizer: Both feature extractor、label predictor and domain classifier are **Adam** (learning rate = 0.0001)
  - Data normalization to (-1, 1)

4. (6) Discussion
    - 在實作DANN之前，直覺上會認為如果直接把target domain data丟到用source domain data去訓練的feature extractor上是沒有辦法讓輸出的feature和source feature有相似的分布的，因為feature extractor根本沒看過這些圖片。而在實作DANN之後才發現可以利用試圖去騙過domain classifer來訓練feature extractor，得到一個adversarial的架構，只能說佩服於前人的巧思。
    - 在原paper中是加上Gradient Reversal Layer並讓三個model一起訓練，而這裡我是透過交換訓練 domain classifier以及feature extractor來實現。

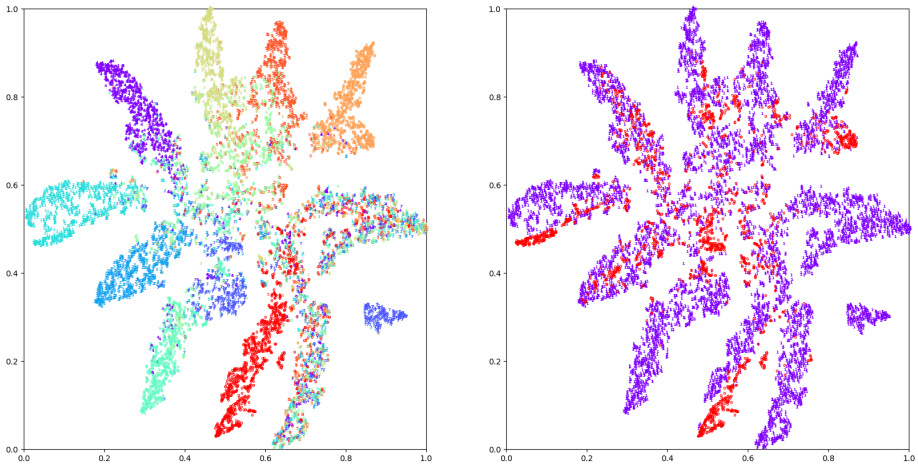
Problem 4: Improved UDA model

1.

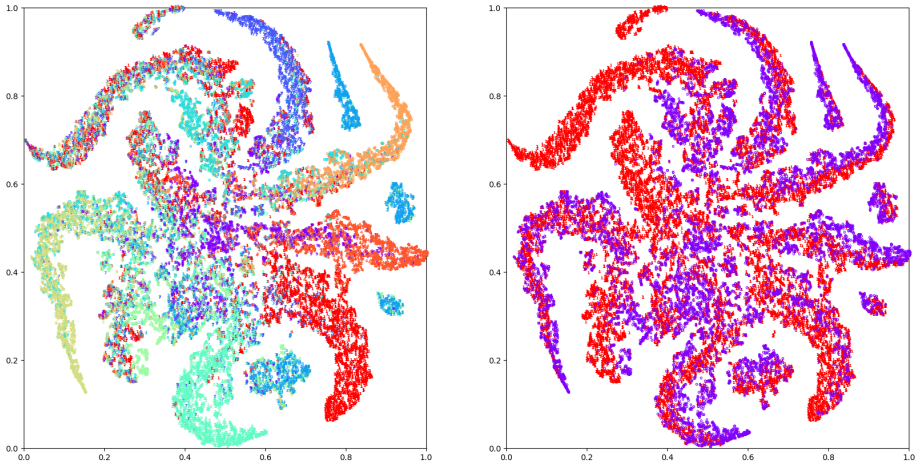
	USPS --> MNIST-M	MNIST-M --> SVHN	SVHN --> USPS
Adaption (Improved)	0.6392	0.4461	0.6870

2.

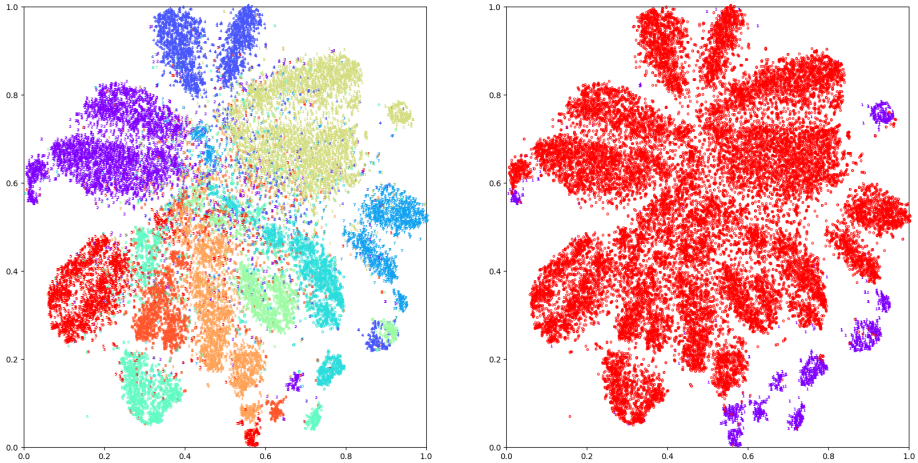
- 每張圖左邊的子圖每個顏色代表不同的digit class，右邊則是不同的domain
- USPS --> MNIST-M



- MNIST-M --> SVHN



- SVHN --> USPS



3.

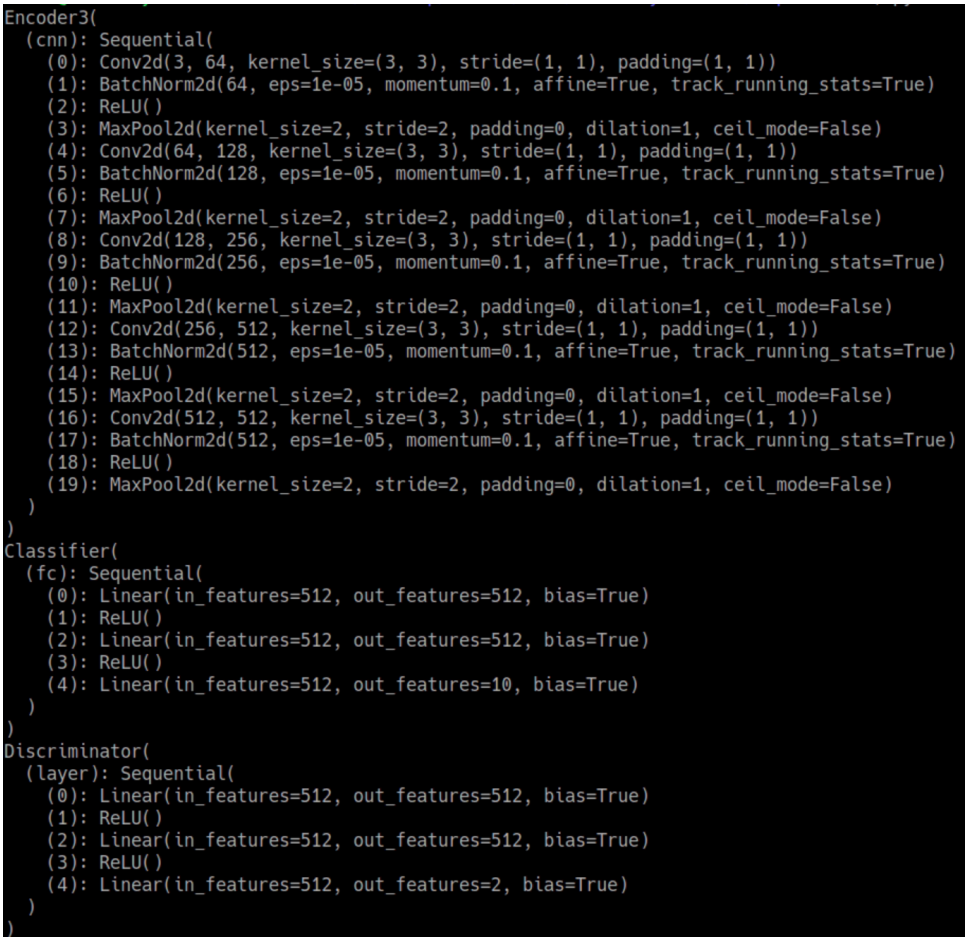
- Network architecture
  - USPS --> MNIST-M

```
Encoder1(
  (cnn): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
    (12): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
  )
)
Classifier(
  (fc): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
Discriminator(
  (layer): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

- MNIST-M --> SVHN

```
Encoder2(
  (cnn): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (20): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (22): ReLU()
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (25): ReLU()
  )
)
Classifier(
  (fc): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
Discriminator(
  (layer): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

- SVHN --> USPS



- Implementation details
  - Common
    - ADDA
    - Training epoch (source domain): 10
    - Training epoch (target domain): 100
    - Optimizer: Adam(target encoder) + Adam(discriminator)
  - USPS --> MNIST-M
    - Batch_size: 256
    - lr=0.0001, 0.0002
    - Data normalization to (-1, 1)
  - MNIST-M --> SVHN
    - Batch_size: 128
    - lr=0.0001, 0.0002
    - Data normalization to (-1, 1)
  - SVHN --> USPS
    - Batch_size: 256
    - lr=0.0001, 0.00002
    - Data normalization to (0, 1)

4. Discussion

- 這個task我實作了老師上課提到過的 **DSN** (Domain Seperation Networks) 和 **ADDA** (Adversarial Discriminative Domain Adaptation) ，但是經歷了長時間的debugging和fine-tuning後還是沒有辦法很好的讓model學到target和source domain的mapping關係。最終，在實作ADDA時透過調降learning rate以及讓target encoder在訓練前就去load已經訓練好的source encoder的參數，才讓model能夠更快速的收斂，並產生較佳的結果。
- 從tSNE的結果來分析，也可以發現相對於第三題DANN的結果，source和target domain的data經過encoder後的feature在高維空間中的分佈更加接近，代表target encoder在經過訓練之後能夠更好的將target domain data map到接近source domain在latent space中的分佈，這也使得classifier產生優於DANN的正確率。