


HW3 Report

學號: r08942087 姓名: 吳彬睿

Problem 1: VAE (30%)

1. Build your VAE encoder and decoder model and print the architecture  (Please use `print(model)` in PyTorch directly). Then, train your model on the [face dataset](#) and describe implementation details of your model. (Include but not limited to training epochs, learning rate schedule, data augmentation and optimizer) (5%)

● 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)
  )
)
```

● 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)
  )
)
```

● Implement details:

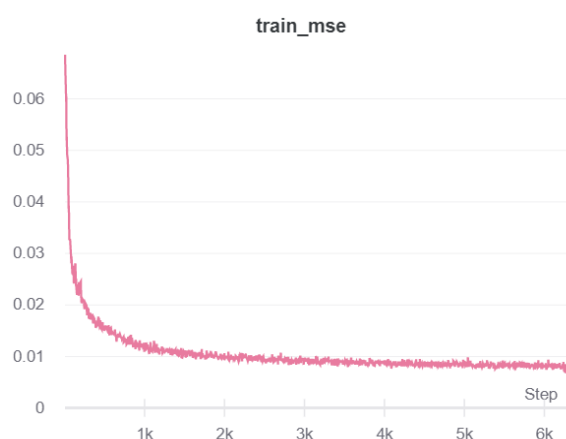
VAE 的架構，我有嘗試過最一般的 Vallina VAE 以及 Resnet VAE，後來發現 Resnet VAE 好像效果不好，收斂再不好的結果。而 Vallina VAE 我總共 train 了 20 個 epoch，到第 15 個 epoch 的時候大概就下不去了，reconstruct mse 大概在 0.0125 左右。

而在 train VAE 的時候我沒有加任何 data augmentation.

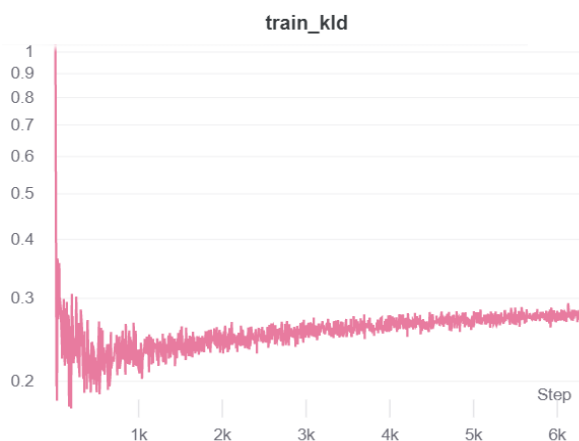
- epochs: 20
- batch size: 128
- latent dim: 64
- learning rate: $5e-3$
- kld lambda: 0.01

2. Please plot the learning curve (reconstruction loss and KL divergence) of your model. [fig1_2.png] (5%)
















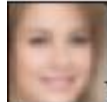

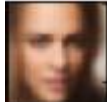
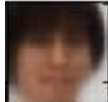

● Reconstruct loss:



● KL divergence



3. Please random choose 10 testing images and get the reconstructed images from your VAE model. Plot 10 testing images and their reconstructed results (reconstructed images and MSE) from your model. [Table 1_3] (5%)

Test Img										
Reco Img										
MSE	0.00648	0.00942	0.00826	0.01463	0.01759	0.01622	0.01340	0.02169	0.01330	0.00727

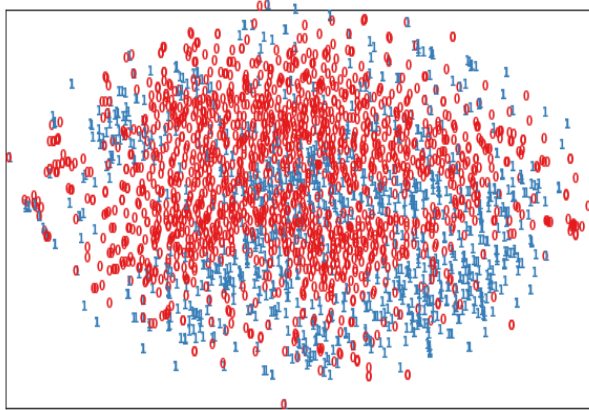
[Table1_3]

4. Now, we utilize decoder in VAE model to randomly generate images by sampling latent vectors from an Normal distribution. Please Plot 32 random generated images from your model. [fig1_4.jpg] (5%)

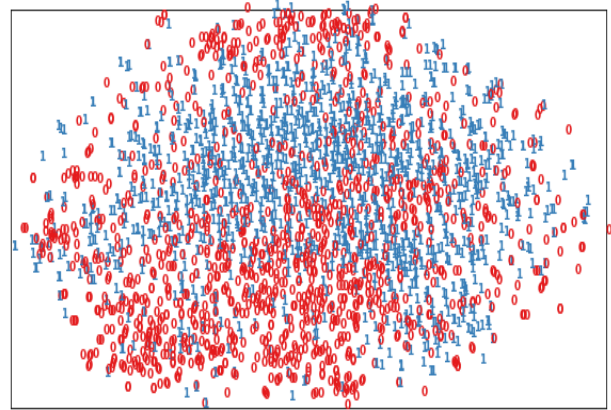


[fig1_4.png]

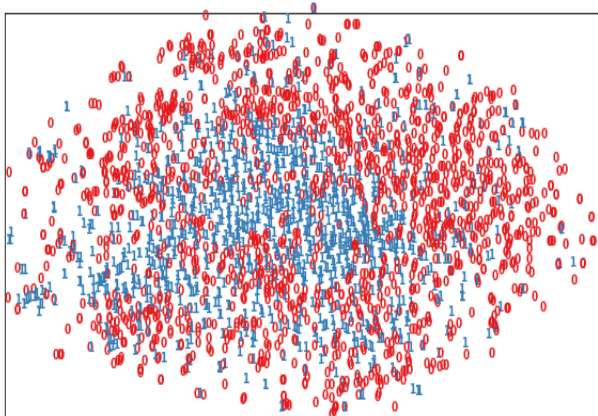
5. To analyze the latent space of your VAE model, please visualize the latent space by mapping the latent vectors of the test images to 2D space (by tSNE) and color them with respect to an attribute (e.g., gender and hair color) of your choice. (5%) (An example is shown below.)



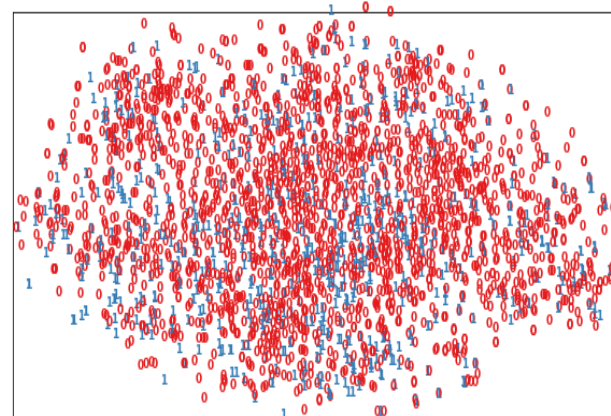
(Gender)



(Wearing lipstick)



(Smiling)



(Brown hair)

Brown hair 看起來沒有不同群的感覺，這樣代表 vae 在壓縮 latent vector 的時候其實把 brown hair 的資訊丟失掉了，因此如果有黃頭髮的人在這個 model 可能不能還原。

6. Discuss what you've observed and learned from implementing VAE. (5%)

我發現 VAE 最中間那層的 latent vector 其實不用那麼大，助教一開始給的建議是 $\text{dim}=512$ ，但是我後來嘗試 $\text{dim}=32$ 其實就可以 train 的出來了，而且效果其實不會輸很多，甚至 mse 算起來一樣。

還有我發現 VAE 跑出來的結果感覺邊緣感覺比較模糊一點，感覺這是 auto-encoder 天生的限制，為了要 minimize mean square error，那 model 只要產出比較平均的臉也許就可以達到不錯的效果，有點像是 maximize likely hood 然後就猜 mean 的地方。

Problem 2: GAN (20%)

1. Build your generator and discriminator in GAN model and print the architecture **[fig1_1.png]** (please use `print(model)` in PyTorch directly). Then, train your model on the face dataset and describe implementation details of your model. (**Include** but not limited to training epochs, learning rate schedule, data augmentation and optimizer) (5%)

```
Generator(  
  (11): 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): ReLU()  
  )  
  (12_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): ReLU()  
    )  
    (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): ReLU()  
    )  
    (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): ReLU()  
    )  
    (3): ConvTranspose2d(64, 3, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1))  
    (4): Tanh()  
  )  
)  
Discriminator(  
  (main): 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): 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)  
    )  
    (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)  
    )  
    (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))  
    (6): Sigmoid()  
  )  
)
```

[fig1_1.png]

- Implementation details
 - Data Augmentation : Random_Horizontal_Flip
 - Learning rate : 1e-4
 - Optimizer : Adam(lr=1e-4, beta=(0.5,0.999))

2. Now, we can use the Generator to randomly generate images. Please samples 32 noise vectors from Normal distribution and input them into your Generator. Plot 32 random images generated from your model.

[fig2_2.jpg] (5%)



[fig2_2.jpg]

3. Discuss what you have observed and learned from implementing GAN. (5%)

Generator 的最後一層要用 tanh 而不適 sigmoid，是我覺得最奇怪的地方，但是上網 google 發現大家這邊都是用 tanh。

在最早 hinton 發表 Gan 的架構的時候他 generator 最後一層的 activation 就是用 tanh，而原始 DCGAN 的 paper 也寫說他是用 tanh 當作最後一層的 activation function，而不是用 relu 的原因是，把 output 限制在一定的範圍，會有助於 Model 的 learning，同時有一說法是 tanh 具有對稱性，也是有助於 Model 的學習，他可以吐出較暗的圖片也可以吐出較亮的圖片，但是我覺得很奇怪，為甚麼不直接用 sigmoid 當作最後一層，因此我稍微實驗了一下，好像 train 不太起來，只能說 tanh 感覺有點神奇。

4. Compare the difference between image generated VAE and GAN, discuss

what you have observed. (5%)

VAE 跟 GAN 最主要的差異就是模糊程度，用 auto-encoder 產生出來的圖片，因為 model 為了要 minimize mse，因此會產生比較平均的照片，此效果就會造成圖片比較模糊，但是 gan 因為加入了 discriminator，當 generator 產生模糊的圖片時，discriminator 會很好的辨識出來，因此 generator 會想方設法產生跟真實一模一樣的圖片，因此圖片會比較清晰。

但 gan 有個缺點就是產生的人臉照片有些可能很符合人類的特徵，但是五官的位置可能會稍微位移，看起來像鐘樓怪人。

因此 vae 跟 gan 都有各自的優缺點就是了。

Problem 3: DAAN (35%)

1~3. Compute the accuracy on **target** domain, while the model is trained on **source** domain only. (lower bound) (3%),

source and **target** domain. (domain adaptation) (3+7%)

target domain. (upper bound) (3%)

- ✓ Use source images and labels for training
- ✓ Use source images and labels + target images for training
- ✓ Use target images and labels for training

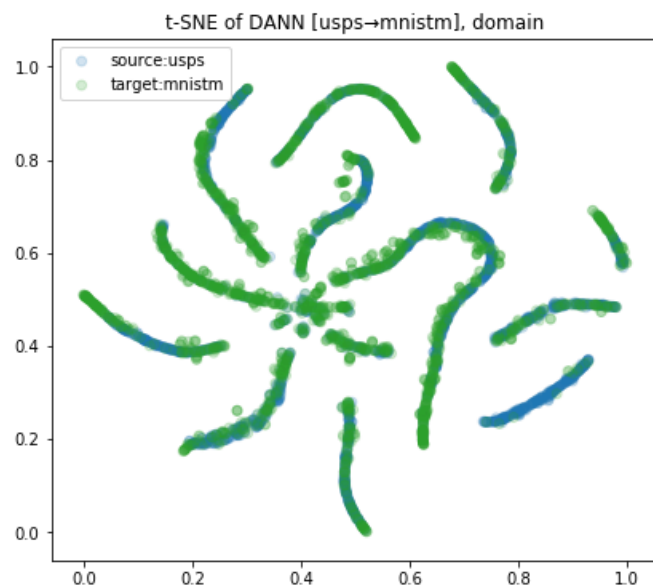
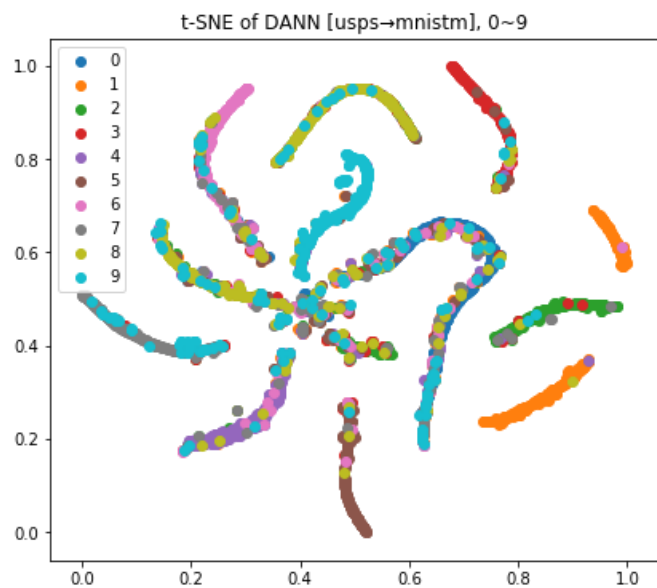
	USPS → MNISTM	MNISTM→SVHN	SVHN→USPS
Train on source	0.436	0.412	0.714
DANN	0.466 / 0.451	0.442 / 0.472	0.758 / 0.735
Train on target	0.949	0.943	0.971

(Train / Test)

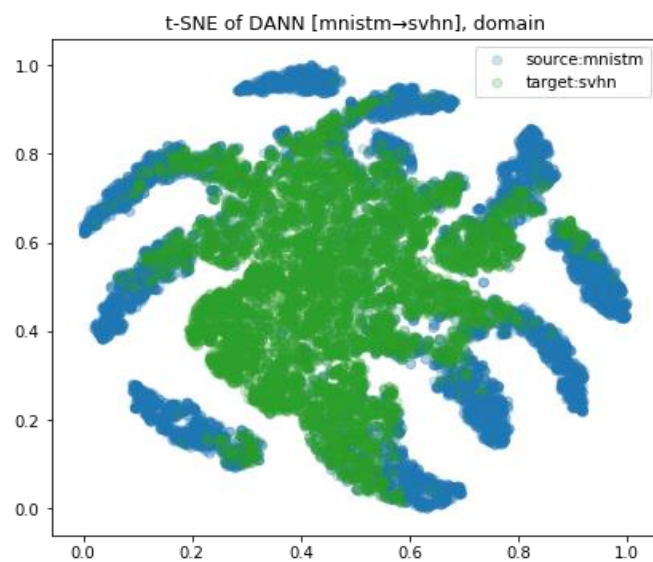
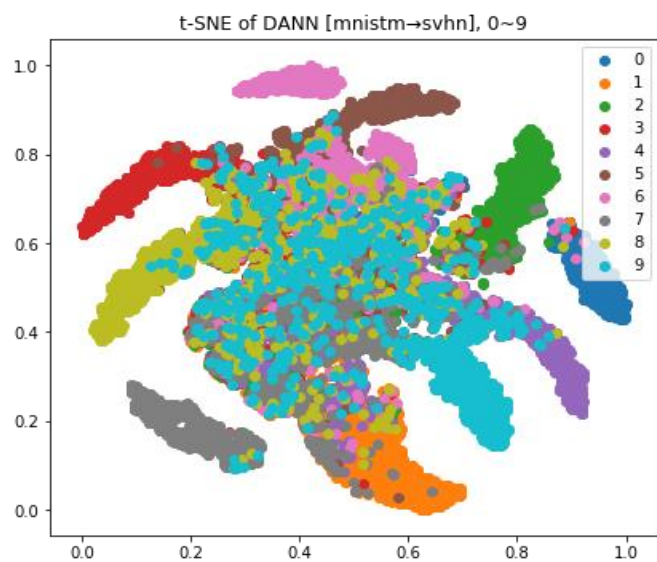
4. Visualize the latent space by mapping the *testing* images to 2D space (**with t-SNE**) and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (**source/target**). (6%)

- ✓ Note that you need to plot the figures of all the three scenarios, so you would need to plot 6 figures in total in this sub-problem.

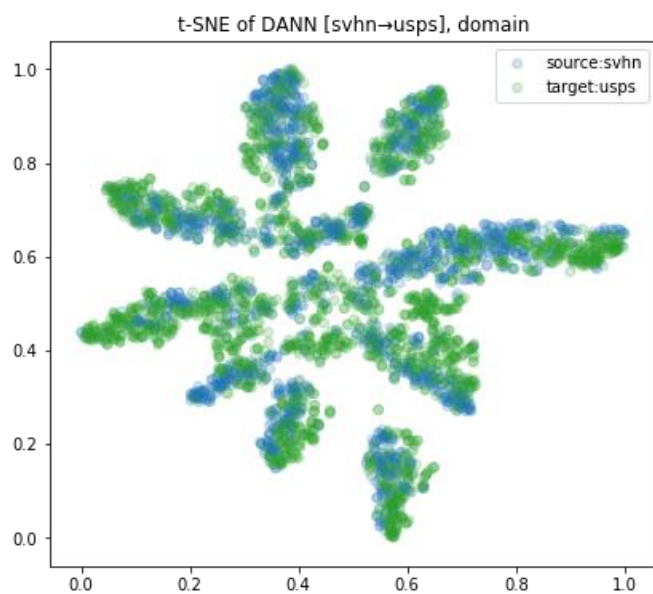
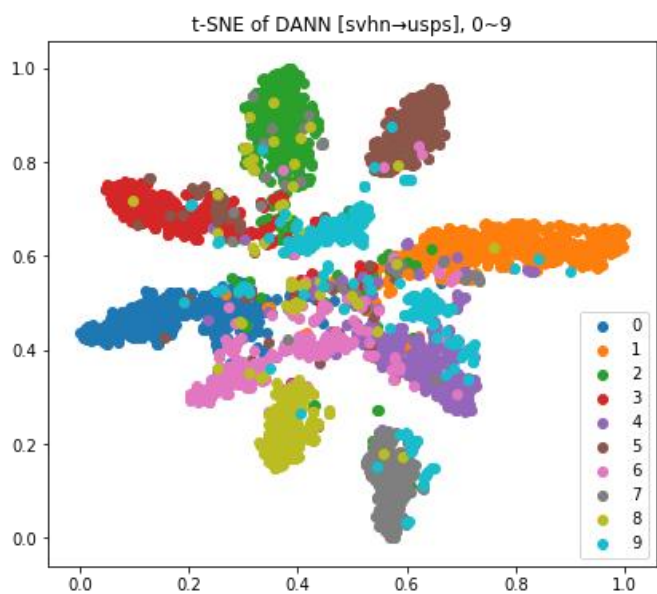
(1) USPS -> MNISTM



(2) MNISTM -> SVHN



(3) SVHN -> USPS



5. Describe the architecture & implementation detail of your model. (6%)

在 dann 架構裡面，我的 feature extractor 是用 resnet18，會用 resnet 的原因是他解決了梯度消失的問題，所以可以收斂得比較快一點，performance 也是相較其他 backbone 比較好的；而我的 classifier 我是簡單架了 3 層的 Feed forward；至於 domain predictor，我是參考網路上他們架了 5 層，並且有加入 batchnorm。

而我 train dann 是用 two-step 的方式，在同一個 iteration 之中會有兩個 step，第一步先更新 domain predictor，讓分辨器變強；第二步在根據 domain predictor 的反向梯度，以及 label classifier 的梯度，更新 feature extractor & label classifier。

6. Discuss what you have observed and learned from implementing DANN. (7%)

Train dann 是用 two-step 的方式，第一步先更新 domain predictor，讓分辨器變強；第二步在根據 domain predictor 的反向梯度，以及 label classifier 的梯度，讓 feature extractor 可以辨認出不同數字，同時做到 domain fusion 的效果，把兩個 domain 拉近。

我發現 dann 的重點其是是在於 domain fusion，既使分類器 train 的再好，如果兩個 domain 沒有很好的混和的話，那麼 target domain 的 performance 並不會好。因此在訓練 feature extractor 的時候會有一個 tunable 的參數 lambda，這個參數是代表 domain fusion 的 loss 的比例，我看網路上的 open source 他們會用個 exp 的函數去 model 他，lambda 從 0 慢慢上升到 1，等於是前期先 train label classifier，而後期在做到 domain fusion，不過我實驗發現直接把 lambda 設定 0.01 也可以訓練得很好，因此我就沒有去讓 lambda 變成一個可調的參數了。

dann 有時候會有梯度爆炸的問題，因此我去網路上搜尋也有人遇到類似的問題，主要在於分類器被訓練太好了，而產生器又要想辦法騙過分類器，有時候就會有產生很大的梯度，因此要把這些梯度做 normalize 到 $[-0.01, 0.01]$ 之間，這樣 model 才不會 train 壞掉。

Problem 4: Improved UDA model (35%)

1. Compute the accuracy on **target** domain, while the model is trained on **source** and **target** domain. (domain adaptation) (6+10%)

Use source images and labels + target images for training

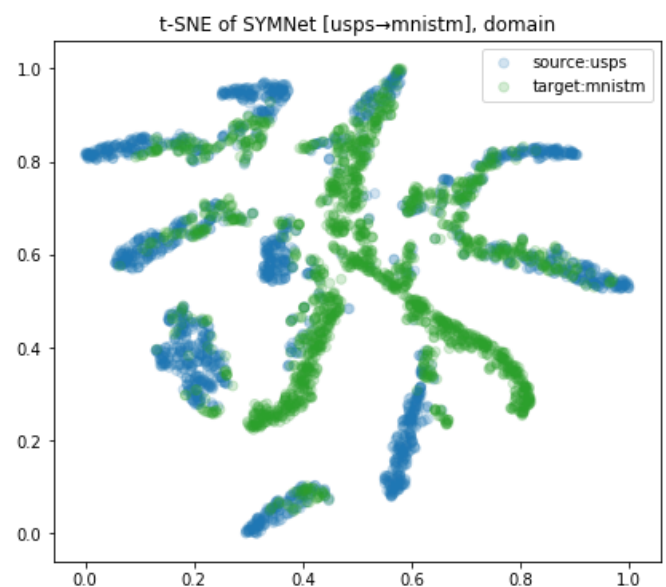
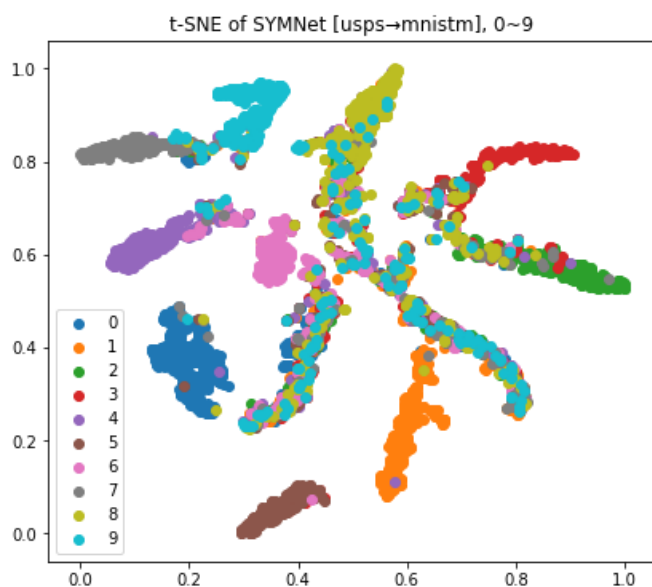
	USPS → MNISTM	MNISTM→SVHN	SVHN→USPS
Train on source	0.436	0.412	0.714
DANN	0.466 / 0.451	0.442 / 0.472	0.758 / 0.735
SYMNET	(X) 0.278 / 0.349 ↓	0.532 / 0.541 ↑	0.861 / 851 ↑
Train on target	0.949	0.943	0.971

(Train / Test)

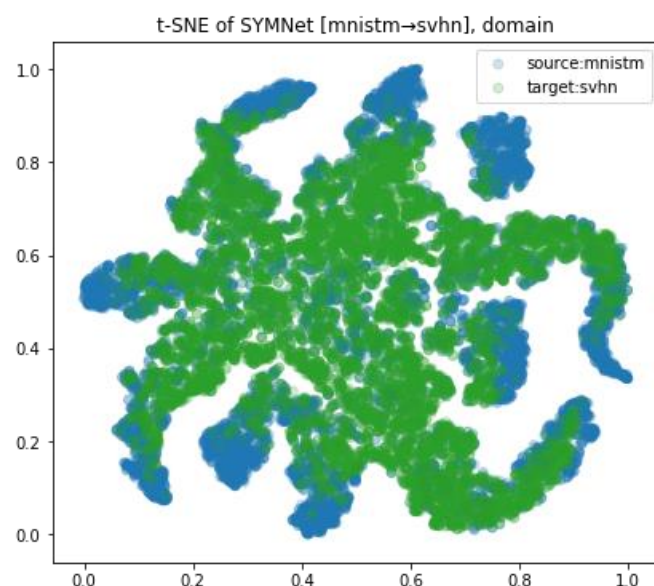
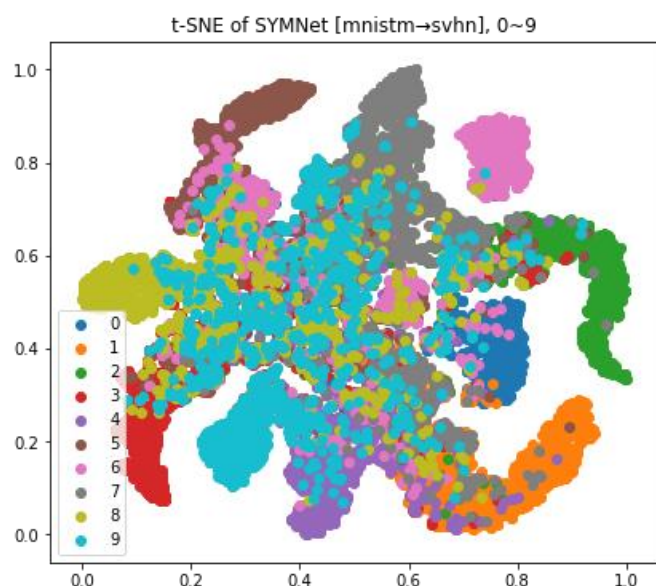
2. Visualize the latent space by mapping the *testing* images to 2D space (**with t-SNE**) and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (**source/target**). (6%)

✓ Note that you need to plot the figures of all the three scenarios, so you would need to plot 6 figures in total in this sub-problem.

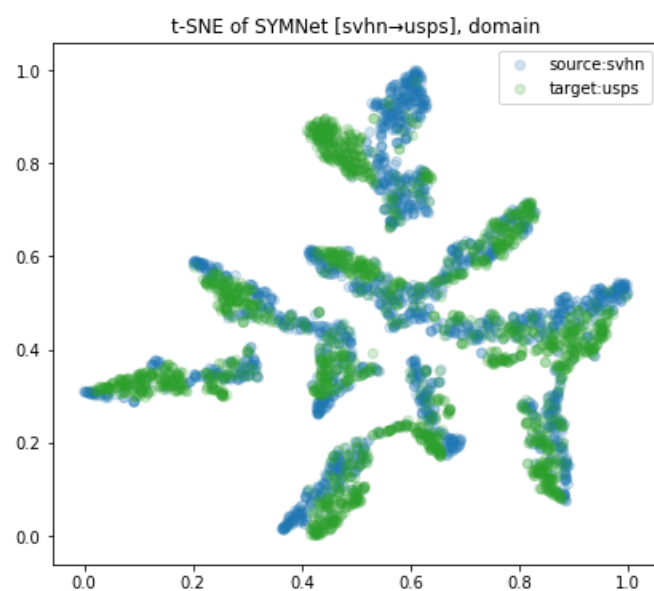
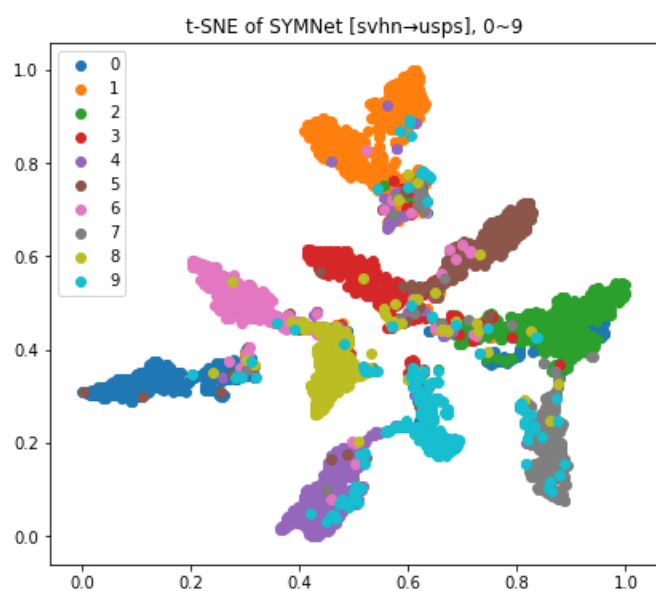
(1) USPS -> MNISTM



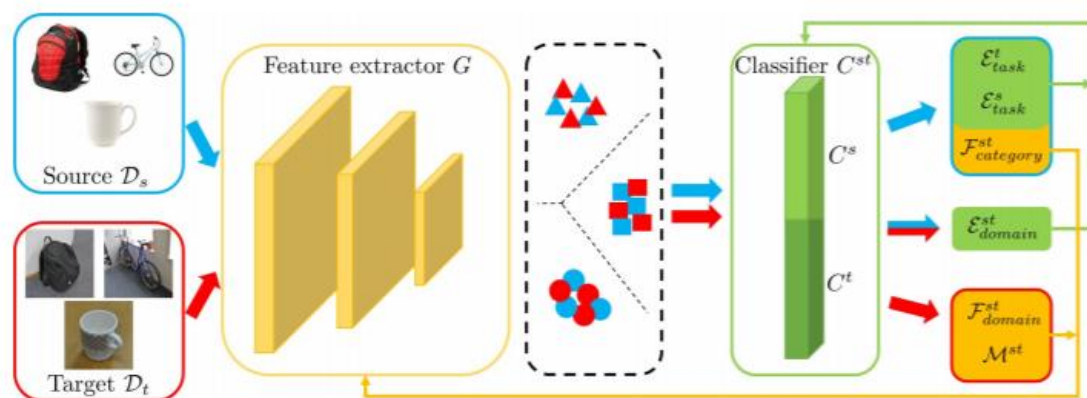
(2) MNISTM -> SVHN



(3) SVHN -> USPS

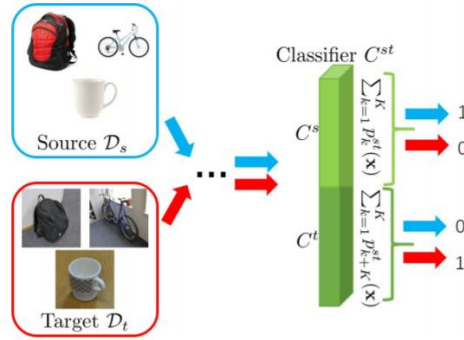


3. Describe the architecture & implementation detail of your model. (6%)



我是實作 SYMNet 的 UDA 架構，而 feature extractor 還是使用 resnet18，這個 model

的 loss 是以下:



- Source task classifier

$$\min_{C^s} \mathcal{E}_{task}^s(G, C^s) = -\frac{1}{n_s} \sum_{i=1}^{n_s} \log(p_{y_i^s}^s(\mathbf{x}_i^s)). \quad (5)$$

- Cross-Domain Learning of Target Task Classifier

$$\min_{C^t} \mathcal{E}_{task}^t(G, C^t) = -\frac{1}{n_s} \sum_{i=1}^{n_s} \log(p_{y_i^s}^t(\mathbf{x}_i^s)). \quad (6)$$

- Domain Discrimination

$$\begin{aligned} \min_{C^{st}} \mathcal{E}_{domain}^{st}(G, C^{st}) = & -\frac{1}{n_t} \sum_{j=1}^{n_t} \log\left(\sum_{k=1}^K p_{k+K}^{st}(\mathbf{x}_j^t)\right) \\ & -\frac{1}{n_s} \sum_{i=1}^{n_s} \log\left(\sum_{k=1}^K p_k^{st}(\mathbf{x}_i^s)\right), \quad (7) \end{aligned}$$

SYMNet 基本上還是 GAN-based 的方法，有個 discriminator 去辨認說他是從哪個 domain 來的資料，而現在沒有一個 domain classifier，他把 discriminator 的任務塞在 task classifier 裡面，讓他們同時存在同一個分類器裡面，而如果是 source domain 來的資料，那麼上面的 classfiier 反應會比較大，反之 target domain 的則是下面的 classifier 會反應較大。

同時還有個 Loss 是在讓兩個 classifier 意見可以一致的，他希望兩個 classifier 可以吐出來的 distriubution 是相似的。

4. Discuss what you have observed and learned from implementing your improved UDA model. (7%)

我覺得第一組 domain transfer(USPS -> MNIST) 很難訓練的好ㄟ，我有嘗試過 ADDA: Adversarial Discriminator Domain Adaptation [[paper](#)]；SBADA: Symmetric Bi-Directional Adaptive GAN [[paper](#)]；SYMNet: Domain-Symmetric Networks for Adversarial Domain Adaptation [[paper](#)]，都 train 不太起來，感覺這組的轉換是比較難學習的，我後來去看了一下 MNIST 資料集的圖片，發現他的圖片還蠻多樣化的，有的顏色對比很特別，有的背景又很雜亂，好像沒有一個固定的 pattern 可以讓 model 辨認，因此我覺得 train 不太起來也許合理，不過因為後來沒時間了所以就沒又繼續嘗試別的 model 了。

最後我是選定 SYMNet，他這個架構可以讓 domain fusion 的非常快，但是這個方法的缺點是比較不穩定，如果 train 太久她結果會變得很爛，感覺收斂到一個我們不想要的結果，通常 SYMNet 的方法，在前 2、3 個 epoch 就可以超過 ADDA 了，如果沒有超過那就是可以停掉重新 train 了。同時我覺得 SYMNet 也是比較不穩定的方法，感覺蠻吃起始的數值的，即使一模一樣的參數下去 train，結果也可能差異到 10%。

[Reference]

- Generator tanh: <https://stackoverflow.com/questions/41489907/generative-adversarial-networks-tanh>
- Dann implementation: https://colab.research.google.com/drive/1cTdIDT_fsBWGbaljhPSmBI6gwkw-tQ2H
- Dann implementation: <https://github.com/fungtion/DANN>
- Symnet: Domain-Symmetric Networks for Adversarial Domain Adaptation [[paper](#)]
- Symnet implementation: <https://github.com/YBZh/SymNets>