

Problem1 GAN

1. Describe the architecture & implementation details of your model

我用 DCGAN，架構如下圖顯示，latent dimension 是 100 維，generator 最後一層的輸出經過 tanh activation function 使輸出在-1 和 1 之間。

Real image 的 pixel 值會先除 255 再 normalize 到-1 和 1 之間。

Discriminator 則用來分辨圖片的真假，是一 binary classification problem

Optimizers :G 和 D 皆使用 Adam，learning rate=2e-4，beta1=0.5

Loss : binary cross entropy

```
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)  
    (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)  
    (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)  
    (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)  
    (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)  
    (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)  
    (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)  
    (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)  
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)  
    (12): Sigmoid()  
  )  
)
```

2. Plot 32 random images generated from your model



3. Discuss what you've observed and learned from implementing GAN
 - a. 以 DCGAN 的結果來看，已經可以生成一些很正常的人臉圖了，但還是會有五官扭曲的狀況發生，尤其是產生人的側臉的時候。
 - b. 設計 model 的架構時要讓 generator 和 discriminator 的參數量不要相差太多，因為兩者要互先對抗，如果訓練的過程中某一方太強，就會使整個 GAN 的訓練失敗。
 - c. Generator 和 discriminator 要分開訓練的結果會比較好，實作時有嘗試 update D 的參數兩次後再 update G 的參數一次，但最終得到的結果和兩者各 update 一次的結果差不多。

Problem2 ACGAN

1. Describe the architecture & implementation details of your model

仍是用 DCGAN，架構如下圖所示，latent dimension 為 101，多的那一個維度用來決定 attribute 的有無，本次實作的 attribute 是 smiling

Discriminator 除了要分辨圖片的真假，還要分辨 attribute 的有無，兩者都是 binary classification problem。

Optimizers :G 和 D 皆使用 Adam，learning rate=2e-4，beta1=0.5

Loss：皆是 binary cross entropy

```

Generator(
  (decoder): Sequential(
    (0): ConvTranspose2d(101, 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)
    (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)
    (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)
    (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)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)

Discriminator(
  (decoder): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (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)
    (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)
    (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)
    (11): Conv2d(512, 64, kernel_size=(4, 4), stride=(1, 1), bias=False)
  )
  (dis): Linear(in_features=64, out_features=1, bias=True)
  (clas): Linear(in_features=64, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)

```

2. Plot 10 random **pairs** of generated images from your model, where each **pair** should be generated from the same random vector input but with opposite attribute. This is to demonstrate your model's ability to disentangle features of interest.

上 : smiling 下 : without smiling



3. Discuss what you've observed and learned from implementing ACGAN
 - a. 和 GAN 相比，因 ACGAN 的 discriminator 還需要多分辨 attribute 的有無，因此 loss 下降的速度比較慢。
 - b. 因為 ACGAN 是 supervised learning 的關係，其實並沒有很難 train，從前一題圖的結果來看，無論是露齒的大笑或是單純嘴角上揚的微笑都可以看出和沒有笑的差異。
 - c. Generator 和 discriminator 也是要分開訓練的結果會比較好，實作的時候我選擇 update D 的參數一次後 update G 的參數兩次，但我也不確定為何這樣做的結果比較好。

Problem3 DANN

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

Source domain	Target domain	Accuracy
USPS	MNIST-M	20.21%
MNIST-M	SVHN	24.02%
SVHN	USPS	68.32%

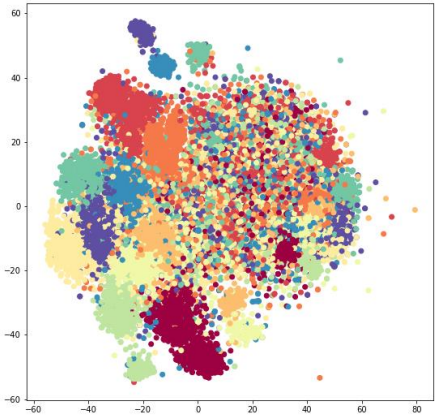
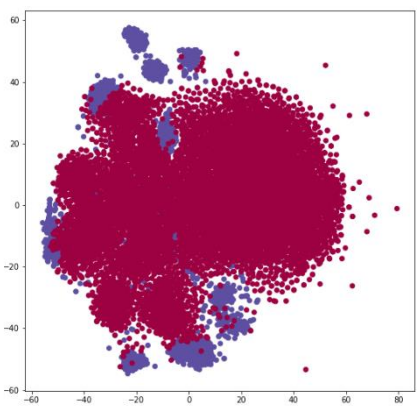
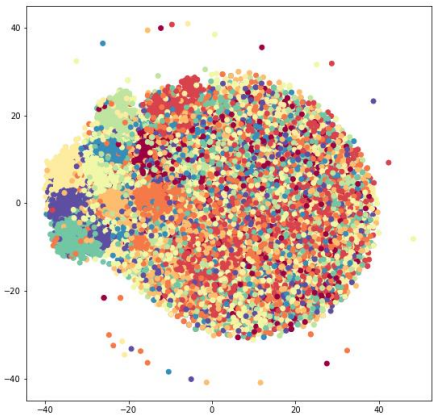
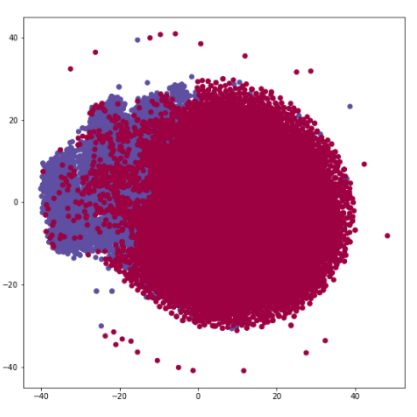
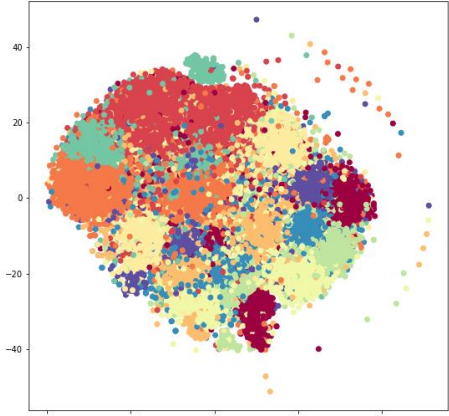
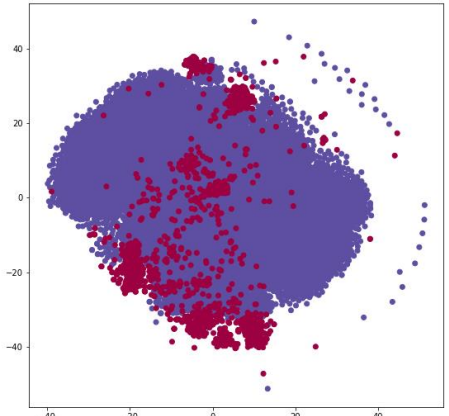
2. Compute the **accuracy** on **target** domain, while the model is trained on **source and target** domain. (domain adaptation)

Source domain	Target domain	Accuracy
USPS	MNIST-M	39.95%
MNIST-M	SVHN	44.89%
SVHN	USPS	60.089%

3. Compute the **accuracy** on **target** domain, while the model is trained on **target** domain only. (upper bound)

Target domain	Accuracy
MNIST-M	97.76%
SVHN	91.11%
USPS	97.27%

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**).

source/target	different digit class	different domains
USPS/ MNITM-M		
MNIST-M/ SVHN		
SVHN/ USPS		

5. Describe the architecture & implementation detail of your model

Model 架構如下圖，我依照 DANN 的示意圖，用 3 層 CNN 來 extract feature，而 digit classifier 和 domain classifier 各為 3 層的 DNN。

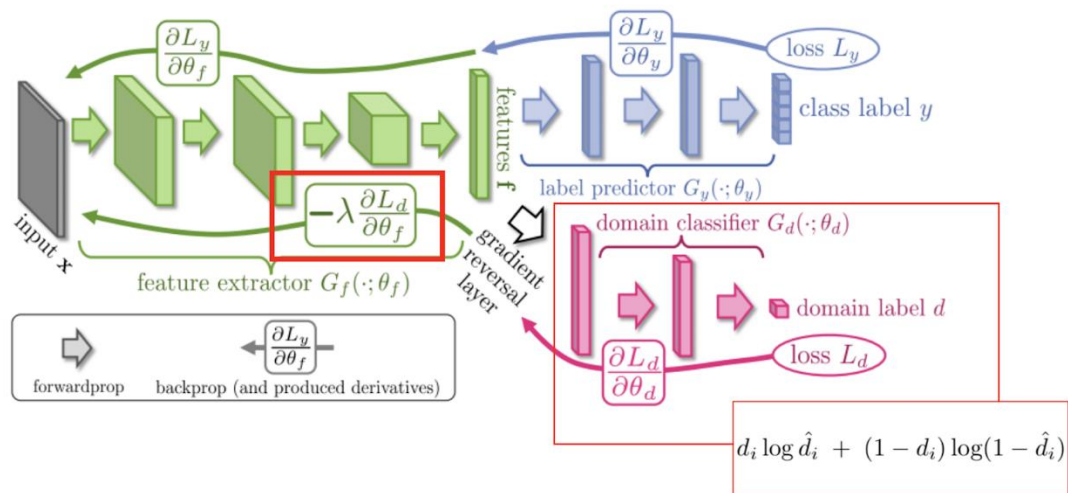
用來畫 TSNE 的 feature 是經 3 層 CNN 後的結果，維度是 1152。

Optimizers : Adam , learning rate=0.001

Loss : digit classifier 用 cross entropy , domain 用 binary cross entropy

```
DANNNet(  
  (conv2): Sequential(  
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (1): LeakyReLU(negative_slope=0.05)  
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (4): Dropout2d(p=0.3)  
  )  
  (conv3): Sequential(  
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (1): LeakyReLU(negative_slope=0.05)  
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (4): Dropout2d(p=0.4)  
  )  
  (conv4): Sequential(  
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (1): LeakyReLU(negative_slope=0.05)  
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (4): Dropout2d(p=0.5)  
  )  
  
  (fc1): Sequential(  
    (0): Linear(in_features=1152, out_features=128, bias=True)  
    (1): ReLU()  
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): Dropout(p=0.5)  
  )  
  (fc2): Sequential(  
    (0): Linear(in_features=128, out_features=128, bias=True)  
    (1): ReLU()  
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): Dropout(p=0.5)  
  )  
  (fc3): Linear(in_features=128, out_features=10, bias=True)  
  
  (do1): Sequential(  
    (0): Linear(in_features=1152, out_features=128, bias=True)  
    (1): ReLU()  
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): Dropout(p=0.5)  
  )  
  (do2): Sequential(  
    (0): Linear(in_features=128, out_features=128, bias=True)  
    (1): ReLU()  
    (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (3): Dropout(p=0.5)  
  )  
  (do3): Sequential(  
    (0): Linear(in_features=128, out_features=1, bias=True)  
    (1): Sigmoid()  
  )  
)
```

DANN 示意圖



6. Discuss what you've observed and learned from implementing DANN

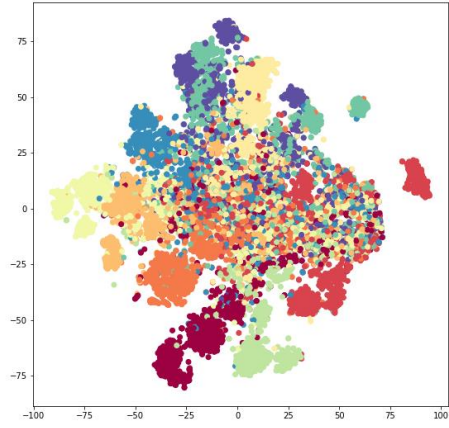
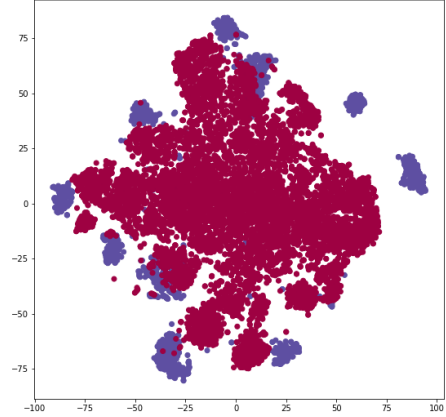
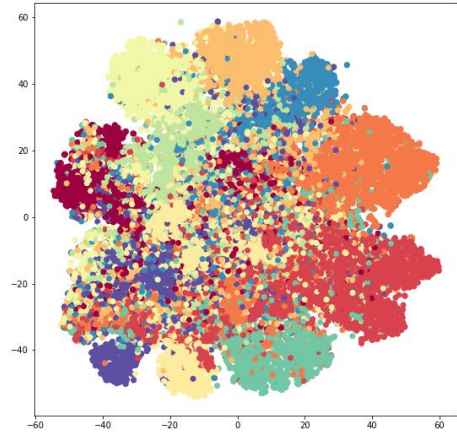
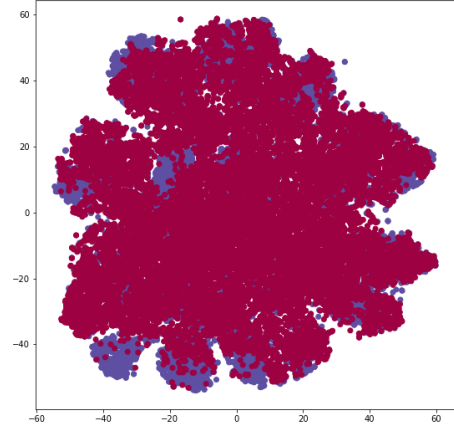
- 因為 model 的架構不大，很快就會知道結果如何，如果 batch size 調太大，可能造成 target domain accuracy 不如預期。
- 在 SVHN->USPS 這個 task，lower bound 高達 68.32%，而 DANN 卻只有大約 60%，推測是因為 USPS 的圖片是黑白的而且 data 數量偏少，lower bound 的 model 雖然只看過 SVHN 的 data，但 USPS 相對 SVHN 來說比較簡單，因此 lower bound model 在 USPS 的表現仍不錯。而 DANN model 因為同時要學 digit label 和 domain label，導致 DANN model 在辨識 SVHN 的 digit label 時就已經做的不夠好，因此最終在 USPS 上的 accuracy 無法突破 lower bound。
- TSNE 的結果皆是用全部的 test data 畫出來的，結果離理想狀態有很大的差距，尤其是 MNIST-M->SVHN，有非常多的點擠在一起，代表單純的 DANN model 的 latent space 無法很好的學會不同 domain 間的 invariant feature。
- 即使 TSNE 的結果不如預期，但在 USPS->MNISTM-M 和 MNIST-M->SVHN 這兩個 task 上，DANN 的結果都比 lower bound 好了大約 20%，代表 DANN 有助於辨識 target domain 的資料。

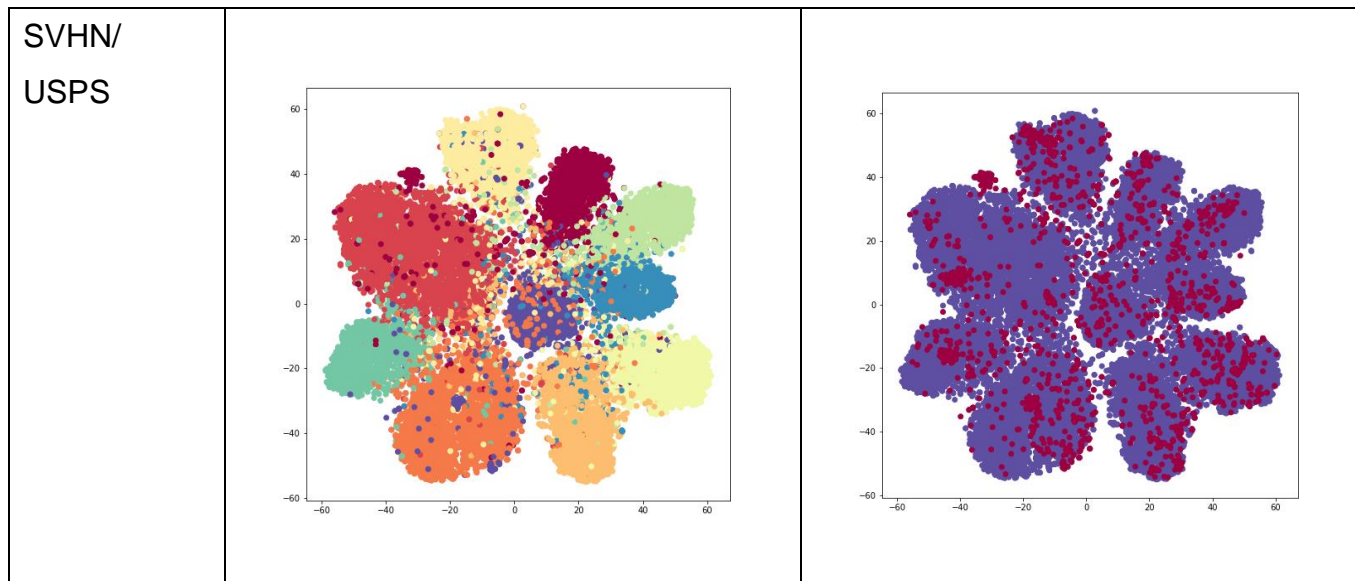
Problem4 Improved UDA model

- Compute the **accuracy** on **target** domain, while the model is trained on **source** and **target** domain. (domain adaptation)

Source domain	Target domain	Accuracy
USPS	MNIST-M	53.4%
MNIST-M	SVHN	53.753%
SVHN	USPS	66.467%

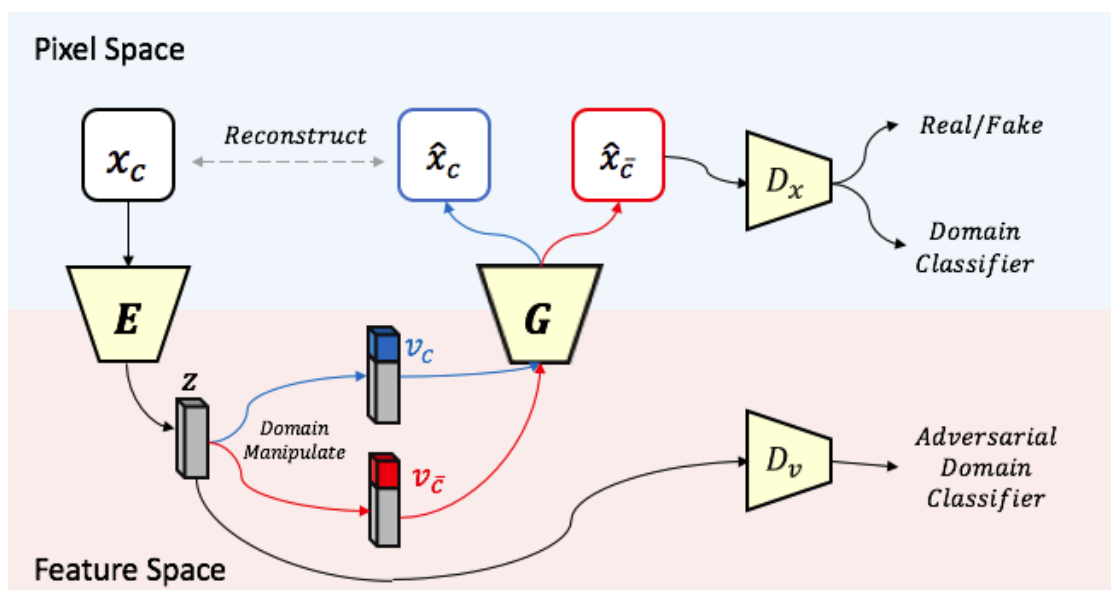
2. Visualize the the latent space by mapping the *testing* images to 2D space (with t-SNE) and use different colors to indicate data of (a) different digits classes 0-9 and (b) different domains ([source](#)/[target](#)).

source / target	different digit class	different domains
USPS/ MNITM-M		
MNIST-M/ SVHN		



3. Describe the architecture & implementation detail of your model.

我參考 UFDN [2]的 model 來實作 improved model，下圖是 UFDN 的架構



UFDN 其實是 VAE-GAN 的架構，上圖的 z 就是 latent space。 D_v 的效果是讓 encoder 能學出 domain invariant 的 latent space。

經過 VAE 後產生的圖，會希望能和原始輸入圖片越像越好，並透過 D_x 來判斷產生的圖屬於哪個 domain，以及 source domain 的圖片的 label。

在 test 的時候，會將 target domain 的圖經過 VAE 還原成類似 source domain 的圖片，再判斷圖片的 label。

用 latent space 的 feature 來話 TSNE，維度是 2048

Optimizers : VAE、digit classifier : Adam，learning rate = $1e-4$

Domain classifier : Adam，learning rate = $1e-4$

Loss : VAE: MSE 和 KL divergence loss , digit classifier : cross entropy

Domain classifier : binary cross entropy

下圖是 VAE 的架構，而 digit 和 domain classifier 都是一層的 DNN

```
UFDN(  
  (enc_0): Sequential(  
    (0): Conv2d(3, 128, kernel_size=(4, 4), 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=0.2)  
  )  
  (enc_1): Sequential(  
    (0): Conv2d(128, 256, kernel_size=(4, 4), 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=0.2)  
  )  
  (enc_2): Sequential(  
    (0): Conv2d(256, 512, kernel_size=(4, 4), 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=0.2)  
  )  
  (enc_3): Sequential(  
    (0): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2)  
  )  
  (enc_mu): Sequential(  
    (0): Conv2d(1024, 2048, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
  )  
  (enc_logvar): Sequential(  
    (0): Conv2d(1024, 2048, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
  )  
  
  (dec_0): Sequential(  
    (0): ConvTranspose2d(2050, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2)  
  )  
  (dec_1): Sequential(  
    (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), 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=0.2)  
  )  
  (dec_2): Sequential(  
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), 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=0.2)  
  )  
  (dec_3): Sequential(  
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), 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=0.2)  
  )  
  (dec_4): Sequential(  
    (0): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))  
    (1): Tanh()  
  )  
)
```

4. Discuss what you've observed and learned from implementing your improved UDA model
- a. 因為 model 的架構比較複雜，需要花比較長的時間訓練，但訓練的過程中 target domain 的正確率會緩慢上升，而且比較不會 overfit 在 source domain 的圖片。
 - b. SVHN->USPS 的正確率還是略低於 lower bound，應該和第三題提到的原因類似，因為 USPS 的圖片相對簡單，因此只看過 SVHN 的 model 表現就很好了。但是有時間調整 UFDN 的 model 參數和訓練的話，我認為有機會超過 lower bound。
 - c. TSNE 也是用全部 test data 畫出，和 DANN 的結果相比，UFDN 的 latent space 學得比較好，能看出相同數字的圖片被分到同一群，而不是不同顏色的點全部擠在一起，其中 SVHN->USPS 的結果是最好的。
 - d. 由 target domain 的正確率和 TSNE 的結果來看，UFDN 的 latent space 確實有學到一些 domain invariant feature，比 DANN 的成果好不少。

Reference

[1] pytorch DCGAN tutorial

https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[2] UFDN paper

<https://arxiv.org/abs/1809.01361>

[3] UFDN github

<https://github.com/Alexander-H-Liu/UFDN>