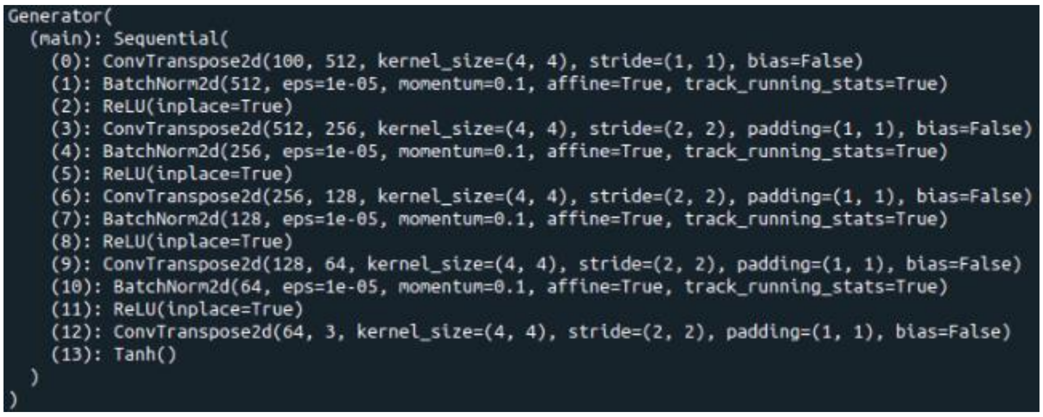
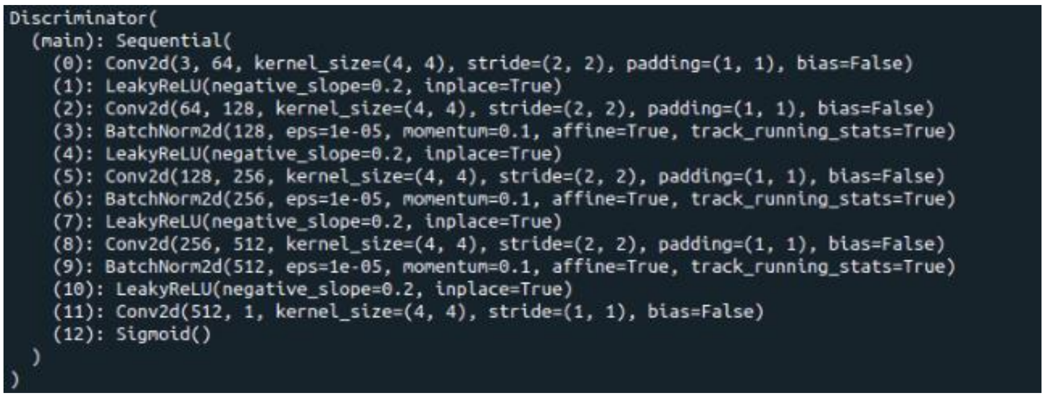
Hw2 R10945004_孫欽鉉

Problem 1: GAN

1. Build your generator and discriminator from scratch and show your model architecture in your report. (5%)



```
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()
  )
)
```

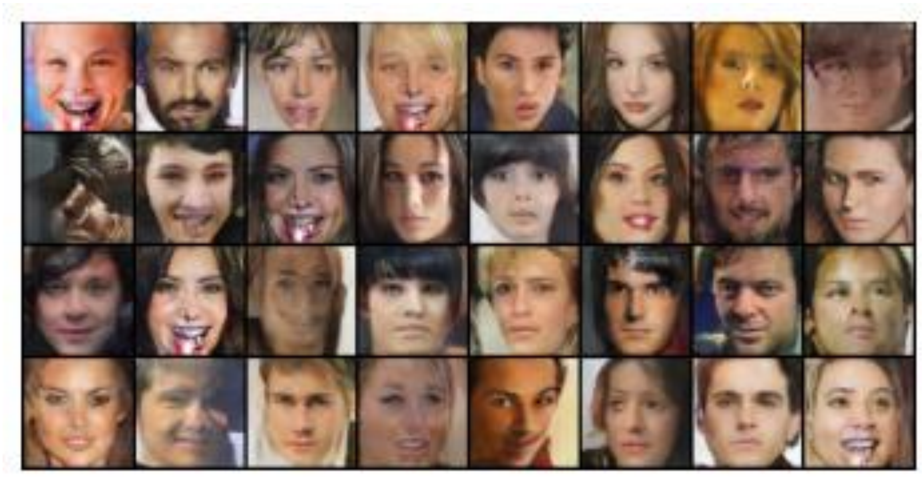


```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, 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()
  )
)
```

Expain:

我的 gan 所採用的架構主要是參考助教提供的那篇 paper 裡面的架構，根據 paper 指出使用 adam optimizer 在 generator 和 discriminator，learning rate 為 0.0002，betas 為(0.5,0.999)，batchsize 為 128，generator 所使用的 latent space size 為 100，在 train 之前會將 data 做前處理使其 normalize 在 -1~1 之間，loss function 則是使用 binary cross entropy loss function，而我 train 了 500 個 epoch 每個 epoch 都會 print 出 IS 的分數，如果 IS 的分數高於 strong baseline 則儲存起來，之後在測試各個 checkpoint 的 FID 分數，找出最好的結果。

2. show the first 32 images in your report. (5%)

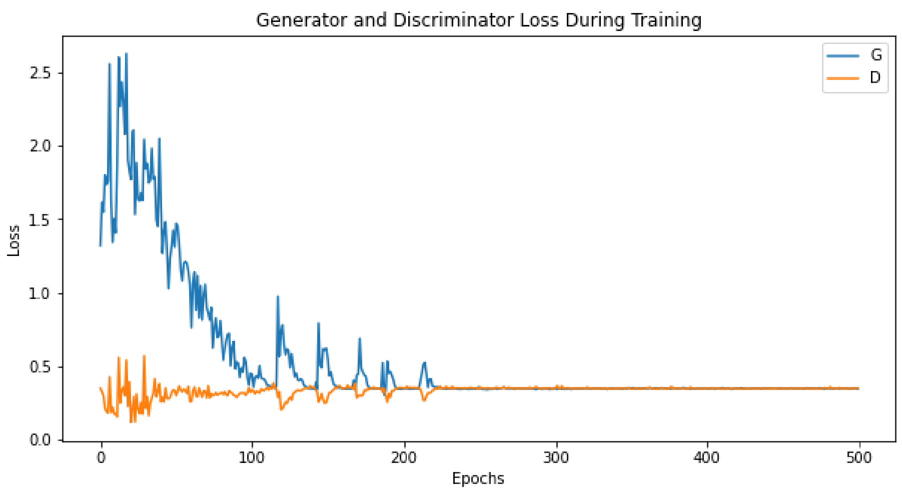


3. calculate FID and IS to evaluate your generated images. (20% total)

FID : 27.06

IS : 2.198

4. Discuss what you’ve observed and learned from implementing GAN. (5%)



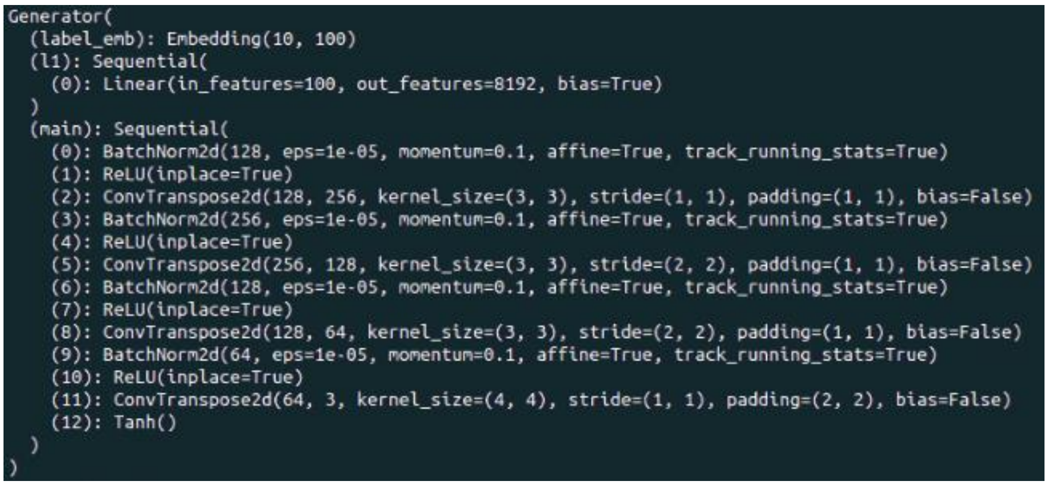
Expain:

在訓練的過程中非常的艱辛，我發現一開始 generator 的 loss 會比較低，之後 loss 會上升，這顯示出我們的 discriminator 有開始在辨識出真正的 data 和假的 data，再過一段時間之後，generator 的 loss 會開始降低，表示我們的 generator 已經慢慢學會真正的 data 該怎麼生出來，該怎麼去騙過 discriminator，但是 loss 下降不一定代表他訓練出來的圖片很像是真的，也有可能是全都是雜訊但我們的 discriminator 太爛所導致的，所以訓練的時候這點也

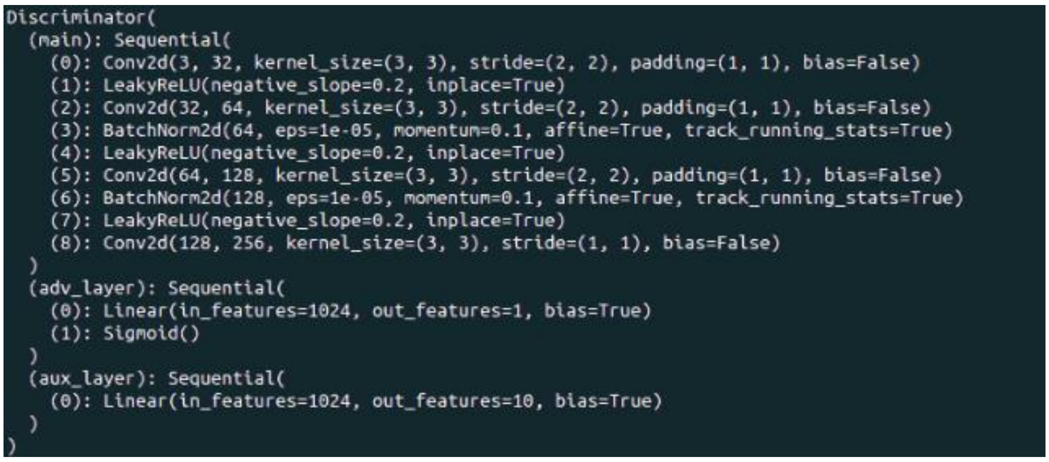
需要好好注意，最後 discriminator 和 generator 的 loss 會達到一個動態平衡，此時我發現訓練越久雖然感覺 loss 沒變或者 loss 微微的上升，但是我們 generator 生出來的圖片的品質是有增加的。還有一個發現就是使用 SGD 在 discriminator 的話 loss 的曲線會變得很平滑，相比 adam 訓練過程中會有許多的震盪，但因為我覺得 adam 出來的結果比較好，所以最後還是使用 adam 來訓練。

Problem 2: ACGAN (30%)

1. Build ACGAN model from scratch and show your model architecture in your report. (10%)



```
Generator(
  (label_emb): Embedding(10, 100)
  (l1): Sequential(
    (0): Linear(in_features=100, out_features=8192, bias=True)
  )
  (main): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): ReLU(inplace=True)
    (5): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): ReLU(inplace=True)
    (8): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2), bias=False)
    (12): Tanh()
  )
)
```



```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), bias=False)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=1024, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

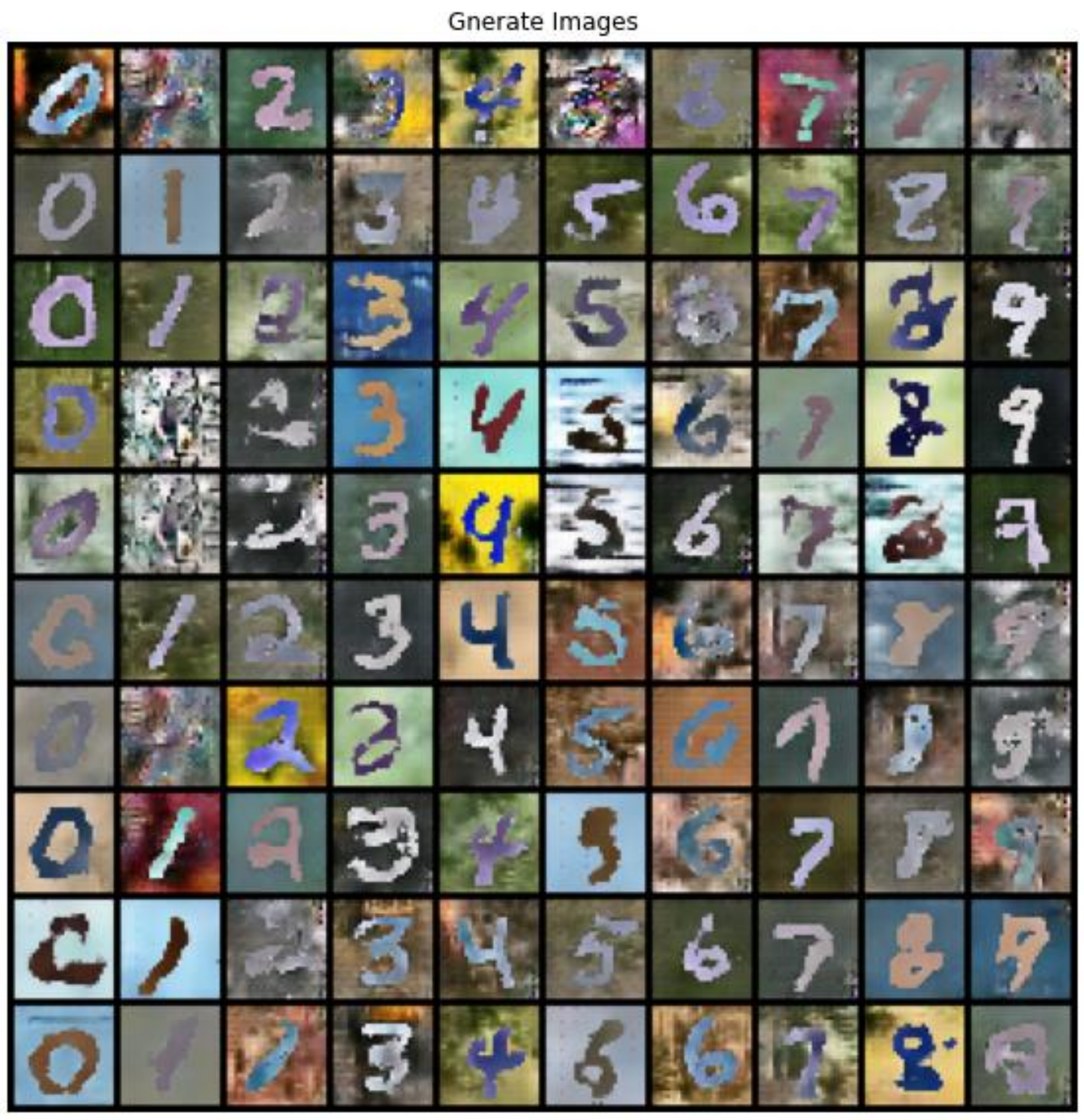
Explain:

主要的架構是承接第一題的 GAN 的架構，但是我發現如果直接丟進去 deconvolution layer 的話，generator 會 train 不太起來，所以我在 generator 前面加了一層 linear 層來接收 input，並把 output size 定為 128*8*8，之後再把一維的 vector reshape 成 128 個 8*8 的向量，也就是說我們的 deconv 層是從 8*8 的圖片開始長慢慢長成 28*28 的大小。而在訓練的時候參數基本都承接第一題的 GAN 設定，只是因為我們多了 label 要判斷，所以在 discriminator linear 層是平行接入 deconv 層的 output，而一個 linear 判斷圖片真假、一個 linear 判斷 label 分類，所以 loss function 的部分判斷真假的 linear layer 使用 binary cross entropy 而判斷 label 的分類則是使用 cross entropy function。

2. We will evaluate your generated output by the classification accuracy with a pretrained digit classifier, and we have provided the model architecture [digit_classifer.py] and the weight [Classifier.pth] for you to test. (15%)

Acc : 92.5%

3. Show 10 images for each digit (0-9) in your report. You can put all 100 outputs in one image with columns indicating different digits and rows indicating different noise inputs. (5%)

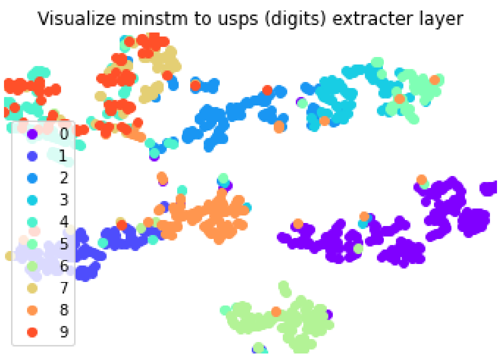


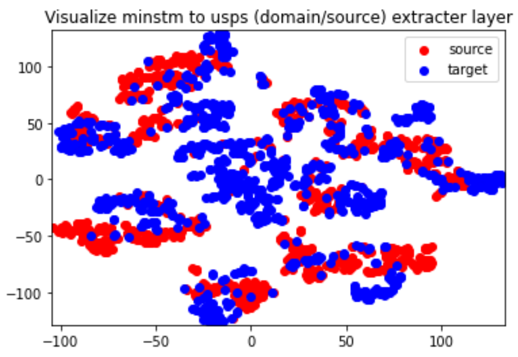
Problem 3: DANN (35%)

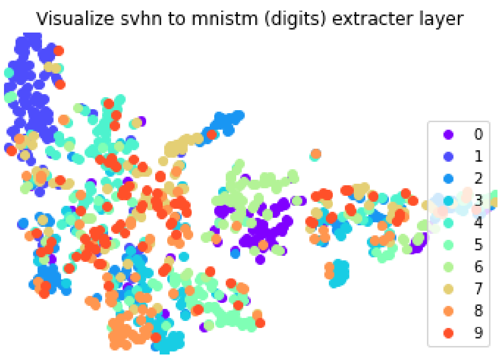
1. Compute the accuracy on target domain, while the model is trained on source domain only. (lower bound) (3%)
2. Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation) (4+9%)
3. Compute the accuracy on target domain, while the model is trained on target domain only. (upper bound) (3%)

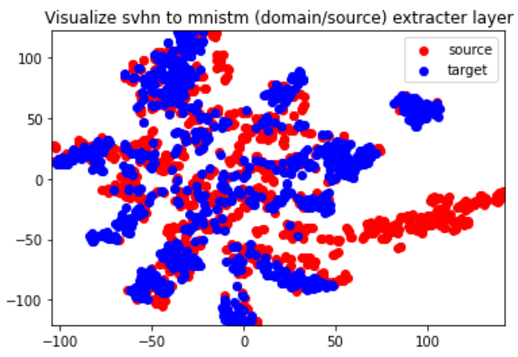
	MNIST-M -> USPS	SVHN -> MNIST-M	USPS -> SVHN
Trained on source	48.53%	36.37%	6.74%
Adaptation (DANN/Improved)	73.59%	43.25%	28.48%
Trained on target	97.36%	93.18%	92.6%

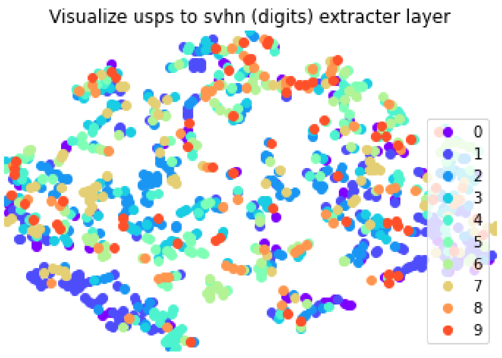
4. Visualize the latent space by mapping the testing images to 2D space with t-SNE and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (source/target). (9%)

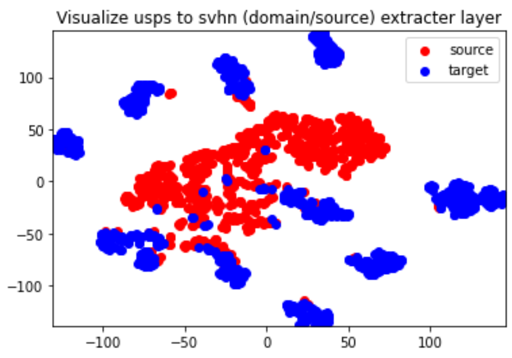




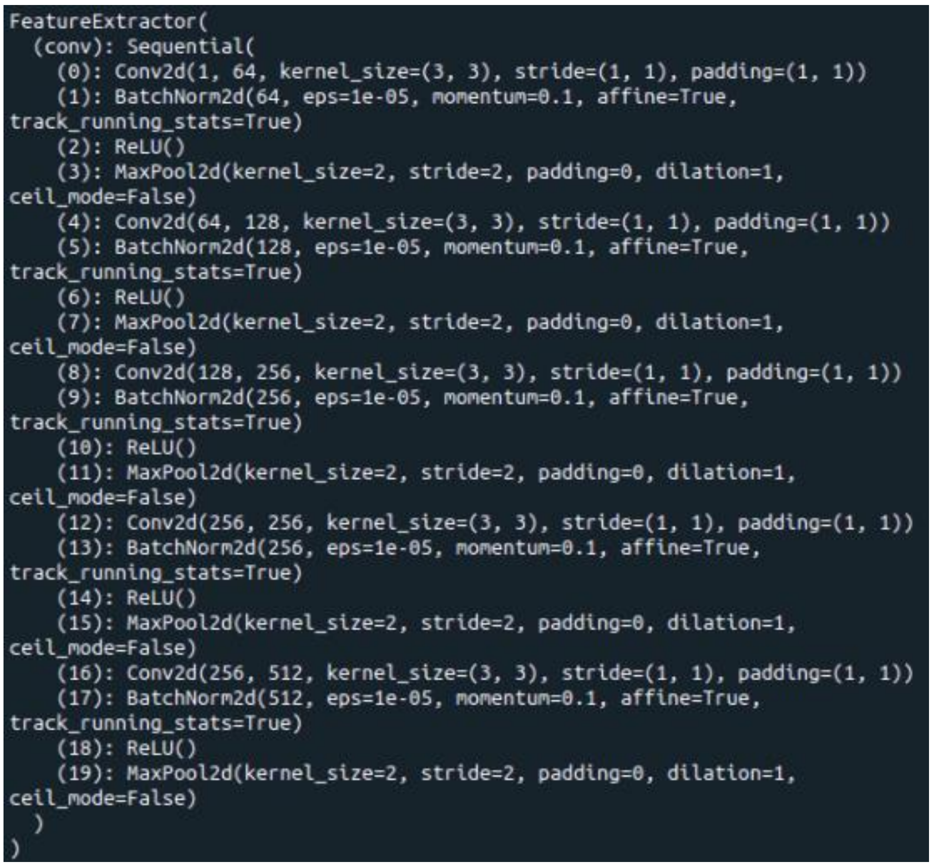




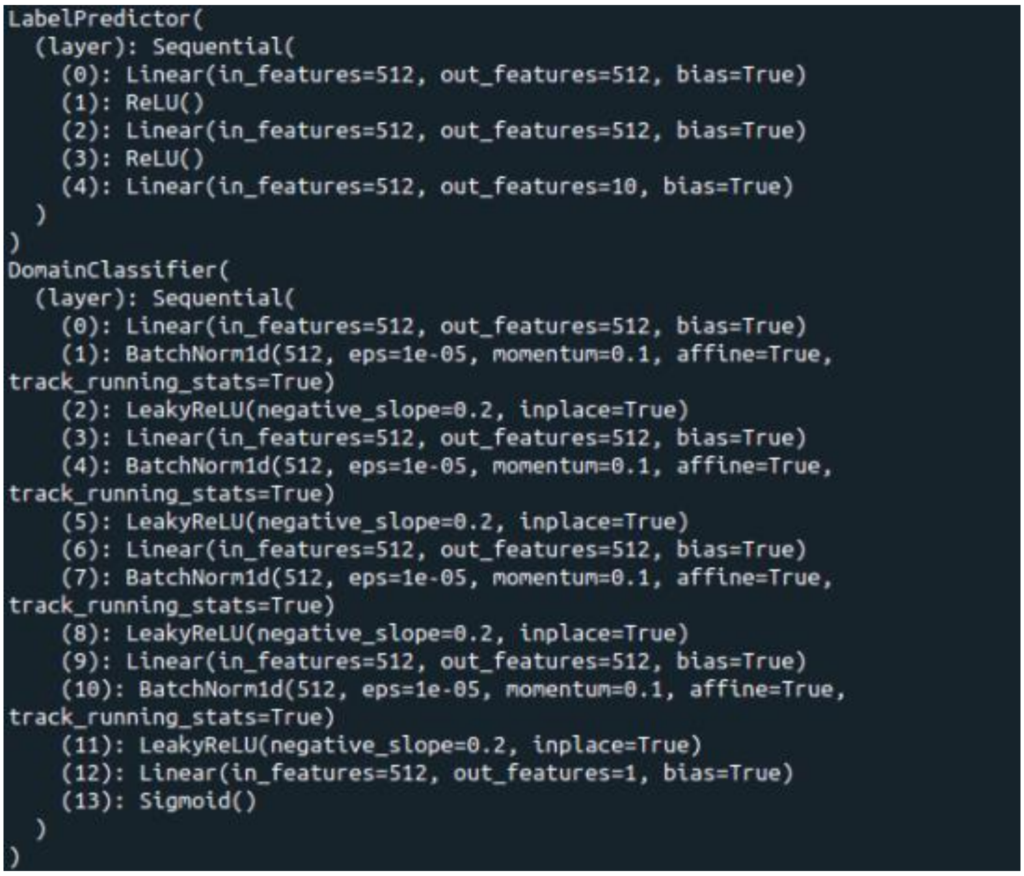




5. Describe the implementation details of your model and discuss what you've observed and learned from implementing DANN. (7%)

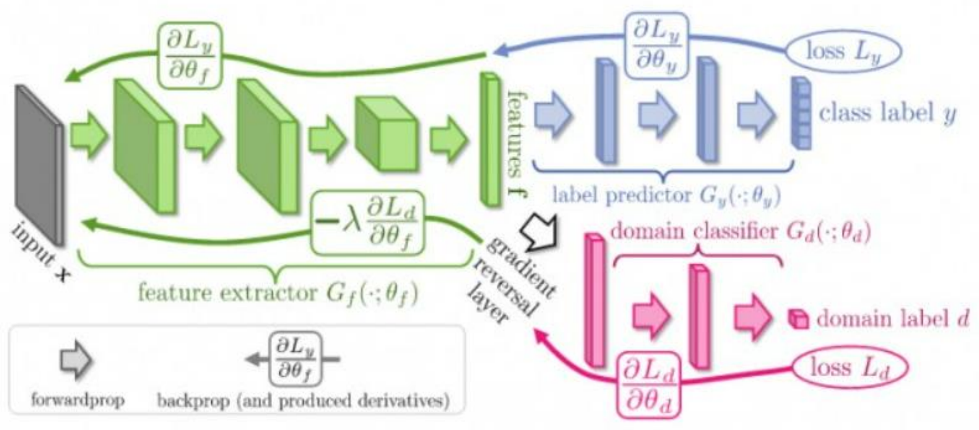


```
FeatureExtractor(
  (conv): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (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, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, 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(256, 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)
  )
)
```



```
LabelPredictor(
  (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=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, inplace=True)
    (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, inplace=True)
    (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, inplace=True)
    (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, inplace=True)
    (12): Linear(in_features=512, out_features=1, bias=True)
    (13): Sigmoid()
  )
)
```

Network Structure



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input x
gradient reversal layer
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Explain:

因為 DANN 需要分三個部份訓練，所以我在創 model 的部分我就創三個 model 分別為 feature extractor、label predictor、domain classifier 如上圖的結構一樣，而 feature extractor 因為是要做圖片特徵擷取的主幹，所以用的是 conv+maxpool 的結構，我發現如果不加入 maxpool 結果會很差而且訓練速度也很慢，而加入了 maxpool 之後結果就直接起飛了訓練速度也快了許多。在訓練部分則是用了大約 10~30 個 epoch 不等，而 batchsize 為 64，而且因為我發現其

中一組 dataset 的資料是 grayscale 而不是 RGB，所以我索性把全部的 dataset 都轉成 grayscale 這樣才能同時一起丟入訓練。

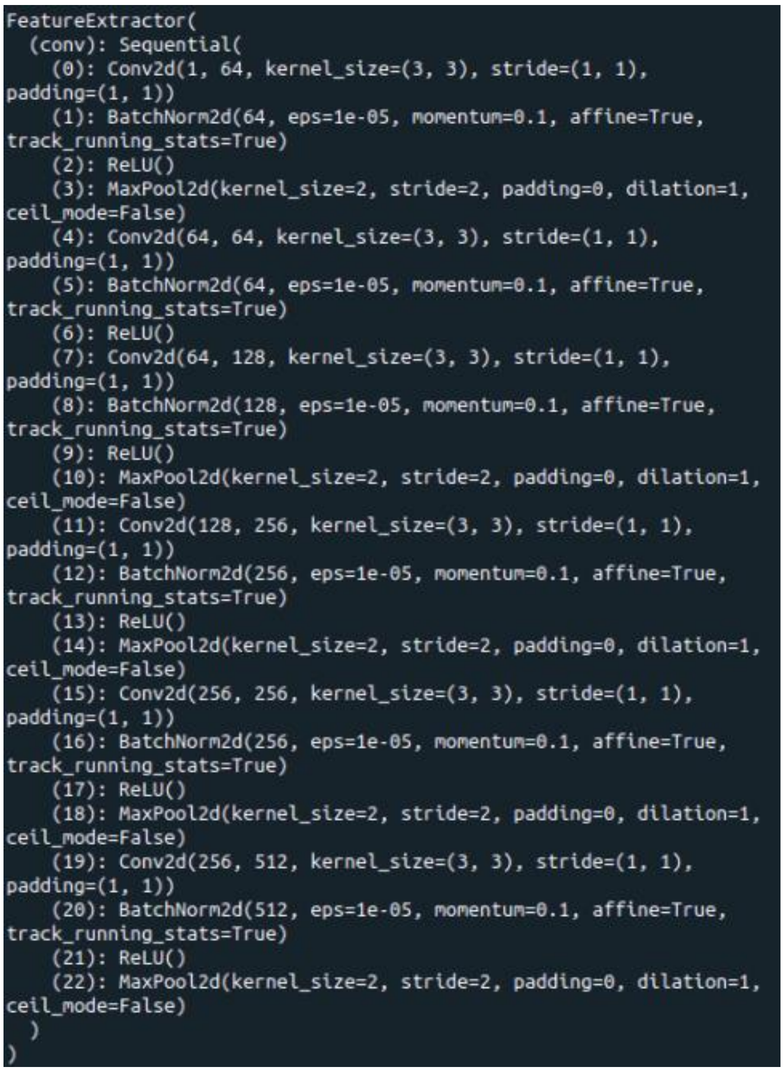
根據訓練結果我發現，如果是用 mnistm 來當 source 而 usps 來當 target 則會訓練得很好，因為 usps 只是單純的數字的圖，而 ministm 則是比較多彩多特色的數字，所以在分布上數字零到九依舊是能夠在 tsne 的圖上看到他能夠分類開來，且 usps 的資料點都能夠跟 mnistm 的資料點混和在一起了。而如果是用只有最基本的數字(黑白)數據集來當作 source 的話，train 在 svhn 的話結果會非常的差，連 50%都沒有，應該是因為 svhn 裡面多是出現不只單一的數字之外、還有不同的顏色、形狀，因此可以看到最後在 tsne 上，svhn 幾乎沒辦法好好地分出 10 群，而 svhn 和 usps 兩個 dataset 也沒有很好的 combine 在一起。

Bonus: Improved UDA model (6%)

1. Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation) (3%)

	MNIST-M -> USPS	SVHN -> MNIST-M	USPS -> SVHN
Original model	73.59%	43.25%	28.48%
Improved model	88.29%	58.96%	30.50%

2. Briefly describe implementation details of your model and discuss what you’ve observed and learned from implementing your improved UDA model. (3%)


```
FeatureExtractor(
  (conv): Sequential(
    (0): Conv2d(1, 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, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU()
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (11): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=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): ReLU()
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (20): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (21): ReLU()
    (22): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
)
```

基本上此題的基本架構我是沿用上題，我認為如果在現有架構下精進的話，重點應該是更改 feature extractor model 的部分，所以在我測試要怎麼增進模型的時候，我有嘗試過將 feature extractor model 裡面每一層結構加上 dropout 或是將 ReLU 換成 LeakyReLU，但是結果發現都不太好，但是我發現如果把一開始的 conv 層(input:1 output:64)後面在接上一層相同大小的 conv 層(input:64 output:64)訓練出來的效果有變好。

Reference:

https://arxiv.org/pdf/1511.06434.pdf

GitHub - soumith/ganhacks: starter from "How to Train a GAN?" at NIPS2016

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

https://github.com/clvrai/ACGAN-PyTorch

https://github.com/fungtion/DANN