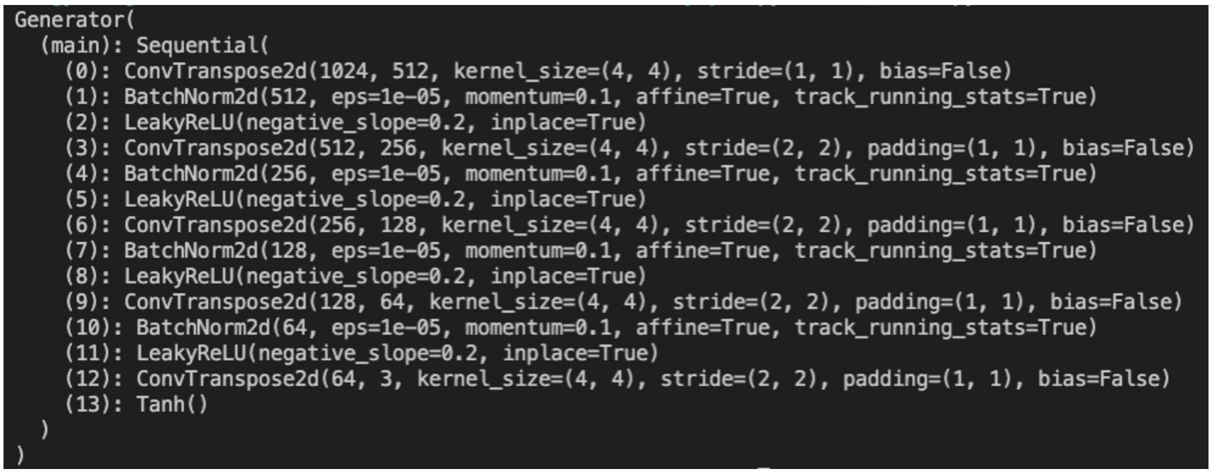
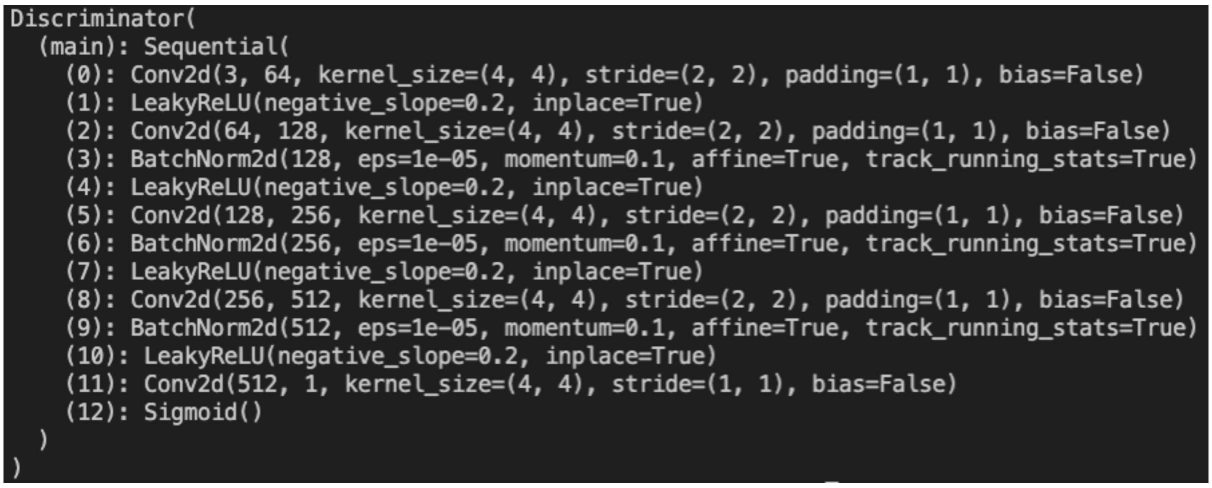
Deep Learning for Computer Vision NTU, Fall 2021 r09945021 洪怡庭
Homework #2

Problem1 GAN

1. Build your generator and discriminator from scratch and show your model architecture in your report (You can use “print(model)” in PyTorch directly). Then, train your model on the face dataset and describe implementation details. (Include but not limited to training epochs, learning rate schedule, data augmentation and optimizer) (5%)



```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(1024, 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): LeakyReLU(negative_slope=0.2, 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): LeakyReLU(negative_slope=0.2, 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): LeakyReLU(negative_slope=0.2, 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): LeakyReLU(negative_slope=0.2, inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

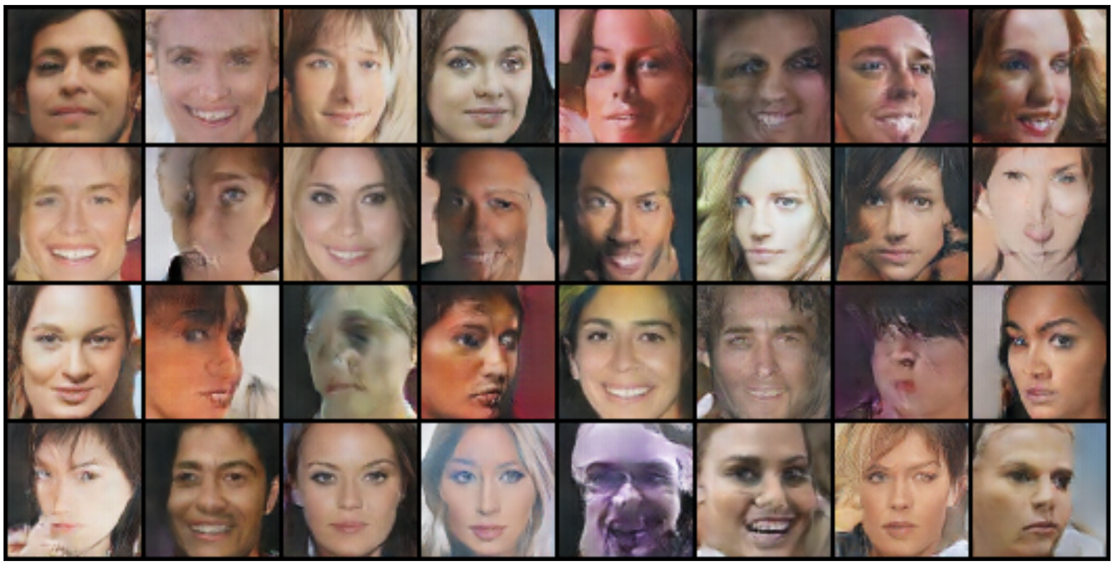


```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05,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

training epochs: 500, learning rate: 0.001, optimizer: Adam(betas=0.5, 0.999), augmentation: None
利用交叉訓練方法，分別訓練 Discriminator 和 Generator，learning rate 設為 0.001，optimizer 使用 Adam(betas=0.5, 0.999)，並以 BCELoss 來更新參數，輸入 Generator 的 latent vector size 設為 1024。

2. Now, we can use the Generator to randomly generate images. Please samples 1000 noise vectors from Normal distribution and input them into your Generator. Save the 1000 generated images in the assigned folder path for evaluation, and

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



3. Evaluate your 1000 generated images by implementing two metrics:

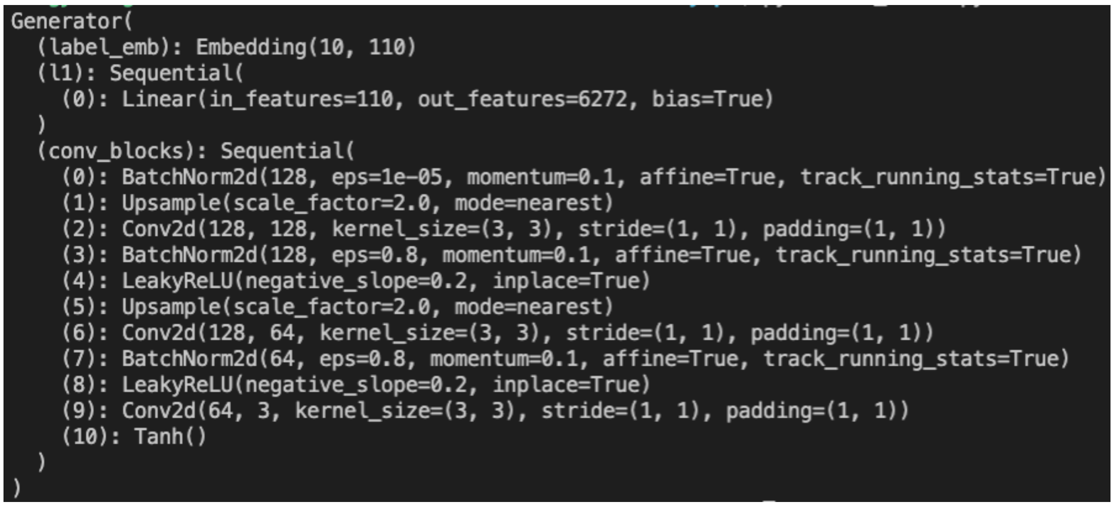
   (1) Fréchet inception distance (FID) : 27.95498

   (2) Inception score (IS): 2.10238

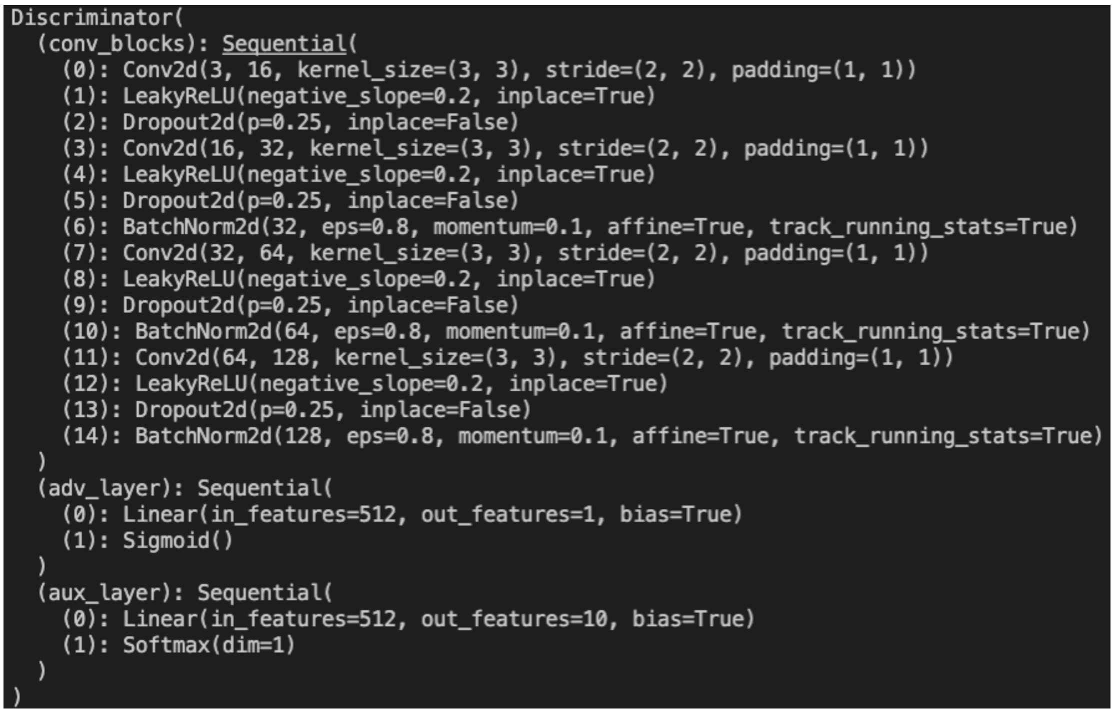
4. Discuss what you've observed and learned from implementing GAN. (5%)
   1. GAN 由 generator 和 discriminator 互相對抗來學習，generator 將輸入的 latent vector 生成為與真實圖片相似的影像使 discriminator 無法分別真假，而 discriminator 目的則為正確分別出真實影像與 generator 產生的影像
   2. 訓練的重點為 generator 跟 discriminator 需要交替訓練並各自更新權重，讓對抗的效果得以展現
   3. discriminator 的能力通常會贏過 generator 的能力，從 loss 可看出 discriminator 會小於 generator，表示對 model 來說分辨影像的真假是比產生假影像更容易的
   4. 設定 Adam optimizer 時將 beta 設定於(0.5 , 0.999)之間，可以讓 Momentum 有快慢的變化
   5. 由於 generator 架構最後一層為 Tanh，因此影像會介於-1~1 之間，需再調整範圍至 0~1

Problem2 ACGAN

1. Build your ACGAN model from scratch and show your model architecture in your report (You can use “print(model)” in PyTorch directly). Then, train your model on the mnistm dataset and describe implementation details. (e.g. How do you input the class labels into the model?) (10%)



```
Generator(
  (label_emb): Embedding(10, 110)
  (l1): Sequential(
    (0): Linear(in_features=110, out_features=6272, bias=True)
  )
  (conv_blocks): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): Upsample(scale_factor=2.0, mode=nearest)
    (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Upsample(scale_factor=2.0, mode=nearest)
    (6): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): Tanh()
  )
)
```



```
Discriminator(
  (conv_blocks): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout2d(p=0.25, inplace=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Dropout2d(p=0.25, inplace=False)
    (6): BatchNorm2d(32, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Dropout2d(p=0.25, inplace=False)
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (11): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): LeakyReLU(negative_slope=0.2, inplace=True)
    (13): Dropout2d(p=0.25, inplace=False)
    (14): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=512, out_features=10, bias=True)
    (1): Softmax(dim=1)
  )
)
```

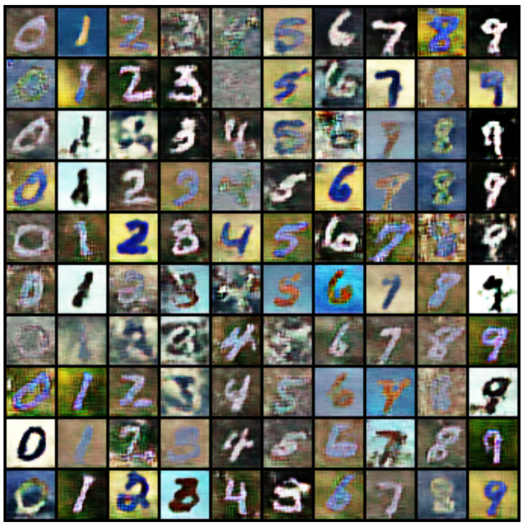
training epochs: 200, learning rate: 0.0002, optimizer: Adam, latent dimension: 100

一開始嘗試直接將 label 和 noise concatenate 在一起輸進 generator 中，但發現雖然圖片能畫出明顯的數字，但往往與給的 label 不同，可能是因為 label 數量遠少於 noise 數量，因此類別對 model 的影響很小。而後來改使用 embedding layer 輸入 label，利用相乘的方式結合 label 與 noise，改善了前述的問題，且此做法有效的減少 fully connected layer 的參數量，使得計算上

更有效率。

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%)
   accuracy = 0.95005

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. [see the below example] (5%)

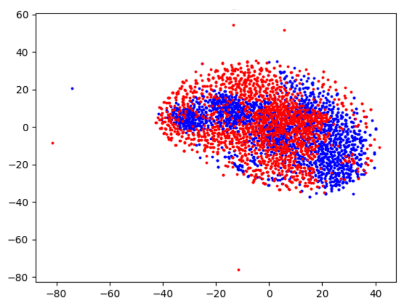
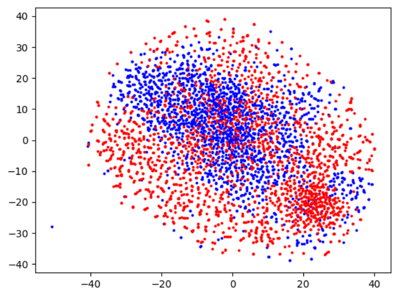
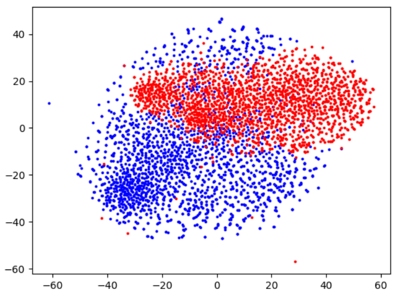
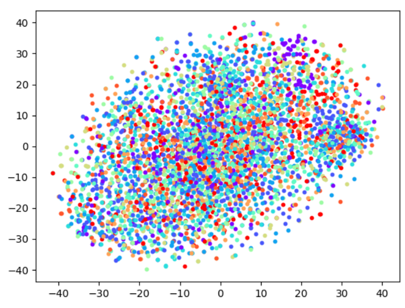
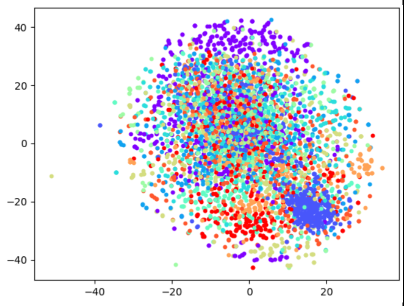
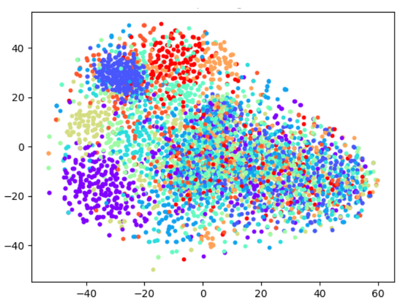


Problem 3 DANN

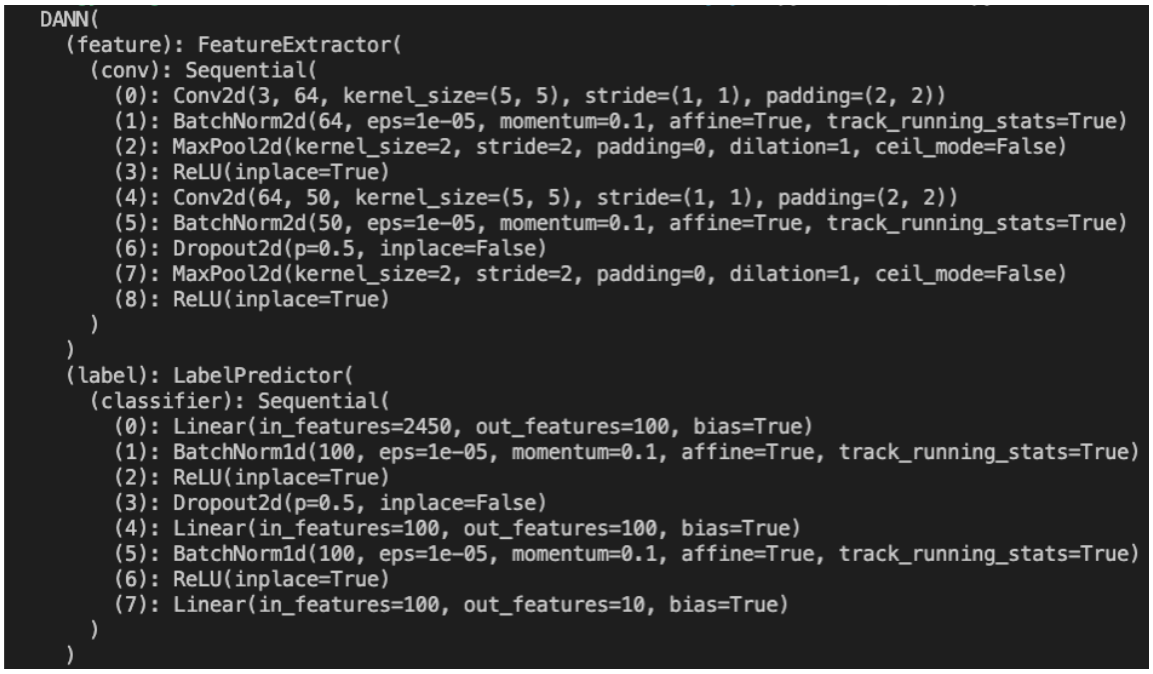
1. Compute the accuracy on target domain, while the model is trained on source domain only. (lower bound) (3%)
2. Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation) (4+9%)
3. Compute the accuracy on target domain, while the model is trained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

	SVHN→MNIST-M	MNIST-M→USPS	USPS→SVHN
Trained on source	0.4096	0.6721	0.1368
Adaptation (DANN/Improved)	0.4213	0.7892	0.2065
Trained on target	0.9400	0.9427	0.8412

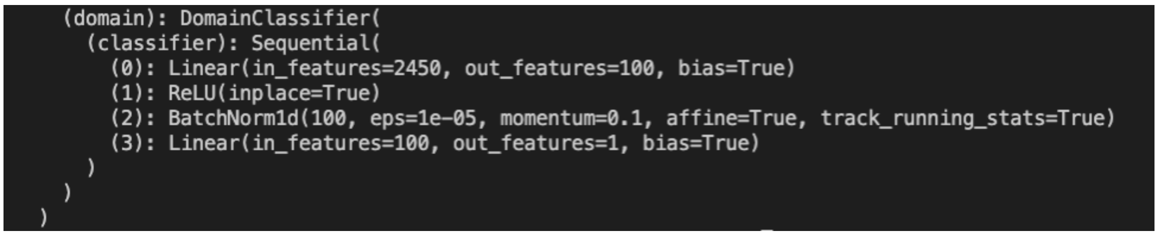
4. Visualize the latent space by mapping the *testing* images to 2D space with t-SNE and use different colors to indicate data of (*a*) different digit classes 0-9 and (*b*) different domains(source/target). (9%)

	SVHN→MNIST-M	MNIST-M→USPS	USPS→SVHN
Domain			
Classification			

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



```
DANN(
  (feature): FeatureExtractor(
    (conv): Sequential(
      (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): ReLU(inplace=True)
      (4): Conv2d(64, 50, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (5): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (6): Dropout2d(p=0.5, inplace=False)
      (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (8): ReLU(inplace=True)
    )
  )
  (label): LabelPredictor(
    (classifier): Sequential(
      (0): Linear(in_features=2450, out_features=100, bias=True)
      (1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Dropout2d(p=0.5, inplace=False)
      (4): Linear(in_features=100, out_features=100, bias=True)
      (5): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (6): ReLU(inplace=True)
      (7): Linear(in_features=100, out_features=10, bias=True)
    )
  )
```



```
    (domain): DomainClassifier(
      (classifier): Sequential(
        (0): Linear(in_features=2450, out_features=100, bias=True)
        (1): ReLU(inplace=True)
        (2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (3): Linear(in_features=100, out_features=1, bias=True)
      )
    )
  )
```

training epochs: 32, learning rate: 0.001 , augmentation: None, optimizer: Adam

1. DANN 架構會先經過 feature extractor（和一般的 CNN 相同），再藉由 label predictor 作為分類器分出 0-9，且還有另一個重要的 domain classifier，可以用來分辨是來自 source domain 還是 target domain，而為了混淆兩個 domain 的 feature，在訓練時會回傳一個負的 loss 來修正 feature extractor 的 weight，使得取出來的 feature 愈來愈不受 domain 的差異影響
2. digit classifier 和 domain classifier 分別使用 CrossEntropyLoss 及 BCELoss 來更新參數
3. MNIST-M→USPS 的準確度最高，USPS→SVHN 準確度最低，可能與 USPS 是黑白影像有關，當 source domain 為黑白時，用來測試彩色的 target domain 似乎比較難得到好的分數
4. 在 tsne 的結果上，SVHN→MNIST-M 的 classification 分群不明顯，但 domain 差異也較小，而準確度較低的 USPS→SVHN 反而較能看出幾個 cluster，然而可以看到在 domain 圖上 USPS→SVHN 是兩個不同 domain 分群最明顯的，表示並未成功的將 source domain 及 target domain 混淆，可能也是其準確度低的原因