Name：陳紘豪 Dep.：資工碩一 Student ID：R07922134

Problem 1: GAN (20%)

1. Describe the architecture & implementation details of your model. (5%)

GAN Training Configuration
- Adam optimizer (lr = 0.0002, beta1 = 0.5) for both G and D
- Batch size = 128
- Latent dimension = 100
- Latent vector is sampled from normal distribution with mean = 0, std = 1
- Epoch = 100
- D train 一次，G 也 train 一次
- Train G 時所使用的 input 個數為 batch size * 2
  - 原因：在 train D 時會先用 batch size 大小的真資料，再用 batch size 大小的假資料，總共用了 2 個 batch 來 train D
- 圖片有先 normalize 到 [-1, 1] 之間

Generator：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 gan-G-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1           [-1, 1024, 4, 4]       1,638,400
      BatchNorm2d-2           [-1, 1024, 4, 4]           2,048
             ReLU-3           [-1, 1024, 4, 4]               0
   ConvTranspose2d-4            [-1, 512, 8, 8]       8,388,608
      BatchNorm2d-5            [-1, 512, 8, 8]           1,024
             ReLU-6            [-1, 512, 8, 8]               0
   ConvTranspose2d-7          [-1, 256, 16, 16]       2,097,152
      BatchNorm2d-8          [-1, 256, 16, 16]             512
             ReLU-9          [-1, 256, 16, 16]               0
  ConvTranspose2d-10          [-1, 128, 32, 32]         524,288
     BatchNorm2d-11          [-1, 128, 32, 32]             256
            ReLU-12          [-1, 128, 32, 32]               0
  ConvTranspose2d-13            [-1, 3, 64, 64]           6,144
            Tanh-14            [-1, 3, 64, 64]               0
================================================================
Total params: 12,658,432
Trainable params: 12,658,432
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 5.81
Params size (MB): 48.29
Estimated Total Size (MB): 54.10
----------------------------------------------------------------
```

在 Generator 中，完全採用 DCGAN 作者所提出的架構，latent space 的維度為 100，之後對這 100 維度的向量過 5 層的 fractionally-strided convolutions，使得 input = (100, 1, 1) 變成 output = (3, 64, 64)。

Discriminator：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 gan-D-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 32, 32]           6,144
         LeakyReLU-2          [-1, 128, 32, 32]               0
            Conv2d-3          [-1, 256, 16, 16]         524,288
       BatchNorm2d-4          [-1, 256, 16, 16]             512
         LeakyReLU-5          [-1, 256, 16, 16]               0
            Conv2d-6            [-1, 512, 8, 8]       2,097,152
       BatchNorm2d-7            [-1, 512, 8, 8]           1,024
         LeakyReLU-8            [-1, 512, 8, 8]               0
            Conv2d-9           [-1, 1024, 4, 4]       8,388,608
      BatchNorm2d-10           [-1, 1024, 4, 4]           2,048
        LeakyReLU-11           [-1, 1024, 4, 4]               0
           Conv2d-12              [-1, 1, 1, 1]          16,384
          Sigmoid-13              [-1, 1, 1, 1]               0
================================================================
Total params: 11,036,160
Trainable params: 11,036,160
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 4.63
Params size (MB): 42.10
Estimated Total Size (MB): 46.77
----------------------------------------------------------------
```

在 Dicriminator 中，採用和 Generator 相反且對稱的架構，input size = (3, 64, 64)，經過 4 層的 convolutions 去提取特徵，最後 1 層 convolution 再把特徵壓縮成 (1, 1, 1) 作為分類的結果。

2. Plot 32 random images generated from your model. [fig1_2.jpg] (10%)

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

訓練 GAN 時很重要的一點就是要平衡 discriminator 和 generator，兩者之間的架構、輸入都會影響最後 GAN 的結果，剛開始 train 的時候 discriminator loss 很快就降到 0 了，發現此時 generator 完全學不到任何東西，產生一堆雜訊，後來把 discriminator 的架構稍做調整，使得其和 generator 互相對稱，就有得到很好的結果了。
還有一點就是 latent space，若沒有從 normal distribution 去取樣，而是從 uniform distribution 去取的話，會發現生出來的人臉每一張幾乎都長一模一樣，缺少了多樣性，改成用 normal distribution 去取就正常許多。

Problem 2: ACGAN (20%)

1. Describe the architecture & implementation details of your model. (5%)

ACGAN Training Configuration
- Adam optimizer (lr = 0.0002, beta1 = 0.5) for both G and D
- Batch size = 128
- Latent dimension = 101，多的一維為 (1: smile, 0: not smile)
- Latent vector is sampled from normal distribution with mean = 0, std = 1
- Epoch = 100
- D train 一次，G 也 train 一次
- 圖片有先 normalize 到 [-1, 1] 之間

Generator：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 acgan-G-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1           [-1, 1024, 4, 4]       1,654,784
       BatchNorm2d-2           [-1, 1024, 4, 4]           2,048
              ReLU-3           [-1, 1024, 4, 4]               0
   ConvTranspose2d-4            [-1, 512, 8, 8]       8,388,608
       BatchNorm2d-5            [-1, 512, 8, 8]           1,024
              ReLU-6            [-1, 512, 8, 8]               0
   ConvTranspose2d-7          [-1, 256, 16, 16]       2,097,152
       BatchNorm2d-8          [-1, 256, 16, 16]             512
              ReLU-9          [-1, 256, 16, 16]               0
  ConvTranspose2d-10          [-1, 128, 32, 32]         524,288
      BatchNorm2d-11          [-1, 128, 32, 32]             256
             ReLU-12          [-1, 128, 32, 32]               0
  ConvTranspose2d-13            [-1, 3, 64, 64]           6,144
             Tanh-14            [-1, 3, 64, 64]               0
================================================================
Total params: 12,674,816
Trainable params: 12,674,816
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 5.81
Params size (MB): 48.35
Estimated Total Size (MB): 54.16
----------------------------------------------------------------
```

在 Generator 的部分，基本架構完全和 GAN 中的 Generator 一模一樣，唯一不一樣的只有 input 變成 101 維，但是這不影響 model 的架構，仍然是過 5 層的 fractionally-strided convolutions，最終 output 出 (3, 64, 64) 的一張圖片。

Discriminator：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 acgan-D-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 32, 32]           6,144
         LeakyReLU-2          [-1, 128, 32, 32]               0
            Conv2d-3          [-1, 256, 16, 16]         524,288
       BatchNorm2d-4          [-1, 256, 16, 16]             512
         LeakyReLU-5          [-1, 256, 16, 16]               0
            Conv2d-6            [-1, 512, 8, 8]       2,097,152
       BatchNorm2d-7            [-1, 512, 8, 8]           1,024
         LeakyReLU-8            [-1, 512, 8, 8]               0
            Conv2d-9           [-1, 1024, 4, 4]       8,388,608
      BatchNorm2d-10           [-1, 1024, 4, 4]           2,048
        LeakyReLU-11           [-1, 1024, 4, 4]               0
           Conv2d-12              [-1, 1, 1, 1]          16,384
          Sigmoid-13              [-1, 1, 1, 1]               0
           Conv2d-14              [-1, 1, 1, 1]          16,384
          Sigmoid-15              [-1, 1, 1, 1]               0
================================================================
Total params: 11,052,544
Trainable params: 11,052,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 4.63
Params size (MB): 42.16
Estimated Total Size (MB): 46.83
----------------------------------------------------------------
```

Discriminator 前半段和 GAN 的 Discriminator 一樣，利用 4 層的 convolutions 來提取特徵值，而這特徵值最後會接上兩種 convolutions 產生兩種分類的結果，一種就是原本 GAN 中的結果，用來分辨 real/fake data，另一種是拿來分辨圖片具有某種 attribute，我的例子就是分辨圖片笑或不笑。

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. [fig1_2.jpg] (10%)

Smile

Not Smile

3. Discuss what you've observed and learned from implementing ACGAN (5%)

ACGAN 也是 GAN 的一種，所以 generator/discriminator 架構對稱、latent space 從 normal distribution 取樣，這些也都是一樣重要的。
還有一點，ACGAN 中圖片的 attribute 選用也是很重要的，一開始選了一些根本看不出來有或沒有的 attribute（例如：Heavy makeup)，產生的圖片完全看不出什麼差異，但改成用 smile 後就可以看出兩者之間具有明顯的差別。

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) (3+7%)
3. Compute the **accuracy** on target domain, while the model is trained on target domain only. (upper bound) (3%)

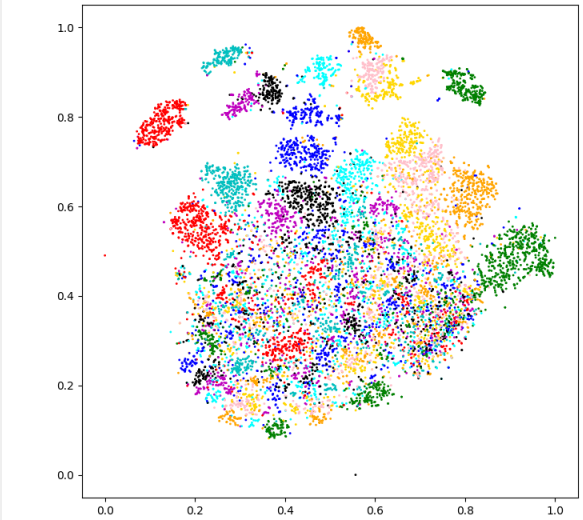
Accuracy	USPS -> MNIST-M	MNIST-M -> SVHN	SVHN -> USPS
Trained on source	32.85% (3285/10000)	36.43% (9484/26032)	55.85% (1121/2007)
Adaptation (DANN)	41.84% (4184/10000)	45.23% (11775/26032)	48.77% (979/2007)
Trained on target	97.45% (9745/10000)	92.07% (23968/26032)	97.01% (1947/2007)

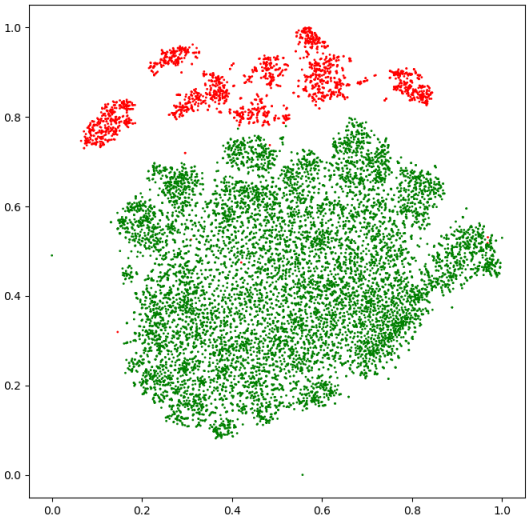
所有的 DANN 中，只有 **SVHN -> USPS** 的 Accuracy 比自己的 lower bound 還低，可能是因為這項 task 的 source domain 的資料量遠大於 target domain（其他兩個 tasks 都是 target domain 資料量較多），以致於在 adaptation 時無法讓 target domain features 好好融入 source domain features 中，不能把 domain 特性給消除，所以導致結果會比 lower bound 還要低。由 4. 的圖中也可以看到 **SVHN -> USPS** 在經過 adaptation 後 source domain 自己本身的分群也壞掉了，同樣的顏色會出現在 source domain 中不同的地方。

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%)

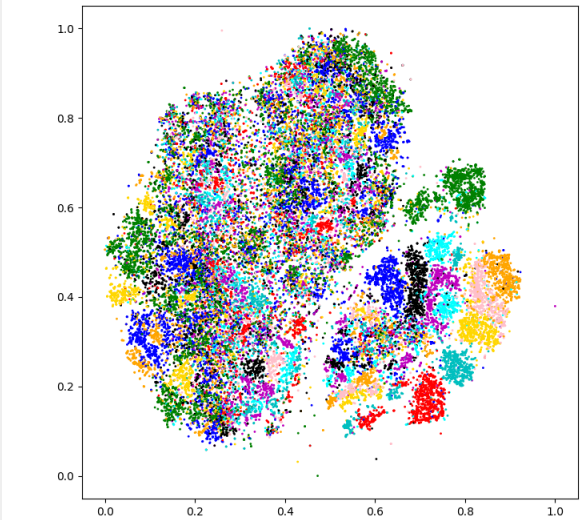
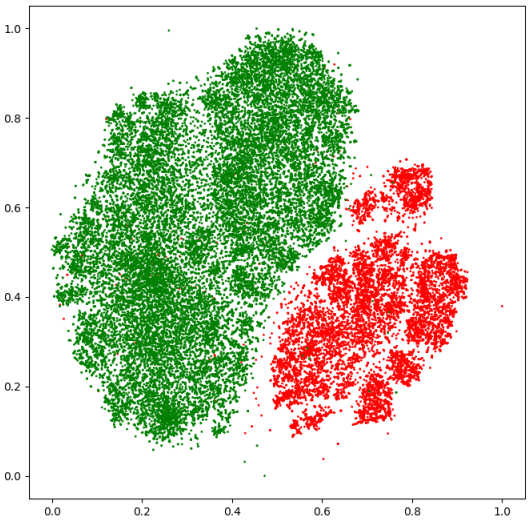
USPS -> MNIST-M

Classes	Domains（紅色：Source, 綠色：Target）



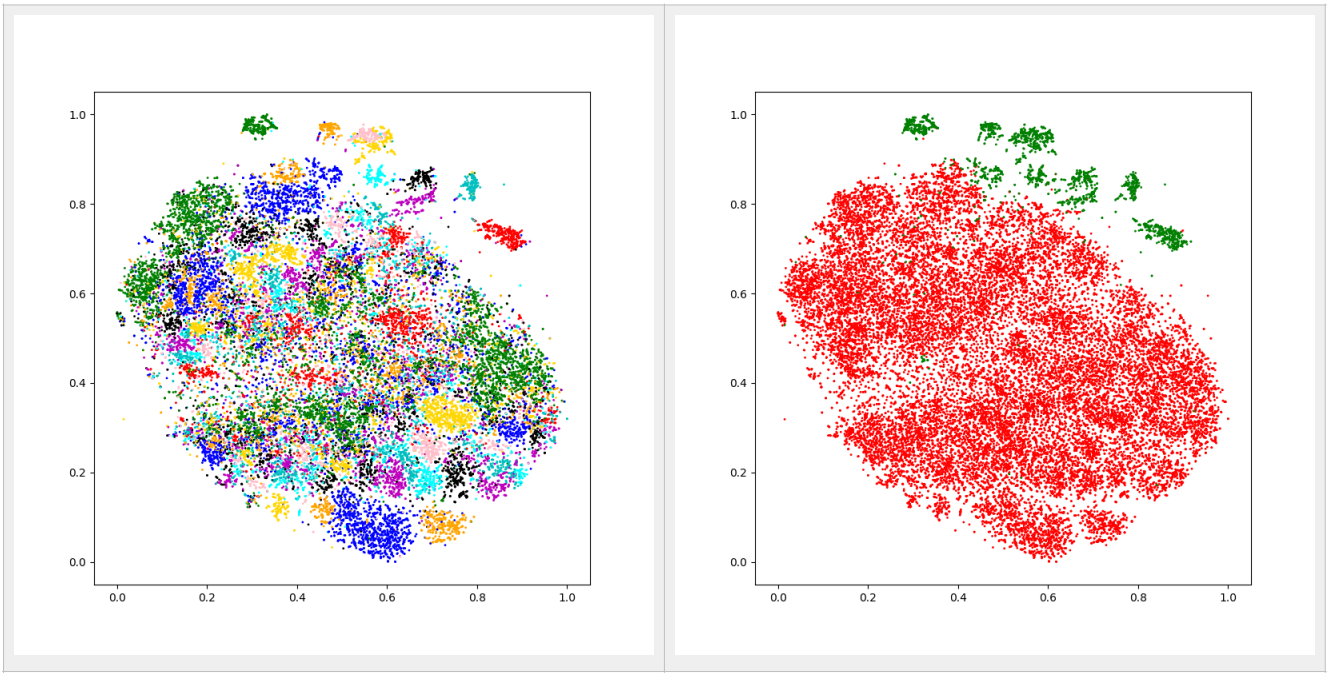


MNISTM -> SVHN

Classes	Domains（紅色：Source, 綠色：Target）
	

SVHN -> USPS

Classes	Domains（紅色：Source, 綠色：Target）



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

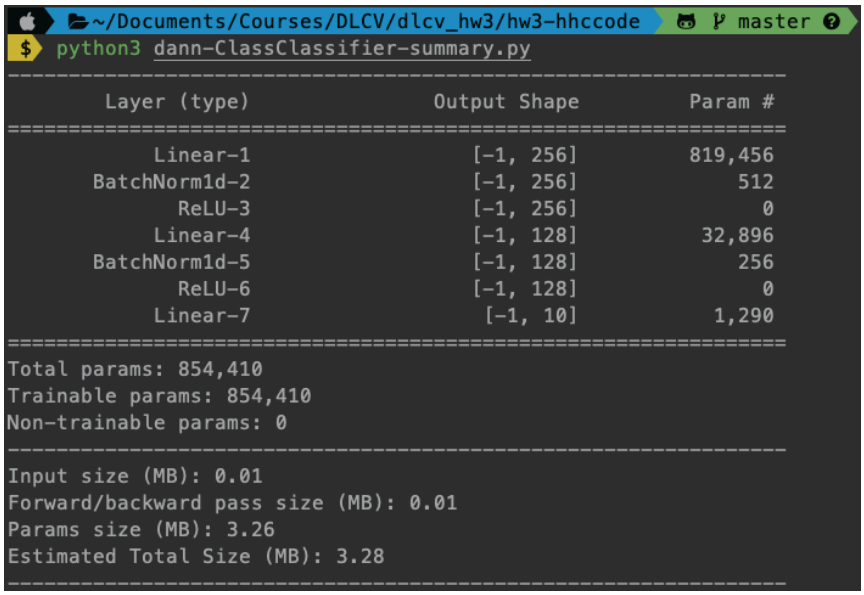
DANN Training Configuration
- Adam optimizer (lr = 0.0001)
- Batch size = 64
- 圖片有先 normalize 到 [0, 1] 之間
- Gradient reversal 在 backward 時有乘上一個 lambda (根據 paper 所設定)

Feature Extractor：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 dann-FeatureExtractor-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 26, 26]           4,864
       BatchNorm2d-2           [-1, 64, 26, 26]             128
              ReLU-3           [-1, 64, 26, 26]               0
         MaxPool2d-4           [-1, 64, 13, 13]               0
         Dropout2d-5           [-1, 64, 13, 13]               0
            Conv2d-6          [-1, 128, 11, 11]         204,928
       BatchNorm2d-7          [-1, 128, 11, 11]             256
              ReLU-8          [-1, 128, 11, 11]               0
         MaxPool2d-9            [-1, 128, 5, 5]               0
        Dropout2d-10            [-1, 128, 5, 5]               0
================================================================
Total params: 210,176
Trainable params: 210,176
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.56
Params size (MB): 0.80
Estimated Total Size (MB): 2.37
----------------------------------------------------------------
```

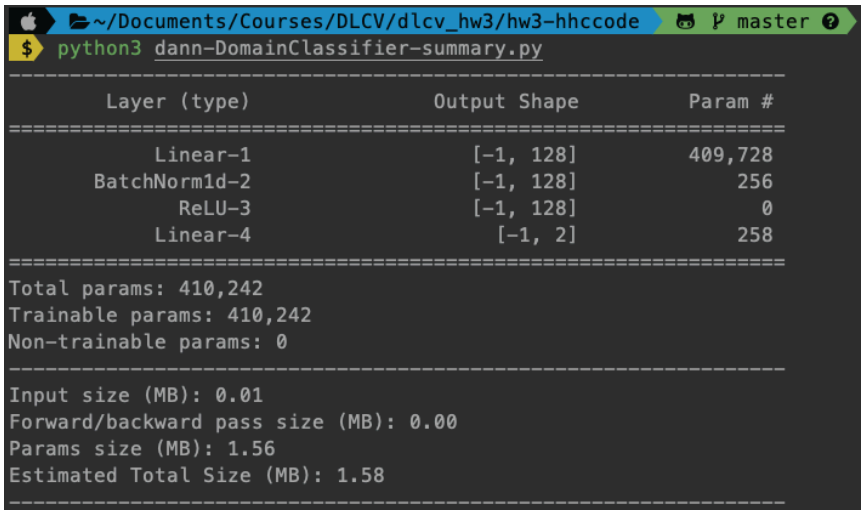
Feature extractor 就是簡單的 CNN 架構所組成，利用兩大層的 convolutions 把 Input 從 (3, 28, 28) 的 rgb 圖轉成 3200 維度的 features，並加入 Batchnorm、Dropout layer 防止 overfitting。

Class Classifier：



Class classifier 的部分使用三大層的 fully connected，來對於 input 為 3200 維的 features 的預測出 features 的 class，並只使用 Batchnorm layer 來避免 overfitting。

Domain Classifier：



和 class classifier 類似的結構，不過採用比較少的 linear layer，對於 input 為 3200 維的 features 來預測 features 是哪個 domain。

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

在實作 DANN 時，feature extractor、class classifier、domain classifier 的設計不能讓彼此能力相差太大，剛開始在實作的時候我的 domain classifier 太弱了，feature extractor 很輕易地就可以騙過 domain classifier，以至於完全無法移除掉 domain 特性就結束了，後來把 domain classifier 架構稍微改一下整體 performance 就有上升。
再來其實如果 source/target domain 之間的差異本身就很大，那就無法單靠 DANN 就完美的消除 domain 特性來大量提升正確率，這次作業的 task 都無法在 adaptation 後達到 80%、90% 以上的正確率，反觀許多 paper 所實驗的對象 source/target domain 本來就沒差異的如此巨大（e.g. 原始 mnist -> usps），所以使用 UDA model 都可以得到不錯的效果。

Problem 4: Improved UDA model (35%)

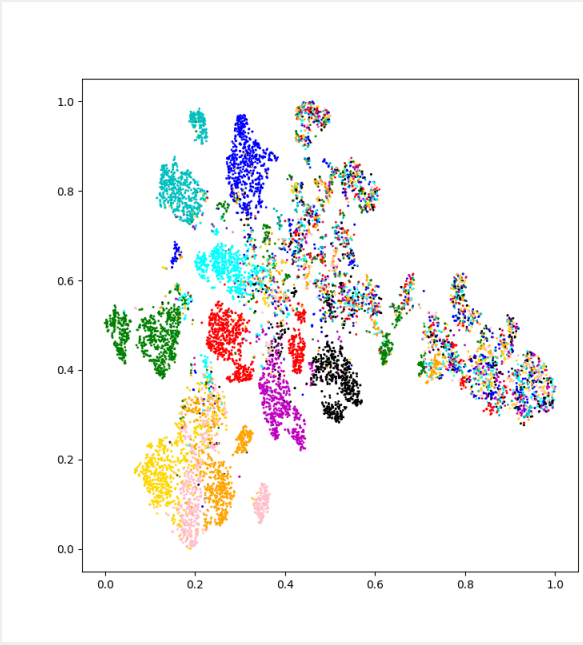
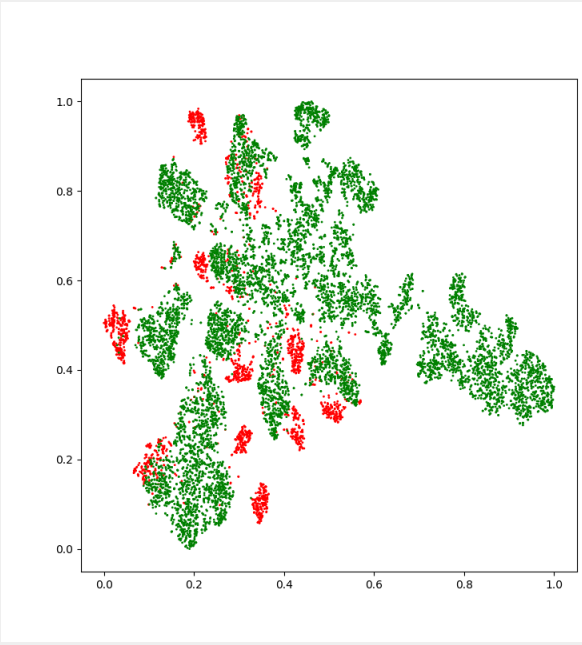
1. Compute the **accuracy** on target domain, while the model is trained on source **and** target domain. (domain adaptation) (6+10%)

Accuracy	USPS -> MNIST-M	MNIST-M -> SVHN	SVHN -> USPS
Adaptation (DANN)	41.84% (4184/10000)	45.23% (11775/26032)	48.77% (979/2007)
Generate to Adapt (GTA)	47.36% (4736/10000)	-	76.48% (1535/2007)
Adversarial Discriminative Domain Adaptation (ADDA)	-	48.61% (12656/2007)	-

在這邊我實作了兩種不同的 model，因為 GTA 在 task2 上的表現無法超過當初 DANN 的 45%，所以另外實作了 ADDA 來跑看看 task2。

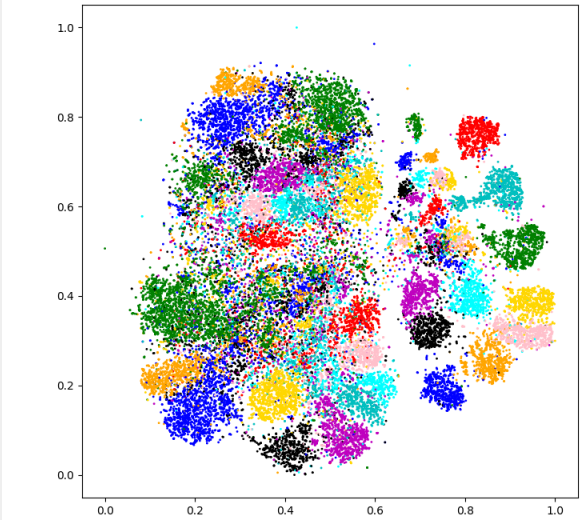
2. Visualize the latent space by mapping the testing images to 2D space (with t-SNE) and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (source/target). (6%)

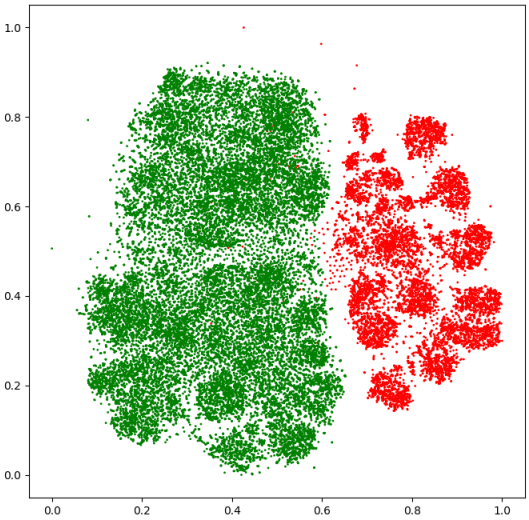
USPS -> MNIST-M

Classes	Domains（紅色：Source, 綠色：Target）
	

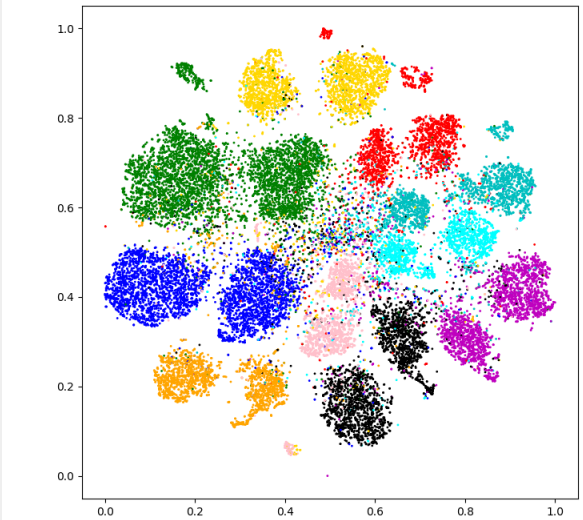
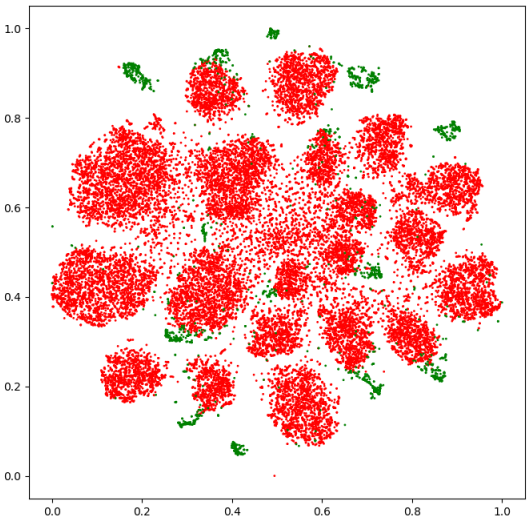
MNISTM -> SVHN

Classes	Domains（紅色：Source, 綠色：Target）





SVHN -> USPS

Classes	Domains（紅色：Source, 綠色：Target）
	

3. Describe the architecture & implementation detail of your model. (6%)
   A. GTA－USPS -> MNISTM-M and SVHN -> USPS

GTA Training Configuration
- Adam optimizer (lr = 0.0005, beta1 = 0.8) for F、C、G、D
- Batch size = 64
- 圖片有先 normalize 到 [-1, 1] 之間
- 圖片 Resize 成 32x32
- Generator 的 input dimension 為 128+512+10，其中 128 維是 F 的 output，512 維是從 N(0,1) sample 出來的 latent vector，10 維是 label 的 one-hot vector
- Training 順序為：D -> G -> C -> F

Discriminator：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 gta-Discriminator-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,792
       BatchNorm2d-2          [-1, 64, 32, 32]             128
         LeakyReLU-3          [-1, 64, 32, 32]               0
         MaxPool2d-4          [-1, 64, 16, 16]               0
            Conv2d-5         [-1, 128, 16, 16]          73,856
       BatchNorm2d-6         [-1, 128, 16, 16]             256
         LeakyReLU-7         [-1, 128, 16, 16]               0
         MaxPool2d-8           [-1, 128, 8, 8]               0
            Conv2d-9           [-1, 256, 8, 8]         295,168
      BatchNorm2d-10           [-1, 256, 8, 8]             512
        LeakyReLU-11           [-1, 256, 8, 8]               0
        MaxPool2d-12           [-1, 256, 4, 4]               0
           Conv2d-13           [-1, 128, 4, 4]         295,040
      BatchNorm2d-14           [-1, 128, 4, 4]             256
        LeakyReLU-15           [-1, 128, 4, 4]               0
        MaxPool2d-16           [-1, 128, 1, 1]               0
           Linear-17                 [-1, 1]             129
          Sigmoid-18                 [-1, 1]               0
           Linear-19                [-1, 10]           1,290
================================================================
Total params: 668,427
Trainable params: 668,427
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.89
Params size (MB): 2.55
Estimated Total Size (MB): 5.45
----------------------------------------------------------------
```

Train discriminator 時，會使用 source images、source images 取完 features 後經 generator 產生的 fake source images、target images 取完 features 後經 generator 產生的 fake target images，用三種不同來源的 images 來讓 discriminator 來分辨真假，這邊採用的 discriminator 架構是類似於 ACGAN 的 discriminator，會有兩種 outputs，一種是數字的 prediction，另外一種是 real/fake prediction。

Generator：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 gta-Generator-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1          [-1, 512, 2, 2]       1,331,200
       BatchNorm2d-2          [-1, 512, 2, 2]           1,024
              ReLU-3          [-1, 512, 2, 2]               0
   ConvTranspose2d-4          [-1, 256, 4, 4]       2,097,152
       BatchNorm2d-5          [-1, 256, 4, 4]             512
              ReLU-6          [-1, 256, 4, 4]               0
   ConvTranspose2d-7          [-1, 128, 8, 8]         524,288
       BatchNorm2d-8          [-1, 128, 8, 8]             256
              ReLU-9          [-1, 128, 8, 8]               0
  ConvTranspose2d-10         [-1, 64, 16, 16]         131,072
      BatchNorm2d-11         [-1, 64, 16, 16]             128
             ReLU-12         [-1, 64, 16, 16]               0
  ConvTranspose2d-13          [-1, 3, 32, 32]           3,072
             Tanh-14          [-1, 3, 32, 32]               0
================================================================
Total params: 4,088,704
Trainable params: 4,088,704
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.75
Params size (MB): 15.60
Estimated Total Size (MB): 16.35
----------------------------------------------------------------
```

Train generator 時，會使用 source images 取完 features 後經 generator 產生的 fake source images，丟給 discriminator 後去算 loss 來更新參數，並且希望產生的 fake images 可以讓 discriminator 在數字可以分辨正確，real/fake 分辨錯誤。

Classifier：

```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 gta-Classifier-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 128]          16,512
              ReLU-2                  [-1, 128]               0
            Linear-3                   [-1, 10]           1,290
================================================================
Total params: 17,802
Trainable params: 17,802
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.07
Estimated Total Size (MB): 0.07
----------------------------------------------------------------
```

Classifier 吃的 input 是 source images 取完 features 後的 128 維 features，之後 output 出數字的預測，用簡單的 fully-connected 架構來實作。

Feature extractor：

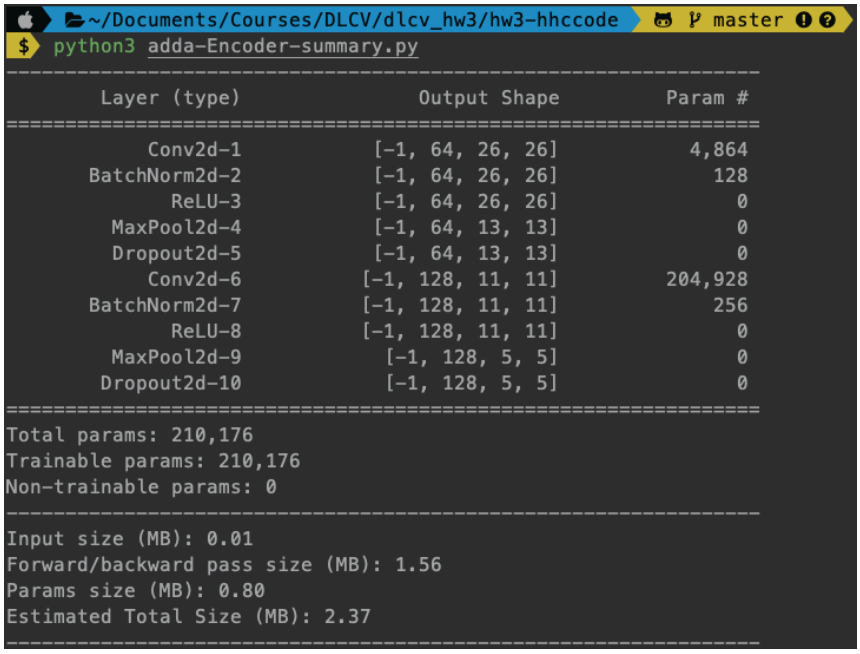
```
~/Documents/Courses/DLCV/dlcv_hw3/hw3-hhccode  master
$ python3 gta-FeatureExtractor-summary.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 28, 28]           4,864
              ReLU-2          [-1, 64, 28, 28]               0
         MaxPool2d-3          [-1, 64, 14, 14]               0
            Conv2d-4          [-1, 64, 10, 10]         102,464
              ReLU-5          [-1, 64, 10, 10]               0
         MaxPool2d-6            [-1, 64, 5, 5]               0
            Conv2d-7           [-1, 128, 1, 1]         204,928
              ReLU-8           [-1, 128, 1, 1]               0
================================================================
Total params: 312,256
Trainable params: 312,256
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.97
Params size (MB): 1.19
Estimated Total Size (MB): 2.18
----------------------------------------------------------------
```

Feature extractor 的架構由三大層 convolutions 所組成，希望可以產生出讓 classifier 正確預測數字的 features，且該 features 經由 generator 所產生的照片會混淆 discriminator 讓它分不清真假。

B. ADDA－MNISTM-M -> SVHN

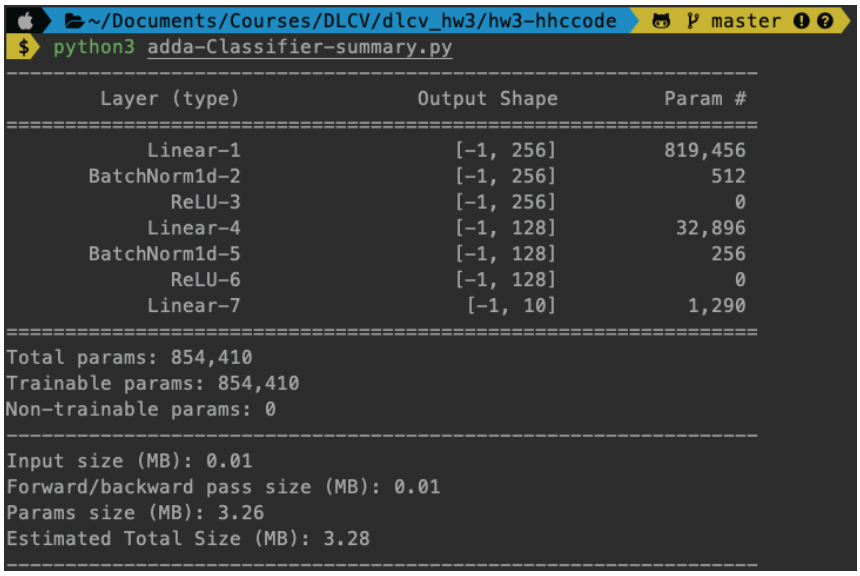
ADDA Training Configuration
- Adam optimizer (lr = 0.0001, beta1 = 0.5)
- Batch size = 64
- 圖片有先 normalize 到 [0, 1] 之間
- 先利用 Encoder+Classifier 架構，把 source domain pre-train 在上面
- 利用 Pre-train source encoder，加上 target encoder、discriminator 來實作 domain adaptation
- 最後利用 Target encoder+Pre-train source classifier 來驗證 adaptation 的效果

Encoder：



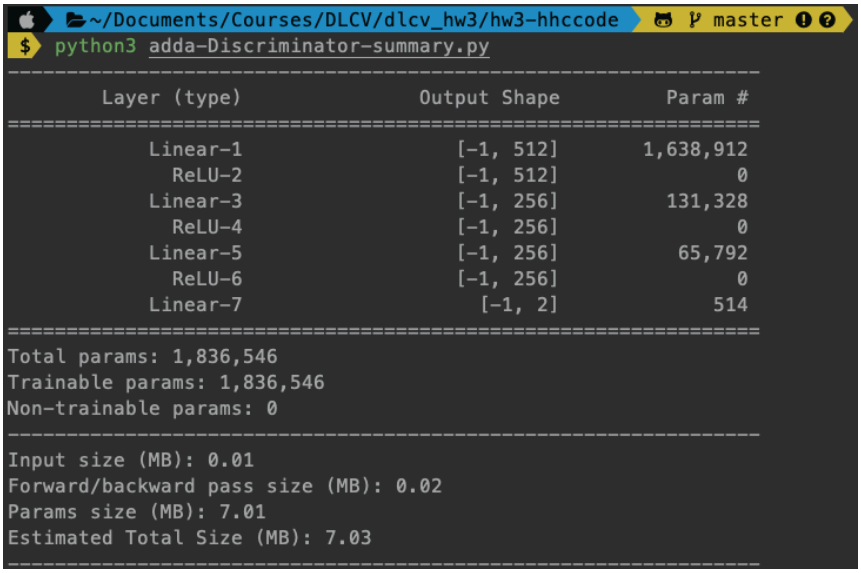
Encoder 的架構用兩大層的 convolutions 所組成，input 為 (3, 28, 28) 的圖片，經過 encoder 後會得到長度為 3200 維的 features，在 train target encoder 的時候會利用 discriminator 的輸出來 backward 達到 adversarial 的效果，不斷地對抗使得 target encoder 把 features 中 domain 的特性給消除。

Classifier：



Classifier 的架構為簡單的 feed-forward network，給定長度為 3200 維度的 features 會輸出對於數字的預測值。

Discriminator：



Discriminator 的功能就是要去分辨出 features 的來源是哪個 domain，藉由不斷的和 target encoder 的對抗來讓 domain 特性消除，這邊採用的架構也是 feed-forward network。

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

在實作這些 UDA model 的時候，學到了一定要先按照作者當初 paper 所提出的架構來實作，不要自己亂改一堆有的沒的，我剛開始在實作 GTA 的時候 Discriminator、Generator 的架構跟作者的有一些些的不一樣（例如 Discriminator 不使用 maxpooling 來 downsample 而是在 convolution 時就透過設定 stride、padding 來 downsample），在沒有任何 bug 的狀況下最後跑出來的結果就是完全爛掉，後來改成當初作者提出的架構就可以順利的 train 起來的。

和DANN一樣，其他 UDA model 也是不見得會在所有的 task 上都可以 work 的，例如 GTA 在 MNIST-M -> SVHN 就是 train 不起來，accuracy 只會有 20% 左右，因此不是所有的 model 都可以一魚多吃，還是得要先觀察 source/target domain 的特性來提出適合的 model才行。