

深度學習於電腦視覺 作業三  
資工所碩一 r07922103 李俊賢

Problem 1 : GAN

1. Describe the architecture & implementation details of your model.

```
WGAN_G(  
  (block1): 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)  
  )  
  (block2): Sequential(  
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block3): Sequential(  
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block4): Sequential(  
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block5): Sequential(  
    (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): Tanh()  
  )  
)
```

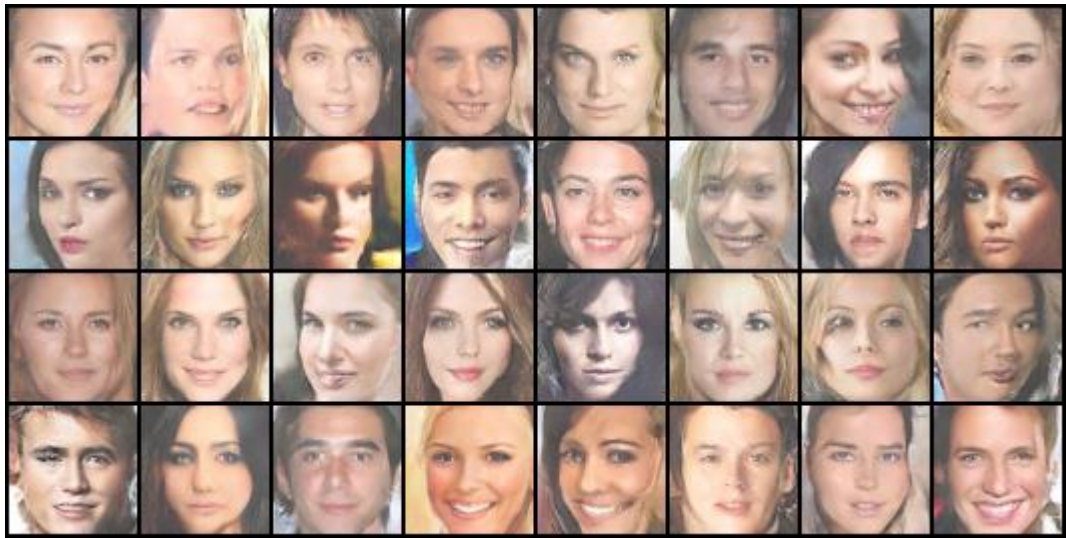
```
WGAN_D(  
  (block1): Sequential(  
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): LeakyReLU(negative_slope=0.2, inplace)  
  )  
  (block2): Sequential(  
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2, inplace)  
  )  
  (block3): Sequential(  
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2, inplace)  
  )  
  (block4): Sequential(  
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(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)  
  )  
  (block5): Sequential(  
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)  
  )  
)
```

在第一題中我實作了以 convolution 架構的 WGAN。

WGAN\_G 代表 WGAN 的 Generator，負責把輸入的(batch\_size, 100, 1, 1)轉化成(batch\_size, 3, 64, 64)的一堆假圖片。WGAN\_G 的實作細節如上圖所示，利用 convTranspose2d 把 100 channel 變成 512 channel 再從 512 -> 256 -> 128 -> 64 -> 3 channel。

WGAN\_D 代表 WGAN 的 Discriminator，負責把輸入的假圖片(batch\_size, 3, 64, 64)轉化成(batch\_size, 1)的格式，其中 1 是用來判斷圖片的真假。實作細節如上圖，利用 conv 從 3 -> 64 -> 128 -> 256 -> 512 -> 1 channel。

2. Plot 32 random images generated from your model.



3. Discuss what you've observed and learned from implementing GAN.

我認為我學到三件事。

一：GAN 真是很難 Train，他的 loss 很容易不穩定起起伏伏，也很容易 mode-collapse，其 loss 數值也沒有實質上的距離意義。因為他太不穩定，所以我採用比較好 train 的 WGAN 來實作，WGAN 的精神就是把 KL divergence 換成 Wasserstein 距離，可以更穩定而且能使 loss 的數值有意義。所以整體做起來會更有成就感也更有實作的感覺，也因為數據都是有意義的，也可以從中看到他們學習的變化。

二：WGAN 的架構其實論文中並沒有限制。主要分成兩大類，第一類是利用 fully connected layer 去完成整個架構，第二類是用 convolution 的做法來完成架構。而其中 fully connected 架構所做出來的圖片整體看起來會比較 smooth，也就是人的臉型以及臉型旁邊的圖像會變得非常不清楚且模糊，但反之 train 的速度會比較快。第二類的 convolution 架構會使得整個圖片的輪廓更為清晰清楚，也更有真人的感覺，但 train 的時間大概會多 1.3~1.5 倍左右。所以在圖片真偽的考慮上，我最後選擇用 convolution 的方法去拿到更好的圖片品質。

三：WGAN 還有一個進階的做法，叫 WGAN-GP (gradient penalty)。他的想法是來自於 WGAN 實作中有利利用 Clamp 去消除更新過度的數值。而這步驟其實會讓一些發散的數值無法學到新的進步。所以 GP 就是利用 random 生成一個真假圖片的 interpolation 圖片來補償其中資訊的損失，也就是創造更多 data 去訓練。但這一過程我實作之後發現，所需的 train 時間會比原本的 WGAN 多上 2 倍多，是非常耗時的，要 train 到好大概要花兩天的時間，所以在時間考量下我捨棄了 GP 的做法，但 GP 的圖片品質會比 WGAN 更有輪廓感，看起來也更真實，真是可惜啊！

## Problem 2 : ACGAN

### 1. Describe the architecture & implementation details of your model.

```
ACGAN_G(  
  (block1): Sequential(  
    (0): Linear(in_features=101, out_features=8192, bias=True)  
    (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block2): Sequential(  
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block3): Sequential(  
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block4): Sequential(  
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): ReLU(inplace)  
  )  
  (block5): Sequential(  
    (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): Tanh()  
  )  
)
```

```
ACGAN_D(  
  (block1): Sequential(  
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): LeakyReLU(negative_slope=0.2, inplace)  
  )  
  (block2): Sequential(  
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2, inplace)  
  )  
  (block3): Sequential(  
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)  
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): LeakyReLU(negative_slope=0.2, inplace)  
  )  
  (block4): Sequential(  
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(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)  
  )  
  (block_realfake): Sequential(  
    (0): Linear(in_features=8192, out_features=1, bias=True)  
  )  
  (block_cls): Sequential(  
    (0): Linear(in_features=8192, out_features=1, bias=True)  
    (1): Sigmoid()  
  )  
)
```

我實作的架構如上圖所示。

ACGAN\_G 代表 ACGAN 的 Generator。他是將(batch\_size, 101)的資料轉化成(batch\_size, 3, 64, 64)的一堆假圖片，其中 input 比 GAN 多了一維，就是要用來綁定特定的 class 數值。實作細節是先用 fully connected 把 101 維變成 512\*4\*4 維，再利用 convTranspose 把 512 -> 256 -> 128 -> 64 -> 3 channel。

ACGAN\_D 代表 ACGAN 的 Discriminator。他是將(batch\_size, 3, 64, 64)的資料轉化成(batch\_size, 2)的資料型態，其中多出來的 1 也是用來分辨特定 class 的不同型態，以笑為例，多出來的一維就是來分辨笑或不笑。

而其架構細節就是利用 conv 把吃進來的 3 -> 64 -> 128 -> 256 -> 512 channel，再來分別把 512\*4\*4 做 fully connected 到 1 維的資料，再把其中一個 1 維資料拿去過 sigmoid 以便分辨 class 的不同型態。

2. Plot 10 random pairs of generated images from your model.



我選擇的 class 是笑與不笑這個 class。上面是我做出來的圖片，上排是不笑的十個人，下排是這十個人笑起來的樣子。

3. Discuss what you've observed and learned from implementing ACGAN.

我認為我學到兩件事。

一：ACGAN 本身也沒決定架構該用什麼，所以我將 WGAN 與 ACGAN 做結合，在 WGAN 的 loss 上面加入 BCEloss 來衡量 class 之間的 difference，如此一來 train 出來的 loss 會變得有意義且可觀察，整個 train 個過程也會比較穩定。

二：ACGAN 的圖片品質因為加入了 fully connected layer 所以稍微模糊一些，但整體看起來還是有深的輪廓，也有些照片是比較像真人，所以整個實作下來的感覺是還蠻有成就感的。

### Problem 3 : DANN

1. Compute the accuracy on target domain, while the model is trained on source domain only.

data \ type	lower bound
USPS -> MNIST-M	24.66
MNIST-M -> SVHN	33.33
SVHN -> USPS	71.99

2. Compute the accuracy on target domain, while the model is trained on source and target domain.

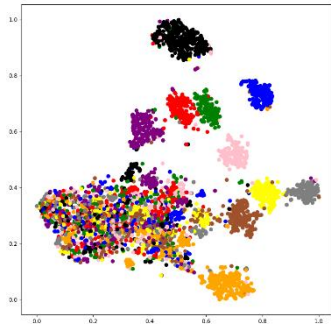
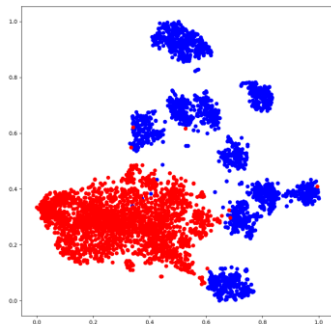
data \ type	model is trained on source and target
-------------	---------------------------------------

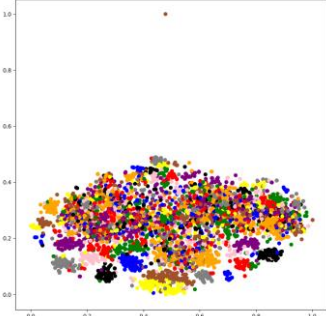
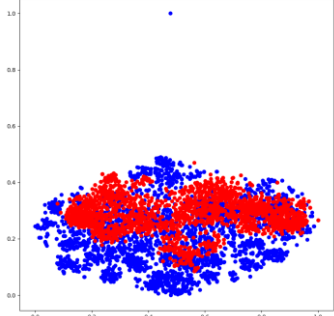
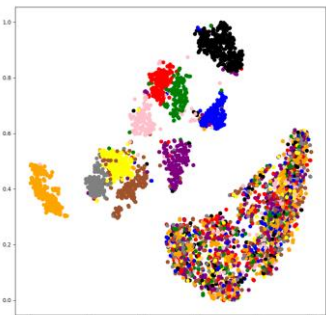
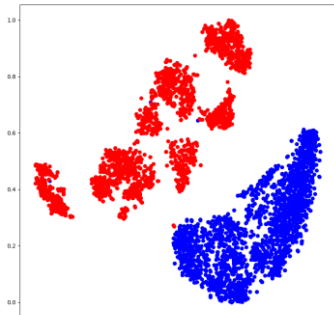
USPS -> MNIST-M	34.02
MNIST-M -> SVHN	47.88
SVHN -> USPS	41.60

3. Compute the accuracy on target domain, while the model is trained on target domain only.

data \ type	upper bound
USPS -> MNIST-M	98.27
MNIST-M -> SVHN	93.58
SVHN -> USPS	97.65

4. Visualize the latent space by mapping the testing images to 2D (with t-SNE)  
(a) different digit classes 0-9 (b) different domains

data \ type	different digit classes 0-9	different domain
USPS -> MNIST-M		

<p>MNIST-M -&gt; SVHN</p>		
<p>SVHN -&gt; USPS</p>		

##### 5. Describe the architecture & implementation detail of your model.

```

Feature_Extractor(
  (block1): Sequential(
    (0): Conv2d(3, 70, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(70, 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()
  )
  (block2): Sequential(
    (0): Conv2d(70, 50, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout2d(p=0.5)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): ReLU()
  )
)

```

```

Label_Classifier(
  (block1): 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()
  )
  (block2): Sequential(
    (0): Linear(in_features=100, out_features=100, bias=True)
    (1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.5)
    (3): ReLU()
  )
  (block3): Sequential(
    (0): Linear(in_features=100, out_features=10, bias=True)
    (1): Softmax()
  )
)

```



```

Domain_Classifier(
  (block1): 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()
  )
  (block2): Sequential(
    (0): Linear(in_features=100, out_features=2, bias=True)
    (1): Softmax()
  )
)

```

以上是我實作的 model 架構。主要分成三個部分。

**Feature\_Extractor :**

吃進(batch\_size, 3, 28, 28)輸出成(batch\_size, 50, 7, 7)的 Feature，主要是利用 conv 把 channel 3 -> 70 -> 50。

**Label\_Classifier :**

吃進(batch\_size, 50\*7\*7)輸出成(batch\_size, 10)，其中 10 是用來區分 0~9 哪個 digit。主要是利用 fully connected layer 把 50\*7\*7 -> 100 -> 100 -> 10。

**Domain\_Classifier :**

吃進(batch\_size, 50\*7\*7)輸出成(batch\_size, 2)，其中 2 是用來區分 source 和 target 兩個 domain。主要是利用 fully connected layer 把 50\*7\*7 -> 100 -> 2。

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

我學習及觀察到的有以下兩點。

一：在做 testing 的過程中，SVHN -> USPS 的 lower bound 會異常高，而在 t-SNE 上也可以看出來 target 的 data 反而分的比 source 的 data 更好，可以推測是整個 USPS data 跟 model 的架構向性比較合，可以讓 USPS data 在數量比較上的情況下依然 train 得很好，更甚之影響了 SVHN data 原本應有的分類情況。

二：我學到 DANN 真的是一個很難的技術，可能最大的原因是每個 dataset 有他們各自最適合的 layer 擺法。因為我的 layer 幾乎是參照網路上別人 train 得很好的 model 去做訓練，但套到我們這三組 dataset 上之後就幾乎沒有這麼好的效果，更甚至 t-SNE 的圖看起來並有效。所以推測可能最好的 DANN 作法是依據不同 model 特別去試各種不同的 layer 擺法，才能各自有最好的效果。

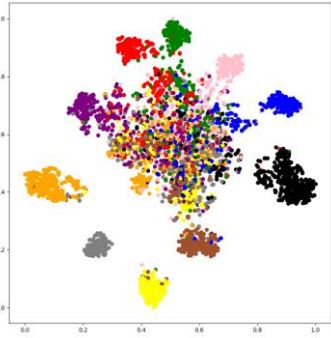
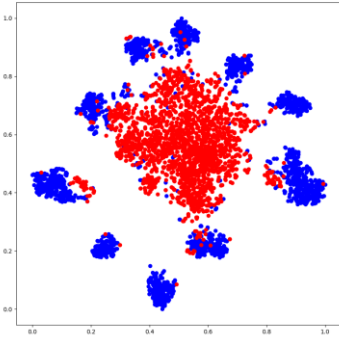
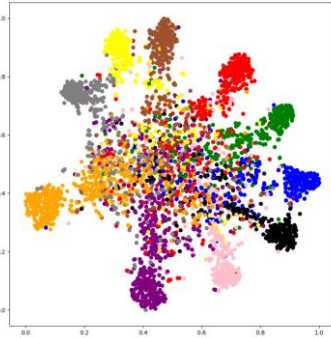
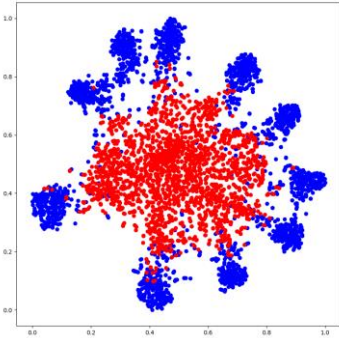
#### Problem 4 : DAN

1. Compute the accuracy in target domain, while the model is trained on source and target domain.

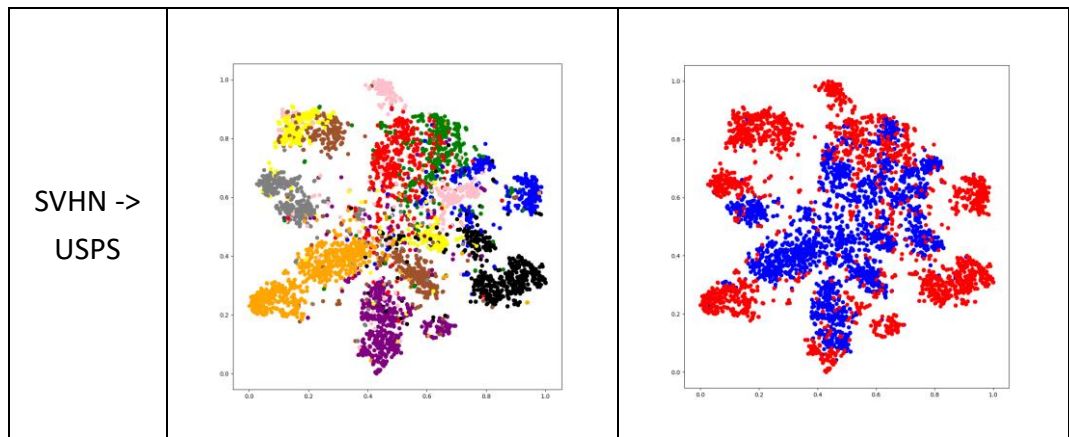
<div>data</div> <div>type</div>	model is trained on source and target
---------------------------------	---------------------------------------

USPS -> MNIST-M	37.26
MNIST-M -> SVHN	48.32
SVHN -> USPS	62.83

2. Visualize the latent space by mapping the testing images to 2D (with t-SNE)  
(a) different digit classes 0-9 (b) different domains

type data	different digit classes 0-9	different domain
USPS -> MNIST-M		
MNIST-M -> SVHN		





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

```

Feature_Extractor(
  (block1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (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()
  )
  (block2): Sequential(
    (0): Conv2d(64, 10, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout2d(p=0.5)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): ReLU()
  )
  (block3): Sequential(
    (0): Linear(in_features=160, out_features=100, bias=True)
    (1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Linear(in_features=100, out_features=100, bias=True)
    (3): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)

Label_Classifier(
  (block1): Sequential(
    (0): Linear(in_features=100, out_features=10, bias=True)
  )
)

```

我實作的是 DAN，以上是我實作的 model，主要分成兩個部分。

Feature\_Extractor：

吃進(batch\_size, 3, 28, 28)輸出成(batch\_size, 100)的 Feature。主要是利用 conv 把 channel 3 -> 64 -> 10，再利用 fully connected layer 把 10\*4\*4 -> 100 -> 100。

Label\_Classifier：

吃進(batch\_size, 100)輸出成(batch\_size, 10)。其中 10 是用來區分 0~9 的哪個 digit。架構上是利用 fully connected layer 把 100 -> 10 維。

### 4. Discuss what you've observed and learn from implementing your improved UDA model.

我所實作的 improved 是 DAN，我認為我學到及觀察到的東西有以下兩點。

一：

DAN 架構跟 DANN 的差別在於，DAN 沒有一個 domain classifier，而有兩個 source 跟 target 的 label classifier，以及一個 feature extractor。他是在原本 source label loss 的基礎上加上 mmd loss，用以衡量 source\_features 和 target\_features 以及 source\_pre\_labels 和 target\_pre\_labels 之間的距離，想辦法讓兩個距離更靠近，再加上原本就有的 source label loss，可以讓 source 的 label\_classifier 更精準地拉近 source 與 target。

二：

DAN 的 t-SNE 以及 accuracy 都要比原本的 DANN 還要好上許多。從 t-SNE 圖片可以看得出來，兩個 domain 彼此有在靠近，而且同類別的 class 是有群聚效果的，所以可以顯示說 DA 在這部份是非常成功的，整題看起來地靠近程度也比 DANN 好上許多。

合作同學：

我整份報告有疑問的地方，都是與我 ImLab 的實驗室同學們一起討論，我們僅限於題目定義與演算法的討論，實作的 code 都是各自寫各自的。以下是同學們的 ID，b04901190、R07922002、R07922024、R07922043、R97922120。

參考資料：

WGAN：

- [1] <https://zhuanlan.zhihu.com/p/25071913>
- [2] <https://github.com/eriklindernoren/PyTorch-GAN>
- [3] <https://www.jianshu.com/p/ddfd7fba11d0>

ACGAN：

- [1] <https://zhuanlan.zhihu.com/p/44177576>

DANN：

- [1] [https://github.com/NaJaeMin92/pytorch\\_DANN/blob/master/model.py](https://github.com/NaJaeMin92/pytorch_DANN/blob/master/model.py)
- [2] [https://github.com/CuthbertCai/pytorch\\_DANN](https://github.com/CuthbertCai/pytorch_DANN)

DAN：

- [1] [https://github.com/CuthbertCai/pytorch\\_DAN/blob/master/models.py](https://github.com/CuthbertCai/pytorch_DAN/blob/master/models.py)
- [2] [https://discuss.pytorch.org/t/maximum-mean-discrepancy-mmd-and-radial-basis-function-rbf/1875?fbclid=IwAR0l6956\\_05qOXlhBJg9eOltTWh870P5lFgOCOHZI\\_QwHZ8wo3ppE2JSX1E](https://discuss.pytorch.org/t/maximum-mean-discrepancy-mmd-and-radial-basis-function-rbf/1875?fbclid=IwAR0l6956_05qOXlhBJg9eOltTWh870P5lFgOCOHZI_QwHZ8wo3ppE2JSX1E)