

# HW2 Report

B08901165 電機四 南策昇

## Problem 1

1.

Discriminator of A:

```
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()  
    )  
)
```

Generator of A:

```
Generator(  
    (main): Sequential(  
        (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1,  
1), bias=False)
```

```

        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
        (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
        (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (8): ReLU(inplace=True)
        (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
        (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
        (13): Tanh()
    )
)

```

Discriminator of B:

```

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)
    )
)

```

Generator of B:

```

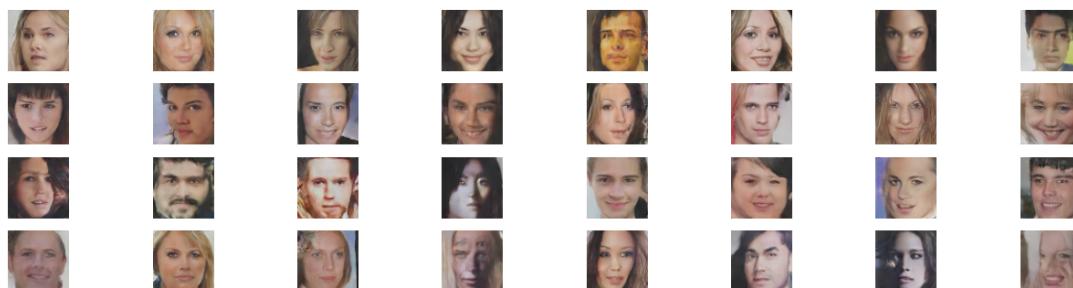
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1,
1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2,
2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)

```

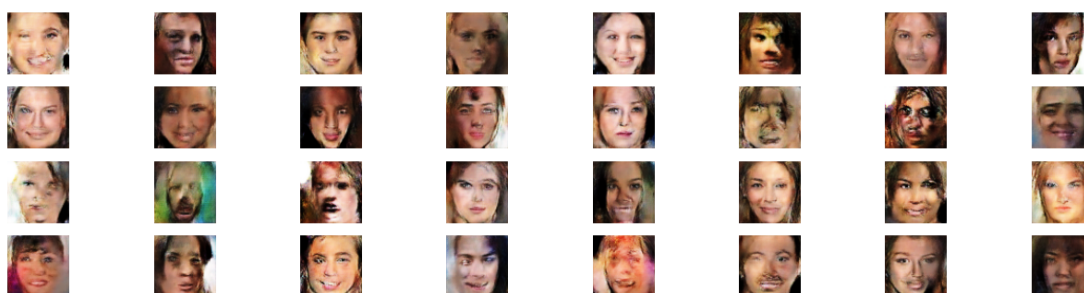
Ref: [DCGAN Tutorial — PyTorch Tutorials 1.13.0+cu117 documentation](#)

2.

Model A:



Model B:



FID 和 `face_recog.py` 的 accuracy 都顯示 Model A 的表現比 Model B 好，而這也顯示在這些生成的圖片上，Model A 生成的人臉大部分輪廓都相當明確，反觀 Model B 生成的圖片在輪廓附近會有模糊或者歪曲的狀況，色塊也會有點雜亂，只能掌握到人臉的大致特徵，因此雖然 `face_recog.py` 上面 accuracy 差異不大（約 3%），但是 FID 就差很多（67 v.s. 25）。

3.

原本 DCGAN 和 WGAN 都 train 不起來，我還以為是我 train 不夠久，沒想到是真的沒辦法。可是看朋友有人有 train 成功，我就花了一點時間好好端詳一下這個看起來沒問題的 code 到底是在哪裡出了問題。後來才發現我自作聰明，把 Normalization 的參數改成 Imagenet 的這組，而不是用 0.5，想說應該會表現好一點，但問題就是出在這裡。用 0.5 標準化是為了要讓 pixels 數值都限定在 -1~1，但用 Imagenet 就會讓 pixels 跑出這個範圍，偏偏我的 generator 的最後一層掛了一個 hyperbolic tangent，所以會讓所有 output 都在 -1~1 之間，這樣 discriminator 只要看 pixels 的數值有沒有落在這個範圍，就能判定他是真圖還是假圖了。把這點改回來後，就有 train 起來了。

原本預期 WGAN 會 train 比較好，沒想到 FID 一直下不去，我猜可能是因為 WGAN 的 D 和 G training 的頻率不一樣，如果要達到一定的效果，要 train 的比 DCGAN 還要久才行，另外也有可能是 parameters clipping 在誤事，讓他很難 train 起來，不過這些還有待進一步的實驗證明。

## Problem 2

1.

Model architecture:

```
DDPM(  
    (nn_model): ContextUnet(  
        (init_conv): ResidualConvBlock(  
            (conv1): Sequential(  
                (0): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1),  
padding=(1, 1))  
                (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True)  
                (2): GELU(approximate=None)  
            )  
            (conv2): Sequential(  
                (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),  
padding=(1, 1))  
                (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True)  
                (2): GELU(approximate=None)  
            )  
        )  
        (down1): UnetDown(  
            (model): Sequential(  
                (0): ResidualConvBlock(  
                    (conv1): Sequential(  
                        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1,  
1), padding=(1, 1))  
                        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1,  
affine=True, track_running_stats=True)  
                        (2): GELU(approximate=None)  
                    )  
                    (conv2): Sequential(  
                        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1,  
1), padding=(1, 1))  
                        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1,  
affine=True, track_running_stats=True)  
                        (2): GELU(approximate=None)
```

```

        )
    )
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
    )
)
(down2): UnetDown(
  (model): Sequential(
    (0): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (2): GELU(approximate=None)
      )
      (conv2): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (2): GELU(approximate=None)
      )
    )
  )
  (1): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
)
(to_vec): Sequential(
  (0): AvgPool2d(kernel_size=7, stride=7, padding=0)
  (1): GELU(approximate=None)
)
(timeembed1): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=256, bias=True)
    (1): GELU(approximate=None)
    (2): Linear(in_features=256, out_features=256, bias=True)
  )
)

```

```

)
(timeembed2): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=128, bias=True)
    (1): GELU(approximate=None)
    (2): Linear(in_features=128, out_features=128, bias=True)
  )
)
(contextembed1): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=10, out_features=256, bias=True)
    (1): GELU(approximate=None)
    (2): Linear(in_features=256, out_features=256, bias=True)
  )
)
(contextembed2): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=10, out_features=128, bias=True)
    (1): GELU(approximate=None)
    (2): Linear(in_features=128, out_features=128, bias=True)
  )
)
(up0): Sequential(
  (0): ConvTranspose2d(256, 256, kernel_size=(7, 7),
stride=(7, 7))
  (1): GroupNorm(8, 256, eps=1e-05, affine=True)
  (2): ReLU()
)
(up1): UnetUp(
  (model): Sequential(
    (0): ConvTranspose2d(512, 128, kernel_size=(2, 2),
stride=(2, 2))
    (1): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1,
1), padding=(1, 1))
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)

```

```

        (2): GELU(approximate=None)
    )
    (conv2): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): GELU(approximate=None)
    )
  )
  (2): ResidualConvBlock(
    (conv1): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): GELU(approximate=None)
    )
    (conv2): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): GELU(approximate=None)
    )
  )
)
)
)
)
  (up2): UnetUp(
    (model): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
      (1): ResidualConvBlock(
        (conv1): Sequential(
          (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

```



```

        (2): GELU(approximate=None)
    )
    (conv2): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): GELU(approximate=None)
    )
  )
  (2): ResidualConvBlock(
    (conv1): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): GELU(approximate=None)
    )
    (conv2): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): GELU(approximate=None)
    )
  )
)
(out): Sequential(
  (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): GroupNorm(8, 128, eps=1e-05, affine=True)
  (2): ReLU()
  (3): Conv2d(128, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
(loss_mse): MSELoss()

```

)

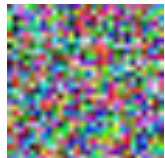

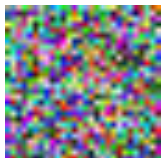



參考來源：[Conditional Diffusion MNIST/script.py at main · TeaPearce/Conditional Diffusion MNIST \(github.com\)](#)

我使用 UNet 當作 backbone 並且在層與層之間都有加 batchnorm。此外，在去除雜訊的時候，我使用 total step = 600，並且送進 model 生成時會有兩個部分在 train，一部分是有 context，一部分是沒有 context，兩者之間的 weight 我調整成 1:0.5。Learning rate 有做 linear schedule，但是實際上 train 的時候，我有刻意先在大的 learning rate train 多一點 epoch，因為我發現 linear schedule 其實有一點降低得太快。其他關於 beta 或者 mean 的計算都是參考原始 paper 做的。

2.



3.

t = 0	t = 120	t = 240	t = 360	t = 480	t = 600
					

4.

我覺得 Diffusion model 應該是這次作業最難的部分，剛開始我參考了幾個網站，產出我的 code，結果發現他一直 train 不起來，而且生成圖片超級慢，於是我仔細研究了一下我的 code，發覺我 timestep 弄反了，全部存在沒有 denoise 的雜訊，改了一下之後就有了一點樣子。之後我又想說他 denoise 好久，我讓他只要生成其中幾個 timestep 的過程就好，沒想到 train 出來馬上又壞了。我就觀察了一下我 print 出來的 timestep，雖然我是設定每隔 120 個 timestep 就顯示一次圖片，但是它們之間感覺不像 denoise 了 120 次，許多隨機的 pixel 都長得一樣，頓時心中有了一個猜想，趕快重新細讀一下 paper，發覺果然是這樣，以前我一直以為 diffusion model 將 timestep 資訊 encode 進去就可以直接生成 denoise 那麼多 timestep 的資料，所以我只要丟 timestep = 600 就可以直接得到生成的圖片，但事實上生成過程要從 600 一層一層 denoise 到 1，跑過這個 model 600 次，才能生好一張圖片，難怪我相隔 120 timestep 的圖片長得差不多，因為中間被跳過的 timestep 我全都沒跑，所以他事實上才 denoise 了一次而已。改回來之後，我 train 起來就沒什麼問題了。

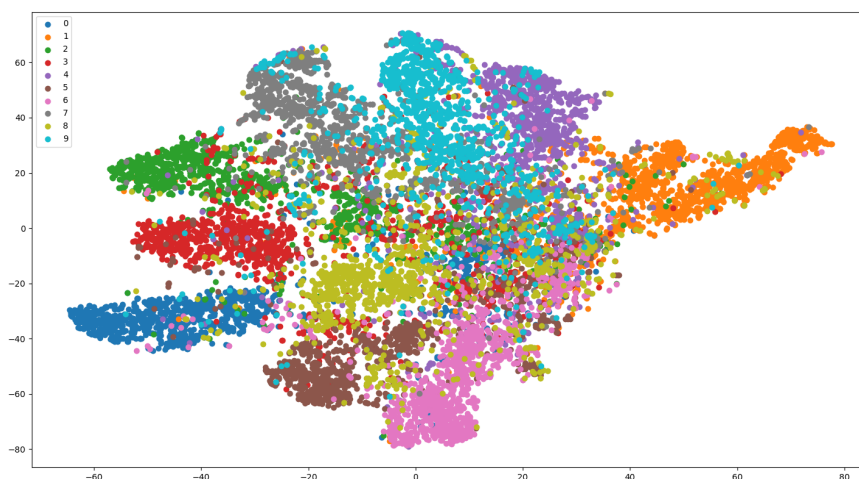
## Problem 3

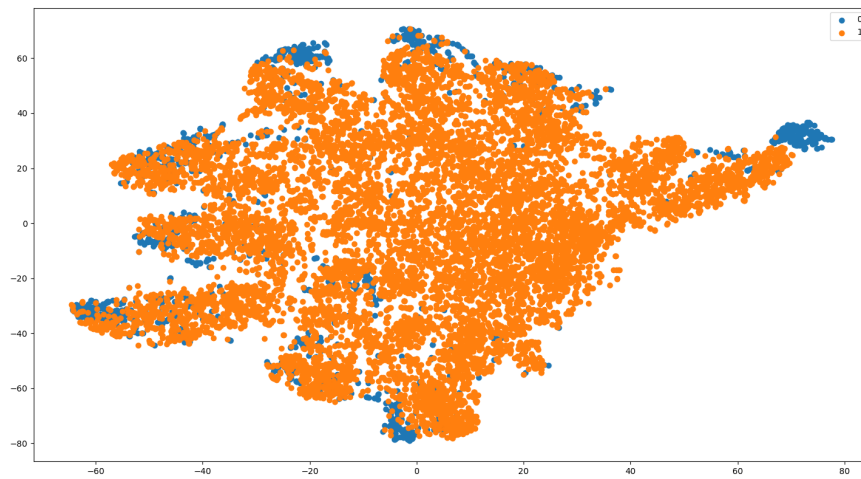
1.

	MNIST-M -> SVHN	MNIST-M -> USPS
Trained on source	0.358	0.708
Adaptation (DANN)	0.504	0.818
Trained on target	0.911	0.983

2.

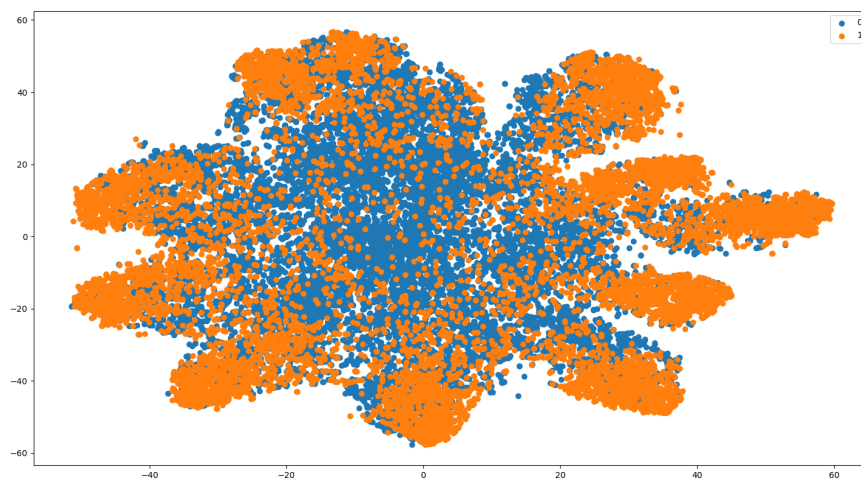
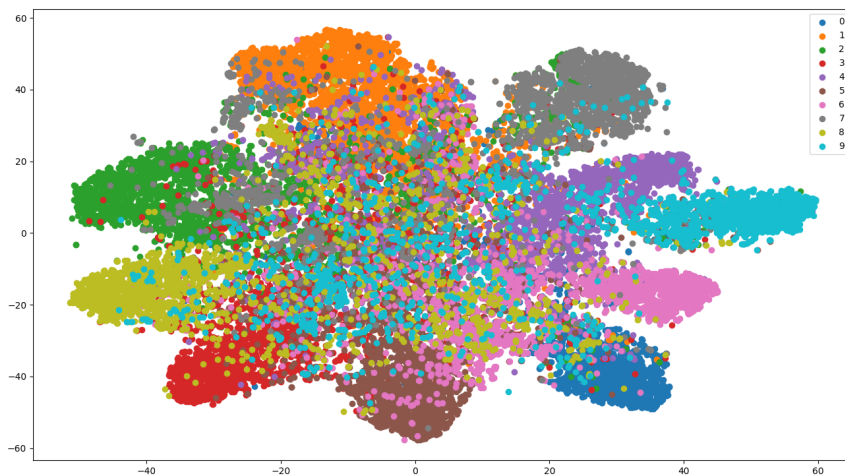
MNIST-M -> usps





0: usps 1: MNIST-M

MNIST-M -> svhn



0: svhn 1: MNIST-M

3.

我的 model 其實就是簡單的幾層 CNN 和 FC layer，大致結構就跟老師上課教的一樣，因為我就把它想成是要 train MNIST-M，眾所皆知這其實並不複雜。關鍵應該是我在 domain classifier 和 feature extractor 中間加了 Reverse layer (GRL)讓 gradient 在經過這裡時會變號，也就是 DANN 的精髓，如下：

```
class GRL(Function):  
    @staticmethod  
    def forward(ctx, x, alpha):  
        ctx.alpha = alpha  
        return x.view_as(x)  
  
    @staticmethod  
    def backward(ctx, grad_output):  
        output = grad_output.neg() * ctx.alpha  
        return output, None
```

model 架構：

```
class DANN(nn.Module):  
    def __init__(self, num_classes=10):  
        super(DANN, self).__init__()  
        self.features = nn.Sequential(  
            nn.Conv2d(3, 32, 5),  
            nn.ReLU(inplace=True),  
            nn.MaxPool2d(2),  
            nn.Conv2d(32, 48, 5),  
            nn.ReLU(inplace=True),  
            nn.MaxPool2d(2),  
        )  
        self.avgpool = nn.AdaptiveAvgPool2d((5, 5))  
        self.task_classifier = nn.Sequential(  
            nn.Linear(48*5*5, 100),  
            nn.ReLU(inplace=True),  
            nn.Linear(100, 100),  
            nn.ReLU(inplace=True),  
            nn.Linear(100, num_classes)  
        )  
        self.domain_classifier = nn.Sequential(  
            nn.Linear(48*5*5, 100),  
            nn.ReLU(inplace=True),
```



```

        nn.Linear(100,2)
    )
    self.GRL=GRL()
    def forward(self,x,alpha):
        x = x.expand(x.data.shape[0], 3, 28, 28)
        x=self.features(x)
        x=self.avgpool(x)
        x=torch.flatten(x,1)
        task_predict=self.task_classifier(x)
        x=GRL.apply(x,alpha)
        domain_predict=self.domain_classifier(x)
        return task_predict,domain_predict

```

這樣主幹就建立好了。

Ref: [function/DANN: pytorch implementation of Domain-Adversarial Training of Neural Networks \(github.com\)](https://github.com/yaoliang0123/DANN)

此外，剛開始 train 的時候，accuracy 一直都起不來，甚至連 source 都沒辦法，只能說幸好以前 train 過 MNIST，所以知道這樣的 model 架構絕對是夠大的，可先從其他地方開始 debug。我猜測應該是 domain loss 那邊站太重了，老師上課也有講，剛開始的時候可以讓 classifier 先 train 一下，有一個底，再修 domain，所以我就做了一個 scheduling，讓 domain loss 不要太快出現，而情況也因此改善了，但是還遠不及 baseline，甚至還出現一個奇怪的問題。我發現我只要 train 到大約第三個 epoch，原本長得好好的 acc 就會突然崩到剩 20%，不管在哪個 task 都是，猜測又是因為 domain loss 突然 dominate 了整個 update，所以我就試著在 domain loss scheduling 前面加一個係數，讓他即使 schedule 到最後，也不會變成 1，結果雖然有改善，但是跑到 20 epochs 時又崩掉了，accuracy 還是差 baseline 一點。所以我就動筆算了一下這兩次崩掉的時候，scheduling 乘上係數後的值大約是多少，發現竟然是幾乎一樣，表示如果 domain loss 和 classifier loss 的 weighting 漲到這個程度，就有很大的機會會崩掉，於是我就逆回去算我加的係數應該要是多少(最後是調  $0.5 * \text{scheduling}$ )，才不會崩掉。加上去之後，就都 train 過 baseline 了。