生醫電資所碩二 程映元 R11945039
2023 Fall DLCV

Problem 1: Problem 1: Conditional Diffusion models (35%) [digit dataset - MNIST-M]

1. (5%) Follow the Github Example to draw your model architecture and describe your implementation details

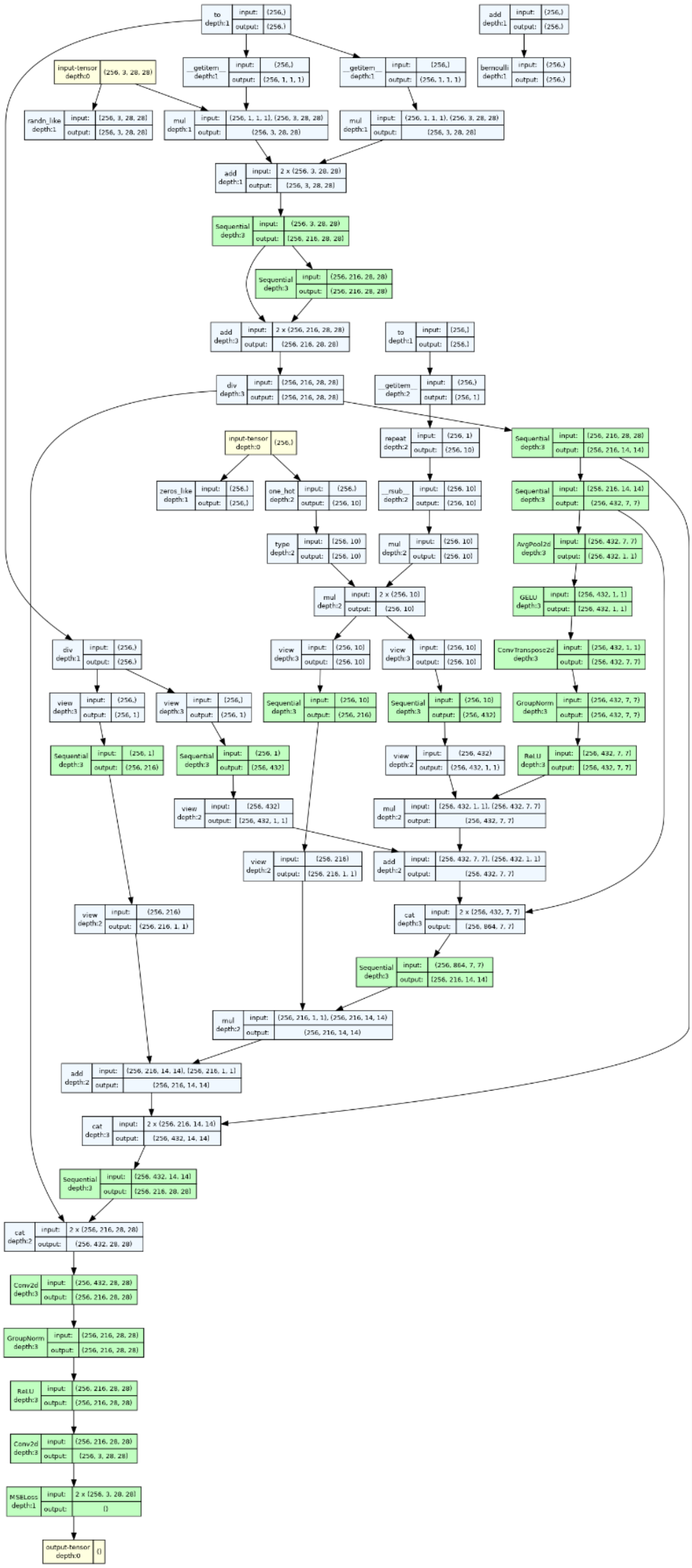
Implementation Details:

Model Name	Conditional Diffusion MNIST (Adapted from: https://github.com/TeaPearce/Conditional_Diffusion_MNIST/tree/main#conditional-diffusion-mnist)
Diffusion model	DDPM
Backbone	U-Net (自己搭小型版本)
Conditioning Method	https://arxiv.org/abs/2207.12598 (Paper: Classifier-Free Diffusion Guidance)
Epochs	25
Batch Size	256
Criterion	nn.MSELoss
lr	1e-4
optimizer	Adam
n_feat (裡面很多的 filter 數都是它的倍數)	216
n_T (generated steps)	400

(提供兩個模型的架構圖，第一個是用: torchview, 第二個是直接 print 模型出來)

生醫電資所碩二 程映元 R11945039

2023 Fall DLCV



```
DDPM(
  (nn_model): ContextUnet(
    (init_conv): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(3, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate='none')
      )
      (conv2): Sequential(
        (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(216, 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(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate='none')
          )
          (conv2): Sequential(
            (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(216, 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(216, 432, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(432, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): GELU(approximate='none')
          )
          (conv2): Sequential(
            (0): Conv2d(432, 432, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(432, 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=432, bias=True)
        (1): GELU(approximate='none')
        (2): Linear(in_features=432, out_features=432, bias=True)
      )
    )
    (timeembed2): EmbedFC(
      (model): Sequential(
        (0): Linear(in_features=1, out_features=216, bias=True)
        (1): GELU(approximate='none')
        (2): Linear(in_features=216, out_features=216, bias=True)
      )
    )
```

```
(timeembed2): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=216, bias=True)
    (1): GELU(approximate='none')
    (2): Linear(in_features=216, out_features=216, bias=True)
  )
)
(contextembed1): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=10, out_features=432, bias=True)
    (1): GELU(approximate='none')
    (2): Linear(in_features=432, out_features=432, bias=True)
  )
)
(contextembed2): EmbedFC(
  (model): Sequential(
    (0): Linear(in_features=10, out_features=216, bias=True)
    (1): GELU(approximate='none')
    (2): Linear(in_features=216, out_features=216, bias=True)
  )
)
(up0): Sequential(
  (0): ConvTranspose2d(432, 432, kernel_size=(7, 7), stride=(7, 7))
  (1): GroupNorm(8, 432, eps=1e-05, affine=True)
  (2): ReLU()
)
(up1): UnetUp(
  (model): Sequential(
    (0): ConvTranspose2d(864, 216, kernel_size=(2, 2), stride=(2, 2))
    (1): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate='none')
      )
      (conv2): Sequential(
        (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate='none')
      )
    )
    (2): ResidualConvBlock(
      (conv1): Sequential(
        (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate='none')
      )
      (conv2): Sequential(
        (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): GELU(approximate='none')
      )
    )
  )
)
```

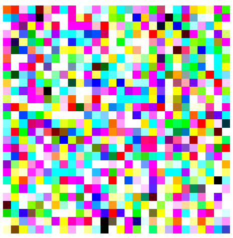
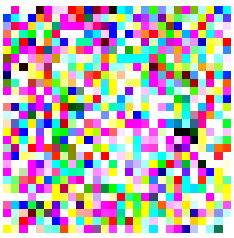
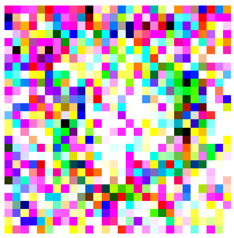
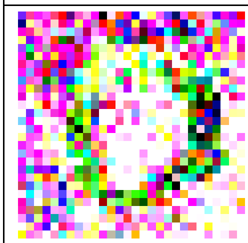
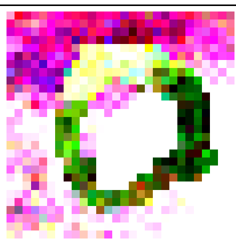
```
  (up2): UnetUp(
    (model): Sequential(
      (0): ConvTranspose2d(432, 216, kernel_size=(2, 2), stride=(2, 2))
      (1): ResidualConvBlock(
        (conv1): Sequential(
          (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate='none')
        )
        (conv2): Sequential(
          (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate='none')
        )
      )
      (2): ResidualConvBlock(
        (conv1): Sequential(
          (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate='none')
        )
        (conv2): Sequential(
          (0): Conv2d(216, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (1): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): GELU(approximate='none')
        )
      )
    )
  )
  (out): Sequential(
    (0): Conv2d(432, 216, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): GroupNorm(8, 216, eps=1e-05, affine=True)
    (2): ReLU()
    (3): Conv2d(216, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
(loss_mse): MSELoss()
)
```

2. (5%) Please show 10 generated images for each digit (0-9) in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits. [see the below example]

Columns: Indicating different digits, Rows: Indicating different noise inputs

3. (5%) Visualize total six images in the reverse process of the first “0” in your grid in (2) with different time steps. [see the below example]

(t = the steps of doing denoise, total time steps=400)

t=1	t=101	t=201
		
t=261	t=361	t=400
		

4. (5%) Please discuss what you've observed and learned from implementing conditional diffusion model.
    1. 做 sampling 要依序走完所有的 steps(400 個)，所以時間相較一步產生圖片的 GAN 系列模型有點久。但相較於 GAN，diffusion models 更能操控它使用它的細節(ex: 能自訂 total time steps of diffusion 以及輸出任意 time step 的圖片)。
    2. 使用的 conditioning (Classifier-Free Diffusion Guidance) 的方式如下:

$$\hat{\epsilon}_t = (1 + w)\psi(z_t, c) - w\psi(z_t).$$

$\psi(z_t, c)$ ：代表 context c

$\psi(z_t)$ ：代表在 timestep t 時的 noise zt

在 generation time 時透過調高 weight w (w>=0)來產生更典型但較不多元的圖片。

Weight experiment:

w = 0.0	
w = 0.5	
w = 2.0	

如上可見正面越往下數字越像【正常】的數字，但風格都較統一。

- ➔ 這種能控制【典型】與【多元】(一個 tradeoff) 的生成式學習的模型很有趣 (且應用起來更方便)
- ➔ 學到很新的 conditional diffusion model approach: Classifier-Free Diffusion Guidance 的實做細節

Problem 2: DDIM (35%)

1. (7.5%) Please generate face images of noise 00.pt ~ 03.pt with different eta in one grid. Report and explain your observation in this experiment.

(第一列到第十列分別代表 eta = 0.0, 0.1, …, 0.9，第一行到第四行則分別代表 noise = 00.pt, 01.pt, 02.pt, 03.pt。)

(兩列的 eta 都是 0.5，第一行到第四行則分別代表 noise = 00.pt, 01.pt, 02.pt, 03.pt)

1. 由第二組圖片可知，eta 越大時，每次產生圖片的差異越大(eta=0.0 時，固定 noise 的情況下每次產生的圖片都一樣)
2. eta 越大時，變動 eta 所產生的圖片的差異越大 (如第一組圖，eta 小時產生圖片的風格都很類似，eta 變大後，產生圖片的風格有很大的差異)

-> eta 是一個控制 deterministic ddim(eta = 0)和 stochastic ddpm(eta=1)之間的 the level of interpolation 的變數，eta 越小就越接近 ddim，反之則接近 ddpm，故由上述觀察驗證，隨著 eta 越大，每次輸出圖片風格差異越大(越接近 ddpm)。

2. (7.5%) Please generate the face images of the interpolation of noise 00.pt ~ 01.pt. The interpolation formula is spherical linear interpolation, which is also known as slerp. What will happen if we simply use linear interpolation? Explain and report your observation.

Slerp(spherical linear interpolation):

$$\boldsymbol{x}_T^{(\alpha)} = \frac{\sin((1-\alpha)\theta)}{\sin(\theta)}\boldsymbol{x}_T^{(0)} + \frac{\sin(\alpha\theta)}{\sin(\theta)}\boldsymbol{x}_T^{(1)}$$

where $\theta = \arccos\left(\frac{(\boldsymbol{x}_T^{(0)})^\top \boldsymbol{x}_T^{(1)}}{\|\boldsymbol{x}_T^{(0)}\|\|\boldsymbol{x}_T^{(1)}\|}\right)$. These values are used to produce DDIM samples.

in this case, α = {0.0, 0.1, 0.2, ..., 1.0}.

(最左邊的圖 noise=00.pt，最右邊的圖 noise=01.pt，eta 都是 0)

Lerp(linear interpolation):

$$\text{out}_i = \text{start}_i + \text{weight}_i \times (\text{end}_i - \text{start}_i)$$

(圖片由左至右的 Weighti 分別等於 (0.0, 0.1, 0.2, ..., 1.0))
(最左邊的圖 noise=00.pt，最右邊的圖 noise=01.pt，eta 都是 0)
觀察(helped by bing AI)：
Lerp：如果我們只是使用線性插值來插值兩個 noise 向量，我們會得到兩個向量的線性組合，這可能不是一個單位向量。這意味著插值的 noise 向量可能有不同的大小，這可能影響人臉生成的質量。此外，線性插值可能不保持 noise 向量的平滑性或連續性，這可能導致人臉圖像出現突然或不自然的變化。

Slerp：相反，如果我們使用球面線性插值來插值兩個 noise 向量，我們會得到一個位於通過兩個向量的大圓弧上的單位向量。這意味著插值的 noise 向量會有與原始 noise 向量相同的大小，這可能提高人臉生成的質量。而且，球面線性插值可以保持 noise 向量的平滑性或連續性，這可能產生更自然的人臉圖像過渡。
-> 故如圖所示，使用 slerp 相較於 lerp 產生的影像較為自然(lerp 影像有奇怪的粉紅色出現)。

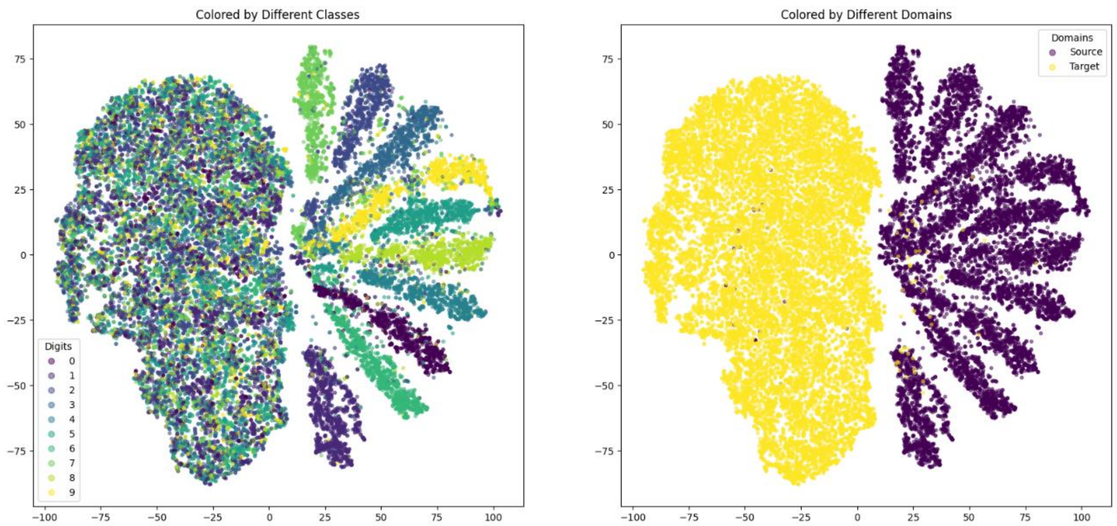
Problem 3: DANN (35%)

1. (10%) Please create and fill the table with the following format in your report:

	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	0.4036	0.7870
Adaptation (DANN)	0.4412	0.8864
Trained on target	0.9388	0.9859

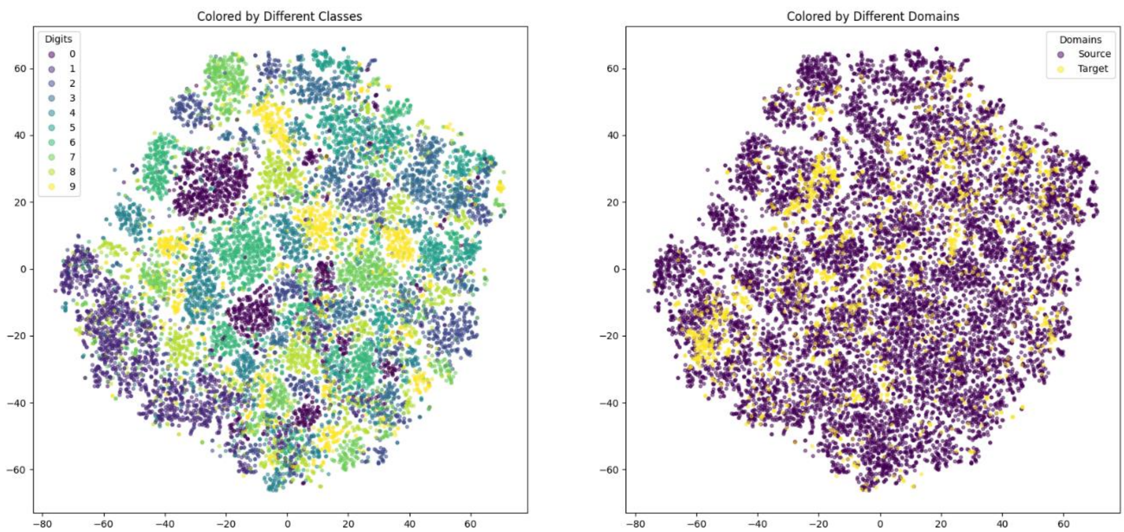
2. (10%) Please visualize the latent space (output of CNN layers) of DANN by mapping the validation images to 2D space with t-SNE. For each scenario, you need to plot two figures which are colored by digit class (0-9) and by domain, respectively. Note that you need to plot the figures of both 2 scenarios, so 4 figures in total.

MNIST-M → SVHN



(Fig1)

MNIST-M → USPS



(Fig2)

3. (3%) Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.

Model Name	Domain-Adversarial Training of Neural Networks ref： 1. https://github.com/NaJaeMin92/pytorch_DANN 2. https://github.com/pha123661/NTU-2022Fall-DLCV/blob/master/HW2/P3_USPS_model.py
Backbone	CNN
Epochs	100
Batch Size	1024
Criterion	nn.CrossEntropyLoss() nn.BCEWithLogitsLoss()
lr	3e-4
optimizer	Adam

使用的架構如下：

```
FeatureExtractor(
  (conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Dropout2d(p=0.5, inplace=False)
    (7): ReLU()
    (8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  )
)
LabelPredictor(
  (l_clf): Sequential(
    (0): Linear(in_features=128, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=256, bias=True)
    (4): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Linear(in_features=256, out_features=10, bias=True)
  )
)
DomainClassifier(
  (d_clf): Sequential(
    (0): Linear(in_features=128, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=256, bias=True)
    (4): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

觀察與收穫：

DANN 用在 MNIST-M → SVHN，從下界的 0.40 只提升了 4%，距離上界的 90%有很大的距離。而用在 MNIST-M → USPS 則是從下界的 0.8 提升到 0.88，距離上界已經不遠 0.98，由此可知 DANN 用在 MNIST-M → SVHN 不能帶來顯著的提升。其可能原因是 MNIST-M 和 SVHN 的影像太不像了，因此抽取出的特徵難以混淆(見 fig1)，故模型在 MNIST-M 學到的能力仍只適用於 MNIST-M(見 fig1，模型在左圖的 target domain 對應到的 feature 的 tsne 都混在一起，代表模型找不出 target domain 有用的特徵)。而 USPS 的影像和 MNIST-M 蠻像的(見 fig2)，故抽取出的特徵容易混淆(見 fig1)，故 DANN 效果較好。

-> DANN 適用於 domain gap(btw source and target)不要太大的 task，否則提升效果有限。