Problem 1.

(1)

Model A

```
Generator(
  (project): 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)
  )
  (conv): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
  (last): Sequential(
    (0): ConvTranspose2d(128, 3, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
    (1): Tanh()
  )
```

```
Discriminator(
(disc): Sequential(
  (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (1): LeakyReLU(negative_slope=0.2)
  (2): 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=True)
  )
  (3): 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=True)
  )
  (4): 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=True)
  )
  (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (6): Sigmoid()
)
```

Model B

```
Generator(
  (project): 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)
  )
  (conv): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
  (last): Sequential(
    (0): ConvTranspose2d(128, 3, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1), bias=False)
    (1): Tanh()
  )
```

```
Discriminator(
(disc): Sequential(
  (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  (1): LeakyReLU(negative_slope=0.2)
  (2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LayerNorm((128, 16, 16), eps=1e-05, elementwise_affine=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LayerNorm((256, 8, 8), eps=1e-05, elementwise_affine=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LayerNorm((512, 4, 4), eps=1e-05, elementwise_affine=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
)
```

(2)

Model A

Model B

Model B 在大部分情況都有正常的臉型，反之 model A 則有許多形變與模糊的臉

(3)

model A 與 model B 的差異主要在 discriminator 與 loss function，model B 結合 WGAN-GP 與 spectrum normalization 達到更穩定的訓練結果，訓練遇到的難處在於不太能夠透過 loss 來判斷 model 使否持續往好的方向訓練下去

Problem 2.

(1)

```
Unet(
  (label_emb): Embedding(10, 256)
  (conv1): WConv(
    (wconv): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (1): GroupNorm(1, 64, eps=1e-05, affine=True)
      (2): GELU(approximate=none)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (4): GroupNorm(1, 64, eps=1e-05, affine=True)
    )
  )
  (down_layers): ModuleList(
    (0): U_Down(
      (maxpool_conv): Sequential(
        (0): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
        (1): WConv(
          (wconv): Sequential(
            (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): GroupNorm(1, 64, eps=1e-05, affine=True)
            (2): GELU(approximate=none)
            (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): GroupNorm(1, 64, eps=1e-05, affine=True)
          )
        )
        (2): WConv(
          (wconv): Sequential(
            (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): GroupNorm(1, 128, eps=1e-05, affine=True)
            (2): GELU(approximate=none)
            (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): GroupNorm(1, 128, eps=1e-05, affine=True)
          )
        )
      )
```

```
      (emb_layer): Sequential(
        (0): SiLU()
        (1): Linear(in_features=256, out_features=128, bias=True)
      )
      (attention): SelfAttention(
        (mha): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=128, out_features=128, bias=True)
        )
        (ln): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
        (ff_self): Sequential(
          (0): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
          (1): Linear(in_features=128, out_features=128, bias=True)
          (2): GELU(approximate=none)
          (3): Linear(in_features=128, out_features=128, bias=True)
        )
      )
    )
```

```
  (1): U_Down(
    (maxpool_conv): Sequential(
      (0): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
      (1): WConv(
        (wconv): Sequential(
          (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): GroupNorm(1, 128, eps=1e-05, affine=True)
          (2): GELU(approximate=none)
          (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (4): GroupNorm(1, 128, eps=1e-05, affine=True)
        )
      )
      (2): WConv(
        (wconv): Sequential(
          (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): GroupNorm(1, 128, eps=1e-05, affine=True)
          (2): GELU(approximate=none)
          (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (4): GroupNorm(1, 128, eps=1e-05, affine=True)
        )
      )
    )
    (emb_layer): Sequential(
      (0): SiLU()
      (1): Linear(in_features=256, out_features=128, bias=True)
    )
```

```
    (attention): SelfAttention(
      (mha): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(in_features=128, out_features=128, bias=True)
      )
      (ln): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
      (ff_self): Sequential(
        (0): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
        (1): Linear(in_features=128, out_features=128, bias=True)
        (2): GELU(approximate=none)
        (3): Linear(in_features=128, out_features=128, bias=True)
      )
    )
  )
)
(bottom): Sequential(
  (0): WConv(
    (wconv): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (1): GroupNorm(1, 256, eps=1e-05, affine=True)
      (2): GELU(approximate=none)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (4): GroupNorm(1, 256, eps=1e-05, affine=True)
    )
  )
  (1): WConv(
    (wconv): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (1): GroupNorm(1, 256, eps=1e-05, affine=True)
      (2): GELU(approximate=none)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (4): GroupNorm(1, 256, eps=1e-05, affine=True)
    )
  )
```

```
    (2): WConv(
      (wconv): Sequential(
        (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): GroupNorm(1, 128, eps=1e-05, affine=True)
        (2): GELU(approximate=none)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): GroupNorm(1, 128, eps=1e-05, affine=True)
      )
    )
  )
  (up_layers): ModuleList(
    (0): U_Up(
      (upsample): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (conv): Sequential(
        (0): WConv(
          (wconv): Sequential(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): GroupNorm(1, 256, eps=1e-05, affine=True)
            (2): GELU(approximate=none)
            (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): GroupNorm(1, 256, eps=1e-05, affine=True)
          )
        )
        (1): WConv(
          (wconv): Sequential(
            (0): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): GroupNorm(1, 64, eps=1e-05, affine=True)
            (2): GELU(approximate=none)
            (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): GroupNorm(1, 64, eps=1e-05, affine=True)
          )
        )
      )
```

```
    (1): U_Up(
      (upsample): ConvTranspose2d(64, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (conv): Sequential(
        (0): WConv(
          (wconv): Sequential(
            (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): GroupNorm(1, 128, eps=1e-05, affine=True)
            (2): GELU(approximate=none)
            (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): GroupNorm(1, 128, eps=1e-05, affine=True)
          )
        )
        (1): WConv(
          (wconv): Sequential(
            (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): GroupNorm(1, 64, eps=1e-05, affine=True)
            (2): GELU(approximate=none)
            (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): GroupNorm(1, 64, eps=1e-05, affine=True)
          )
        )
      )
      (emb_layer): Sequential(
        (0): SiLU()
        (1): Linear(in_features=256, out_features=64, bias=True)
      )
```

```
      (attention): SelfAttention(
        (mha): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
        )
        (ln): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (ff_self): Sequential(
          (0): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
          (1): Linear(in_features=64, out_features=64, bias=True)
          (2): GELU(approximate=none)
          (3): Linear(in_features=64, out_features=64, bias=True)
        )
      )
    )
  )
  (out): Conv2d(64, 3, kernel_size=(1, 1), stride=(1, 1))
)
```

Model architecture 就是 Unet (修改自 https://github.com/dome272/Diffusion-Models-pytorch)，內部共做了 2 + 2 次 down sample + up sample，期間利用 position encoding 把 timestep 與 label 資訊 encode 至 model 內，整體就是一個 transformer。Diffusion 的過程則採用 cosine schedule 增加 noise，然後透過 ema 更新參數，training 過程使用小的 batch size = 32 以及 learning rate = 2×10^{-5}有較好的表現

(2)

(3)

(4)

Diffusion model 生成出的圖片品質非常好，但是 sample 需要花費的時間也較多。撇除掉背後的數學原理，diffusion 的過程實作上並無太大困難，反倒是 Unet 的搭建以及如何把 timestep 與 label 資訊 encode 進 model 內的部分花了較多心力

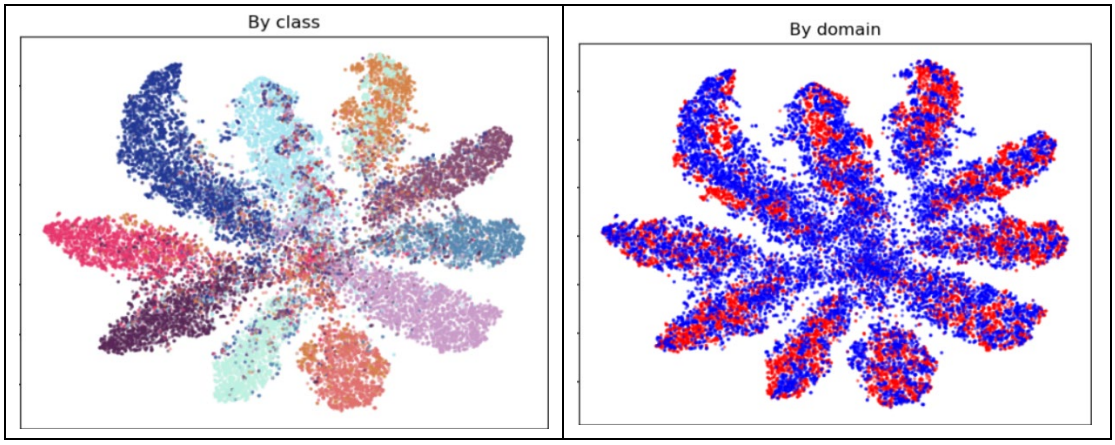
Problem 3.

(1)

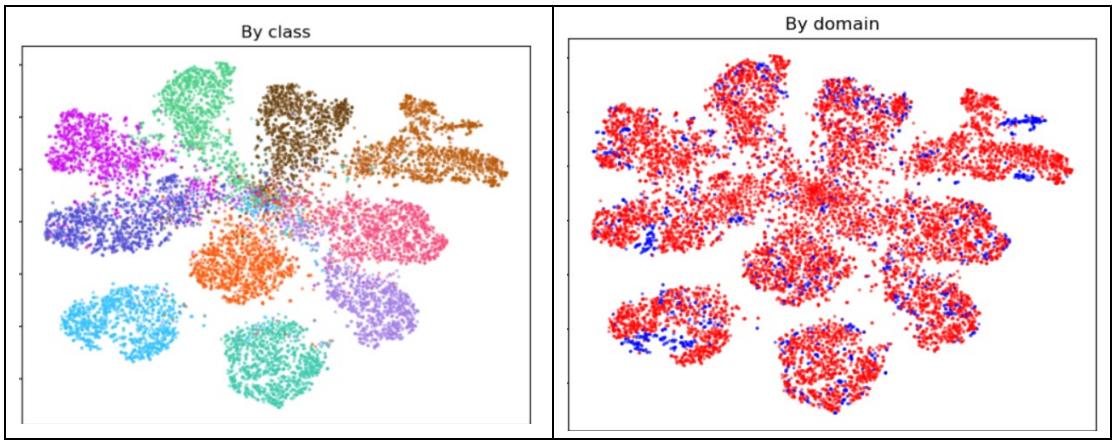
	MNIST-M→SVHN	MNIST-M→USPS
Trained on sourse	35.50%	74.40%
Adaption	61.57%	89.05%
Trained on target	94.15%	99.26%

(2)

MNIST-M→SVHN



MNIST-M→USPS



(3)

實作上 Feature extractor 與 label predictor 就是一般 CNN 的前半與後半部分，以 Fully connected layer 為分界，而 Domain Classifier 則就是單純的幾層 Fully connected layer，但 DANN 的 training 流程就與一般的 Supervise learning 很不一樣，它同時使用到兩種資料集(source/target domain)訓練，並使用不同的 loss 更新 3 個部份的參數，在做 back propagation 就要注意模型梯度的流向。