2-1

1. model A

```
Generater: Generator(
  (main): 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): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.2, inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
########################################
Discrminator Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): ReLU(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): ReLU(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): ReLU(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): ReLU(inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

modelB

```
Generater: Generator(
  (main): 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=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
########################################
Discrminator Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): ReLU(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): ReLU(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): ReLU(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): ReLU(inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
```

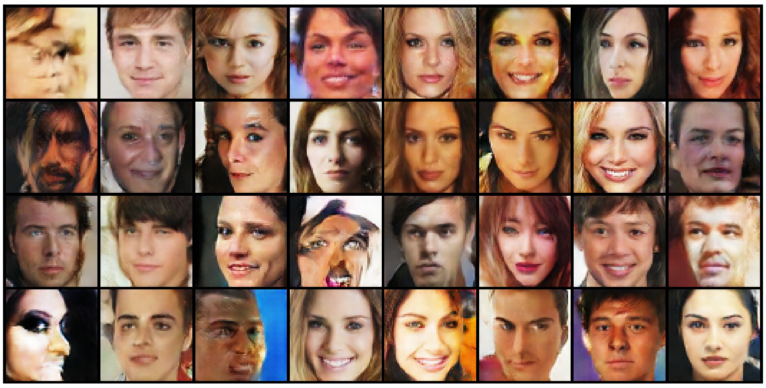
2.

model A及model B的模型架構幾乎相同，而在model A2的activation function是採用leackyrelu而modelB則是relu，另外在採用modelB時我將input data多做了一個tanh的處理。

從下面兩組圖發現，model A的人臉較模糊甚至有些人臉是五官很不清楚的，而model B的影像都相當清晰，只有幾張人臉比較畸形。

model A

model B



3.

在本次訓練時有參考助教提供的一個github教如何訓練好的GAN，也又照內容修改程式， 包含將label不要只設成0,1，而是設成0-0.3,0.7-1.2之類，或是將activation function調整成都是leackyrelu，或是將optimizer_G用adam而optimizer_D用SGD等等的方式，然而有些方法有幫助，有些反而更糟(把label設成特定範圍的隨機數字)，但最後發現將learning rate下降後就得到相當不錯的效果，並微調model結構即可。

2-2

1.

```
UNet_conditional(
  (inc): DoubleConv(
    (double_conv): 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)
    )
  )
  (down1): Down(
    (maxpool_conv): Sequential(
      (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (1): DoubleConv(
        (double_conv): 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): DoubleConv(
        (double_conv): 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)
```

```
  (sa1): 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)
    )
  )
  (down2): Down(
    (maxpool_conv): Sequential(
      (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (1): DoubleConv(
        (double_conv): 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): DoubleConv(
        (double_conv): 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)
        )
      )
    )
```

```
  (emb_layer): Sequential(
    (0): SiLU()
    (1): Linear(in_features=256, out_features=256, bias=True)
  )
)
(sa2): SelfAttention(
  (mha): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256, bias=True)
  )
  (ln): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (ff_self): Sequential(
    (0): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (1): Linear(in_features=256, out_features=256, bias=True)
    (2): GELU(approximate=none)
    (3): Linear(in_features=256, out_features=256, bias=True)
  )
)
(down3): Down(
  (maxpool_conv): Sequential(
    (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (1): DoubleConv(
      (double_conv): 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): DoubleConv(
      (double_conv): 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)
```

```
  (emb_layer): Sequential(
    (0): SiLU()
    (1): Linear(in_features=256, out_features=256, bias=True)
  )
)
(sa3): SelfAttention(
  (mha): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256, bias=True)
  )
  (ln): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (ff_self): Sequential(
    (0): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (1): Linear(in_features=256, out_features=256, bias=True)
    (2): GELU(approximate=none)
    (3): Linear(in_features=256, out_features=256, bias=True)
  )
)
(bot1): DoubleConv(
  (double_conv): 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)
  )
)
(bot3): DoubleConv(
  (double_conv): 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)
  )
)
```

```
(up1): Up(
  (up): Upsample(scale_factor=2.0, mode=bilinear)
  (conv): Sequential(
    (0): DoubleConv(
      (double_conv): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): GroupNorm(1, 512, eps=1e-05, affine=True)
        (2): GELU(approximate=none)
        (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): GroupNorm(1, 512, eps=1e-05, affine=True)
      )
    )
    (1): DoubleConv(
      (double_conv): Sequential(
        (0): Conv2d(512, 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, 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)
  )
)
(sa4): 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)
```

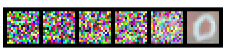
```
    (2): GELU(approximate=none)
    (3): Linear(in_features=128, out_features=128, bias=True)
  )
)
(up2): Up(
  (up): Upsample(scale_factor=2.0, mode=bilinear)
  (conv): Sequential(
    (0): DoubleConv(
      (double_conv): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 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): DoubleConv(
      (double_conv): 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, 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)
  )
)
(sa5): 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)
    )
  )
  (up3): Up(
    (up): Upsample(scale_factor=2.0, mode=bilinear)
    (conv): Sequential(
      (0): DoubleConv(
        (double_conv): 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): DoubleConv(
        (double_conv): 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)
    )
  )
```

```
  (sa6): 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)
    )
  )
  (outc): Conv2d(64, 3, kernel_size=(1, 1), stride=(1, 1))
  (label_emb): Embedding(10, 256)
)
```

2.

3.



4.

一開始將input image resize成64*64並丟進去模型訓練，出來的結果非常好，丟進去classifier回傳的準確度是1，但輸出1000張照片卻要非常久的時間，後來將輸出照片的程式都改成用numpy去處理，速度提昇約4倍，但還是超出規定的15分鐘，後來將input size改成16*16後，效果沒原先的model好，但還是能過baseline且輸出照片的時間約10分鐘就能完成。還有在sample照片時，如果將condition model的output資訊結合uncondition model的output資訊，能讓輸出照片有更好的效果。

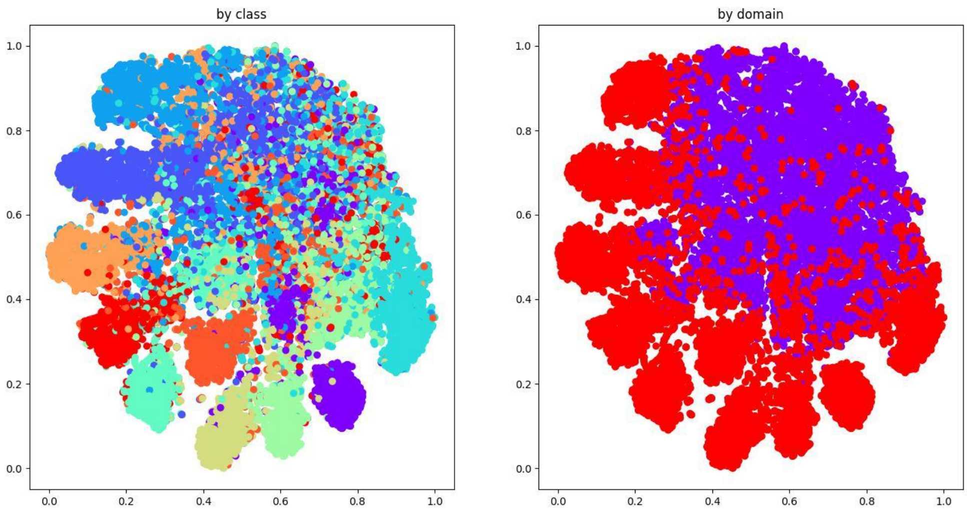
2-3

1.

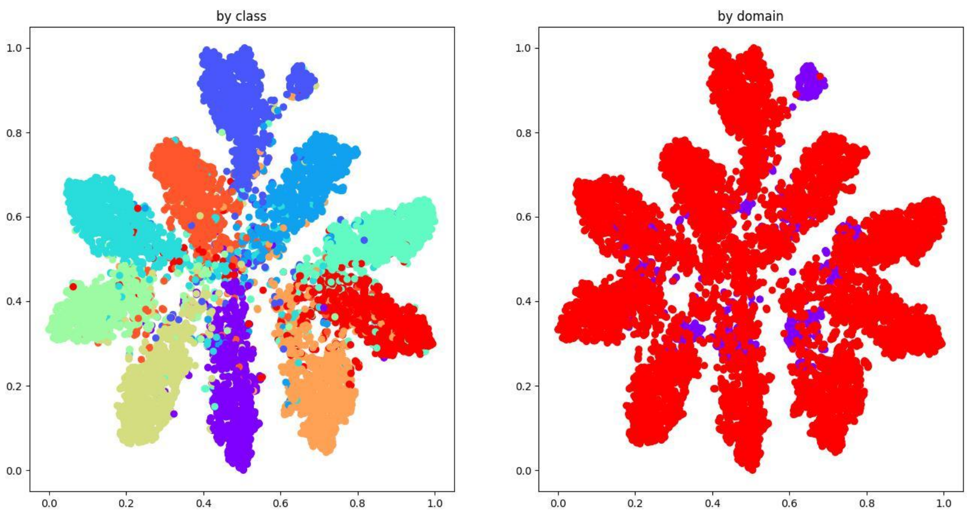
	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	`accuracy for svhn: 0.31258261471643484`	`accuracy for usps: 0.7083333333333334`
Adaptation (DANN)	`accuracy for svhn: 0.46062818656763393`	`accuracy for usps: 0.782258064516129`
Trained on target	`accuracy for svhn: 0.869390067350664`	`accuracy for usps: 0.9663978494623656`

2.

SVHN



USPS



3.

訓練DANN相較前面兩題容易許多，模型架構類似於普通CNN只是多了要分類domain的分類器。也有可能是手寫數字比較容易分辨，所以在訓練時epoch不用調太高，效果就已經很不錯了。從之後單純用CNN訓練source data或是target data的結果來看，確實 DANN的準確度比在用target data訓練的CNN表現來的差，而比用source data訓練的CNN表現的好，結果與一開始我自己的猜測相同。