DLCV HW2 Report

Student ID: R10943117
Name: 陳昱仁

Problem 1.

1. Please print the model architecture of method A and B.

```
Generator(
  (layer): 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()
  )
)
Discriminator_DC(
  (layer): 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()
  )
)
```

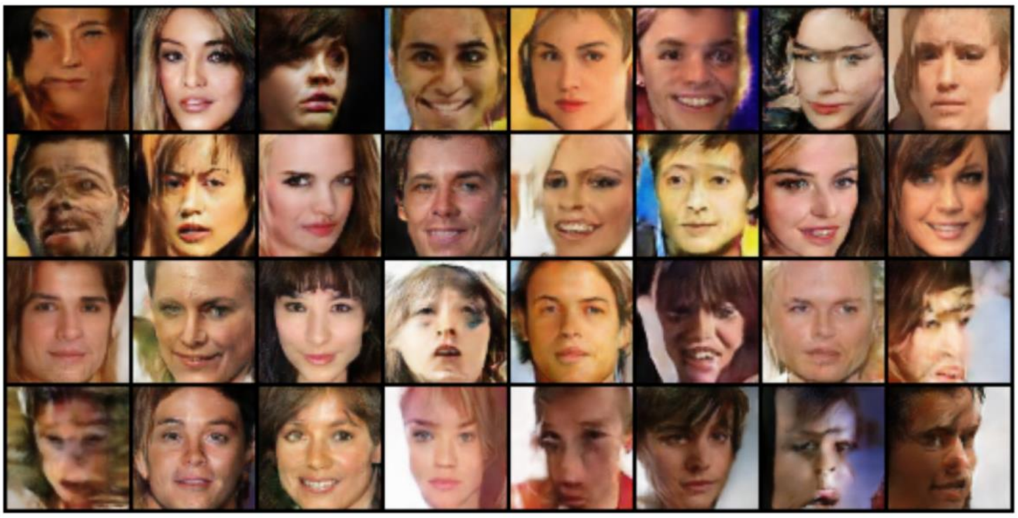
Method A

```
Generator(
  (layer): 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()
  )
)
Discriminator_W(
  (layer): 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): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (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): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (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): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  )
)
```

Method B

2. Please show the first 32 generated images of both method A and B then discuss the difference between method A and B.

Method B WGAN-GP 和 Method A DCGAN 相比，判別器最後一層拿掉 sigmoid，loss 拿掉 log，loss 多了 GP 項，讓訓練更穩定，也減少 mode collapse，但 DCGAN 的結果看起來比 WGAN-GP 還要好，method B 的圖有些地方還會有雜訊，模糊的情形也比 method A 還明顯，從量化的數據來看，A 及 B 的 FID 分別為 22.97 及 41.03，數據和視覺化的結果一致，A 表現較佳，原因應該是 B 的參數較不好調整，在這次 case 中不容易調到比 A 還好的結果。



Method A

Method B

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

GAN 的訓練和其他 task 相比相當困難，且收斂較為緩慢，paper 上常常都是訓練數萬個 epoch，不容易訓練出好的模型，另外實驗中加了 data augmentation 後結果甚至更差。WGAN-GP 理論上應該比 DCGAN 還要更強，但參數不好調正，結果比 DCGAN 更差，最後繳交的版本為 DCGAN 且沒使用任何 data augmentation。

Problem 2.

1. Please print your model architecture and describe your implementation details.

Model 本身較大是 Unet 再加上 time 和 label 的 embedding，model 中間會有許多 resblock，使用的 optimizer 是 Adam，lr = 0.0002，betas = (0.9，0.999)，lr 每 20 epoch 會變成原本 0.8 倍，資料會 normalize 到-1 和 1 之間，beta 介於 0.0001 到 0.02 之間，time step 總共為 400，loss 使用 MSE loss。

```
UNet(
  (time_embedding): TimeEmbedding(
    (timembedding): Sequential(
      (0): Embedding(400, 128)
      (1): Linear(in_features=128, out_features=512, bias=True)
      (2): Swish()
      (3): Linear(in_features=512, out_features=512, bias=True)
    )
  )
  (cond_embedding): ConditionalEmbedding(
    (condEmbedding): Sequential(
      (0): Embedding(11, 128, padding_idx=0)
      (1): Linear(in_features=128, out_features=512, bias=True)
      (2): Swish()
      (3): Linear(in_features=512, out_features=512, bias=True)
    )
  )
  (head): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (downblocks): ModuleList(
    (0): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
```

```
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Identity()
      (attn): AttnBlock(
        (group_norm): GroupNorm(32, 128, eps=1e-05, affine=True)
        (proj_q): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        (proj_k): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        (proj_v): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        (proj): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
      )
    )
    (1): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Identity()
```

```
    (attn): AttnBlock(
      (group_norm): GroupNorm(32, 128, eps=1e-05, affine=True)
      (proj_q): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
      (proj_k): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
      (proj_v): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
      (proj): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (2): DownSample(
    (c1): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (c2): Conv2d(128, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
  )
  (3): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 128, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): AttnBlock(
      (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
      (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (4): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Identity()
    (attn): AttnBlock(
      (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
      (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (5): DownSample(
    (c1): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (c2): Conv2d(256, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
  )
  (6): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Identity()
    (attn): AttnBlock(
      (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
      (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
  )
```

```
    (7): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Identity()
      (attn): AttnBlock(
        (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
        (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      )
    )
    (8): DownSample(
      (c1): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (c2): Conv2d(256, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    )
    (9): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Identity()
      (attn): AttnBlock(
        (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
        (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      )
    )
    (10): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Identity()
      (attn): AttnBlock(
        (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
        (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      )
    )
  )
  (middleblocks): ModuleList(
    (0): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
```

```
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Identity()
    (attn): AttnBlock(
      (group_norm): GroupNorm(32, 256, eps=1e-05, affine=True)
      (proj_q): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_k): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj_v): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (proj): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (1): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Identity()
    (attn): Identity()
  )
)
(upblocks): ModuleList(
  (0): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): Identity()
  )
  (1): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): Identity()
  )
  (2): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
```

```
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): Identity()
  )
  (3): UpSample(
    (c): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (t): ConvTranspose2d(256, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1))
  )
  (4): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
```

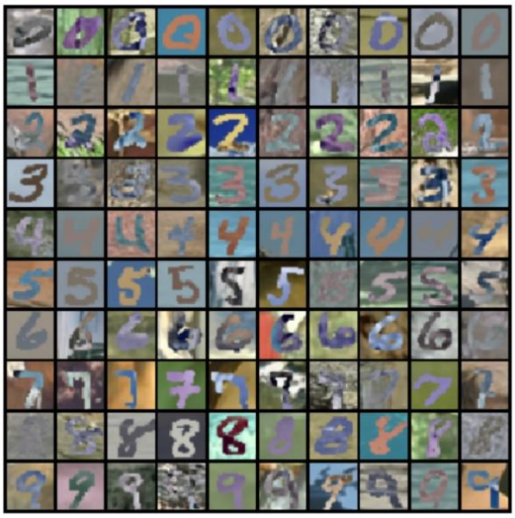
```
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): Identity()
  )
  (5): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): Identity()
  )
```

```
  (6): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (cond_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
    )
    (block2): Sequential(
      (0): GroupNorm(32, 256, eps=1e-05, affine=True)
      (1): Swish()
      (2): Dropout(p=0.1, inplace=False)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (attn): Identity()
  )
  (7): UpSample(
    (c): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (t): ConvTranspose2d(256, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1))
  )
  (8): ResBlock(
    (block1): Sequential(
      (0): GroupNorm(32, 512, eps=1e-05, affine=True)
      (1): Swish()
      (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
    (temb_proj): Sequential(
      (0): Swish()
      (1): Linear(in_features=512, out_features=256, bias=True)
```

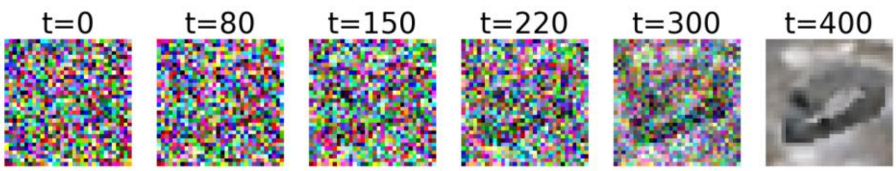
```
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
      (attn): Identity()
    )
    (9): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 512, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
      (attn): Identity()
    )
    (10): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 384, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=256, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Conv2d(384, 256, kernel_size=(1, 1), stride=(1, 1))
      (attn): Identity()
    )
    (11): UpSample(
      (c): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (t): ConvTranspose2d(256, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1, 1))
    )
    (12): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 384, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(384, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1))
      (attn): Identity()
    )
    (13): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
      (attn): Identity()
    )
```

```
    (14): ResBlock(
      (block1): Sequential(
        (0): GroupNorm(32, 256, eps=1e-05, affine=True)
        (1): Swish()
        (2): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (temb_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (cond_proj): Sequential(
        (0): Swish()
        (1): Linear(in_features=512, out_features=128, bias=True)
      )
      (block2): Sequential(
        (0): GroupNorm(32, 128, eps=1e-05, affine=True)
        (1): Swish()
        (2): Dropout(p=0.1, inplace=False)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (shortcut): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
      (attn): Identity()
    )
  )
  (tail): Sequential(
    (0): GroupNorm(32, 128, eps=1e-05, affine=True)
    (1): Swish()
    (2): Conv2d(128, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
```

2. **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.**



3. **Visualize total six images in the reverse process of the first “0” in your grid in (2) with different time steps.**



4. **Please discuss what you’ve observed and learned from implementing conditional diffusion model.**

實驗中發現把 time step 調整越大，效果會越好，但收斂速度較慢，condition 放入 model 中和 ACGAN 相似，都是先將此 condition embedding，除此之外還需要將 timestep embedding，再一併丟入 Unet 中。

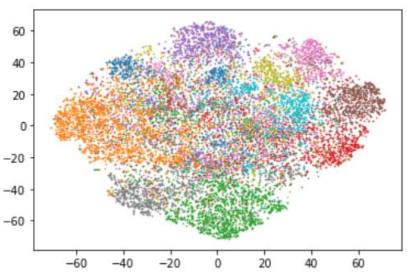
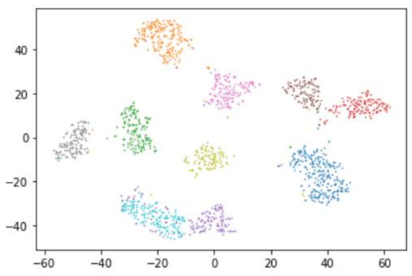
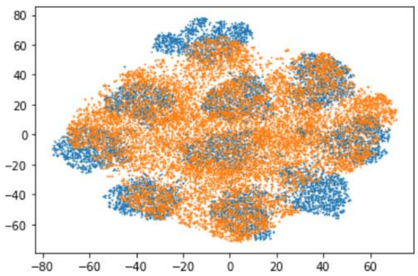
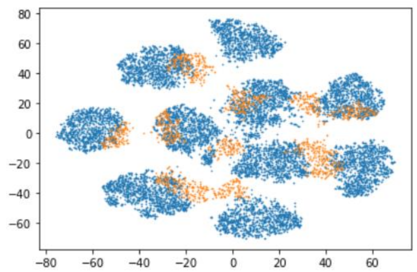
令人比較值得注意的是，一般的模型都是希望訓練時間久，但在 inference 使用上可以快速得到結果，但 diffusion model 的 inference 時間極為緩慢，time step 越大時間越久，和一般的模型很不一樣，此為 diffusion model 的缺點。

Problem 3.

1. **Please create and fill the table.**

	MNIST-M -> SVHN	MNIST-M -> USPS
Trained on source	34.4%	75.3%
Adaptation (DANN)	51.1%	87.3%
Trained on target	91.0%	98.3%

2. **Please visualize the latent space 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.**

	MNIST-M -> SVHN	MNIST-M -> USPS
By class		
By domain		

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

Model 的部分 feature extract 使用兩層 CNN 加上一層 linear 來抽取 feature，feature 在分支到 class 及 domain classifier，兩個分類器皆是用兩層 linear。Loss 皆是使用 CrossEntropyLoss，optimizer 使用 Adam，lr = 0.0002，betas = (0.9, 0.999)，lr 每 20 epoch 會變成原本的 0.8 倍，並做許多 data augmentation 如下圖。

這次的 task 在訓練 MNIST-M -> SVHN 會相較困難，從上方表格 upper bound 可知，SVHN 這份 dataset 本身就相較困難，DANN 原本準度只有 30 幾左右，但加上許多 data augmentation，增加 source domain 資料多樣性，就輕易可以達到 baseline 標準，因此可知 data augmentation 在 DANN 中是個重要提高性能的技巧。

```
transforms.ColorJitter(brightness=0.2, contrast=0.3, saturation=0.2, hue=0.3),
transforms.RandomGrayscale(),
transforms.RandomAdjustSharpness(2),
transforms.RandomPosterize(3),
transforms.RandomRotation((-15, 15)),
```