Homework #3

Problem 1: GAN (20%)

1. Describe the architecture & implementation details of your model. (5%)
   參考[1]與助教在 hw3_pdf 給的提示：將 pooling layers 換成 strided convolutions、在 generator, discriminator 都加上 batch normalization、generator 使用 ReLU activations , discriminator 使用 LeakyReLU activations

```
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()
  )
)
```

```
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()
  )
)
```

2. Plot 32 random images generated from your model. [fig1_2.jpg] (10%)

3. Discuss what you've observed and learned from implementing GAN. (5%)
   發現如果像原先 GAN 使用 FC layer 來 implement Generator 和 Discriminator 的話，效果頗糟。把 FC layer 換成 Convolutional layer 會好許多。雖然大致上可以生成人臉圖，但是某些人臉還是會出現 distort 的情況。

Problem 2: ACGAN (20%)

1. Describe the architecture & implementation details of your model. (5%)
   ACGAN 的 Generator 架構基本上與 DCGAN 的差不多，只多加了一維 input channel。(原來如果單純 input noise 是 100 維，加上一維的 condition，變成 101 維)
   Discriminator 的架構也與 DCGAN 相近，只不過要在原來的 adv_layer 外再加上一個 aux_layer 來 classify 不同的 class。

```
ACGAN_Generator(
  (main): Sequential(
    (0): ConvTranspose2d(101, 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()
  )
)
```

```
ACGAN_Discriminator(
  (conv_blocks): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout2d(p=0.25, inplace=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Dropout2d(p=0.25, inplace=False)
    (6): BatchNorm2d(32, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Dropout2d(p=0.25, inplace=False)
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (11): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): LeakyReLU(negative_slope=0.2, inplace=True)
    (13): Dropout2d(p=0.25, inplace=False)
    (14): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=2048, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=2048, out_features=2, bias=True)
    (1): Softmax(dim=None)
  )
)
```

2. Plot 10 random pairs of generated images from your model, where each pair should be generated from the same random vector input but with opposite attribute. This is to demonstrate your model's ability to disentangle features of interest. [fig2_2.jpg] (10%)

   上排的是 no smiling，下排的是有 smiling，可以觀察到的確有笑容的差異。

3. Discuss what you've observed and learned from implementing ACGAN. (5%)
   我一開始在做 ACGAN 的時候，是讓 Generator 維持原架構，把 condition 用 dot product 的方式乘以原來的 noise，只有把 Discriminator 加上 aux_layer，但是出來的效果很糟。後來把 condition 用 concatenate 的方式加到原 noise 上，並讓 Generator 加上一維，才順利產生出結果。

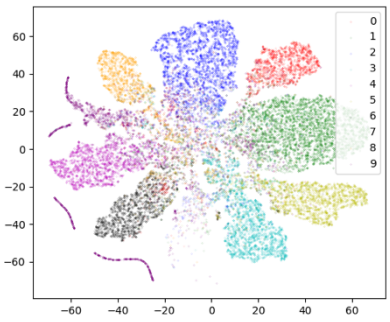
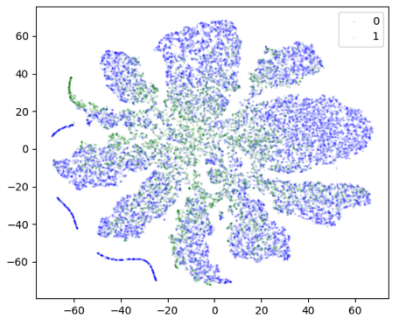
Problem 3: DANN (35%)

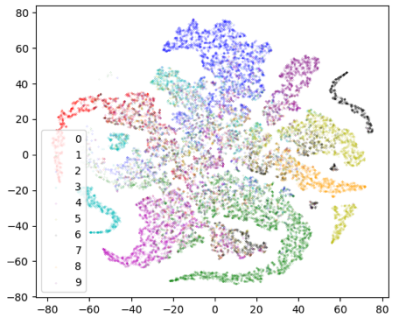
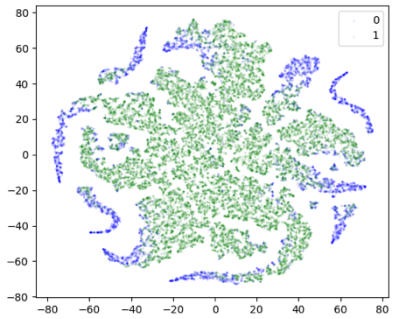
(1) MNIST-M -> SVHN (2) SVHN -> MNIST-M

1., 2., 3. Compute accuracy

	SVHN -> MNIST-M	MNIST-M -> SVHN
Trained on source	5173/10000 (51.73%)	11822/26032 (45.41%)
Adaptation	6286/10000 (62.86%)	14547/26032 (55.88%)
Trained on target	9725/10000 (97.25%)	24056/26032 (92.40%)

4. Visualize the latent space by mapping the testing images to 2D space (with t-SNE) and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (source/target). (6%)

	Digit classes	domains
SVHN -> MNIST-M		

MNIST-M -> SVHN		

5. Describe the architecture & implementation detail of your model. (6%)
   DANN 由三個部分組成：feature extractor, label predictor, domain classifier
   Feature extractor 的部分我是像 hw2 一樣，使用在 ImageNet 上 pretrained 好的 resnet 18。Label predictor 及 domain classifier 則是使用 linear, batch norm, relu 這樣的疊法，不同的是 label predictor 多疊了一層，且 label predictor 最後輸出 10 個 classes，而 domain classifier 則輸出 2 個。
   另外，domain classifier 需要有 gradient reversal layer，這部分的 implement 我是參考[2]。
   *Note: 因為 feature extractor 我是直接使用 torchvision 的 resnet18，所以就不再貼架構圖了

```
Label_Predictor(
  (model): Sequential(
    (0): Linear(in_features=512, out_features=100, bias=True)
    (1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=100, out_features=100, bias=True)
    (4): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=100, out_features=10, bias=True)
  )
)
```

```
Domain_Classifier(
  (model): Sequential(
    (0): Linear(in_features=512, out_features=100, bias=True)
    (1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=100, out_features=2, bias=True)
  )
)
```

6. Discuss what you've observed and learned from implementing DANN. (7%)
   在 implement 的過程中，遇到的比較大困難是 gradient reversal layer 的實作以及 label predictor、domain classifier 分別該怎麼疊。最後是參考[2]的方式來解決此困難。 還有就是 learning rate 的調整，一開始調太大，導致 accuracy 不高、adaptation 的效果也不明顯，慢慢試了幾個更小的 learning

rate 才好轉。

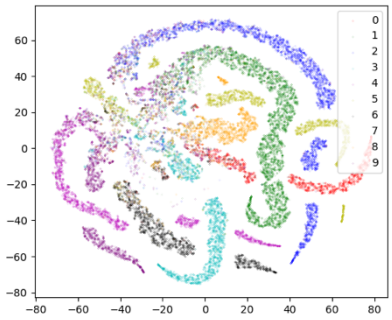
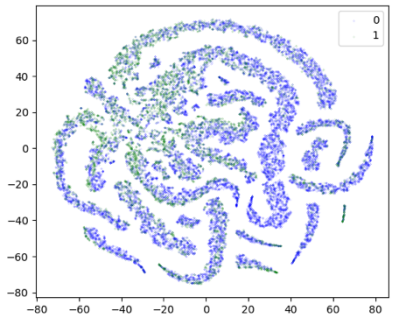
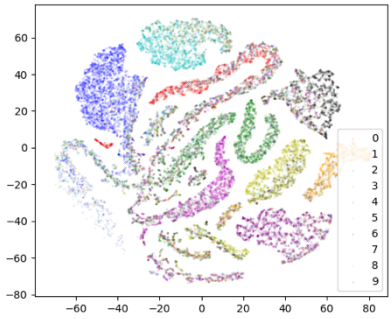
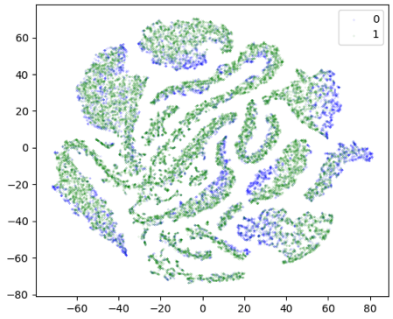
Problem 4: Improved UDA model (35%)

(1) MNIST-M -> SVHN (2) SVHN -> MNIST-M

1. Compute domain adaptation accuracy

	SVHN -> MNIST-M	MNIST-M -> SVHN
Adaptation	6759/10000 (67.59%)	14512/26032 (55.74%)

2. Visualize the latent space by mapping the testing images to 2D space (with t-SNE) and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (source/target). (6%)

	Digit classes	domains
SVHN -> MNIST-M		
MNIST-M -> SVHN		

3. Describe the architecture & implementation detail of your model. (6%)
助教提供的五個 improved UDA model 中，我選擇 implement 第一個 ”Adversarial Discriminative Domain Adaptation. CVPR 2017”。因為在看過 paper 裡面描述的架構後，我覺得這是與上一題 DANN model 架構最相近的。也就是說，feature extractor, label predictor 我都可以直接使用上一題 implement 好的，只要再加上一個 domain discriminator，不用原先的 gradient reversal layer，改用 train discriminator, generator 的想法，將 source domain label 跟 target domain label 調換著 train 即可。因為 feature extractor 與 label predictor 與上一題的架構一模一樣，所以只貼上新增的 discriminator

的架構：

```
ADDA_Discriminator(
  (layer): Sequential(
    (0): Linear(in_features=512, out_features=100, bias=True)
    (1): ReLU()
    (2): Linear(in_features=100, out_features=20, bias=True)
    (3): ReLU()
    (4): Linear(in_features=20, out_features=1, bias=True)
  )
)
```

4. Discuss what you've observed and learned from implementing improved UDA model. (7%)

   一開始我使用跟上一題基本上一樣的參數設定，但是 train 起來的結果超級糟糕！以為是自己 implement 錯了，所以測試了很多不同的 open source code，但都沒有明顯的好轉。本來想放棄了，後來決定再把 learning rate 調的更小(0.00001, 0.000002)，結果 accuracy 就上來了！

Reference:

[1] https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[2] https://github.com/wogong/pytorch-dann

[3] https://github.com/jvanvugt/pytorch-domain-adaptation