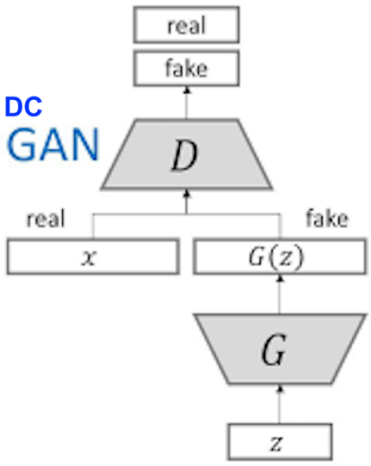
Name: 李尚倫　Dep.:電機碩一　Student ID:R07921001

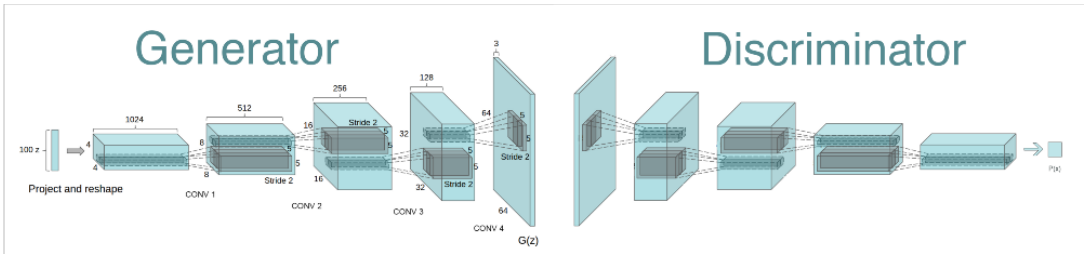
Problem 1: GAN (20%)

1. Describe the architecture & implementation details of your model. (5%)

I implement the DCGAN. Refer to paper[2] and pytorch official tutorial[1] :



←Main Structure ↓Generator and Discriminator




```
(robotpower) → dcgan git:(master) ✗ python train.py
# images in trainset: 40000
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)
    (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)
    (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)
    (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)
    (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)
    (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)
    (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)
    (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)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

(1) Weight Initialization: 原作者有提到，all model weights shall be randomly initialized from a Normal distribution with mean=0, stdev=0.02，故我的source code裡有一個weight_init function，在model load進來時即執行。

(2) Generator: 主要是要接受一個input latent space vector z (100維)，輸出3x64x64的彩圖。實作細節則照著論文，一層project and reshape後接四層conv，每層的元素都是先convtranspose2d後batchnorm2d然後ReLU，如果不接batchnorm2d的話會train不起來。然後最後一層接tanh layer 得到都是[-1,1]值的矩陣就是我們要的圖片了。也因為這樣在load data時要normalize成mean=(0.5,0.5,0.5), std=(0.5,0.5,0.5)，這樣input也才會是[-1,1]值的矩陣。最後在畫結果的時候，記得在把normalize設成true即可得到正確的顏色。

(3) Discriminator:
主要是要接受一個3x64x64的圖片來判斷真假。實作細節則是如上圖，和generator相反的大小conv回去，每層一樣是先convtranspose2d後batchnorm2d然後ReLU，然後在最後一層用加入sigmoid activation function激活。

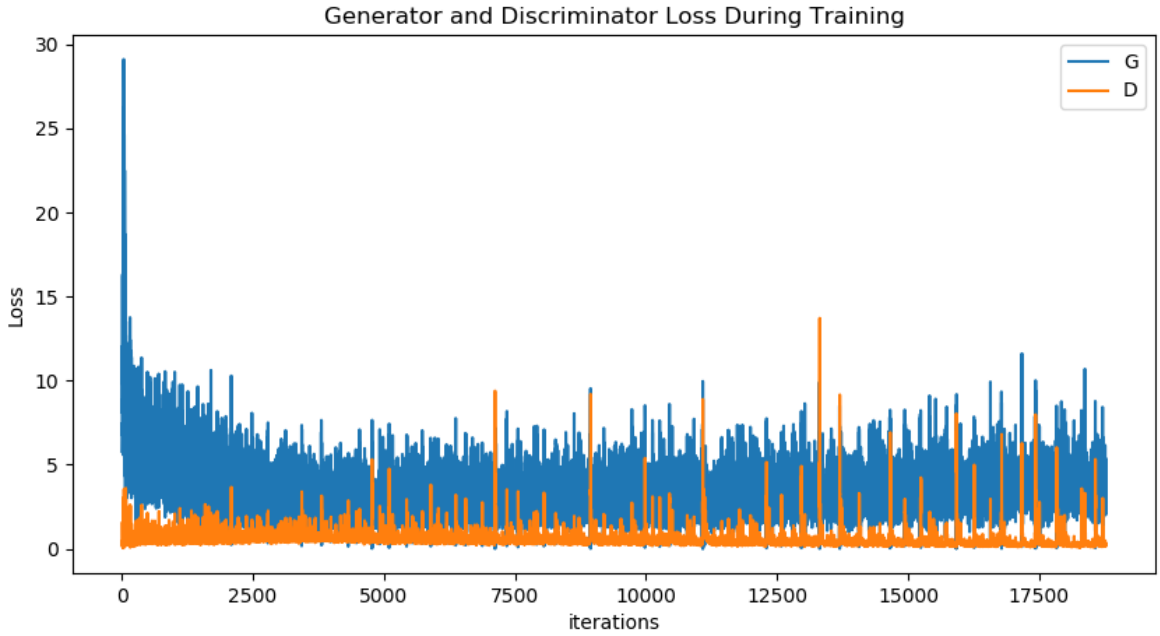
(4) Loss function and optimizer: 選用Binary Cross Entropy(BCELoss)和Adam optimizer，lr=0.0002, Beta1=0.5。

(5) Training:
第一階段：Update D network: maximize discriminator loss
Loss_D - discriminator loss calculated as the sum of losses for the all real and all fake batches (log(D(x))+log(D(G(z))))
第二階段：Update G network: maximize generator loss
Loss_G - generator loss calculated as log(D(G(z)))

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%)

基本上關於實作的細節在第一題已經說明了，而下圖則為實際train的結果，G和D的loss對每個500個iteration(每train完一個batch為一個iteration)記錄一次。由這張圖可以觀察出其實model收斂的蠻快的loss很快就被壓下來。



但若實際以generator生成出來的圖來看，loss雖然同樣很低的iterations，但效果卻不同，基本上iteration<4000，效果都還有隨iteration次數變多而變好：

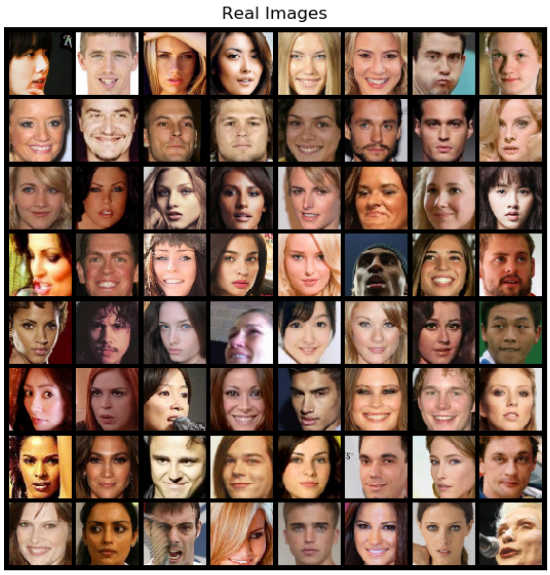
Iteration= (0, 500, 1000)， 屬於早期，圖片還很模糊，只有臉型

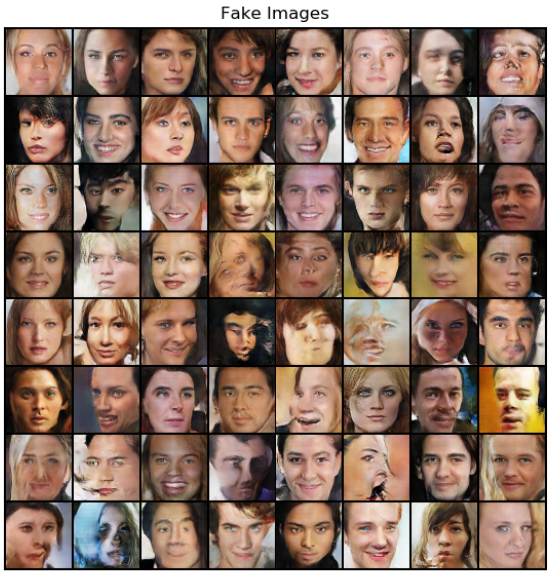
Iteration= (1500, 2000, 2500)， 屬於中期，圖片品質還是不太好，但五官已生成

Iteration= (3000, 3500, 4000)， 屬於後期，圖片品質已接近真實，但臉會扭曲

然後之後的Iteration結果其實大同小異，畫質基本上都達到和真實照片差不多，只是一組裡面臉是扭曲的比例佔多少而已，而最後我選在iteration=17000時的照片作為最後結果。

而下圖則是拿dataset的圖和generator生成的圖做比較，可以看出dataset裡的人臉有些也是很扭曲(因為角度很怪或是有道具入鏡)，所以fake images作出的效果其實算是很好的。

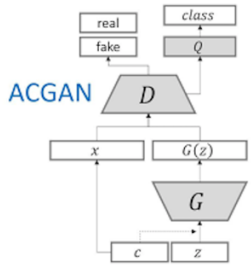




Problem 2: ACGAN (20%)

1. Describe the architecture & implementation details of your model. (5%)

Refer to paper[3] and pytorch implementation on other datasets[4][5]：



←Main Structure ↓Generator and Discriminator

```
(robotpower) → acgan git:(master) ✗ python train.py
Random Seed:  1077
# images in trainset: 40000
Image tensor in each batch: torch.Size([64, 3, 64, 64]) torch.float32
Label tensor in each batch: torch.Size([64]) torch.int64
Generator(
  (label_emb): Embedding(2, 100)
  (conv1): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1))
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv5): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (relu): ReLU(inplace)
  (tanh): Tanh()
)
Discriminator(
  (lrelu): LeakyReLU(negative_slope=0.2, inplace)
  (conv1): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv5): Conv2d(512, 64, kernel_size=(4, 4), stride=(1, 1))
  (gan_linear): Linear(in_features=64, out_features=1, bias=True)
  (aux_linear): Linear(in_features=64, out_features=2, bias=True)
  (sigmoid): Sigmoid()
)
```

(1) Weight Initialization and Generator:
基本上和dcgan的一模一樣，詳見問題一的第一題。

(2) Discriminator: 也和 dcgan類似，但為了做到auxiliary classifier多加了兩層linear，然後最後一樣接sigmoid activation function激活。

(3) Loss function and optimizer: 沿用 Binary Cross Entropy(BCELoss)作為 adversarial loss，而為了做到 auxiliary classifier 多類別分類，多使用了 Cross Entropy Loss。Optimizer 則和 dcgan 同樣為 Adam optimizer, lr=0.0002, Beta1=0.5。

(4) Training:

- **Overall objective function**

$$G^* = \arg\min_G \max_D \mathcal{L}_{GAN}(G, D) + \mathcal{L}_{cls}(G, D)$$

- **Adversarial Loss**

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}[\log(1 - D(G(z, c)))] + \mathbb{E}[\log D(y)]$$

- **Disentanglement loss**

$$\mathcal{L}_{cls}(G, D) = \underbrace{\mathbb{E}[-\log D_{cls}(c'|y)]}_{\text{Real data w.r.t. its domain label}} + \underbrace{\mathbb{E}[-\log D_{cls}(c|G(x, c))]}_{\text{Generated data w.r.t. assigned label}}$$

$$L_S = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})] \quad (2)$$

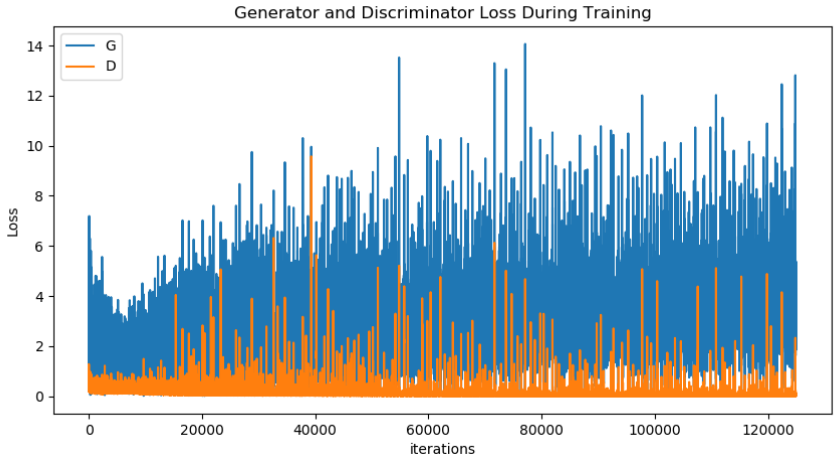
$$L_C = E[\log P(C = c \mid X_{real})] + E[\log P(C = c \mid X_{fake})] \quad (3)$$

D is trained to maximize $L_S + L_C$ while G is trained to maximize $L_C - L_S$. AC-GANs learn a representation for z that is independent of class label (e.g. (Kingma et al., 2014)).

先train Generator再train Discriminator。

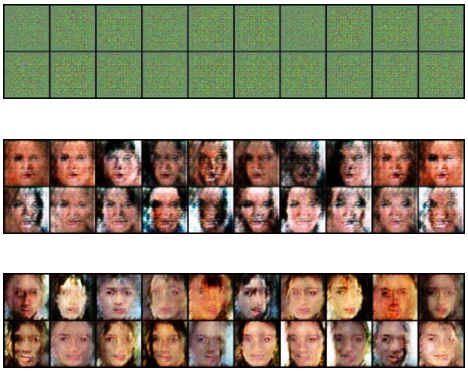
2. Plot 10 random pairs of generated images from your model, where each pair should be generated from the same random vector input but with opposite attribute. This is to demonstrate your model's ability to disentangle features of interest. [fig2_2.jpg] (10%)
我取的label為笑and不笑

3. Discuss what you've observed and learned from implementing ACGAN. (5%)
下圖則為實際train的結果，G和D的loss對每個400個iteration(每train完一個batch為一個iteration)記錄一次。由這張圖可以觀察出其實model收斂的蠻快的loss很快就被壓下來。且後面Generator的loss還有一點小成長。



但是，和dcgan的結果雷同，雖然loss一樣小，甚至後期generator的loss還變大，圖片的品質與效果，還是train較多iteration的較好。詳見以下圖：

Iteration= (0, 800, 1600)，屬於早期圖片還很模糊，只有臉型

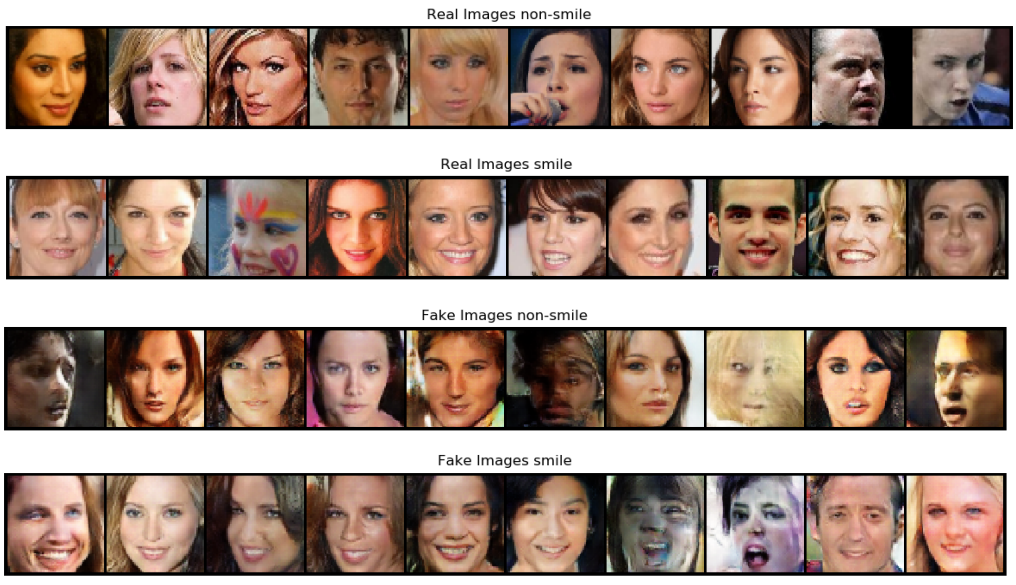


Iteration= (2000, 4000, 5200)，屬於中期圖片品質還是不太好，但五官已生成

Iteration= (8000, 9200)，屬於後期，Ganerator loss的最低點附近，圖片品質已接近真實，有些臉會扭曲，不過笑or不笑已經分的還不錯

然後之後的Iteration結果其實大同小異，畫質基本上都達到和真實照片差不多，只是一組裡面臉是扭曲的比例佔多少，和有沒有笑出來而已，而最後我選在iteration=99200時的照片作為最後結果。

而下圖則是拿dataset的圖和generator生成的圖做比較，可以看出dataset裡的人臉在笑的label有些也是只有嘴笑而已，沒有露齒笑，在不笑的label也有嘴角微微上揚的(最左邊第一張)，所以fake images作出的效果其實算是很好的皆有露齒或微笑。



Problem 3: DANN (35%)

In this problem, you need to implement DANN and consider the following 3 scenarios(source domain to target domain): USPS → MNIST-M, MNIST-M → SVHN, SVHN → USPS

1. Compute the accuracy on target domain, while the model is trained on source domain only. (lower bound) (3%)
   See the table below. Row1.

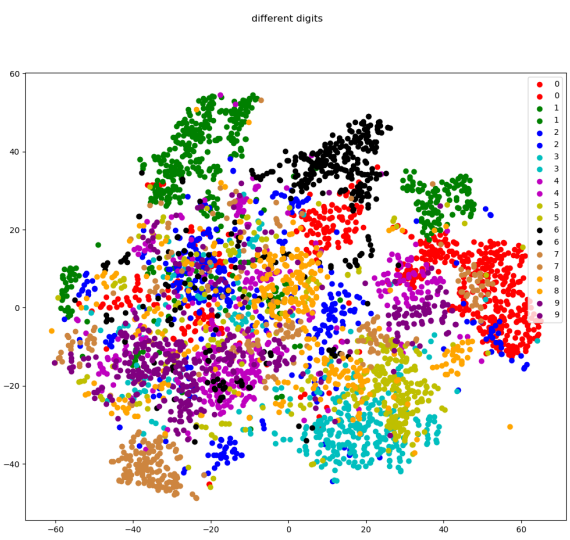
2. Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation) (3+7%)
   See the table below. Row2.

3. Compute the accuracy on target domain, while the model is trained on target domain only. (upper bound) (3%)
   See the table below. Row3.

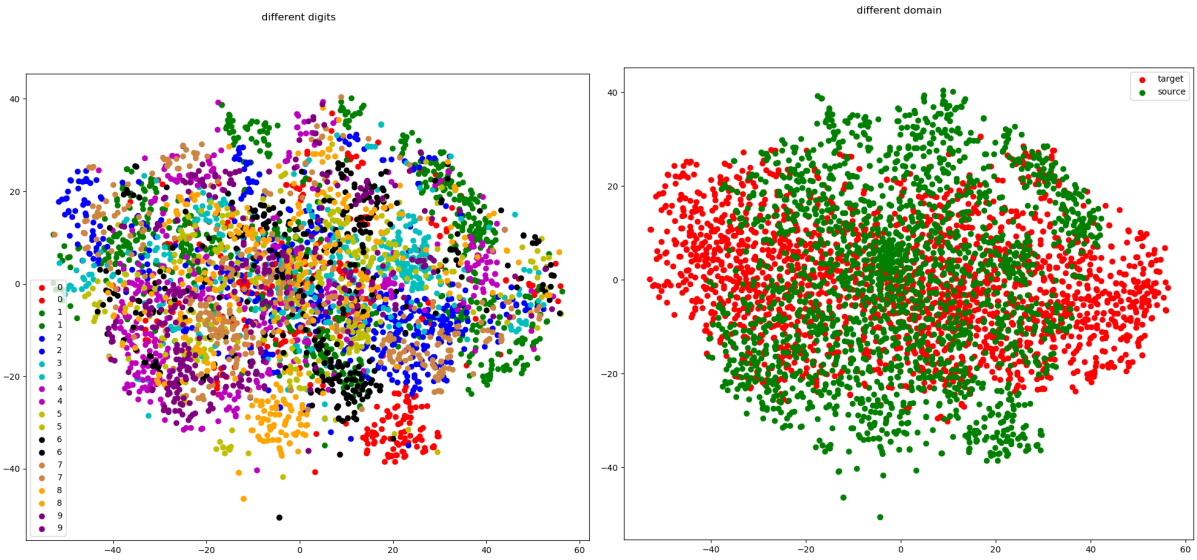
Total ep_nums = 50 (因為三者皆在 20 幾個就停止成長，因此不 train 到 100)

(accuracy, ep, model_name)	USPS → MNIST-M	MNIST-M → SVHN	SVHN → USPS
1.Trained on source (lower bound)	29.47%, ep03 models_UpreM/best.pth	38.24%, ep00 models_MpreS/best.pth	63.18%, ep44 models_SpreU/best.pth
2.Adaptation (DANN)	47.69%, ep20 models_UtoM/best.pth	52.42%, ep28 models_MtoS/best.pth	61.23%, ep6 models_StoU/best.pth
3.Trained on target (upper bound)	98.29%, ep41 models_MpreM/best.pth	92.67%, ep48 models_SpreS/best.pth	97.41%, ep36 models_UpreU/best.pth

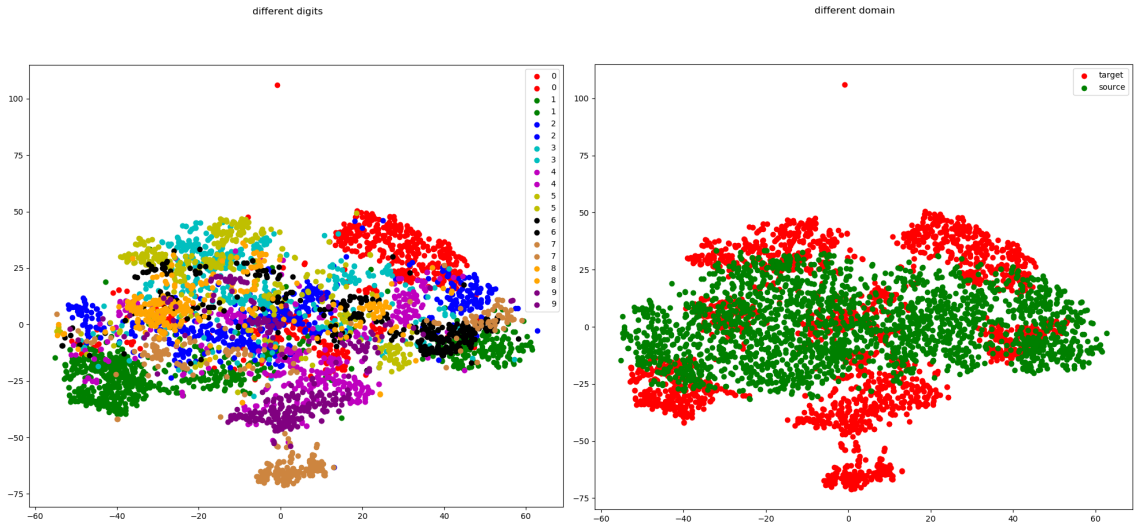
4. Visualize the latent space by mapping the testing images to 2D space (with t-SNE) and use different colors to indicate data of (a) different digit classes 0-9 and (b) different domains (source/target). (6%)
   Refer to sklearn tutorial[8] :
   (input source and target each 2000 image from test)
   UtoM(a)(b):



MtoS(a)(b):



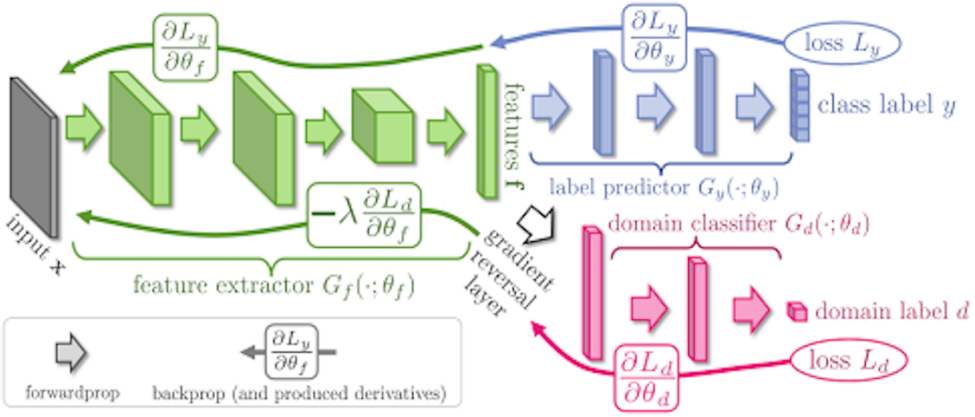
StoU(a)(b):



5. Describe the architecture & implementation detail of your model. (6%)

Refer to paper[6] and pytorch implementation on other datasets[7] :

↓ Main Structure



↓ Generator and Discriminator

```
# images in dataset_source: 73257
Image tensor in each batch: torch.Size([128, 3, 28, 28]) torch.float32
Label tensor in each batch: torch.Size([128]) torch.int64
# images in dataset_target: 7291
Image tensor in each batch: torch.Size([128, 3, 28, 28]) torch.float32
Label tensor in each batch: torch.Size([128]) torch.int64
CNNModel(
  (feature): Sequential(
    (f_conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (f_bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (f_pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (f_relu1): ReLU(inplace)
    (f_conv2): Conv2d(64, 50, kernel_size=(5, 5), stride=(1, 1))
    (f_bn2): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (f_drop1): Dropout2d(p=0.5)
    (f_pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (f_relu2): ReLU(inplace)
  )
  (class_classifier): Sequential(
    (c_fc1): Linear(in_features=800, out_features=100, bias=True)
    (c_bn1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (c_relu1): ReLU(inplace)
    (c_drop1): Dropout2d(p=0.5)
    (c_fc2): Linear(in_features=100, out_features=100, bias=True)
    (c_bn2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (c_relu2): ReLU(inplace)
    (c_fc3): Linear(in_features=100, out_features=10, bias=True)
    (c_softmax): LogSoftmax()
  )
  (domain_classifier): Sequential(
    (d_fc1): Linear(in_features=800, out_features=100, bias=True)
    (d_bn1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (d_relu1): ReLU(inplace)
    (d_fc2): Linear(in_features=100, out_features=2, bias=True)
    (d_softmax): LogSoftmax()
  )
)
```

(1) 3 models – feature extractor, class_classifier, domain_classifier：實作細節同上圖print出來的架構。

(2) Loss function and optimizer: class和domain的classifier都使用 Negative Log Liklihood (NLLLoss)。Optimizer 則同樣為 Adam optimizer，lr=0.001。

(3) Training: 先training model by source data，loss= class_classify loss+domain_classify loss，然後再training model by target data，loss= domain_classify loss，total loss = train on source的loss + train on target的loss，做backpropagation。

6. Discuss what you've observed and learned from implementing DANN. (7%)

這次DANN的實作中，UtoM和MtoS的transfer learning大致上很成功，除了accuracy有高過baseline，且大概有50%左右外，亦有符合理論上的大於only training on source accuracy，小於directly training on target accuracy。於tSNE上的表現也大致符合預期，source domain的點有較好的分群效果，而target domain則較無分群效果，較為散亂。這和accuracy只有一半左右(50%)有關，畢竟accuracy就告訴我們這個model沒有把target分得很好，自然tSNE的表現也不會太好)。

然而DANN在StoU的表現就較為奇怪了，雖然依靠著原本就很高的directly training on source accuracy，讓transfer accuracy不會低於baseline的分數，但卻也沒高於directly training on source accuracy，相當於我們做了transfer甚至比沒做，直接拿原本的model來用還爛。觀察其tSNE的圖可以發現其分群結果反而是target domain的點有較好的分群效果，而source domain則較無分群效果，較為散亂，但這並不能代表model真的有把target domain訓練好，因為由後面GTA的結果可知，transfer accuracy沒有很高的情況下tSNE圖還是可以有很分明的分群。而由於dataloader在讀data時亦有shuffle到，因此不考慮是data imbalance引起的異常現象。

最後，猜測可能是和接下來要訓練的GTA有類似的現象，可能DANN模型不擅長訓練彩色到黑白圖片的這種transfer關係，導致StoU沒訓練好。

詳見 problem4

Problem 4: Improved UDA model (35%) In this problem, you are asked to implement an improved model and consider the following 3 scenarios: (Source domain → Target domain) : USPS → MNIST-M, MNIST-M → SVHN, SVHN → USPS

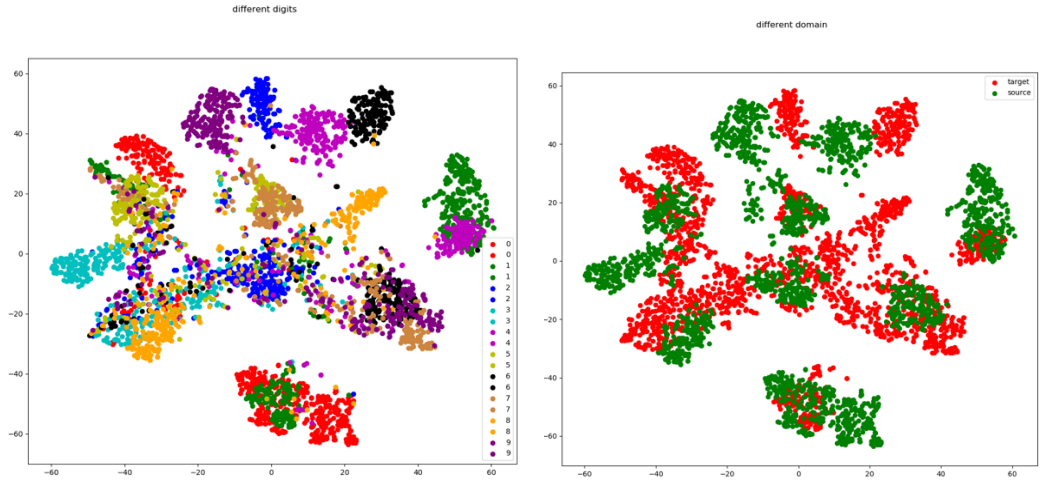
1. Compute the accuracy on target domain, while the model is trained on source and target domain. (domain adaptation) (6+10%)
My improved model is GTA(Generate to Adapt)[9],
but it could not cover all situations (MtoS is not better than DANN).
So I slightly modify the GTA model to handle MtoS situation,
I would introduce it at problem3.

Total ep_nums = 100

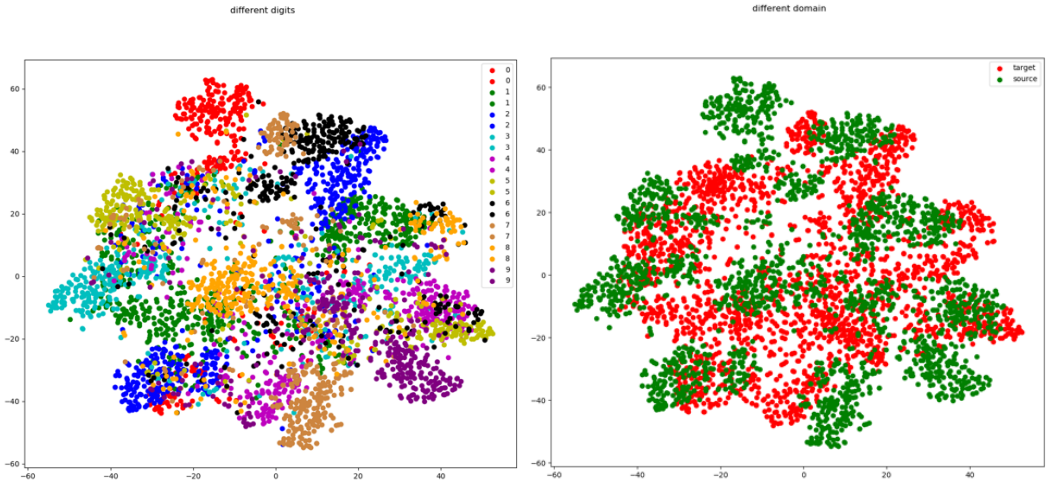
(accuracy, ep, model_name)	USPS → MNIST-M	MNIST-M → SVHN	SVHN → USPS
Adaptation (DANN)	47.69%, ep20 models_UtoM/best.pth	52.42%, ep28 models_MtoS/best.pth	61.23%, ep6 models_StoU/best.pth
Adaptation (Improved model: GTA)	56.22%, ep100 models_UtoM_org/*best*	44.58%, ep46 models_MtoS_org/*best*	72.15%, ep92 models_StoU_org/*best*
Adaptation (Improved model: Improved GTA)	40.9%, ep27 models_UtoM/*best*	66.51%, ep100 models_MtoS/*best*	65.47%, ep86 models_StoU/*best*

2. Visualize the the latent space by mapping the testing images to 2D space (with t-SNE) and use different colors to indicate data of (a) different digits classes 0-9 and (b) different domains (source/target). (6%)
Refer to sklearn tutorial[8] :

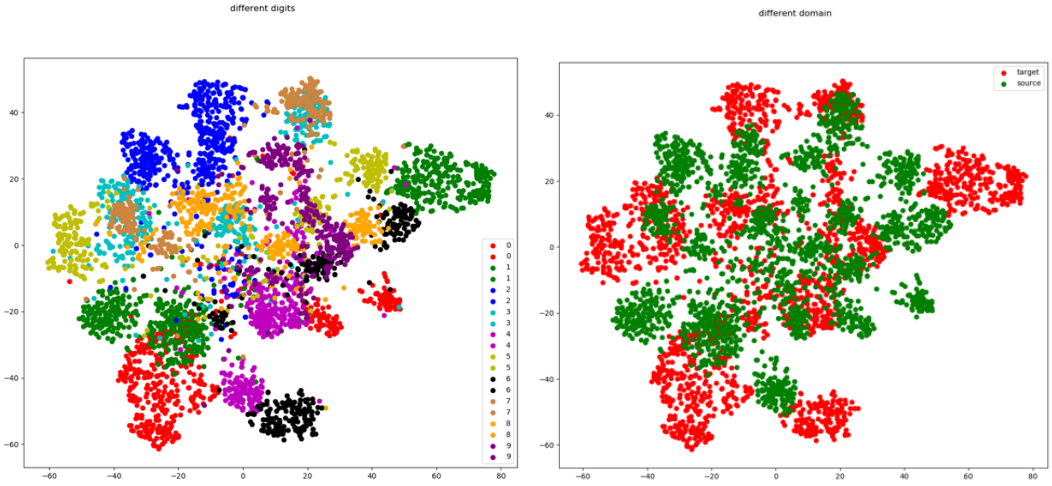
(input source and target each 2000 image from test)
UtoM GTA (a)(b):



MtoS GTA_improved (a)(b):



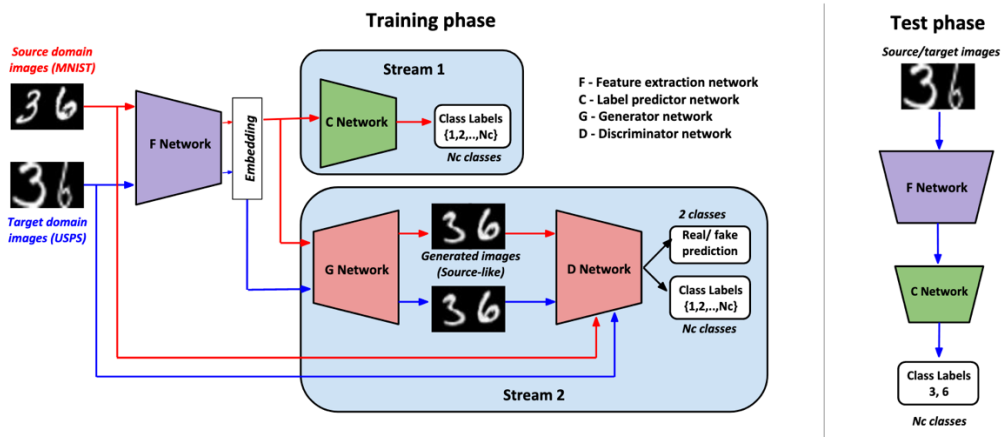
StoU GTA (a)(b):



3. Describe the architecture & implementation detail of your model. (6%)

Refer to paper[9] and pytorch implementation on other datasets[10] :

↓ Main Structure



↓ F, C, G, D network

```
_netG(
  (main): Sequential(
    (0): ConvTranspose2d(651, 512, kernel_size=(2, 2), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (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)
    (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)
    (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)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
_netF(
  (feature): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU(inplace)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1))
    (7): ReLU(inplace)
  )
_netC(
  (main): Sequential(
    (0): Linear(in_features=128, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=128, out_features=10, bias=True)
  )
)
_netD(
  (feature): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.2, inplace)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): LeakyReLU(negative_slope=0.2, inplace)
    (15): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier_c): Sequential(
    (0): Linear(in_features=128, out_features=10, bias=True)
  )
  (classifier_s): Sequential(
    (0): Linear(in_features=128, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

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

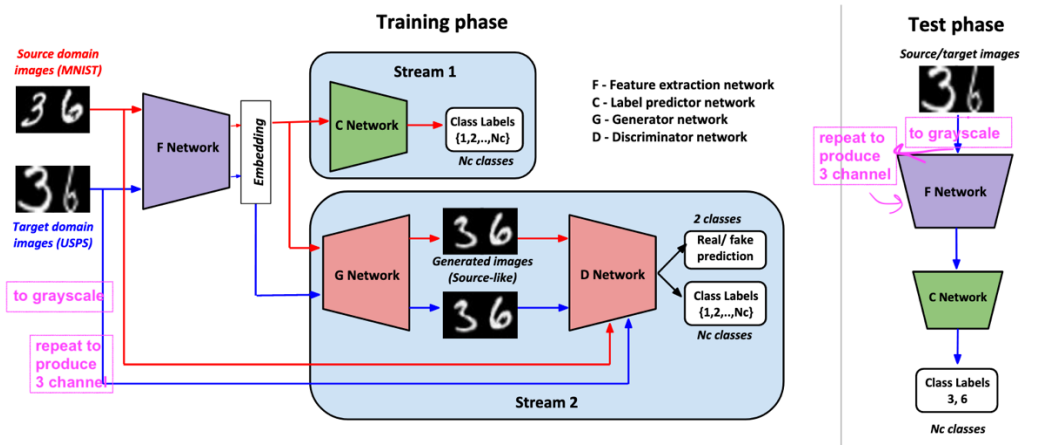
GTA的training結果在三種tranfer裡，都有符合理論上的大於only training on source accuracy，小於directly training on target accuracy。tSNE上的表現非常符合預期(皆比DANN分的好)，展現了良好的分群結果。

然而如果硬要比transfer的accuracy的話，雖然GTA在UtoM和StoU上都有比DANN好，但在MtoS的accuracy卻不比DANN好(雖然還是有過baseline)。

因此在多方嘗試後發現MtoS有一個其他兩個沒有的特質，那就是他是彩色數字transfer到彩色數字，前兩者不是黑白到彩色，就是彩色到黑白，因此我將MtoS在train之前，把target domain的影像轉成灰階再丟進去，同理

predict時也先轉成灰階再predict，如此改善後在MtoS的transfer learning獲得了良好的改善，accuracy成功超越了DANN，tSNE的分群結果亦良好，但卻會使另外兩個transfer的結果不如預期。

因此最後hybrid的作法就是在進行MtoS時採用我的improved方法，而其他則採用原model。



Reference:

[1] Pytorch dcgan tutorial https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
[2] Unsupervised representation learning with deep convolutional generative adversarial networks. A Radford, L Metz, S Chintala - arXiv preprint arXiv:1511.06434, 2015 - arxiv.org
[3] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. arXiv preprint arXiv:1610.09585, 2016.
[4] pytorch acgan on mnist github https://github.com/eriklindernoren/PyTorch-GAN#auxiliary-classifier-gan
[5] pytorch acgan on Cifar10 github https://github.com/TuXiaokang/ACGAN.PyTorch
[6] Domain-Adversarial Training of Neural Networks (DANN) . Y. Ganin et al., ICML 2015
[7] pytorch dann github https://github.com/fungtion/DANN_py3
[8] tSNE sklearn tutorial https://scipy-lectures.org/packages/scikit-learn/auto_examples/plot_tsne.html
[9] Swami Sankaranarayanan, Yogesh Balaji, Carlos D. Castillo and Rama Chellappa. “Generate To Adapt: Aligning Domains using Generative Adversarial Networks”, CoRR, 2017, http://arxiv.org/abs/1704.01705
[10] GTA pytorch implementation: https://github.com/yogeshbalaji/Generate_To_Adapt