DLCV HW3 Report

學號：R08945006 學生：葉政樑

VAE Problem1-1:

```
VAE(
  (encoder): 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): ReLU(inplace=True)
    (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): ReLU(inplace=True)
    (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): ReLU(inplace=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (encoder_fc1): Sequential(
    (0): Linear(in_features=8192, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (encoder_fc2): Sequential(
    (0): Linear(in_features=8192, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (decoder_fc): Sequential(
    (0): Linear(in_features=1024, out_features=8192, bias=True)
    (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (decoder): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(64, 3, kernel_size=(2, 2), stride=(2, 2))
    (10): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): Sigmoid()
  )
)
(DLCV_HW3) yehcl@yehcl-System-Product-Name:~/Desktop/DLCV_HW3/Q1$
```

Model detail:

在 VAE 中 encoder 架構使用 Conv+BatchNorm+ReLU+MaxPool 經典架構堆疊成。Decoder 架構使用 ConvTranspose（放大 image size ）+BatchNorm+LeakyReLU（與 ReLU 比較效果較好）。最後的 Activation function 則是用 Sigmoid（與 Tanh 比較效果較好）。在 Latent space 則是使用 fully connected layer 拉伸成 1024 維後再進行 reparameterize。

Training detail:

Batch size = 64

Latent vector dimensions = 1024

Learning rate = 0.0005

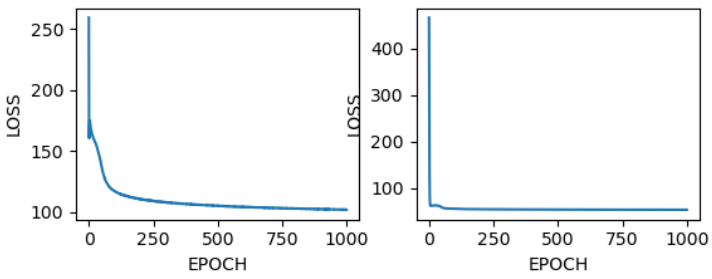
Learning rate schedule: Epoch 分別在>150、300、450 之後 Learning rate * 0.5

Optimizer = Adam(betas=(0.5, 0.999), weight decay=0)

Data augmentation: 使用 RandomHorizontalFlip, CenterCrop, ToTensor

Loss function : KLD weight 使用 0.00001 效果不好 -> 最終調整後使用 0.1~1 較好

VAE Problem1-2:



左圖為 MSE loss（左）與 KLD（右）的學習曲線，特別注意的是我在 KLD 表現上與助教給的 example 特別不一樣，有嘗試使 KLD 曲線分佈與助教的類似，但是 generate images 成效不甚理想，最終才修改至此。

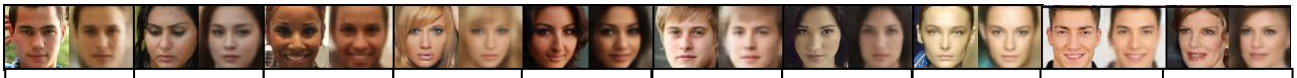
VAE Problem1-3:

下圖: 左圖為原圖，右圖為重建影像。

MSE 計算方式為:

criterion = nn.MSELoss(reduction='sum')

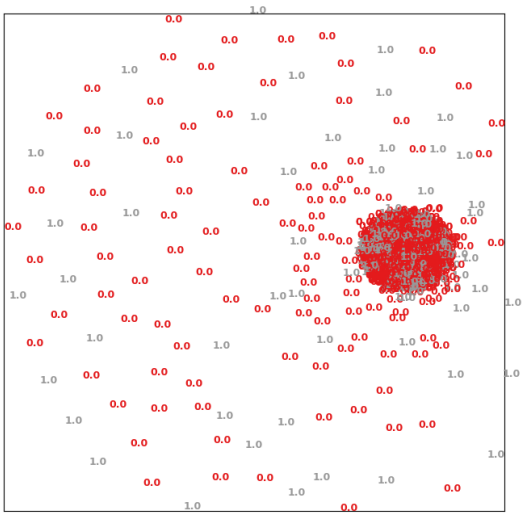
MSE_loss = criterion(reconstruction_img, original_img)



92.3118	62.8337	90.9674	62.0230	87.0707	75.9791	78.5198	101.597	128.591	133.058

VAE Problem1-4:

VAE Problem1-5:



右圖為將 VAE latent vector 經過 tSNE 視覺化結果:

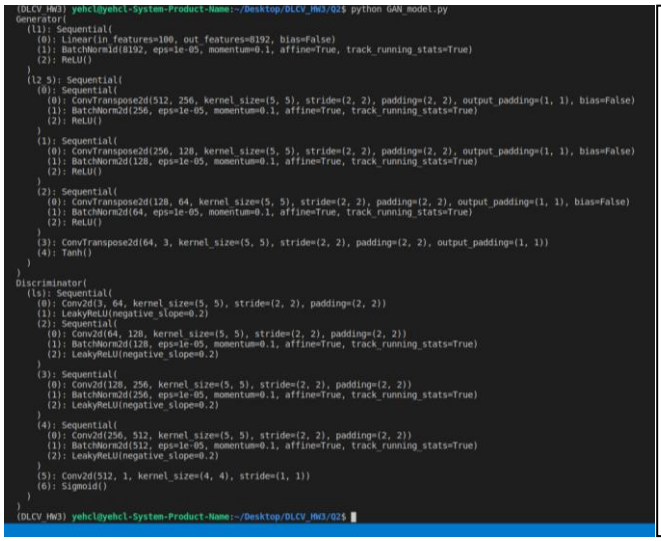
此選取頭髮顏色為黑髮當作 attribute， 0 代表非黑色， 1 代表黑色。 可以看到在結果上並不十分理想，猜想可能是 model 對於頭髮顏色不好區分，又或者是 VAE 沒有訓練到最佳性能。

VAE Problem1-6:

在本次作業中我首次接觸到 VAE，透過實做 VAE 的架構我開始對於 unsupervise learning 有一點認識。以架構來說:實際上我更加了解經典的 autoencoder 架構，並且在其中加入 reparameterize 的技巧使網路架構能夠學到不同資料 distribution 的訊息，進而能夠從雜訊中產生新的影像。 在實際上 training 時我發現並不是那麼容易，尤其是在調整 KLD 和 MSE loss 的 weight 時，我一開始一直在 0.00001 上下調整，後來發現怎麼 train 都 train 不好，表現形式是在 generate 影像時會壞掉，因此我後來逐步調整 weight 後慢慢在 reconstruct image 品質和 generate image 品質中取得能較接受的結果，另外我也花非常多時間在調整網路架構，但並沒有發現什麼有效的突破點。

最後使用 data augmentation 能夠使 training 成效更好！但要注意有些常用的方法可能不適用，如：把影像上下 flip，這樣其實對影像重建來說它會重建成反的，所以需要特別小心不能亂用，而水平翻轉則是沒問題。

GAN Problem2-1:



Implement detail:
model detail:
在這次的作業 GAN 我採用的是 DCGAN 架構(主架構參考自 pytorch tutorials)，Generator 架構主要使用 ConvTranspose + BatchNorm (參考自論文中建議)+ ReLU(or LeakyReLU)堆疊成，並且在第一層使用 fc 連接，最後用 tanh 輸出(論文中建議的 tip)。在 Discriminator 方面架構使用 Conv + BatchNorm (參考自論文中建議) + LeakyReLU（與 ReLU 比較後效果較好，參考自 VAE 訓練經驗，且論文中建議 tip）的堆疊方式, 最後一層則是用 Sigmoid（與 Tanh 比較後效果較好，參考自 VAE 訓練經驗）。
因此總結來說 G、D 各為 4 層 Conv、5 層 ConvTranspose。在架構中不使用 Pooling layers。(為論文中建議之結果)

Training detail:
Batch size = 64
Latent vector dimension= 100 維
Learning rate = 0.0001
Optimizer = Adam (betas=(0.5, 0.999),weight_decay=0)
Data augmentation：使用 RandomHorizontalFlip, CenterCrop, ToTensor, Normalize(0.5,0.5) -> 此為搭配 tanh 之建議 tip，詳見 github: https://github.com/soumith/ganhacks
Loss function : nn.BCELoss()
額外嘗試 tips: 參考自: https://github.com/soumith/ganhacks

- Random flip labels(T->F, F->T): 使 Discriminator 在一開始訓練時會混淆，而抑制訓練剛開始 discriminator 就壓制 generator。成效: 在一開始幾個 epoch 有效，但須注意不能這樣訓練太久，Discriminator 有可能會壞掉。
- Soft label: 嘗試後發現效果沒有顯著差異，可能是 soft 範圍定義太小或太寬。
- 一次訓練 D+多次訓練 G: 在訓練初期 D 可能會很快壓制 G，使得 G 很難成長，因此在訓練初期嘗試訓練一次 D 後，訓練 G 兩次或三次。效果在初期訓練有效，但是需要思考在訓練期間何時要換成正常訓練方式。
- Add noise to inputs: 試著加入雜訊到 input 中，如:高斯模糊化，發現效果沒有顯著提升，轉而使用其他 data augmentation 方式。

GAN Problem2-2:

左圖為 G 隨機生成 32 張影像，可以看到人臉的輪廓明顯，五官大致上端正，唯有右下角倒數第三張是大側臉的影像，生成效果不好，影像扭曲。

GAN Problem2-3:

同樣在本次作業中我首次接觸到 GAN 這項技術，透過實作 GAN 更加認識了 GAN 的原理與 Generative + adversarial 真正精神 。

以架構來說我參考了 DCGAN 原文 paper:

'UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS'

在 paper 中提到許多 architecture guideline，所以我是站在前人的肩膀上去 implement DCGAN ，因此少走了很多調參數的彎路與架構設計問題，如: remove pooling layer、 use batchnorm、use leakyReLU 等等做法，並且在參考其架構後也取得不錯的結果。

另外在訓練的過程中我也在網路上看了很多人訓練 GAN 的技巧，並且將其實作看看，發現其中 flip label、training G twice 等等都在初期對於 D 壓制 G 的情況較有效，另外一種方式則是使用 data augmentation 使 input 變的複雜，讓 D 難以初期就很強大，這也是有效的方式。

最後我發現 GAN 很像是一種拔河方式，你要讓 D、G 都維持差不多的戰力去對抗，接著同步慢慢成長這樣才有好的結果，因此後來我花了非常多時間確認自己訓練的過程中，是否有一方是被壓制的。

如:D loss -> 0: model fail， G loss -> 0: G is fooling D 等。

GAN Problem2-4

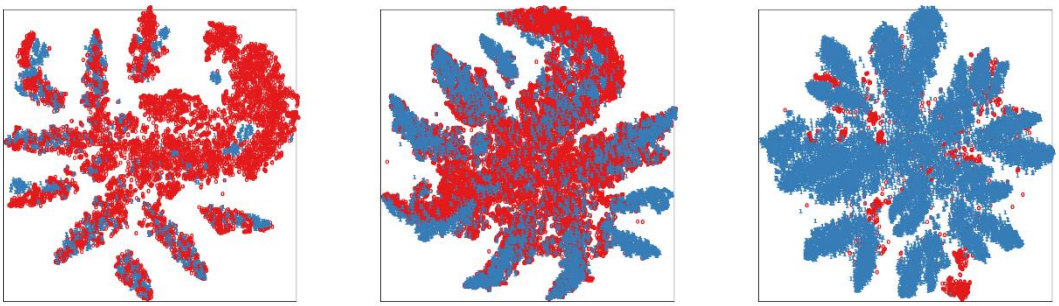
透過比較 VAE 與 GAN 產生的結果，我發現由 VAE 產生的影像結果較模糊，不太能表現出細緻的影像品質，但是以肉眼來看，由於較模糊不太會產生很明顯的影像扭曲感。而由 GAN 產生的影像則是較為細緻，人臉的五官輪廓等都清晰很多，邊界也明顯許多，但也因為影像清晰使得若是五官稍微有點扭曲，以肉眼看就非常明顯。

DANN Problem3-1~3-3:

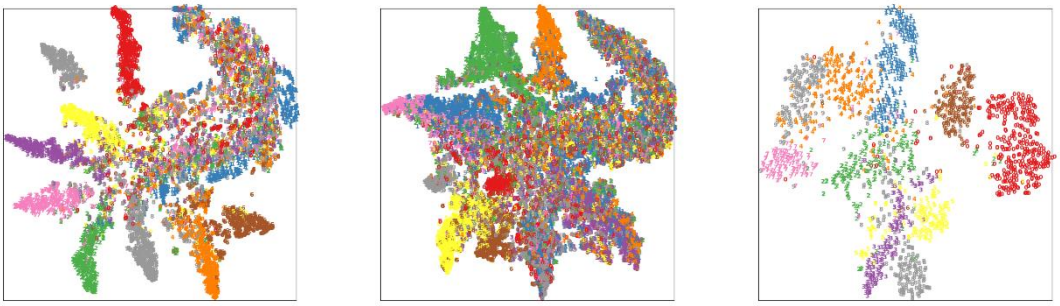
	USPS -> Mnist-M	Mnist-M -> SVHN	SVHN -> USPS
Trained on source Accuracy:	0.355	0.274	0.549
Adaptation (DANN) Accuracy:	0.548	0.406	0.622
Trained on target Accuracy:	0.958	0.917	0.958

DANN Problem3-4:

藍色代表 1=souce domain，紅色代表 2=target domain



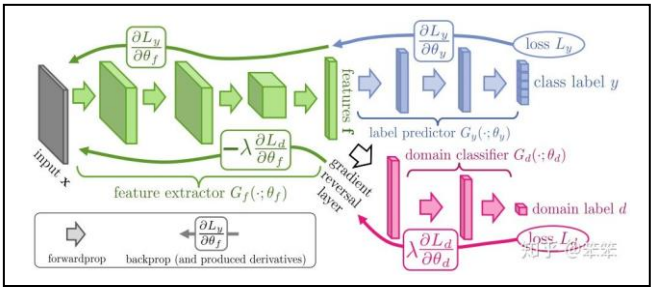
上下圖由左至右分別是 1. USPS -> Mnist-M ，2. Mnist-M -> SVHN，3. SVHN -> USPS



由不同顏色分別出 label 0 ~9。備註: 若因為 8、9 之間顏色相近或是資料點太多太複雜，可以放大圖片來看會有數字表示。

DANN Problem3-5:

Architecture:
本題我採用原始 DaNN paper 中的架構，如右圖所示:
架構中共分成 3 部分
1. FeatureExtractor: 由 Conv+Batchnorm+ReLU+MaxPool 組成，共 4 層。
2.LabelPredictor: 由 Fully connected layer 組成(FC + ReLU)，共三層。
3.DomainClassifier: 同樣由 Fully connected layer 組成(FC + Batchnorm + ReLU)



參考自: https://zhuanlan.zhihu.com/p/51499968

Implementation and Training details:
Learning rate = 0.0001
Batch size = 32
Optimizer 在 FeatureExtractor、LabelPredictor 和 DomainClassifier 都是使用 Adam
Adam(betas=(0.5, 0.999), weight_dacay = 0)
Data Augmentation 則是將原本 28*28 的影像 resize 成 32*32，配合 FeatureExtractor 架構，並且將影像 Normalize。
訓練流程參考李宏毅老師課程中提到，可以把 DaNN 中 FeatureExtractor、LabelPredictor、和 DomainClassifier(以下稱 F、L、D)一起 train、或是參考 GAN 的訓練模式: Step1: 先訓練 D。Step2: 訓練 F + L + D。
而在 Loss function 設計則參考李宏毅老師課程中以 class loss - weight * domain loss 表示，weight 類似 train VAE 時需要調整 loss 的權重。參考資料連結:
https://www.youtube.com/watch?v=qD6iD4TFsdQ&list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49&index=28&t=2660s

DANN Problem3-6:

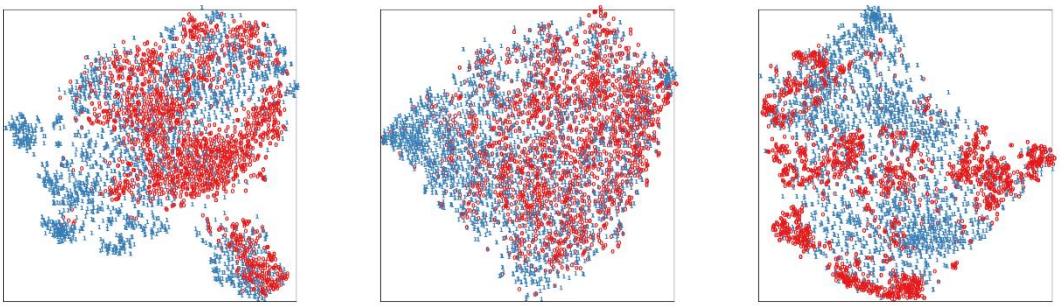
同樣的也是第一次接觸 DANN 這個架構，我發現 DANN 在架構上相當容易了解，可以拆成 3 部分個別去看，每個部分都是用非常熟悉的架構即可堆疊而成，然而其中使用到的 gradient reversal layer 則是非常聰明的做法，使得一個簡單的模型能夠藉由類似 GAN 中 Discriminator v.s generator 的形式，讓 F 去 fool D，而 D 最終會慢慢 fail，但是 D 在訓練過程中我們需要調控他，讓他 struggle 到最後才 fail，進而把 F 的能力提升到能夠消除 domain 的隔閡，因此我認為有些像 GAN 一樣，在訓練中需要達到平衡，而不能有一邊特別強勢，不然會容易 model failure。

Improved UDA Problem4-1: Implement PixelDA

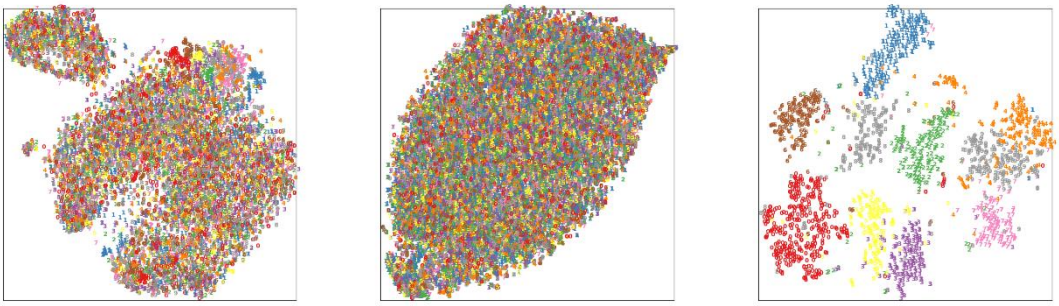
	USPS -> Mnist-M	Mnist-M -> SVHN	SVHN -> USPS
Adaptation (PixelDA) Accuracy:	0.603	0.433	0.647
Adaptation (DANN) Accuracy:	0.548	0.406	0.622

Improved UDA Problem4-2: Implement PixelDA:

藍色代表 1=souce domain，紅色代表 2=target domain



上下圖由左至右分別是 1. USPS -> Mnist-M ，2. Mnist-M -> SVHN，3. SVHN -> USPS



由不同顏色分別出 label 0 ~9。備註: 若因為 8、9 之間顏色相近或是資料點太多太複雜，可以放大圖片來看會有數字表示。

另外有疑問的地方為跟 DANN 比較時在 accuracy 上雖然表現較好，但在 target domain = Mnist-M、SVHN 在 t-SNE 視覺化 latent space 上分類效果看起來並不是很好，但在 USPS 表現不錯。

Improved UDA Problem4-3: Implement PixelDA:

本題我採用的是 PixelDA 這個架構，架構如右圖所示。PixelDA 是基於 GAN 架構上再加以延伸，並且加入了 task classifier 幫助進行任務分類，同時傳統的 GAN 架構也維持住，使得 G 能夠以 noise 及 domainA 的影像去產生出風格類似於 domain B 的影像，影像結果能夠很好的消除 domain 隔閡與進行不同 domain 資料分類等任務。實作網路架構我參考了此 github: https://github.com/eriklindernoren/PyTorch-GAN 主架構為 Generator (G)，Discriminatior(D)和 Classifier(C)共三部分，其中又以 ResidualBlock 堆疊 (Conv+ Batchnorm+ReLU)。

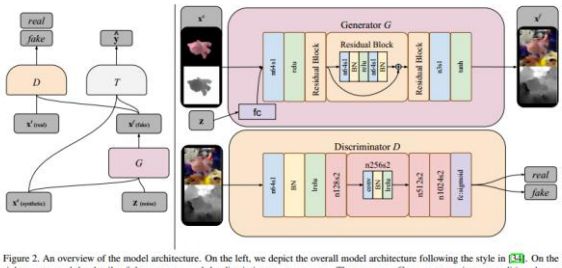


Figure 2. An overview of the model architecture. On the left, we depict the overall model architecture following the style in [34]. On the right, we expand the details of the generator and the discriminator components. The generator G generates an image conditioned on a synthetic image $\mathbf{x}^s$ and a noise vector $\mathbf{z}$. The discriminator D discriminates between real and fake images. The task-specific classifier T assigns task-specific labels $\mathbf{y}$ to an image. A convolution with stride 1 and 64 channels is indicated as n64s1 in the image. lrelu stands for leaky ReLU nonlinearity. BN stands for a batch normalization layer and FC for a fully connected layer. Note that we are not displaying the specifics of T as those are different for each task and decoupled from the domain adaptation process.

參考自:
https://openaccess.thecvf.com/content_cvpr_2017/papers/Bousmalis_Unsupervised_Pixel-Level_Domain_CVPR_2017_paper.pdf

Training details:
Batch size = 64
Learning rate= 0.00001
Optimizer = Adam(beta=(0.5,0.999))
Latent vector dimension = 1024
Data augmentation: transforms.RandomRotation，transforms.CenterCrop，transforms.Resize，transforms.ToTensor，transforms.Normalize 等等

Improved UDA Problem4-4: Implement PixelDA:

在本次實做 UDA model 的題目，我首先遇到的問題是應該要使用哪個模型，由於之前並沒有甚麼經驗，因此在上完課程後與參考提供的幾篇 paper 後我決定採用 PixelDA 這個 model。在實作 PixelDA 的過程中我更加了解這個”較先進”的 model，發現對我來說這個新的 model 也是結合前人提出的觀念與技巧而產生的，讓我更加覺得本次作業的扎實程度，從 GAN、DANN 再到 PixelDA 等，都讓我更加了解其中共同用到的技巧，如: GAN Generator 與 Discriminator 對抗的觀念。

另外我也發現 PixelDA 非常不好訓練，其一可能是我對此 model 不夠熟悉，並不清楚如何增加或修改網路架構才能夠增加表現，因此大多數時候我都是 trial and error，其二可能是對於訓練 GAN 技巧不足導致，即使參考了許多網路上訓練 GAN 的 tips，但很多 tips 並沒有對於結果有顯著提升，因此也損失許多時間嘗試。

最後則是實做 PixelDA 有顯著的收穫為看到自己訓練後的結果，此結果有點類似於 style transform，結果將原 domain A -> domain B ，以 generator 去生成 fake image，fake image 能夠成功消除 domain 的隔閡，保留原始 domain 的資訊但風格轉換成 domain B。如下圖:

參考資料:

- https://arxiv.org/pdf/1511.06434.pdf
- https://github.com/soumith/ganhacks
- https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html#
- https://www.youtube.com/watch?v=DMA4MrNieWo&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw&index=4
- https://github.com/eriklindernoren/PyTorch-GAN
- http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML20.html
- https://www.youtube.com/watch?v=YNUek8ioAJk&list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49&index=26
- https://blog.csdn.net/StreamRock/article/details/81096105
- https://blog.csdn.net/just_sort/article/details/84581400
- https://towardsdatascience.com/generative-adversarial-network-gan-for-dummies-a-step-by-step-tutorial-fdefff170391
- https://www.chainnews.com/zh-hant/articles/498710369918.htm