HW2 Report

B08901165電機四 南策昇

Problem 1

Discriminator of A:

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()

)

)

Generator of A:

Generator(

(main): Sequential(

(0): ConvTranspose2d(100, 1024, kernel\_size=(4, 4), stride=(1, 1), bias=False)

(1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

(6): ConvTranspose2d(512, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(8): ReLU(inplace=True)

(9): ConvTranspose2d(256, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(11): ReLU(inplace=True)

(12): ConvTranspose2d(128, 3, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(13): Tanh()

)

)

Discriminator of B:

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)

)

)

Generator of B:

Generator(

(main): Sequential(

(0): ConvTranspose2d(100, 1024, kernel\_size=(4, 4), stride=(1, 1), bias=False)

(1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

(6): ConvTranspose2d(512, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(8): ReLU(inplace=True)

(9): ConvTranspose2d(256, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(11): ReLU(inplace=True)

(12): ConvTranspose2d(128, 3, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(13): Tanh()

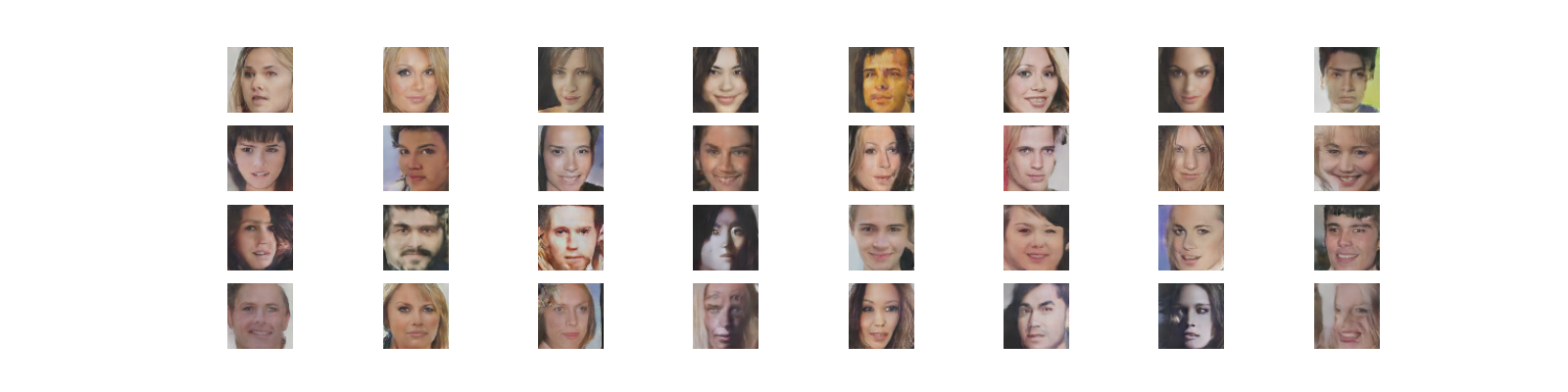
)

)

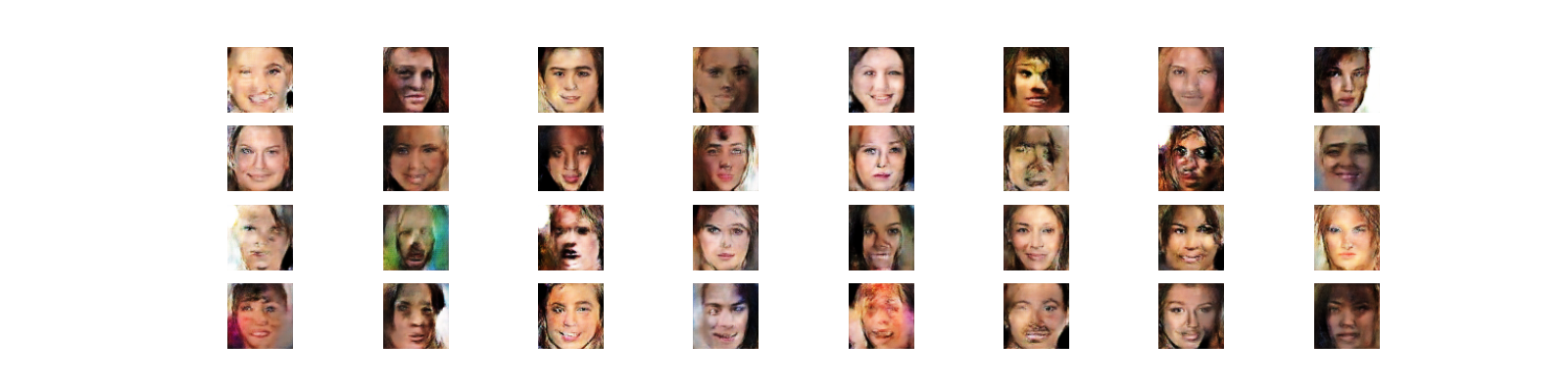
Ref: [DCGAN Tutorial — PyTorch Tutorials 1.13.0+cu117 documentation](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html)

2.

Model A:



Model B:



FID和face\_recog.py的accuracy都顯示Model A的表現比Model B好，而這也顯示在這些生成的圖片上，Model A生成的人臉大部分輪廓都相當明確，反觀Model B生成的圖片在輪廓附近會有模糊或者歪曲的狀況，色塊也會有點雜亂，只能掌握到人臉的大致特徵，因此雖然face\_recog.py上面accuracy差異不大（約3%），但是FID就差很多（67 v.s. 25）。

3.

原本DCGAN和WGAN都train不起來，我還以為是我train不夠久，沒想到是真的沒辦法。可是看朋友有人有train成功，我就花了一點時間好好端詳一下這個看起來沒問題的code到底是在哪裡出了問題。後來才發現我自作聰明，把Normalization的參數改成Imagenet的這組，而不是用0.5，想說應該會表現好一點，但問題就是出在這裡。用0.5標準化是為了要讓pixels數值都限定在-1~1，但用Imagenet就會讓pixels跑出這個範圍，偏偏我的generator的最後一層掛了一個hyperbolic tangent，所以會讓所有output都在-1~1之間，這樣discriminator只要看pixels的數值有沒有落在這個範圍，就能判定他是真圖還是假圖了。把這點改回來後，就有train起來了。

原本預期WGAN會train比較好，沒想到FID一直下不去，我猜可能是因為WGAN的D和G training的頻率不一樣，如果要達到一定的效果，要train的比DCGAN還要久才行，另外也有可能是parameters clipping在誤事，讓他很難train起來，不過這些還有待進一步的實驗證明。

Problem 2

1.

Model architecture:

DDPM(

(nn\_model): ContextUnet(

(init\_conv): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(3, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

(down1): UnetDown(

(model): Sequential(

(0): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

(1): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

)

)

(down2): UnetDown(

(model): Sequential(

(0): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

(1): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

)

)

(to\_vec): Sequential(

(0): AvgPool2d(kernel\_size=7, stride=7, padding=0)

(1): GELU(approximate=none)

)

(timeembed1): EmbedFC(

(model): Sequential(

(0): Linear(in\_features=1, out\_features=256, bias=True)

(1): GELU(approximate=none)

(2): Linear(in\_features=256, out\_features=256, bias=True)

)

)

(timeembed2): EmbedFC(

(model): Sequential(

(0): Linear(in\_features=1, out\_features=128, bias=True)

(1): GELU(approximate=none)

(2): Linear(in\_features=128, out\_features=128, bias=True)

)

)

(contextembed1): EmbedFC(

(model): Sequential(

(0): Linear(in\_features=10, out\_features=256, bias=True)

(1): GELU(approximate=none)

(2): Linear(in\_features=256, out\_features=256, bias=True)

)

)

(contextembed2): EmbedFC(

(model): Sequential(

(0): Linear(in\_features=10, out\_features=128, bias=True)

(1): GELU(approximate=none)

(2): Linear(in\_features=128, out\_features=128, bias=True)

)

)

(up0): Sequential(

(0): ConvTranspose2d(256, 256, kernel\_size=(7, 7), stride=(7, 7))

(1): GroupNorm(8, 256, eps=1e-05, affine=True)

(2): ReLU()

)

(up1): UnetUp(

(model): Sequential(

(0): ConvTranspose2d(512, 128, kernel\_size=(2, 2), stride=(2, 2))

(1): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

(2): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

)

)

(up2): UnetUp(

(model): Sequential(

(0): ConvTranspose2d(256, 128, kernel\_size=(2, 2), stride=(2, 2))

(1): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

(2): ResidualConvBlock(

(conv1): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

(conv2): Sequential(

(0): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): GELU(approximate=none)

)

)

)

)

(out): Sequential(

(0): Conv2d(256, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): GroupNorm(8, 128, eps=1e-05, affine=True)

(2): ReLU()

(3): Conv2d(128, 3, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

)

)

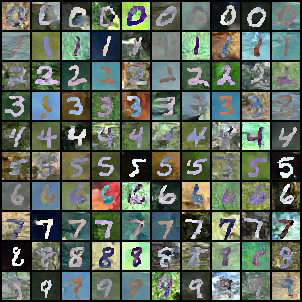
(loss\_mse): MSELoss()

)

參考來源：[Conditional\_Diffusion\_MNIST/script.py at main · TeaPearce/Conditional\_Diffusion\_MNIST (github.com)](https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/script.py?fbclid=IwAR3pBmDkgAgzrGesBSOaXYqIaj-9tz2Q1Nb7NapVX8p2OJD2svFd8dbWf6A)

我使用UNet當作backbone並且在層與層之間都有加batchnorm。此外，在去除雜訊的時候，我使用total step = 600，並且送進model生成時會有兩個部分在train，一部分是有context，一部分是沒有context，兩者之間的weight我調整成1:0.5。Learning rate有做linear schedule，但是實際上train的時候，我有刻意先在大的learning rate train 多一點epoch，因為我發現linear schedule其實有一點降低得太快。其他關於beta或者mean的計算都是參考原始paper做的。

2.



3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| t = 0 | t = 120 | t = 240 | t = 360 | t = 480 | t = 600 |
|  |  |  |  |  |  |

4.

我覺得Diffusion model應該是這次作業最難的部分，剛開始我參考了幾個網站，產出我的code，結果發現他一直train不起來，而且生成圖片超級慢，於是我仔細研究了一下我的code，發覺我timestep弄反了，全部存在沒有denoise的雜訊，改了一下之後就有了一點樣子。之後我又想說他denoise好久，我讓他只要生成其中幾個timestep的過程就好，沒想到train出來馬上又壞了。我就觀察了一下我print出來的timestep，雖然我是設定每隔120個timestep就顯示一次圖片，但是它們之間感覺不像denoise了120次，許多隨機的pixel都長得一樣，頓時心中有了一個猜想，趕快重新細讀一下paper，發覺果然是這樣，以前我一直以為diffusion model將timestep資訊encode進去就可以直接生成denoise那麼多timestep的資料，所以我只要丟timestep = 600就可以直接得到生成的圖片，但事實上生成過程要從600一層一層denoise到1，跑過這個model 600次，才能生好一張圖片，難怪我相隔120timestep的圖片長得差不多，因為中間被跳過的timestep我全都沒跑，所以他事實上才denoise了一次而已。改回來之後，我train起來就沒什麼問題了。

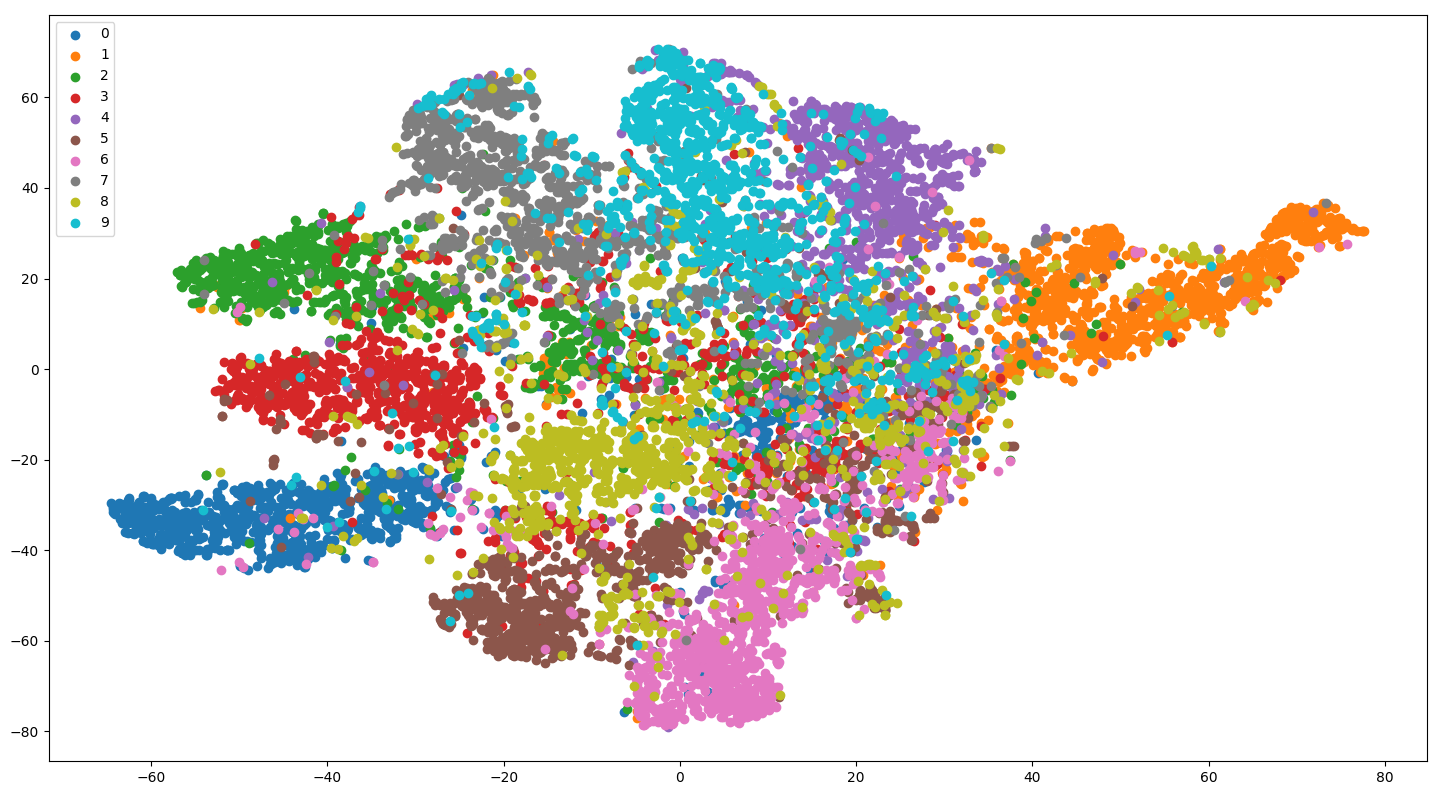
Problem 3

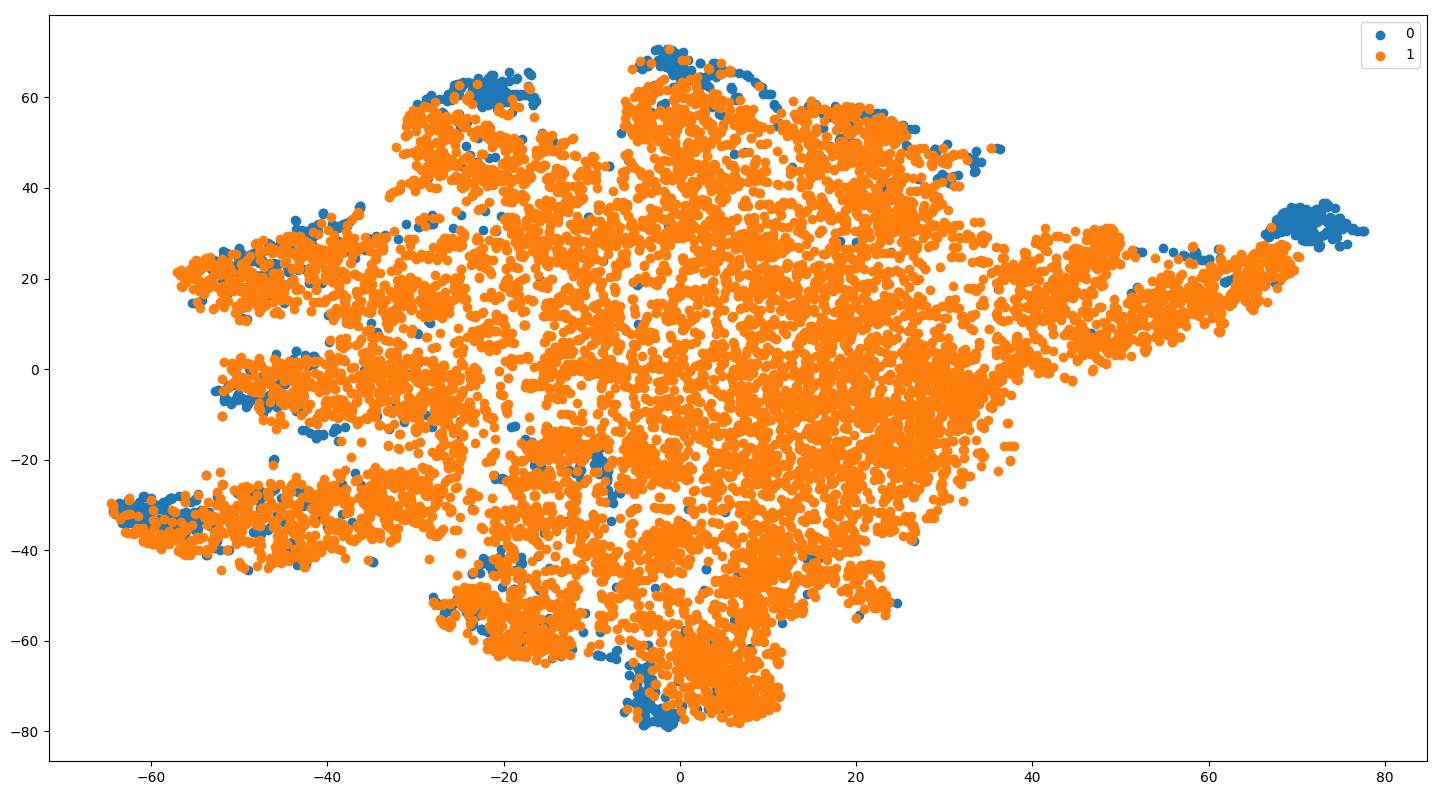
1.

|  |  |  |
| --- | --- | --- |
|  | MNIST-M -> SVHN | MNIST-M -> USPS |
| Trained on source | 0.358 | 0.708 |
| Adaptation (DANN) | 0.504 | 0.818 |
| Trained on target | 0.911 | 0.983 |

2.

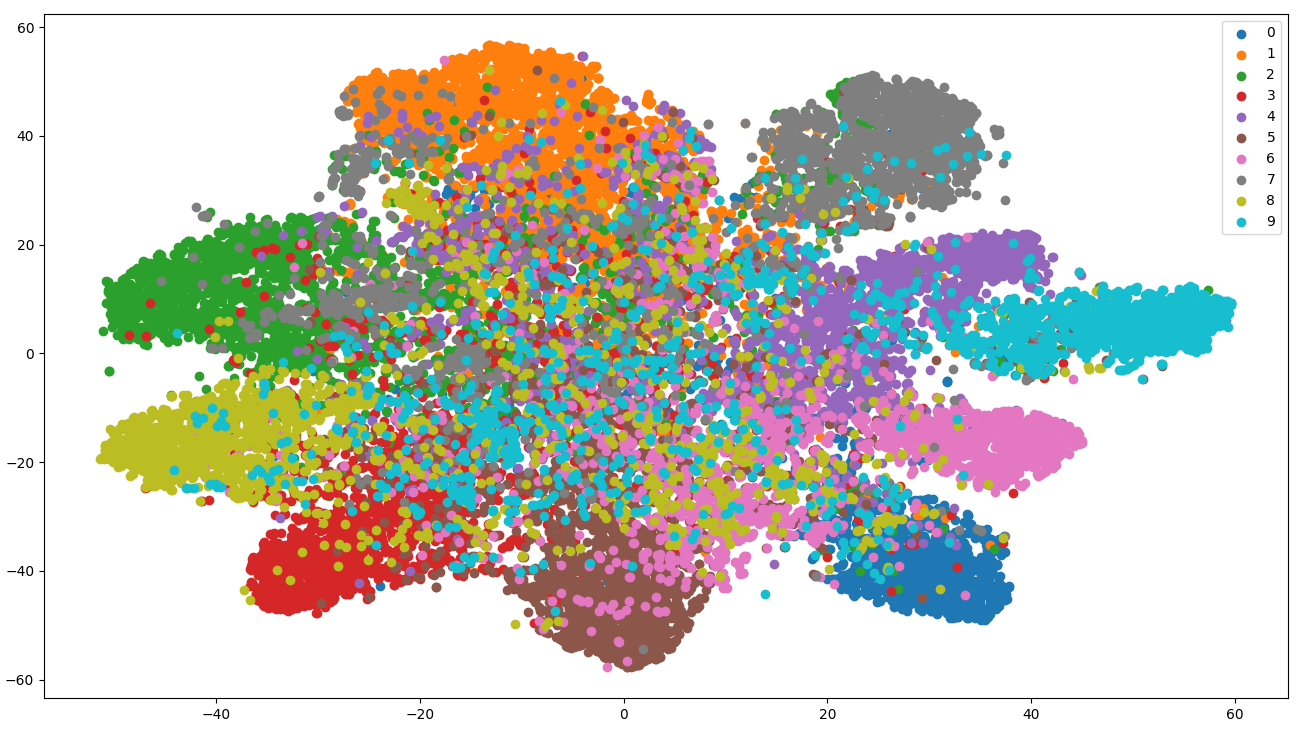
MNIST-M -> usps

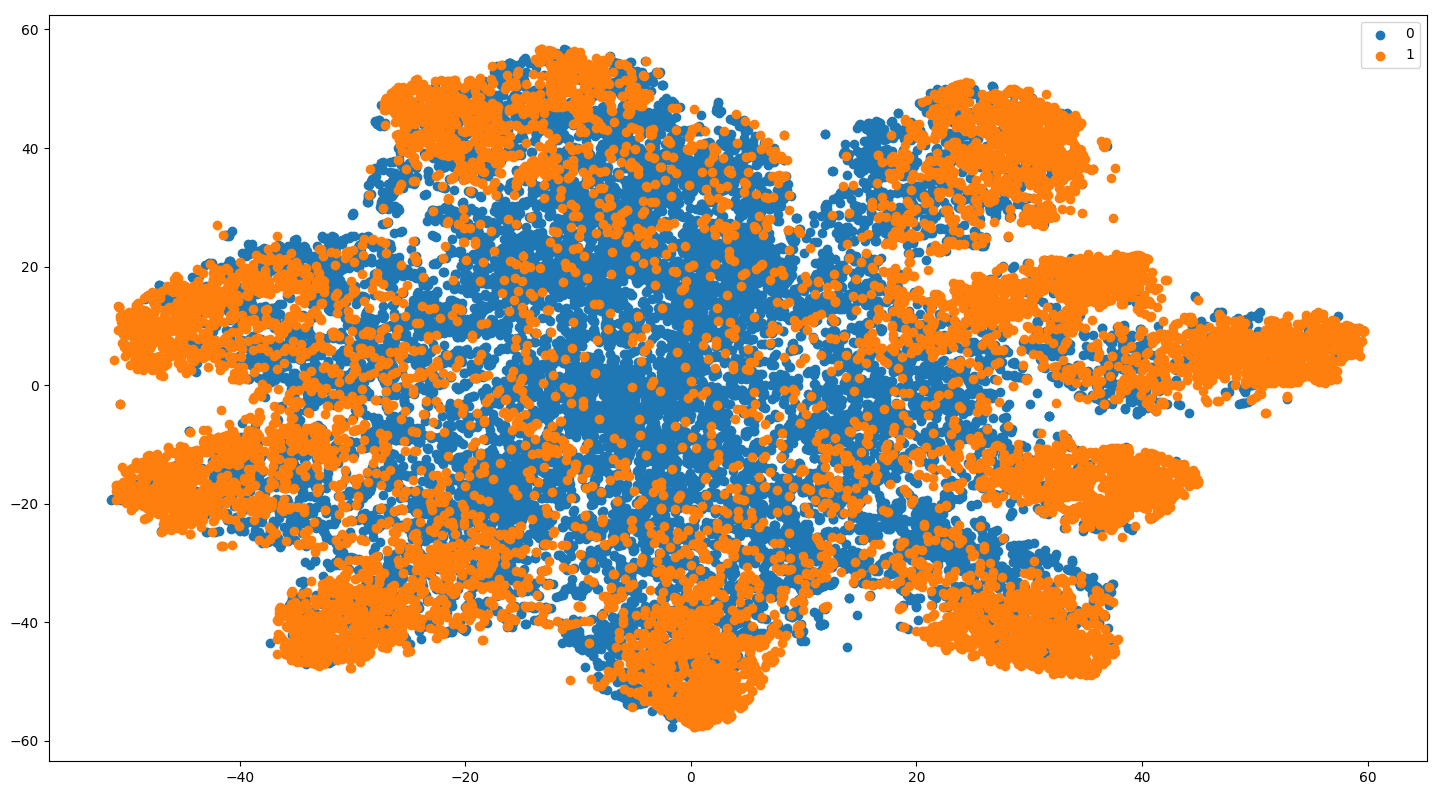




0: usps 1: MNIST-M

MNIST-M -> svhn





0: svhn 1: MNIST-M

3.

我的model其實就是簡單的幾層CNN和FC layer，大致結構就跟老師上課教的一樣，因為我就把它想成是要train MNIST-M，眾所皆知這其實並不複雜。關鍵應該是我在domain classifier和feature extractor中間加了Reverse layer (GRL)讓gradient在經過這裡時會變號，也就是DANN的精髓，如下：

class GRL(Function):

@staticmethod

def forward(ctx, x, alpha):

ctx.alpha = alpha

return x.view\_as(x)

@staticmethod

def backward(ctx, grad\_output):

output = grad\_output.neg() \* ctx.alpha

return output, None

model 架構：

class DANN(nn.Module):

def \_\_init\_\_(self,num\_classes=10):

super(DANN,self).\_\_init\_\_()

self.features=nn.Sequential(

nn.Conv2d(3,32,5),

nn.ReLU(inplace=True),

nn.MaxPool2d(2),

nn.Conv2d(32,48,5),

nn.ReLU(inplace=True),

nn.MaxPool2d(2),

)

self.avgpool=nn.AdaptiveAvgPool2d((5,5))

self.task\_classifier=nn.Sequential(

nn.Linear(48\*5\*5,100),

nn.ReLU(inplace=True),

nn.Linear(100,100),

nn.ReLU(inplace=True),

nn.Linear(100,num\_classes)

)

self.domain\_classifier=nn.Sequential(

nn.Linear(48\*5\*5,100),

nn.ReLU(inplace=True),

nn.Linear(100,2)

)

self.GRL=GRL()

def forward(self,x,alpha):

x = x.expand(x.data.shape[0], 3, 28, 28)

x=self.features(x)

x=self.avgpool(x)

x=torch.flatten(x,1)

task\_predict=self.task\_classifier(x)

x=GRL.apply(x,alpha)

domain\_predict=self.domain\_classifier(x)

return task\_predict,domain\_predict

這樣主幹就建立好了。

Ref: [fungtion/DANN: pytorch implementation of Domain-Adversarial Training of Neural Networks (github.com)](https://github.com/fungtion/DANN)

此外，剛開始train的時候，accuracy一直都起不來，甚至連source都沒辦法，只能說幸好以前train過MNIST，所以知道這樣的model架構絕對是夠大的，可先從其他地方開始debug。我猜測應該是domain loss那邊站太重了，老師上課也有講，剛開始的時候可以讓classifier先train一下，有一個底，再修domain，所以我就做了一個scheduling，讓domain loss不要太快出現，而情況也因此改善了，但是還遠不及baseline，甚至還出現一個奇怪的問題。我發現我只要train到大約第三個epoch，原本長得好好的acc就會突然崩到剩20%，不管在哪個task都是，猜測又是因為domain loss突然dominate了整個update，所以我就試著在domain loss scheduling前面加一個係數，讓他即使schedule到最後，也不會變成1，結果雖然有改善，但是跑到20 epochs時又崩掉了，accuracy還是差baseline一點。所以我就動筆算了一下這兩次崩掉的時候，scheduling乘上係數後的值大約是多少，發現竟然是幾乎一樣，表示如果domain loss 和classifier loss的weighting漲到這個程度，就有很大的機會會崩掉，於是我就逆回去算我加的係數應該要是多少(最後是調0.5\*scheduling)，才不會崩掉。加上去之後，就都train過baseline了。