DLCV HW2

b07901169 楊宗桓

1 GAN

1.1 Model Implementation

我實作了DCGAN的model(fig.1跟fig.2):Generator Input維度為100維，Generator的loss function 目標為G(D(Z))接近1(real label)，而Discriminator的目標則是讓D(G(Z))逼近0且D(real-data)接近1。Data augmentaion的部分則是用RandomHorizontalFlip來增加training data的數量。我一開始設定Generator深度及dimensions皆與論文相同(第一層ConvTranspose2d完為1024channels)，但後來發現圖片出來精緻度很差，試著將第一層改為512channels後反而精緻度變高。

```
Generator(
  (proj): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
  (conv1): Sequential(
    (0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (conv2): Sequential(
    (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (conv3): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (conv4): Sequential(
    (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (out): Tanh()
)
```

Figure 1: DCGAN generator

```
Discriminator(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (classifier): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): Sigmoid()
  )
)
```

Figure 2: DCGAN discriminator

learning rate 也對GAN的表現影響很大，為了限制Discriminator的強勢，我將Discriminator的learning rate調成Generator的1/4，發現可以幫助Generator達到收斂(fig.3為lr-d=2e-4,lr-g=2e-4, fig.4為lr-d=5e-5,lr-g=2e-4)

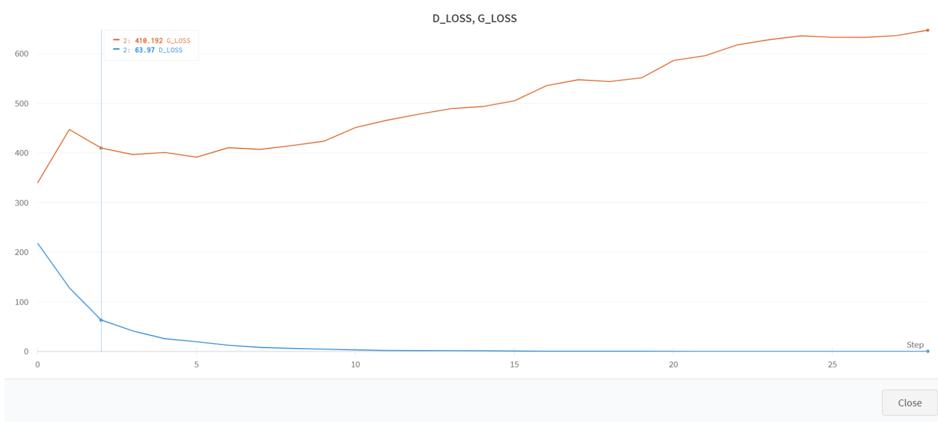

Figure 3: original lr

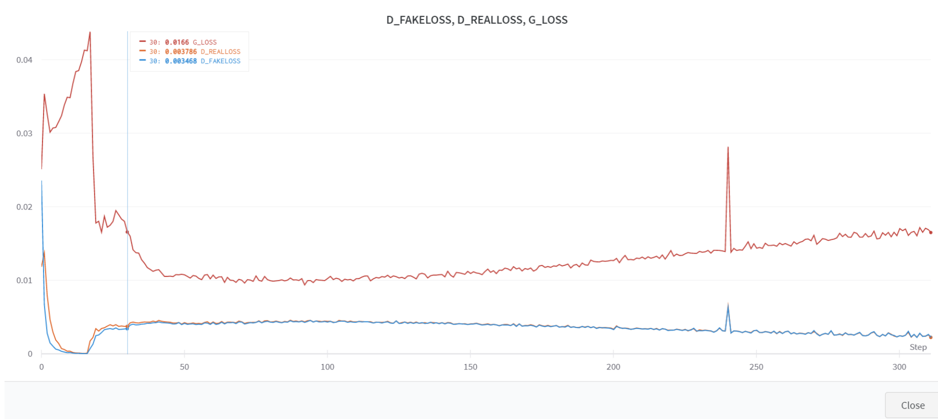

Figure 4: modified lr

另外Training時採用label smoothing(隨機對real, fake label做浮動)的FID, IS score結果也比較好(fig.5為沒有做label smoothing的結果)

1.2 Generated Images

最後採用的model使用label smoothing(fig.6)

Figure 5: unsmoothed

Figure 6: smoothed

1.3 Evaluation

Metric	Score
FID	33.5269
IS	2.0834

Table 1: Evaluation

1.4 What I learn

如果是用類似WGAN等對GAN做regulation的model，GAN真的蠻難train的，重要的是要一直懂得track generator跟discriminator的loss，若discriminator的loss在初期就迅速降低f且generator的loss也一直居高不下那麼通常這個GAN已經爛掉了因為generator已經跟不上discriminator的能力了。根據這些我做了一些更改learning rate,sceduler, 還有label smoothing 結果都還不錯，這些的大致方向都是為了限縮discriminator的學習速度。

2 ACGAN

2.1 Model Implementation

```
Generator(
  (label_emb): Embedding(10, 100)
  (l1): Linear(in_features=100, out_features=25088, bias=True)
  (conv1): Sequential(
    (0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ConvTranspose2d(512, 512, kernel_size=(2, 2), stride=(2, 2))
    (2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv2): Sequential(
    (0): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2))
    (1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv3): Sequential(
    (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (conv4): Sequential(
    (0): Conv2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Tanh()
  )
)
```

Figure 7: ACGAN generator

```
Discriminator(
  (conv_blocks): Sequential(
    (0): Conv2d(3, 32, 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(32, 64, 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): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Dropout2d(p=0.25, inplace=False)
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Dropout2d(p=0.25, inplace=False)
    (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (13): LeakyReLU(negative_slope=0.2, inplace=True)
    (14): Dropout2d(p=0.25, inplace=False)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Figure 8: ACGAN discriminator

Input端先將10維的class 我一開始做是直接將class label 跟noise直接concatenate成110維，結果model類似直接把數字搞混了(如fig.9)，後來改為將label透過nn.embedding成和noise同樣100維，之後再和noise內積丟進generator，generator的架構大致和DCGAN差不多，不過為了能抽取額外的class information在generator多加了兩層不縮小圖片大小的layer，discriminator端則output class跟real/fake. Loss function 的部分則是實作wgan和gradient penalty來更好的regularize discriminator，而相對於DCGAN，WGAN的discrimi output是沒有經過sigmoid的。

Figure 9: Concatenated Label

2.2 Evaluation

Metric	Score
Accuracy	81.2

Table 2: Evaluation

2.3 Generated Images

Figure 10: Generated Images

3 DANN

3.1 Target domain trained on source domain only

Accuracy	SVHN →MNIST-M	MNIST-M →USPS	USPS →SVHN
Accuracy	46.86%	74.8879%	24.2998%

Table 3: Evaluation

3.2 Target domain trained on source and target domain

Accuracy	SVHN →MNIST-M	MNIST-M →USPS	USPS →SVHN
Accuracy	47.03%	75.3363%	33.4116%

Table 4: Evaluation

3.3 Target domain trained on target domain only

Accuracy	SVHN →MNIST-M	MNIST-M →USPS	USPS →SVHN
Accuracy	98.31%	98.8638%	93.2314%

Table 5: Evaluation

3.4 Implementation details & what I learn

模型架構如下, 先建立一個由2層CONV layers的feature extractor之後再接一個Reverse gradient layer最後再分接class classifier/domain classifier。Data augmentation的部分為了能讓train在黑白數字的model也能很好adapt到彩色domain上，我將黑白照片轉為RGB之後再做colorJitter讓他有更多的色彩資訊。
從前面的Evaluation可以推測，source domain如果是黑白的攜帶資訊也會比較少因此adapt到彩色domain時accuracy會很低，相對的，彩色domain adapt到黑白domain上accuracy就高了許多。

```
DANN(
  (feature_extractor): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): ReLU(inplace=True)
    (4): Conv2d(64, 50, kernel_size=(5, 5), stride=(1, 1))
    (5): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Dropout2d(p=0.5, inplace=False)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): ReLU(inplace=True)
  )
  (class_classifier): Sequential(
    (0): Linear(in_features=800, 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=10, bias=True)
    (4): LogSoftmax()
  )
  (domain_classifier): Sequential(
    (0): Linear(in_features=800, 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)
    (4): LogSoftmax()
  )
)
```

Figure 11: DANN

3.5 TSNE

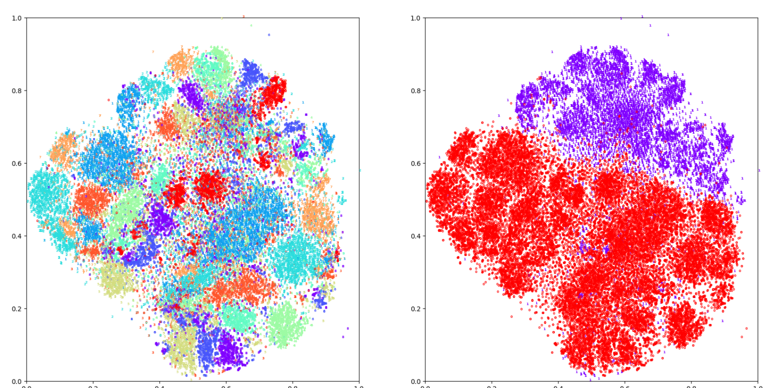

Figure 12: SVHN →MNIST-M

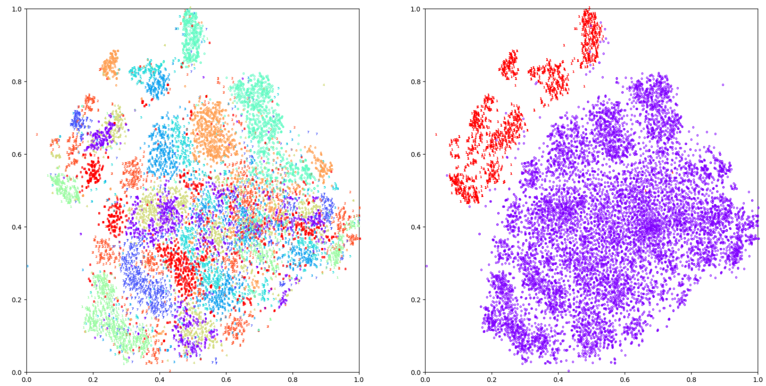

Figure 13: MNIST-M →USPS

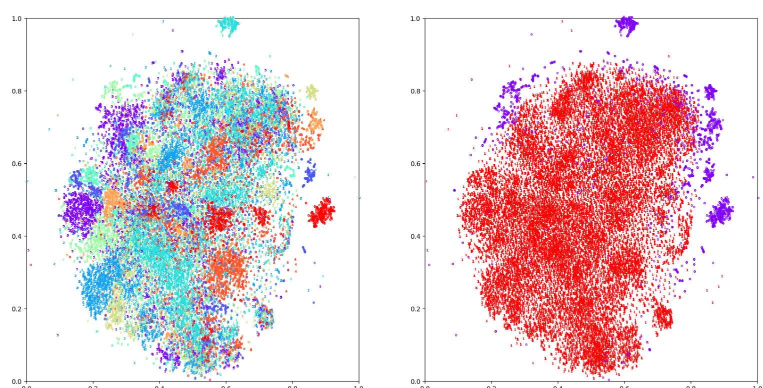

Figure 14: USPS →SVHN