ADLxMLDS2017 - HW4

電信碩一 宋易霖 r06942076

1 Problem Define

利用 conditional adversarial generative network 學出給定的 anime data 的 distribution，並且根據 testing 的條件產生對應的圖片。

2 Model description

我的 generator 和 critic 都是參考 DCGAN 的架構，用 convolution stride 取代 pooling layer，並且在 generator 加了 batch normalization，把 critic 的 activation function 換成 leaky relu 等等， 詳細的架構如 Figure 1所示。
我用的 objective funtion 是 WGAN 和 ACGAN 綜合起來的，critic 會有兩種 output，一個是輸出一個 scalar，用來計算眞的圖片以及假的圖片的 Wasserstein distance，用途是希望 generator 製作的圖片越眞越好， 並且會加上 gradient penalty 使 critic 符合 WGAN 的假設。 另一種是多維度的輸出，每一個維度對應一個頭髮的眼色或是眼睛的顏色 (label)，目的是希望 generator 製作出讓 critic 特徵分類正確的圖片。整個對抗性的架構在 Figure 2。以下列出 generator 和 critic 兩者的 objective function:
generator 要 maximize: (f 是 critic 要學習成的function， P_G則是 generator 要學習的)

$$\mathbb{E}_{x\sim P_G}(f(x)+\log P(label|x))$$

critic 要 maximize:

$$\mathbb{E}_{x\sim P_{data}}(f(x)+\log P(label|x))+\mathbb{E}_{x\sim P_G}(-f(x)+\log P(label|x))-10\times\mathbb{E}_{x\sim P_{penalty}}((||\nabla_x f(x)||-1)^2$$

其中 $logP(label|x)$ 其實就是給定 image 後， critic 輸出與眞的 label 的 maximum likelihood，實作上我是 minimize critic 輸出與 label 的 cross entropy。而 $P_{penalty}$ 就如同老師投影片的作法，是從 sample 中眞的圖片以及假的圖片隨機比例內插出來的圖片。

3 How do you improve your performance

3.1 data augmentation

使用 data augmentation 可以使 critic 看過更多眞實的圖片，也加強了 critic 分辨眞實照片的能力，如此應該也能夠逼迫 generator 要產生更眞實的圖片。

3.2 increase dimension of noise z

z 的維度可以看作是 output image 的 feature，所以 z 的維度高低同時也決定了 output image 眞正的維度。大多數人在產生 image 的時候都把 z 的維度設成 100 上下，表示他們認爲 image 的維度實際上大概就只有100左右。但是我認爲如果把維度設高一點，甚至設成跟 output image 同一個維度，這樣照理說可以包含所有 output image 可能的眞正維度，這時產生出來的 image 應該也會更眞實。

3.3 stackGAN

這部份是聽從老師在社團裡給的建議，讓機器產生 96×96 的圖片後再壓縮成 64×64 的圖片。而要直接產生 96×96 的圖片可能比較困難，因此先用一個 generator 產生 64×64 的圖，再用第二個 generator 產生 96×96 的圖。因爲我希望兩個 generator 各司其職，所以不是 end-to-end 的 training，而是分段訓練：先 train 好第一個 generator 後固定這個 generator，再把第一個 generator 的 output 當成第二個 generator 的 input 來 train 第二個 generator。

4 Experiment settings and observation

4.1 Experiment settings

基本的設定如下
1. batch size: 64

```
Generator (
  (module_list): ModuleList (
    (0): Sequential (
      (0): ConvTranspose2d(123, 512, kernel_size=(4, 4), stride=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU (inplace)
    )
    (1): Sequential (
      (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU (inplace)
    )
    (2): Sequential (
      (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU (inplace)
    )
    (3): Sequential (
      (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU (inplace)
    )
    (4): Sequential (
      (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): Sigmoid ()
    )
```

(a) Generator (text)

```
Critic (
  (binary_classifier): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
  (style_classifier): Sequential (
    (0): Conv2d(512, 23, kernel_size=(4, 4), stride=(1, 1))
    (1): Sigmoid ()
  )
  (module_list): ModuleList (
    (0): Sequential (
      (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): LeakyReLU (0.2, inplace)
    )
    (1): Sequential (
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): LeakyReLU (0.2, inplace)
    )
    (2): Sequential (
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): LeakyReLU (0.2, inplace)
    )
    (3): Sequential (
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): LeakyReLU (0.2, inplace)
    )
```

(b) Critic (text)

Figure 1: Model structures

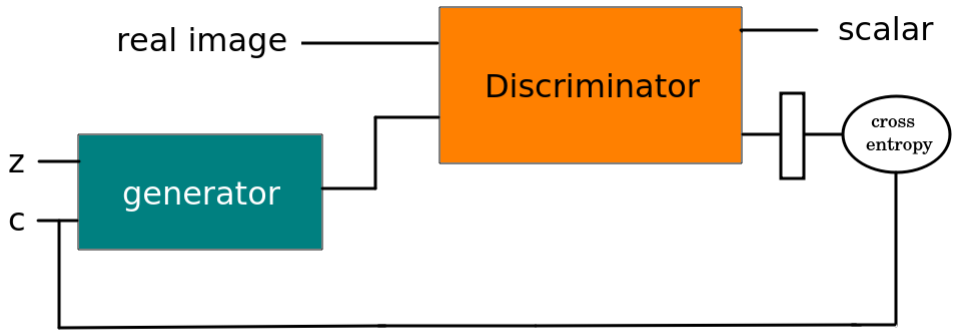


Figure 2: Generative Learning flow chart

2. z 的分佈： 輸出在 0, 1 之間的 uniform distribution
3. optimizer: Adam
4. learning rate: 0.0001
5. gradient penalty 參數： 10
6. tags 的取法： 把所有可能的頭髮顏色以及眼睛顏色形成一個 1 of N encoding，若某張圖有符合某個顏色的頭髮或是眼睛，就把該 encoding 對應的維度設成1。
7. data augmentation 的方法： sample 每筆data時，隨機左右翻轉 data 或是旋轉 -30 至 30 度。

4.2 observation

因爲 GAN 不太容易以一個 metric 來衡量 model 的好壞， 以下我對幾種不同的實驗參數或 model 跑出了幾張圖用肉眼來比較他們的好壞。若是沒特別標注的參數就是以 4-1 呈現的參數爲準。並且這些圖產生的 noise 是固定 seed 的。
A: z: 100, w/o data augmentation (train 500 epochs)
B: z: 100, w/ data augmentation (train 500 epochs)
C: z: 100 w/ data augmentation, stackGAN (兩個 generator 各 train 500 epochs)
D: z: 4096 w/ data augmentation (train 2000 epochs)

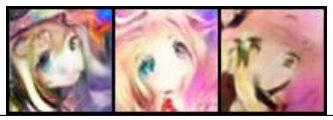
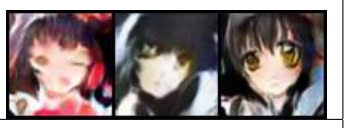
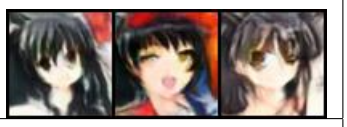
type of model	green hair, blue eyes	pink hair, gray eyes	black hair, yellow eyes
A			
B			
C			
D			

Table 1: 多種參數設定下 genetator 產生出的圖片

從 Table 1可以得到一些觀察：
1. A 和 B 的比較可以看出在這個 dataset 做了 augmentation 效果是有變好的，沒有 data augmentation 大部分圖片也都還可以，但是偶爾會出現特別崩壞的圖。
2. C 的 model 有兩個 generator，第一個 generator 就是拿 B model 的 generator， 然後固定住這個 generator 再 train 第二個 generator 500 個 epoch。B 和 C 可以比較出：先生成 96×96 再壓縮後的解析度好像有高一點。像是綠頭髮的第二張及第三張都明顯比較清楚，機器也畫得出嘴巴。紅頭髮的狀況下也是比較清楚，特別是眼睛畫的比較好。因此用 stackGAN 的作法的確得到了比較好的結果，但是因爲同時也用了比較多參數，所以如果 B 的方法加了參數也不一定會輸 C 這個作法。
3. D 的 model train 起來結果是最糟的，圖片不但比較模糊，連 condition 都錯了。不過這樣的結果後來想想也是可以預期的：首先是模糊的問題，因爲 z 的 dimension 太大，導致其實在 train 的時候有很多可能性其實 train 不太到，但是 testing 時卻有可能會 sample 到那些沒 train 過的 z，所以結果就很模糊。而 condition 吃不太到的原因也是因爲 dimension 有大約4000維，而 condition 的 dimension 大概只有 20 維左右，導致 condition 很容易被稀釋掉，因此產生的圖都對不上 condition。

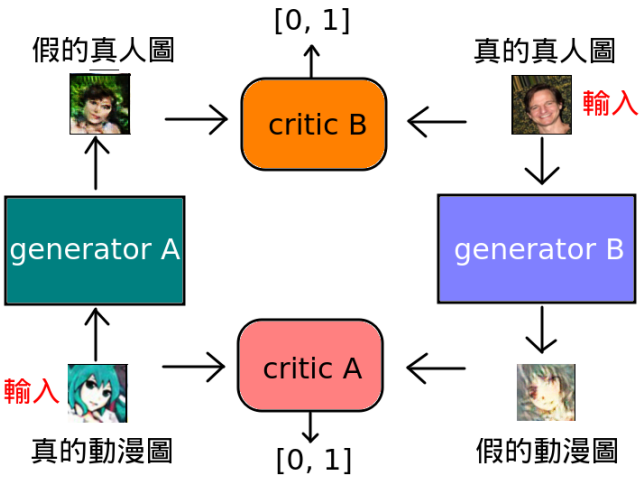
5 bonus: style transfer

5.1 dataset

我的 style transfer 做在兩個不同的 dataset 間的轉換：這個作業的 anime dataset 以及 celebA dataset。因爲這兩個 dataset 都是把頭像切在中間並且都是人臉，感覺關聯性比較大，可能做起來會比較容易，所以挑了這兩個 dataset。

5.2 implementation

實做的方法完全按照 cycleGAN，假設 anime dataset 是 domain A, celebA dataset 是 domain B，分別有兩個 critic 以及 generator: generator A 把 domain A 的圖片轉成 domain B， critic A 學會分辨眞的 domain A 的圖以及從 domain B 轉過來的假圖，而 generator B 和 critic B 做一樣的事情，只是是應用在 domain



(a) Training flow chart

```
Generator (
  (main): Sequential (
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU (inplace)
    (3): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU (inplace)
    (6): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU (inplace)
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (11): ReLU (inplace)
    (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (14): ReLU (inplace)
    (15): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (16): Sigmoid ()
  )
```

(b) generator structure

Figure 3: cycleGAN training flow chart and generator structure

B 上。整個 cycleGAN 的架構以及流程如 Figure 3a 的圖所示。
Critic 的架構和 Figure 1b 一樣， generator 則是從原本的 image 用 convolution layer downsample 三層後，再用 convolution transpose upsample 回原本 image 的大小 (Figure 3b)。
而和一般 GAN 的 objective function 不同的是，cycleGAN 除了原本的 adversarial loss 以外還加上了 cycle loss，以 domain A 的圖片爲例： 假設先取一張 domain A 的圖片，先用 generator A 轉成 domain B 的假圖後，再用 generator B 把這假圖轉回 domain A， cycle loss 就是希望轉回來的這張圖和原本的圖 pixel distance 要越近越好，加上了這個 loss 可以讓 generator A 轉換的圖不要和原圖差異太大，不然用 generator B 可能會轉不回原圖，而在 domain B 也是使用一樣的 loss。接著我們可以把generator 和 critic 的 objective function 寫出來 (只寫出一個 domain)：
generator A 要 minimize adversarial loss 和 cycle loss的和 (D 就是critic 要模擬的 function):

$$\mathbb{E}_{x\sim P_{data,A}}[D_B(G_A(x))-1]^2+10.0\,\mathcal{L}_{cycle}$$

其中：

$$\mathcal{L}_{cycle}=\mathbb{E}_{x\sim P_{data,A}}[||G_B(G_A(x))-x||_1]+\mathbb{E}_{y\sim P_{data,B}}[||G_A(G_B(y))-y||_1]$$

critic A 要 minimize:

$$\mathbb{E}_{x\sim P_{data,A}}[D_A(x)-1]^2+\mathbb{E}_{y\sim P_{data,B}}[D_A(G_B(y))^2]$$

需要注意的是這邊 adversarial loss 是要 minimize chi-square distance，而不是 Wasserstein distance，這兩個 distance 我都有做，而使用 WGAN 會讓 generator 太強調 train 成很像另一個 domain 的圖，導致跟原本的圖長比較不像，而用 chi-square distance 的結果是比較好的。

5.3 result

Figure 4 是我的 style transfer 成果，實驗設定其實跟以上的作法都一樣，這邊的成果大概 train 了 100 epochs，train 太久反而可能會爛掉 (大約 300 的時候) ... 可以發現轉成眞人應該是比較困難的，因爲有些臉都不太清楚。而動漫人物的頭髮通常誇張的長，所以轉成眞人的時候頭髮常常就會轉成背景。反之從眞

(a) Anime samples and their transform

(b) CelebA samples and their transform

Figure 4: Demo of style transfer

人轉成動畫時，背景就會變成頭髮。

6 Reference

https://arxiv.org/pdf/1703.10593.pdf

https://docs.google.com/presentation/d/1cX5hyAnP5nEsZO5NDfGribA0yfUqGmXt2RV6ETUJhec/edit#slide=id.g2caa8fd3d5_100_0

https://arxiv.org/pdf/1701.07875.pdf