MLDS HW3 Report

許芯瑜、李祐賢、熊展軒

❑ Model Description:

❑ Image Generation:

-Generator:

input(64,100)
Dense(4*4*64*8,'linear')
Reshape((64,4,4,64*8))
Conv2d_transpose(64*4, [4, 4], [2, 2])
relu_batch_norm
Conv2d_transpose(64*2, [4, 4], [2, 2])
relu_batch_norm
Conv2d_transpose(64, [4, 4], [2, 2])
relu_batch_norm
Conv2d_transpose(3, [4, 4], [2, 2])
tanh

-**Discriminator:**

Conv2d(64, [4, 4], [2, 2])
leaky_relu
Conv2d(128, [4, 4], [2, 2])
leaky_relu_batch_norm
Conv2d(256, [4, 4], [2, 2])
leaky_relu_batch_norm
Conv2d(512, [4, 4], [2, 2])
Dense(1,'linear')

❑ Text-to-image Generation:

-Generator:

input(64, 100(noise)+23(tags))
fc(64*12*12) + BN + Relu
16*Resblock: 2*(k3n64s1 + BN +Relu)
3*(k3n256s1 + Pixel shuffle + BN + Relu)
k9n3s1 conv layer
tanh

-Discriminator:

input(64,64,64,3)
k4n32s2 conv layer
2*Resblock: 2*(k3n32s1 + leaky relu)

k4n64s2 conv layer
2*Resblock: 2*(k3n64s1 + leaky relu)
k4n128s2 conv layer
2*Resblock: 2*(k3n128s1 + leaky relu)
k4n256s2 conv layer
2*Resblock: 2*(k3n256s1 + leaky relu)
k4n512s2 conv layer
2*Resblock: 2*(k3n512s1 + leaky relu)
k4n1024s2 conv layer
flatten
fc(1) # 用來分類真假
fc(23) # 用multilabel classify分類頭髮、眼睛

❑ Experiment setting and observation:

❑ Image Generation:

❑ Experiment setting:

batch_size = 64
iteration: 50000

❑ Observation:

batch_size不適合太大,會學到比較模糊的images,並且train超過100000iteration之後多樣性會下降.

❑ Text-to-image Generation:

❑ Experiment setting:

data augmentation: 水平翻轉、左右旋轉5度
batch size = 64
iteration = 50,000
learning rate = 2e-4
beta1 = 0.5
beta2 = 0.9
loss比例 = 25(分類真假):1(分類attribute)
training次數比例= 1(discriminator):3(generator)

❑ Obervation:

在loss比例方面，一開始我們設為1:1，但這樣會使得gan太重視分類attribute，瓜分掉分類真假的重要性，而間接使得generator產生出的照片不像動漫人物。為了避免這種情況，我們將分類真假的比例調高。

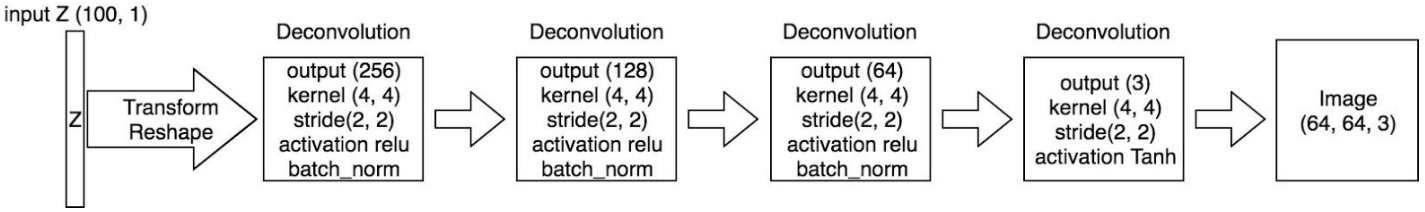
在training次數方面，一開始我們也將次數比例設為1:1，但discriminator太強會導致generator沒有一個學習的施力點，會導致generator一蹶不振，始終無法產生清晰的動漫圖像。所以我們將generator的training次數提高，在discriminator和generator勢均力敵的情況下，產生的動漫圖像會

比較清晰。

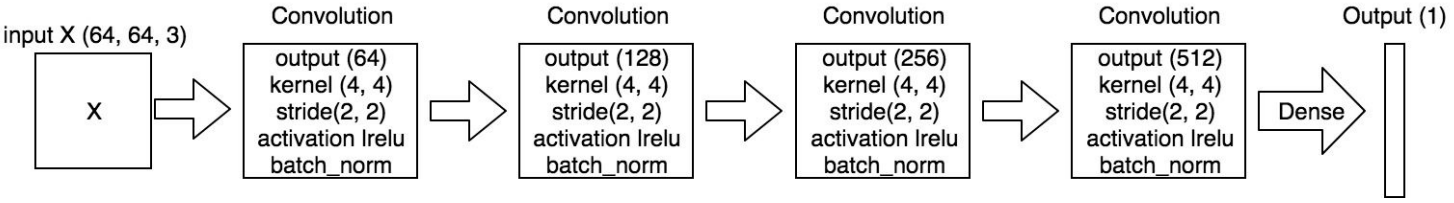
❑ Compare your model with WGAN-GP:

❑ Model Description of the choosed model

❑ Generator



❑ Discriminator



❑ Loss

不取log並使用gradient panelty。

❑ Result of the model

❑ **Comparison Analysis**

Original GAN	WGAN-GP
	

WGAN-GP的結果看起來比較好，一樣會產生糟糕的頭像，但比例上明顯較少，且不少頭像都很像人畫的。Original GAN產生的圖像大多數都有很多雜訊，高品質的頭像也少很多。

由結果可以推斷Gradient Panelty的確可以幫助GAN學習到較好的Distribution（或是可以更快的學到），且Original GAN訓練過多的epoch會炸掉，WGAN-GP似乎沒有這個問題，實驗上WGAN-GP確實有穩定訓練過程的效果。

❑ Training tips for improvement(only for image generation)

❑ Tip 1: Normalize the inputs

❑ Implementation:

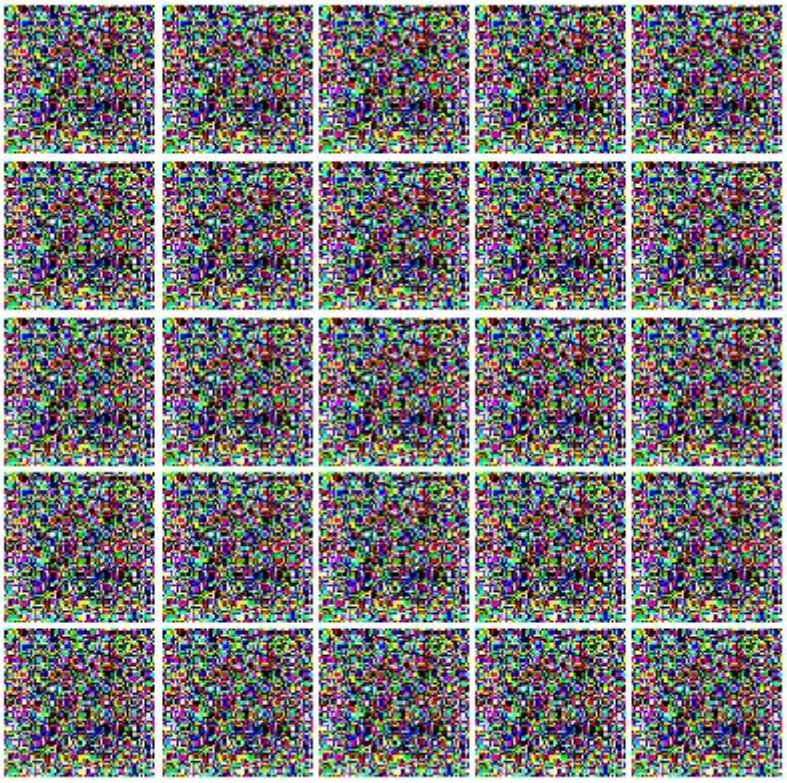
code:

```
X_mb = self.xs[self.batch_offset:self.batch_offset+self.batch_size]
xs = (X_mb/255.)
```

若沒有做此步驟是train不起來的, 我認為是因為給generation的noise是介於0~1之間的random vector,所以input也要介於0~1之間.

❑ **Results & Analysis:**

以下展示沒有Input normalization的結果:



❑ **Tip 2: Batch_Norm**

❑ **Implementation:**

code:

```
activation_fn=leaky_relu_batch_norm)
activation_fn=leaky_relu_batch_norm)
activation_fn=leaky_relu_batch_norm)
```

❑ **Results & Analysis:**

以下展示沒有Batch_Norm的結果:

❑ Tip 3: A modified loss function

❑ Implementation:

Code:

```
self.G_loss = tf.reduce_mean(G_fake)
self.D_loss = tf.reduce_mean(D_real) - tf.reduce_mean(D_fake)

# gradient penalty
epsilon = tf.random_uniform([], 0.0, 1.0)
x_hat = epsilon * self.xs + (1 - epsilon) * self.gs
d_hat = self.build_D(x_hat, True)
ddx = tf.gradients(d_hat, x_hat)[0]
ddx = tf.sqrt(tf.reduce_sum(tf.square(ddx), axis=1))
ddx = tf.reduce_mean(tf.square(ddx - 1.0) * 10.)

self.D_loss = self.D_loss + ddx
```

我們使用Gradient Panelty Loss(WGAN-GP)來優化訓練過程，從實驗中可以看出使用這樣的Loss的確可以有效的穩定學習的過程，就算Train到很多Epochs產生圖片的品質也是穩定的上升（目測）。

❑ Results & Analysis:

比較分析結果如“Compare your model with WGAN-GP”

❑ Style Transfer (Horse <==> Zebra)

❑ Results

❑ Analysis

大多數的結果都相當的好，可以有效的辨識出馬的形體並正確的轉換成斑馬，但是還是有不少錯誤，像是有些圖看起來完全沒有變化（例如右下角），應該是model沒有辨識出馬出現在這張圖上，導致幾乎沒有變化的結果，推測應該是dataset中缺乏只有馬臉的照片，導致model只看到馬臉的話沒有辦法正確的辨識出馬。

還有有時候model會大幅的改動背景畫面，model會明顯的改變色調或是扭曲圖形（或產生雜訊），可能model除了辨識馬轉換為斑馬外，還對整張圖像的結果有特定的偏好（推測）。

❑ 分工表:

- ❑ HW3-1: 李祐賢、熊展軒
- ❑ HW3-2: 許芯瑜
- ❑ HW3-3: 熊展軒