DLCV HW4

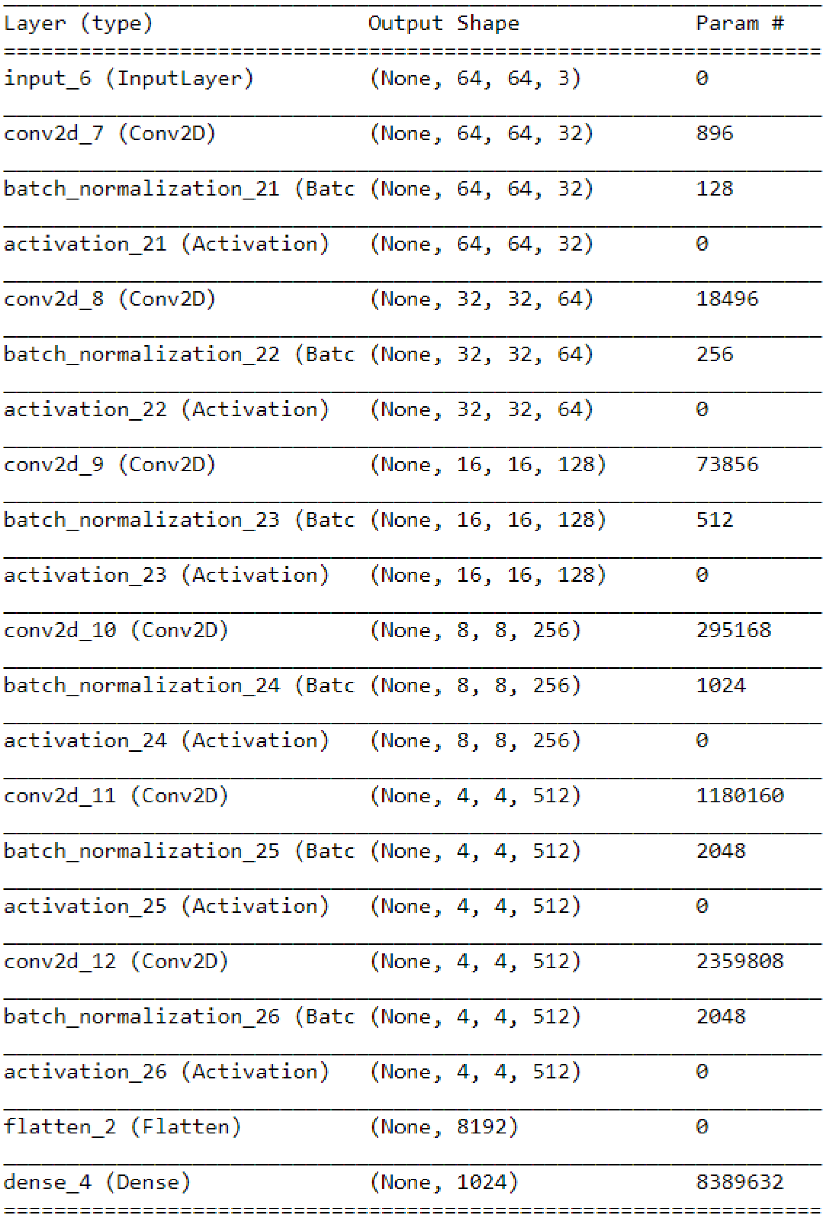
Name: 王冠驊 Dep.:電信碩二 Student ID:R05942102

Problem 1. VAE

1.

Encoder:

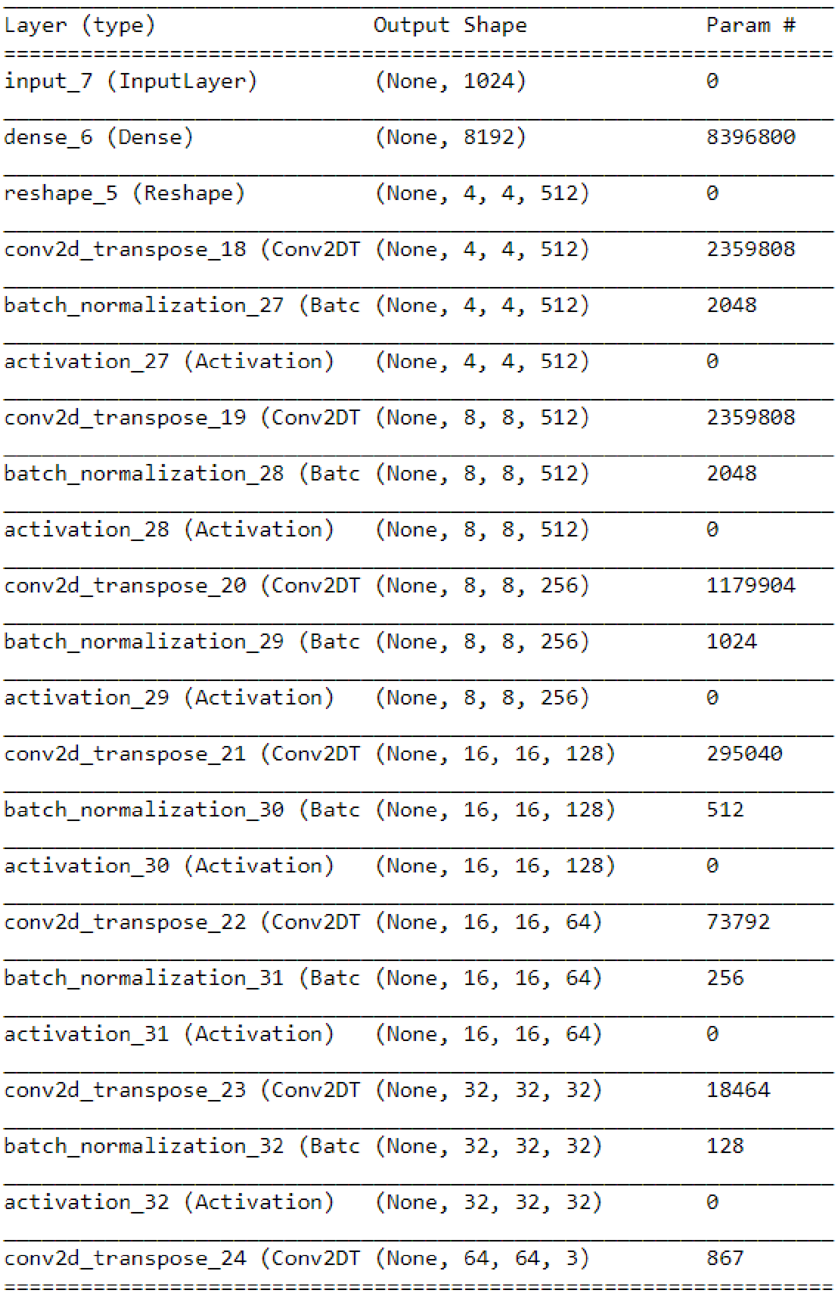
在 encoder 的部分，我使用了 6 層的 Convolutional layers，每一層均使用 batch normalization，並且使用 ReLu 當做 activation function。最終，我將每一張 input image 壓縮成一個 1024 維的 vector。詳細的模型參數如下:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
batch_normalization_21 (Batc (None, 64, 64, 32)        128
_________________________________________________________________
activation_21 (Activation)   (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_22 (Batc (None, 32, 32, 64)        256
_________________________________________________________________
activation_22 (Activation)   (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
batch_normalization_23 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
activation_23 (Activation)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 256)         295168
_________________________________________________________________
batch_normalization_24 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
activation_24 (Activation)   (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 512)         1180160
_________________________________________________________________
batch_normalization_25 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
activation_25 (Activation)   (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
batch_normalization_26 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
activation_26 (Activation)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              8389632
=================================================================
```

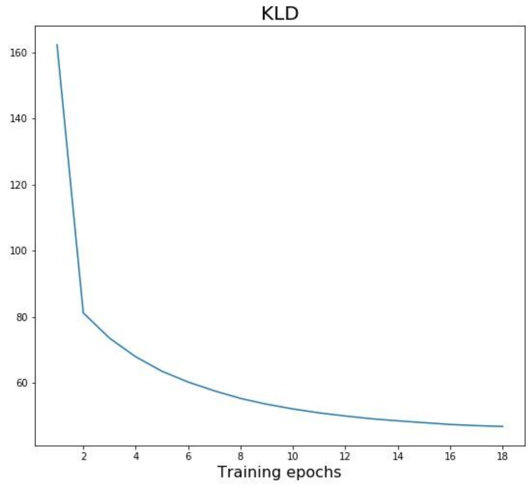
Decoder:

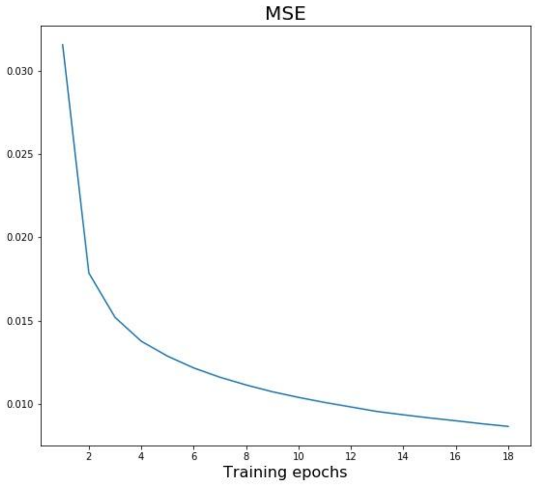
在 decoder 的部分，我使用了 7 層的 Convolutional transpose layers，每一層均使用 batch normalization，並且使用 ReLu 當做 activation function。最終，我將一個 1024 維的 vector 解回大小為 64*64*3 的 image。詳細的模型參數如下:



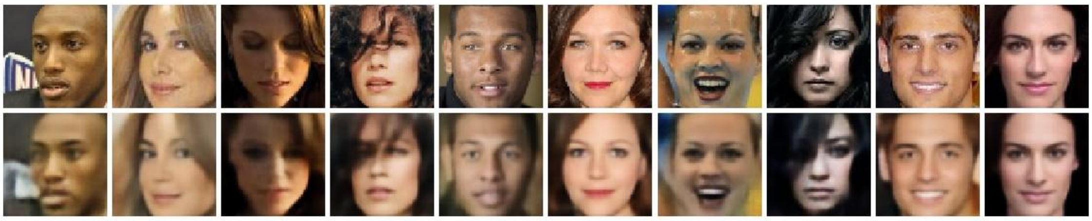
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 8192)              8396800
_________________________________________________________________
reshape_5 (Reshape)          (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_transpose_18 (Conv2DT (None, 4, 4, 512)         2359808
_________________________________________________________________
batch_normalization_27 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
activation_27 (Activation)   (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_transpose_19 (Conv2DT (None, 8, 8, 512)         2359808
_________________________________________________________________
batch_normalization_28 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
activation_28 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_transpose_20 (Conv2DT (None, 8, 8, 256)         1179904
_________________________________________________________________
batch_normalization_29 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
activation_29 (Activation)   (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_transpose_21 (Conv2DT (None, 16, 16, 128)       295040
_________________________________________________________________
batch_normalization_30 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
activation_30 (Activation)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_transpose_22 (Conv2DT (None, 16, 16, 64)        73792
_________________________________________________________________
batch_normalization_31 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
activation_31 (Activation)   (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_transpose_23 (Conv2DT (None, 32, 32, 32)        18464
_________________________________________________________________
batch_normalization_32 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
activation_32 (Activation)   (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_transpose_24 (Conv2DT (None, 64, 64, 3)         867
=================================================================
```

2.



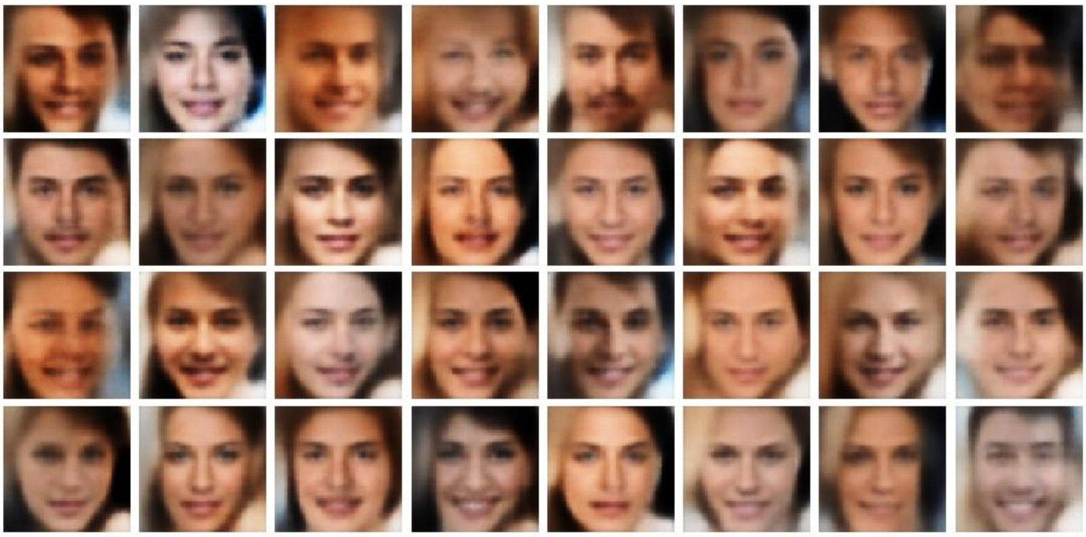


3.

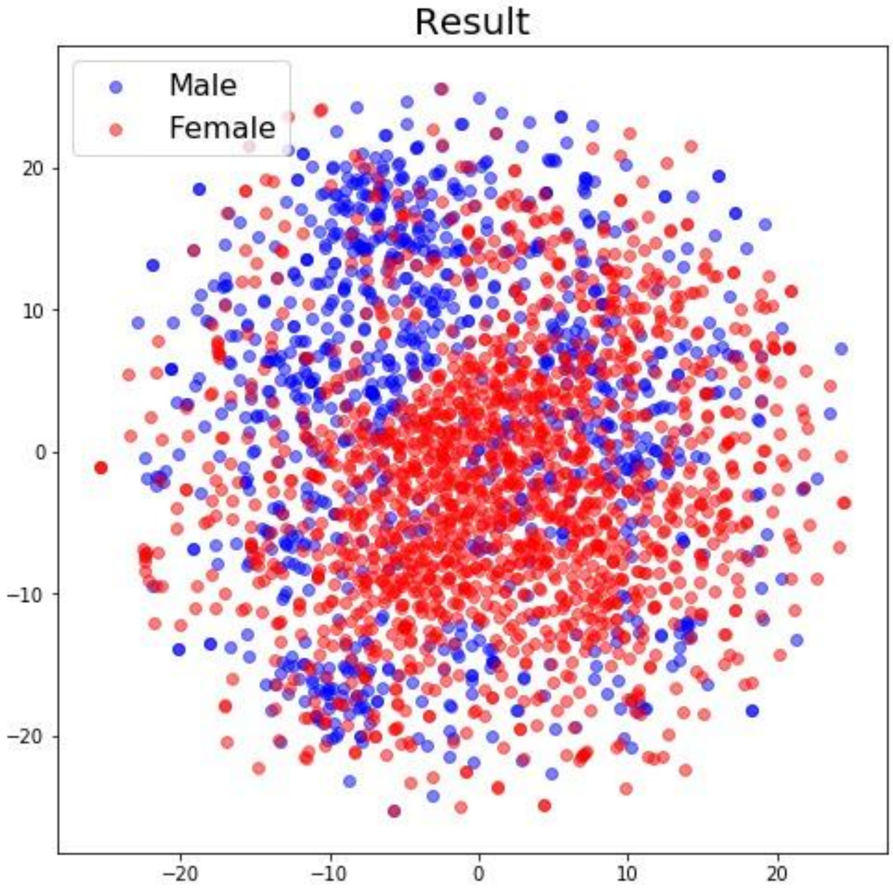


MSE = 0.8643 (pixel-wise)

4.



5.



6.

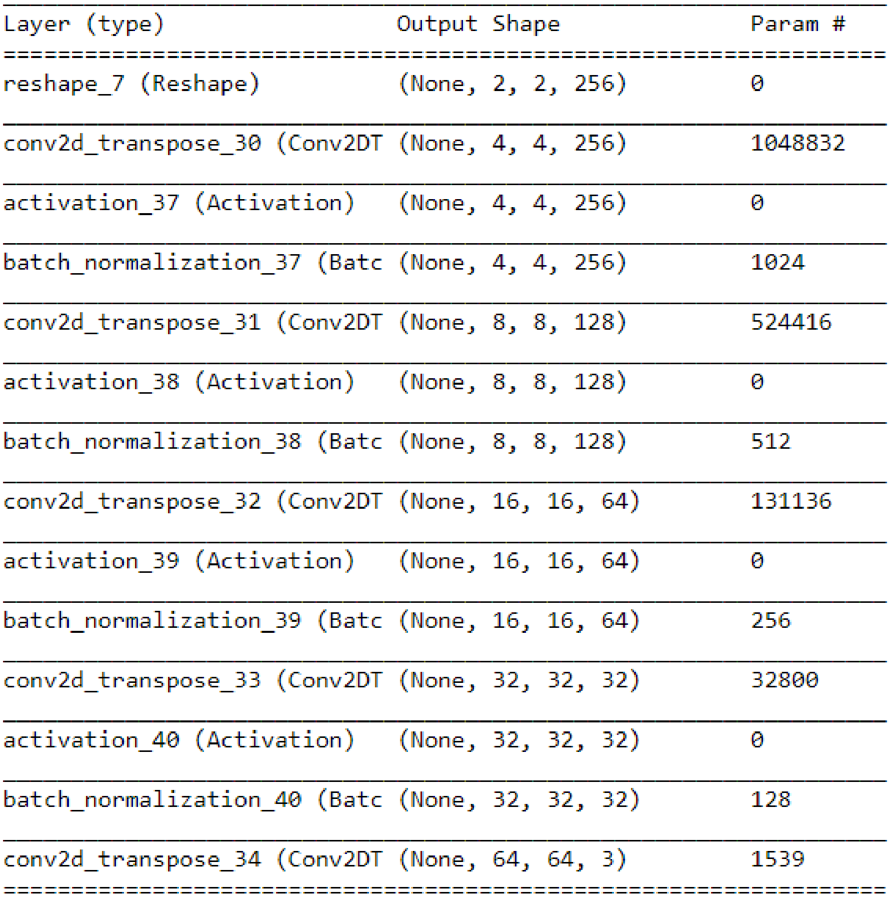
Vae 與一般的 autoencoder 最主要的差別就是他加入了 KL Divergence 來限制 latent space 必須呈現常態分佈。這個 loss 將與 reconstruction error 合併為 VAE 真正的 loss。我們可以夠過調整 2 者的比例來控制整個 model 的特性。如果 VAE loss 中 reconstruction error 占很大的比例(lambda 小)會發現使用 decoder random generate 的圖變得不像人臉，但如果將 KL Divergence loss 的比例調太高，會發現 reconstruction 出來的 image 會很糊。

Problem 2. GAN

1.

Generator:

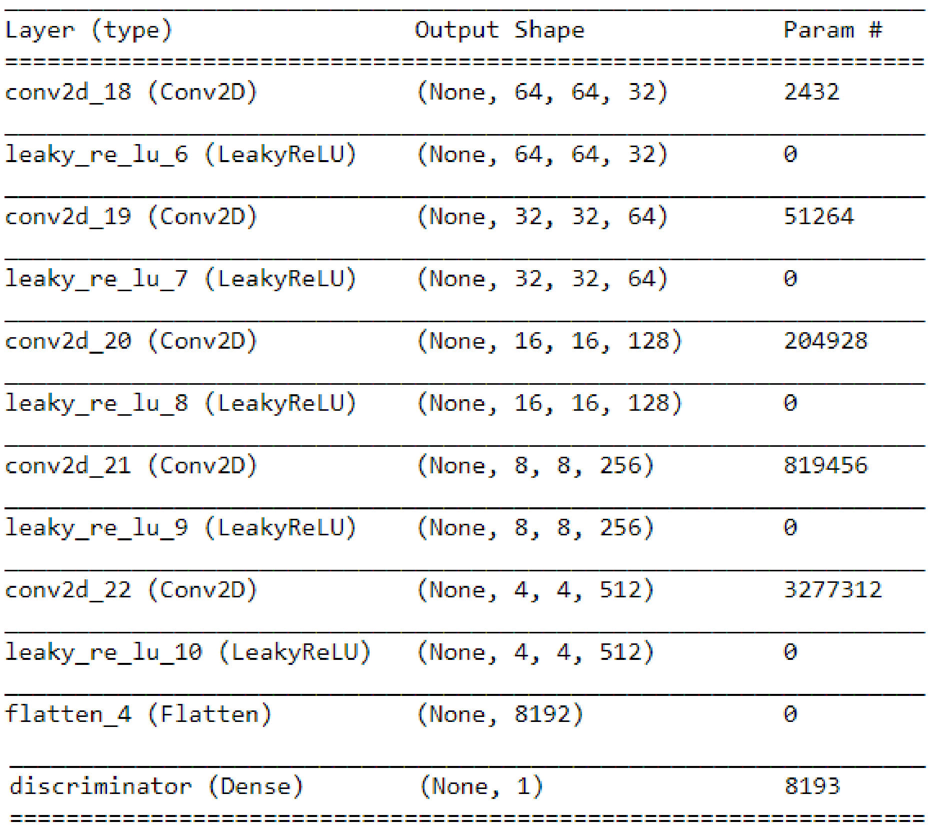
Generator 中包含 5 層的 convolutional layers，每一層均使用 batch normalization，並且使用 ReLu 當做 activation function。我們將 generator 的 input reshape 成 2*2*256 的大小，透過 5 層的 convolutional layers，最終得到 64*64*3 的 image。詳細的模型參數如下:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
reshape_7 (Reshape)          (None, 2, 2, 256)         0         
_________________________________________________________________
conv2d_transpose_30 (Conv2DT (None, 4, 4, 256)         1048832   
_________________________________________________________________
activation_37 (Activation)   (None, 4, 4, 256)         0         
_________________________________________________________________
batch_normalization_37 (Batc (None, 4, 4, 256)         1024      
_________________________________________________________________
conv2d_transpose_31 (Conv2DT (None, 8, 8, 128)         524416    
_________________________________________________________________
activation_38 (Activation)   (None, 8, 8, 128)         0         
_________________________________________________________________
batch_normalization_38 (Batc (None, 8, 8, 128)         512       
_________________________________________________________________
conv2d_transpose_32 (Conv2DT (None, 16, 16, 64)        131136    
_________________________________________________________________
activation_39 (Activation)   (None, 16, 16, 64)        0         
_________________________________________________________________
batch_normalization_39 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_transpose_33 (Conv2DT (None, 32, 32, 32)        32800     
_________________________________________________________________
activation_40 (Activation)   (None, 32, 32, 32)        0         
_________________________________________________________________
batch_normalization_40 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_transpose_34 (Conv2DT (None, 64, 64, 3)         1539      
=================================================================
```

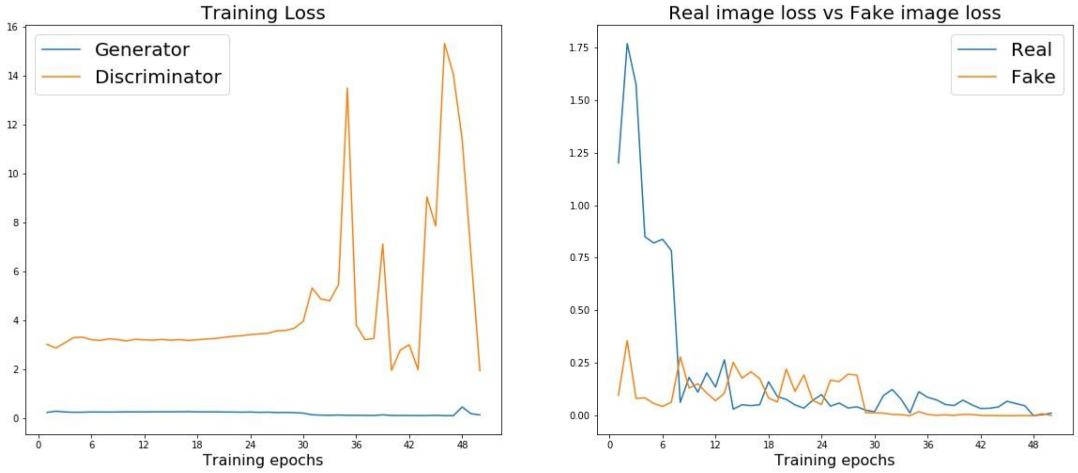
Discriminator:

在 discriminator 的部分，我使用了 5 層的 Convolutional layers，每一層均使用 LeakyReLu 當做 activation function。最終，我將每一張 input image 經過 discriminator 後會的到一個代表 discriminator 認為這張圖片真偽的數值。詳細的模型參數如下:



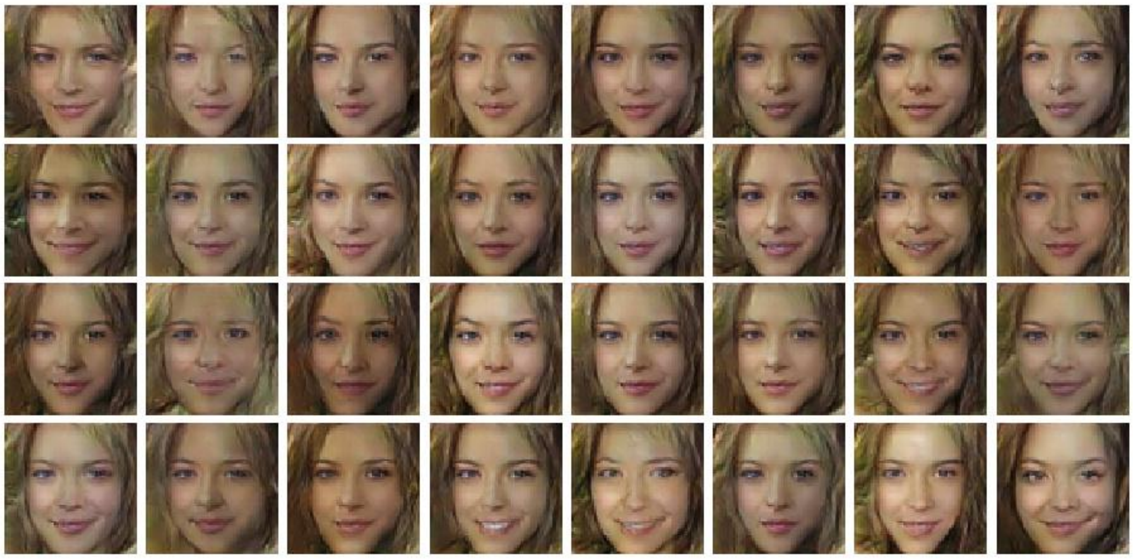
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 64, 64, 32)        2432
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 32, 32, 64)        51264
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 128)       204928
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 8, 8, 256)         819456
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 4, 4, 512)         3277312
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 8192)              0

_________________________________________________________________
discriminator (Dense)        (None, 1)                 8193
=================================================================
```

2.



GAN 是透過 Generator 與 Discriminator 彼此互相對抗，互相學習。在對抗的過程中 Generator 與 Discriminator 必須是有差不多強度的(生成圖片的強度與分辨真偽的強度)。由左邊的圖可以發現 Generator 與 Discriminator 的 loss 在前 30 個 epoch 都是十分穩定的，並沒有隨著訓練而升高或變低，這也顯示 Generator 與 Discriminator 彼此的強度相當。而在 30 個 epoch 之後發現 Discriminator 的 loss 變得十分不穩，推測 Generator 與 Discriminator 彼此的對抗出現失衡，我們的確也發現在此 epoch 後，Generator 所生成的圖片已經徹底爛掉。

3.



4.
GAN 真的是一個十分難訓練的網路。訓練 GAN 的 trick 眾說紛紜，但都不見得有效。在 implement 的過程中花了很多的時間在嘗試不同的 model 架構以及參數，雖然最後有成功 train 出一個看似不錯的 GAN，然而依然不太清楚確切訓練 GAN 的要點。最後推測似乎是太大的網路容易造成 GAN 的訓練失敗。

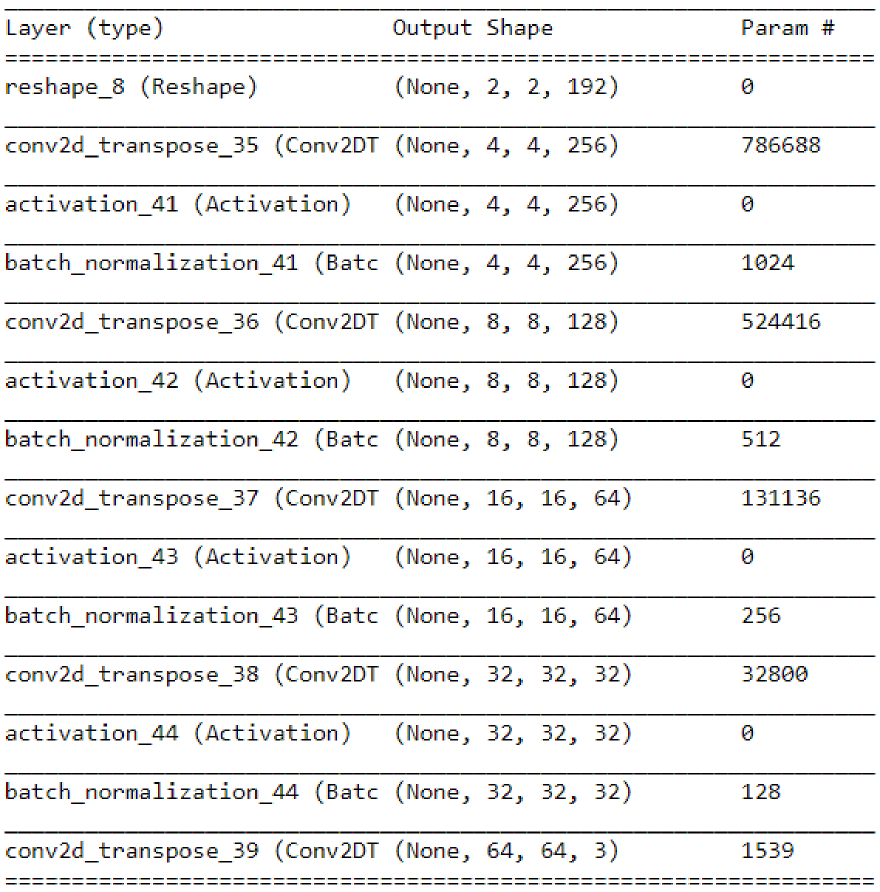
5.
從 VAE 與 GAN 的結果看來，最明顯的差異就是 image 成像的品質。GAN 所產生的 image 相較於 VAE 清晰許多，不會再有如 VAE 產生的 image 有糊糊的感覺。然而 GAN 的缺點就是比 VAE 難訓練很多，失敗率極高，且常常不知為何失敗。但是透過多次調參數或是修改 model 架構我們可以得到比 VAE 好上許多的 image 品質。

Problem 2. ACGAN

1.

ACGAN 的架構與 GAN 的架構大同小異，只是我們分別在 Generator 的 input 端多加了一個可以吃 attribute 的接口，以及在 Discriminator 端多輸出一個分辨是哪一個 attribute 的 output。此外，為了使 ACGAN 中 Generator 與 Discriminator 的參數比例接近 GAN 中 Generator 與 Discriminator 的參數比例，我們將原本在 GAN 中的 latent space dimension 由 1024 降低至 512。詳細的模型參數如下:

Generator

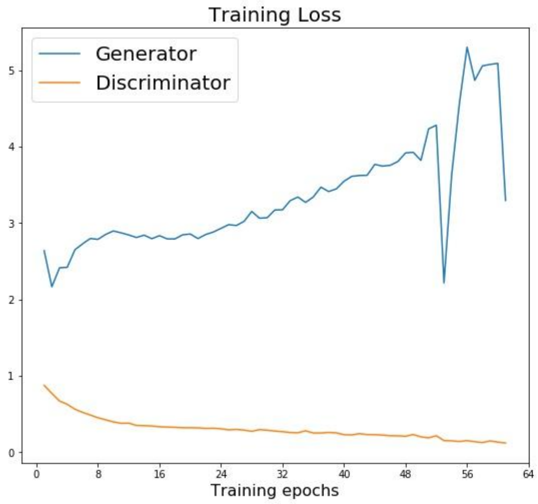


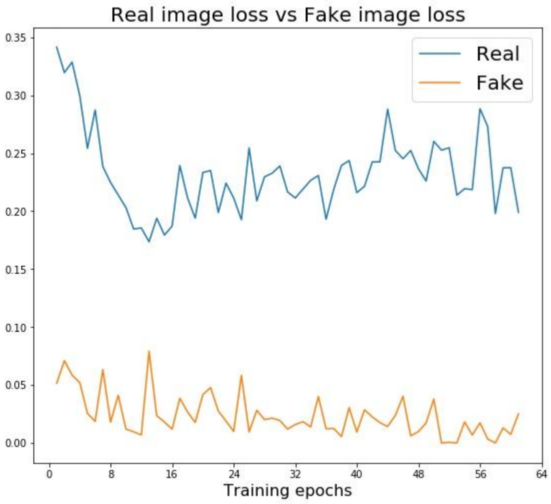
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
reshape_8 (Reshape)          (None, 2, 2, 192)         0         
_________________________________________________________________
conv2d_transpose_35 (Conv2DT (None, 4, 4, 256)         786688    
_________________________________________________________________
activation_41 (Activation)   (None, 4, 4, 256)         0         
_________________________________________________________________
batch_normalization_41 (Batc (None, 4, 4, 256)         1024      
_________________________________________________________________
conv2d_transpose_36 (Conv2DT (None, 8, 8, 128)         524416    
_________________________________________________________________
activation_42 (Activation)   (None, 8, 8, 128)         0         
_________________________________________________________________
batch_normalization_42 (Batc (None, 8, 8, 128)         512       
_________________________________________________________________
conv2d_transpose_37 (Conv2DT (None, 16, 16, 64)        131136    
_________________________________________________________________
activation_43 (Activation)   (None, 16, 16, 64)        0         
_________________________________________________________________
batch_normalization_43 (Batc (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_transpose_38 (Conv2DT (None, 32, 32, 32)        32800     
_________________________________________________________________
activation_44 (Activation)   (None, 32, 32, 32)        0         
_________________________________________________________________
batch_normalization_44 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_transpose_39 (Conv2DT (None, 64, 64, 3)         1539      
=================================================================
```

Discriminator:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 64, 64, 32)        2432
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 32, 32, 64)        51264
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 16, 16, 128)       204928
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 8, 8, 256)         819456
_________________________________________________________________
leaky_re_lu_24 (LeakyReLU)   (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 4, 4, 512)         3277312
_________________________________________________________________
leaky_re_lu_25 (LeakyReLU)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten_7 (Flatten)          (None, 8192)              0
_________________________________________________________________
generation (Dense)                (None, 1)               8193
_________________________________________________________________
auxiliary (Dense)                 (None, 1)               8193
=================================================================
```

2.





與 GAN 相同我們觀察 ACGAN 中 Generator 與 Discriminator 的 training loss(左圖)與 Discriminator 在吃入 real image 與 Generator 所生出來的 fake image 的 loss。透過觀察這些 loss，我們可以發現在訓練 ACGAN 時是比訓練 GAN 時要穩定的許多。另外我們也發現 Generator 的 training loss 大約在第 50 個 epoch 的時候出現劇烈的震盪，推測 Generator 與 Discriminator 彼此的對抗出現失衡，我們的確也發現在此 epoch 後，Generator 所生成的圖片已經徹底爛掉。

3.

上為 Female(0)，下為 Male(1)