Name: 黃宇謙　Dep.:資工碩一　Student ID:R06922072

# Problem 1. VAE (6%)

1. (1%) Describe the architecture & implementation details of your model.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 64, 64, 3) | 0 | |
| encoder_conv1 (Conv2D) | (None, 64, 64, 3) | 39 | input_1[0][0] |
| encoder_conv2 (Conv2D) | (None, 32, 32, 64) | 832 | encoder_conv1[0][0] |
| batch_normalization_1 (BatchNor | (None, 32, 32, 64) | 256 | encoder_conv2[0][0] |
| encoder_conv3 (Conv2D) | (None, 16, 16, 64) | 16448 | batch_normalization_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 16, 16, 64) | 256 | encoder_conv3[0][0] |
| encoder_conv4 (Conv2D) | (None, 8, 8, 128) | 32896 | batch_normalization_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 8, 8, 128) | 512 | encoder_conv4[0][0] |
| encoder_conv5 (Conv2D) | (None, 4, 4, 128) | 65664 | batch_normalization_3[0][0] |
| batch_normalization_4 (BatchNor | (None, 4, 4, 128) | 512 | encoder_conv5[0][0] |
| encoder_conv6 (Conv2D) | (None, 2, 2, 256) | 131328 | batch_normalization_4[0][0] |
| batch_normalization_5 (BatchNor | (None, 2, 2, 256) | 1024 | encoder_conv6[0][0] |
| encoder_conv7 (Conv2D) | (None, 1, 1, 256) | 262400 | batch_normalization_5[0][0] |
| batch_normalization_6 (BatchNor | (None, 1, 1, 256) | 1024 | encoder_conv7[0][0] |
| mean_layer (Conv2D) | (None, 1, 1, 512) | 131584 | batch_normalization_6[0][0] |
| logvar_layer (Conv2D) | (None, 1, 1, 512) | 131584 | batch_normalization_6[0][0] |
| mean_reshape (Reshape) | (None, 512) | 0 | mean_layer[0][0] |
| logvar_reshape (Reshape) | (None, 512) | 0 | logvar_layer[0][0] |
| latent_layer (Lambda) | (None, 512) | 0 | mean_reshape[0][0]<br>logvar_reshape[0][0] |
| z_reshape (Reshape) | (None, 1, 1, 512) | 0 | latent_layer[0][0] |
| decoder_deconv1 (Conv2DTranspos | (None, 2, 2, 256) | 524544 | z_reshape[0][0] |
| decoder_deconv2 (Conv2DTranspos | (None, 6, 6, 256) | 1048832 | decoder_deconv1[0][0] |
| cropping2d_1 (Cropping2D) | (None, 4, 4, 256) | 0 | decoder_deconv2[0][0] |
| decoder_deconv3 (Conv2DTranspos | (None, 10, 10, 128) | 524416 | cropping2d_1[0][0] |
| cropping2d_2 (Cropping2D) | (None, 8, 8, 128) | 0 | decoder_deconv3[0][0] |
| decoder_deconv4 (Conv2DTranspos | (None, 18, 18, 128) | 262272 | cropping2d_2[0][0] |
| cropping2d_3 (Cropping2D) | (None, 16, 16, 128) | 0 | decoder_deconv4[0][0] |
| decoder_deconv5 (Conv2DTranspos | (None, 34, 34, 64) | 131136 | cropping2d_3[0][0] |
| cropping2d_4 (Cropping2D) | (None, 32, 32, 64) | 0 | decoder_deconv5[0][0] |
| decoder_deconv6 (Conv2DTranspos | (None, 66, 66, 64) | 65600 | cropping2d_4[0][0] |
| cropping2d_5 (Cropping2D) | (None, 64, 64, 64) | 0 | decoder_deconv6[0][0] |
| decoder_mean_squash (Conv2D) | (None, 64, 64, 3) | 195 | cropping2d_5[0][0] |
| KL_loss (Lambda) | (None, 1) | 0 | mean_reshape[0][0]<br>logvar_reshape[0][0] |

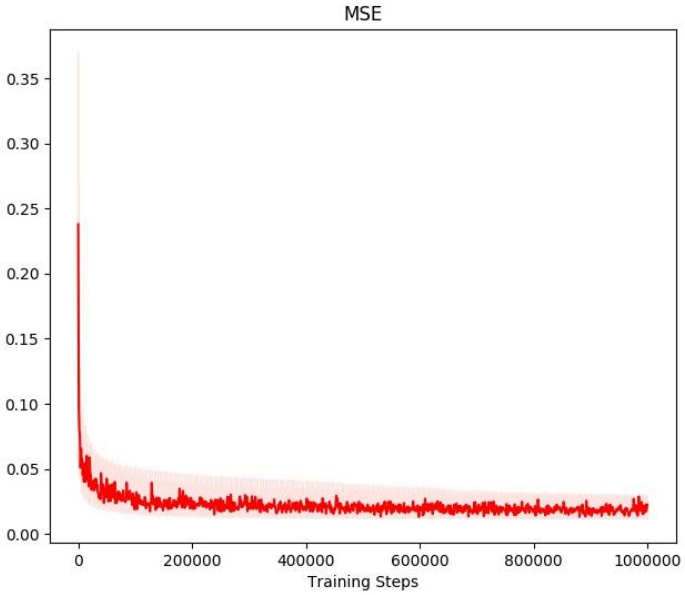
Total params: 3,333,354
Trainable params: 3,331,562
Non-trainable params: 1,792

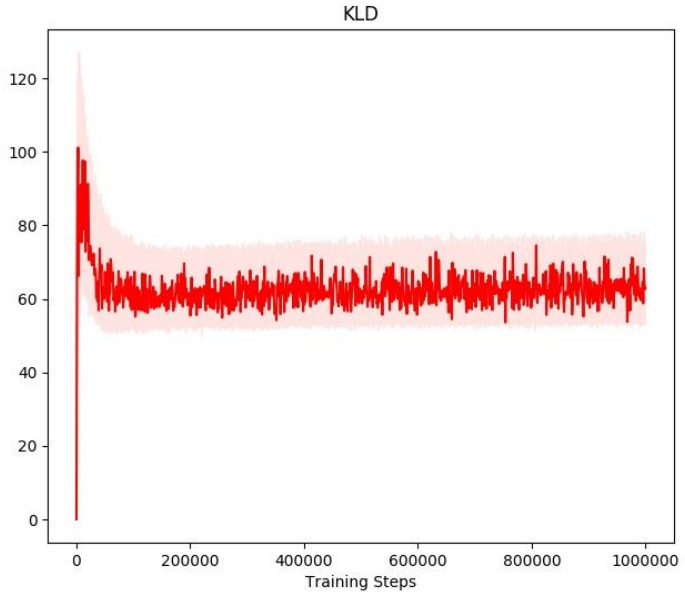
上圖為 VAE model 的架構，主要分為兩個部分 encoder 和 decoder。在 encoder 的地方，我使用多層 convolution layers 對原始影像作提取 feature 的動作，並將其壓縮成 $1 \times 1 \times 512$的大小，將其作為 latent vector(512 維)。而 decoder 的部分則是使用

conv2dtranspose 的方式將向量還原成圖像的大小，以此達到 decode 的行為。

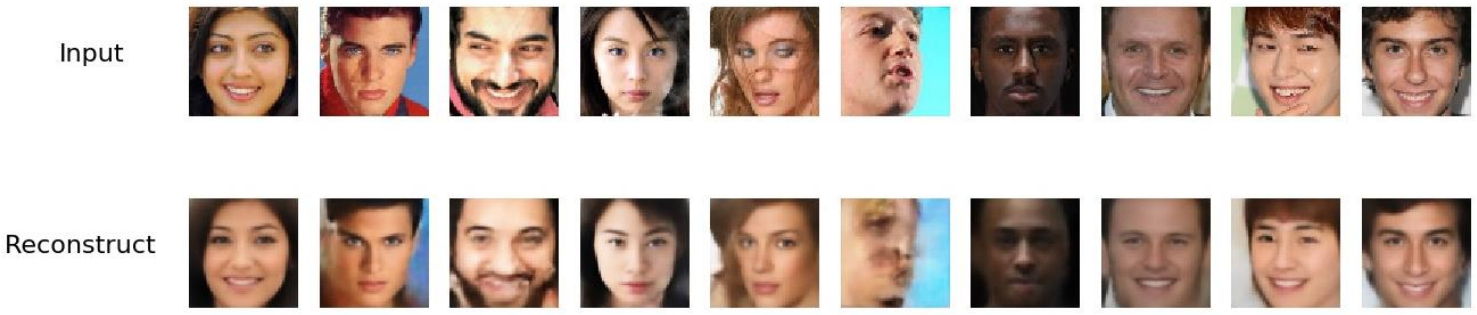
Optimizer 使用 adam 並將 learning rate 設為$10^{-4}$，而 Loss 分別有 KL loss 和 MSE，分別評估產生出來的 latent vector 和高斯分布( N(0,1) )的差距和產生出的影像與原始影像的距離。$\lambda_{KL}$設定成$10^{-4}$，使 VAE model 不會只看重 MSE 的部分，更加考慮到生成的 latent vector 與高斯分布的差距。(越高越重視 KL loss。)

2. (1%) Plot the learning curve (reconstruction loss & KL divergence) of your model.



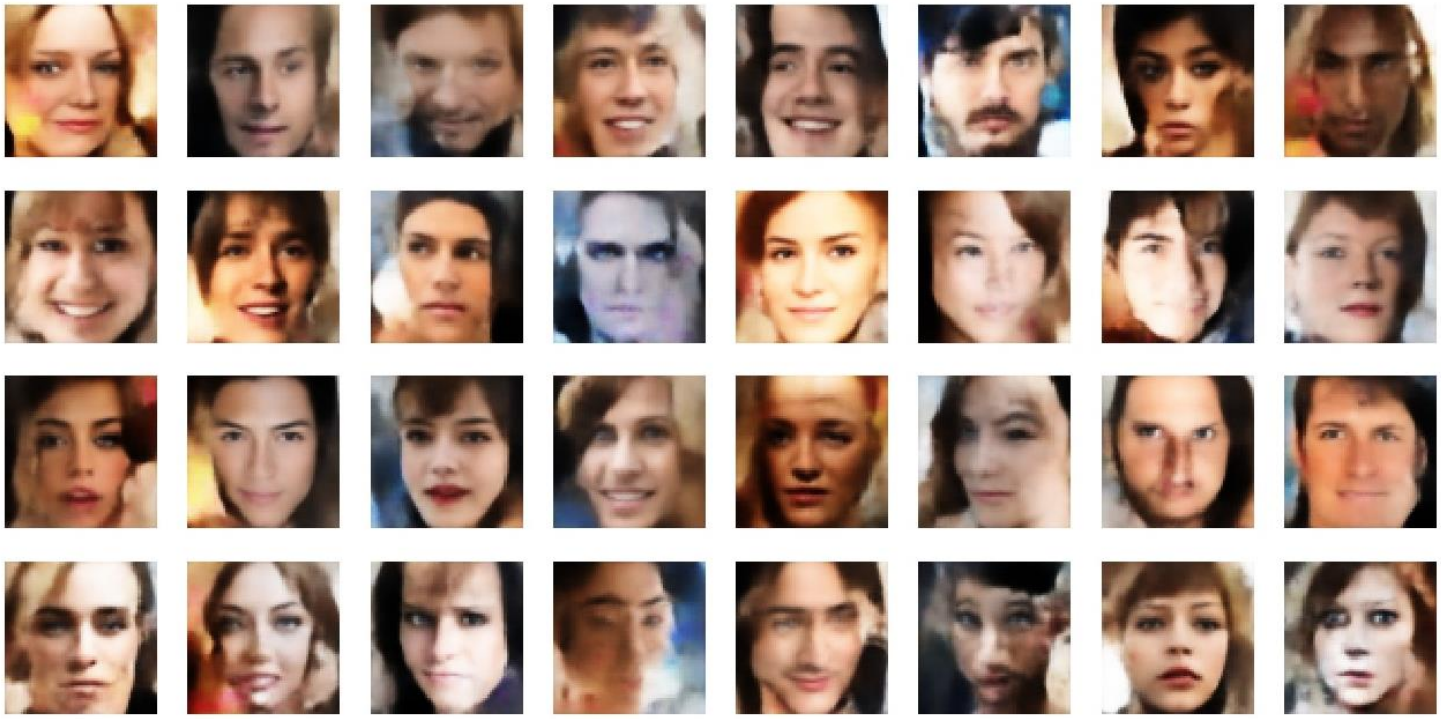




3. (1%) Plot 10 testing images and their reconstructed result of your model and report your testing MSE of the entire test set.

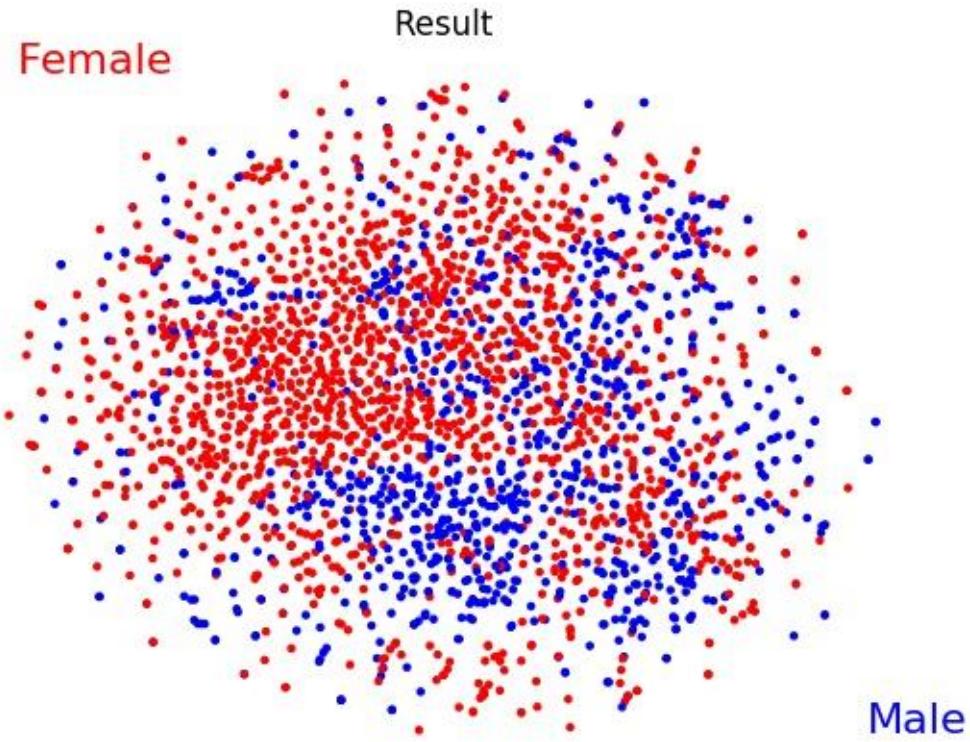



| | MSE | Normalization MSE |
|---|---|---|
| Entire test set | 73.64245 | 0.02529 |

4. (1%) Plot 32 random generated images of your model.



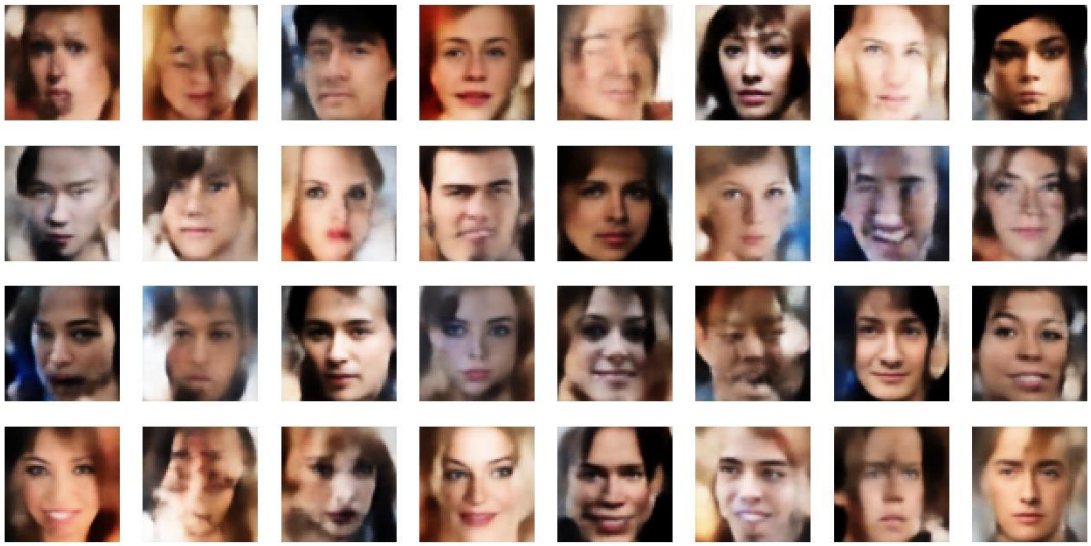
5. (1%) Visualize the latent space by mapping test images to 2D space (with tSNE) and color them with respect to an attribute of your choice.

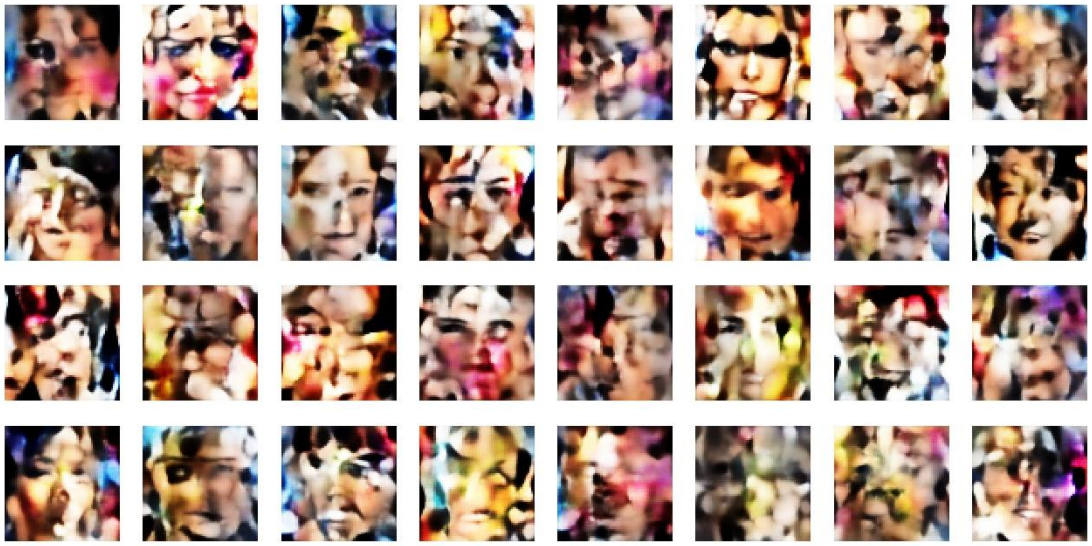



從上圖可以看到，大致上點可以分成左上一類(紅點 Female)，右下一類(藍點 Male)。

6. Discuss what you've observed and learned from implementing VAE.

藉由這次調整$\lambda_{KL}$參數，可以發現此參數在做 VAE 的第四小題占了很重要的份量，若$\lambda_{KL}$設定很小，儘管可以在 MSE 有很好的表現(reconstructed 的部分)，但在 random generated images 的部分卻相當的糟糕。(下圖上側為$\lambda_{KL}$=$10^{-4}$的圖像，下側為$\lambda_{KL}$=$10^{-12}$的圖像)





從 random generated images 出來的結果也可以看到，VAE 做出來的結果比較糊，沒有像照片一樣那麼細緻。且 VAE 產生出來的影像，人臉的皮膚顏色都是光滑平順，跟一般的人臉比較沒有那麼相像。

Problem 2. GAN (5%)

1. (1%) Describe the architecture & implementation details of your model.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | (None, 64, 64, 3) | 0 |
| discriminator_conv1 (Conv2D) | (None, 32, 32, 64) | 4864 |
| leaky_re_lu_1 (LeakyReLU) | (None, 32, 32, 64) | 0 |
| batch_normalization_1 (Batch | (None, 32, 32, 64) | 256 |
| dropout_1 (Dropout) | (None, 32, 32, 64) | 0 |
| discriminator_conv2 (Conv2D) | (None, 16, 16, 64) | 102464 |
| leaky_re_lu_2 (LeakyReLU) | (None, 16, 16, 64) | 0 |
| batch_normalization_2 (Batch | (None, 16, 16, 64) | 256 |
| dropout_2 (Dropout) | (None, 16, 16, 64) | 0 |
| discriminator_conv3 (Conv2D) | (None, 8, 8, 128) | 204928 |
| leaky_re_lu_3 (LeakyReLU) | (None, 8, 8, 128) | 0 |
| batch_normalization_3 (Batch | (None, 8, 8, 128) | 512 |
| dropout_3 (Dropout) | (None, 8, 8, 128) | 0 |
| discriminator_conv4 (Conv2D) | (None, 4, 4, 128) | 409728 |
| leaky_re_lu_4 (LeakyReLU) | (None, 4, 4, 128) | 0 |
| batch_normalization_4 (Batch | (None, 4, 4, 128) | 512 |
| dropout_4 (Dropout) | (None, 4, 4, 128) | 0 |
| flatten_1 (Flatten) | (None, 2048) | 0 |
| discriminator_fc1 (Dense) | (None, 1) | 2049 |

首先是 Discriminator 在這邊我用了四層 convolution layer 去提取影像中的資訊，並在最後接上 full connection layer 去判斷這張圖片是真實的還是假的。 在 Discriminator 裡 optimizer 使用到的是 adam。

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 8192)              1056768
_________________________________________________________________
batch_normalization_5 (Batch (None, 8192)              32768
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 8, 128)         0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 128)       0
_________________________________________________________________
generator_conv1 (Conv2D)     (None, 16, 16, 128)       409728
_________________________________________________________________
batch_normalization_6 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 32, 32, 128)       0
_________________________________________________________________
generator_conv2 (Conv2D)     (None, 32, 32, 64)        204864
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 64, 64, 64)        0
_________________________________________________________________
generator_conv3 (Conv2D)     (None, 64, 64, 64)        102464
_________________________________________________________________
batch_normalization_8 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
generator_conv4 (Conv2D)     (None, 64, 64, 3)         1731
=================================================================
Total params: 1,809,347
Trainable params: 1,792,451
Non-trainable params: 16,896
_________________________________________________________________
```

再來是 generator 的部分，一開始使用到 full connection layer 先把 input 的 128 維向量，做成稍微大一些的向量$(8 \times 8 \times 128)$，再來就使用 upsampling 配合 convolution 去對向量做 deconvolution 的行為，將向量延展成臉部影像大小$64 \times 64 \times 3$。其 generator 同樣用 adam 當作其 optimizer。

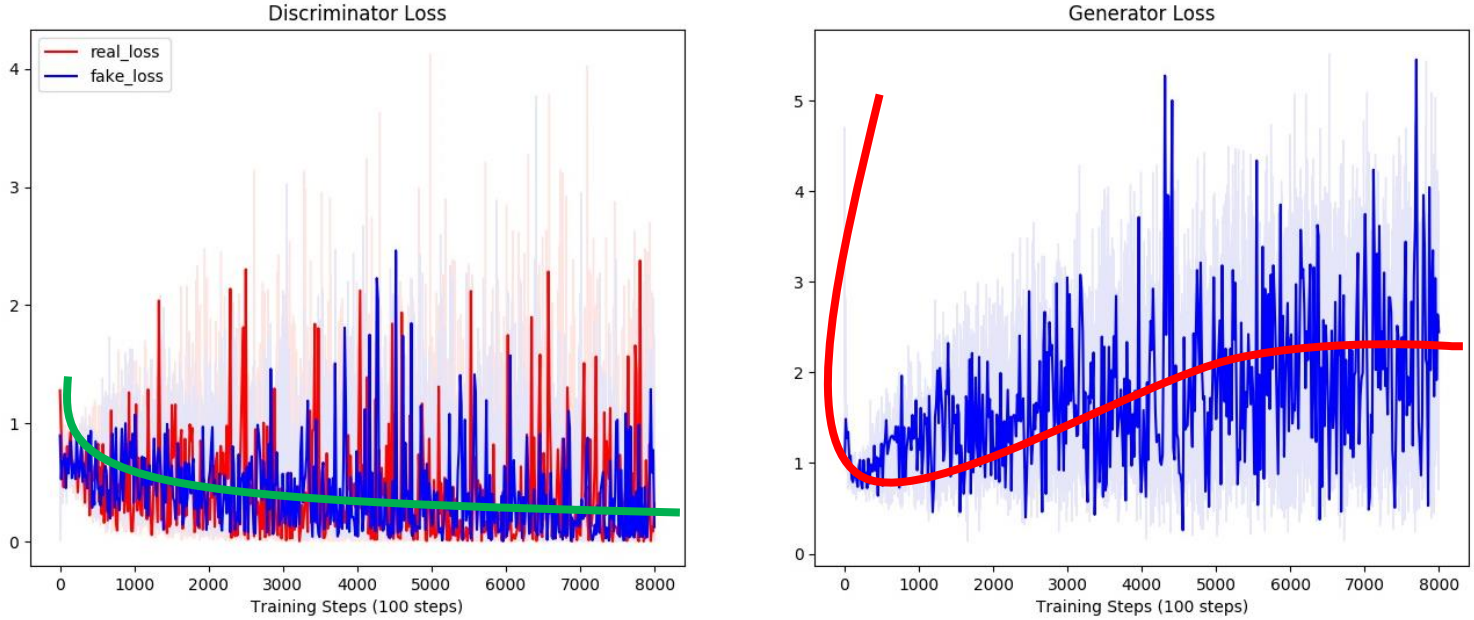
```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 128)               0
_________________________________________________________________
model_2 (Model)              (None, 64, 64, 3)         1809347
_________________________________________________________________
model_1 (Model)              (None, 1)                 725569
=================================================================
Total params: 2,534,916
Trainable params: 1,792,451
Non-trainable params: 742,465
_________________________________________________________________
```

上圖為 GAN 最後的架構，將 generator 接在 discriminator 後面，在做 discriminator 的訓練時，會同時使用 real data 和 fake data 當作它輸入，real data 為真實影像，而 fake data 則是 generator 產生出來的影像，藉由此方式來讓 discriminator 能夠分辨哪些是真實的影像，而

哪些又是由 generator 產生的。而在 generator 訓練的部分，則是會使用到整個 GAN 的系統，由之前訓練過的 discriminator 來判斷 generator 生成影像的好壞，在這裡我們希望 generator 出來的影像能夠欺騙過 discriminator，以此來達到強化 generator 的功能，並且希望藉由 generator 與 discriminator 的對抗，來讓這個 model 有更好的成果。

在訓練時，discriminator 和 generator 是交錯訓練(每個 batch)，以此希望 discriminator 不會太過強大，導致 generator 的結果不好。而 discriminator 和 generator 的 model 基本上是互相對稱，藉由此方式讓兩個 model 可以有較相近的收斂速度，而不會某個 model 過於強大，導致另一個 model 訓練不起來。而在輸入的資料方面，generator 的輸入都是由 random 的形式去產生，而在資料量方面，每一個 batch 中，discriminator 輸入的 real data 數和 fake data 數相加會等於 generator 的輸入量。

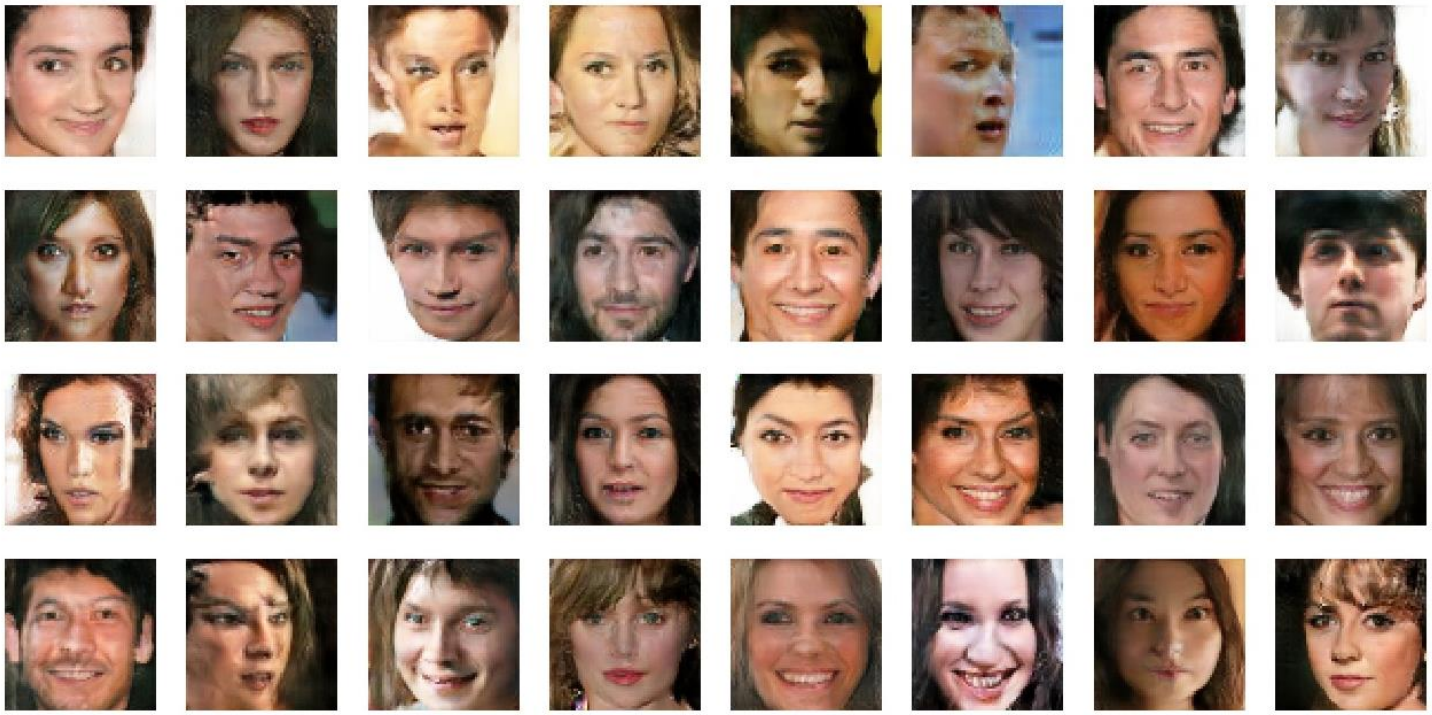
2. (1%) Plot the learning curve (in the way you prefer) of your model and briefly explain what you think it represents.




※ 上圖中較粗的線條(綠、紅)是用來輔助說明用，而非原始影像產生的曲線。

上圖左側為 Discriminator loss (discriminator 訓練時)，紅線和藍線分別對應到 discriminator 對 real data 和 fake data 的 loss。由上圖可以看到兩個 loss 彼此之間有大幅度震盪的趨勢，並且有稍微下降的形況產生，由此可以知道 discriminator 跟 generator 有互相抗衡，而不是單一個 model 強大而已，並且 discriminator 有逐漸在變強。而右側為 Generator loss 是對於 generator 產生出來的影像能欺騙過 discriminator 的評估。

3. (1%) Plot 32 random generated images of your model.



4. (1%) Discuss what you've observed and learned from implementing GAN.

經由這次 GAN 的訓練，我發現訓練 GAN 最重要的一點，並是平衡 discriminator 和 generator，彼此之間的 layer 層數、參數量以及輸入 data 的量都會影響到其結果。起初在建立 GAN model 時，generator 使用 VAE 的 decoder 部分(8 層)，而 discriminator 是使用新寫的架構(5 層)，在做訓練時，明顯發現 discriminator 過於強大，generator 完全跟不上，導致 generator 產生出的影像都是一堆雜訊，看不到人臉的型態，在參數的部分也有相同的形況發生過。而輸入的 data 量能夠平衡好也可以帶來比較好的結果，一開始我讓每個 batch 中的 real data 、 fake data (都是 discriminator 的輸入)和 generator 的輸入量都一樣，而導致 discriminator 比較強大些(discriminator 的輸入量為 generator 的兩倍)。最後將每個 batch 中的 real data 、 fake data 的資料量砍半後(discriminator 的輸入量和 generator 相同)，產生了比較好的結果。

而另一點則是由於 GAN 的 loss 彼此之間是將對應關係，因此基本上無法知道它何時收斂到最好，也無法直接從 loss 就看出來你產生的哪個 generator 是最好的。

5. (1%) Compare the difference between image generated by VAE and GAN, discuss what you've observed.

在 VAE 與 GAN 的比較方面，由上一大題提到的 VAE 產生出來的影像，基本上都是比較模糊的，真實性並沒有很高(只是單純 MSE 較低)，而 GAN 則不同了，在 generator 的結尾接上 discriminator 來判別生成影像的真實性，大幅地幫助 generator 對產出影像的品質，由第三小題的影像可以看得出來，GAN 產生出來的影像更為細緻，臉的細微部分都有跑出來，相較於 VAE 產

出的臉輪廓更加真實。

## Problem 3. ACGAN(4%)

1. (1%) Describe the architecture & implementation details of your model.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 64, 64, 3) | 0 | |
| discriminator_conv1 (Conv2D) | (None, 32, 32, 64) | 4864 | input_1[0][0] |
| leaky_re_lu_1 (LeakyReLU) | (None, 32, 32, 64) | 0 | discriminator_conv1[0][0] |
| batch_normalization_1 (BatchNor | (None, 32, 32, 64) | 256 | leaky_re_lu_1[0][0] |
| dropout_1 (Dropout) | (None, 32, 32, 64) | 0 | batch_normalization_1[0][0] |
| discriminator_conv2 (Conv2D) | (None, 16, 16, 64) | 102464 | dropout_1[0][0] |
| leaky_re_lu_2 (LeakyReLU) | (None, 16, 16, 64) | 0 | discriminator_conv2[0][0] |
| batch_normalization_2 (BatchNor | (None, 16, 16, 64) | 256 | leaky_re_lu_2[0][0] |
| dropout_2 (Dropout) | (None, 16, 16, 64) | 0 | batch_normalization_2[0][0] |
| discriminator_conv3 (Conv2D) | (None, 8, 8, 128) | 204928 | dropout_2[0][0] |
| leaky_re_lu_3 (LeakyReLU) | (None, 8, 8, 128) | 0 | discriminator_conv3[0][0] |
| batch_normalization_3 (BatchNor | (None, 8, 8, 128) | 512 | leaky_re_lu_3[0][0] |
| dropout_3 (Dropout) | (None, 8, 8, 128) | 0 | batch_normalization_3[0][0] |
| discriminator_conv4 (Conv2D) | (None, 4, 4, 128) | 409728 | dropout_3[0][0] |
| leaky_re_lu_4 (LeakyReLU) | (None, 4, 4, 128) | 0 | discriminator_conv4[0][0] |
| batch_normalization_4 (BatchNor | (None, 4, 4, 128) | 512 | leaky_re_lu_4[0][0] |
| dropout_4 (Dropout) | (None, 4, 4, 128) | 0 | batch_normalization_4[0][0] |
| flatten_1 (Flatten) | (None, 2048) | 0 | dropout_4[0][0] |
| discriminator_real (Dense) | (None, 1) | 2049 | flatten_1[0][0] |
| discriminator_label (Dense) | (None, 3) | 6147 | flatten_1[0][0] |

在 Discriminator model 方面，基本的架構跟前面提到的 GAN 相同，最大的不同點在於輸出多了一項，這項主要是用來判斷影像的類別(在這裡是 attribute)，在這部分的 loss 使用的是 categorical_crossentropy，從架構也可以看到最後一層多了一個 full connection layer，它的類別主要有三類(0~2)，0 用來代表沒有該 attribute，1 用來代表有該 attribute，而 2 則是用來作為 background 類別。

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
noise_input (InputLayer)     (None, 128)               0
__________________________________________________________________________________________________
label_input (InputLayer)     (None, 1)                 0
__________________________________________________________________________________________________
model_input (Concatenate)    (None, 129)               0           noise_input[0][0]
                                                                   label_input[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, 8192)              1064960     model_input[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 8192)           32768       dense_1[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)          (None, 8, 8, 128)         0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
up_sampling2d_1 (UpSampling2D)  (None, 16, 16, 128)    0           reshape_1[0][0]
__________________________________________________________________________________________________
generator_conv1 (Conv2D)     (None, 16, 16, 128)       409728      up_sampling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 16, 16, 128)    512         generator_conv1[0][0]
__________________________________________________________________________________________________
up_sampling2d_2 (UpSampling2D)  (None, 32, 32, 128)    0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
generator_conv2 (Conv2D)     (None, 32, 32, 64)        204864      up_sampling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 32, 32, 64)     256         generator_conv2[0][0]
__________________________________________________________________________________________________
up_sampling2d_3 (UpSampling2D)  (None, 64, 64, 64)     0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
generator_conv3 (Conv2D)     (None, 64, 64, 64)        102464      up_sampling2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 64, 64, 64)     256         generator_conv3[0][0]
__________________________________________________________________________________________________
generator_conv4 (Conv2D)     (None, 64, 64, 3)         1731        batch_normalization_8[0][0]
==================================================================================================
Total params: 1,817,539
Trainable params: 1,800,643
Non-trainable params: 16,896
```

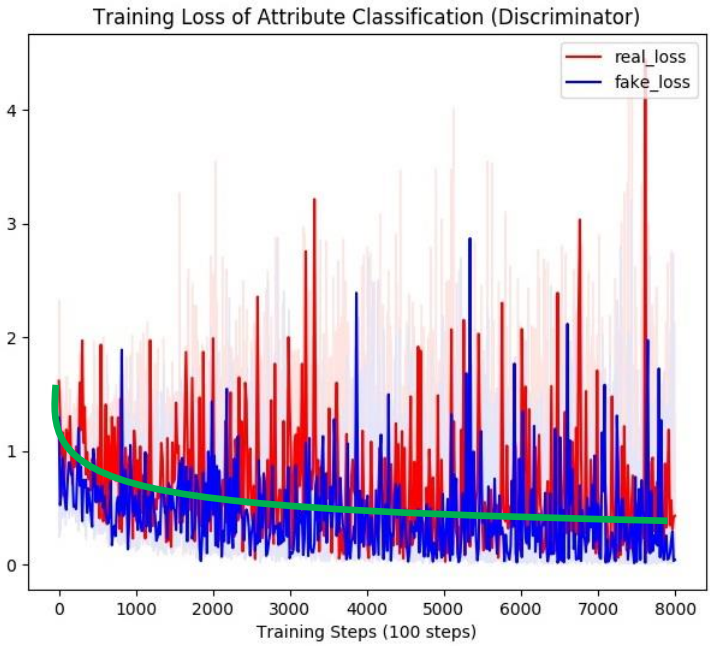
Generator model 方面，一樣如之前的 GAN 相同的架構，不過多了一項 label_input 主要是用來作為產生圖片是否有 attribute 的根據，由於這次只有一個 attribute 且只有分為有或沒有，因此在 input 方面，我們沒有使用到 merge 的方式(multiply)將兩個 input 結合在一起，而是直接把 label_input 接在 noise_input 後面，形成一個 129 維的 input layer。

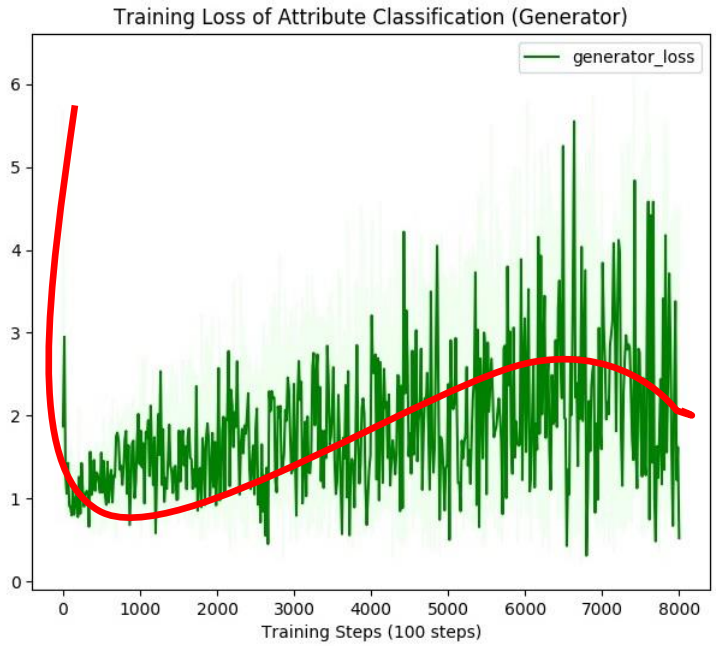
```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_2 (InputLayer)         (None, 128)               0
__________________________________________________________________________________________________
input_3 (InputLayer)         (None, 1)                 0
__________________________________________________________________________________________________
model_2 (Model)              (None, 64, 64, 3)         1817539     input_2[0][0]
                                                                   input_3[0][0]
__________________________________________________________________________________________________
model_1 (Model)              [(None, 1), (None, 3      731716      model_2[1][0]
==================================================================================================
Total params: 2,549,255
Trainable params: 1,800,643
Non-trainable params: 748,612
```

同樣於 GAN，將 discriminator 接在 generator 後面，訓練方式基本上跟一般的 GAN 一模一樣，差別在於這個 model 變為兩個 input 和兩個 output，藉由多了一個 label 的 input 讓我們能使 ACGAN 學到影像中的 attribute，然後如同 GAN 一樣 discriminator 會用來判別生成出影像的真實性，且做 attribute 的 classify，以此來使 ACGAN 生成的影像能夠有展現 attribute 的功能。前面提到 discriminator 的 output 為三維，而非二維，在這邊 background 的類別用在於訓

練的時候，在訓練 discriminator 時，我希望它除了能將 generator 生成的影像辨識成 fake 之外，也想使其 attribute 的辨識為 background，因此 output 才多了這一個維度。

2. (1%) Plot the learning curve (in the way you prefer) of your model and briefly explain what you think it represents.




※ 上圖中較粗的線條(綠、紅)是用來輔助說明用，而非原始影像產生的曲線。

在 GAN 的部分中，討論過 discriminator 對其真實性評估的 loss，因此這邊以 attribute classification 的 loss 做討論，左圖為 discriminator (discriminator 訓練時) 的 attribute loss，同樣有震盪的情況，不過一樣有下降的情形。而右圖為 generator (generator 訓練時) 的 attribute loss 的圖示，前幾個 steps 可以看到都處於高點，主要是因為在前期 discriminator 和 generator 都並沒有學到什麼東西，導致 loss 是亂跳的行為，不過到了中期可以看到曲線的趨勢是逐漸趨於平穩的，雖然有很多震盪的地方(generator 和 discriminator 互相競爭導致)，不過在最後的部分有開始準備下降的趨勢。

3. (2%) Plot 10 pair of random generated images of your model, each pair generated from the same random vector input but with different attribute. This is to demonstrate your model's ability to disentangle feature of interest.

此題是以有沒有笑最為 attribute 的參數，上排為沒有笑的臉，而下排為有笑的臉，可以清楚得看到 ACGAN 有學到此 attribute，並且成功的產生出對應 attribute 的圖片，且因為在產生 input vector 時，是直接將 attribute 接在 noise vector 後面，因此可以看到上排和下排的臉基本上都是相同的，最大的差別只在於 attribute 所導致的部分。(有沒有笑)

參考資料：

訓練過程、架構：https://github.com/eriklindernoren/Keras-GAN