討論對象：洪建豪、林恩妤、簡上淵　　電信碩一　　R06942143　藍聖皓

Problem1. VAE(參考 github：https://github.com/keras-team/keras/blob/master/examples/variational_autoencoder_deconv.py (5/10 之前的那個版本，因為他更新了，所以點進去會看到不一樣的 code))

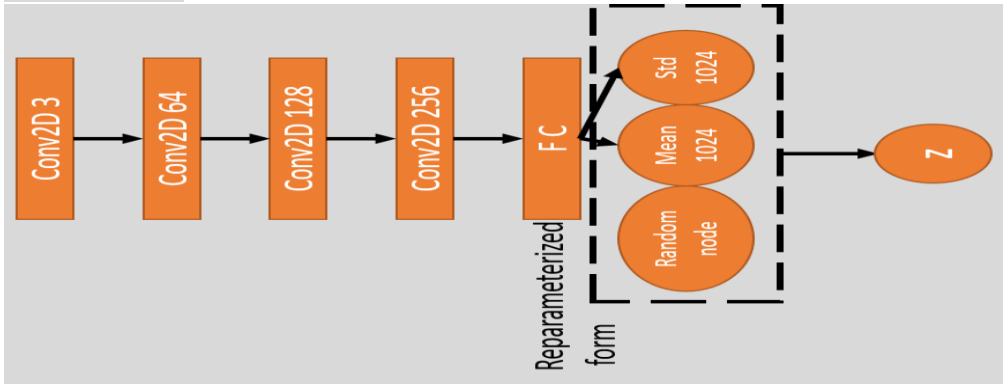
1. Describe the architecture & implementation details of yours model (1%)

這邊使用 keras 來 implement model，總體架構如下：

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 64, 64, 3)	0	
conv2d_1 (Conv2D)	(None, 32, 32, 3)	147	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 32, 32, 3)	12	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 3)	0	batch_normalization_1[0][0]
dropout_1 (Dropout)	(None, 32, 32, 3)	0	leaky_re_lu_1[0][0]
conv2d_2 (Conv2D)	(None, 16, 16, 64)	3136	dropout_1[0][0]
batch_normalization_2 (BatchNor	(None, 16, 16, 64)	256	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 64)	0	batch_normalization_2[0][0]
dropout_2 (Dropout)	(None, 16, 16, 64)	0	leaky_re_lu_2[0][0]
conv2d_3 (Conv2D)	(None, 8, 8, 128)	131200	dropout_2[0][0]
batch_normalization_3 (BatchNor	(None, 8, 8, 128)	512	conv2d_3[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 128)	0	batch_normalization_3[0][0]
dropout_3 (Dropout)	(None, 8, 8, 128)	0	leaky_re_lu_3[0][0]
conv2d_4 (Conv2D)	(None, 4, 4, 256)	524544	dropout_3[0][0]
batch_normalization_4 (BatchNor	(None, 4, 4, 256)	1024	conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 4, 4, 256)	0	batch_normalization_4[0][0]
dropout_4 (Dropout)	(None, 4, 4, 256)	0	leaky_re_lu_4[0][0]
flatten_1 (Flatten)	(None, 4096)	0	dropout_4[0][0]
dense_1 (Dense)	(None, 1024)	4195328	flatten_1[0][0]
batch_normalization_5 (BatchNor	(None, 1024)	4096	dense_1[0][0]
activation_1 (Activation)	(None, 1024)	0	batch_normalization_5[0][0]
dense_2 (Dense)	(None, 1024)	1049600	activation_1[0][0]
dense_3 (Dense)	(None, 1024)	1049600	activation_1[0][0]
lambda_1 (Lambda)	(None, 1024)	0	dense_2[0][0] dense_3[0][0]
dense_4 (Dense)	(None, 1024)	1049600	lambda_1[0][0]
batch_normalization_6 (BatchNor	(None, 1024)	4096	dense_4[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 1024)	0	batch_normalization_6[0][0]
dense_5 (Dense)	(None, 4096)	4198400	leaky_re_lu_5[0][0]
batch_normalization_7 (BatchNor	(None, 4096)	16384	dense_5[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 4096)	0	batch_normalization_7[0][0]
reshape_1 (Reshape)	(None, 4, 4, 256)	0	leaky_re_lu_6[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 8, 8, 256)	0	reshape_1[0][0]
conv2d_5 (Conv2D)	(None, 8, 8, 256)	1048832	up_sampling2d_1[0][0]
batch_normalization_8 (BatchNor	(None, 8, 8, 256)	1024	conv2d_5[0][0]
leaky_re_lu_7 (LeakyReLU)	(None, 8, 8, 256)	0	batch_normalization_8[0][0]
dropout_5 (Dropout)	(None, 8, 8, 256)	0	leaky_re_lu_7[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 16, 16, 256)	0	dropout_5[0][0]
conv2d_6 (Conv2D)	(None, 16, 16, 128)	524416	up_sampling2d_2[0][0]
batch_normalization_9 (BatchNor	(None, 16, 16, 128)	512	conv2d_6[0][0]
leaky_re_lu_8 (LeakyReLU)	(None, 16, 16, 128)	0	batch_normalization_9[0][0]
dropout_6 (Dropout)	(None, 16, 16, 128)	0	leaky_re_lu_8[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 32, 32, 128)	0	dropout_6[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 64)	131136	up_sampling2d_3[0][0]
batch_normalization_10 (BatchNo	(None, 32, 32, 64)	256	conv2d_7[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 32, 32, 64)	0	batch_normalization_10[0][0]
dropout_7 (Dropout)	(None, 32, 32, 64)	0	leaky_re_lu_9[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 64, 64, 64)	0	dropout_7[0][0]
conv2d_8 (Conv2D)	(None, 64, 64, 3)	3075	up_sampling2d_4[0][0]
activation_2 (Activation)	(None, 64, 64, 3)	0	conv2d_8[0][0]

Total params: 13,937,186
Trainable params: 13,923,100
Non-trainable params: 14,086

Encoder 部分



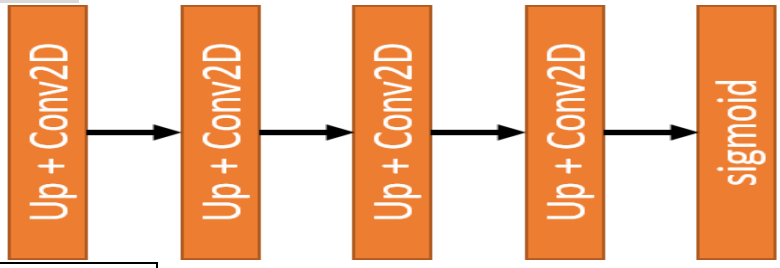
Model architecture：

這邊接了四層的 conv2d，而 filter 數量各別為(3,64,128,256)，中間都有加上 Dropout、Batchnormalize 以及 Leakyrelu。最後 flatten 成 4096 維，然後 fully connected 之後 sample 出 z。

Implementation details:

而原本嘗試了使用 selu 來替換 leakyrelu，但是效果並沒有變好，然後還沒辦法 implement 出來，因此最後還是使用 leakyrelu。也有嘗試過多疊一層 512，不過最後 reconstruct 的圖形會解析度變得很差，因為沒有做 gradient ascent，所以不大清楚原因，不過這邊猜測應該是對於一張 64*64*3 的圖，太多層 conv2d layer 反而會讓其喪失解析能力。

Decoder 部分



Model architecture：

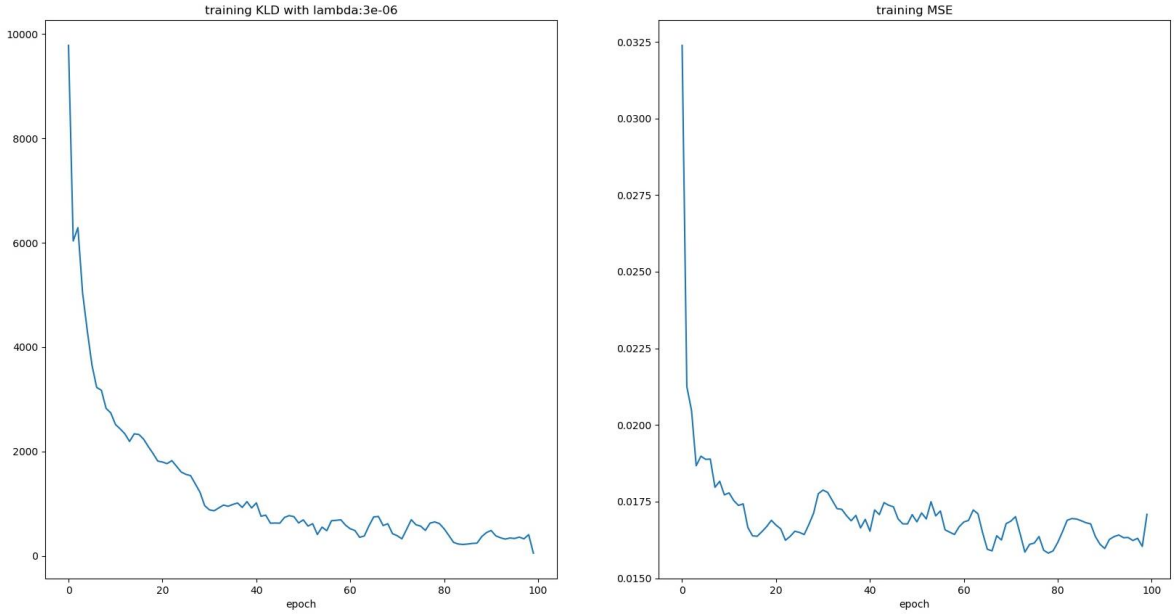
用了 upsampling 以及 conv2d layer 組成的四個 block 把圖片 decoder 回 64*64*3，最後接了一個 sigmoid 把 output 數值壓回 0~1 之間。

Implementation details:

原本是採用 Conv2dtransposed，不過後來採用了 upsampling + conv2d 的結合，因為比較了 upsampling+conv2d 與 conv2dtransposed 的結果之後，覺得 conv2dtranspose 的結果沒有比較好，於是研究了一下其中的差異，參考了這個 github 的內容 https://github.com/vdumoulin/conv_arithmetic，upsample2D 之後再 conv2d，將本來的圖變成兩倍，這樣直覺上有比較多的資訊有被抓到，不過可能圖片會變得比較模糊一些。

而最後的 Activation Function 則改成 sigmoid，因為這邊想說，如果中間的 Activation Function 都使用了 leakyrelu，那其實進來的數值應該大多都是正的數字，若是以 tanh 來轉換的話，負的值本來就會小許多，這樣可能轉回來的時候會有些差異，因此最後採用 sigmoid。

2. Plot the learning curve (reconstruction loss & KL divergence) of your model [fig1_2.jpg] (1%)

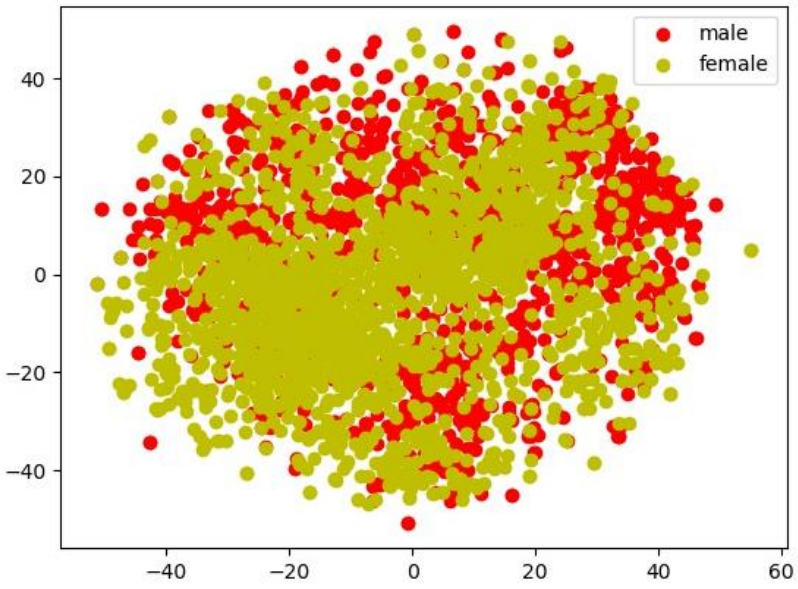


這邊 KLD 不是抓 mean 的數值，是抓 sum 出來的結果，所以數值比其他同學的大，但最後出來的結果還是可以 Random 產生圖片以及 reconstruct。

3. Plot 10 testing images and their reconstructed result of your model [fig1_3.jpg] and report your testing MSE of the entire test set (1%)
   MSE of test set： 0.0167

4. Plot 32 Random generated images of yours model [fig1_4.jpg] (1%)

5. Visualize the latent space by mapping test images to 2D space (with tSNE) and color them with respect to an attribute of your choice [fig1_5.jpg] (1%)



6. Discuss what you've observed and learned from implementing VAE (1%)

這邊在 train VAE 的時候遇到了很多困難，在上面有描述到許多觀察到的地方，比如說 upsampling2d+conv2d vs conv2dtranspose，雖然 upsamplin2d+conv2d 就原理來說，應該是會比較模糊的，但有比較好的結果，這邊猜測可能是 keras 處理 conv2dtranspos 內部插值的方式對於這張圖片來說沒有比 upsampling 好。

Lambda 的數值調整對於，會對於 resonstruct 與 random generator 造成很大的影響，調低 lambda 的數值可以讓 reconstruct 的結果很好，不過卻會喪失了 generator 出來的結果的，結果出來會比較 fit，看起來像是同一張圖片 。

Problem2. GAN

參考文章(知乎：https://zhuanlan.zhihu.com/p/35574753)

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

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 100)	0
dense_1 (Dense)	(None, 16384)	1654784
reshape_1 (Reshape)	(None, 4, 4, 1024)	0
batch_normalization_1 (Batch	(None, 4, 4, 1024)	4096
up_sampling2d_1 (UpSampling2	(None, 8, 8, 1024)	0
conv2d_1 (Conv2D)	(None, 8, 8, 512)	13107712
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 512)	0
batch_normalization_2 (Batch	(None, 8, 8, 512)	2048
up_sampling2d_2 (UpSampling2	(None, 16, 16, 512)	0
conv2d_2 (Conv2D)	(None, 16, 16, 256)	3277056
leaky_re_lu_3 (LeakyReLU)	(None, 16, 16, 256)	0
batch_normalization_3 (Batch	(None, 16, 16, 256)	1024
up_sampling2d_3 (UpSampling2	(None, 32, 32, 256)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	819328
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 128)	0
batch_normalization_4 (Batch	(None, 32, 32, 128)	512
up_sampling2d_4 (UpSampling2	(None, 64, 64, 128)	0
conv2d_4 (Conv2D)	(None, 64, 64, 3)	9603

Total params: 18,876,163
Trainable params: 18,872,323
Non-trainable params: 3,840

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 64, 64, 3)	0
conv2d_5 (Conv2D)	(None, 32, 32, 128)	3584
leaky_re_lu_5 (LeakyReLU)	(None, 32, 32, 128)	0
batch_normalization_5 (Batch	(None, 32, 32, 128)	512
conv2d_6 (Conv2D)	(None, 16, 16, 128)	147584
leaky_re_lu_6 (LeakyReLU)	(None, 16, 16, 128)	0
batch_normalization_6 (Batch	(None, 16, 16, 128)	512
conv2d_7 (Conv2D)	(None, 8, 8, 256)	295168
leaky_re_lu_7 (LeakyReLU)	(None, 8, 8, 256)	0
batch_normalization_7 (Batch	(None, 8, 8, 256)	1024
conv2d_8 (Conv2D)	(None, 4, 4, 512)	1180160
leaky_re_lu_8 (LeakyReLU)	(None, 4, 4, 512)	0
batch_normalization_8 (Batch	(None, 4, 4, 512)	2048
conv2d_9 (Conv2D)	(None, 2, 2, 1024)	4719616
leaky_re_lu_9 (LeakyReLU)	(None, 2, 2, 1024)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_2 (Dense)	(None, 1)	4097

Total params: 6,354,305
Trainable params: 6,352,257
Non-trainable params: 2,048

Generator

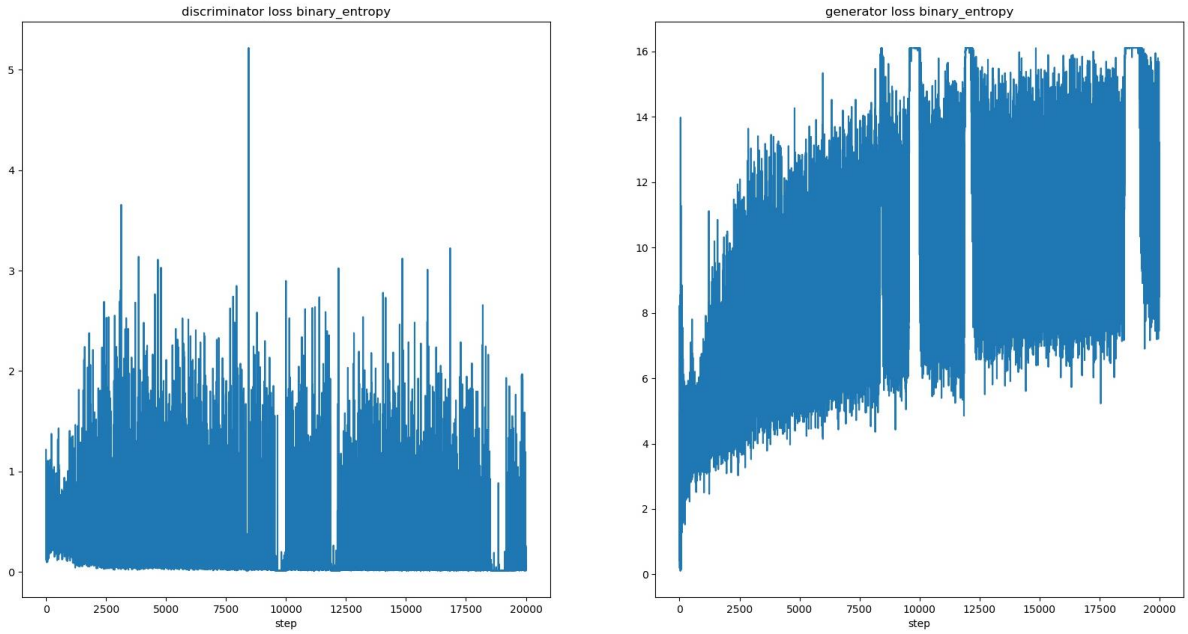
給一個 100 維的 noise，把 VAE 的 decoder 放在這邊，然後在 tune 參數，但架構與上面的 decoder 一樣是四層的 upsampling2D+conv2d。

Discriminator

是將原本 VAE encoder 的部分拿過來，不過這邊拿了五層的 convolution(128、128、256、512、1024)，當作 discriminator 最後面 fully connected 之後接到一個 sigmoid 來判斷是否為真，而在這邊有使用助教連結中的 tips，label 的方式不是只是 0 和 1，而是

有多給一個 noise(np.abs(np.random.normal(0,0.02)))隨著時間下降，讓 discriminator 先不要那麼強，先給 generator 一點時間成長。

2. Plot the learning curve (in the way you prefer) of your model and briefly explain what you think it represents [fig2_2] (1%)



Discriminator 的 loss 在一直震盪，代表他有被騙過，然後變強變得準確之後，在被騙過，不過後面的數值有比前面小一些，代表 discriminator 後面其實不是那麼容易被騙的，而 generator 的 loss 對應到實際 plt 出來的圖片，可以看出來其實這個 loss 上升，圖片效果會比較好。不過還是透過觀察 50 個 step 之後生成出的圖片，比較可以 monitor 這個 model 的好壞。

3. Plot 32 random generated images of your model [fig2_3] (1%)

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

optimizer

這邊嘗試了 tip 的內容，discriminator 使用 sgd(因為是隨機抽取來算 gradient 所以應該比較難收斂，是為了讓 discriminator 變強較慢一些)，generator 使用 adam，但是效

果好似沒有比較好(沒辦法生成圖片)，最後還是使用了 RMSprop，原因推測可能是因為 RMSprop 有個比較適合處理非平穩的目標的特性。

dropout

這邊也加入了 dropout，但是卻發現加入了之後反倒是沒辦法 train 起來了(跑了 4000 個 step 之後還是雜訊)，從這點推知到，應該是因為 model 其實要應付這個 64*64*3 的圖片 這個的 node 已經很吃緊了，所以可以在讓 model 複雜一點，再加入 dropout 可能可以加強最後的結果。

Loss

從 loss 的曲線之中可以看的出來，discriminator 是不是太強了，可以由此判斷是不是要讓 discriminator 變強慢一些。

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

VAE 所產生的照片大多都比較模糊，圖片是比較模糊平滑的樣子，不過反觀 GAN 所生成出來的圖片 ，解析度比較 VAE 起來是比較高的，同樣都是拿了一個 normal distribution 的 noise，但是 GAN 的結果卻好很多。

VAE

產生的結果少雜訊(大部分人臉的位置也都是對的)比較模糊，不過比較好調整參數與圖片的關係。

GAN

結果比較清晰(不過雜訊影響得比較大)，但是很難調整好參數讓這個網路更強。

Problem3. ACGAN

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

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 64, 64, 3)	0
conv2d_5 (Conv2D)	(None, 32, 32, 128)	3584
batch_normalization_5 (Batch	(None, 32, 32, 128)	512
leaky_re_lu_5 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_6 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_6 (Batch	(None, 16, 16, 128)	512
leaky_re_lu_6 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_7 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_7 (Batch	(None, 8, 8, 256)	1024
leaky_re_lu_7 (LeakyReLU)	(None, 8, 8, 256)	0
conv2d_8 (Conv2D)	(None, 4, 4, 512)	1180160
batch_normalization_8 (Batch	(None, 4, 4, 512)	2048
leaky_re_lu_8 (LeakyReLU)	(None, 4, 4, 512)	0
conv2d_9 (Conv2D)	(None, 2, 2, 1024)	4719616
batch_normalization_9 (Batch	(None, 2, 2, 1024)	4096
leaky_re_lu_9 (LeakyReLU)	(None, 2, 2, 1024)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_2 (Dense)	(None, 1)	4097

Total params: 6,358,401
Trainable params: 6,354,305
Non-trainable params: 4,096

discriminator

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 100)	0	
input_2 (InputLayer)	(None, 1)	0	
concatenate_1 (Concatenate)	(None, 101)	0	input_1[0][0] input_2[0][0]
dense_1 (Dense)	(None, 65536)	6684672	concatenate_1[0][0]
reshape_1 (Reshape)	(None, 8, 8, 1024)	0	dense_1[0][0]
batch_normalization_1 (BatchNor	(None, 8, 8, 1024)	4096	reshape_1[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 16, 16, 1024)	0	batch_normalization_1[0][0]
conv2d_1 (Conv2D)	(None, 16, 16, 512)	13107712	up_sampling2d_1[0][0]
batch_normalization_2 (BatchNor	(None, 16, 16, 512)	2048	conv2d_1[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 512)	0	batch_normalization_2[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 32, 32, 512)	0	leaky_re_lu_2[0][0]
conv2d_2 (Conv2D)	(None, 32, 32, 256)	3277056	up_sampling2d_2[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 32, 32, 256)	0	conv2d_2[0][0]
batch_normalization_3 (BatchNor	(None, 32, 32, 256)	1024	leaky_re_lu_3[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 64, 64, 256)	0	batch_normalization_3[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 128)	819328	up_sampling2d_3[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 64, 64, 128)	0	conv2d_3[0][0]
batch_normalization_4 (BatchNor	(None, 64, 64, 128)	512	leaky_re_lu_4[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 3)	387	batch_normalization_4[0][0]

Total params: 23,896,835
Trainable params: 23,892,995
Non-trainable params: 3,840

generator

從 DCGAN 到 ADGAN 就比較簡單了，在 train 的時候把 label 的資料 concatenate2 到 generator 的 model 裡面，而 discriminator 這邊再多加入一個 sigmoid，來判斷是不是有那個特徵(笑臉)。

Generator

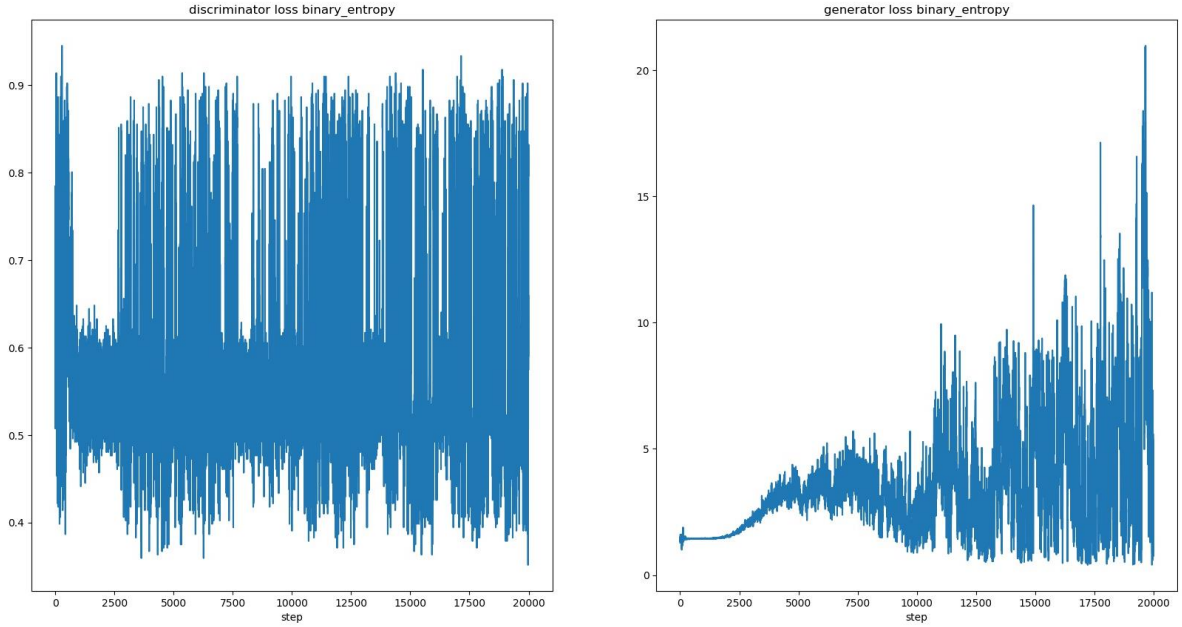
延續 DCGAN 之中的 generator 架構，只是在最開始進入 Generator 的時候多給了一個 label。

Discriminator

也是使用 DCGAN 中的 discriminator 架構，不過最後輸出加上一個 sigmoid 判斷屬於哪個 label。

2. Plot the learning curve (in the way you prefer) of your model and briefly explain what you think it represents [fig3_2] (1%)

這邊可以大概觀察到 discriminator 和 generator 的 loss 都有一個週期性，互相慢慢變強，不過到後面可能 discriminator 變得太強了，所以 generator 震盪變太大了，代表不太容易騙過 discriminator，因此可能可以在後面讓 discriminator 的 weight 改成前面幾個 step 的去跑，可能會得到好一些的結果。



3. 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. [fig3_3] (2%)