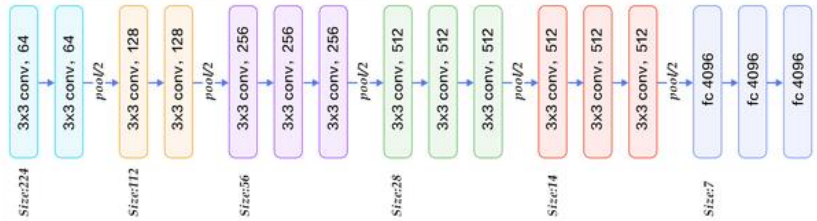
學號：R06942143 系級： 電信一 姓名：籃聖皓

1. (1%) 請說明你實作的 CNN model，其模型架構、訓練參數和準確率為何？
(Collaborators:)

答：說明模型架構、訓練參數和準確率。



https://www.quora.com/What-is-the-VGG-neural-network

這邊使用三個模型

a. VGG16 with relu:

Data normalize 和 augmentation 之後接上 VGG16。參數數量：20,941,735，準確率：0.66982

b. VGG16 with selu:

Data normalize 和 augmentation 之後接上 VGG16，這邊不一樣的是 activation 使用 selu。參數數量：20,941,735，準確率：0.68180

不過這邊跟原本設計的不一樣的就是，都把 relu 改成 selu，因為 relu 會有神經元死亡的問題，造成沒辦法 activation，因此最後採取這個方式。

c. 以 VGG19 implement:

Data normalize 和 augmentation 之後接上 VGG16，這邊不一樣的是 activation 使用 selu。參數數量： 43,070,631，準確率：0.69295

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
activation_1 (Activation)	(None, 48, 48, 64)	0
conv2d_2 (Conv2D)	(None, 48, 48, 64)	36928
batch_normalization_2 (Batch	(None, 48, 48, 64)	256
activation_2 (Activation)	(None, 48, 48, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
activation_3 (Activation)	(None, 24, 24, 128)	0
conv2d_4 (Conv2D)	(None, 24, 24, 128)	147584
batch_normalization_4 (Batch	(None, 24, 24, 128)	512
activation_4 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
conv2d_5 (Conv2D)	(None, 12, 12, 256)	295168
batch_normalization_5 (Batch	(None, 12, 12, 256)	1024
activation_5 (Activation)	(None, 12, 12, 256)	0
conv2d_6 (Conv2D)	(None, 12, 12, 256)	590080
batch_normalization_6 (Batch	(None, 12, 12, 256)	1024
activation_6 (Activation)	(None, 12, 12, 256)	0
conv2d_7 (Conv2D)	(None, 12, 12, 256)	590080
batch_normalization_7 (Batch	(None, 12, 12, 256)	1024
activation_7 (Activation)	(None, 12, 12, 256)	0
conv2d_8 (Conv2D)	(None, 12, 12, 256)	590080
batch_normalization_8 (Batch	(None, 12, 12, 256)	1024
activation_8 (Activation)	(None, 12, 12, 256)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 256)	0
dropout_3 (Dropout)	(None, 6, 6, 256)	0
conv2d_9 (Conv2D)	(None, 6, 6, 512)	1180160
batch_normalization_9 (Batch	(None, 6, 6, 512)	2048
activation_9 (Activation)	(None, 6, 6, 512)	0
conv2d_10 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_10 (Batc	(None, 6, 6, 512)	2048
activation_10 (Activation)	(None, 6, 6, 512)	0
conv2d_11 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_11 (Batc	(None, 6, 6, 512)	2048
activation_11 (Activation)	(None, 6, 6, 512)	0
conv2d_12 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_12 (Batc	(None, 6, 6, 512)	2048
activation_12 (Activation)	(None, 6, 6, 512)	0
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
dropout_4 (Dropout)	(None, 3, 3, 512)	0
conv2d_13 (Conv2D)	(None, 3, 3, 512)	2359808
batch_normalization_13 (Batc	(None, 3, 3, 512)	2048
activation_13 (Activation)	(None, 3, 3, 512)	0
conv2d_14 (Conv2D)	(None, 3, 3, 512)	2359808
batch_normalization_14 (Batc	(None, 3, 3, 512)	2048
activation_14 (Activation)	(None, 3, 3, 512)	0
conv2d_15 (Conv2D)	(None, 3, 3, 512)	2359808
batch_normalization_15 (Batc	(None, 3, 3, 512)	2048
activation_15 (Activation)	(None, 3, 3, 512)	0
conv2d_16 (Conv2D)	(None, 3, 3, 512)	2359808
batch_normalization_16 (Batc	(None, 3, 3, 512)	2048
activation_16 (Activation)	(None, 3, 3, 512)	0
max_pooling2d_5 (MaxPooling2	(None, 1, 1, 512)	0
dropout_5 (Dropout)	(None, 1, 1, 512)	0
flatten_1 (Flatten)	(None, 512)	0
batch_normalization_17 (Batc	(None, 512)	2048
dropout_6 (Dropout)	(None, 512)	0
activation_17 (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 4096)	2101248
batch_normalization_18 (Batc	(None, 4096)	16384
dropout_7 (Dropout)	(None, 4096)	0
activation_18 (Activation)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
batch_normalization_19 (Batc	(None, 4096)	16384
dropout_8 (Dropout)	(None, 4096)	0
activation_19 (Activation)	(None, 4096)	0
dense_3 (Dense)	(None, 1000)	4097000
batch_normalization_20 (Batc	(None, 1000)	4000
activation_20 (Activation)	(None, 1000)	0
dense_4 (Dense)	(None, 7)	7007

架構如上圖：

2. (1%) 請嘗試 data normalization, data augmentation,說明實行方法並且說明對準確率有什麼樣的影響？

答：Data normalization

在 normalization(scaling 0 ~1)之前，會一直卡在 0.25 左右，不論 epoch 幾次都沒有辦法提高準確率，去查了一下 google，這好像叫做 **Dead Neurons**，過於大的 gradient 的數值經過這個神經元，讓他沒有辦法再被 activate，這邊推測兩個原因如下：

1. learning rate：
   這個的數值設計太大了，因此讓神經元都無法 activated，解決方法如下：
   改使用別的 activation function 或是調整 learning rate。
2. input data：
   沒有 normalization，再算 optimization 時的值太大，也造成神經元不能 activate，因此 normalize 到 0~1，並且加上 BatchNormalization。

	使用前	使用後
Validation score	0.2547	0.623
Public score	Validation score 太低了就沒上傳了	0.60713
Private score	Validation score 太低了就沒上傳了	

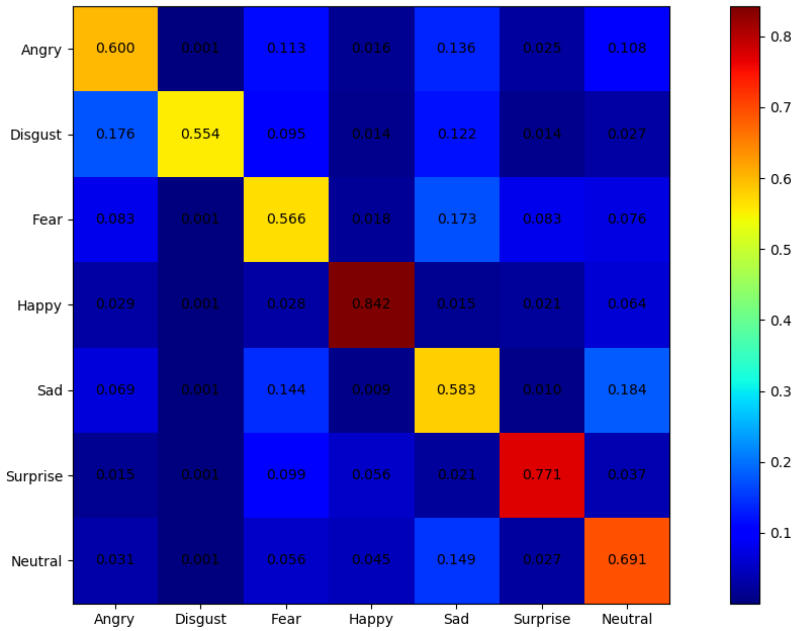
data augmentation(下面的數據是已經做過 data normalize)

是使用 keras 的 imagepreprocessing，把原圖做 rotation、平移、放大，這樣原本的 dataset 就會多了些變化，有了更多的資料量也可以讓 model 變得更準確。

	使用前	使用後
Validation score	0.623	0.67335
Public score	0.60713	0.67818
Private score		

3. (1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析]

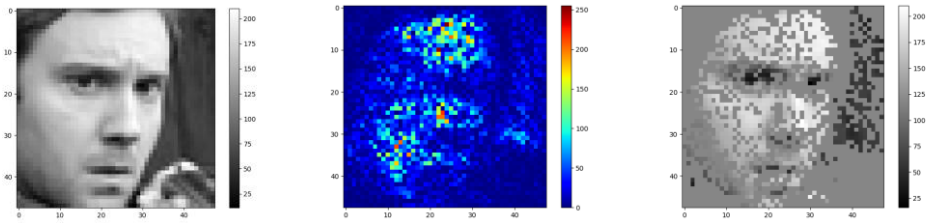
答：貼出 confusion matrix -> 1 分



看藍色較為明亮的可以知道，這些是容易混淆的分類，而七種情緒最容易被誤認為是：

情緒	Angry	Disgust	Fear	Happy	Sad	Surprise	neutral
誤判成	sad	angry	sad	neutral	neutral	fear	sad

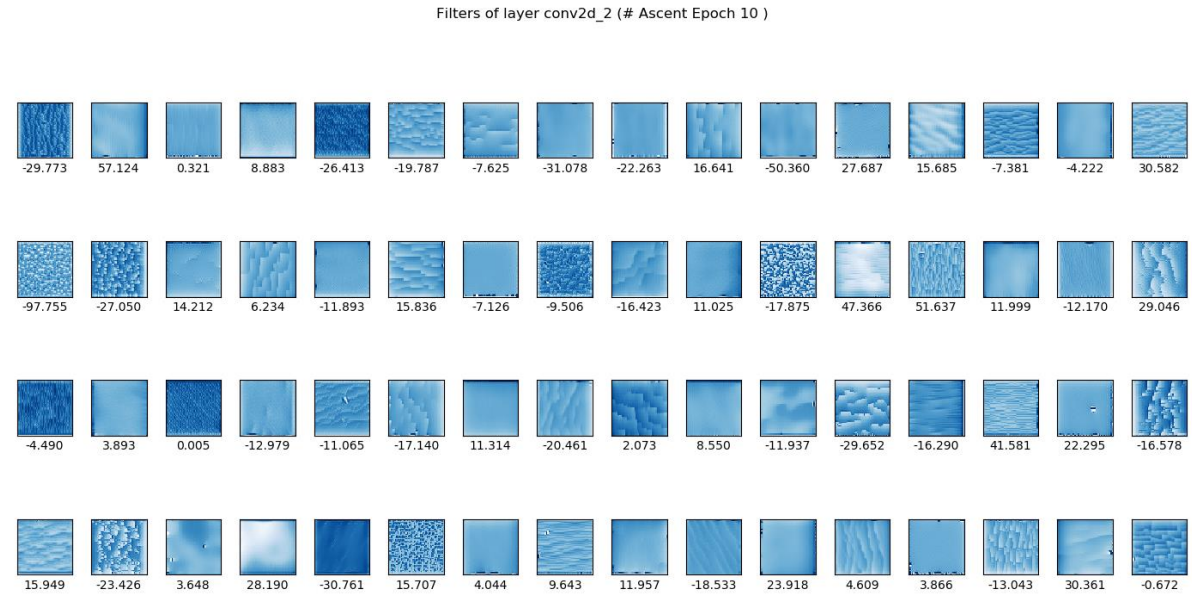
4. (1%) 從(1)(2)可以發現，使用 CNN 的確有些好處，試繪出其 saliency maps，觀察模型在做 classification 時，是 focus 在圖片的哪些部份？
答：合理說明 test 的圖片和觀察到的東西 -> 0.5 分
貼出 saliency 圖片 -> 0.5 分



可以看出 model 用來判別的地方：
像是頭髮、皮膚等等地方，被 model 採用的程度就不是那麼的高，而眼睛、鼻子、嘴巴、臉頰附近的皮膚，就是 model 比較 focus 的地方，這也與我們人類分辨情緒是觀察，眼睛鼻子嘴巴一樣的道理。

5. (1%) 承(1)(2)，利用上課所提到的 gradient ascent 方法，觀察特定層的 filter 最容易被哪種圖片 activate。
答：合理說明 test 的層數和觀察到的東西 -> 0.5 分
貼出 filter input and output 的圖片 -> 0.5 分

(1) 這是還沒 batch_normalization 的 conv2d_2，輸入白雜訊之後得到的結果，可以看出來有些的 ascent 數值較低，像是最後一層-0.672，與其他的數值比起來是較低的，比較不容易被 activate。
而第 29 個 filter 有 51.637 的數值，在 batch_normalization 之後放到 selu，應該是相較之下比較容易被 activate 的一層 filter



(2) 因為運算量太大了，所以只取了(convolution2d_15)512 個 filter 其中的 16filter 做呈現，可以看出其中影像較大的 filter，像是  184.007 就可以看出臉部的紋理，而背景以及頭髮部分的 pixel 也會比較不明顯