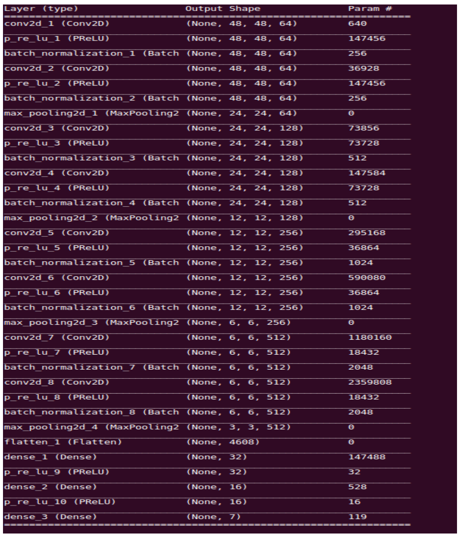
Homework 3 Report - Image Sentiment Classification

學號：R06921002 系級：電機所碩一 姓名：張哲誠

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？

模型架構一：



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
p_re_lu_1 (PReLU)	(None, 48, 48, 64)	147456
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
conv2d_2 (Conv2D)	(None, 48, 48, 64)	36928
p_re_lu_2 (PReLU)	(None, 48, 48, 64)	147456
batch_normalization_2 (Batch	(None, 48, 48, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
p_re_lu_3 (PReLU)	(None, 24, 24, 128)	73728
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
conv2d_4 (Conv2D)	(None, 24, 24, 128)	147584
p_re_lu_4 (PReLU)	(None, 24, 24, 128)	73728
batch_normalization_4 (Batch	(None, 24, 24, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
conv2d_5 (Conv2D)	(None, 12, 12, 256)	295168
p_re_lu_5 (PReLU)	(None, 12, 12, 256)	36864
batch_normalization_5 (Batch	(None, 12, 12, 256)	1024
conv2d_6 (Conv2D)	(None, 12, 12, 256)	590080
p_re_lu_6 (PReLU)	(None, 12, 12, 256)	36864
batch_normalization_6 (Batch	(None, 12, 12, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 256)	0
conv2d_7 (Conv2D)	(None, 6, 6, 512)	1180160
p_re_lu_7 (PReLU)	(None, 6, 6, 512)	18432
batch_normalization_7 (Batch	(None, 6, 6, 512)	2048
conv2d_8 (Conv2D)	(None, 6, 6, 512)	2359808
p_re_lu_8 (PReLU)	(None, 6, 6, 512)	18432
batch_normalization_8 (Batch	(None, 6, 6, 512)	2048
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 32)	147488
p_re_lu_9 (PReLU)	(None, 32)	32
dense_2 (Dense)	(None, 16)	528
p_re_lu_10 (PReLU)	(None, 16)	16
dense_3 (Dense)	(None, 7)	119

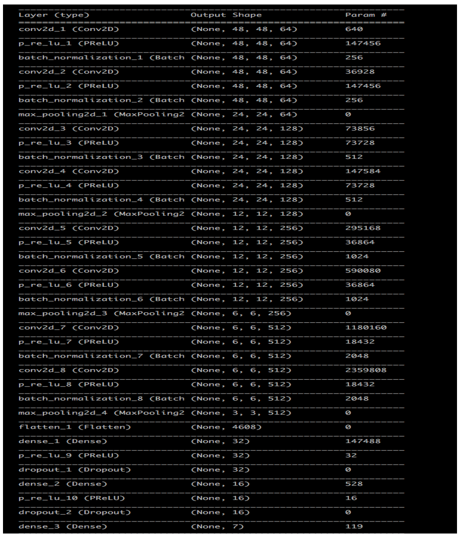
fig. 1 模型架構一

參數一欄表	
Total params	5393047
Trainable params	5389207
Non-trainable params	3840
Batch_size	100
epoch	80

準確率	
Training_val_accu	0.7061
Kaggle_accu	0.67957

Model 檔名：weighrs.best_67957.hdf5

模型架構二：



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
p_re_lu_1 (PReLU)	(None, 48, 48, 64)	147456
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
conv2d_2 (Conv2D)	(None, 48, 48, 64)	36928
p_re_lu_2 (PReLU)	(None, 48, 48, 64)	147456
batch_normalization_2 (Batch	(None, 48, 48, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
p_re_lu_3 (PReLU)	(None, 24, 24, 128)	73728
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
conv2d_4 (Conv2D)	(None, 24, 24, 128)	147584
p_re_lu_4 (PReLU)	(None, 24, 24, 128)	73728
batch_normalization_4 (Batch	(None, 24, 24, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
conv2d_5 (Conv2D)	(None, 12, 12, 256)	295168
p_re_lu_5 (PReLU)	(None, 12, 12, 256)	36864
batch_normalization_5 (Batch	(None, 12, 12, 256)	1024
conv2d_6 (Conv2D)	(None, 12, 12, 256)	590080
p_re_lu_6 (PReLU)	(None, 12, 12, 256)	36864
batch_normalization_6 (Batch	(None, 12, 12, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 256)	0
conv2d_7 (Conv2D)	(None, 6, 6, 512)	1180160
p_re_lu_7 (PReLU)	(None, 6, 6, 512)	18432
batch_normalization_7 (Batch	(None, 6, 6, 512)	2048
conv2d_8 (Conv2D)	(None, 6, 6, 512)	2359808
p_re_lu_8 (PReLU)	(None, 6, 6, 512)	18432
batch_normalization_8 (Batch	(None, 6, 6, 512)	2048
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 32)	147488
p_re_lu_9 (PReLU)	(None, 32)	32
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
p_re_lu_10 (PReLU)	(None, 16)	16
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 7)	119

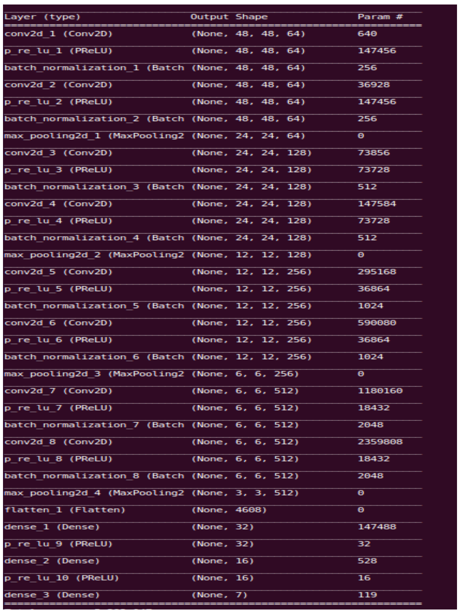
fig. 2 模型架構二

參數一欄表	
Total params	5,393,047
Trainable params	5,389,207
Non-trainable params	3,840
Batch_size	64
epoch	100
Dropout_rate	0.2

準確率	
Training_val_accu	0.6928
Kaggle_accu	0.67539

Model 檔名：weighrs.best_67539.hdf5

模型架構三(best)：



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
p_re_lu_1 (PReLU)	(None, 48, 48, 64)	147456
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
conv2d_2 (Conv2D)	(None, 48, 48, 64)	36928
p_re_lu_2 (PReLU)	(None, 48, 48, 64)	147456
batch_normalization_2 (Batch	(None, 48, 48, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
p_re_lu_3 (PReLU)	(None, 24, 24, 128)	73728
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
conv2d_4 (Conv2D)	(None, 24, 24, 128)	147584
p_re_lu_4 (PReLU)	(None, 24, 24, 128)	73728
batch_normalization_4 (Batch	(None, 24, 24, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
conv2d_5 (Conv2D)	(None, 12, 12, 256)	295168
p_re_lu_5 (PReLU)	(None, 12, 12, 256)	36864
batch_normalization_5 (Batch	(None, 12, 12, 256)	1024
conv2d_6 (Conv2D)	(None, 12, 12, 256)	590080
p_re_lu_6 (PReLU)	(None, 12, 12, 256)	36864
batch_normalization_6 (Batch	(None, 12, 12, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 256)	0
conv2d_7 (Conv2D)	(None, 6, 6, 512)	1180160
p_re_lu_7 (PReLU)	(None, 6, 6, 512)	18432
batch_normalization_7 (Batch	(None, 6, 6, 512)	2048
conv2d_8 (Conv2D)	(None, 6, 6, 512)	2359808
p_re_lu_8 (PReLU)	(None, 6, 6, 512)	18432
batch_normalization_8 (Batch	(None, 6, 6, 512)	2048
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 32)	147488
p_re_lu_9 (PReLU)	(None, 32)	32
dense_2 (Dense)	(None, 16)	528
p_re_lu_10 (PReLU)	(None, 16)	16
dense_3 (Dense)	(None, 7)	119

fig. 3 模型架構三

參數一欄表	
Total params	5,393,047
Trainable params	5,389,207
Non-trainable params	3,840
Batch_size	128
epoch	100

準確率	
Training_val_accu	0.69105
Kaggle_accu	0.68375

Model 檔名：weighrs.best_68375.hdf5

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：

以下數據分析皆基於第一題之模型架構 1 回答

(1) data normalization → 將 28709 張 image 的每個對應的位置，例如每張 image 的(i,j) = (1,1)，作 standardization 的 normalization，即找出每個位置的灰階值的平均再除以標準差，要注意的是除的標準差不能為零，因此我會在分母加一個很小的數值(1e-21)。

準確率	Val_acc	Kaggle
Before normalizaed	0.6707	0.68264
After normalizaed	0.7061	0.67957

(2) data augmentation → 我分別對我的 training data 與 validation data 做一樣的 augmentation。而我使用的函式是 keras 在影像處理套件中的 ImageDataGenerator，分別賦予 featurewise_center(使輸入數據集去中心化)、samplewise_center(使輸入數據的每個樣本均值為 0)、feature_std_normalization(將輸入除以數據集的標準差以完成標準化)、samplewise_std_normalzation(將輸入的每個樣本除以自身的標準差)，四個參數為 False；Width_shift_range, height_shift_range 為 0.2；zoom_range=0.1，zca_whitening, horizontal_flip 與 vertical_flip 皆為 False。我發現若是沒有做 data augmentation，會在比較前面的 epoch 就開始 overfitting。

準確率	Val_acc	Kaggle
Before data augmentation	0.65448	0.63304
After data augmentation	0.7061	0.67957

3. **(1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？**

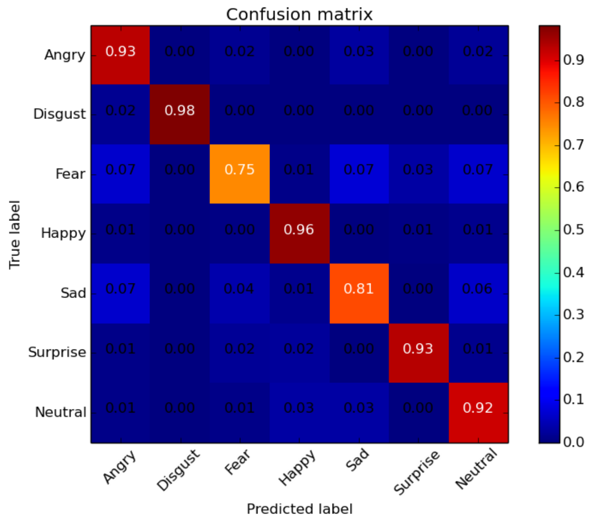


fig. 4 confusion matrix of structure 1

我使用模型架構 (fig. 1)來估測我從 training data 所切割出來的 validation data (0.1% 的 training data) 並與 labels 做比對，使用 sklearn.metrics 中的 confusion_matrix 套件，將 predict 的結果與 label 的結果當作參數傳入，並使用 matplotlib.pyplot 畫出圖形。

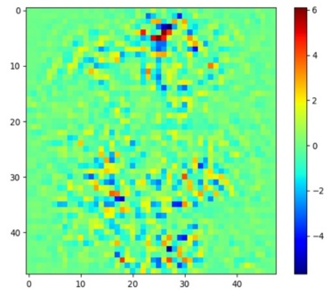
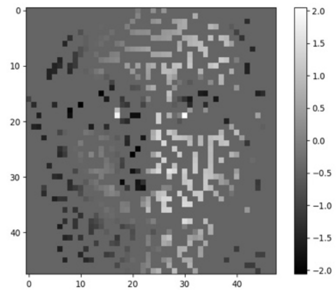
從 fig. 4 中我發現，準確率最低的是 fear 表情，其次是 sad，兩者都屬於負面的表情。而在 fear 的 row 資料來看，我發現這類的表情最容易與 angry, sad 和 neutral 搞混，分別都有 0.07%的機率會判斷錯誤。而 sad 的部分，與 angry 搞混的機率也高達 0.07%。其實這個結果不難在日常生活中體會到，有些人本來就很容易無法分辨這幾種表情，當然就這次的作業來說，這個結果也很符合我們平時的判斷結果。

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？**

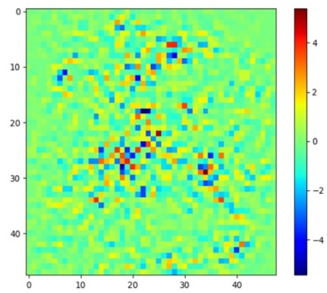
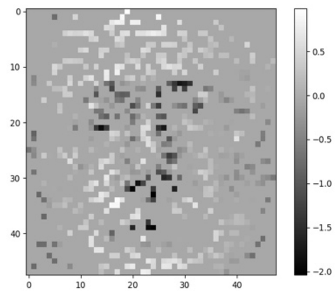
答：合理說明 test 的圖片和觀察到的東西 -> 0.5 分、貼出 saliency 圖片 -> 0.5 分

(1)

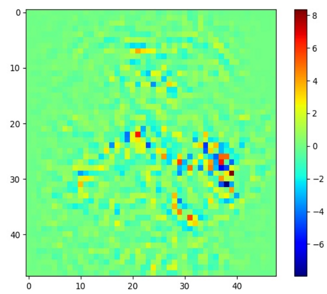
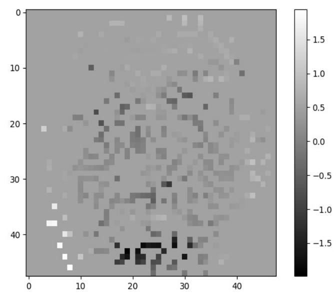
原圖	Saliency Map	Mask 掉 heat 小的部份

(2)

原圖	Saliency Map	Mask 掉 heat 小的部份
		

(3)

原圖	Saliency Map	Mask 掉 heat 小的部份
		

從(1)~(3)的結果我發現，CNN 所截取出的 saliency map 大多集中考慮在人臉的臉頰部分，從 saliency map 可以看到，三者在於臉頰的部分比重都在紅色的區間，這部分我認為跟人類判斷的基準也符合，也就是說，其實我們可以透過看臉頰上的紋路，例如：開心的時候嘴角會上揚、臉頰紋路會有上揚的形狀，相對的，其他表情也有相似的興致。其次像眉毛、額頭處也可以看到有較高的比重存在，這部分可能是透過眉毛的緊鎖程度，來看這個人的表情是正面與否。

5. (1%) 承(4) 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 與觀察 filter 的 output。

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

我所展示的是 CNN 中第一層的 convolution 與下一層 PReLU 的輸出，而第一層的輸出同時也時 PReLU 的 input，總共取 32 個 filter，而這 32 張 image 也是讓 filter 最 activated 的 image。每種 filter 的工作都是偵測邊緣，我想因為是第一層的 convolution 的關係，所以還看不到出每個 filter 的工作內容上的差異。而這點也合理，一般來說我們在做影像處理時，如果要判斷一個人的表情，基本上都會先做 edge detection，因此我認為這層 convolution 的輸出符合分析結果。

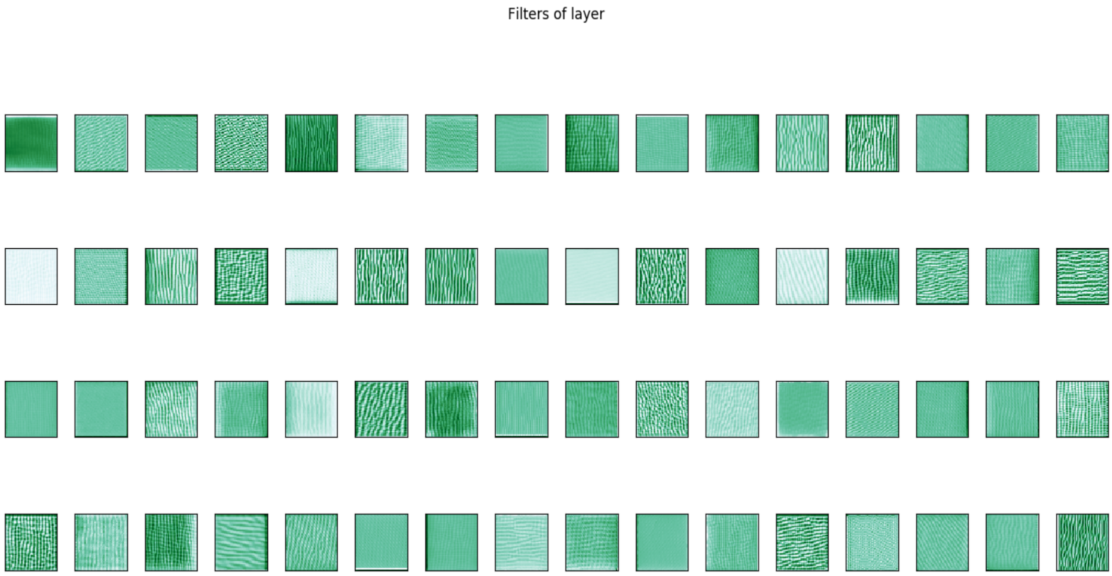


fig. 5 filters of layer ‘conv2d_2’

fig. 6 Output of layer ‘conv2d_2’ (input of ‘p_re_lu_2’)

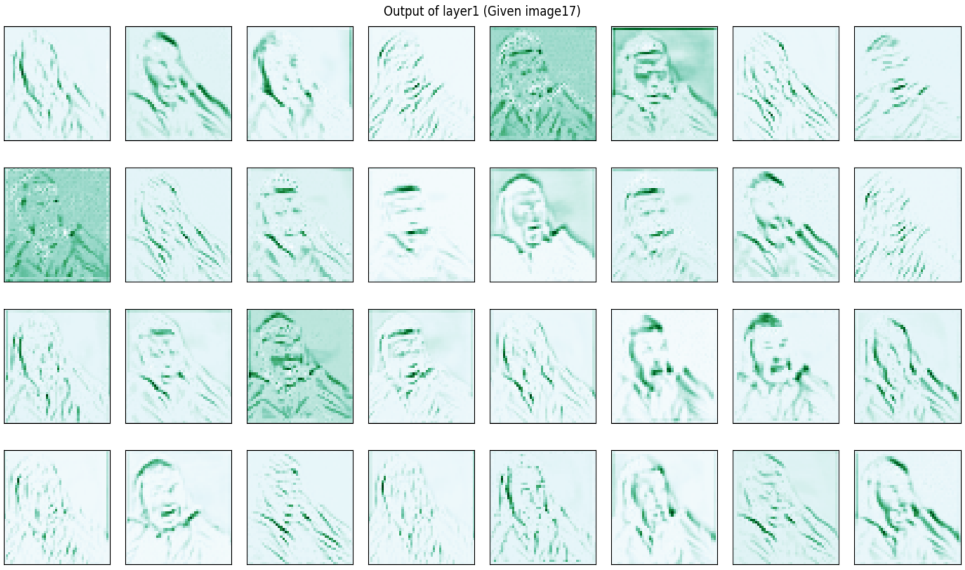


fig. 7 Output of layer ‘p_re_lu_2’