學號：R06522828 系級： 機械碩一 姓名：王榆昇

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:) 模型架構:
簡圖如下:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
conv2d_2 (Conv2D)	(None, 48, 48, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
conv2d_4 (Conv2D)	(None, 24, 24, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
conv2d_5 (Conv2D)	(None, 12, 12, 256)	295168
conv2d_6 (Conv2D)	(None, 12, 12, 256)	590080
conv2d_7 (Conv2D)	(None, 12, 12, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 256)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 256)	2359552
dense_2 (Dense)	(None, 256)	65792
dense_3 (Dense)	(None, 7)	1799

每個 convolution layer 的 kernal_size 皆是 3x3，kernal_initializer 設定 'random_uniform'，activation function 皆是 LeakyReLU($\alpha = 0.1$)，其後都會接上 Dropout，並在每個 max pooling 前加上 BatchNormalization。在 flatten 後，dense_1 和 dense_2 的 activation functions 都是 LeakyReLU($\alpha = 0.1$)，其後再接上 BatchNormalization 和 Dropout。
< 訓練參數 >
取 train_data 內 90%資料做 training，並採用 keras 內建的 ImageDataGenerator 生成資料(同地 2 題之參數設定)，設定 batch_size = 256，steps_per_epoch = 3 * len(x_train) // batchsize，epochs = 400。
< 準確率 >

	Train	Validation	Public	Private
Accuracy	0.8448	0.6991	0.70771	0.69350

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？ (Collaborators:)

* 以下結果都已先把圖片從 8-bit 轉成 float (每個 pixel 值同除 255) 再使用，設定 epochs=50

Normalization	Augmentation	Training acc.	Public Score	Private Score
x	x	0.9020	0.63471	0.63806
v	x	0.9069	0.64363	0.64424
x	v	0.60967	0.67456	0.67177

Normalization:

計算全部圖片的 pixel 的平均和標準差(mean = np.mean(x_all)，sigma = np.std(x_all))，再把每張圖片的每個 pixel 做高斯標準化((image[i.j] – mean) / sigma)。

由於我們已經先知道原始圖片為 8-bit，除以 255 轉成 float 型式時就已相當程度的優化了 data，因此在做 gaussion normalization 後雖有改進 accuracy，但效果並不明顯。

Augmentation:

使用 kears 內建函數 ImageDataGenerator 生成資料，

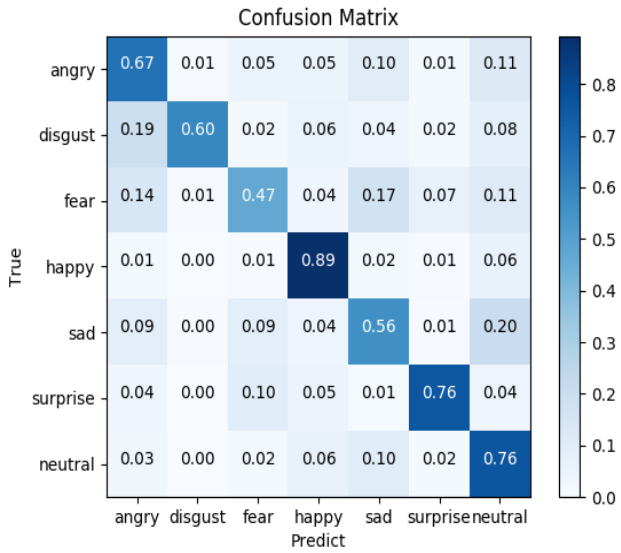
```
ImageDataGenerator(rotation_range=20, width_shift_range=0.2, height_shift_range=0.2, zoom_range=[0.8, 1.2], shear_range=0.2, horizontal_flip=True)
```

如此能隨機產生旋轉 20 度內、上下左右平移 0.2 圖片大小、縮放 0.8~1.2 倍間、錯切變換和鏡像的圖片，再配合 model.fit_generator()做訓練。

由上表可注意到，在生成資料後，training acc.的收斂速度比起生成資料前慢了很多 (0.60967)，但 Kaggle 上得到的 test accuracy 卻增加許多；這說明了對圖片做旋轉、平移等處理後，對機器來講就是一張新的圖片，使得機器能學習更多資料而提高預測準確率。

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] (Collaborators:

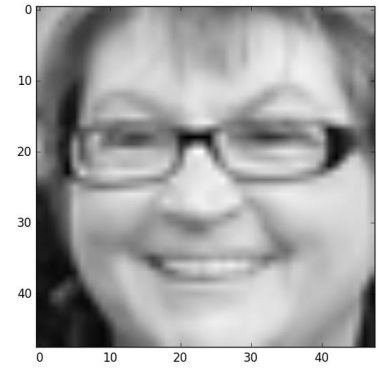
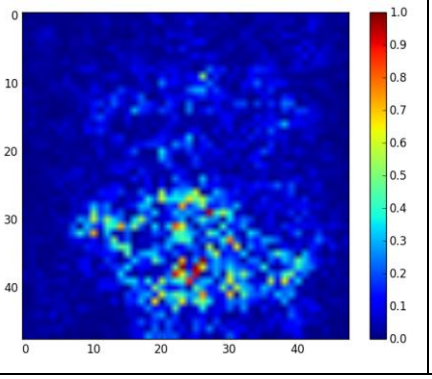
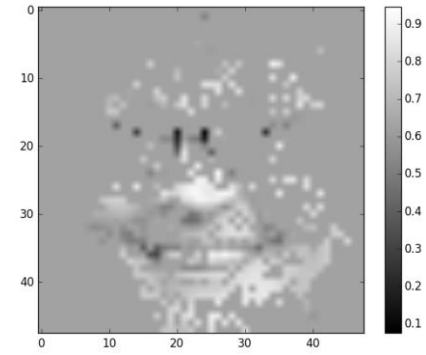
http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html)



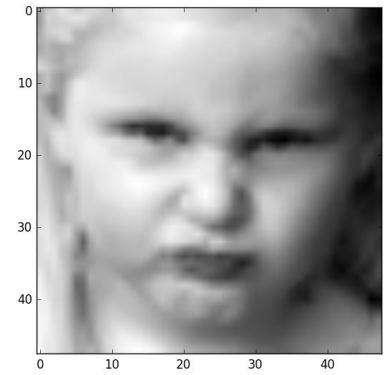
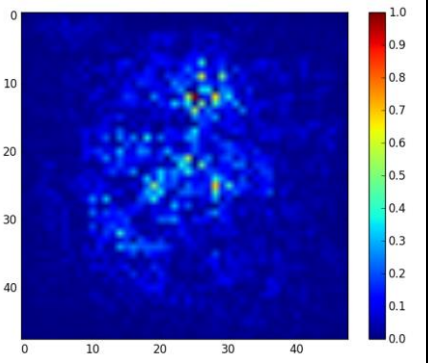
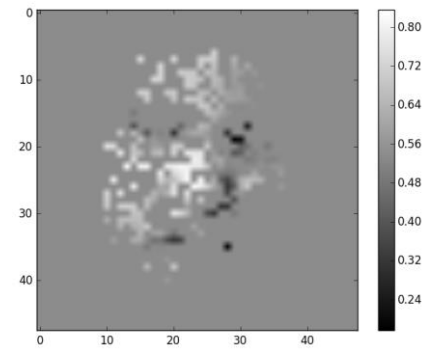
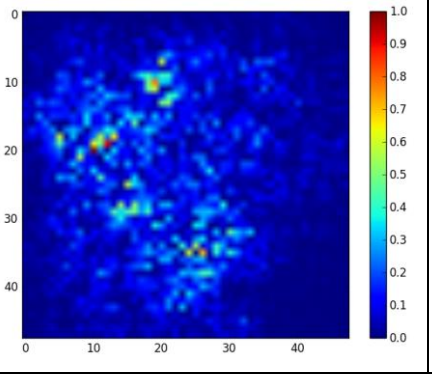
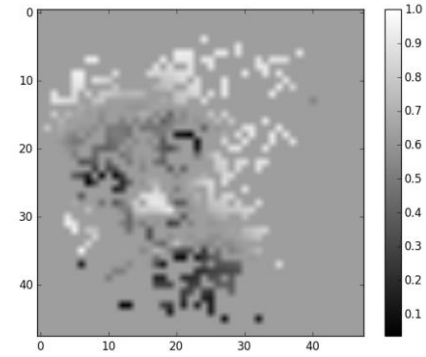
由左圖可見，angry 容易被誤認為 sad、neutral，disgust 非常容易被誤認成 angry，fear 容易被誤認為 angry、sad、neutral，而 sad 則是容易被誤認為 neutral。分析以上結果可以發現，容易被誤認的都是負面情緒，且易被誤認成 angry 和 neutral，我認為是因為負面情緒的特徵較不明顯且相似性高，導致負面情緒的準確率無法提高且容易被判成難以判斷(某方面的特徵不明顯)的 neutral 類別。

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？

(Collaborators: https://github.com/raghakot/keras-vis/blob/master/examples/mnist/attention.ipynb)

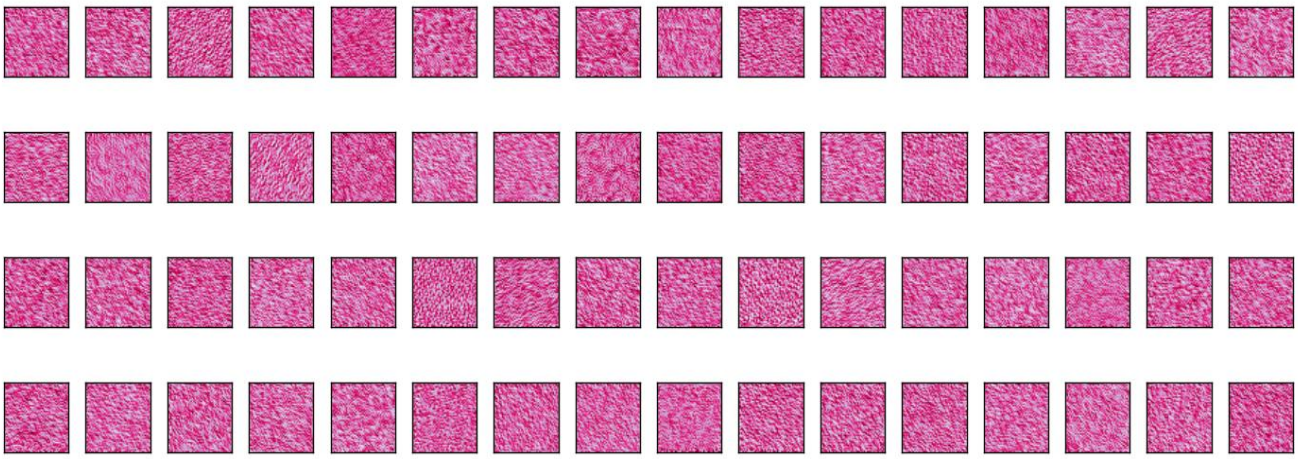
Happy	Heatmap	Masked
		

人笑的時候，眉毛會上彎、嘴巴會成切片西瓜狀，而 masked map 也顯示出 CNN 的確集中在眉毛、嘴巴和鼻子的特徵點。

Angry	Heatmap	Masked
		
sad	Heatmap	Masked
		

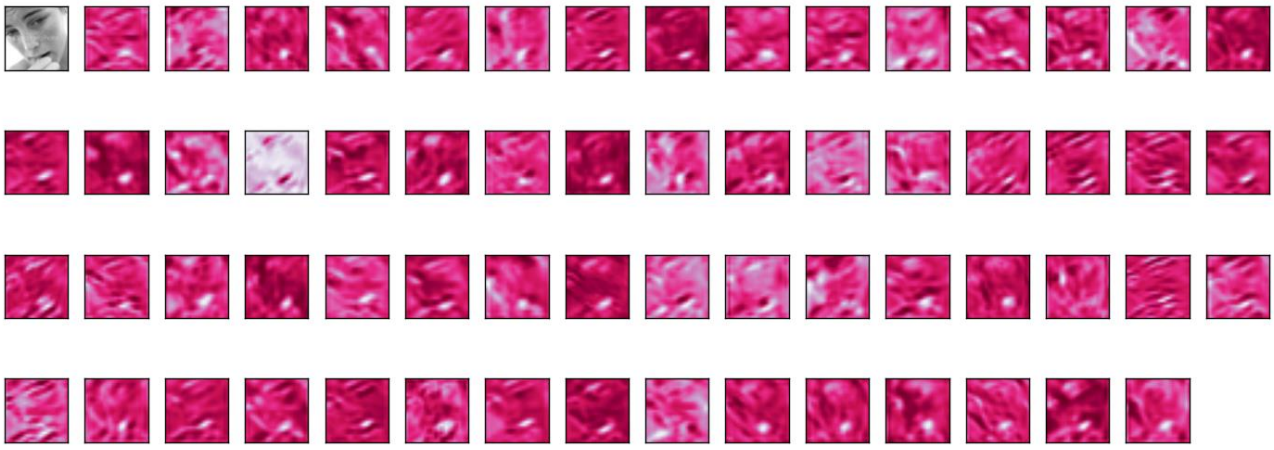
從負面情緒的圖片上來看，人在做負面情緒時臉部動作較小，且嘴巴大多緊閉，對應到 masked map 時可發現 CNN 集中注意臉、鼻子組成的區域和嘴巴。我推測嘴巴的特徵在負面情緒中大多是緊閉，不若 happy、surprise 會有突出的特點，才會導致我的模型在分類時容易把負面情緒搞混。

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。
(Collaborators:
https://github.com/keras-team/keras/blob/master/examples/conv_filter_visualization.py)
觀察 model 第三個 convolution 層(128 filters)中的前 64 個 filter，可以最大化該層 filter output 的圖片:



由於層數還算前面，filter 看到的重點還是在整張圖上規律出現的特徵，因此能最大化的圖還不算複雜，而這 64 種 filters 所觀察的地方都不盡相同，因此我們會需要足夠多的 filtes，才能夠完整形容 training data 上的特徵。

將 validation set 隨機一張圖片放入該層觀察:



由上圖可看到，經過第三層 filter 的圖片已大約可辨識出人的輪廓，且有些圖片已經很強調眉毛、嘴巴等重點部位，但由於還是屬於淺層的 filter，人的特徵依然還是不明顯。