

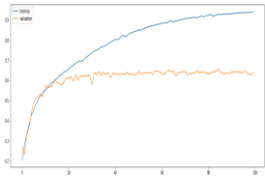
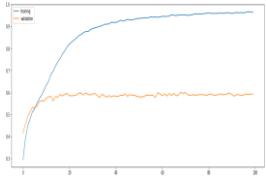
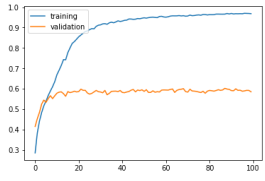

1. (1%) 請說明你實作的 CNN model，其模型架構、訓練參數和準確率為何？

在下表中我列出了四種我嘗試的 Model，第一個是參考

VGG16(<https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3>)並改進過後

的 Model，也是 performance 最好的，另外則是使用較小的 Model，並更改其 filter 大小去觀察，我們可以發現似乎較大的 filter 可以得到較快的收斂，然而整體來看最後的 performance 差異並不大，而我最後也嘗試直接使用 DNN，但可以看到結果並不好，而且波動極大，尚未到達收斂。

參數: batch_size = 128, epoch = 100

Model	1. VGG-like	2. CNN-Small	3. CNN-Small	4. DNN
Filter	(3, 3)	(3, 3)	(5, 5)	(3, 3)
ACC	0.6429	0.5886	0.5924	0.448275
				

```
def CNN_Small():
    filt_size = (3, 3)
    model2 = Sequential()
    model2.add(Convolution2D(32, filt_size, input_shape=(48,48,1), activation='relu', padding='same'))
    model2.add(BatchNormalization())
    model2.add(MaxPooling2D((2,2)))
    model2.add(Dropout(0.2))
    model2.add(Convolution2D(64, filt_size, activation='relu', padding='same'))
    model2.add(BatchNormalization())
    model2.add(MaxPooling2D((2,2)))
    model2.add(Dropout(0.3))

    model2.add(Flatten())
    model2.add(Dense(512, activation='relu'))
    model2.add(BatchNormalization())
    model2.add(Dropout(0.5))
    model2.add(Dense(512, activation='relu'))
    model2.add(BatchNormalization())
    model2.add(Dropout(0.5))
    model2.add(Dense(7))
    model2.add(Activation('softmax'))
    return model2
```

```
def VGG16_like():
    filt_size = (3, 3)
    model2 = Sequential()
    model2.add(Convolution2D(32, filt_size, input_shape=(48,48,1), activation='relu', padding='same'))
    model2.add(Convolution2D(32, filt_size, activation='relu', padding='same'))
    model2.add(BatchNormalization())
    model2.add(MaxPooling2D((2,2)))
    model2.add(Dropout(0.1))

    model2.add(Convolution2D(64, filt_size, activation='relu', padding='same'))
    model2.add(Convolution2D(64, filt_size, activation='relu', padding='same'))
    model2.add(BatchNormalization())
    model2.add(MaxPooling2D((2,2)))
    model2.add(Dropout(0.3))

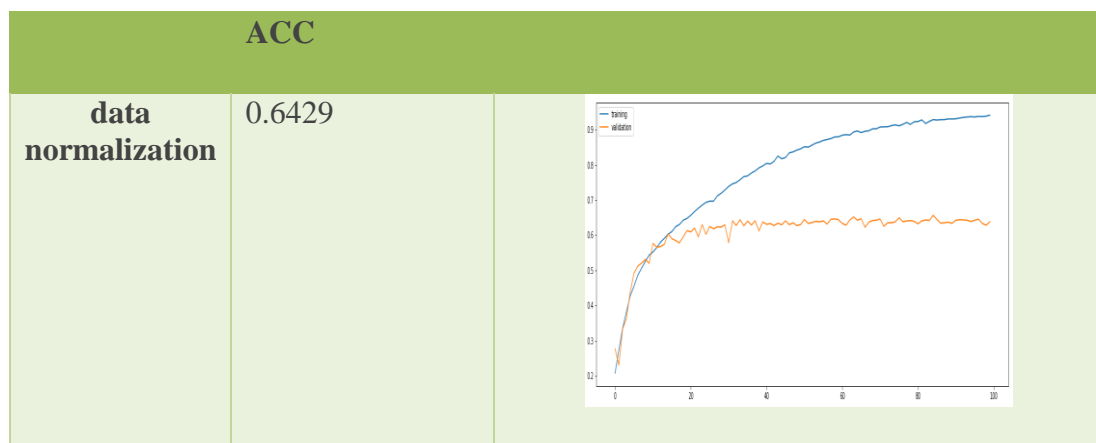
    model2.add(Convolution2D(128, filt_size, activation='relu', padding='same'))
    model2.add(Convolution2D(128, filt_size, activation='relu', padding='same'))
    model2.add(Convolution2D(128, filt_size, activation='relu', padding='same'))
    model2.add(BatchNormalization())
    model2.add(MaxPooling2D((2,2)))
    model2.add(Dropout(0.4))

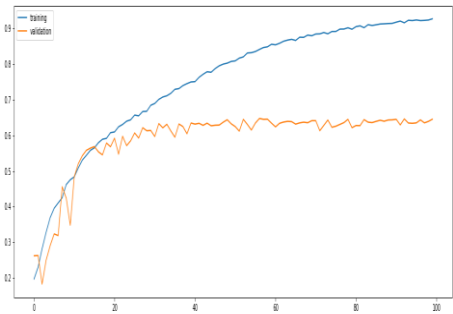
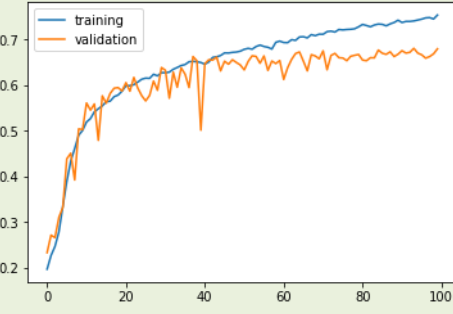
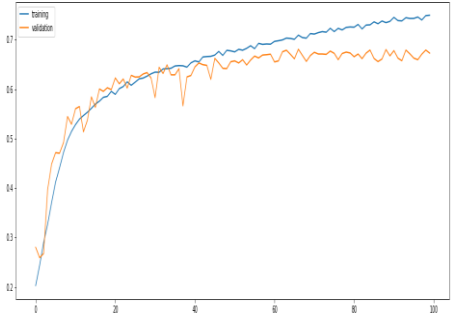
    model2.add(Convolution2D(256, filt_size, activation='relu', padding='same'))
    model2.add(Convolution2D(256, filt_size, activation='relu', padding='same'))
    #model2.add(Convolution2D(256, filt_size, activation='relu', padding='same'))
    model2.add(BatchNormalization())
    model2.add(MaxPooling2D((2,2)))
    model2.add(Dropout(0.5))

    model2.add(Flatten())
    model2.add(Dense(1024, activation='relu'))
    model2.add(BatchNormalization())
    model2.add(Dropout(0.5))
    model2.add(Dense(1024, activation='relu'))
    model2.add(BatchNormalization())
    model2.add(Dropout(0.5))
    model2.add(Dense(7))
    model2.add(Activation('softmax'))
```

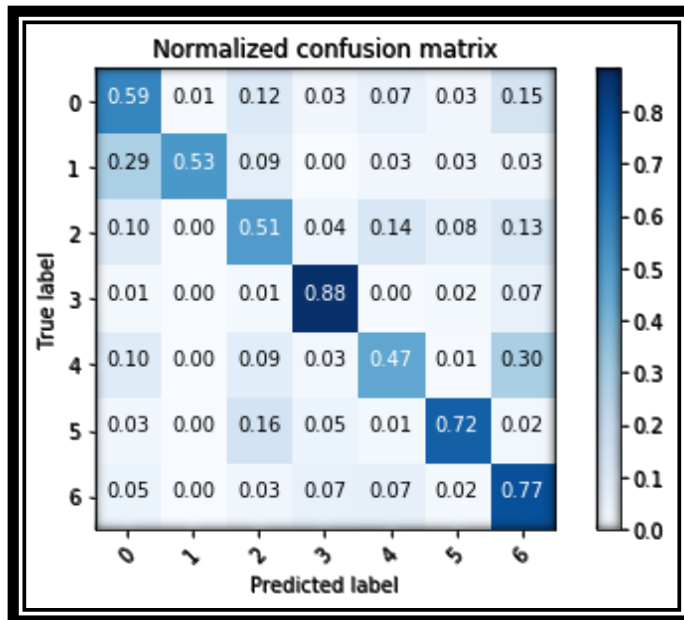
2. (1%) 請嘗試 data normalization, data augmentation, 說明實行方法並且說明對準確率有什麼樣的影響？

在 normalization 的實驗中是使用 Standard Score ($(X-\mu) / \sigma$)，若以結果來看兩者相差並不大，但觀察沒有 normalization 的組別，在前 20 個 epoch 明顯出現比較大的波動，最後的波動也比較大，收斂的速度似乎較慢。而 data augmentation 我則是使用 Keras 的 image generator，並給圖片進行部分的 shift 及 rotate，最後則是得到相當好的結果，也成功突破 Strong Baseline！



Without data normalization & Without data augmentation	0.6412	
data augmentation	0.6799	
Using Both Scheme	0.6729	

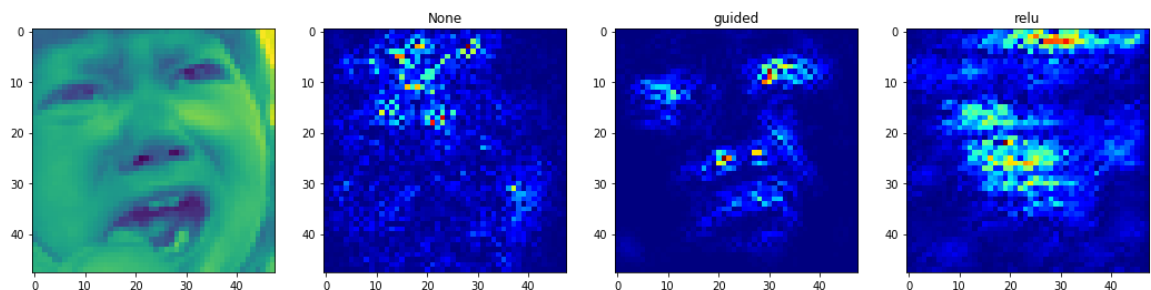
3. (1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？



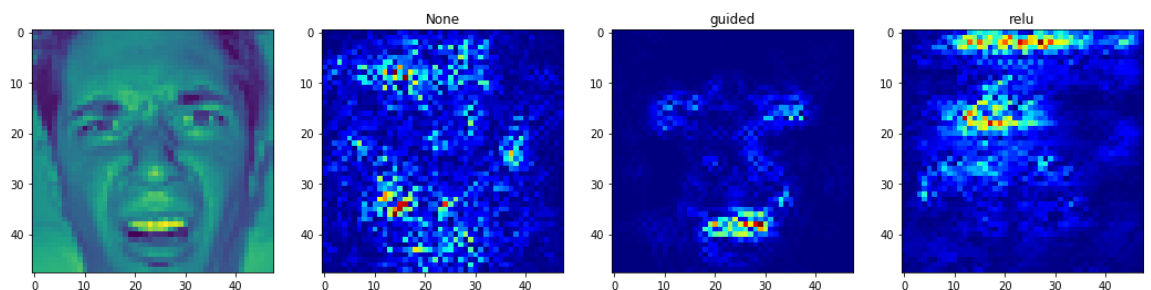
從 confusion matrix 中可以觀察到，最準確的是高興這個分類。而容易混淆的則是中立 vs 難過，以及生氣 vs 厭惡。

- (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？

Class 0: 生氣



Class 1: 厭惡

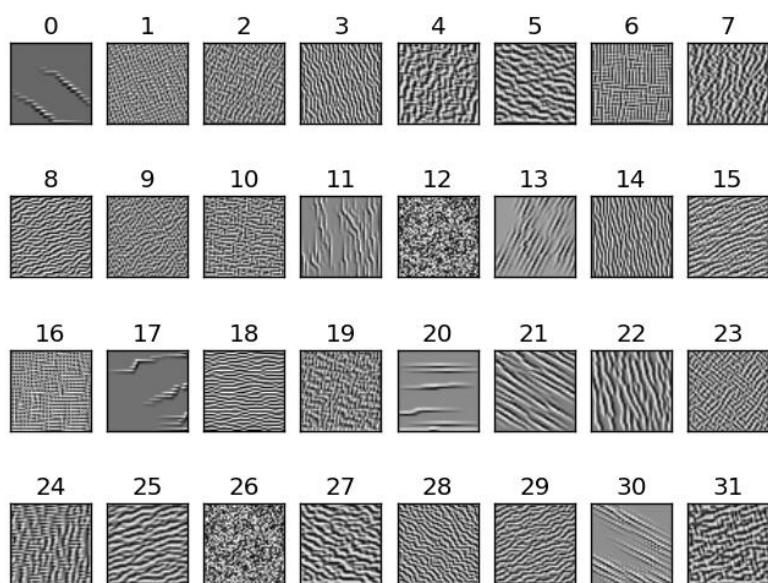


我們選擇較易被混淆的 class 0 及 class 1 來觀察，可以看到大多聚焦在眼睛及嘴型上，而其實即便以人眼都很難做出判斷。

- (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:)

下面兩張圖分別是 CNN 第一層及最後一層，可以看到第一層 activate 都是一些紋路 pattern，然而到了最後一層，卻幾乎無從辨別，我想這是因為經過多次 MaxPooling 的關係，經過 Subsampling 之後圖片用肉眼已經難以辨認。

第一層



最末層

