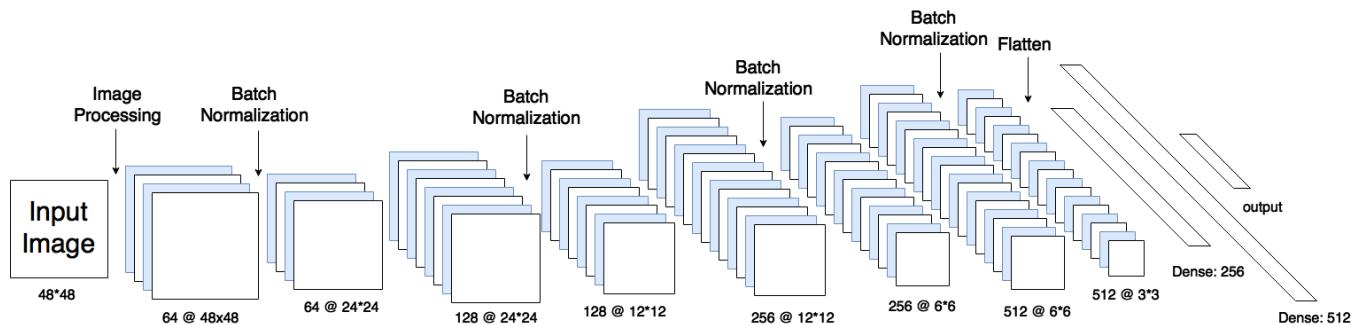


1. (1%) 請說明你實作的 CNN model，其模型架構、訓練過程和準確率為何？

答：[整體架構]



一開始我使用了 keras 的 ImageDataGenerator 針對圖片做前處理，透過旋轉、平移、剪力轉換、放大和水平翻轉，使得 training 資料量大增許多。

```
ImageDataGenerator(rotation_range=25, width_shift_range=0.1, height_shift_range=0.1,
                    shear_range=0.1, zoom_range=[0.8, 1.2],
                    horizontal_flip=True)
```

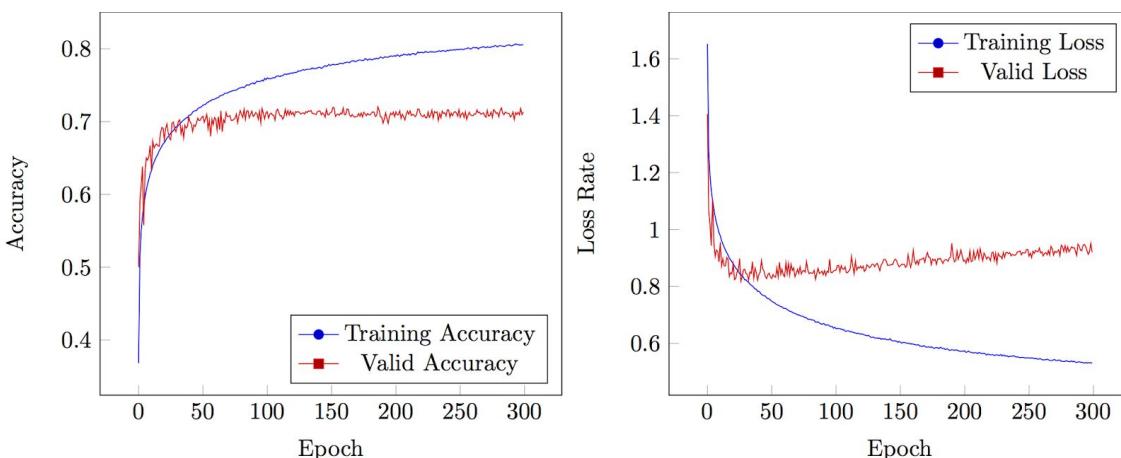
接著，我連續操作四次一個convolution接一個maxpooling的行為，在一組「convolution配上maxpooling」中的設計如下：

1. Conv2D: 4次中我都使用 kernel size=(3, 3) 的 filter 做 convolution，並加上以 same 作為規則的 padding，而 filter 數目以 64, 128, 256, 512 遞增
2. LeakyReLU: 使用 alpha=0.03 的 LeakyReLU 作為 convolution 後的 activation function
3. BatchNormalization: 針對每個 batch 做 normalization
4. MaxPooling2D: 4次中皆使用 pool_size=(2, 2)，並用 same 作為 padding 的規則
5. Dropout: 4次中dropout的比例依序為 0.2, 0.25, 0.3, 0.4 遞增

然後Flatten攤平接兩層Dense layer，最後才接output layer，在一層Dense layer中的設計：

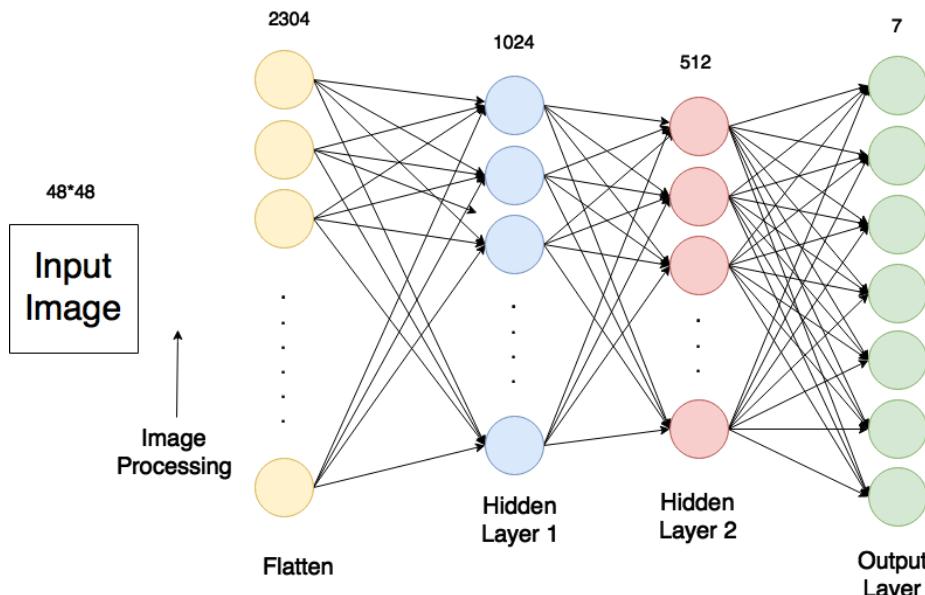
1. Dense: 第一層是 256，第二層是 512
2. BatchNormalization: 針對每個 batch 做 normalization
3. Dropout: 皆為 0.5

在 batch size = 128 和 epochs = 300 下，訓練資料與驗證資料的過程如下圖。我取了 training data 中的最後 3000 筆當作 valid data。Valid accuracy最後可收斂到 0.70~0.71左右，然後就一直在波動；而 valid loss 會先快速下降後緩緩上升。



2. (1%) 承上題，請用與上述 CNN 接近的參數量，實做簡單的 DNN model。其模型架構、訓練過程和準確率為何？試與上題結果做比較，並說明你觀察到了什麼？

答：[整體架構]



在這樣的架構下，參數數量約為289萬，而CNN版本的參數數量為287萬，極為相近。

一開始，如CNN版本，我也做了圖片的前處理（讓變因盡量降低方便比較），使用的是 keras 的 ImageDataGenerator，參數也同上題。然後 Flatten 將圖片攤平。

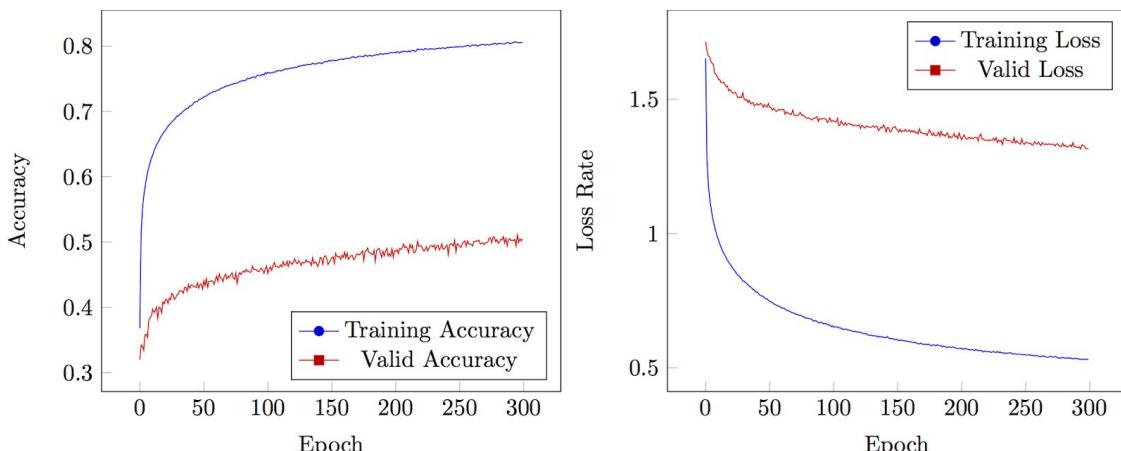
接著加上兩層Dense layer，設計如下：

1. 使用 alpha=0.03 的 LeakyReLU 作為 activation function
2. 加上 Batch Normalization
3. 都是 0.5 的 Dropout rate

兩層的差別僅在於第一層使用 1024 個 nodes，第二層使用 512 個 nodes

在 batch size = 128 和 epochs = 300 下，訓練資料與驗證資料對 epoch 的過程如下圖。Valid accuracy最後會落在0.5左右。

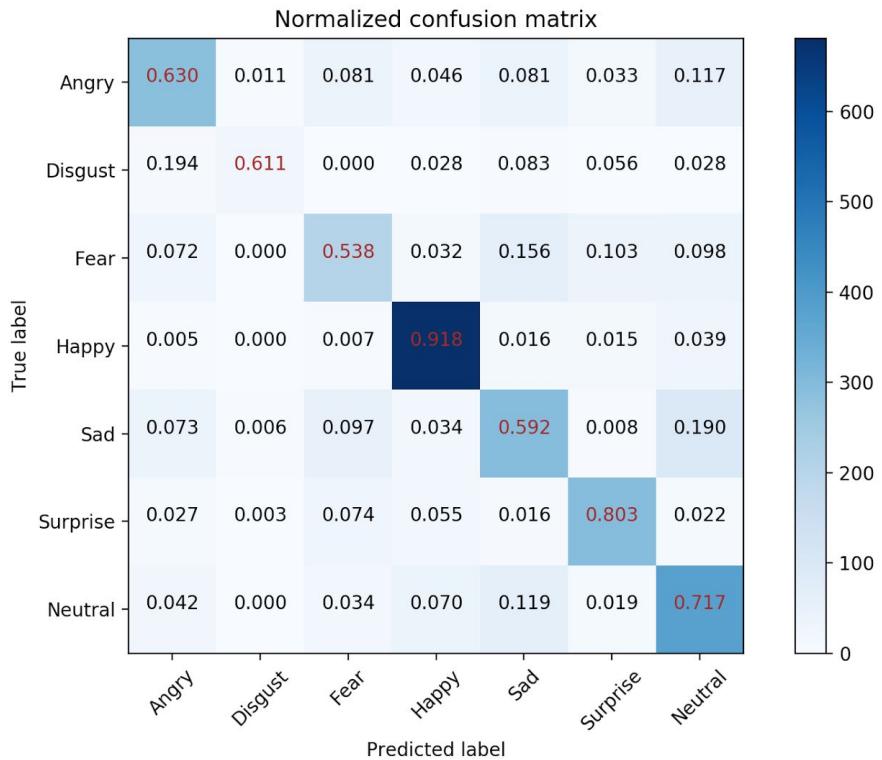
(這樣的 epoch 和 batch size 參數設計是為了要跟 CNN 版本作比較)



我發現在 DNN 版本下，對於一次 epoch 的訓練時間快 CNN 版本許多，應該是因為做 convolution 比較耗資源。但在 validation set 上的準確度差上不少，loss 下降的速度也比較慢 (validation set 切法同 CNN，是 training data 的最後 3000 筆)。

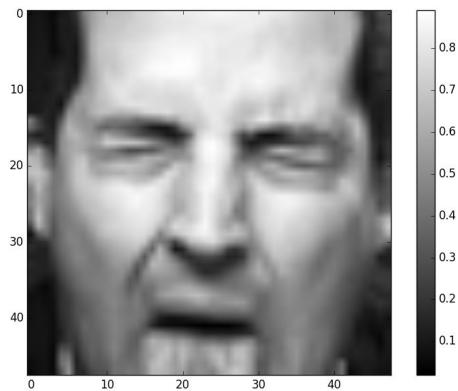
3. (1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析]

答：[confusion matrix 圖如下]

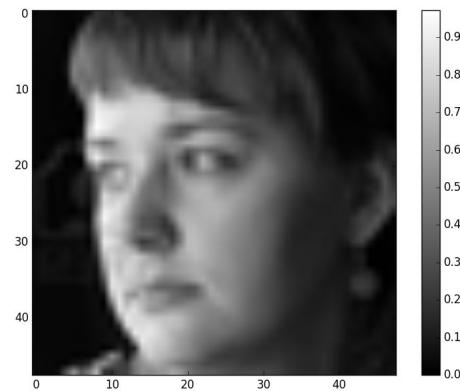


根據 confusion matrix 來看，厭惡很容易會被誤判為生氣；傷心會被誤判為中立；害怕和中立很容易都被誤判為傷心，所以我取出一些圖片做觀察。

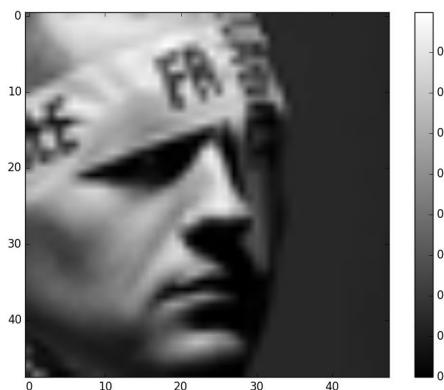
(1) [True: 厭惡; Predicted: 生氣]



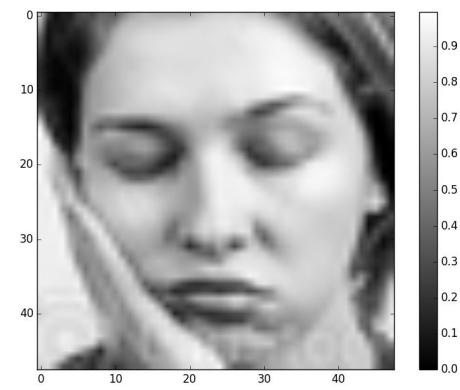
(2) [True: 傷心; Predicted: 中立]



(3) [True: 害怕; Predicted: 傷心]



(4) [True: 中立; Predicted: 傷心]



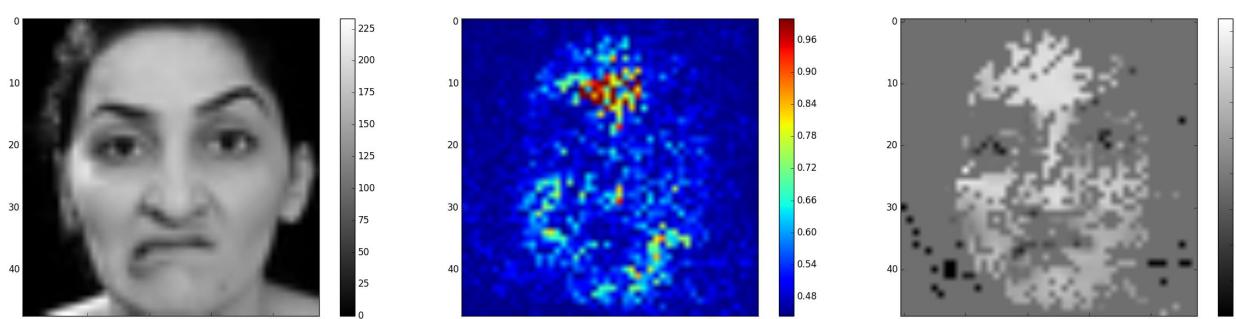
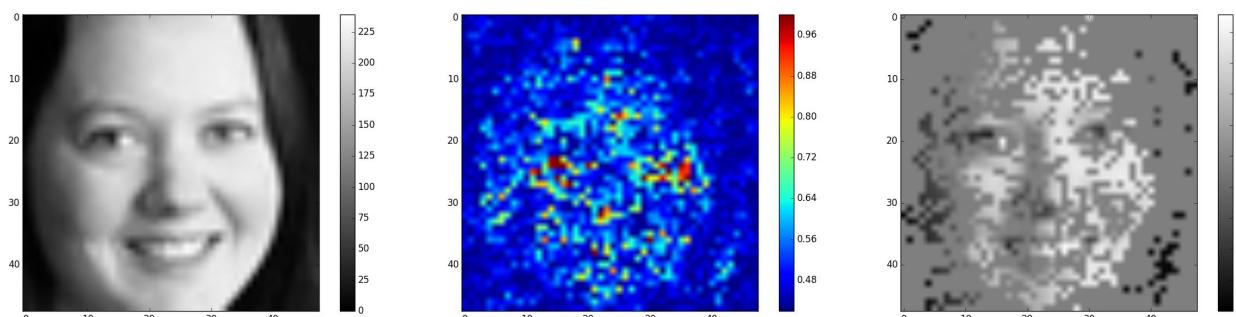
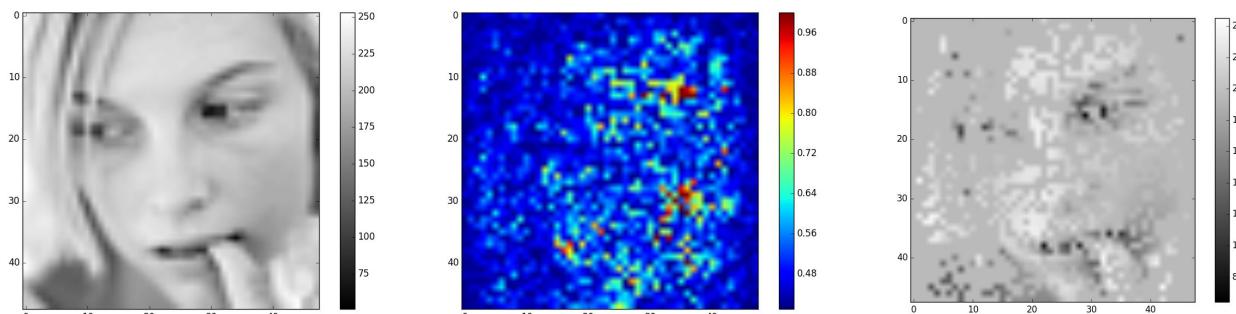
(紅色: true label, 藍色: predicted label)

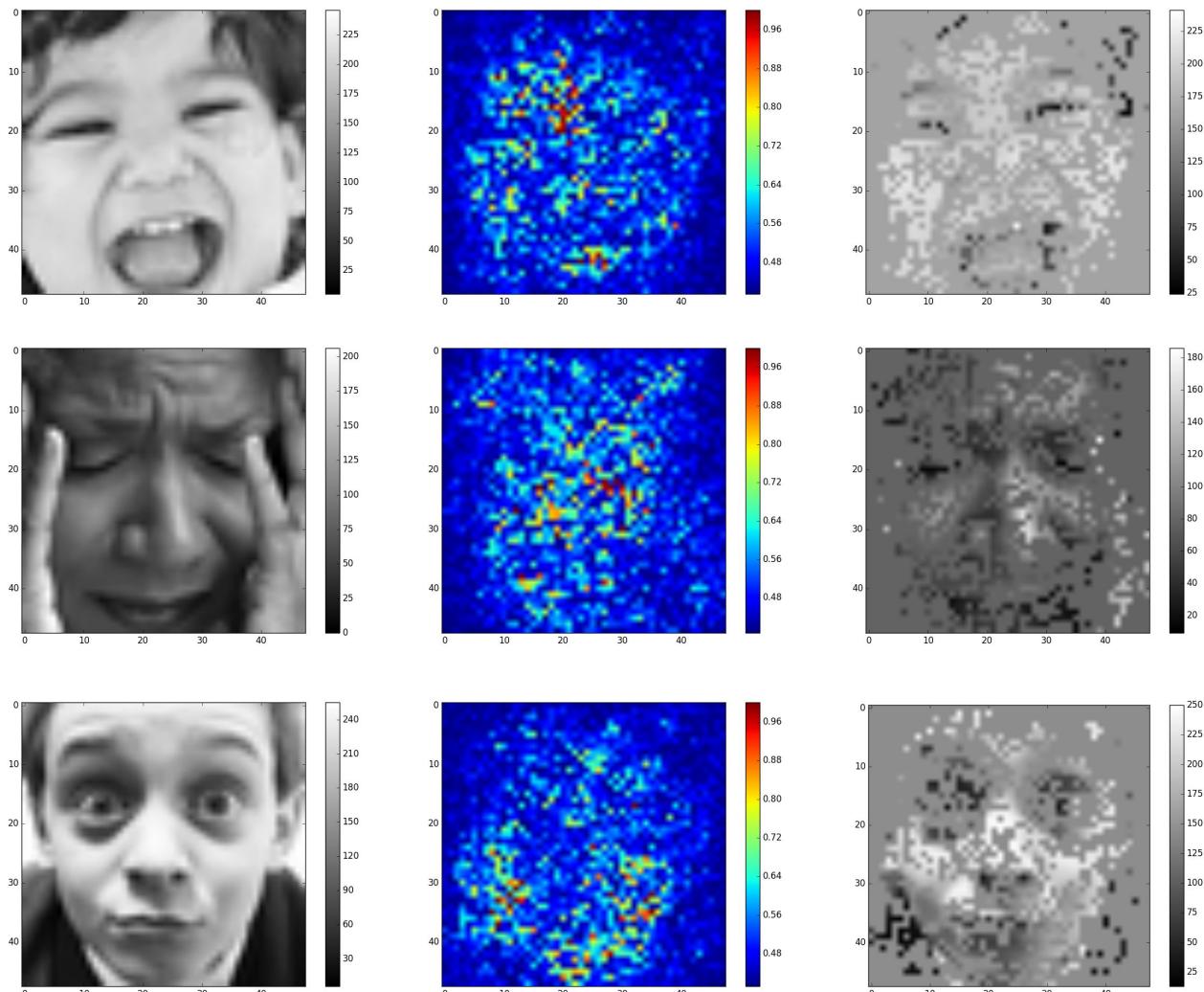
	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
1	0.495	0.320	0.128	0.004	0.021	0.001	0.032
2	0.203	0.000	0.071	0.012	0.102	0.012	0.598
3	0.022	0.000	0.380	0.000	0.498	0.002	0.098
4	0.014	0.000	0.122	0.000	0.523	0.002	0.338

從機率分佈來看，被取出的圖片的確是會造成判斷困擾的，尤其當最大機率跟其他機率差別沒有太大，並沒有特別突出或絕對勝出的時候。但只從數字觀察尚不明顯，所以我印出圖片輔以人眼辨別，發現這些圖片被 predict 錯並沒有很意外，就算透過人眼判斷也很容易辨識錯誤。

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：我特別將幾張不同 emotion 的圖片挑出來，顯示其 saliency maps 後分析，結果如下：

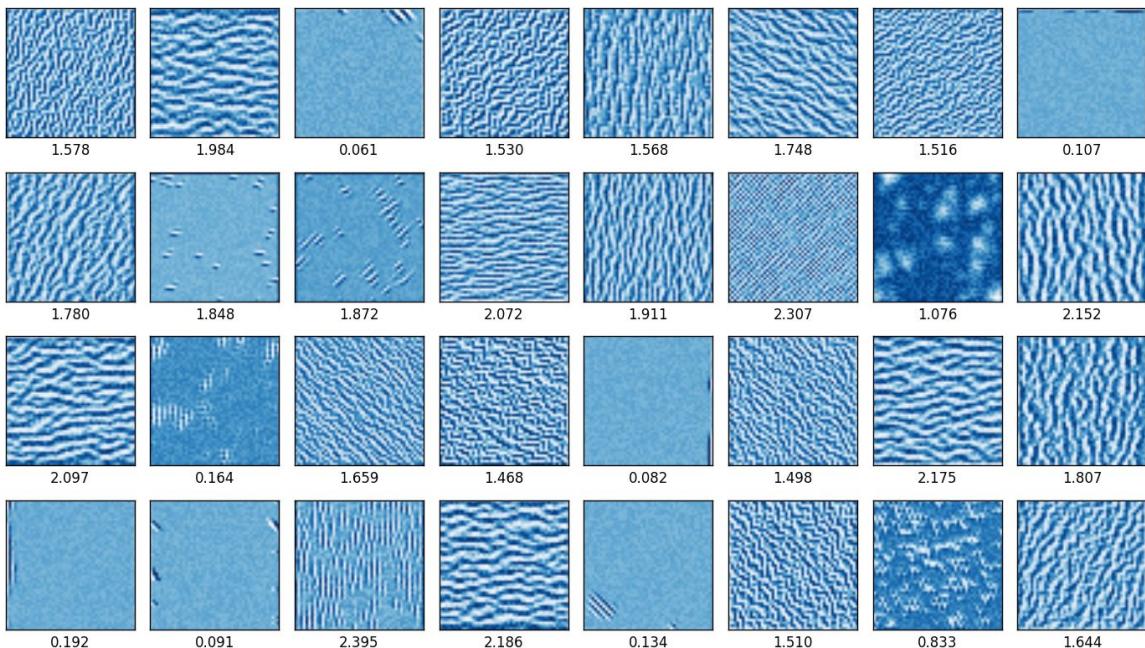




從上到下分別是：害怕 (Fear)、開心 (Happy)、生氣 (Angry)、開心 (Happy)、傷心 (Sad)、驚訝 (Surprise)。根據這些圖片，我發現 classification 會 focus 在「眼睛、臉頰跟嘴巴」的部分。

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：我挑第一層 convolution layer 過 activation 後的 filters 來觀察，得到以下圖片（取前 32 個作為

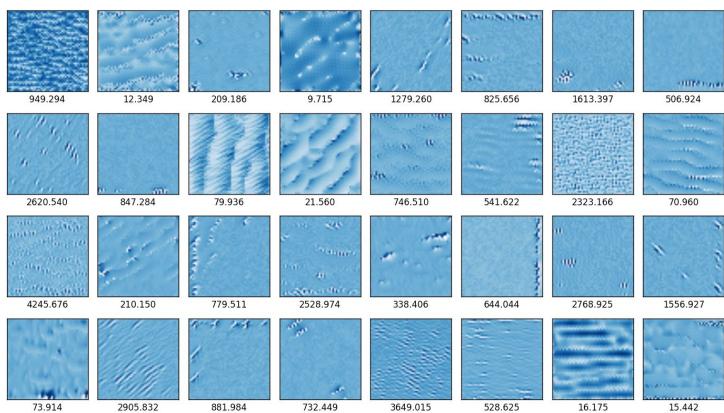


代表印出），這分別代表他們能感知到的 texture 的長相。若同時將我挑選的圖片（在 validation set 中編號為 17）作為輸入，會得到以下結果：

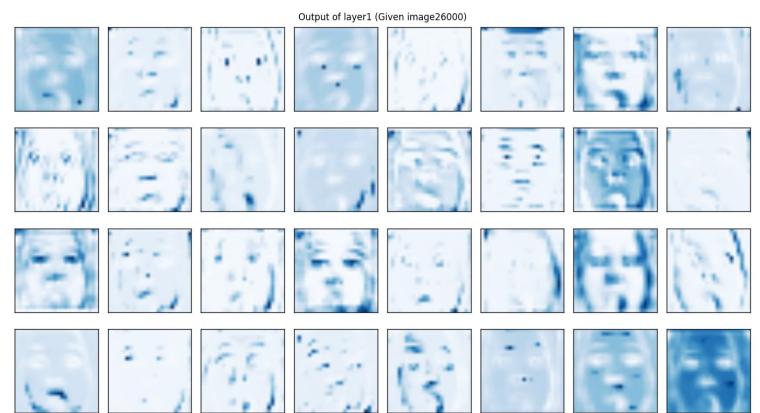


從中可以觀察到這 32 個作為第一層 hidden output layer 的圖片，就是在原圖片上找尋第一層 convolution layer 上 32 個各自的 filter 能抓到的 texture，並將之顯示出來。之後數層能使 filters 被 activate 最顯著的圖片和對應這些 filters 的 hidden output layer 的圖片如下：

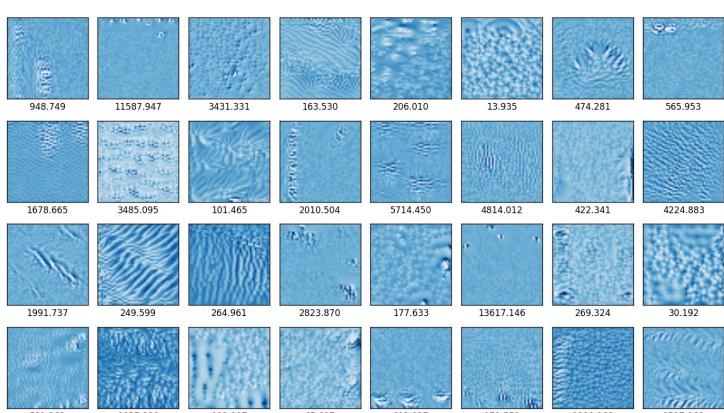
[第二層 convolution + activation 後的 filters]



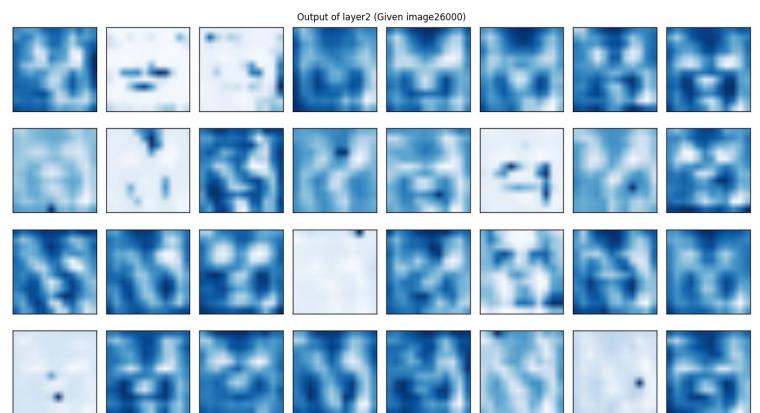
[在左邊 filters 作用下 output 的圖片]



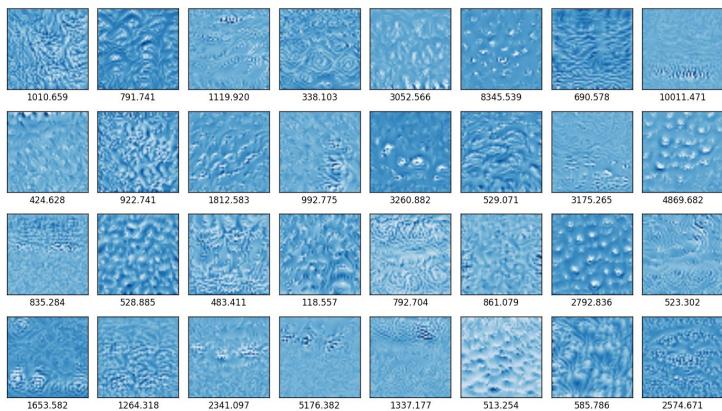
[第三層 convolution + activation 後的 filter]



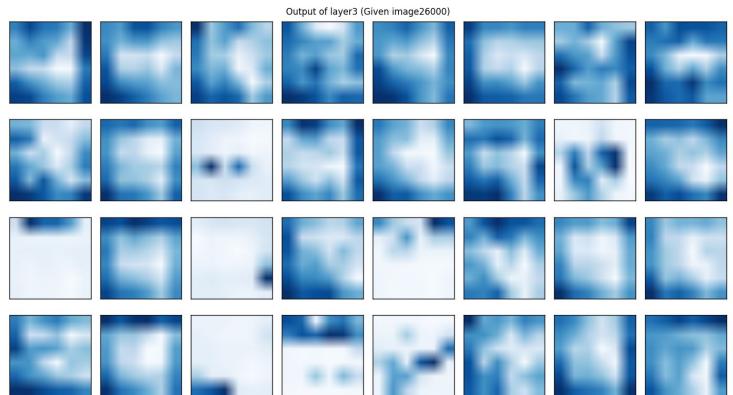
[在左邊 filters 作用下 output 的圖片]



[第四層 convolution + activation 後的 filter]



[在左邊 filters 作用下 output 的圖片]



不過還是有許多張 output 圖片跟他對應的 filters 負責抓出來的 texture 有點不符。我覺得應該是因為 maxpooling 的關係會超級模糊，幾乎無法辨識了。[這時使用的是能使我切出的 valid data accuracy = 0.71967 的 model]

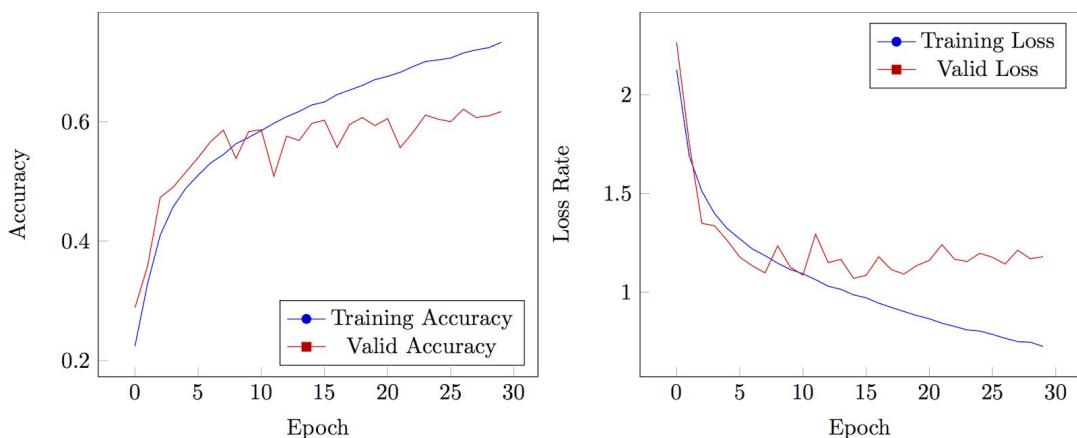
[Bonus] (1%) 從 training data 中移除部份 label，實做 semi-supervised learning

原來圖片有 28709 張，由於我切 3000 張作為 validation set，所以只剩 25709 張。這時切出 20000 張作為 unlabeled data，而 labeled data 會有 5709 張。這樣設計是為了盡量能模擬真實的狀況，即讓 unlabeled data 的數量遠大於 labeled data (不過在這裡考量到 labeled data 太少可能會造成 unlabeled data 的 label 嚴重判錯，所以做了點折衷)。

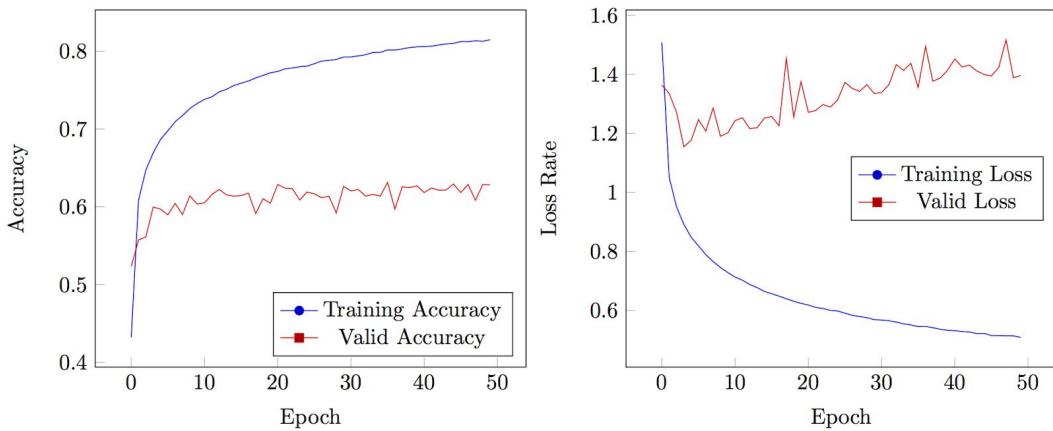
Labeled data	5709
Unlabeled data	20000
Valid data	3000

模型架構跟 CNN 版本相同，差別僅訓練 labeled data 的 epoch = 30，然後用此模型 (model 0) 預測 unlabeled data 的 label (這時 valid accuracy 差不多 0.61)，然後把這些新預測的資料全數塞入新的 training data 中，然後 train 一個新 model (model 1)，新的 model 則跑了 epoch = 50 (因為考慮到機器效能可能沒辦法在作業期限內跑完，所以使用的 epochs 數比較少，然後這時的 valid accuracy 差不多 0.62)。將他們兩個模型的訓練過程畫出如下：

[model 0]

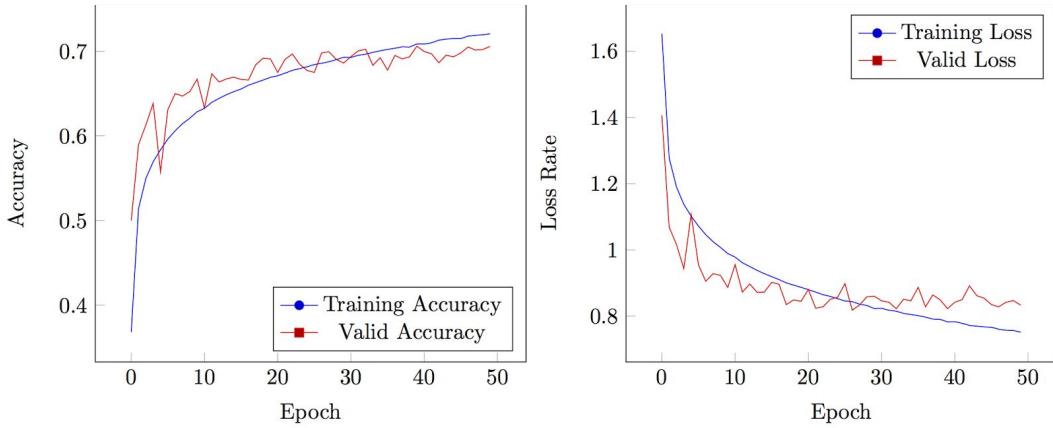


[model 1]



而將其與 CNN 的版本在相同 epochs 數下做比較 (即同為 epochs = 50), 可以發現 semi-supervised 的方法在精確度上收斂較快, 然後 valid loss 很快就開始上升了；相較下, CNN 能爬升的精確度較高, 然後根據第一題時附的圖, valid loss 也在比較後面的 epoch 才開始上升。比較圖如下：

[同樣 epoch 下的 CNN]



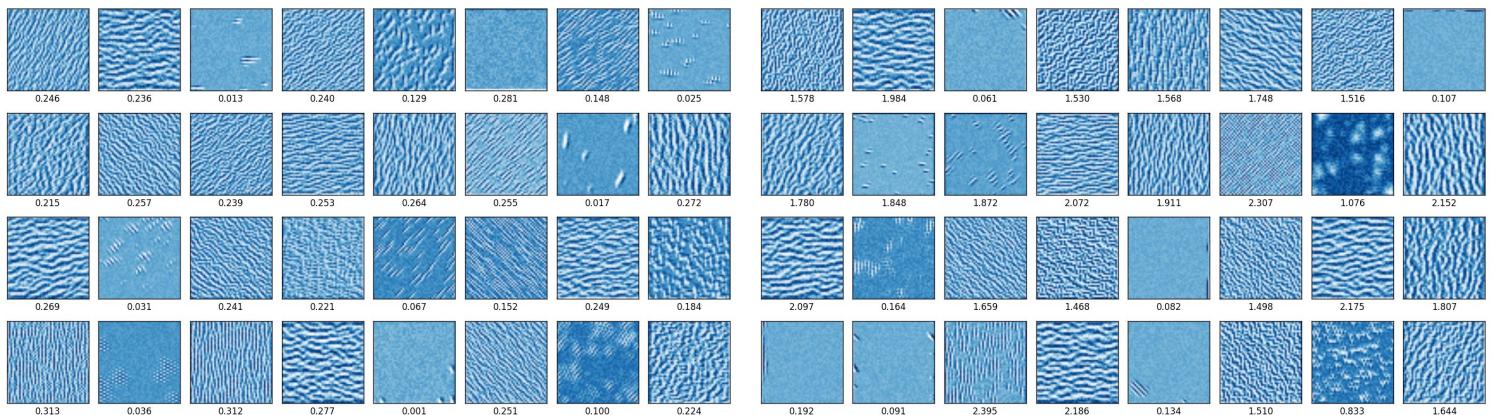
[Bonus] (1%) 在 Problem 5 中, 提供了 3 個 hint, 可以嘗試實作及觀察 (但也可以不限於 hint 所提到的方向, 也可以自己去研究更多關於 CNN 細節的資料), 並說明你做了些什麼？

答：

1. 在第五題中, 我使用的 model 的 valid accuracy = 0.71967, 所以這邊我挑了另一個 performance 比較差的 model, valid accuracy = 0.63833 的來做比較。

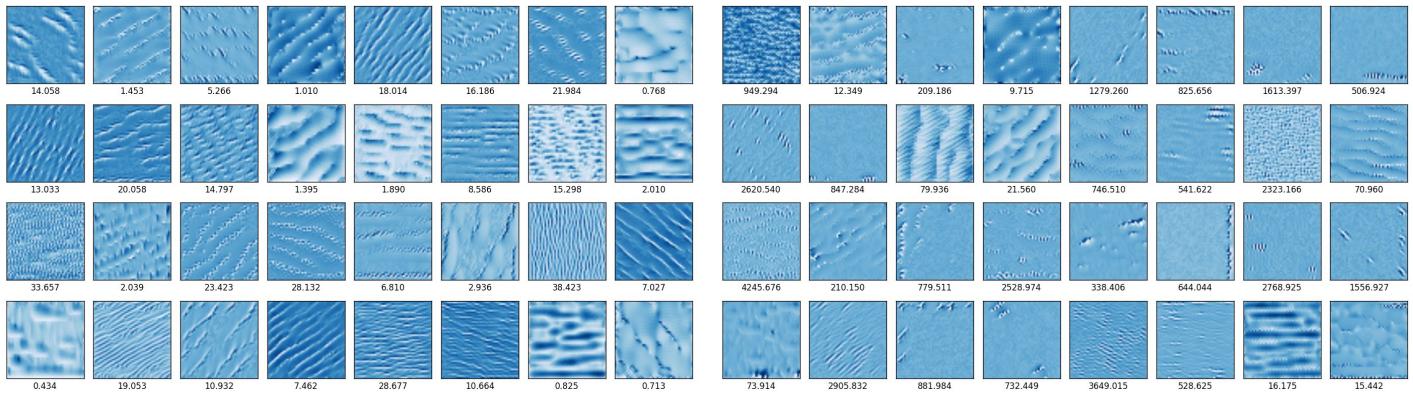
(第一層) [valid accuracy = 0.63833]

[valid accuracy = 0.71967]



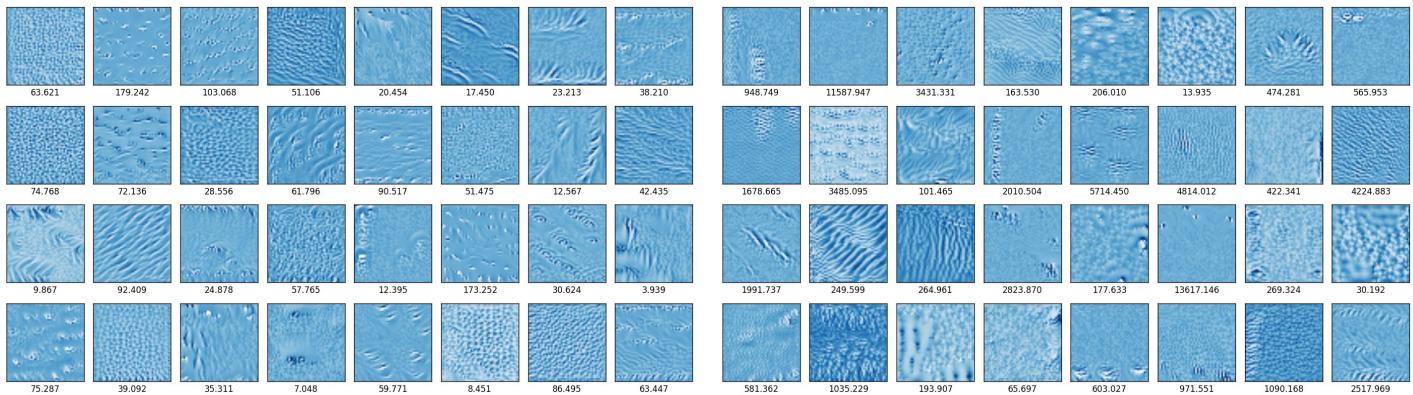
(第二層) [valid accuracy = 0.63833]

[valid accuracy = 0.71967]



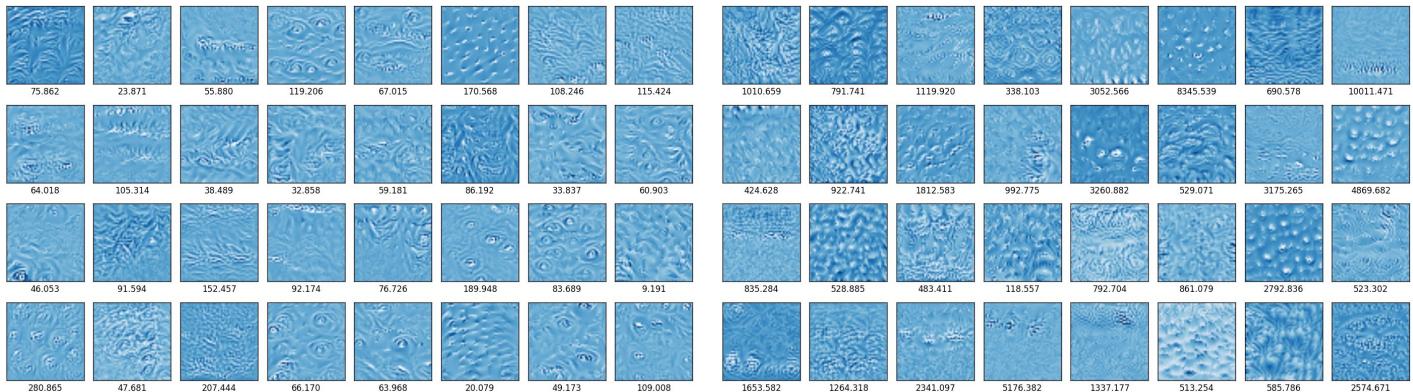
(第三層) [valid accuracy = 0.63833]

[valid accuracy = 0.71967]



(第四層) [valid accuracy = 0.63833]

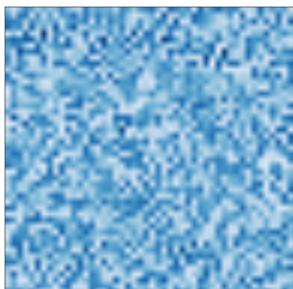
[valid accuracy = 0.71967]



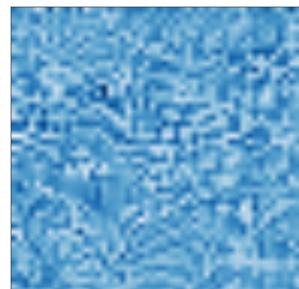
結果意外發現 performance 比較低的 model 畫出來的圖比平滑，感覺能使這些 filters 被 activation 最顯著的圖的 texture 比較清晰。

2. 使用跟第五題差不多的 code, 但把 target 設置成 class 的編號, 而不是 filter 的編號。

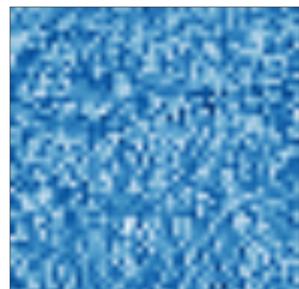
[Angry]



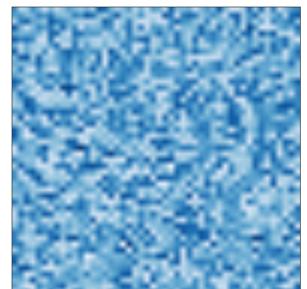
[Disgust]



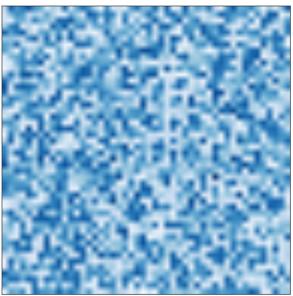
[Fear]



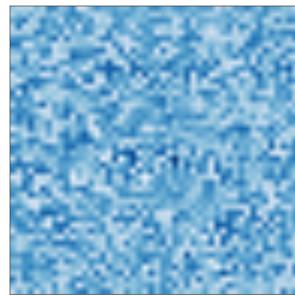
[Happy]



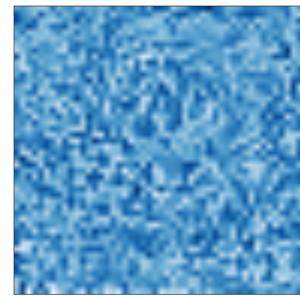
[Sad]



[Surprise]



[Neutral]



用人眼看起來會是一團亂，但查了一下資料，發現可以用 median filter 來修復圖片，所以在 gradient ascent 中嘗試看看。但微調許多次參數後，發現並沒有適當改善，把混亂的點透過 filter 糊開。不過根據理論上，median filter 的效果就是抓一塊區域，然後用附近的點來估計壞掉的點，所以本身很適合用來修復 corrupted pixel

3. 我取了一張在 validation set 中 emotion 是「快樂」的照片，然後強制讓他辨識成「生氣」。結果發現 model 的確如我在第三題答題中猜測的一樣，他會 focus 在「眼睛、臉頰跟嘴巴」的部分，並以之作為標準做修改。



Angry