

ML2017 HW3 Report

系級：生技三 學號：b03b02014 姓名：張皓鈞

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？

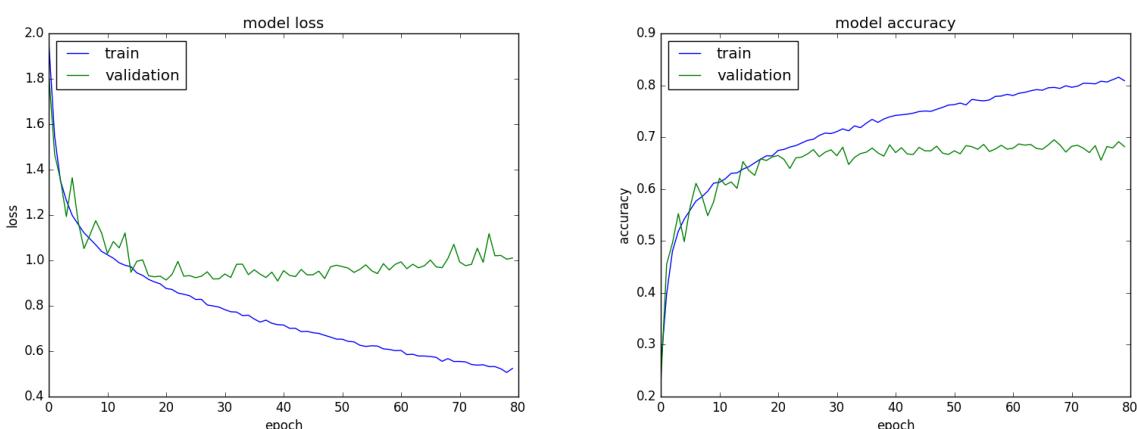
- CNN 模型架構：

主要構成為三組 Convolutional layer x2, Max pooling layer, Dropout(0.25)，在通過每層 Convolutional layer 之前的輸入特徵都先用 0 擴增特徵圖的邊界，通過 layer 之後都經過 LeakyReLU ($\alpha = 0.001$) 的運算，並且經過 Batch Normalization。在 Dense 部份，則是用兩層 128 個 neuron 的 layer，activation function 一樣是 LeakyReLU ($\alpha = 0.001$)，Dropout 一樣是用機率 =0.25，最後以 softmax 來輸出 7 種 class 的分數。

- 訓練過程：

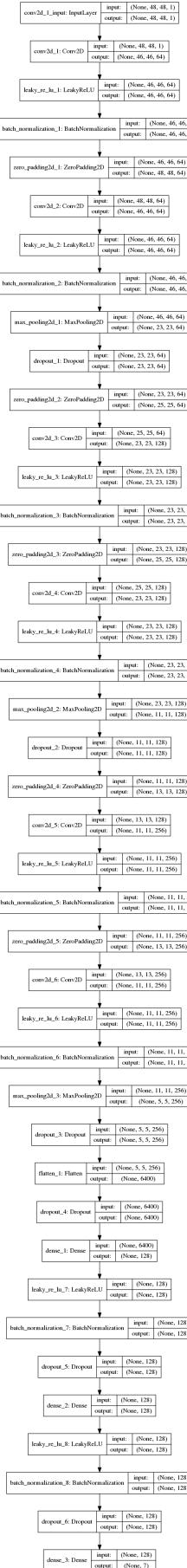
首先隨機取樣 3000 筆測資作為 validation data，剩餘測資透過 Keras 內建的 data generator，隨機將剩餘測資進行水平翻轉、垂直和水平移動 0.1 倍的長度距離。每次訓練前皆隨機變更測資的順序，每次以 64 筆測資作為一個 batch。我用 validation 準確度作為標準，每個 epoch 訓練速度約為 600 秒，從 80 個 epoch 當中選出最高準確度的模型，發現此模型是訓練了 67 個 epoch 的模型，在 Kaggle 的 public dataset 上面分類準確度為 0.69407。(註：使用 data generator 使 CNN 的預測準確度從 0.60 左右上升至 0.69)

- Loss and Accuracy Plot



CNN Model Architecture Detail

Layer (type)	Output Shape	Parameters
Conv2D 1	(None, 46, 46, 64)	640
Batch normalization 1	(None, 46, 46, 64)	256
Conv2D 2	(None, 46, 46, 64)	36928
Batch normalization 2	(None, 46, 46, 64)	256
Max pooling 1	(None, 23, 23, 64)	0
Dropout 1	(None, 23, 23, 64)	0
Conv2D 3	(None, 23, 23, 128)	73856
Batch normalization 3	(None, 23, 23, 128)	512
Conv2D 4	(None, 23, 23, 128)	147584
Batch normalization 4	(None, 23, 23, 128)	512
Max pooling 2	(None, 11, 11, 128)	0
Dropout 2	(None, 11, 11, 128)	0
Conv2D 5	(None, 11, 11, 256)	295168
Batch normalization 5	(None, 11, 11, 256)	1024
Conv2D 6	(None, 11, 11, 256)	590080
Batch normalization 6	(None, 11, 11, 256)	1024
Max pooling 3	(None, 5, 5, 256)	0
Dropout 3	(None, 5, 5, 256)	0
Flatten 1	(None, 6400)	0
Dropout 4	(None, 6400)	0
Dense 1	(None, 128)	819328
Batch normalization 7	(None, 128)	512
Dropout 5	(None, 128)	0
Dense 2	(None, 128)	16512
Batch normalization 8	(None, 128)	512
Dropout 6	(None, 128)	0
Dense 3	(None, 7)	903
Total params: 1,985,607.0		
Trainable params: 1,983,303.0		
Non-trainable params: 2,304.0		



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？

- DNN 模型架構：

如圖，主要構成為 3 層 512、2 層 256 和 4 層 128 的 layer，每層的 activation function 一樣是 LeakyReLU ($\alpha = 0.001$)，Dropout 一樣是用機率 =0.25，且都有 Batch normalization。最後以 softmax 來輸出 7 種 class 的分數。

- 訓練過程：

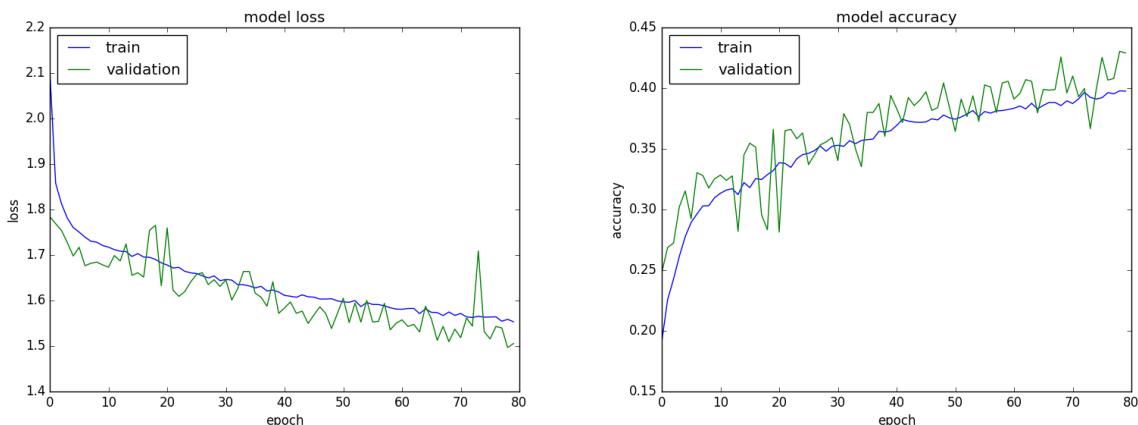
與 CNN 一樣使用 validation 準確度作為標準，每個 epoch 訓練的速度約在 20 秒左右，從 80 個 epoch 當中選出最高準確度的模型，發現此模型是訓練了 67 個 epoch 的模型，在 Kaggle 的 public dataset 上面分類準確度為 0.41794。

- 觀察：

在相近的參數數目和層數下，經過相同的訓練過程和訓練次數，可以發現 CNN 的準確度比 DNN 還要高很多，符合我們對 CNN 是一種比較具有抽取影像特徵能力之 Neural Network 的了解。

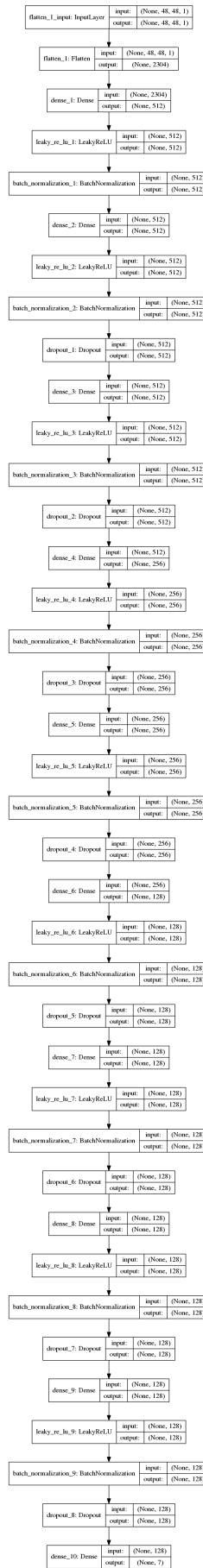
其中，epoch 的訓練速度上，CNN 的速度比 DNN 要慢的許多，原因可能包含 CNN 可以看作是某一 DNN 的參數區域化的版本，所以在此 CNN 本身是一個更多參數的 DNN 的變形，因此跟我們的 DNN 相比之下，訓練速度會慢許多。

- Loss and Accuracy Plot



DNN Model Architecture Detail

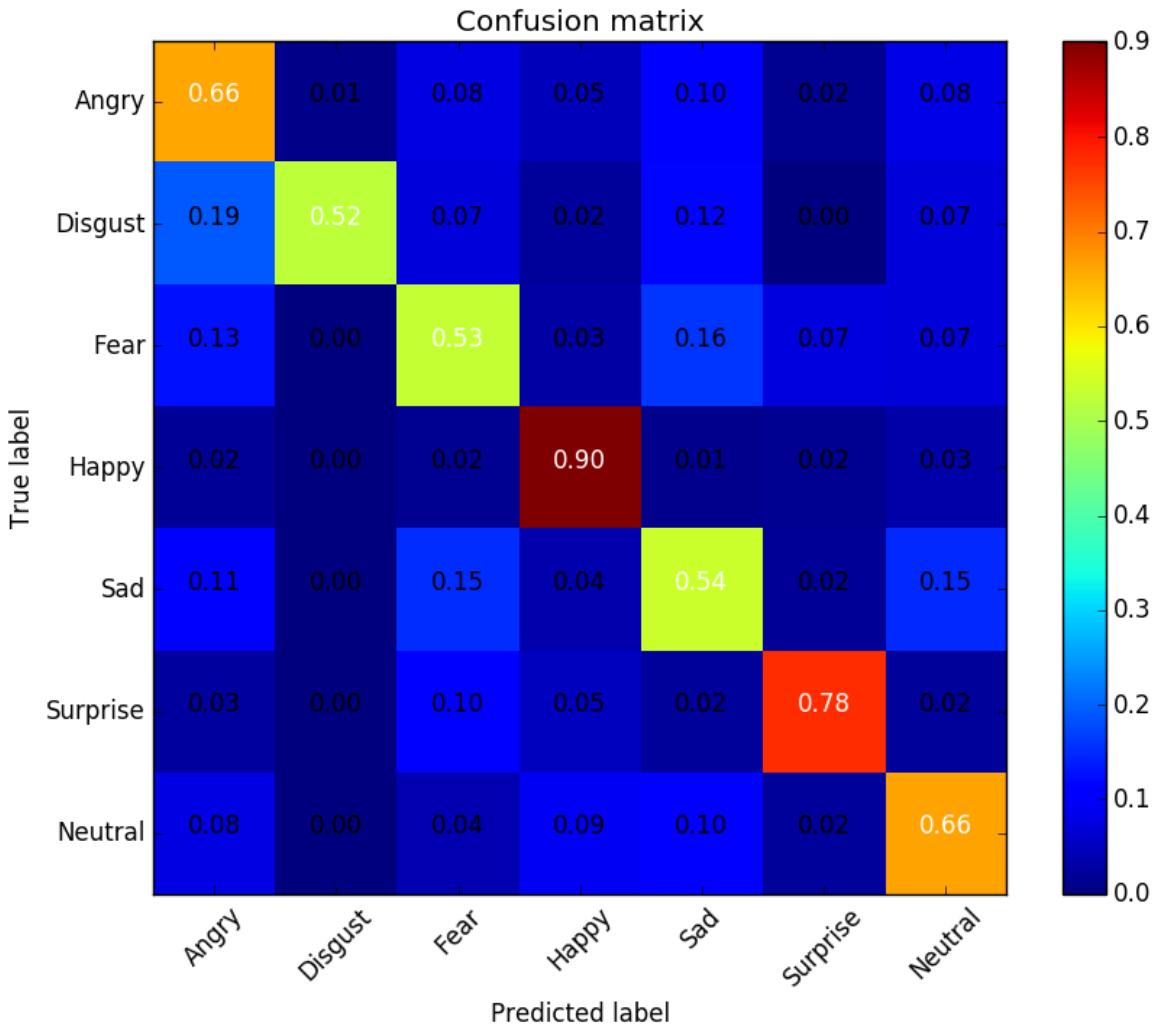
Layer (type)	Output Shape	Parameters
Flatten 1	(None, 2304)	0
Dense 1	(None, 512)	1180160
Batch normalization 1	(None, 512)	2048
Dense 2	(None, 512)	262656
Batch normalization 2	(None, 512)	2048
Dropout 1	(None, 512)	0
Dense 3	(None, 512)	262656
Batch normalization 3	(None, 512)	2048
Dropout 2	(None, 512)	0
Dense 4	(None, 256)	131328
Batch normalization 4	(None, 256)	1024
Dropout 3	(None, 256)	0
Dense 5	(None, 256)	65792
Batch normalization 5	(None, 256)	1024
Dropout 4	(None, 256)	0
Dense 6	(None, 128)	32896
Batch normalization 6	(None, 128)	512
Dropout 5	(None, 128)	0
Dense 7	(None, 128)	16512
Batch normalization 7	(None, 128)	512
Dropout 6	(None, 128)	0
Dense 8	(None, 128)	16512
Batch normalization 8	(None, 128)	512
Dropout 7	(None, 128)	0
Dense 9	(None, 128)	16512
Batch normalization 9	(None, 128)	512
Dropout 8	(None, 128)	0
Dense 10	(None, 7)	903
Total params: 1,996,167.0		
Trainable params: 1,991,047.0		
Non-trainable params: 5,120.0		



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]

- Confusion Matrix :

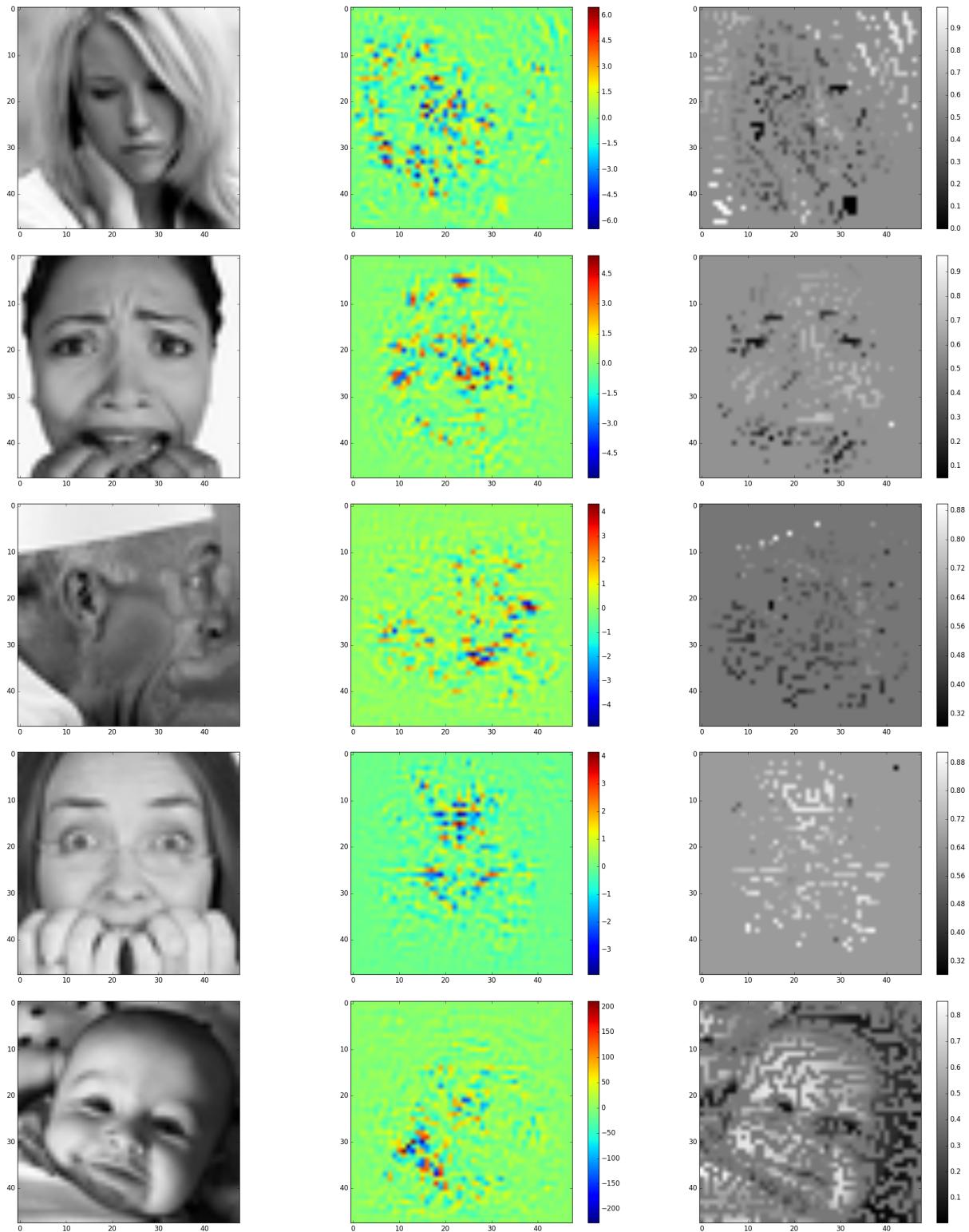
此 Matrix 是由(1)所隨機取樣的 3000 筆 validation data 所計算出來的。



- 觀察：

- 從 confusion matrix 的分布可以發現此模型在每種情緒上都有 0.5 以上的比例分類正確，尤其在快樂的部份有最高的準確度，高達 0.9; 第 2 高的則是驚喜的情緒，有 0.78; 但是，在噁心、恐懼和難過這 3 個情緒分類上準確度就比較低，都只有稍微大於 0.5 而已。
- 在各種情緒當中，其中噁心容易和生氣、難過搞混(比例分別是 0.19 和 0.12); 恐懼也是容易和生氣、難過搞混(比例為 0.13 和 0.16); 而難過的情緒則是容易和生氣、恐懼還有中立搞混(比例分別是 0.11、0.15 和 0.15)。
- 有趣的是，從第 2 個直排可以看到，在各種情緒下，分類器最不會將情緒分類到'噁心'。(只有生氣會有 0.01 的比例被分到噁心)
- 另外，從第 6 個直排也發現，我們所訓練出的分類器也比較不會將情緒分類到'驚喜'。(其他種情緒分類到驚喜的比例都在 0.02 以下，只有恐懼有 0.07 的比例分類到驚喜)

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？



- 觀察：

- (a) 從 3000 筆 validation data 中，取出編號 0 到 4 的訓練資料，分別繪出影響 CNN 分類比較大的 heatmap 以及把對 CNN 影響不大的部份 Mask 掉。從中發現 CNN 主要 focus 在人臉的嘴巴、眼睛部份來分類情緒，例如第 3 張的側臉圖也是專注在右邊人臉正面的方向。
- (b) 第 2、4 張中因為嘴巴部份有被手擋住，而 heatmap 上嘴巴部份皆沒有太大的影響，在 masked 的部份嘴巴也不太明顯，因此推測這可能是恐懼、難過、噁心分類準確度較低的原因。

5. (1%) 承 (1)(2)，利用上課所提到的 gradient ascent，觀察特定層的 filter 最容易被哪種圖片 activate。

(a) 能最容易 activate filters 的影像：

- 說明：

這些影像是利用 gradient ascent 的方式，從隨機 white noise 開始，經過 100 個 epoch 所產生的影像，每次更新影像的 step 為 $0.01 * \text{gradient}$ ，各影像為最容易 activate 該 filter 的影像。其中各影像下排的數字為最後產生的 loss。

- 觀察：

我取第 1 和第 2 層 Convolutional layer 的 filter 來進行分析，發現第 1 層所產生出來的影像大都是點霧狀的影像，看不出什麼特別的圖案，原因可能是因為 CNN 的第 1 層 filter 主要會先偵測比較 general 的特徵。

而第 2 層就有明顯的紋路圖案出現，各個 filter 都有不同的紋路，表示此層的 filter 發展出比較能偵測特定紋路的能力。符合我們對 CNN 可以”自動進行 feature transformation”的預期。

(b) Layer Output Image

- 說明：

這些圖片是從 Validation data 中隨機選取一張圖片，經過 CNN model 的計算之後，特定 filter 所 output 出來的影像。因為從(a)中的影像當中發現第 2 層開始有明顯紋路，而預期上第 3 層應該會輸出更複雜、更接近分類時的特徵，因此我選取了第 2 層和第 3 層的 Convolutional layer 的 output 影像作為分析對象。

- 觀察：

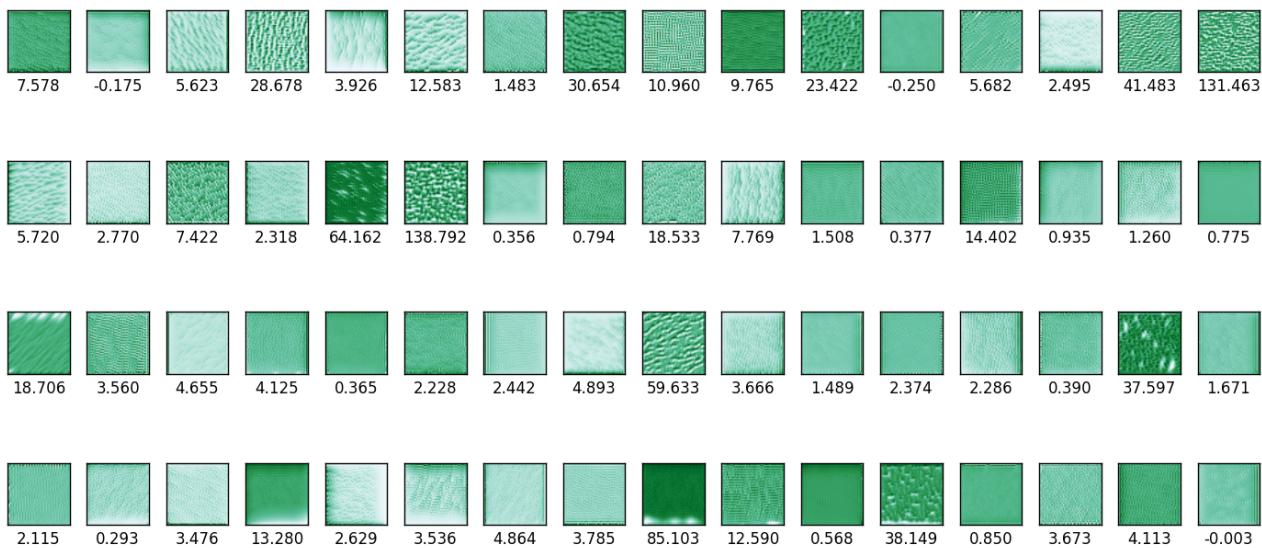
從第 2 層的輸出影像可以發現各個 filter 輸出的影像已經開始著重在人臉的眼睛和嘴巴部份，當然還是有一些 filter 的輸出是不同質地的人臉或是人臉的輪廓。

到了第 3 層，各個 filter 的紋路皆有些許不同，但是各個 filter 的輸出主要都著重在眼睛嘴巴部份，而且輪廓更深，有些的眼睛連成一橫線或是成為明顯的兩個點。由這裡更加確定，訓練出來的 CNN 是逐漸從人臉的特徵中抽取眼睛嘴巴的部份來進行分類。

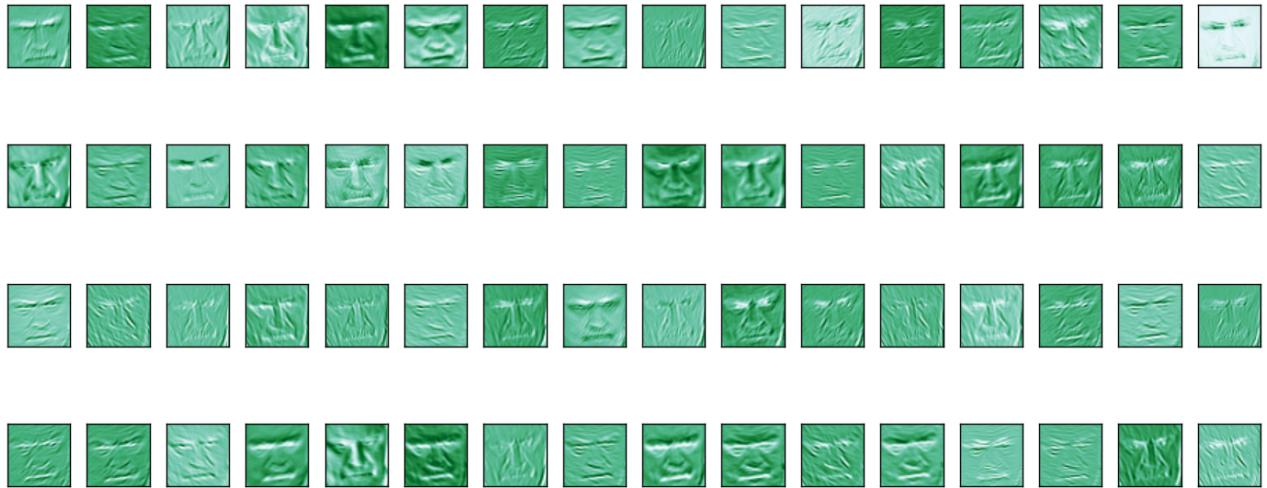
Filters of layer conv2d_1 (# Ascent Epoch 90)



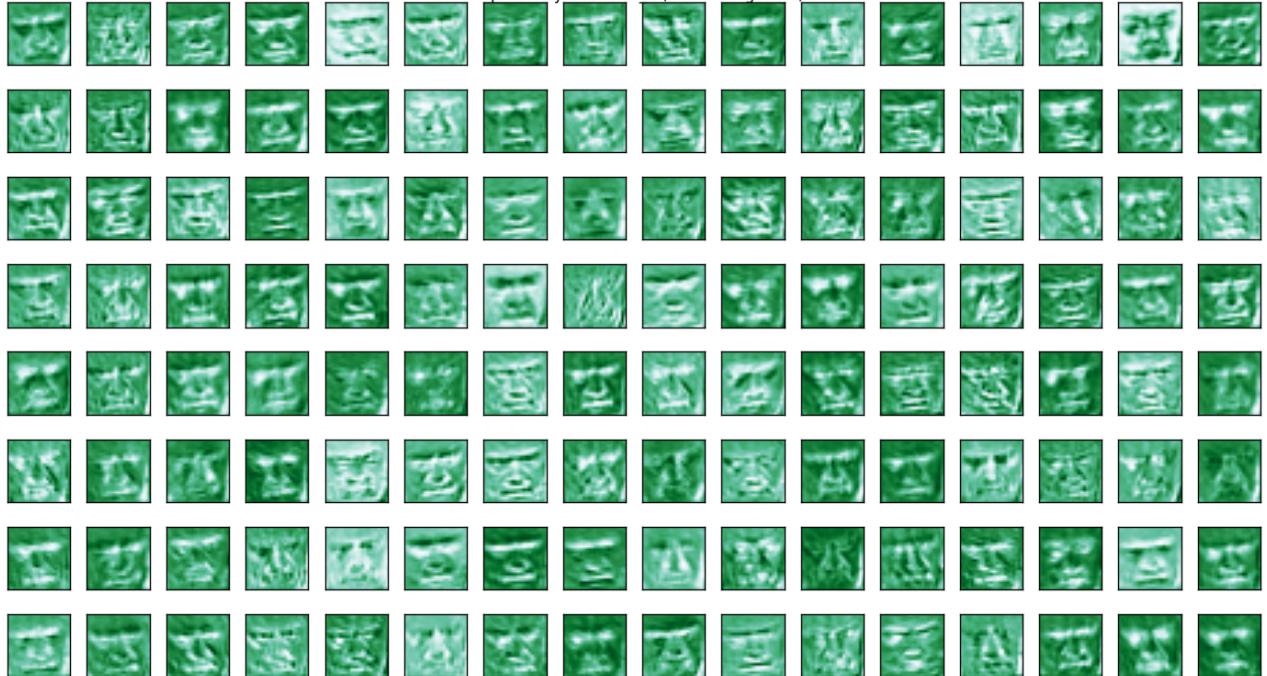
Filters of layer conv2d_2 (# Ascent Epoch 90)



Output of layer conv2d_2 (Given image 666)



Output of layer conv2d_3 (Given image 666)

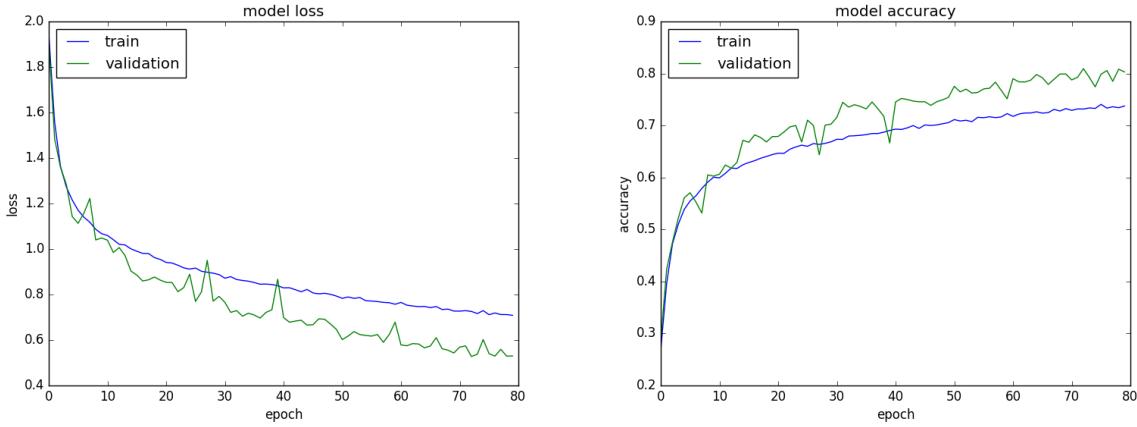


Bonus 1 (1%) 從 training data 中移除部份 label，實做 semi-supervised learning

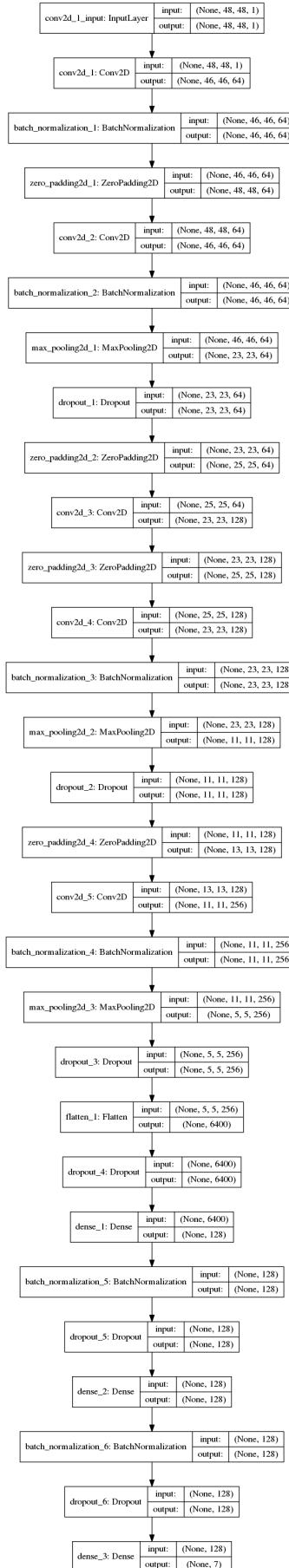
- 實作方式：

我實作的是 self-training 的 semi-supervised learning。首先去除掉 label 的 data 比例為 10% 的訓練資料，並且用(1)所提及的 CNN 模型預測這些 unlabeled data，最後再將這些加上 label 的 data 加入其他訓練資料之後，再另外訓練另一個 CNN 模型(詳細模型架構如下)，其詳細架構如附圖，訓練過程和(1)所提及的 CNN 模型一樣。在 Kaggle public data 上的預測準確度為 0.67902。

- Loss and Accuracy Plot



Layer (type)	Output Shape	Parameters
Conv2D 1	(None, 46, 46, 64)	640
Batch normalization 1	(None, 46, 46, 64)	256
Conv2D 2	(None, 46, 46, 64)	36928
Batch normalization 2	(None, 46, 46, 64)	256
Max pooling 1	(None, 23, 23, 64)	0
Dropout 1	(None, 23, 23, 64)	0
Conv2D 3	(None, 23, 23, 128)	73856
Conv2D 4	(None, 23, 23, 128)	147584
Batch normalization 3	(None, 23, 23, 128)	512
Max pooling 2	(None, 11, 11, 128)	0
Dropout 2	(None, 11, 11, 128)	0
Conv2D 5	(None, 11, 11, 256)	295168
Batch normalization 4	(None, 11, 11, 256)	1024
Max pooling 3	(None, 5, 5, 256)	0
Dropout 3	(None, 5, 5, 256)	0
Flatten 1	(None, 6400)	0
Dropout 4	(None, 6400)	0
Dense 1	(None, 128)	819328
Batch normalization 5	(None, 128)	512
Dropout 5	(None, 128)	0
Dense 2	(None, 128)	16512
Batch normalization 6	(None, 128)	512
Dropout 6	(None, 128)	0
Dense 3	(None, 7)	903
Total params: 1,393,991.0		
Trainable params: 1,392,455.0		
Non-trainable params: 1,536.0		

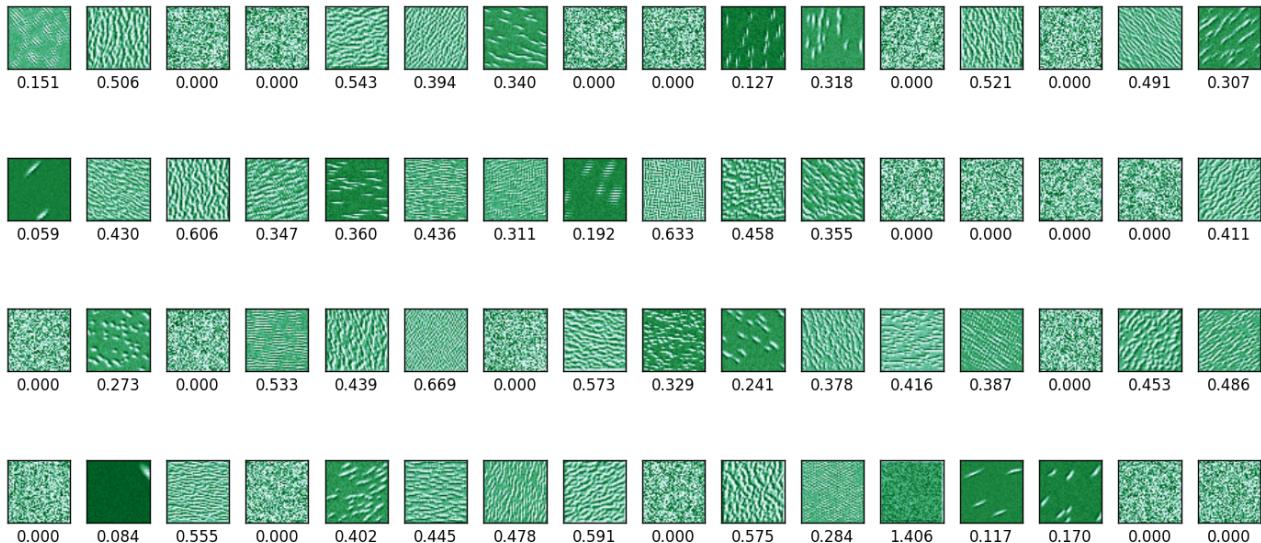


Bonus 2 (1%) 在 Problem 5 中，提供了 3 個 hint，可以嘗試實作及觀察 (但也可以不限於 hint 所提到的方向，也可以自己去研究更多關於 CNN 細節的資料)，並說明你做了些什麼？[完成 1 個: +0.4%，完成 2 個: +0.7%，完成 3 個: +1%]

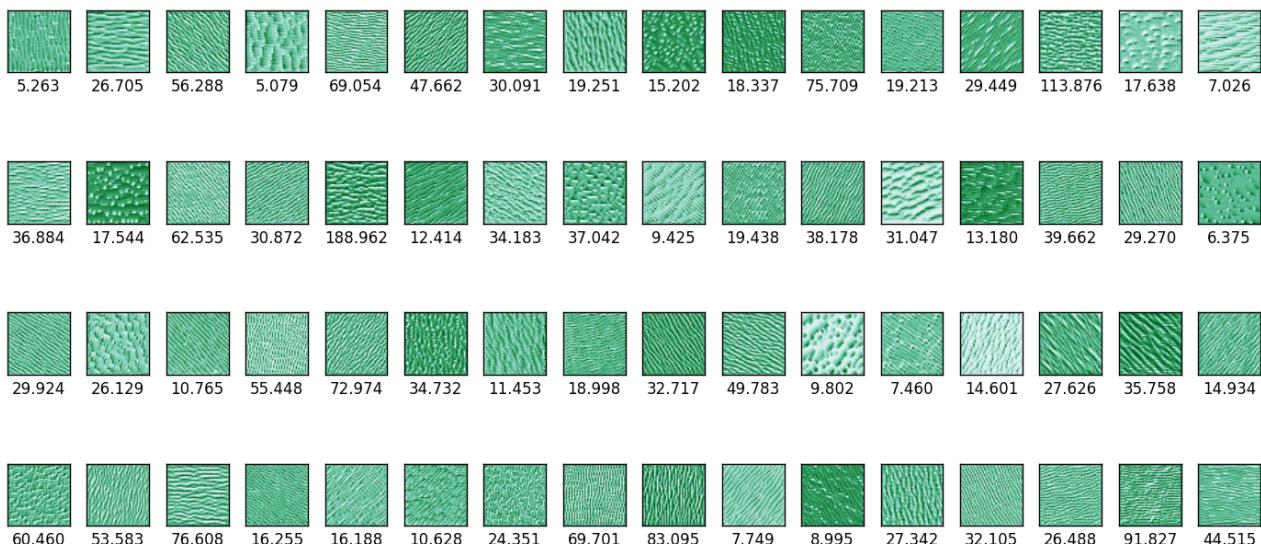
1. Filters of Semi-supervised CNN:

(a) 能最容易 activate filters 的影像：

Filters of layer conv2d_1 (# Ascent Epoch 90)



Filters of layer conv2d_2 (# Ascent Epoch 90)



- 說明：

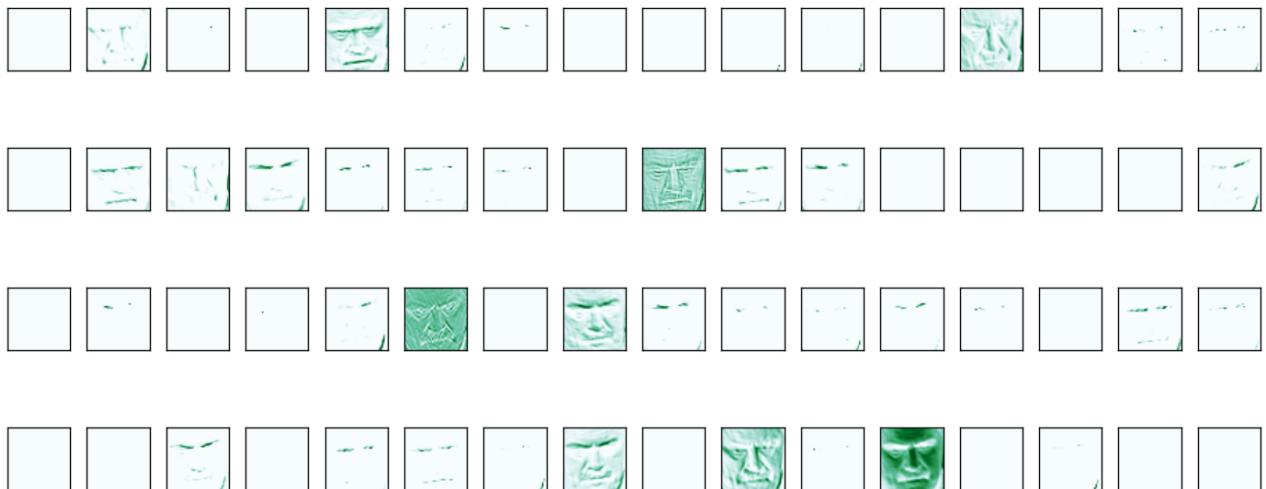
這些影像是利用 gradient ascent 的方式，從 white noise 開始，經過 100 個 epoch 所產生的影像，各影像為最容易 activate 該 filter 的影像，所使用的 model 是 Bonus 1 當中所實作的 semi-supervised CNN。其中各影像下排的數字為最後產生的 loss。我選取第 1 和第 2 層的 Convolutional layer 作為分析對象。

- 觀察：

兩層的影像皆是各種不同的紋路，顯示從第一層開始 CNN 就在偵測輸入影像的特定紋路，而每個 filter 都各有偵測特定紋路的能力，但是卻看不出任何人臉的輪廓。

(b) Layer Output Image

Output of layer conv2d_1 (Given image 666)



Output of layer conv2d_2 (Given image 666)



- 說明：

以上這些圖片是從 Validation data 中隨機選取一張圖片，經過(Bonus 1)的 Semi-supervised CNN 的計算之後，特定 filter 所 output 出來的影像。其中我選取了第 1 層和第 2 層的 Convolutional layer 的 output 影像作為分析對象。

- 觀察：

與(1)所提到的 CNN 不同，第 2 層的 filter 輸出影像大部份都是呈現白色，雖然依稀可以看出人臉輪廓，也是著重在眼睛和嘴巴部份，但是顏色方面卻是和前面的 CNN 相反。