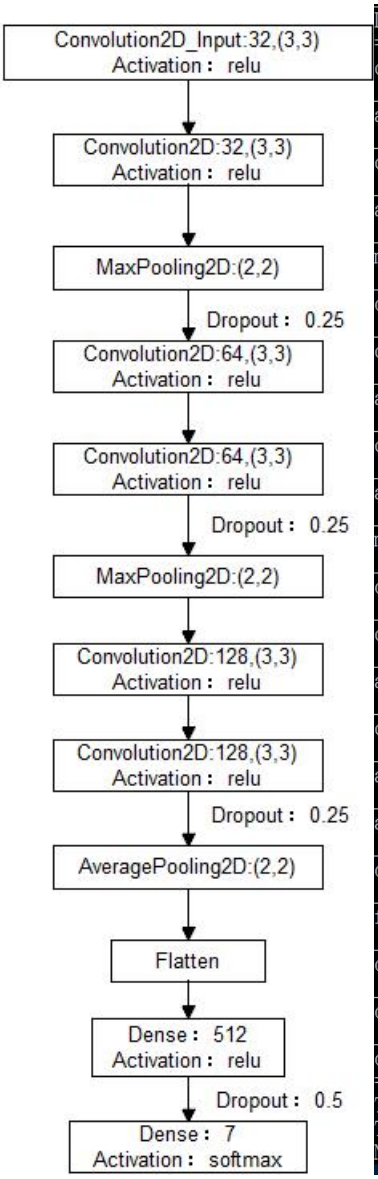
學號：R05944043 系級： 網媒碩一 姓名：宋焱檳

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：使用了 6 層 Conv2d，2 層 MaxPooling，1 層 AveragePooling，2 層 FC 層，對 input image 做了一次反轉，使得 data 數量擴大了 1 倍，在 predict 的時候，也是結合反轉前後 image 通過 model 的概率和，選擇最大概率作為 test_data 的 label。使用了 4 個 Dropout 來防止 overfitting。訓練了 200epoch。

模型架構如下：

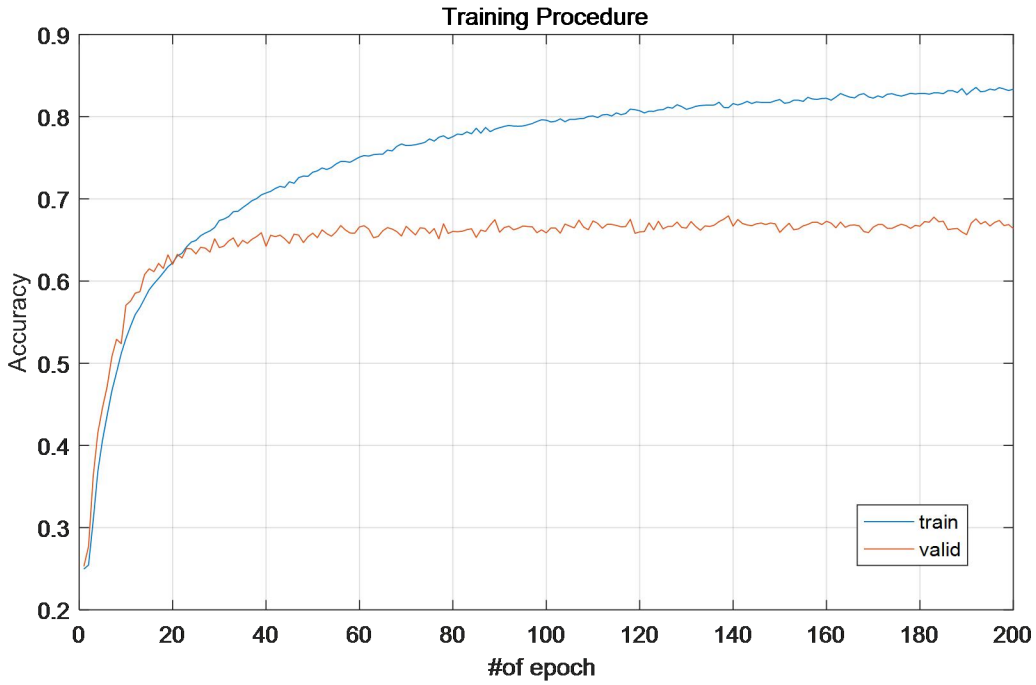


Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 46, 46, 32)	320
activation_1 (Activation)	(None, 46, 46, 32)	0
conv2d_2 (Conv2D)	(None, 44, 44, 32)	9248
activation_2 (Activation)	(None, 44, 44, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 22, 22, 32)	0
dropout_1 (Dropout)	(None, 22, 22, 32)	0
conv2d_3 (Conv2D)	(None, 20, 20, 64)	18496
activation_3 (Activation)	(None, 20, 20, 64)	0
conv2d_4 (Conv2D)	(None, 18, 18, 64)	36928
activation_4 (Activation)	(None, 18, 18, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 9, 9, 64)	0
dropout_2 (Dropout)	(None, 9, 9, 64)	0
conv2d_5 (Conv2D)	(None, 7, 7, 128)	73856
activation_5 (Activation)	(None, 7, 7, 128)	0
conv2d_6 (Conv2D)	(None, 5, 5, 128)	147584
activation_6 (Activation)	(None, 5, 5, 128)	0
average_pooling2d_1 (Average	(None, 2, 2, 128)	0
dropout_3 (Dropout)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_4 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 7)	3591

Total params: 552,679
Trainable params: 552,679
Non-trainable params: 0

共使用 552,679 個參數

訓練過程如下（使用 0.1 的 training data 作為 validation）：

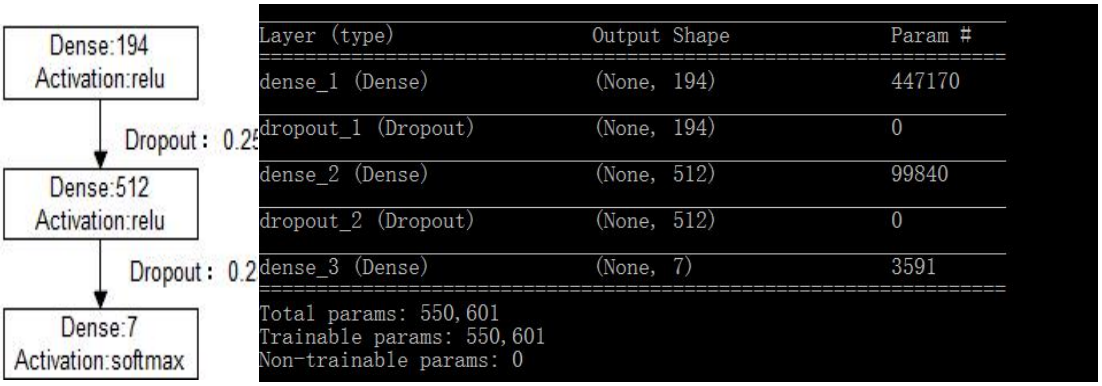


在 40 組 epoch 後幾乎就不發生改變了。Train 的準確率一直在增加，最後在 0.83 左右，而 validation 因為防止了 overfitting 的緣故，從而不會變化太大。最後 train 的 loss 為 0.458，還在減小，validation 的 loss 維持在 1.0 附近，說明這個 model 還沒有太優秀，無法把 train 上的 loss 再調小一點。

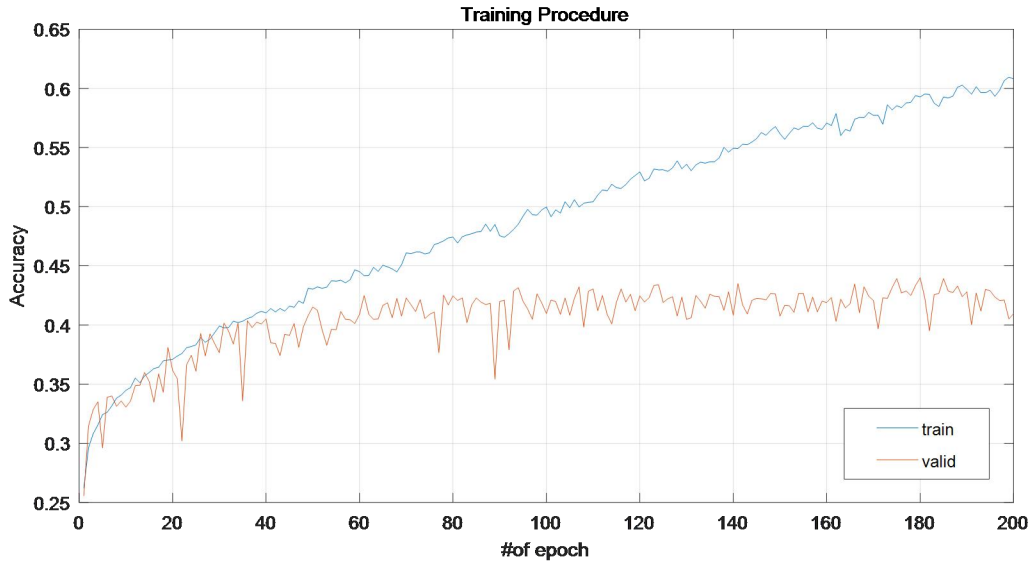
選擇了其中 validation 正確率 0.67 的 model 對 test 進行預測。最後在 public 上的準確率有 0.651，略微超過 strong line。如果再多對 input 做一些轉換，可能效果會更好。但是由於對 keras 不是很熟練，imagegenerate 方法調用失敗，只能自己手刻，所以花費了蠻多的時間。

2. 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：使用了 3 層 FC 層，和 2 個 dropout 防止 overfitting。訓練了 200epoch。主要架構如下：



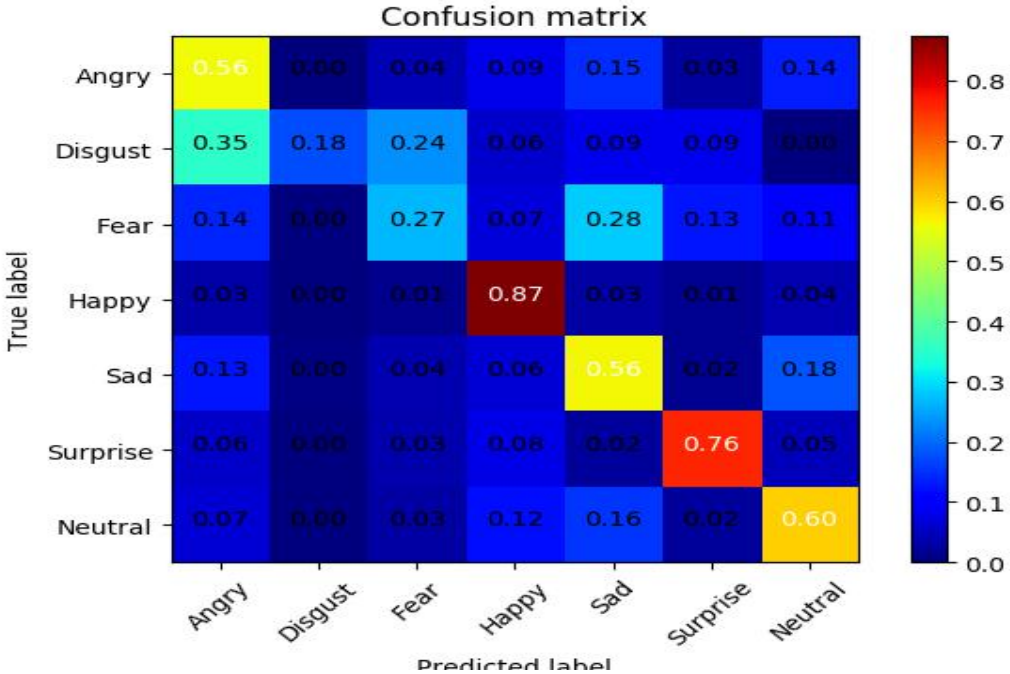
使用的參數數量和 CNN 的類似，訓練過程如下（同樣用 10%做 validation）：



大概 validation 在 60 組後在 0.45 附近來回，training data 一直在增加。經過 200 組 epoch 後在 0.62. Training data 的 loss 比較大，在 1.0 附近，validation 的 loss 在 1.5，都比較大，說明可能模型構建的不是太好。

可以看出，在 CNN 和 DNN 參數差不多的情況下，CNN 的效果比較好，不過，這也有可能是因為對於 DNN 我使用的層數不夠深而導致的。比較上面的 CNN 也可以看出，loss 下降到最後，區間不一樣。DNN 的波動比 CNN 厲害，在表情的識別方面，可能 CNN 的結果會相對好一些。

3. 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析]
把作為 validation 的 data 匯入 confusion matrix，結果圖如下：

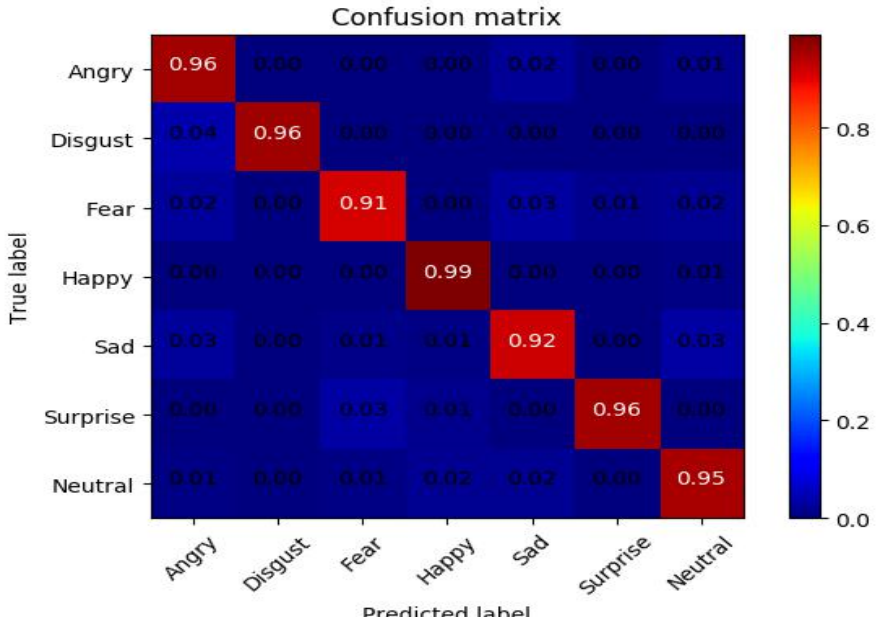


由圖可以觀察出，對角線上的的概率為判斷表情正確的率。如果不在對角線上，就可以依據同一行判斷是誤判成什麼表情，觀察概率是多少。可以發現有幾個表情特別容易混淆起來。如下：

Right	Mistake（rate）	Right	Mistake（rate）
Angry	Sad 15%	Fear	Angry 14%
Angry	Neutral 14%	Disgust	Fear 24%
Disgust	Angry 35%	Fear	Surprise 13%
Fear	Sad 28%	Sad	Neutral 18%
Sad	Angry 13%	Surprise	Happy 8%
Neutral	Sad 12%	Neutral	Happy 16%

綜合上表，可以發現，Disgust 很容易被誤判成 Angry。很可能因為在人的主觀上這兩種感情就很類似，所以做出的反應也比較相似，使得機器在訓練識別上出現了一定的問題。Fear 和 Disgust 是比較難判斷的兩個表情，他們的識別率都很低。Angry 是最容易被其他表情誤判成為的表情，最大的誤判為表中的 35%。辨識度最高的是 Happy 和 Surprise 的表情。

下圖是我使用了一部分的 training data 作為 confusion matrix 的 input 得到的結果：



同樣地，在之前誤差最大位置，還是存在著相對較大的 error rate。不過 disgust

的準確率還可以，可能是因為之後在這個表情的識別上 overfitting 的比較厲害吧。

4. 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：

以下三張圖是對於同一表情，CNN 的 filter 的結果圖。圖 1 是一張 angry 的表情。通過 classification 查看 sliency maps，得到的結果如圖 2，focus 在圖片的嘴巴，鼻子附近，眼睛和眼角，臉部中央，眉毛，以及一些在特定表情下容易產生皺紋的地方。圖三是通過 threshold 為 0.4 的 filter 提取出原圖 2 中的主要的 feature 點。整體看來人眼並不是一眼可以辨認這些 feature 是什麼樣的表情的。不過這些 feature 對於機器進行人臉表情的識別已經夠用了。

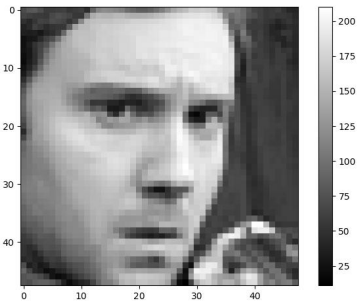


圖 1

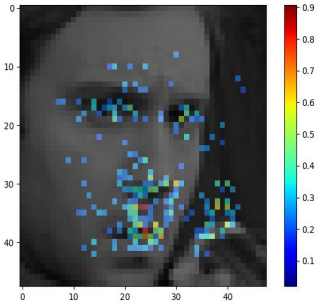


圖 2

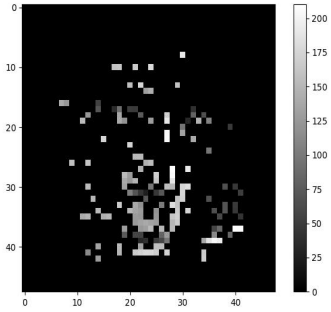
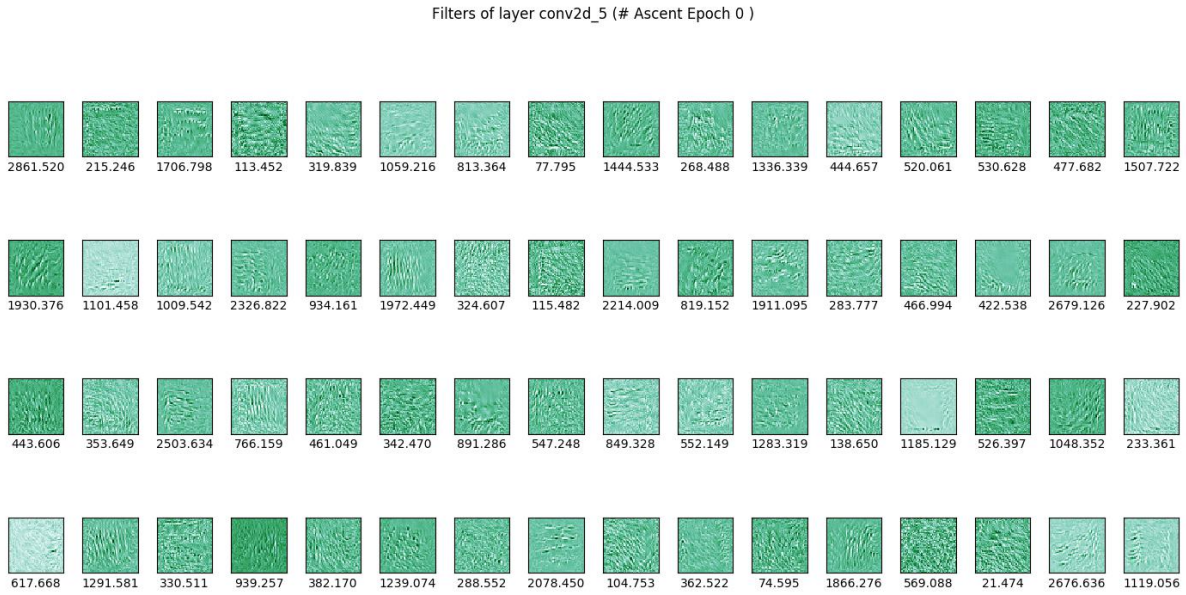


圖 3

5. 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：gradient ascent epoch=160，path=0.1

使用高斯白噪聲作為輸入，觀察第五層的 filter，結果如下：



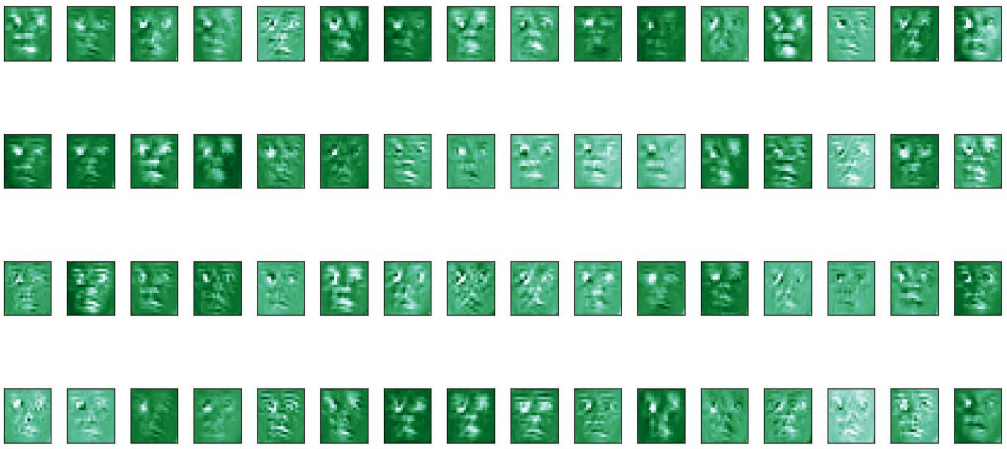
Activate Layer5 的圖片中，主要都是包含著一些各個方向的紋路，可能是偵測在面部附近肌肉或者皺紋的突起來判斷是否是某一類的 feature。還有一些是在偵測輪廓部分的 feature。可以看出，每個 filter 應該有不同的偵測 feature 的種類。使用真實的 data 進行。

先看第二層的結果，還是可以看出個輪廓的，通過不同的對比度抓取了人臉的 feature，說明不同的 filter 有不同的偵測的特徵。可以看出這個是一個 fear 的表情。偵測嘴巴和眼睛處 feature 的 filter 可能會被 activate。

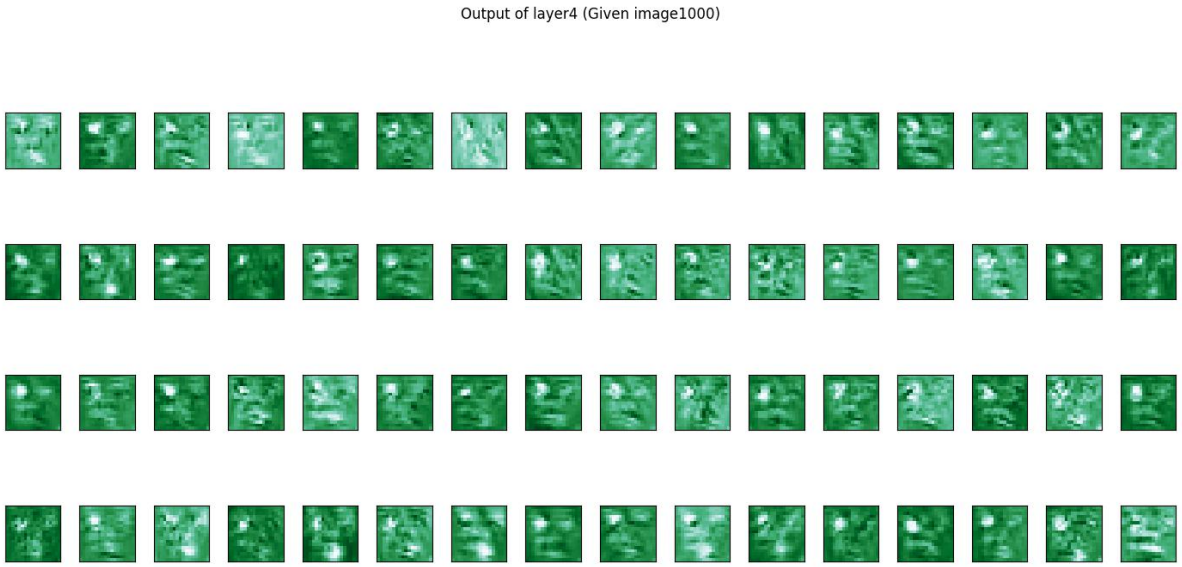
Output of layer2 (Given image1000)

第三層的結果，通過了一次的 Max Pooling，剩下了 pool 中的最大值，變得開始有些模糊，但是整體特徵變得清晰，眼睛嘴巴鼻子以及臉部朝向確定：

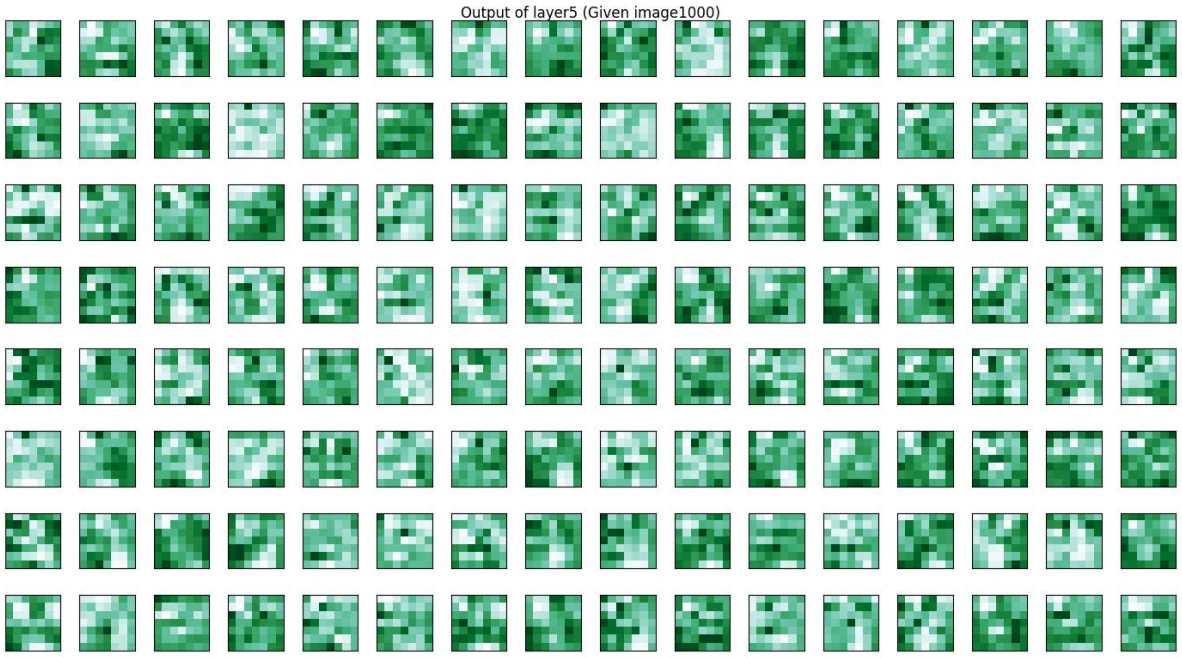
Output of layer3 (Given image1000)



第四層結果，和第三層類似，變得略微又模糊了一點，很多圖片上的臉消失了。



再次通過一個 Max Pooling 然後看第五層的結果圖如下：



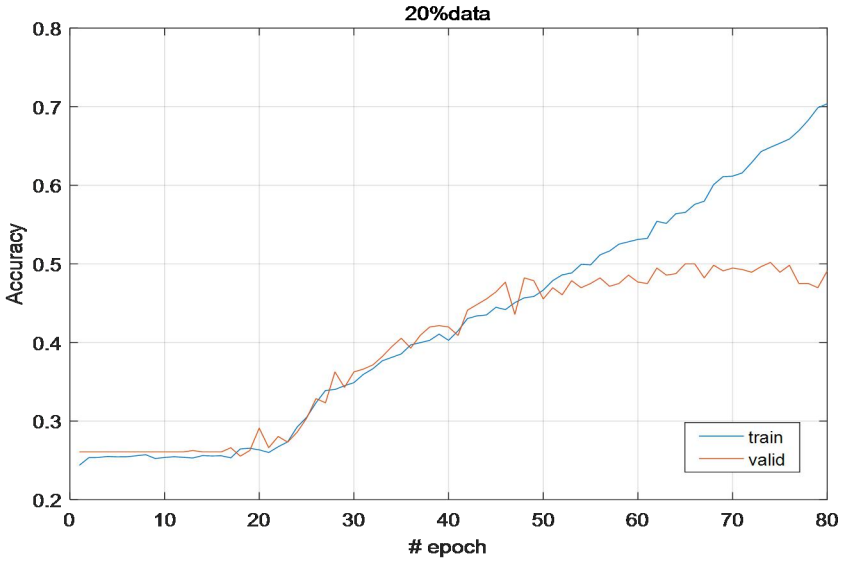
可以觀察出，由於通過了 2 個 Max Pooling,所以 pixel 數有點少，不過還是可以通過和第二層的對比大致上看出是很很多臉部的 feature 和整體臉的形狀的。不過似乎沒有抓的很准。所以，在新的沒見過的類型的數據集上，filter 提取特徵的效果，比在和訓練集分佈相同的數據集上，效果要差很多。不過比使用高斯白噪聲的時候結果還是好很多的。因此考慮說，對於一個分佈不完全一致的數據集，並不一定要從頭開始

訓練，而是可以利用在類似數據集上訓練好的模型參數，做初始化。因為很可能兩個不同 model 抓的 feature 很類似。這樣的預訓練，也許可以減少所需要的 label 的新數據的數目。

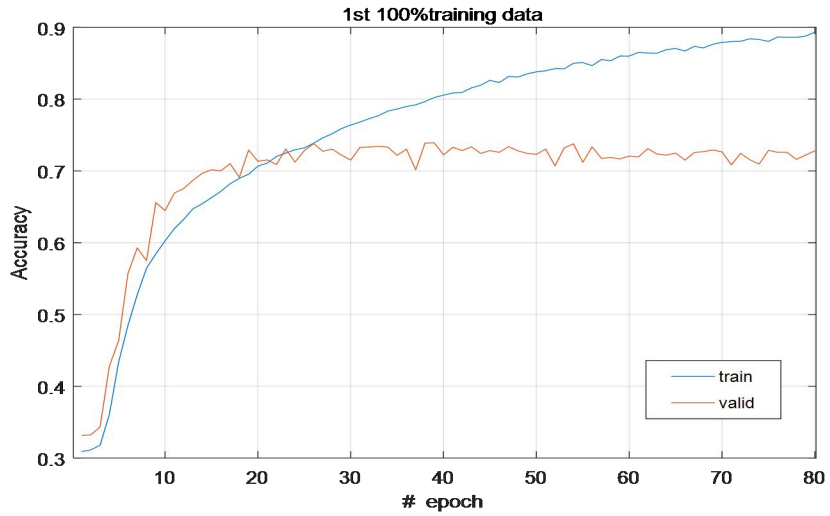
[Bonus] 從 training data 中移除部份 label，實做 semi-supervised learning

實做了 self-learning

一開始使用 20%的 training data train 出一個模型，準確率如下圖：

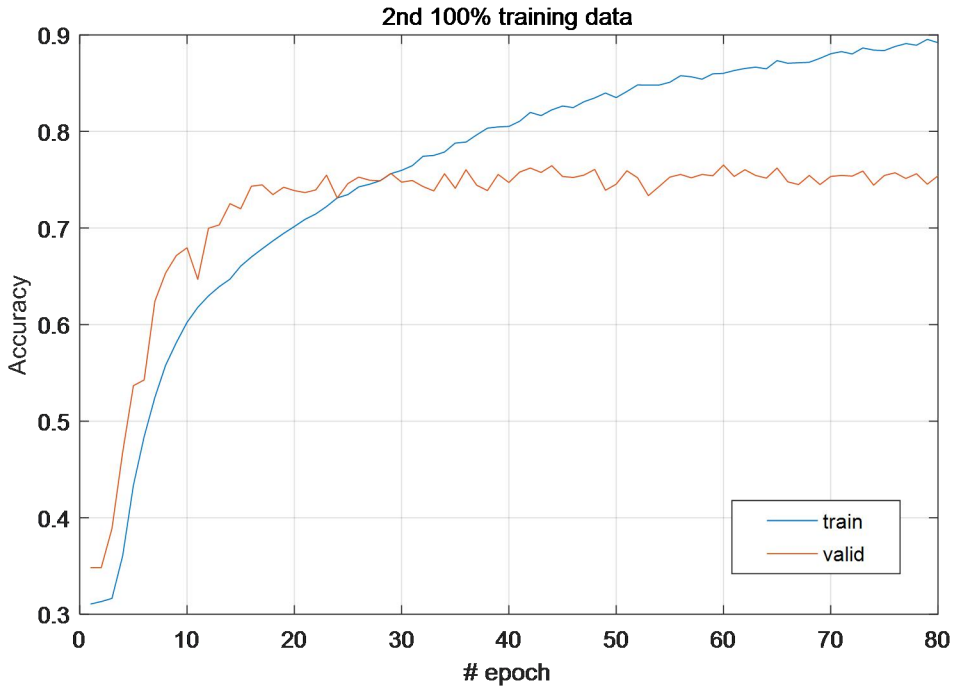


準確率都不是很高 0.5。然後，把剩下 80%的 training data 作為 unlabelled data，然後用之前 train 好的 model 進行 predict，得到所有 data 的 label 之後，回過頭去再用所有的自己製作的“training data”，train 出一個 model，訓練過程如下：



可以看出，因為 data 量的增加，所以準確率也得到了提升，validation 的準確

率有 0.73 左右。記錄對 testdata 的 predict。然後把得到的 model，對之前 80%的 data 再進行一次的 predict，刷新之前的 label，然後再次用 100%更新後的 training data 去 train 一個 model。訓練過程如下：



相比之前的 valid 上的準確率提高了 4%，training data 上的差不多。說明，更新 label 後，再次訓練對於 model 的性能有一定的提升，self-train 在 trainning data 的準確率上會有一定的提升。同樣，記錄 predict test data 的結果。然後如此循環。以下是之前三次 predcit 在 kaggle public 上的得分。

Model	Train accuracy	Valid accuracy	Public score
20%training data	70%	50%	0.1531
1st 100%training data	90%	73%	0.47311
2nd 100%training data	90%	77%	0.48342

所以總結來看，self-train 在 accuracy 上對整體是有提升的，但是第二次處理結束後，就提升的不是很明顯了。也可能問題出在我一開始拿出太多比例的 data 去做初始化了。