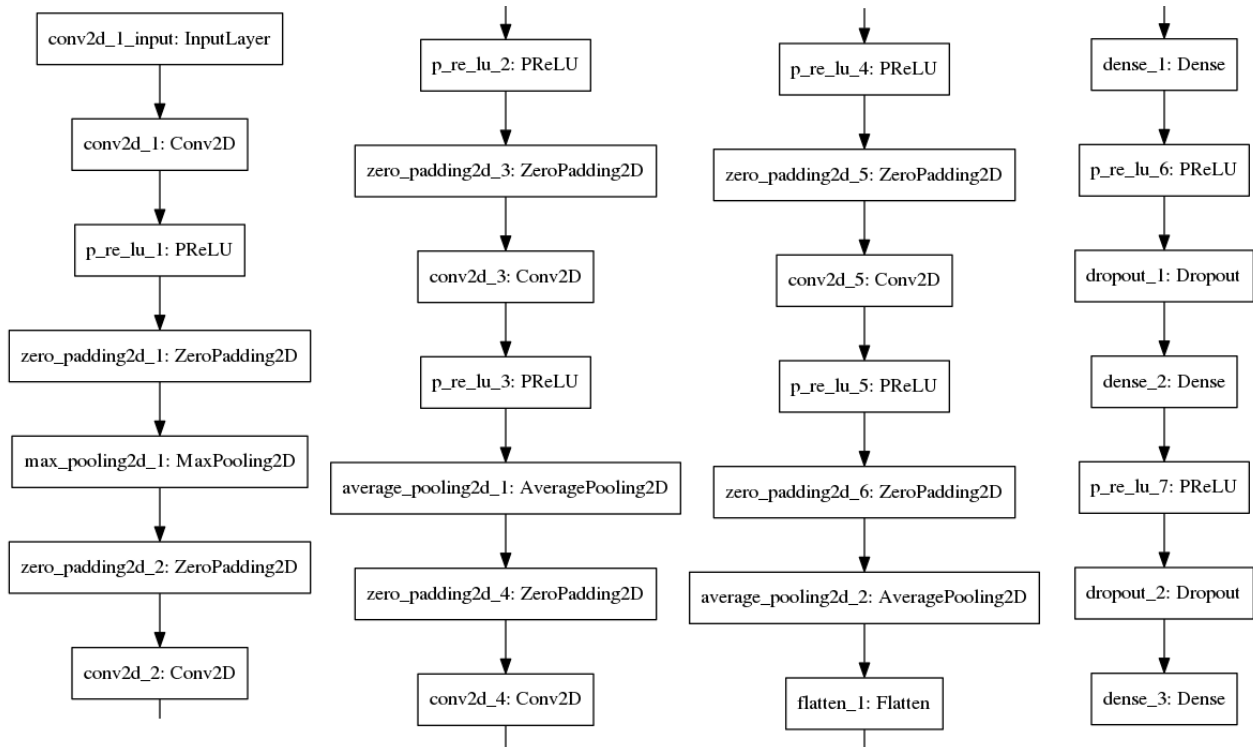


1. (1%) 請說明你實作的 CNN model，其模型架構、訓練過程和準確率為何？

答：



上圖為我使用的 CNN 模型架構，conv2d 即為一般的 CNN layer，可以觀察到我總共使用五層，而每層的設定分別如下：

conv2d_1：filter 數 64、filter size 為 5x5、參數量為 1664

conv2d_2：filter 數 64、filter size 為 3x3、參數量為 36928

conv2d_3：filter 數 64、filter size 為 3x3、參數量為 36928

conv2d_4：filter 數 128、filter size 為 3x3、參數量為 73856

conv2d_5：filter 數 128、filter size 為 3x3、參數量為 147584

activation 的部分則使用 keras 的 PReLU function，此函數類似於 Leaky ReLU，在 threshold 之下的值會乘上一個 alpha(小於 1)，比較不同的是 PReLU 的 alpha 也是可以經過學習而調整的，故其中亦有些參數，每個 activation 的參數量如下：

p_re_lu_1：123904，p_re_lu_2：30976，p_re_lu_3：30976，p_re_lu_4：12800，p_re_lu_5：12800，p_re_lu_6：512，p_re_lu_7：512

在 Pooling 的部份，我分別使用 max_pooling2d 與 average_pooling2d，讓在比較後層的值基本上都可以被考慮到，而不只是單純做 downsampling，同時因為我最後的模型有調整 stride size，這樣子也比較能 cover 到該 cover 的值，設定如下：

max_pooling_2d_1：pool_size 為 5x5、strides 為 2x2(亦即每兩排取一次 pooling)

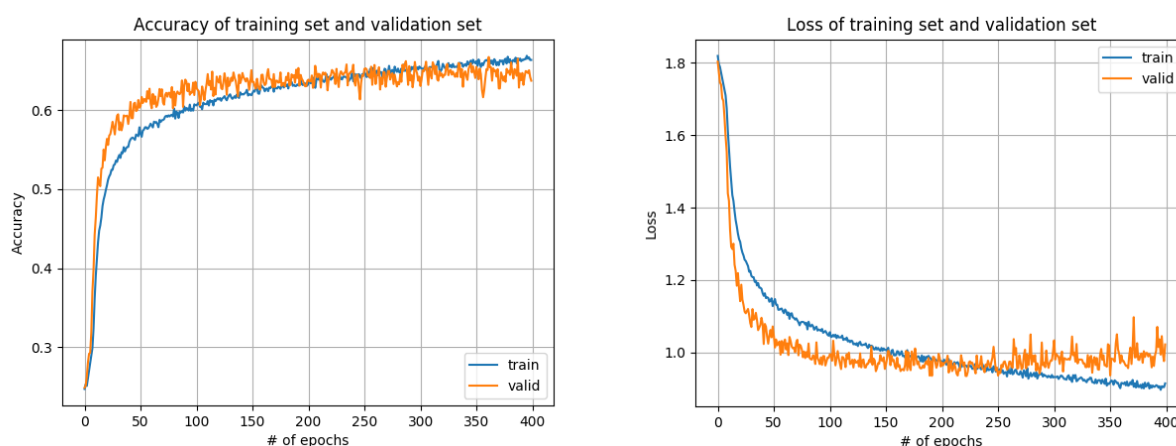
average_pooling_2d_1：pool_size 為 3x3、strides 為 2x2

average_pooling_2d_1：pool_size 為 3x3、strides 為 2x2

此外，由於原圖大小只有 48x48，為了使 conv 跟 pooling 時有足夠的值可取，我加了數層大小為 1 的 zero_padding，以利模型之訓練。最後是 flatten 與 dense neural network 的部份，我在 flatten 後接了兩層大小為 512 的 NN，並設定 dropout 率為 0.3，再接到大小為 7 的 NN(此層的 activation function 為 softmax)以作為 output，flatten、兩層 512 的 NN、output 層的參數量分別為 1638912、265656 和 3591，因此總參數量 2414599。

在其他設定上，我設定 batch_size 為 64，從 training data 中切 1/10 作為 validation set，並使用 keras 的 ImageDataGenerator，加上旋轉(範圍為正負 40 度)或位移(範圍為圖檔大小的 0.2 倍)，產生部份不同的訓練資料，以增加訓練模型的強度，而實際上效果，在未使用此 preprocessing 技巧前，上傳至 Kaggle 準確率約為 57%，而使用後可超過 strong baseline。

下圖分別為我的 CNN 模型跑 400 個 epoch，training set 和 validation set 的 loss 及 accuracy：

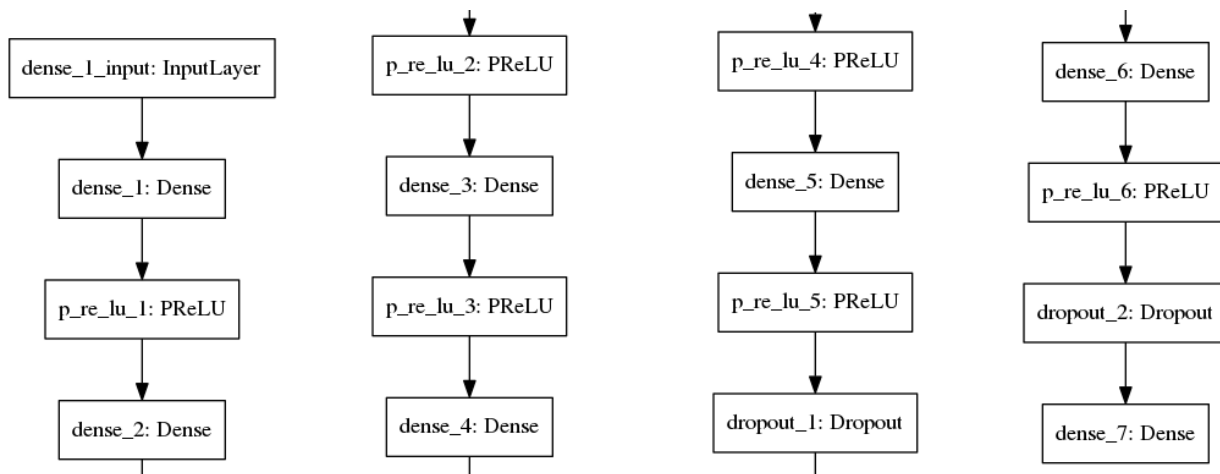


從上面兩張圖可以發現，CNN 模型的收斂速度較慢，而且 validation set 在前期甚至比 training set 的結果來得好，表示前期所訓練的模型較為 generalized，是大概到了兩百多個 epochs 之後，training set 的 loss 和 accuracy 才表現的比較好，且 validation set 的 loss 有微微上升的狀況；而整體而言，CNN 模型的訓練狀況可以達到六成多的準確率，雖然 CNN 模型需要較多的時間才能被訓練出來，但表現出相當不錯的效果，可以看出 CNN 模型對於圖片分類的訓練的確是有幫助的。

傳上 Kaggle 的最好結果(public accuracy 約為 0.658 左右)，為使用 adadelata 作為 optimizer 之方式(繪圖使用的是以 adam 作為 optimizer，訓練速度較快且也有不錯效果)，訓練約 650 個 epochs 之後的結果，由於使用此方式訓練費時較久，且第二題 DNN 約數十個 epochs 便收斂，用 adam 訓練已足以比較兩個模型的差異，故未將 adadelata 作為 optimizer 的結果繪出。

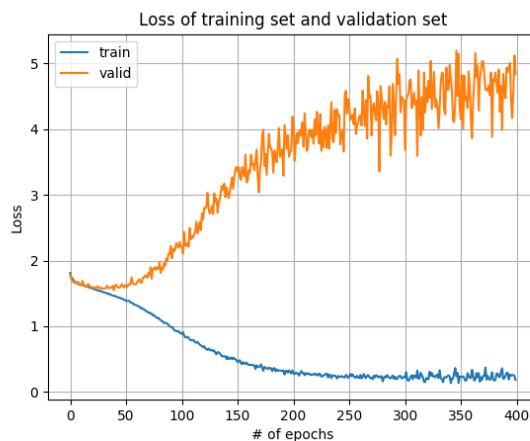
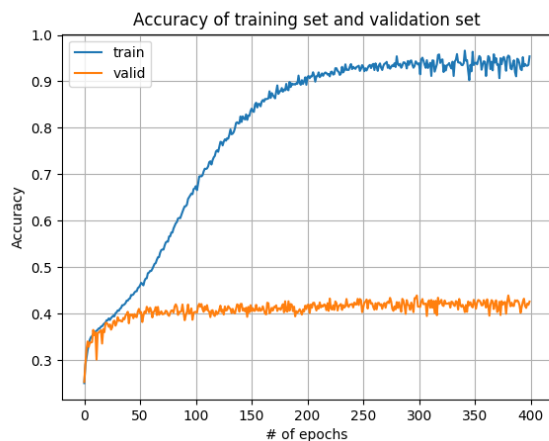
2. (1%) 承上題，請用與上述 CNN 接近的參數量，實做簡單的 DNN model。其模型架構、訓練過程和準確率為何？試與上題結果做比較，並說明你觀察到了什麼？

答：



上圖為我使用的 DNN 模型架構，dense 即為一般的 neural network 架構，除了最後一層用 softmax 之外，中間一樣都是使用 PReLU 作為 activation function，為了使得此模型的參數量與第一題所使用的 CNN 模型差不多，我總共使用六層 dense，每層皆為大小 512 的 NN，故第一層 NN 的參數量為 1180160(因為 input size 是 48x48)，最後一層參數量為 3591，其他層皆為 265656；同時，於最後兩層使用 dropout，dropout 率也是 0.3，output 一樣為大小為 7 的 NN，故總參數量為 2500103。

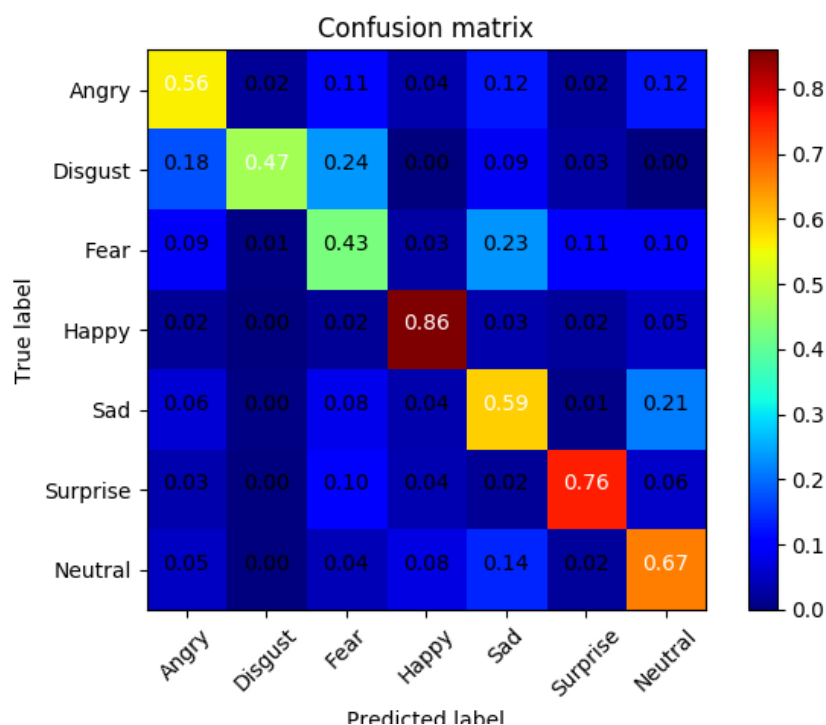
在其他設定上，我設定 batch_size 為 64，從 training data 中切 1/10 作為 validation set，由於對於 DNN 來說沒有 Image 的概念，故沒有使用 ImageDataGenerator 做 Preprocessing，故比較的部份也與 CNN 沒有使用 ImageDataGenerator 的版本做比較，出來的結果也如預期般的比較差，成功率大約四成出頭，下圖分別為我的 DNN 模型跑 400 個 epoch，training set 和 validation set 的 loss 及 accuracy：



我們可以觀察到，在 DNN 模型下，跑數十個 epoch 感覺就收斂了，validation set 的 loss 在 20~30 個 epochs 左右就開始上升，而準確率的部份也在 50 個 epochs 左右就停滯不前，於四成出頭徘徊，而在那之後不斷地 overfitting training set，使其 loss 下降，準確率上升至九成多；相較於在 CNN 模型，training set 和 validation set 沒有這麼快收斂，在 400 個 epochs，兩者的 loss 和 accuracy 皆處於相近的狀況，DNN 模型因為無法掌握圖片的 locality，及相同特徵會在圖片重複出現的特性，即使在使用參數量約相等的情況下，訓練效果依然不佳，因為

對 DNN 模型來說，圖片的 pixel 不管在哪個位置是不重要的，所以便不能掌握住人臉中如眉形、嘴巴變化的狀況(這點會在第四題中提到)，造成訓練準確率，較之 CNN 模型來得差。

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：



上圖為分析 validation set(原訓練資料的 10%)中，各個 class 間彼此混用的狀況。從上圖可以看到，最容易分辨出來的表情是”開心”，推測可能的原因是這七個 class 中，只有開心是屬於正面的表情(其他 class 的嘴形或眉形等可表現情緒的特徵都不容易和其他的搞混)，所以辨識正確率可達 86%；其次分辨準確率也很高的是”驚訝”，推測原因應該是驚訝算是中立的表情，而且表情也比較獨特(嘴形容易張大)，所以又能跟”中立”表情做區別，只是偶爾會與”恐懼”搞混(約 10%，因為恐懼也會有嘴形張大的狀況，從第四題挑選的例圖也可以看出)。

接下來觀察”中立”與”難過”，雖然難過屬於較為負面的情緒，但呈現出來的表情不會太誇張，因此容易跟”中立”，從 confusion matrix 可看出，如果是”中立”或”難過”的情緒，有 80% 會被認為是”中立”與”難過”；”生氣”的部份準確率也不高，容易與”恐懼”、”中立”、”難過”混用，可能是因為”生氣”的表情比較多元(可以挑眉抿嘴，如第四題例圖，也可以開嘴瞪大眼睛，如說明檔中的例圖)，所以沒特定跟某一個 class 互相搞混。

最後是”恐懼”以及”厭惡”，分別是辨識成功率最低的兩個 classes，”恐懼”的表情也比較多元(可以張大嘴巴瞪大眼睛，如第四題例圖，也可以因害怕而哭泣，像是說明檔中的範例圖)，所以容易與多個 classes 搞混，其中最容易搞錯的 class 是”難過”。而”厭惡”的部份，最容易與”生氣”和”恐懼”搞錯，推測可能是因為眉形的變化方式與嘴形的張開方式較為接近。

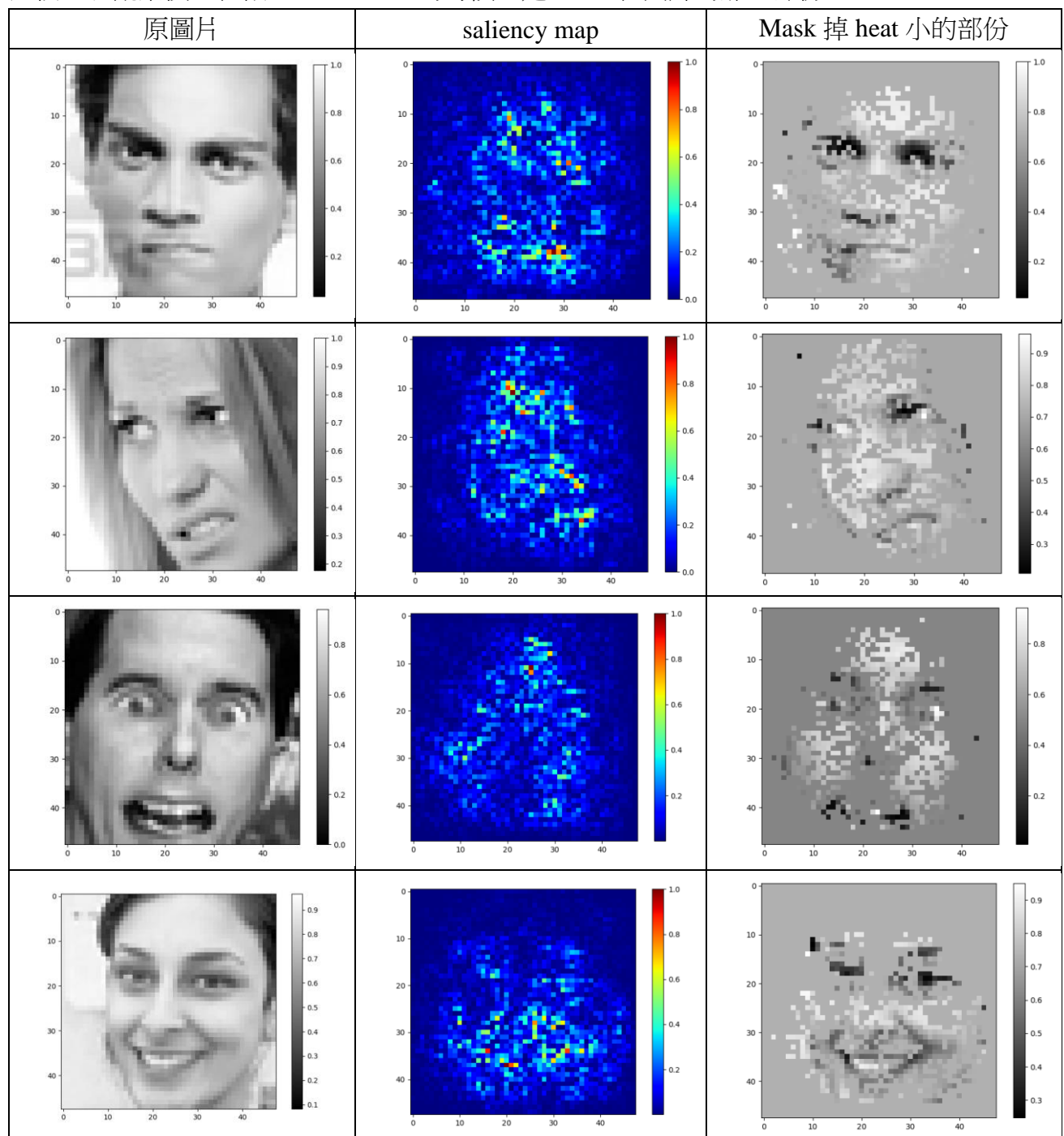
從上述分析大致可以看出如果表情有一些明顯的特徵，像是”開心”的表情，笑容明顯、”驚訝”的表情，眼睛瞪大且嘴巴圓圓張開，就比較容易辨識正確；如果某種表情呈現方式比較多

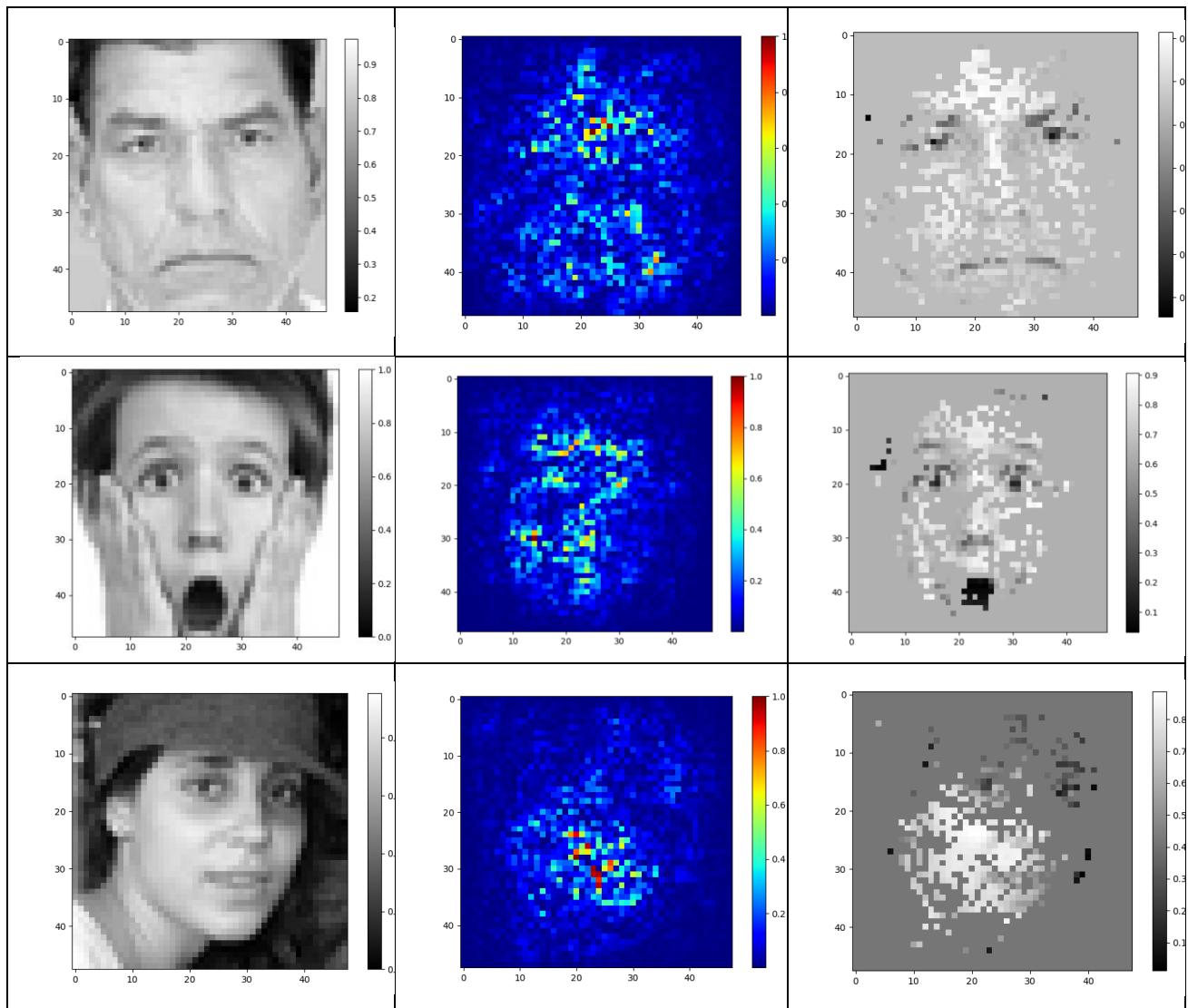
元，且與其他情緒有共通之處(如生氣、恐懼)，則比較難被區分出來，而特徵的部份，下列第四題中的觀察亦會提到。

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：

以下我從 validation set 中，從每個 class 中分別挑選一張圖片(依序為生氣、厭惡、恐懼、開心、難過、驚嚇、中立)，繪出其 saliency map，並且將原圖以及 mask 掉 heat 小的部份後做比較，以觀察模型在做 classification 的時候，是 focus 在圖片的那些部份。





從上面七張圖來看，其實可以很明顯看到，**heat** 較高的地方集中臉的部份，特別是在眉毛、眼睛以及嘴巴，舉一些例子：生氣的表情，眉形從鼻端往外向上、恐懼的表情，嘴巴會橫向的張開(或像說明檔中那樣快要哭泣的感覺)、開心的表情，嘴巴的笑容與其他 **classes** 可以分的滿明顯的、驚訝的表情，眼睛瞪大且嘴巴縱向張開，中立的表情在眉宇與嘴巴沒有特殊的變化，因此只框出臉等等，這和我們的直覺也相符合，人的表情基本上是從這幾處所展現，從程式中便能清楚地看到，這些部份對於 **CNN** 模型做表情分類是較有幫助的，而且若這些特徵相當明顯，且單屬於某個 **class**(比方說開心的笑容、驚訝的縱向張嘴等)，就能讓 **CNN** 模型從圖片中擷取到的 **features** 發揮功用，進而達成分類成功的效果！

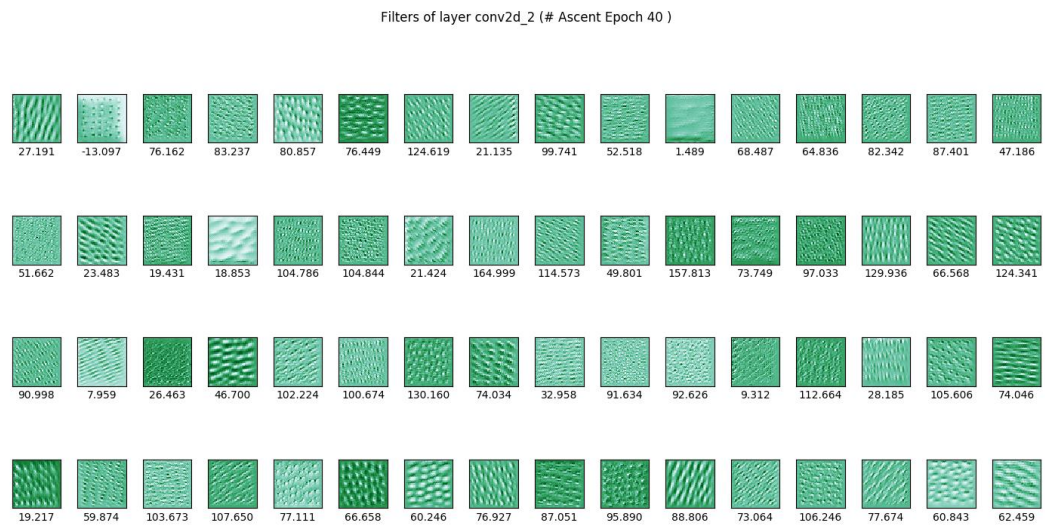
5. (1%) 承(1)(2)，利用上課所提到的 **gradient ascent 方法，觀察特定層的 **filter** 最容易被哪種圖片 **activate**。**

答：

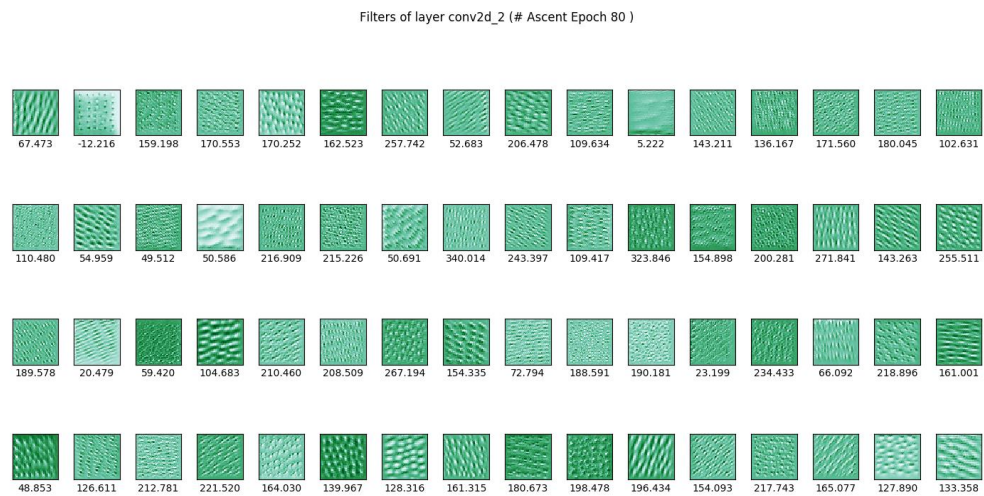
利用梯度遞增法，使用白噪音找出最能激活特定 **filter** 的圖片之部份，我挑選第一題模型中的 **conv2d_2** 和 **conv2d_3** 兩層的輸出做觀察，分別為跑 40 與 80 個 **epochs**，看看其輸出的效

果；而給定輸入圖片，取出特定層的輸出，我以第四題例圖中的”開心”及”驚訝”表情作為輸入圖片，同時分別觀察 conv2d_2 跟 conv2d_3 這兩層的輸出。

conv2d_2 (40 epochs):

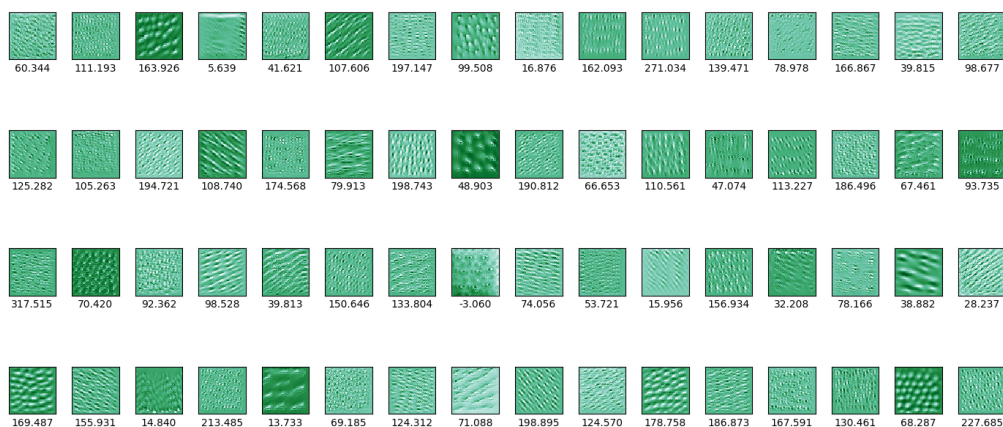


conv2d_2 (80 epochs):



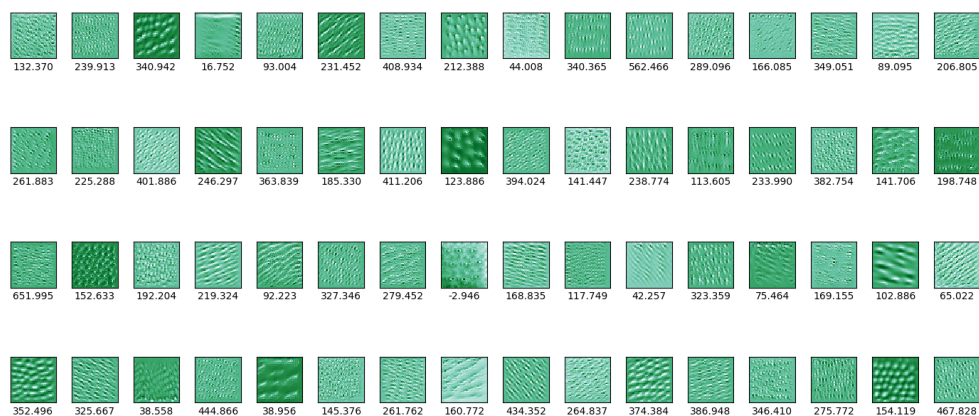
conv2d_3 (40 epochs):

Filters of layer conv2d_3 (# Ascent Epoch 40)



conv2d_3 (80 epochs):

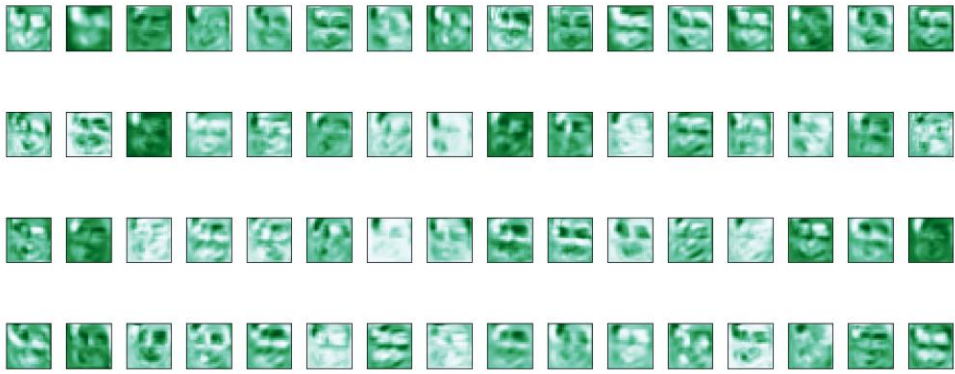
Filters of layer conv2d_3 (# Ascent Epoch 80)



可以觀察到，在兩層之中的 **filter** 長相，其實不太一樣，可見每一層取出的 **features** 是不同一樣的；同時，輸入白噪音可以對於每一個 **filter** 的激活程度皆不同(從圖片底下的數字可以觀察到)，而且隨著 **epoch** 變多，激活程度也會增加。接著，我將展示輸入圖片後特定層的輸出果。

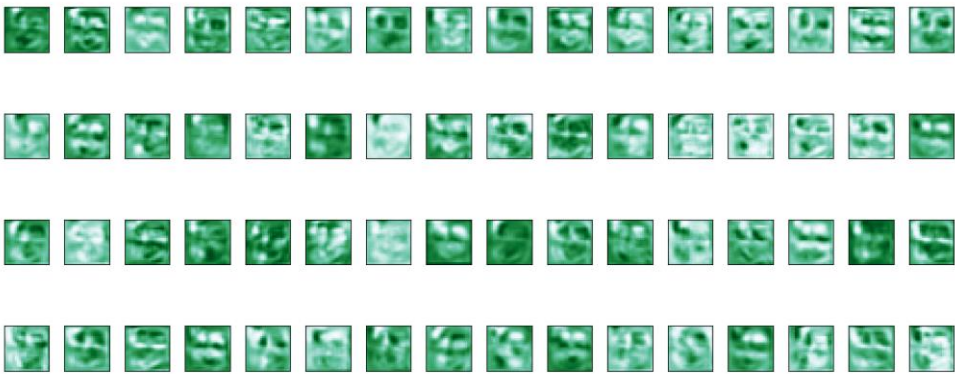
conv2d_2 (開心):

Output of layer0 (Given image28600)



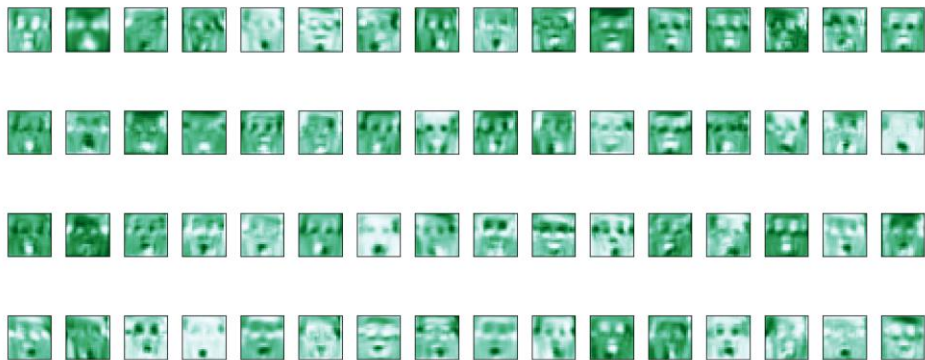
conv2d_3 (開心):

Output of layer1 (Given image28600)

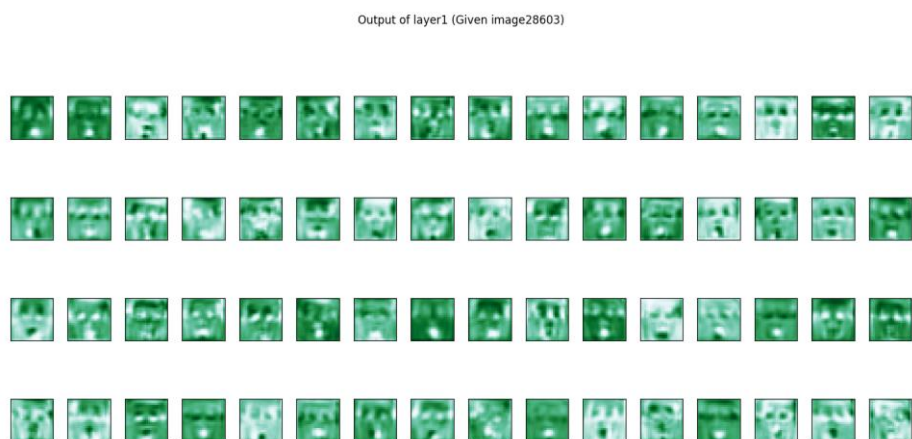


conv2d_2 (驚訝):

Output of layer0 (Given image28603)



conv2d_3 (驚訝):



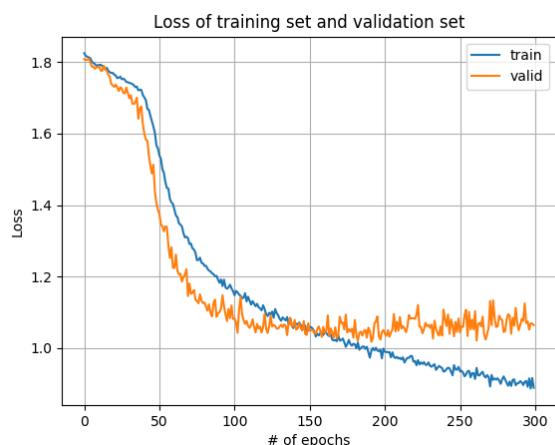
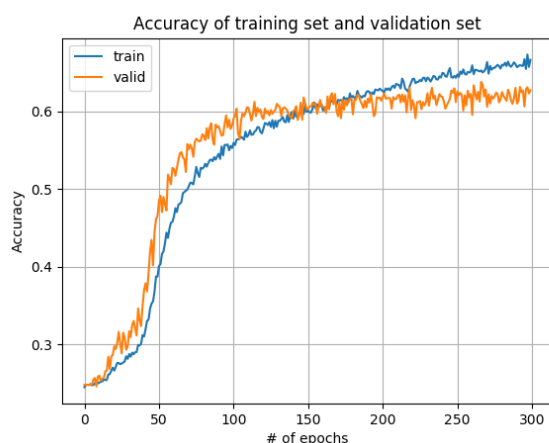
從上述幾個例子，可以觀察到這些 **filter** 部份確實有抓到圖片中重要的特徵，不論是”開心”表情中的笑臉，或者是”驚訝”表情中的縱向張嘴跟瞪大眼睛，都可以從這兩層 **filter** 觀察出來，而其實到比較後層(如 conv2d_5)，這些反而比較不明顯，可能是因為我的模型中有加 **zeropadding_2d**，或是那部份取出比較高維度的 **feature**，所以人眼比較看不出什麼特徵。

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

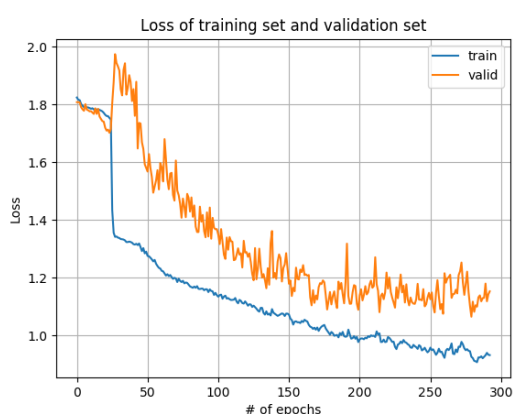
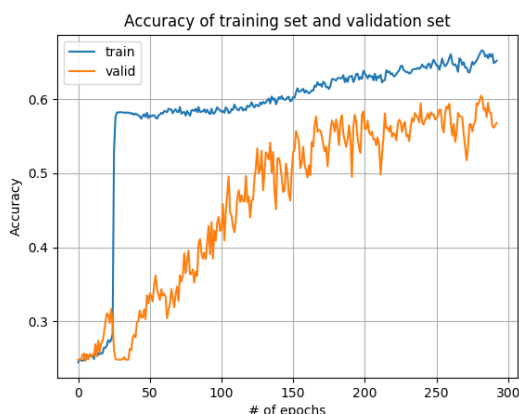
答：此部分我將一半的 training data 的 label 移除，並使用 self-training 的方式實作 semi-supervised learning，每過一個 iteration，便會將沒有 label 的 data 拿去做預測，預測出來的如果最後一層 softmax 的結果大於 0.45，便會在下一輪加入 training set 去訓練模型，每一輪都會重新選出該被放進 training set 的資料，並運行 300 個 epochs 觀看結果。

我同時展示出單 train 一半 data 的結果以及使用 self-training 的結果：

單 train 一半 data：



使用 self-training：



看兩者的結果比較，可以發現 self-training 的效果在 training set 上表現的比單 train 來的好，accuracy 在 20~30 個 epochs 便急速上升，loss 也在 20~30 個 epochs 便急速下降，相較於單 train 的結果在這兩者上都是較為緩慢的變化；然而，在 validation set 的結果卻不太好，不論是 accuracy 或 loss，變化都比較不穩定，且值也沒有比較好 (accuracy 平均而言較低、loss 平均而言較高)，這可能是因為 threshold 選的不好 (0.45)，或者 epochs 數太少 (300)，使其沒辦法在 validation set 上有穩定且較好的結果，由於時間關係並未做其他的實驗，但從 training set 的變化，還是可以感覺到，若 threshold 選的好且跑足夠的 epochs 數量，應能跑出比單 train 只有 label 的 data 來的好！

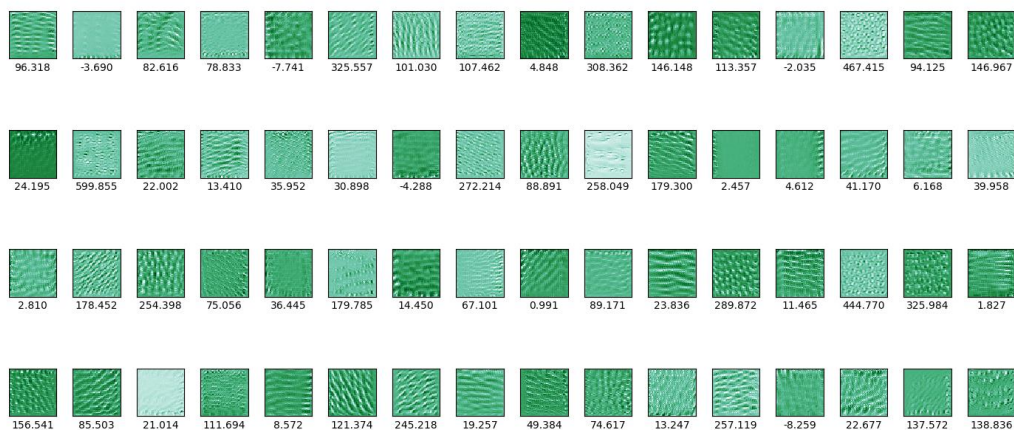
[Bonus] (1%) 在 **Problem 5** 中，提供了 3 個 hint，可以嘗試實作及觀察（但也可以不限於 hint 所提到的方向，也可以自己去研究更多關於 CNN 細節的資料），並說明你做了些什麼？ [完成 1 個: +0.4%，完成 2 個: +0.7%，完成 3 個: +1%]

































































答：

(1) use other model with poor performance to see what is the difference

我使用 semi-supervised learning 所學習出的模型，輸入白噪音並取出 conv2d_3 的圖像 (因為 conv2d_2 與 conv2d_3 效果差不多，故取其中一種觀察)。

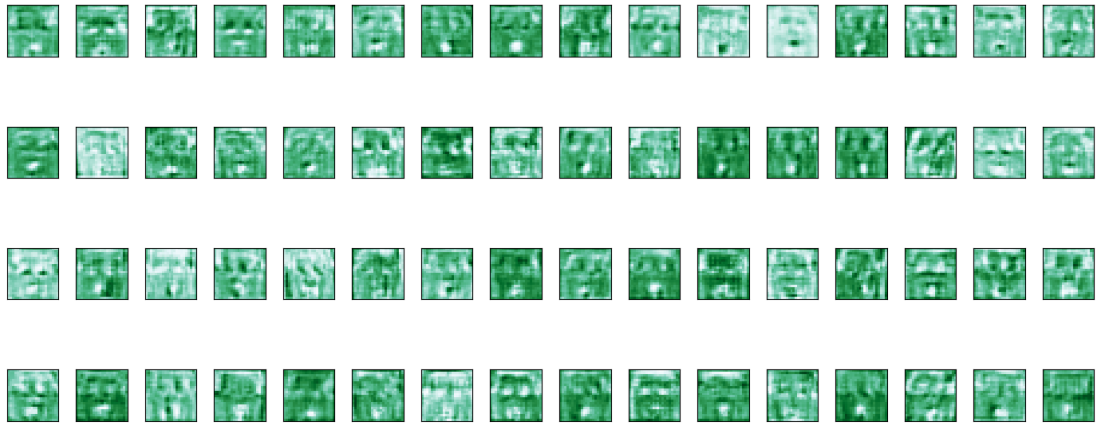
Filters of layer conv2d_3 (# Ascent Epoch 40)



																251.225	1.713	199.104	187.830	-2.954	706.058	246.202	248.699	18.357	653.318	325.298	254.994	0.208	961.861	232.875	334.675
																64.371	1266.594	57.578	39.299	87.949	79.253	1.585	593.939	213.510	576.470	412.802	13.151	20.283	97.287	17.734	103.405
																9.864	397.849	542.890	180.901	92.517	407.511	42.118	164.074	6.735	208.043	67.702	614.672	35.196	920.052	677.939	16.183
																345.674	204.553	51.916	269.513	32.958	290.636	546.391	52.126	123.470	176.564	34.909	575.157	-7.078	59.339	309.606	307.240

接著將第五題的測試圖片，輸入同樣的模型，並取出 conv2d_3 的結果(因為 conv2d_2 與 conv2d_3 效果差不多，故取其中一種觀察)

Output of layer0 (Given image28603)



若仔細觀察比較此兩圖，雖然仍舊可以看出有些輪廓，但也可以發現這些情緒所擁有的特徵較之第五題不明顯，不論是”開心”情緒中的笑容，以及”驚訝”情緒中瞪大眼睛與張嘴，都比較看不出來一些，可看出特徵是否被清楚辨識出來，對於準確率其實是有差別的。