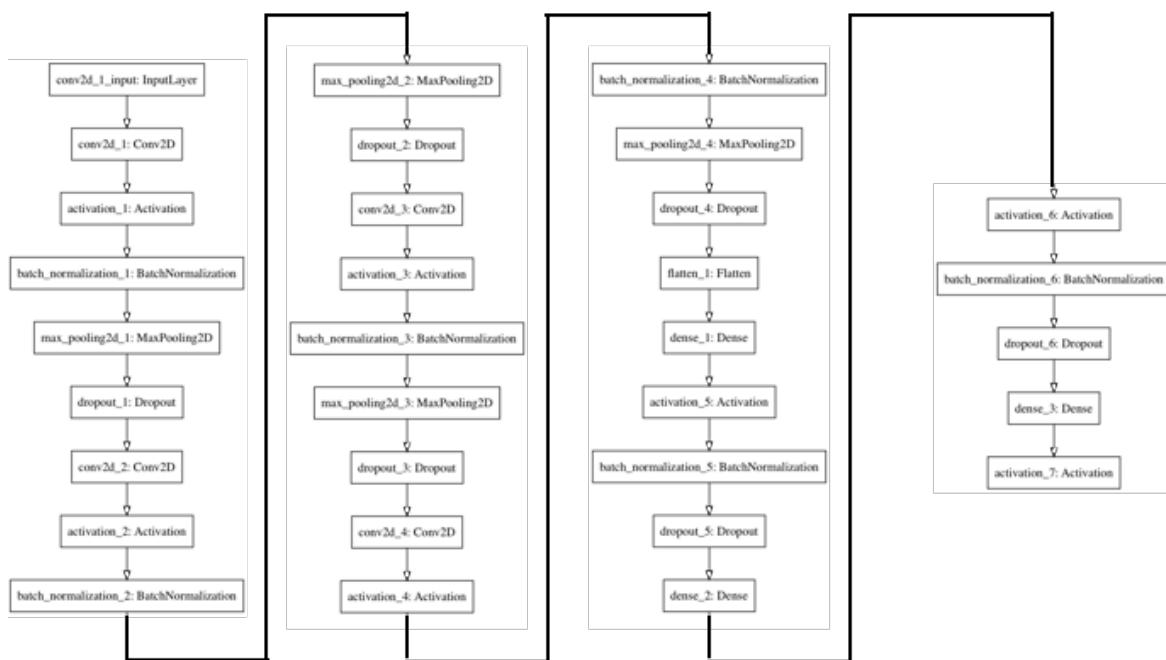


- (1%) 請說明你實作的 CNN model，其模型架構、訓練過程和準確率為何？

答：

模型的內容在 CNN 的部分的順序皆為 Convolution Layer → Activation Layer (全部為 relu) → Batch Normalization Layer → Max Pooling Layer → Dropout Layer，此結構共接了四層，每層的差別只在於 Convolution Layer 的 filter 的數目以及 Dropout 的比例；在 DNN 的部分則接了兩層，每層內部順序為 Dense Layer → Activation Layer (全部為 relu) → Batch Normalization Layer → Dropout Layer，同樣地，每層的差別只在於 Dense Layer 的 neuron 數目以及 Dropout 的比例，最後再接到 Output Layer，用的 Activation function 為 softmax。而整個模型的 Loss function 是 categorical cross entropy，optimizer 為 adamx。模型訓練架構如下：



訓練過程則有採用 keras 內建的 Image Data Generator，來增加 training data 的雜訊，經過實驗發現這樣的效果還滿不錯的。以上傳到 Kaggle 的 public 的部分的準確度來判斷，在同樣的模型下，若使用 Image Data Generator 的話，大致上準確率能到 66% ~ 67%，相較於未使用該法時的準確率 (61% ~ 62%)，提高了將近 5% 的準確率。至於最後叫上來的 model，我則採取先用原先沒有經過 Image Data Generator 改變的資料簡單訓練 5 個 epochs，在由經過處理過的資料訓練 100 個 epochs。

```

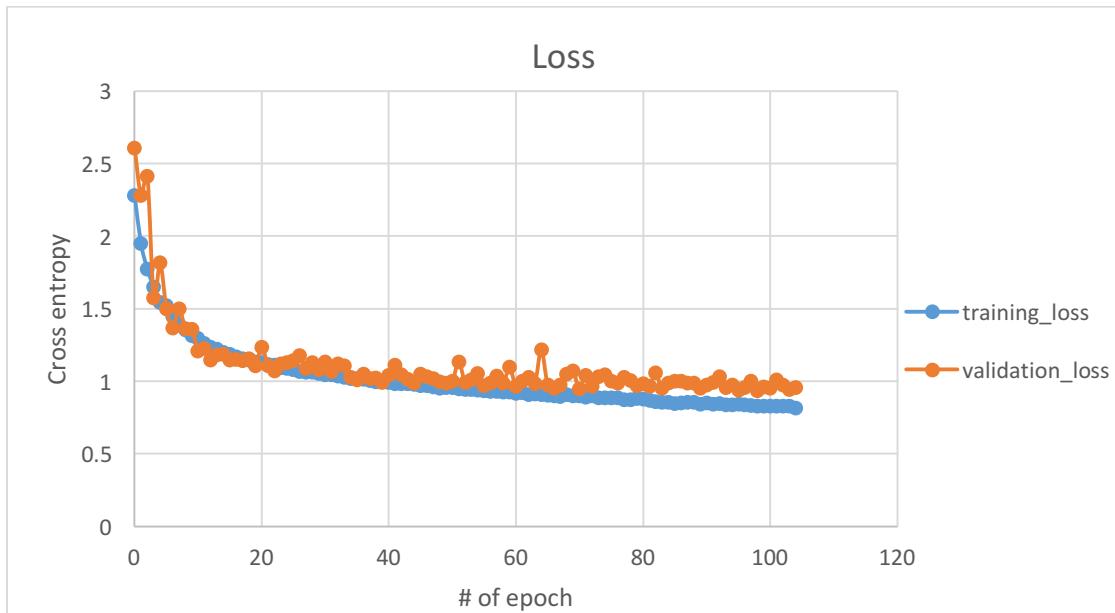
model.fit(train_feature, train_label, validation_data=(valid_feature, valid_label), batch_size=batch, epochs=5)
model.fit_generator(datagen.flow(train_feature, train_label, batch_size=batch),
                    steps_per_epoch = train_feature.shape[0]/batch,
                    epochs=100,
                    validation_data = (valid_feature, valid_label),
                    )

```

而 Image Data Generator 則設定為將圖片隨機進行選轉 0 ~ 10 度、水平翻轉、水平位移、垂直位移。

```
datagen = ImageDataGenerator(  
    featurewise_center=False,  
    samplewise_center=False,  
    featurewise_std_normalization=False,  
    samplewise_std_normalization=False,  
    zca_whitening=False,  
    rotation_range=10,  
    width_shift_range=0.1,  
    height_shift_range=0.1,  
    horizontal_flip=True,  
    vertical_flip=False)  
    # set input mean to 0 over the dataset  
    # set each sample mean to 0  
    # divide inputs by std of the dataset  
    # divide each input by its std  
    # apply ZCA whitening  
    # randomly rotate images in the range (degrees, 0 to 180)  
    # randomly shift images horizontally (fraction of total width)  
    # randomly shift images vertically (fraction of total height)
```

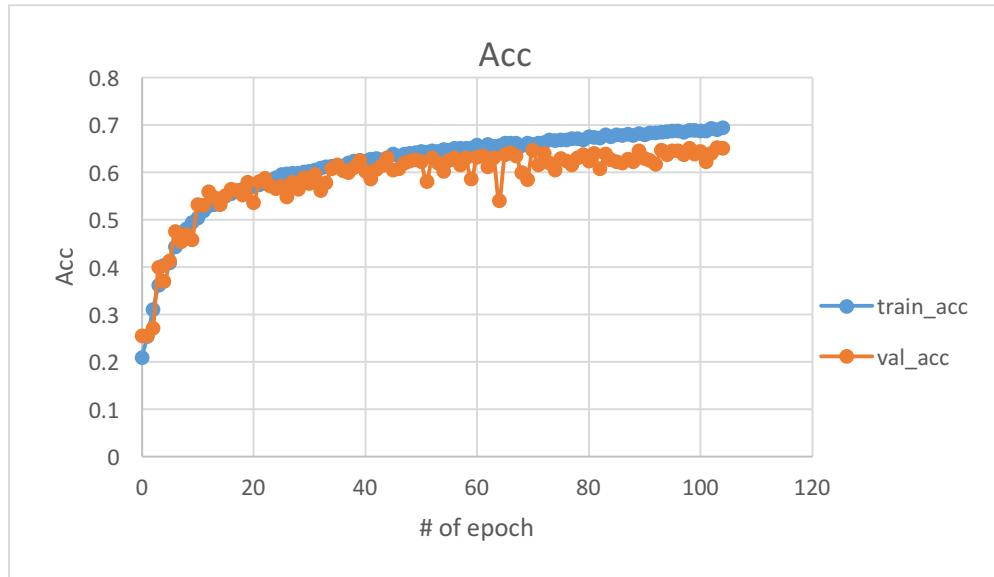
依照上述方式訓練完之後，將訓練過程中 model 在 training data 上的 loss 變化和 validation data (取 train.csv 的前 5000 筆資料) 的 loss 變化記錄下來並作比較：



可以發現訓練出來的 model 在 training data 上的 loss 在前幾個 epochs 中，下降幅度是很顯著的，但等到大約過了 20 個 epochs 後，雖然 loss 仍有在下降的趨勢，但已經慢了很多；validation data 的 loss 變化大致上跟著 training data 的趨勢在下降，但相較於 training data 的穩定下降，validation data 的 loss 偶爾會在某幾個 epoch 突然往上升，而且從第 60 個 epoch 左右，validation data 的 loss 的基本上只在上下浮動而已，並不像 training data 還有再下降的趨勢。

在準確度方面，model 在 training data 上的準確度一開始上升很快，但和 loss 一樣，在差不多第 20 個 epochs 後上升幅度便開始趨緩；而 validation data 的準確度的變化走向大致和 training data 的一樣，不過也是差不多在 60 個 epochs 之後，準確度便不再上升，只會上下浮動而已。

對 training data 和 validation data 的準確度如下圖：



值得一提的是，上傳到 Kaggle 的辨識結果在 public 的部分通常準確度會比我用 validation data 切出來的準確度要高出 1% ~ 2%。

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：

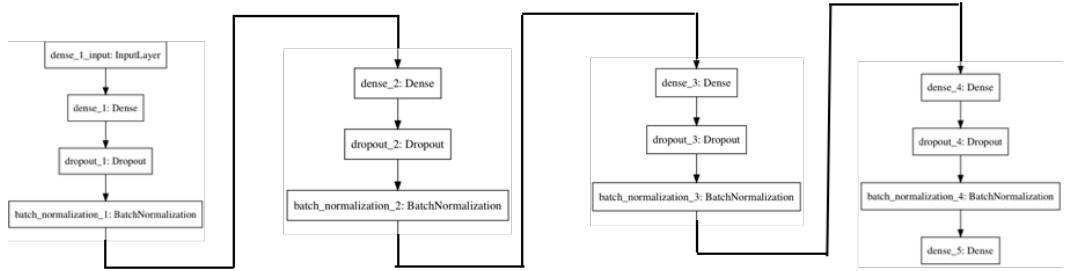
根據 model.summary() 所得到的 CNN 的參數量如下：

```
=====
Total params: 1,765,511.0
Trainable params: 1,762,439.0
Non-trainable params: 3,072.0
=====
```

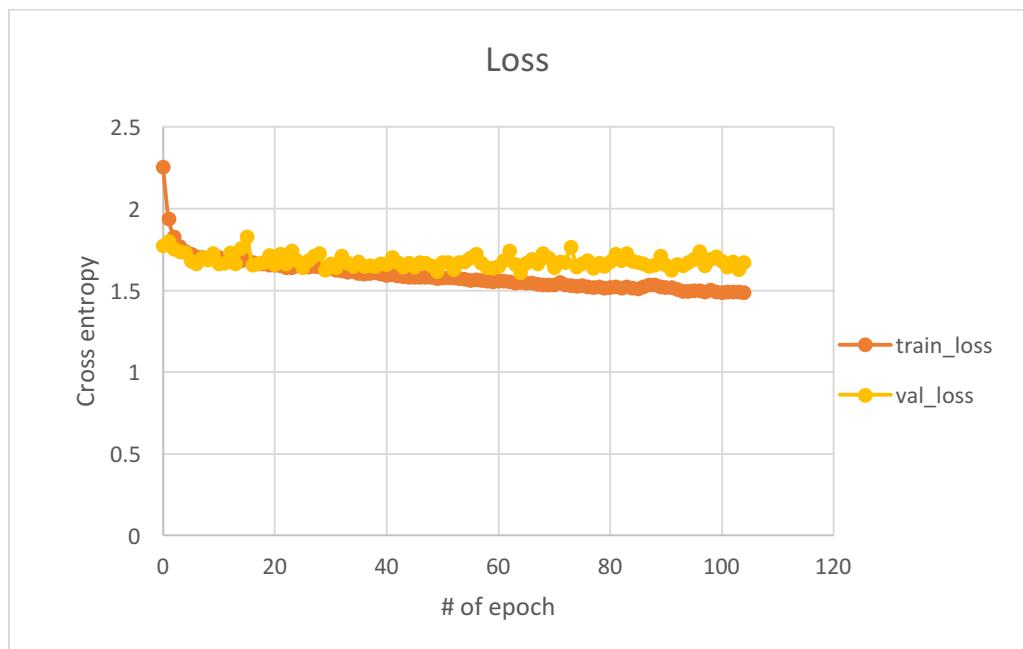
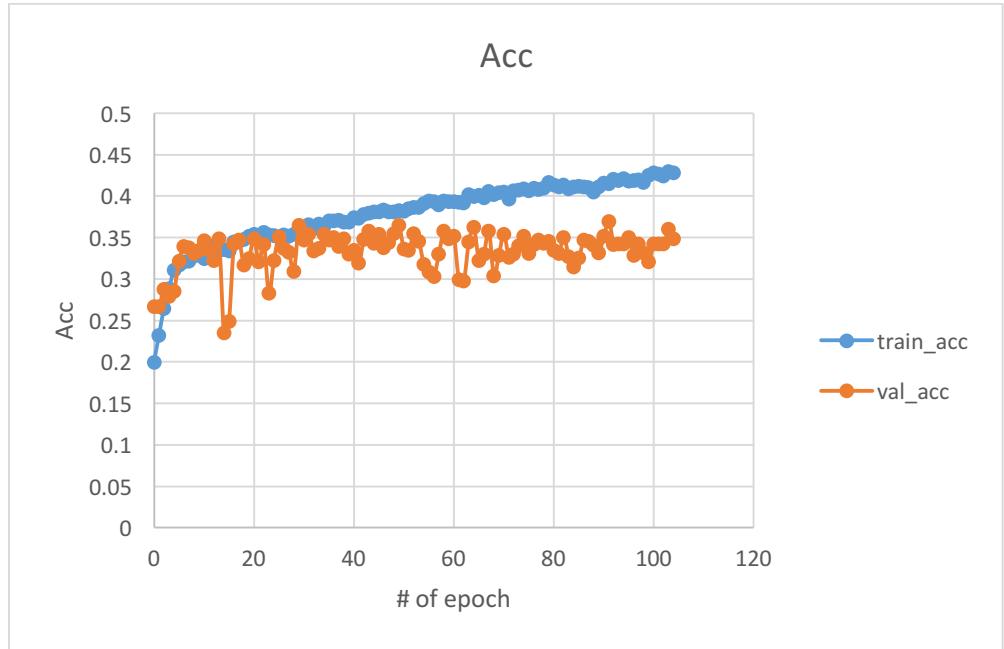
依此設計出參數量大致相仿的 DNN 模型，參數量如下：

```
=====
Total params: 1,723,271.0
Trainable params: 1,718,407.0
Non-trainable params: 4,864.0
=====
```

DNN 的模型部分則接了四層，每層內依序為 Dense Layer (Activation: relu) → Dropout Layer → Batch Normalization Layer。其餘設定如 Loss function、Optimizer、validation data 的選取等皆和 CNN 相同。



訓練結果如下：



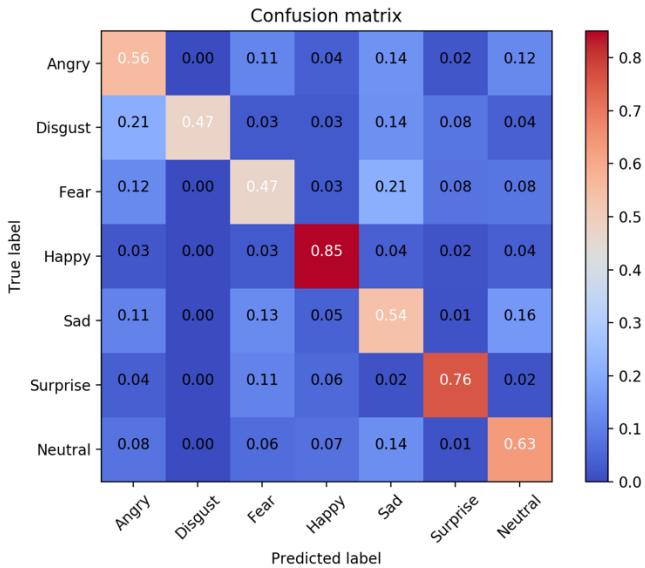
比較第一題的結果可以發現在差不多的參數量下，DNN 訓練出來的結果差了

CNN 許多，辨識準確率只有 0.35左右，而且基本上 validation data 只有在前 10 個不到的 epoch 在準確度和 loss 方面有進步而已，後面基本上都只在上下浮動而已。這樣的結果也驗證了課堂上所說的，在同樣的參數量下，CNN 較 DNN 有較佳的圖片辨識能力，因為 CNN 本身的架構有透過許多的 filter 將圖片分區塊辨識並利用 max pooling 的方式將每層 layer 的重點擷取出來的能力；但 DNN 則不然，DNN 是將所有圖片的像素不分主次的全部一起處理，而這種方式則造成容易受到圖片其他雜訊干擾，而在參數量相對有限的情況下便無法提升辨識結果。

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 如下：



根據結果，可以發現情緒為 Happy 的圖片辨識結果最佳，Surprise 的辨識結果次之，而我覺得這樣的結果也不令人意外，因為這些表情除 Happy 外，其他種類大都屬於負面情緒，而 Surprise 通常表情最誇張，所以不同情緒所相對應的表情若相差愈大，則愈容易分辨。

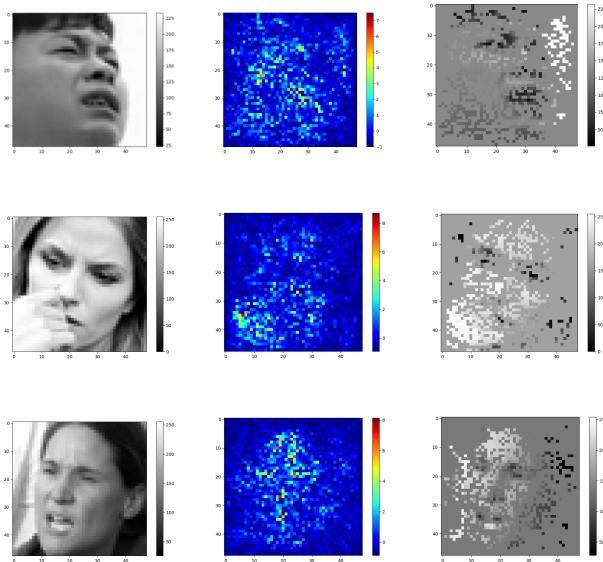
在答錯的圖片中，可以發現 Disgust 的圖片有 0.21 的機率被辨識成 Angry，但比較令人意外的是，在這次訓練出來的模型中，被辨識為 Disgust 的圖片就一定是 Disgust，完全沒有錯誤。另外 Fear、Sad、Angry 這三者也有不小的機率被誤認，其實也可以想像，當情緒沒有太強烈的時候，這三者情緒所代表的表情其實是差不多的，大概都是眉頭微鎖的臭著一張臉。

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：

CNN 在臉部辨識的時候大部份都著重在眼睛、嘴巴，偶爾有部分的臉頰及額頭，如下列幾組圖片：

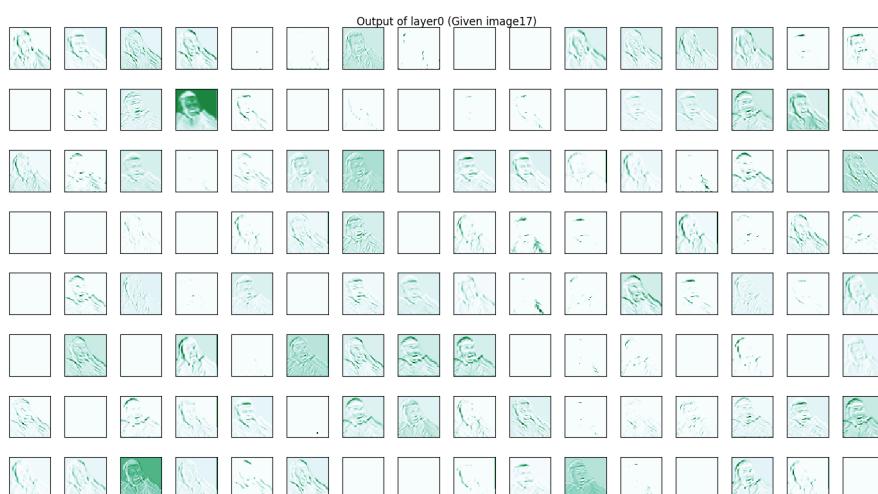


根據以上結果，我想也說明了機器真的有學到臉部表情辨識的重點，因為我們人一般主要也是從這些區域去辨識一個人的喜怒哀樂的。

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：

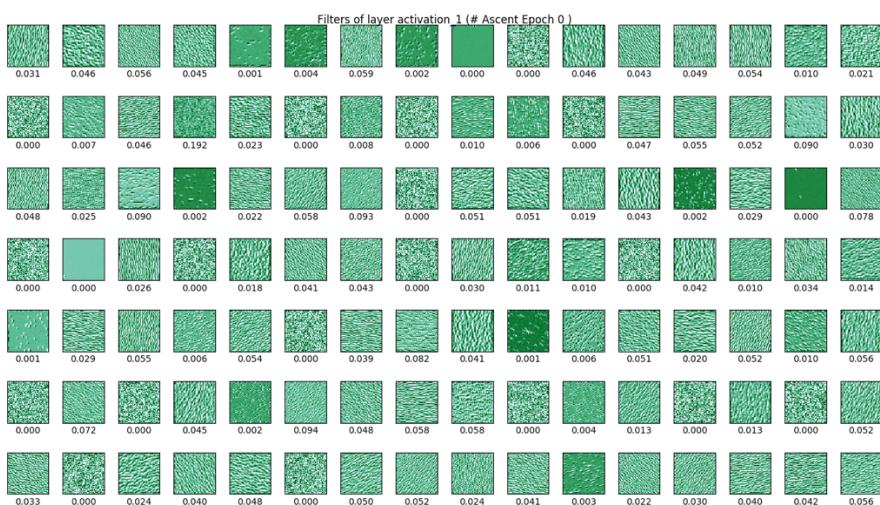
給定 validation data 中的第 17 張圖，分別取出前三層 activation layer 其中 128 個 filter 的 output：

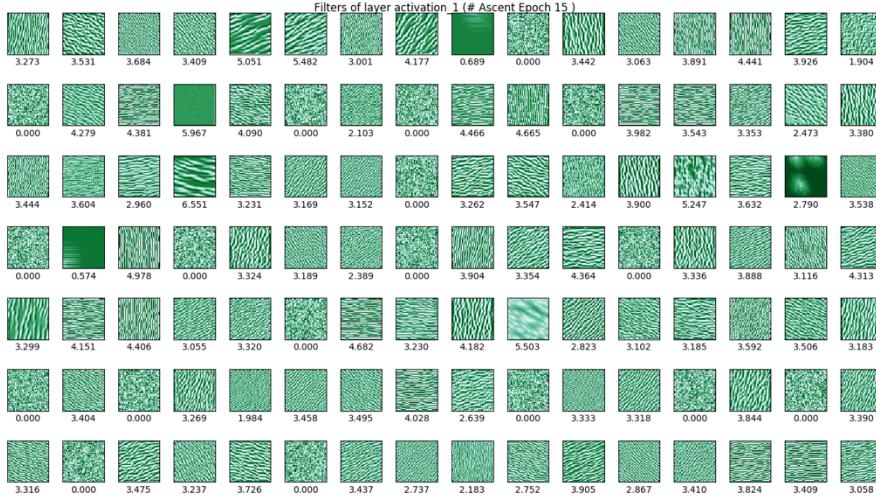




可以發現除了第一層的輸出還算比較清楚以外，愈往深層的 layer，輸出愈來愈無法用人眼來判斷。我想這也表示機器所訓練出來的模型在每一層的 CNN 都做了一些我們無法理解的轉換，而機器也是經由這些轉換才能達到最後辨別分類的結果。

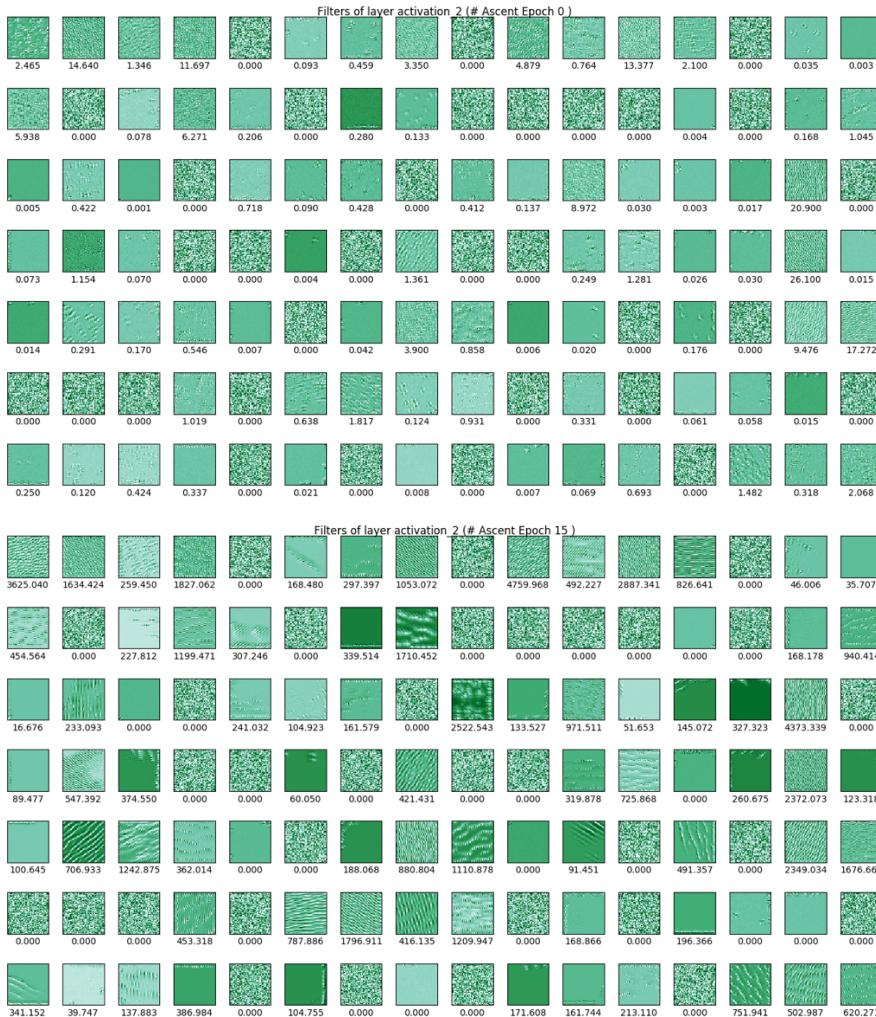
利用梯度遞增法，找出最能激活第一層 Activation 的 input:





可以發現經過 15 個 epochs 的 gradient ascent 之後，第一層 Activation Layer 大部分的 filter 都有被激活，而且各自有各自的激活對象，filter 有的專門尋找直線條紋，有的判斷斜線條紋，有的則是判斷點點的圖形，但仍會有部份 filter 的圖形仍是白噪音而無法被激活。

利用梯度遞增法，找出最能激活第二層 Activation 的 input:



可以發現第二層 Activation Layer 的 filter 同樣有特定的激活對象，但相較於第一層，激活的圖案多了點變化，而且 loss 也較第一層的 filter 大，但是沒被激活的 filter 也變得比較多。

至於每一層都有一些沒被激活的 filter，我猜想或許有可能是因為這個模型本身的複雜度不夠或者是還沒有被訓練得很好所導致的結果，而越往深層因為越難訓練，所以沒被訓練的 filter 數量也越多。

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

答：

根據和第一題同樣的 CNN 架構，並將 train.csv 的前 5000 筆當作 validation data，第 5001~7000 筆當作 unlabeled data，剩下的資料當作 training data 來實作 semi-supervised learning。

訓練過程中，同樣有使用 Image Data Generator 來增加 training data 的變化。

實作過程如下：

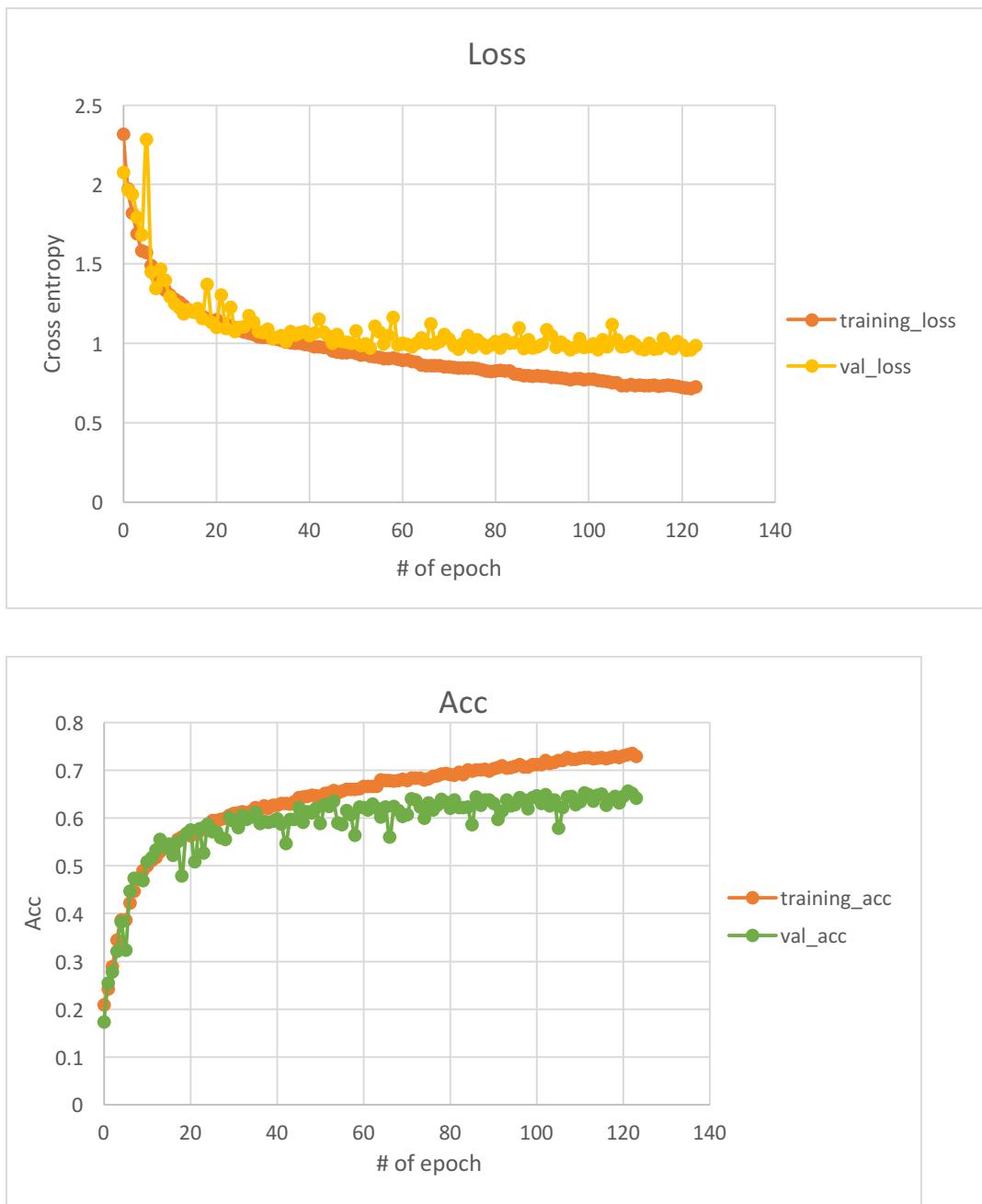
同樣先以未經過 Image Data Generator 處理過的資料簡單訓練 5 個 epochs 後，在開始用 Image Data Generator 產生的資料來訓練，經過 20 個 epochs 後，再用目前訓練出來的模型，去預測 unlabeled data，若預測出來某項 label 的機率高於 0.9，則將該資料標記上預測的標籤並加入 training data set 中。

```
model.fit(train_feature, train_label, validation_data=(valid_feature, valid_label), batch_size=batch, epochs=5, callbacks=[csv_logger0])
model.fit_generator(datagen.flow(train_feature, train_label, batch_size=batch),
                    steps_per_epoch = train_feature.shape[0]/batch,
                    epochs=20,
                    validation_data = (valid_feature, valid_label),
                    callbacks=[csv_logger1])
bound = 0.9
prob = model.predict(un_feature)
label = prob.argmax(1)
decision = prob.max(1) > bound
label = label[decision]
feature = un_feature[decision,:,:,:]
un_feature = np.delete(un_feature, decision, 0)
train_feature = np.concatenate((train_feature, feature), axis=0)
label = np_utils.to_categorical(label, num_classes)
train_label = np.concatenate((train_label, label), axis=0)
```

並再開始用 Image Data Generator 訓練 20 個 epochs，重複上述的過程 5 次。

```
for i in range(5):
    model.fit_generator(datagen.flow(train_feature, train_label, batch_size=batch),
                        steps_per_epoch = train_feature.shape[0]/batch,
                        epochs=20,
                        validation_data = (valid_feature, valid_label),
                        callbacks = [csv_logger2])
    )
if (len(un_feature)):
    prob = model.predict(un_feature)
    label = prob.argmax(1)
    decision = prob.max(1) > bound
    label = label[decision]
    feature = un_feature[decision,:,:,:]
    un_feature = np.delete(un_feature, decision, 0)
    train_feature = np.concatenate((train_feature, feature), axis=0)
    label = np_utils.to_categorical(label, num_classes)
    train_label = np.concatenate((train_label, label), axis=0)
```

訓練過程如下：



可以發現在 training data 方面，不論是準確率或是 loss，都有隨著訓練的 epoch 增加而獲得進步；在 validation data 方面，則約莫在第 60 個 epoch 左右，準確度和 loss 都開始上下浮動而不再進步，而最後上傳到 Kaggle 的分類結果，public 部分的準確度為 0.64781。

根據實作方式以及產生出來的訓練過程，我想也不難想像 model 在 training data 上的表現會一直進步，因為每次都會把預測機率超過一定比例的 label，將該筆資料標記為該 label，並丟進去 training data 中，無形中便增加了 model 有高機率能答對的資料項；至於 validation data 的表現並沒有比較好，反而還有一點退步，我想或多或少和訓練資料量的減少有關。在這個作業剛開始的時候，我曾經用

過別的模型，實作 semi-supervised training，而那時候的 unlabeled data 是直接用 test.csv 裡的資料，在同樣的模型架構下，根據最後丟到 Kaggle 上的 public 部分的準確度，可以發現實作 semi-supervised learning 會比原本直接下去訓練的模型的準確率要高出 1% ~ 2%左右。

[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%]