

學號：B04902045 系級：資工二 姓名：孫凡耘

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？

模型架構

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
activation_1 (Activation)	(None, 48, 48, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 24, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	73856
activation_2 (Activation)	(None, 24, 24, 128)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	147584
activation_3 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
conv2d_4 (Conv2D)	(None, 12, 12, 256)	295168
activation_4 (Activation)	(None, 12, 12, 256)	0
conv2d_5 (Conv2D)	(None, 12, 12, 256)	590080
activation_5 (Activation)	(None, 12, 12, 256)	0
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_3 (Dropout)	(None, 6, 6, 256)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 1024)	9438208
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
dropout_5 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 7)	7175
activation_6 (Activation)	(None, 7)	0
Total params: 11,602,311.0		

訓練過程

剛開始我依照cnn一般的架構:

1. filter 少-->多 (剛開始只看比較簡單的filter)
2. Use relu as activation function for cnn(to avoid gradient vanishing)
3. add fully feed network after cnn layers

但一直train得很慘, accuracy在training data上面就上不去。後來發現optimizer從adam換成adadelta或sgd就好很多了, 但很明顯train久一點就會overfitting (training data的accuracy一直提高但validation data的accuracy就會停止上升甚至減少)。To avoid overfitting, I added more dropout layer. 之後我 tune了一下filter的數量與調整了dropout layer的數量 (太多dropout layer也會train不起來), 最後決定使用大一點的network, 雖然要train比較久但結果比較好。基本上包含三層convolution layer(在一層convolution layer使用兩層filter的layer才會接著max pooling與dropout layer)與兩層fully feedforward network, 每層基本上接著一個dropout layer, activation 除了output使用softmax其他都使用relu.

我參考這個論文中的作法, 做image cropping與翻轉, 把data變成12倍在拿去train.

<https://arxiv.org/pdf/1611.04251.pdf>

至於對於image的preprocessing, 我把所有pixel的值都線性的移到[0,1]之間 (直接除255)

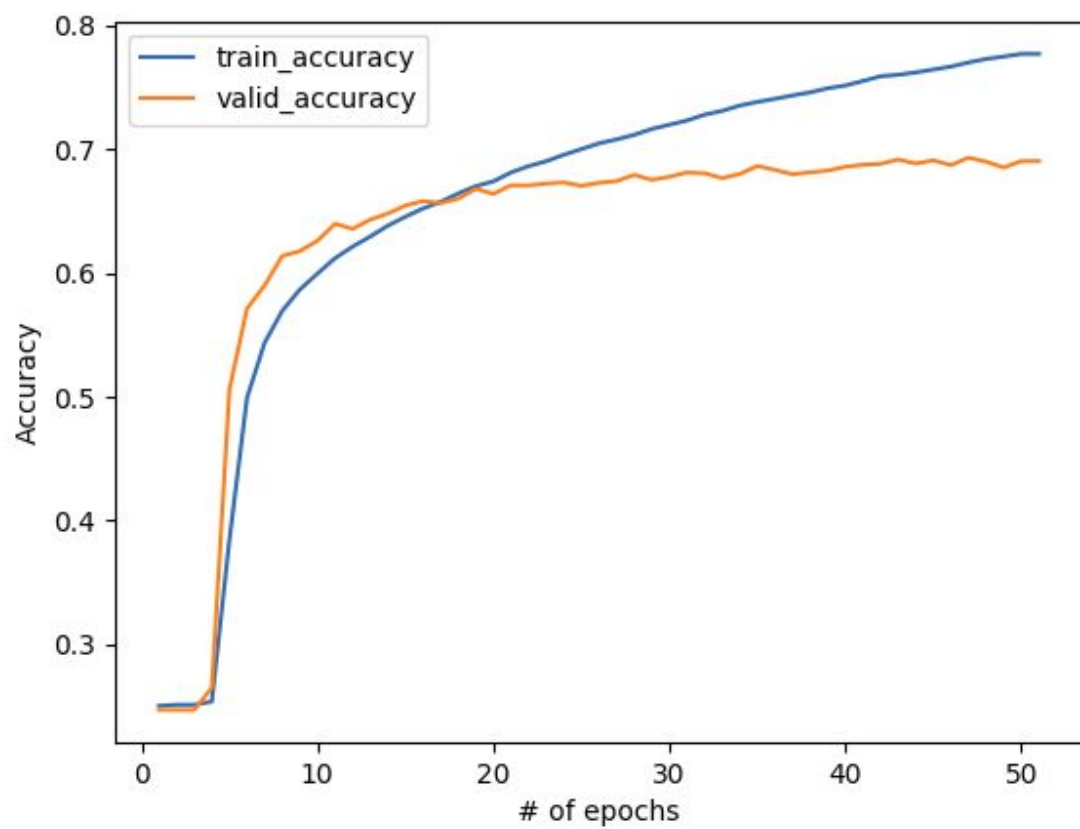
SGD(lr=0.005, decay=0.00001, momentum=0.9)

loss='categorical_crossentropy', batch_size=50

Tuned parameters:

- learning rate of SGD
- filter 的數量
- dropout layer的dropout rate(0.5)

Tune 的方法基本上都是先查查看一般別人用的參數是怎麼樣, 然後再做一些trial and error



準確率 Best accuracy on public data: 0.69072

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？

Model Structure

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 1024)	2360320
dropout_1 (Dropout)	(None, 1024)	0
activation_1 (Activation)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
activation_2 (Activation)	(None, 1024)	0
dense_3 (Dense)	(None, 1024)	1049600
dropout_3 (Dropout)	(None, 1024)	0
activation_3 (Activation)	(None, 1024)	0
dense_4 (Dense)	(None, 1024)	1049600
dropout_4 (Dropout)	(None, 1024)	0
activation_4 (Activation)	(None, 1024)	0
dense_5 (Dense)	(None, 1024)	1049600
dropout_5 (Dropout)	(None, 1024)	0
activation_5 (Activation)	(None, 1024)	0
dense_6 (Dense)	(None, 1024)	1049600
dropout_6 (Dropout)	(None, 1024)	0
activation_6 (Activation)	(None, 1024)	0

dense_8 (Dense)	(None, 1024)	1049600
dropout_8 (Dropout)	(None, 1024)	0
activation_8 (Activation)	(None, 1024)	0
dense_9 (Dense)	(None, 1024)	1049600
dropout_9 (Dropout)	(None, 1024)	0
activation_9 (Activation)	(None, 1024)	0
dense_10 (Dense)	(None, 1024)	1049600
dropout_10 (Dropout)	(None, 1024)	0
activation_10 (Activation)	(None, 1024)	0
dense_11 (Dense)	(None, 1024)	1049600
dropout_11 (Dropout)	(None, 1024)	0
activation_11 (Activation)	(None, 1024)	0
dense_12 (Dense)	(None, 7)	7175
activation_12 (Activation)	(None, 7)	0
=====		
Total params: 12,863,495.0		
Trainable params: 12,863,495.0		
Non-trainable params: 0.0		

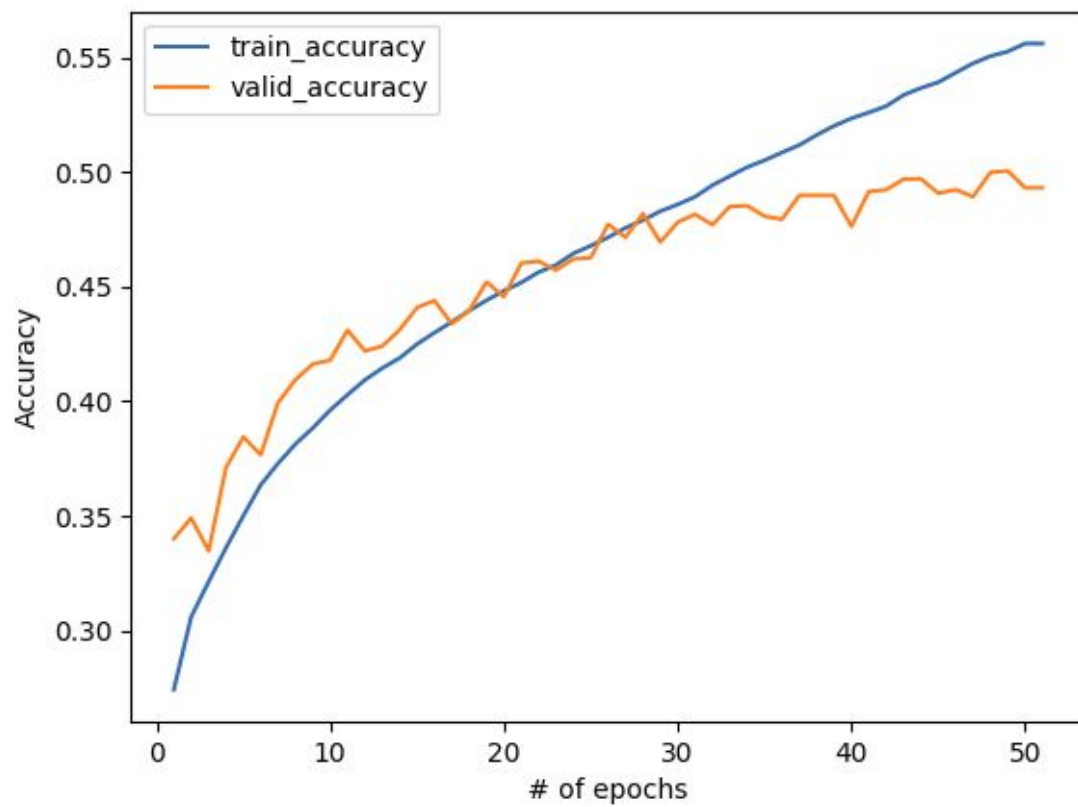
Number of trainable params: 12, 836, 495

Training Procedure

我疊了10層的hidden layer, 每層為regular densely-connected NN layer with units=1024(I learned that people nowadays tend to give every hidden layer the same unit. If the number of units are more than needed, the network will learn that too.), 參數量與上述的CNN差不多

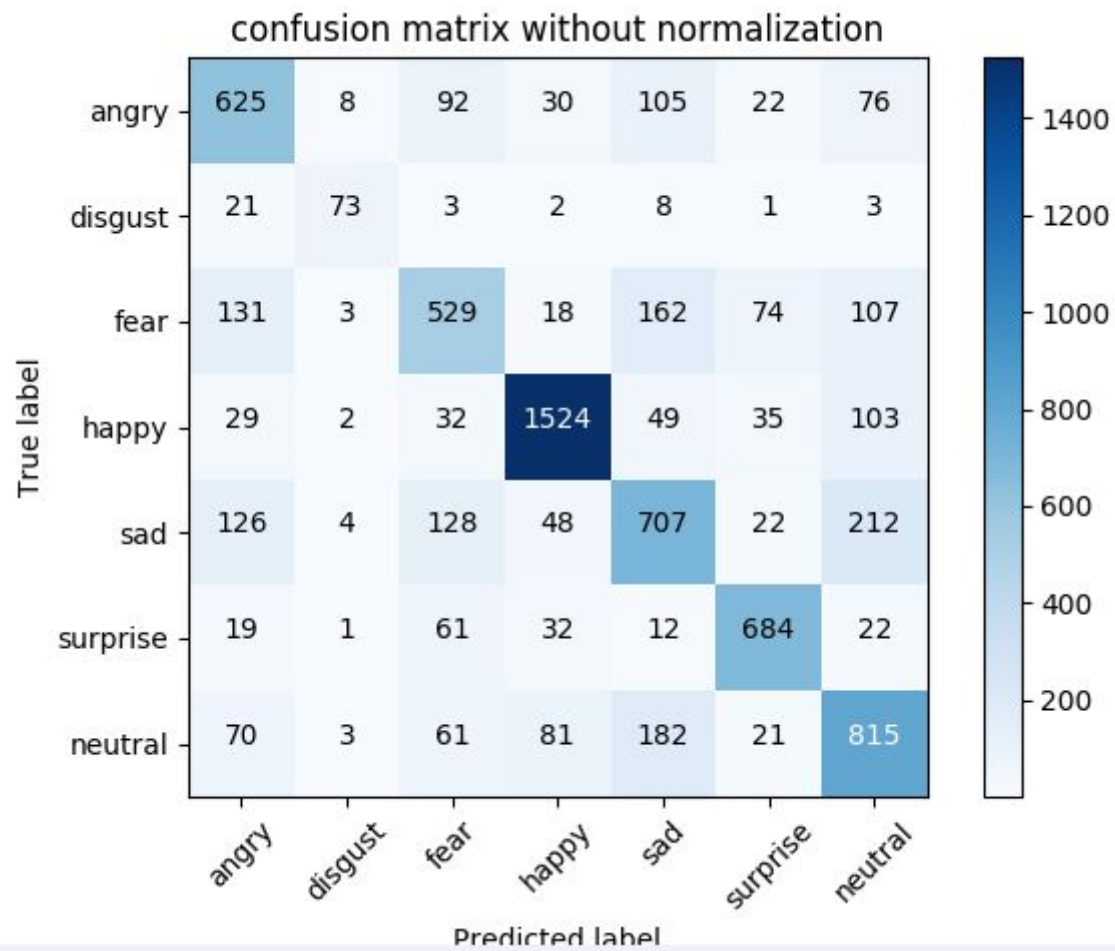
(皆為一千萬左右), 除了network的架構以外與epoch次數不同以外 (I did early stopping on dnn to avoid overfitting), 其餘的方法完全一樣(activation 除了output使用softmax其他都使用 relu to avoid gradient vanishing problem)。

可以發現再相同參數的情況下, cnn再這個task上的performance明顯比較好。而且dnn也train得比較慢 (cnn在10個epoch以內就到達0.6以上的accuracy, 30個epoch就接近best performance), dnn上升的很慢, 50個epoch左右才能到達model最好的performance。



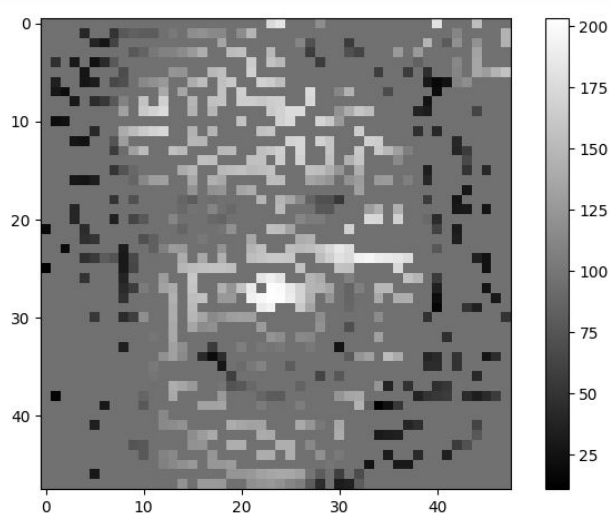
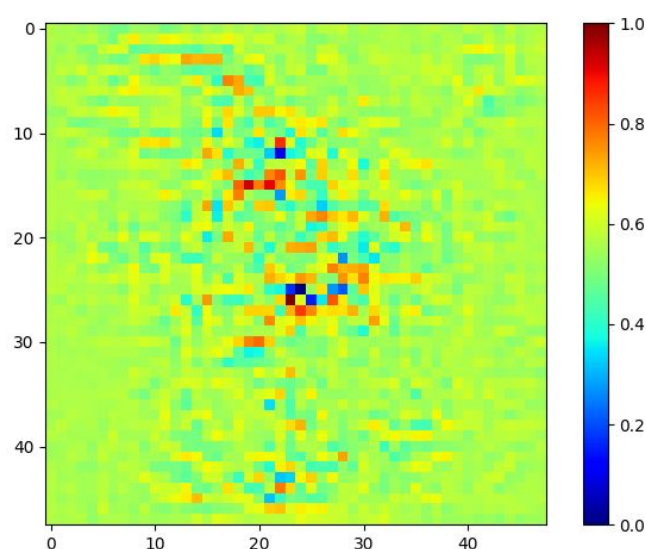
準確率 Best accuracy on public data: 0.52299

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]



- pictures labeled disgust are likely to be classified as angry
- pictures labeled fear are likely to be classified as angry and sad
- pictures labeled sad are likely to be classified as neutral
- pictures labeled neutral are likely to be classified as sad

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？



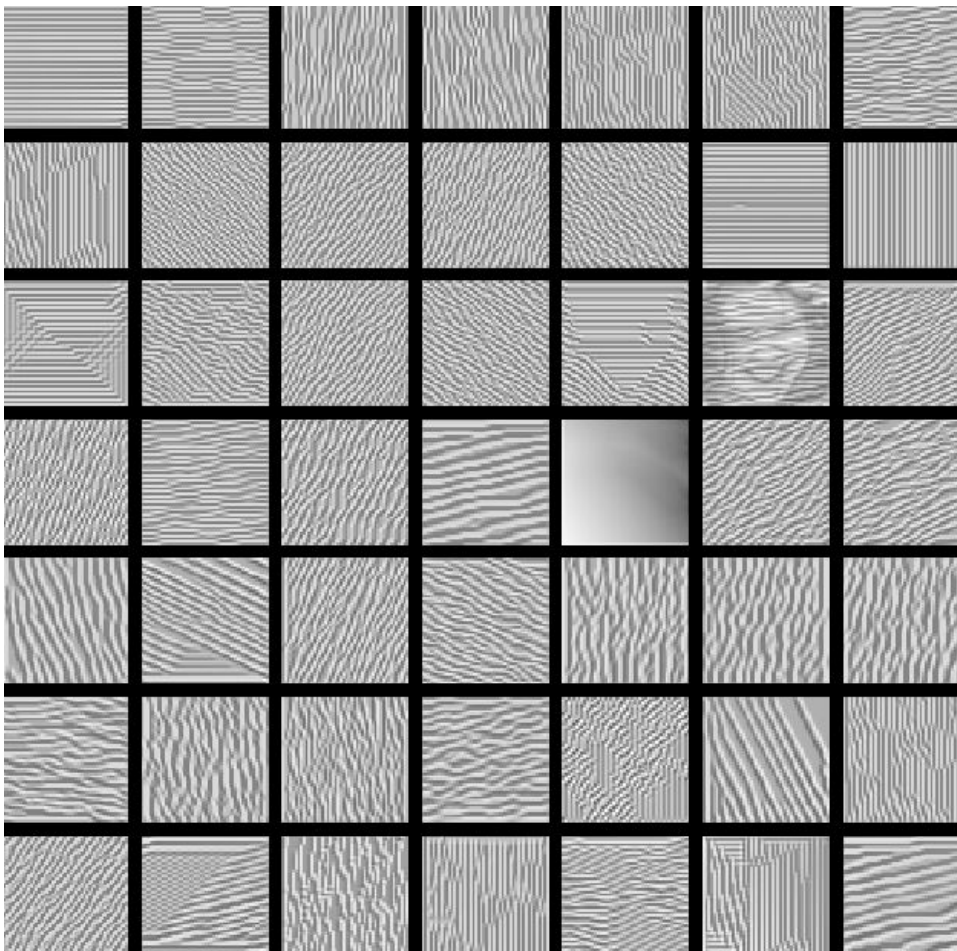
大致上在這張圖上，眼睛跟鼻子是圖片的重點區域。在下面那張圖中可以看到雖然牙齒都不太高，但唇型大致上並沒有被mask掉，所以估計唇型也是一個high level feature(但是很不明顯的因為從heatmap中看不出唇型，不太清楚是什麼原因)。

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。



Filters of layer activation_1 (# Ascent steps: 100)

利用梯度遞增法, 找出最能激活特定filter的圖片(從白噪音開始)




```

for i in range(20):

    loss_value, grads_value = iterate([input_img_data])

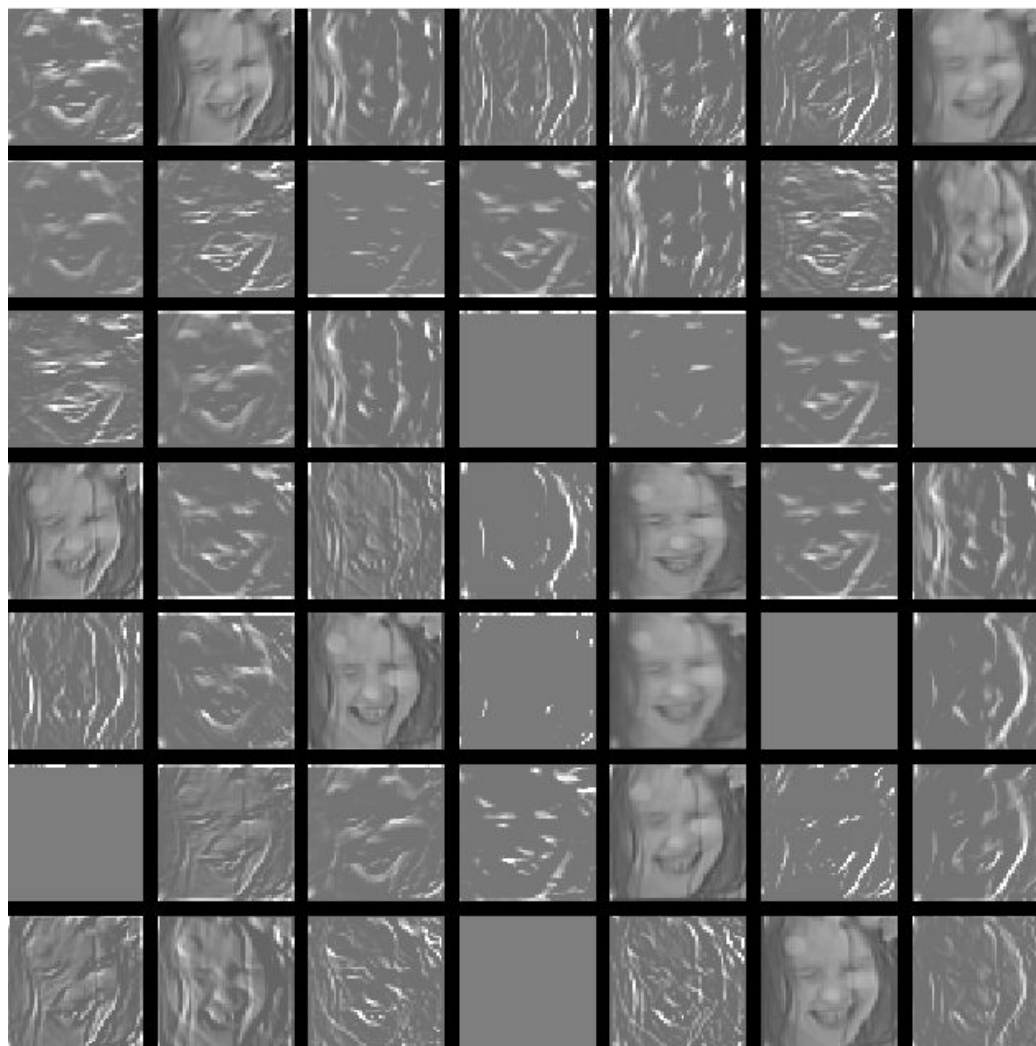
    input_img_data += grads_value * step

return input_img_data

```

Output of layer activation_1

給定輸入圖片，取出特定層的輸出



基本上可以從第1張圖中觀察到，第1層的filter主要是偵測簡單的feature像是點，線，不同粗細與不同角度。能在第一層看出臉形的大概只有一個（圖中的49個是從64個裡面選出gradient ascent 之後loss比較高的）。同一層的output則很多還能看出原圖的樣子，但也有很多圖幾乎式灰色一片看不出任何東西了，我猜應該是因為我第1層用了64個filter，但實際上也沒需要那麼多，所以可能model在自己learn完之後，學出不少filter是多餘的（這張圖裡49個是從64個裡面random取的）。

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

My full code at ML2017/hw3/semi_supervised/main.py

Remove some label in training data as **unlabeled data**

```
unlabeled_size = 10000
global unlabeled_x
unlabeled_x = x_test[:unlabeled_size]

x_test = x_test[unlabeled_size:]
y_test = y_test[unlabeled_size:]
```

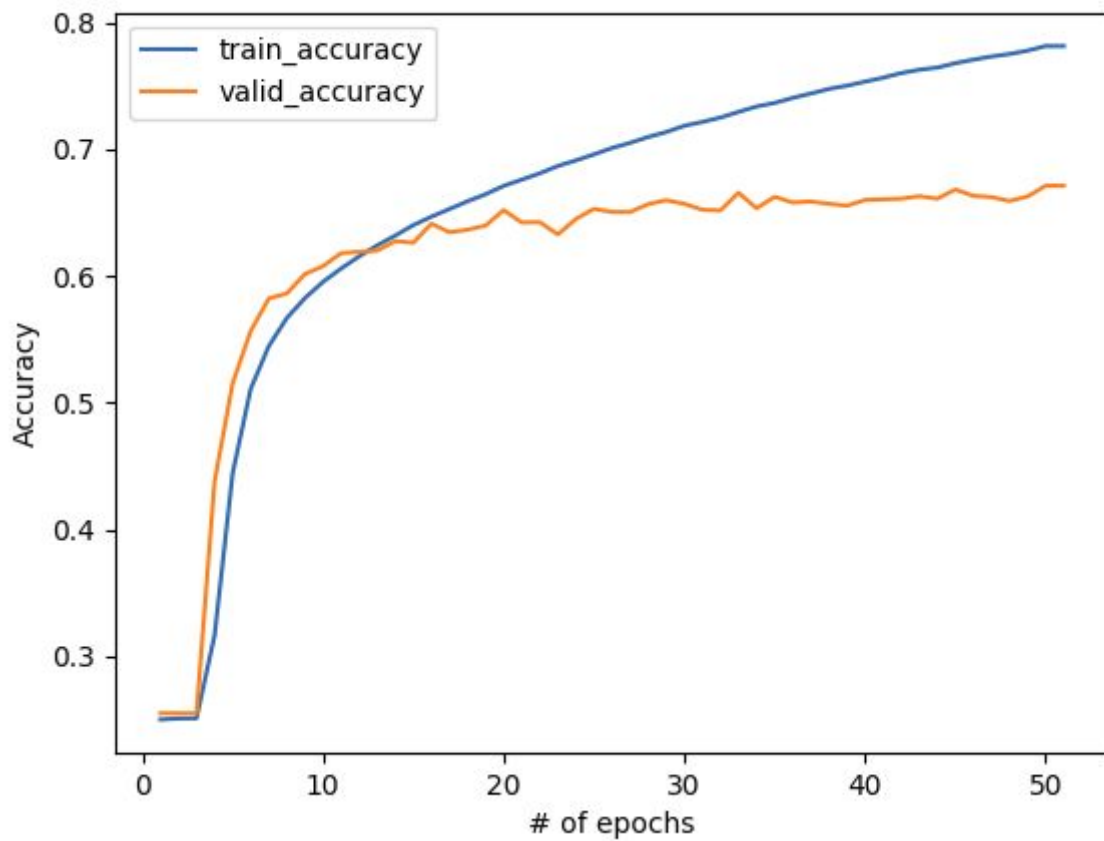
Implement semi-supervised learning algorithm(semi supervised by entropy-based minimization, or regularization)

```
def semi_loss( y_test, y_pred ):
    labeled_loss = K.categorical_crossentropy(y_test, y_pred)
    unlabeled_pred = model.predict( unlabeled_x ).flatten()
    unlabeled_loss = sum(np.log( unlabeled_pred ) * unlabeled_pred)
    return labeled_loss + lamda*unlabeled_loss
```

compile my model with semi_loss

```
#sgd = SGD(lr=0.005, decay=0.00001, momentum=0.9)
sgd = SGD(lr=0.005, decay=0.00001, momentum=0.9)
model.compile(loss=semi_loss, optimizer=sgd, metrics=['accuracy'])
```


Result



準確率 Best accuracy on public data: **0.67456**

[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%]

