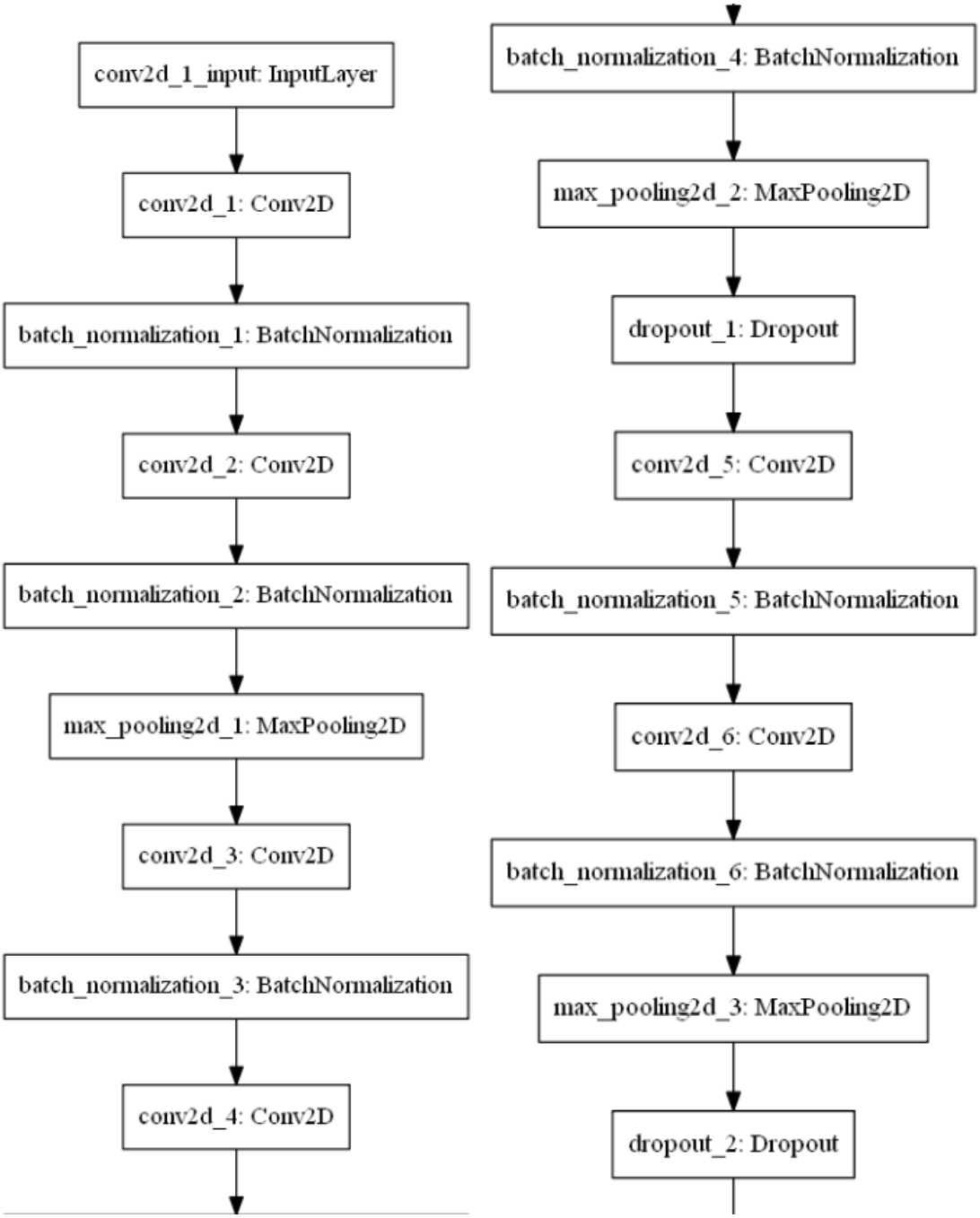
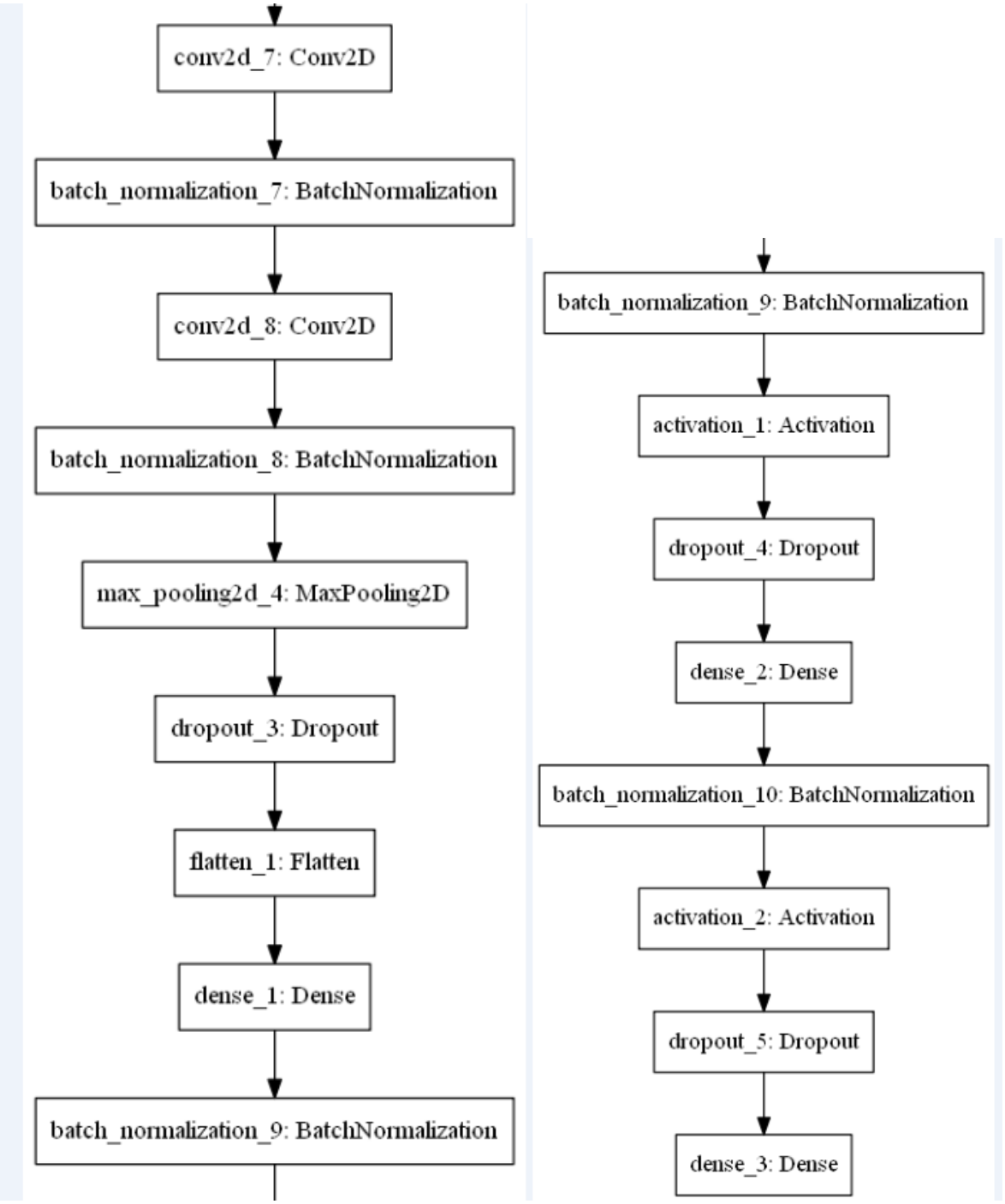
學號：R04945022 系級： 生醫電資二 姓名：張君澤

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：(model 實在是太長了，由左至右，由上至下，圖片以下說明)





本實做模型參考了 keras 官網的 cifar10 的範本，並不斷加深他的 CNN 從輸入的 64 層、128 層…慢慢倍增到 512 層，並且最後使用兩個 1024 層的 fully connected layer 進行最後的判斷。CNN 在疊超過 512 層後對結果並沒有顯著的影響，而且超過 512 層的 CNN 我在自己的電腦上無法讀取自己 train 好的 model(可能是顯卡記憶體不夠)，因此最後我刪掉了一層 512 的 layer 即可讀取。本 model 在未經任何調校之前在 Kaggle 上的分數大約落在 0.6268 分左右，因此添加了 keras 內訓練影像時的左右翻轉的功能來增加訓練強度(如下紅字處)：

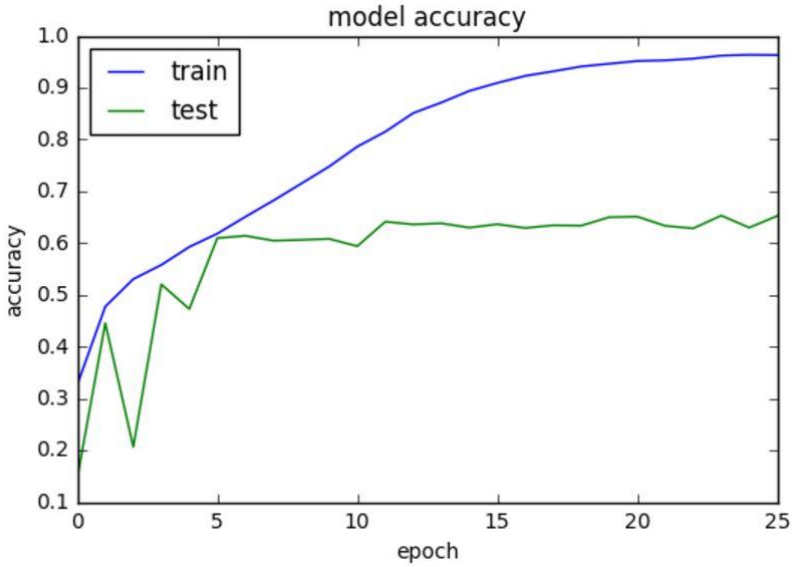
datagenerator = ImageDataGenerator(

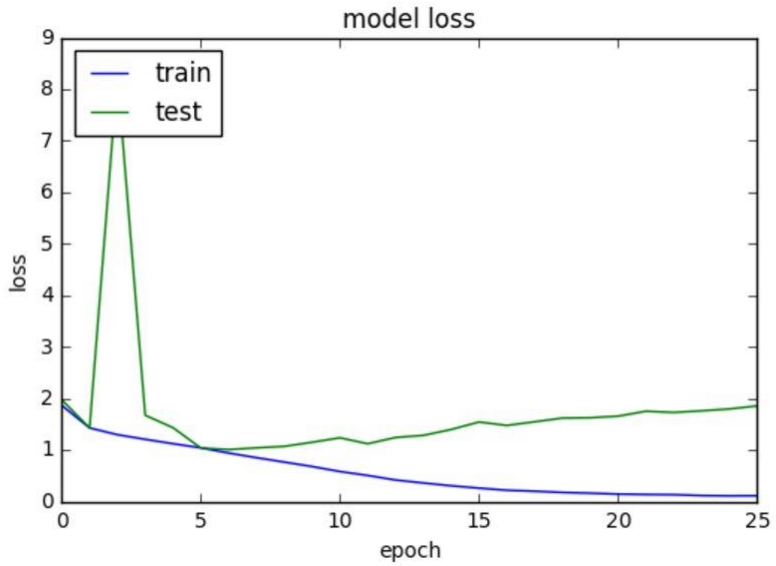
width_shift_range=0.1

height_shift_range=0.1,

horizontal_flip=True, # randomly flip images)

以下為本 Training model 在 epoch27 次的訓練過程，經由觀察發現，在超過 25 次 epoch 之後，validation accuracy 大概就會落在 0.67 左右上上下下，與我 kaggle 上最好的結果 0.68041 相差不遠。





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？

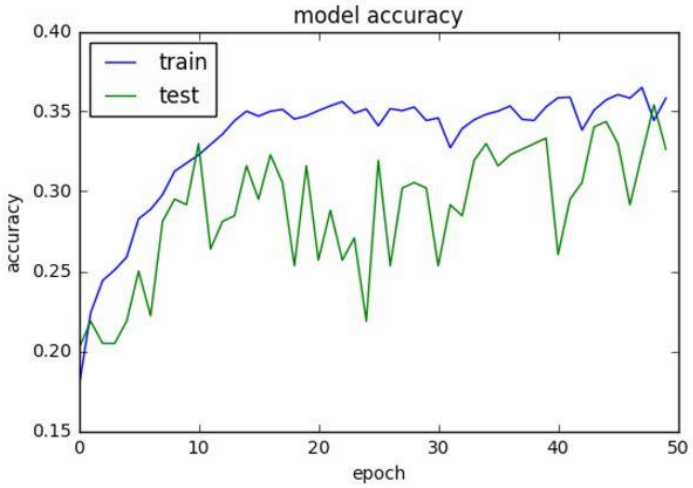
答：

我的 CNN model 總參數約 8 百萬個，我實做的 DNN model 總參數為 664 萬個(如下圖)：

Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 1288)	2968840
batch_normalization_25 (Batc	(None, 1288)	5152
dropout_25 (Dropout)	(None, 1288)	0
dense_28 (Dense)	(None, 1024)	1319936
batch_normalization_26 (Batc	(None, 1024)	4096
dropout_26 (Dropout)	(None, 1024)	0
dense_29 (Dense)	(None, 784)	803600
batch_normalization_27 (Batc	(None, 784)	3136
dropout_27 (Dropout)	(None, 784)	0
dense_30 (Dense)	(None, 666)	522810
batch_normalization_28 (Batc	(None, 666)	2664
dropout_28 (Dropout)	(None, 666)	0
dense_31 (Dense)	(None, 512)	341504
batch_normalization_29 (Batc	(None, 512)	2048
dropout_29 (Dropout)	(None, 512)	0
dense_32 (Dense)	(None, 512)	262656
batch_normalization_30 (Batc	(None, 512)	2048
dropout_30 (Dropout)	(None, 512)	0
dense_33 (Dense)	(None, 384)	196992
batch_normalization_31 (Batc	(None, 384)	1536
dropout_31 (Dropout)	(None, 384)	0
dense_34 (Dense)	(None, 256)	98560
batch_normalization_32 (Batc	(None, 256)	1024
dropout_32 (Dropout)	(None, 256)	0
dense_35 (Dense)	(None, 256)	65792
batch_normalization_33 (Batc	(None, 256)	1024
dropout_33 (Dropout)	(None, 256)	0
dense_36 (Dense)	(None, 128)	32896
batch_normalization_34 (Batc	(None, 128)	512
dropout_34 (Dropout)	(None, 128)	0
dense_37 (Dense)	(None, 64)	8256
batch_normalization_35 (Batc	(None, 64)	256
dropout_35 (Dropout)	(None, 64)	0
dense_38 (Dense)	(None, 32)	2080
batch_normalization_36 (Batc	(None, 32)	128
dropout_36 (Dropout)	(None, 32)	0
dense_39 (Dense)	(None, 16)	528
batch_normalization_37 (Batc	(None, 16)	64
dropout_37 (Dropout)	(None, 16)	0
dense_40 (Dense)	(None, 7)	119

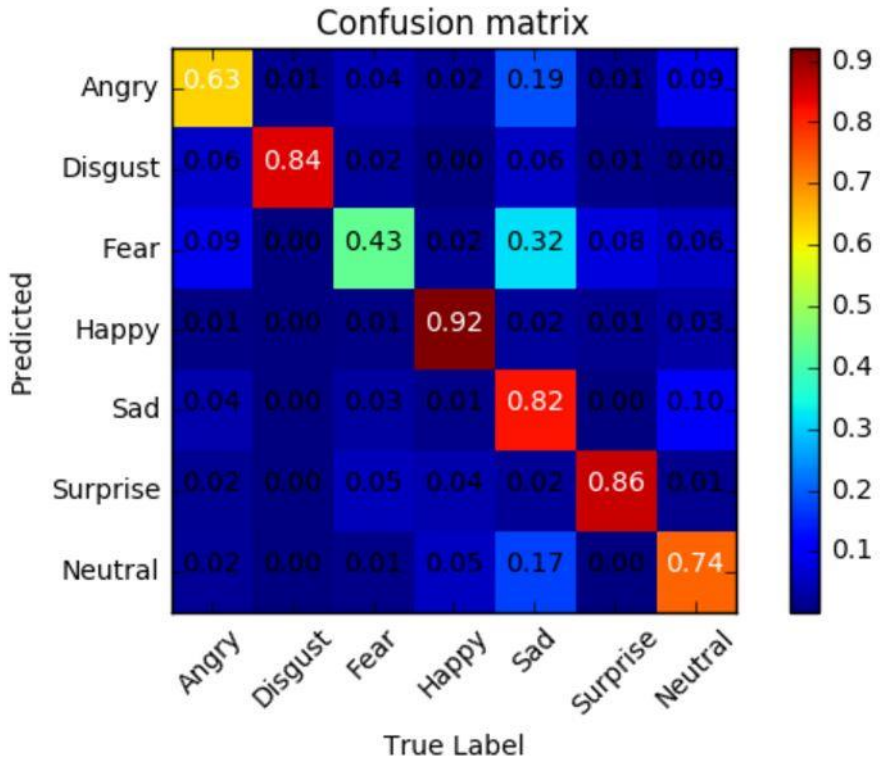
Total params: 6,648,257
Trainable params: 6,636,413
Non-trainable params: 11,844

訓練準確度如下圖，在 training 時大概就只有 0.35 的準確度，上傳至 kaggle 上的成績為 0.3073 只比亂猜的 0.148 多了一倍。另外發現一件事就是我如果只有使用 20 萬個參數左右的 model，上傳至 kaggle 的成績還有 0.4254，比這個六百多萬參數的 model 還要高。分析了一下原因，個人覺得 CNN 效果好很多的原因是因為 CNN 透過 3*3 或者 5*5 的 filter 來過濾出影像的 edge，很多個 edge 慢慢描繪出形狀，並且在後來愈來愈多層的情況可以慢慢組合出合理的部分影像，比如說眉毛的形狀或是嘴角上揚或是嘴巴張很大。透過這些組合出來的影像電腦可以更精確的分類影像，因此 CNN 很適合拿來做影像分類，反觀如果從頭到尾都是 fully connected network，那麼都是用在某點的 pixel 是否為某個值來判斷，其效果有限而且參數多不一定效果就好。



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]

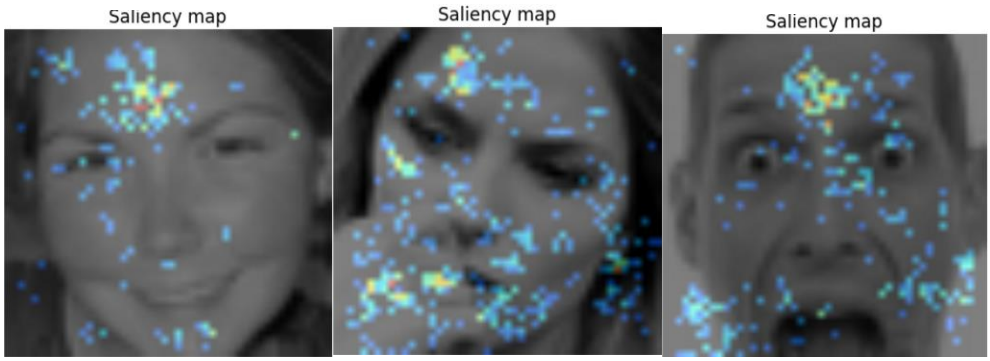
答：我的 confusion matrix 如下：

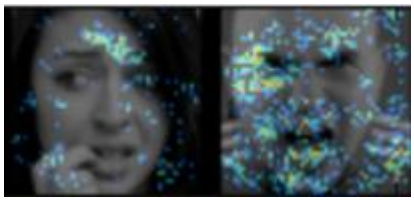


經由以上的結果推測，恐懼(Fear)時的表情最容易跟難過(Sad)搞混，其他比較容易搞混的表情有 Angry→Sad， Neutral→Sad，可能人類悲傷的表情比較多變，電腦只要分辨不出來的奇怪表情一律歸類為 Sad 囉！

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： 利用 Keras_vis 呈現出與影像疊合的 heatmap。



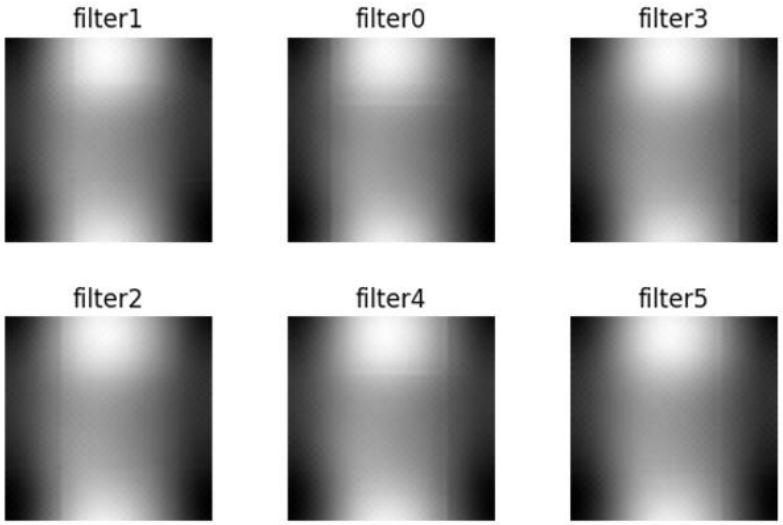


由 Saliency 的 Heatmap 觀察了二十多張圖片，發現了大部分影響電腦決定最後判斷結果的區域常常落在額頭還有嘴巴下方，合理推測是額頭是生氣、困惑、中立……等變化最明顯的區域，而嘴巴周圍也是正常人類用來判斷表情的經典區域。

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： 我觀察了本張圖片

並利用了 keras_vis 套件查看了最容易被 activate 的 filter，該套件輸出的結果即是以 gradient ascent 的方式呈現，因此我抓取了前六個 filter 的 output 並輸出成圖片，發現結果跟第四題的觀察很類似，都是以上面額頭處，與嘴巴下方處為 activation 最明顯的地方。下圖為影響最顯著的六個 filter。



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

參考範例寫了一份簡單的 auto-encoder，將 training data 的前半變成 unlabeled data，效果只比亂猜好一點點(val_accuracy 0.225)，我想這部分可能要好好讀個幾分 paper 才有辦法實作得很完整囉!

In [19]:
```
input_img = Input(shape=(48,48,1))
x = Conv2D(32,(3,3), padding='same', activation='relu')(input_img)
x = Conv2D(32,(3,3), padding='same', activation='relu')(x)
x = MaxPooling2D((2,2))(x)
x = Conv2D(16,(3,3), padding='same', activation='relu')(input_img)
x = Conv2D(16,(3,3), padding='same', activation='relu')(x)
x = MaxPooling2D((2,2))(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
# encoder create
x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
```

[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？