學號：B06502162　系級：　電機三　姓名：陸亮宇

1. (1%) 請說明這次使用的model架構，包含各層維度及連接方式。

```
    self.cnn = nn.Sequential(
        nn.Conv2d(1, 32, kernel_size=5, padding=2), #32*48*48
        nn.LeakyReLU(negative_slope=0.05),
        nn.BatchNorm2d(32),
        nn.MaxPool2d(2), #32*24*24

        nn.Conv2d(32, 64, kernel_size=3, padding=1),#64*24*24
        nn.LeakyReLU(negative_slope=0.05),
        nn.BatchNorm2d(64),
        nn.MaxPool2d(2), #64*12*12

        nn.Conv2d(64, 128, kernel_size=3,padding=1),#128*12*12
        nn.LeakyReLU(negative_slope=0.05),
        nn.BatchNorm2d(128),
        nn.MaxPool2d(2), #128*6*6

        nn.Conv2d(128, 128, kernel_size=3,padding=1), #128*6*6
        nn.LeakyReLU(negative_slope=0.05),
        nn.BatchNorm2d(128),
        nn.MaxPool2d(2),  #128*3*3




    )
    self.fc = nn.Sequential(
        nn.Linear(3*3*128, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 7)
    )
```

這次的model架構為利用基本CNN模型train出多個model，每個模型差別只在於利用random seed來控制training set 和 validation set選擇，並且利用每個模型去共同預測testing data，然後選擇預測出現次數最多的選項。

而CNN架構如上圖，分成四個segment，第一個segment input gray image(1*48*48)，然後padding=2填0，kernel_size選擇3，output channel 選擇為64，所以為64*24*24，然後activate function選擇為LeakyReLU斜率0.05，然後做BatchNorm2d，再做max pooling，所以第一個segment output為64*12*12。

第二個segment 輸入第一個segment output(64*12*12)，然後連接方式相同，但改變output channel = 64，kernel_size = 3，padding = 1，各層維度如上圖所示。

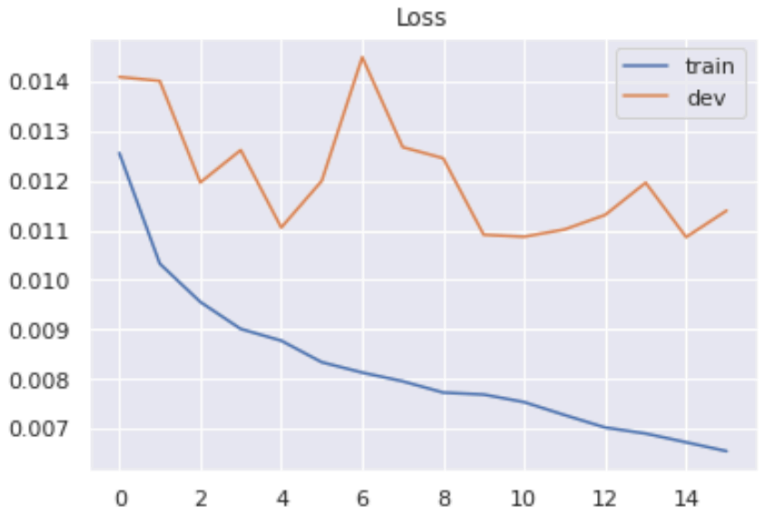
第三/四個segment連接架構相同，如上如圖所示，最後output為128*3*3，然後再攤開通過linear network、ReLU、做BatchNorm1d，最後output layer為linear network的classification。

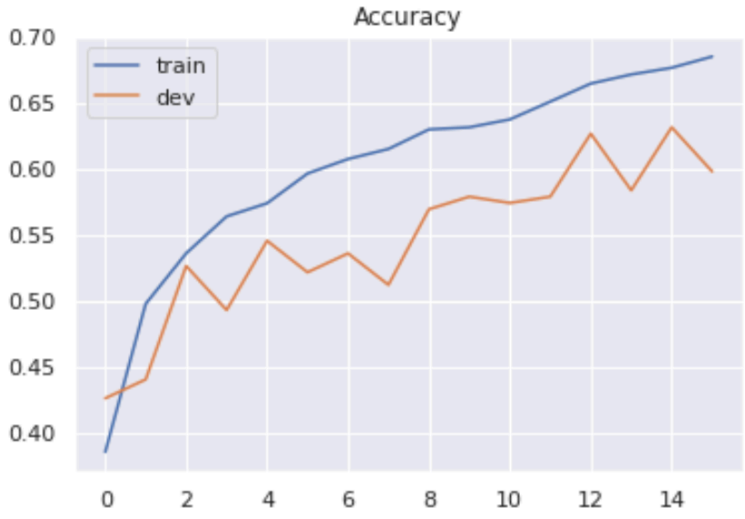
上述為這次model使用的CNN架構，我嘗試了使用9個model共同預測，和15個model共同預測，結果如表格：

	9 model	15 model
public	0.68013	0.68375
private	0.68682	0.68849

可以發現無論在public或是private上，15個model的共同預測皆比9個model表現來的好，符合我們假設。

2. (1%) 請附上model的training/validation history (loss and accuracy)。





0
[001/050] 10.95 sec(s) Train Acc: 0.406175 Loss: 0.012184 | Val Acc: 0.444976 loss: 0.012994
1
[002/050] 10.49 sec(s) Train Acc: 0.502667 Loss: 0.010198 | Val Acc: 0.473684 loss: 0.012798
2
[003/050] 10.62 sec(s) Train Acc: 0.532947 Loss: 0.009625 | Val Acc: 0.488038 loss: 0.012851
3
[004/050] 10.66 sec(s) Train Acc: 0.560386 Loss: 0.009134 | Val Acc: 0.526316 loss: 0.011955
4
[005/050] 10.49 sec(s) Train Acc: 0.574667 Loss: 0.008779 | Val Acc: 0.540670 loss: 0.012087
5
[006/050] 10.52 sec(s) Train Acc: 0.592982 Loss: 0.008421 | Val Acc: 0.564593 loss: 0.011932
6
[007/050] 10.57 sec(s) Train Acc: 0.605579 Loss: 0.008184 | Val Acc: 0.588517 loss: 0.011938
7
[008/050] 10.77 sec(s) Train Acc: 0.614351 Loss: 0.008019 | Val Acc: 0.555024 loss: 0.012091
8
[009/050] 10.92 sec(s) Train Acc: 0.625509 Loss: 0.007785 | Val Acc: 0.564593 loss: 0.011508
9
[010/050] 10.54 sec(s) Train Acc: 0.632175 Loss: 0.007597 | Val Acc: 0.559809 loss: 0.011179
10
[011/050] 10.85 sec(s) Train Acc: 0.641509 Loss: 0.007466 | Val Acc: 0.593301 loss: 0.012054
11
[012/050] 10.76 sec(s) Train Acc: 0.652667 Loss: 0.007230 | Val Acc: 0.559809 loss: 0.011331
12
[013/050] 10.63 sec(s) Train Acc: 0.655789 Loss: 0.007165 | Val Acc: 0.559809 loss: 0.012032
13
[014/050] 10.70 sec(s) Train Acc: 0.661684 Loss: 0.007069 | Val Acc: 0.555024 loss: 0.012705
14
[015/050] 10.67 sec(s) Train Acc: 0.677404 Loss: 0.006782 | Val Acc: 0.545455 loss: 0.012993
15
[016/050] 10.80 sec(s) Train Acc: 0.682807 Loss: 0.006629 | Val Acc: 0.540670 loss: 0.012462
16
[017/050] 10.75 sec(s) Train Acc: 0.692246 Loss: 0.006449 | Val Acc: 0.574163 loss: 0.012347
17
[018/050] 10.64 sec(s) Train Acc: 0.694947 Loss: 0.006342 | Val Acc: 0.569378 loss: 0.011888
18
[019/050] 10.71 sec(s) Train Acc: 0.703228 Loss: 0.006243 | Val Acc: 0.516746 loss: 0.012298
19
[020/050] 10.77 sec(s) Train Acc: 0.703825 Loss: 0.006191 | Val Acc: 0.545455 loss: 0.013228
20
[021/050] 10.51 sec(s) Train Acc: 0.699930 Loss: 0.006282 | Val Acc: 0.555024 loss: 0.012754
21
[022/050] 10.64 sec(s) Train Acc: 0.724386 Loss: 0.005794 | Val Acc: 0.593301 loss: 0.012772
22
[023/050] 10.62 sec(s) Train Acc: 0.734807 Loss: 0.005580 | Val Acc: 0.578947 loss: 0.012908

```
21
[022/050] 10.64 sec(s) Train Acc: 0.724386 Loss: 0.005794 | Val Acc: 0.593301 loss: 0.012772
22
[023/050] 10.62 sec(s) Train Acc: 0.734807 Loss: 0.005580 | Val Acc: 0.578947 loss: 0.012908
23
[024/050] 10.57 sec(s) Train Acc: 0.741930 Loss: 0.005474 | Val Acc: 0.569378 loss: 0.012775
24
[025/050] 10.71 sec(s) Train Acc: 0.746947 Loss: 0.005349 | Val Acc: 0.564593 loss: 0.013930
25
[026/050] 10.77 sec(s) Train Acc: 0.750526 Loss: 0.005272 | Val Acc: 0.569378 loss: 0.013284
26
[027/050] 10.80 sec(s) Train Acc: 0.756982 Loss: 0.005166 | Val Acc: 0.564593 loss: 0.014127
27
[028/050] 10.58 sec(s) Train Acc: 0.768211 Loss: 0.004962 | Val Acc: 0.521531 loss: 0.016607
28
[029/050] 10.52 sec(s) Train Acc: 0.770316 Loss: 0.004867 | Val Acc: 0.593301 loss: 0.013755
29
[030/050] 10.66 sec(s) Train Acc: 0.776000 Loss: 0.004744 | Val Acc: 0.588517 loss: 0.014904
30
[031/050] 10.73 sec(s) Train Acc: 0.785298 Loss: 0.004576 | Val Acc: 0.574163 loss: 0.013995
31
[032/050] 10.67 sec(s) Train Acc: 0.786035 Loss: 0.004558 | Val Acc: 0.607656 loss: 0.013869
32
[033/050] 10.74 sec(s) Train Acc: 0.787158 Loss: 0.004500 | Val Acc: 0.598086 loss: 0.013710
33
[034/050] 10.87 sec(s) Train Acc: 0.803298 Loss: 0.004261 | Val Acc: 0.612440 loss: 0.014469
34
[035/050] 10.68 sec(s) Train Acc: 0.799825 Loss: 0.004221 | Val Acc: 0.569378 loss: 0.017233
35
[036/050] 10.62 sec(s) Train Acc: 0.809404 Loss: 0.004088 | Val Acc: 0.598086 loss: 0.015270
36
[037/050] 10.77 sec(s) Train Acc: 0.813158 Loss: 0.003977 | Val Acc: 0.593301 loss: 0.015057
37
[038/050] 11.01 sec(s) Train Acc: 0.812702 Loss: 0.003964 | Val Acc: 0.607656 loss: 0.015357
38
[039/050] 10.78 sec(s) Train Acc: 0.822596 Loss: 0.003794 | Val Acc: 0.598086 loss: 0.015726
39
[040/050] 10.84 sec(s) Train Acc: 0.819789 Loss: 0.003844 | Val Acc: 0.588517 loss: 0.015350
40
[041/050] 10.83 sec(s) Train Acc: 0.826421 Loss: 0.003726 | Val Acc: 0.574163 loss: 0.014967
41
[042/050] 10.72 sec(s) Train Acc: 0.834772 Loss: 0.003559 | Val Acc: 0.578947 loss: 0.018282
42
```

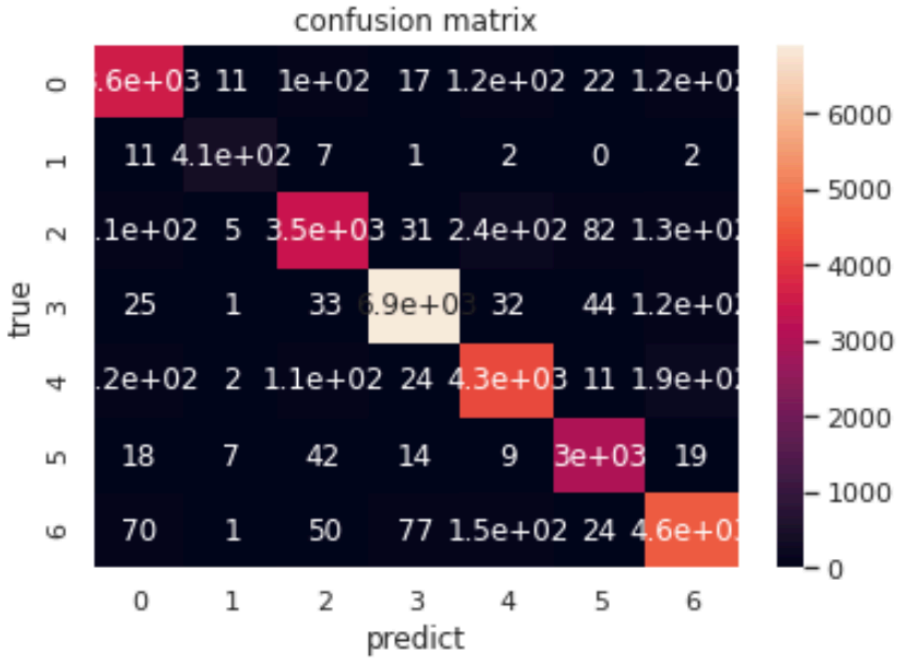
```
41
[042/050] 10.72 sec(s) Train Acc: 0.834772 Loss: 0.003559 | Val Acc: 0.578947 loss: 0.018282
42
[043/050] 10.62 sec(s) Train Acc: 0.836596 Loss: 0.003504 | Val Acc: 0.540670 loss: 0.020175
43
[044/050] 10.67 sec(s) Train Acc: 0.840140 Loss: 0.003421 | Val Acc: 0.578947 loss: 0.015896
44
[045/050] 10.83 sec(s) Train Acc: 0.843368 Loss: 0.003356 | Val Acc: 0.622010 loss: 0.016938
45
[046/050] 10.78 sec(s) Train Acc: 0.846982 Loss: 0.003263 | Val Acc: 0.583732 loss: 0.017525
46
[047/050] 10.91 sec(s) Train Acc: 0.847509 Loss: 0.003224 | Val Acc: 0.607656 loss: 0.017967
47
[048/050] 10.85 sec(s) Train Acc: 0.856667 Loss: 0.003138 | Val Acc: 0.574163 loss: 0.016613
48
[049/050] 10.80 sec(s) Train Acc: 0.854912 Loss: 0.003103 | Val Acc: 0.593301 loss: 0.016652
49
[050/050] 10.90 sec(s) Train Acc: 0.861298 Loss: 0.003009 | Val Acc: 0.583732 loss: 0.017299
```


我的model train了50個epochs來觀察loss和accuracy，從validation set的正確率可以發現，在epoch 10附近就到了準確率上限，然後來回震盪，我試過改變各式各樣的

參數還有調整CNN架構(改變layer)，發現進步皆有限，所以猜測單一個CNN model是有極限的，所以我嘗試train多個相同架構model共同預測，發現準確率大幅提升。

3. (1%) 畫出confusion matrix分析哪些類別的圖片容易使model搞混，並簡單說明。
(ref: https://en.wikipedia.org/wiki/Confusion_matrix)
confusion matrix:

```
[[3569   11  103   17  125   22  117]
 [  11  409    7    1    2    0    2]
 [ 111    5 3475   31  236   82  127]
 [  25    1   33 6910   32   44  119]
 [ 122    2  111   24 4326   11  193]
 [  18    7   42   14    9 3043   19]
 [  70    1   50   77  150   24 4560]]
```

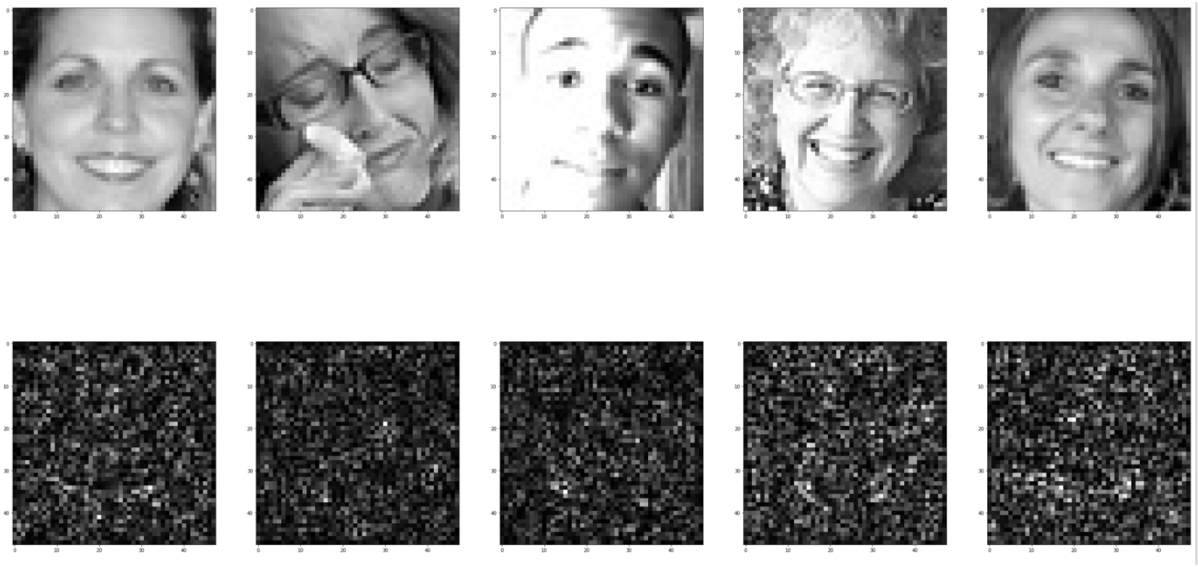


此處的作圖並沒有將confusion matrix做normalize，因為實作normalize後發現各個數值差距有些大，反而不如原本confusion matrix來的可讀，最上方為confusion matrix的各個數值，然後上方為將這些數值畫圖出來，可以發現CNN model仍能正確分類出對應類別，所以圖中數值集中在對角方向，但仍可以觀察出model在分類某些類別能力較弱，容易搞混，例如類別0（生氣）有機率被判斷為類別2（恐懼）和類別4（難過），可以推測這些相關的負面情緒可能會被搞混，同理類別2（恐懼）也有一定

機率被預測為類別0（生氣）和類別4（難過），類別4（難過）也是同理，所以可以推測在生氣、恐懼、難過有時會使model搞混。

[關於第四及第五題]
可以使用簡單的 3-layer CNN model [64, 128, 512] 進行實作。

4. (1%) 畫出CNN model的saliency map, 並簡單討論其現象。
以下第4題、第5題皆使用簡單3-layer CNN model [64,128,512]架構實作

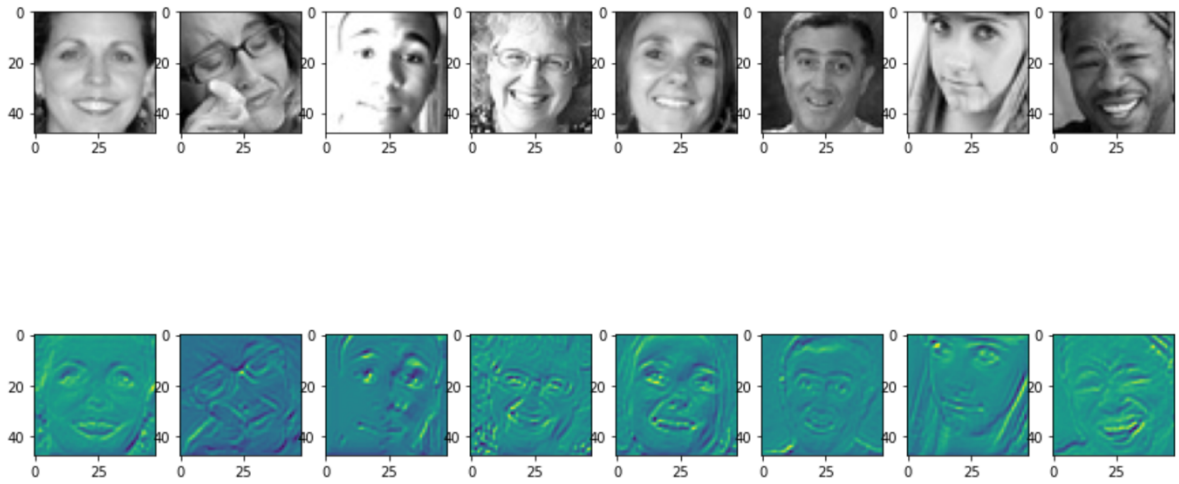


取出部分圖片的saliency map觀察發現，由於可能使用簡單3-layer CNN model效果不夠理想，所以saliency map無法呈現非常好看的特徵，但仍可以從中判斷model是從圖片中哪些部分判斷類別，途中白色部分為gradient較大的地方，也就是白色部分改變會影響較大model判斷類別，由上面五張圖仔細觀察發現，第一張圖在微笑部分有較大的輪廓，可以猜測model可能由微笑的輪廓來判斷為類別3（高興），第四張和第五張圖也是同樣發現在微笑的地方有較大的gradient，可以從saliency map得到訊息，猜測在判斷高興類別時model會較大偵測嘴巴對應部分的features，而第二張圖可以發現在嘴角下揚的輪廓有較大的gradient，可以猜測model側重嘴角附近的輪廓的features來判斷是否是類別4（難過），相較這些類別，第三張中立情緒的圖片，saliency map就沒有這麼明顯，猜測可能model在中立情緒的判斷能力沒有高興和難過那麼厲害。

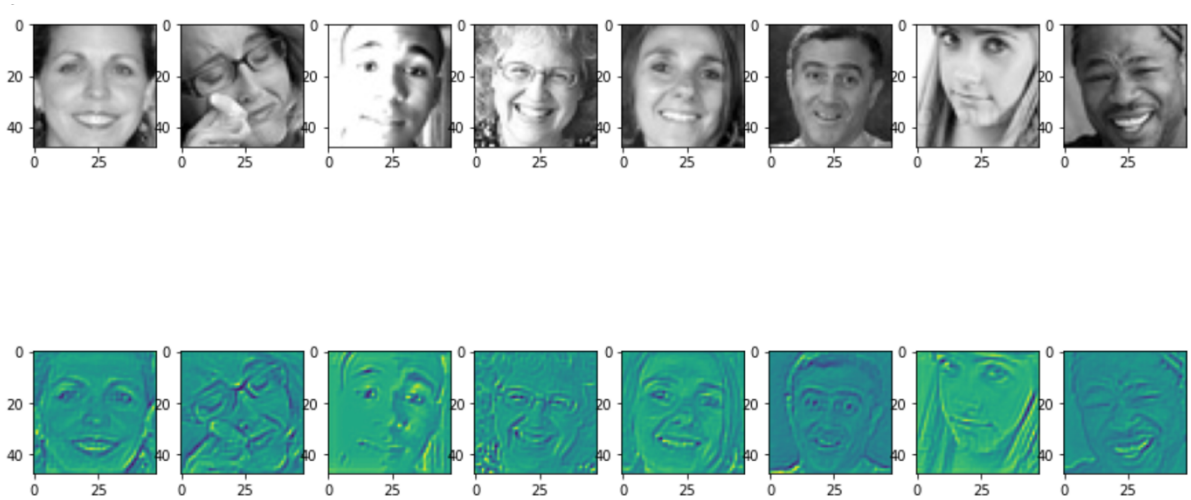
5. (1%) 畫出最後一層的filters最容易被哪些feature activate。

以下圖片分別為最後一層中挑十個filters對應到最被activate的image:

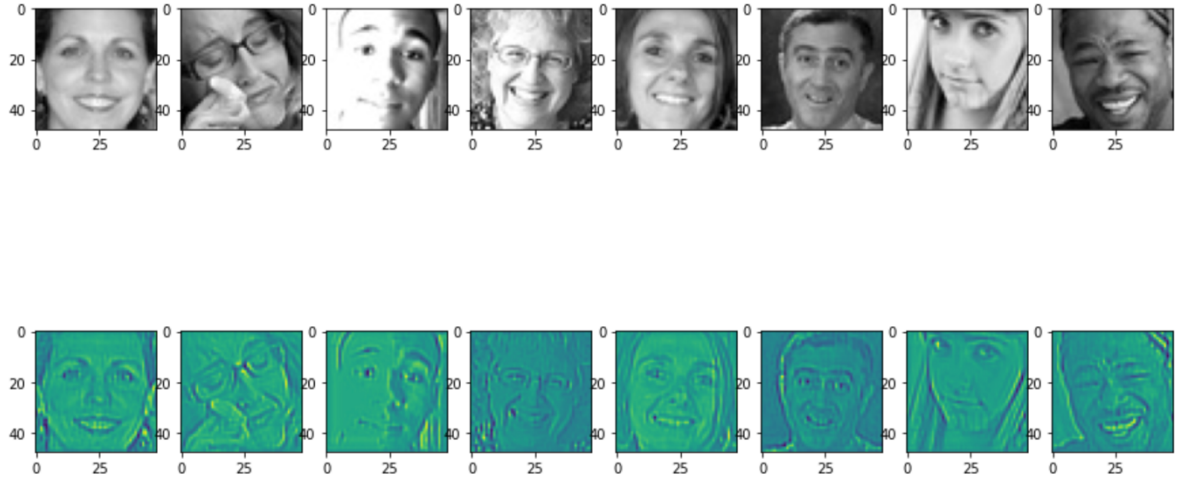
filter1:



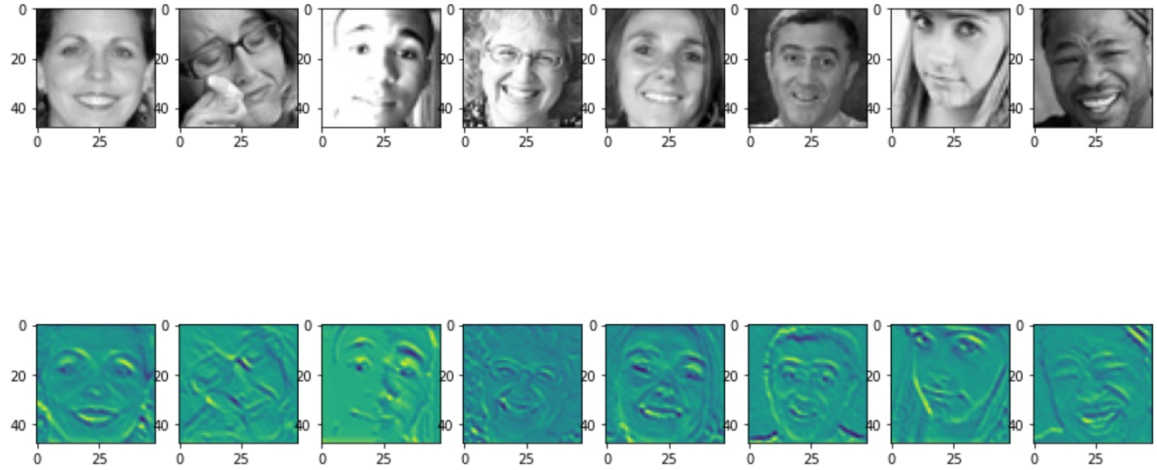
filter2:



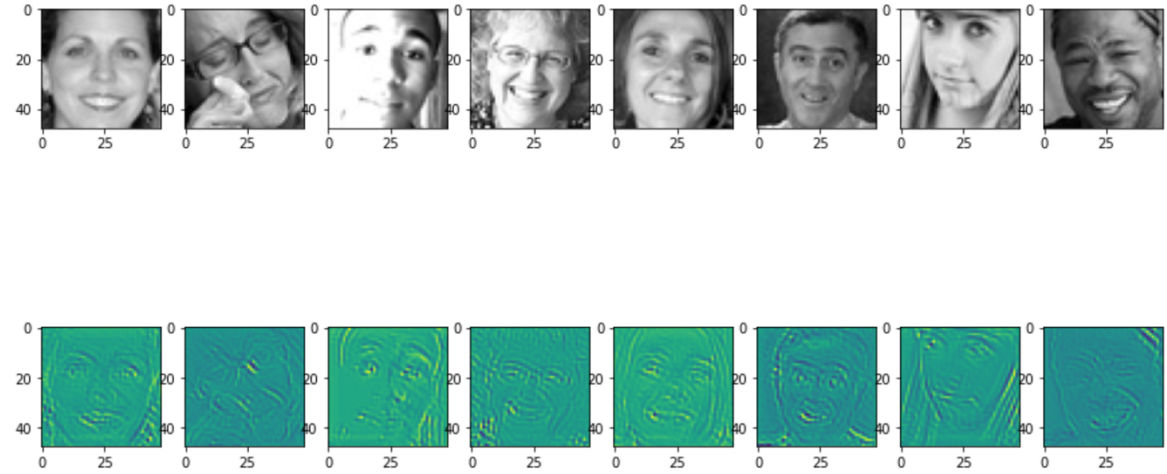
filter3:



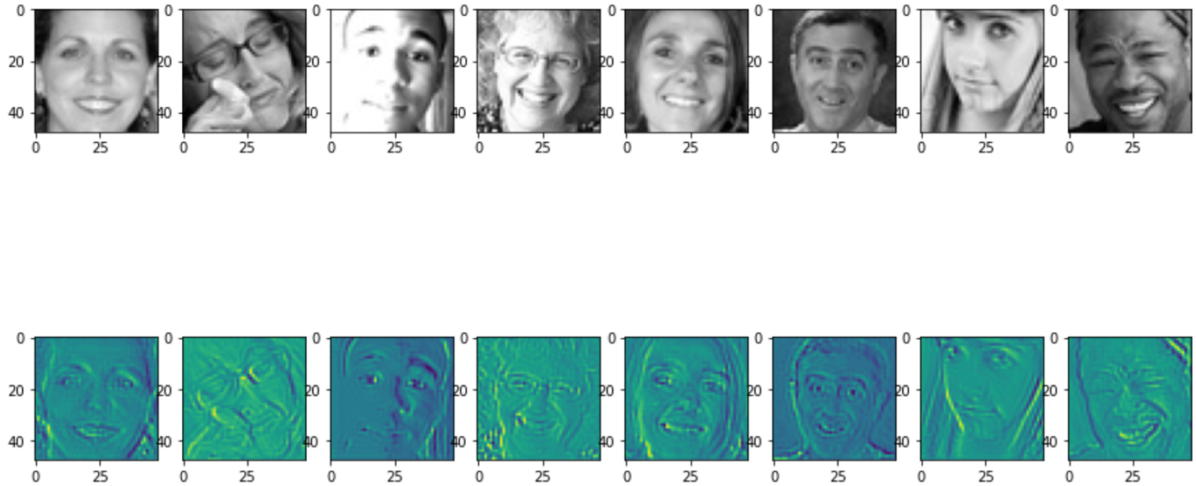
filter4:



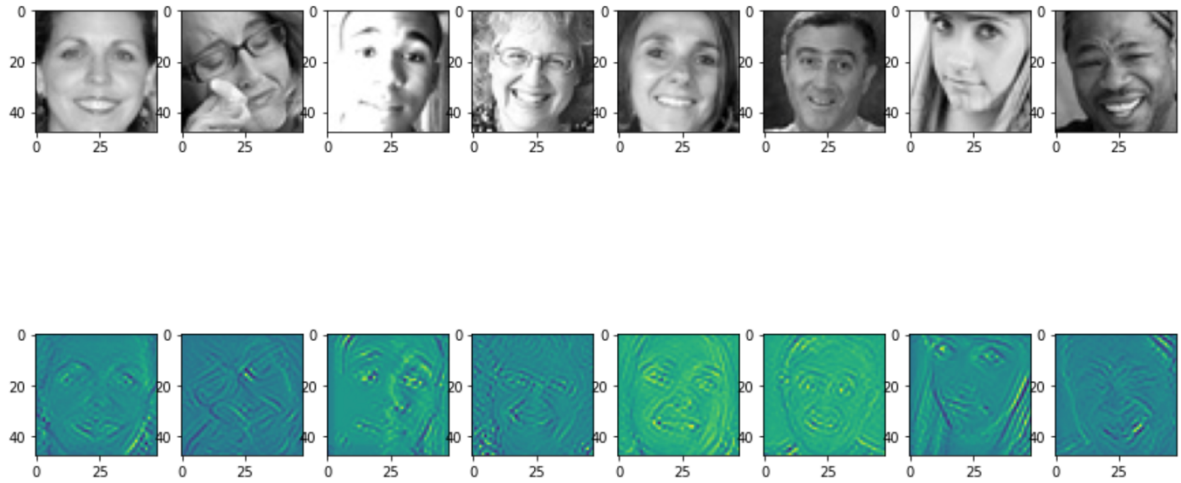
filter5:



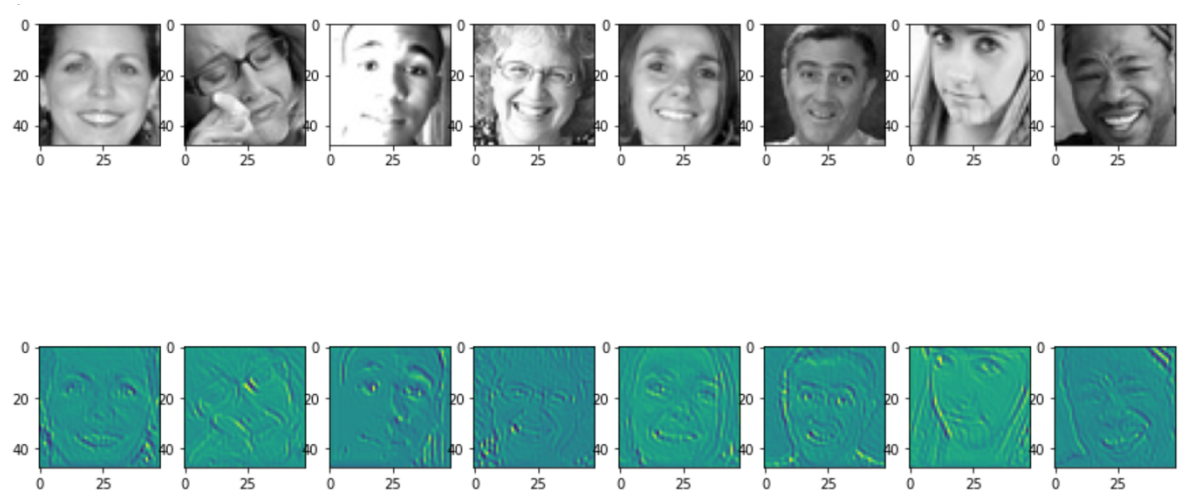
filter6:



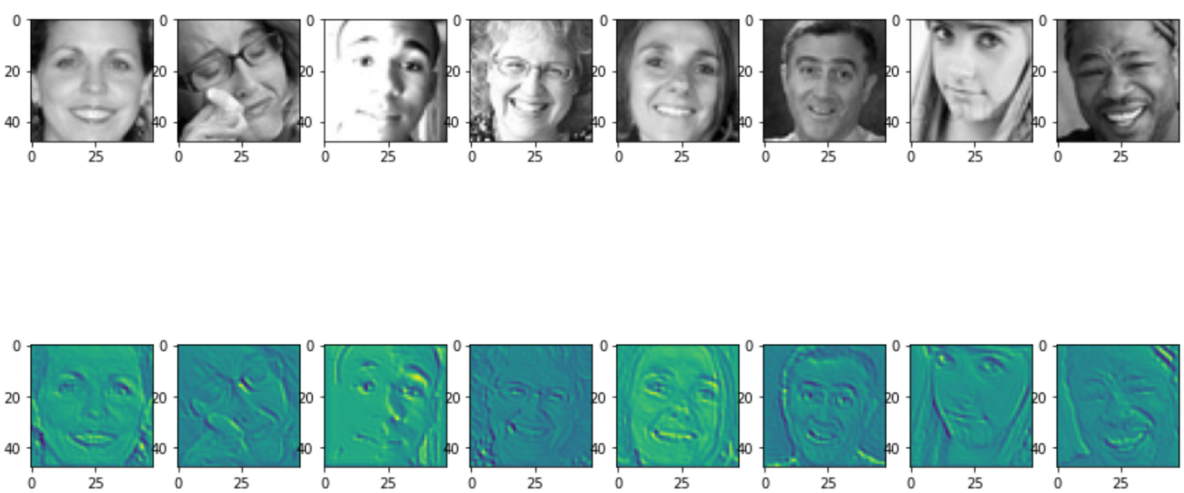
filter7:



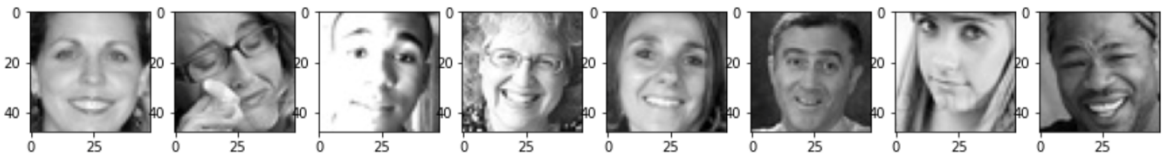
filter8:

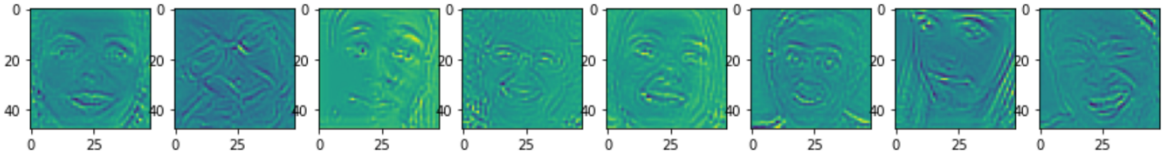


filter9:



filter10:

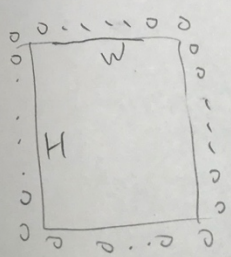




從上面這些圖片觀察到，filters有針對臉部一些特徵進行轉化，例如部分加強臉部的輪廓和邊緣，部分強調眼睛、鼻子的形狀，甚至濾掉大部分features只留下這些部分，特別值得注意的地方在於第二張圖片手的部分被filter給濾掉了，可以發現filters認為手的部分較不影響判斷情緒，從上面這些觀察發現，filter大部分是各種角度的線條紋路來activate人臉的輪廓，或是佈滿坑洞來activate眼睛或鼻子等，我們可以從這些圖片了解到model是如何學習到辨認特徵。

6. (3%)Refer to math problem
   https://hackmd.io/@ASZWRvp7SjOEdYLqF3JYdg/HJMbtPOdD

1. $\Rightarrow (B, \frac{W+2p_1-k_1}{s_1}+1, \frac{H+2p_2-k_2}{s_2}+1, output\_channels)$ #



padding = 1

可觀察出來

示意圖

2. $\frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i}\gamma$

$$\frac{\partial l}{\partial \sigma_\beta^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \times (x_i - u_\beta) \cdot \frac{-1}{2}(\sigma_\beta^2 + \varepsilon)^{-\frac{3}{2}}$$

$$\frac{\partial l}{\partial u_\beta} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_\beta^2+\varepsilon}}\right) + \frac{\partial l}{\partial \sigma_\beta^2} \frac{\sum_{i=1}^{m} -2(x_i - u_\beta)}{m}$$

$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_\beta^2+\varepsilon}} + \frac{\partial l}{\partial \sigma_\beta^2} \times \frac{2(x_i - u_\beta)}{m} + \frac{\partial l}{\partial u_\beta} \times \frac{1}{m}$$

$$\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i}$$

3、case 1: $y_t=1$

$$\Rightarrow \frac{\partial L_t}{\partial z_t} = -\frac{\partial y_t \log \hat{y}_t}{\partial z_t} = -y_t \frac{\partial \log \hat{y}_t}{\partial z_t} = -y_t \frac{1}{\hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t}$$

$$= -y_t \frac{1}{\hat{y}_t}(\hat{y}_t - \hat{y}_t^2) = y_t \hat{y}_t - y_t = \hat{y}_t - 1$$

$$= \hat{y}_t - y_t$$

case 2: $y_t = 0$

$$\Rightarrow \frac{\partial L_t}{\partial z_t} = \hat{y}_t - y_t$$