

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：模型架構如 Figure 1 所示，其中的  $C_x$ ,  $x=1, 2, 3, 4$  為一 Convolutional Block，依序經過  $\text{Conv2D}()$ ,  $\text{LeakyReLU}()$ ,  $\text{BatchNormalization}()$ ,  $\text{MaxPooling}()$ ，以及  $\text{Dropout}()$ 。詳細的參數設置如下：

- $\text{Conv2D}()$ ：除了  $C1$  的 kernel size 為  $(5, 5)$ ，其餘的 Convolutional Block 皆設置為  $(3, 3)$ ，而 strides 皆為  $(2, 2)$ ，padding 為 “same”。
- $\text{LeakyReLU}()$ ：每個 Convolution Block 的  $\alpha$  為  $\frac{1}{20}$ 。
- $\text{BatchNormalization}()$ ：使用預設參數，能加快收斂速度和減少 overfitting。
- $\text{MaxPooling}()$ ：每個 Convolution Block 的 pool\_size 和 strides 皆為  $(2, 2)$ 。
- $\text{Dropout}()$ ： $C1, C2, C3, C4, FC1, FC2$  的 dropout rate 依序為  $0.25, 0.30, 0.35, 0.40, 0.50$ 。

整個 model 的參數量總計為 5,660,679，其中有 4,480 個為  $\text{BatchNormalization}()$  所用到的 non-trainable 參數。

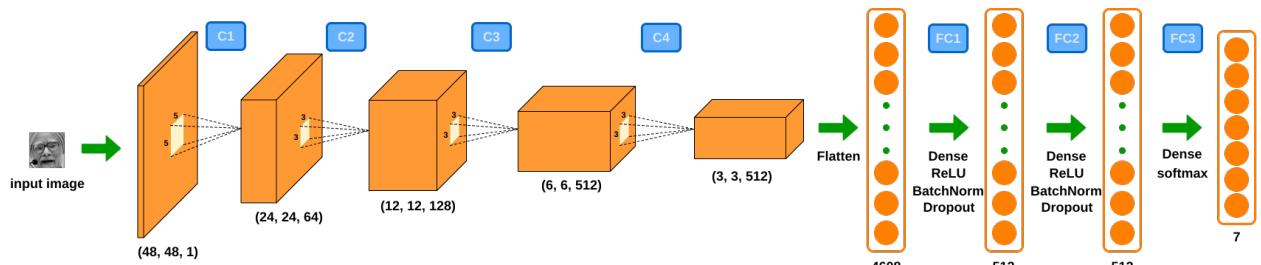


Figure 1: CNN 模型架構。

在資料的處理上，首先，我將 training data 做標準化，取出最後 5000 筆（約莫 20%）的資料當作 validation data，再做 data augmentation，也就是對圖片做平移 (shift)、旋轉 (rotate)、縮放 (zoom)、翻轉 (flip)，以及推移 (shear)，藉此產生更多的 training data，增加模型的抗噪性，減少 overfitting 的可能性，而這部份，keras 中的 `keras.preprocessing.image.ImageDataGenerator` 能完成這件事情。

由於使用 `ImageDataGenerator` 的緣故，在訓練模型時，需要調用 `fit_generator()`，我將 generator 一次生成的資料量（即為 `batch_size`）設為 128，一個 epoch 所需的 `step` 設為 1852，因此，一個 epoch 實際訓練的資料量為  $1852 \times 128 = 237056$  筆。`loss` 採用的是 cross entropy，而 `optimizer` 則是 adam。

訓練過程如 Figure 2 所示，可以觀察到模型大約在 10 個 epoch 左右，valid loss 最低，而 valid accuracy 也約莫在第 10 個 epoch 陡升。最好的 valid accuracy 則出現在第 156 個 epoch，validation accuracy 為 0.70540。可以發現在 10 個 epoch 後 valid loss 雖然不斷上升，但 valid accuracy 却也有持續上升的趨勢，推測應該是使用了 data augmentation 的緣故。

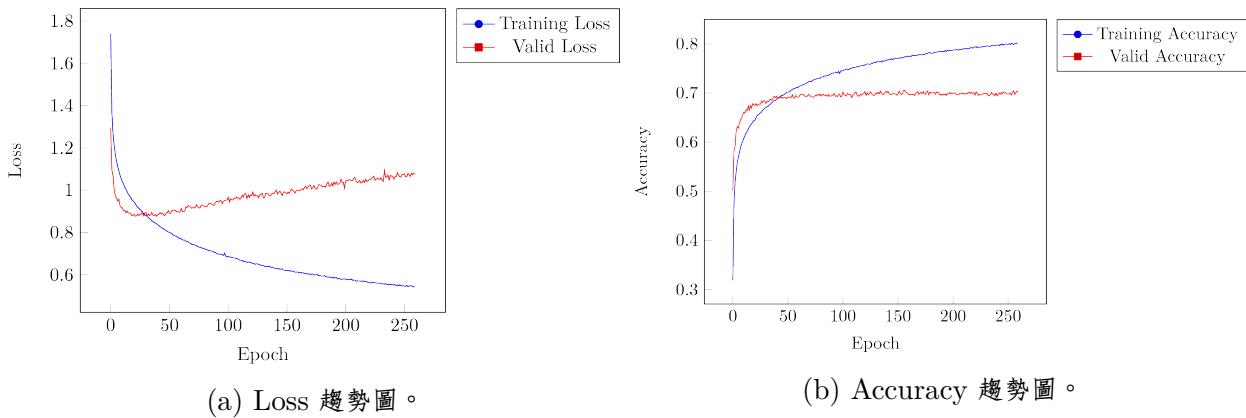


Figure 2: CNN 訓練過程。

至於 kaggle 上最好的分數，是透過 ensemble learning 取得的，我用了 4 個架構相仿且對 training data 做 data bagging 訓練的 CNN 模型，做 average weighting，藉此讓模型更穩定。

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：模型架構如 Figure 3 所示，FC1, FC2 的 dropout rate 皆為 0.5。

整個 model 的參數量總計為 5,783,559，其中有 5,120 個為 BatchNormalization() 所用到的 non-trainable 參數，整體參數量和 CNN 接近。

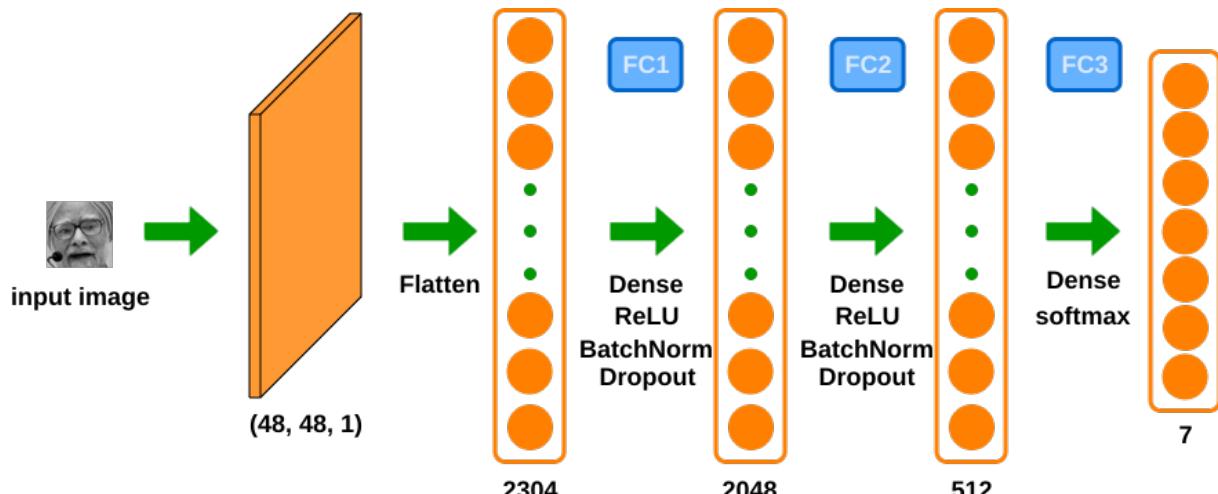


Figure 3: DNN 模型架構。

在資料的處理上，和上一題採用一樣的規格。

訓練過程如 Figure 4 所示，首先注意到 DNN 跑了較多個 epoch，因為在訓練過程中，觀察到僅跑 250 個 epoch 不足以讓 DNN 收斂，於是將 epoch 數增加至 760，藉此觀察 DNN 的完整訓練過程。而以下是我觀察到的幾個現象：

- 相同 epoch 數量，DNN 的準確度完全比不上 CNN，而單就趨勢來看，DNN 所收斂到的值也會比 CNN 差。
- 訓練速度上，雖然 DNN 的參數量略多於 CNN，但 DNN 比 CNN 快了不少，我覺得是因為做 convolution 需要相對大的運算量。
- 無論是在 loss 還是 accuracy 方面，都可以看到 valid 都比 training 好很多，推測是因為我用了 data augmentation，加了許多 noise 在原本的圖片上，而 DNN 不像 CNN 能透過 convolution 濾出圖片的特徵，因此受到這些 noise 影響了在 training 時的 loss 和 accuracy。

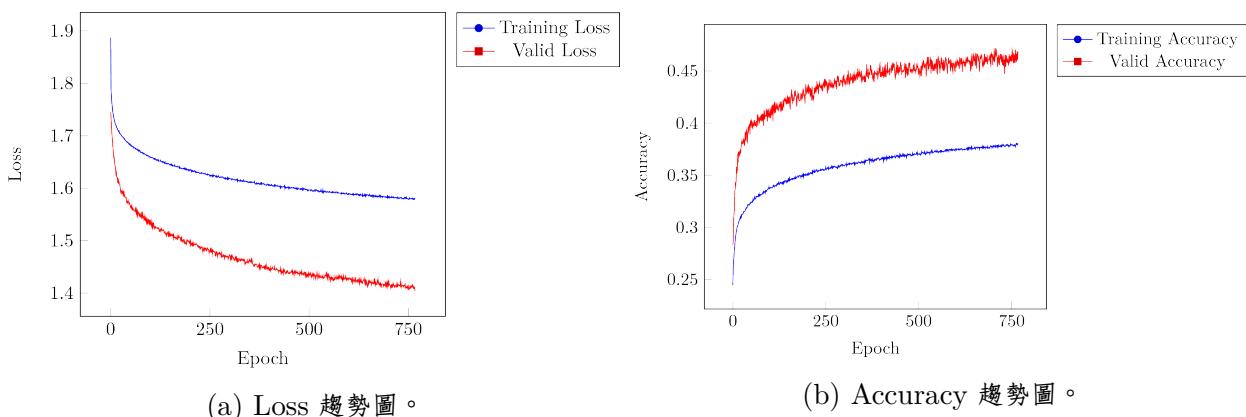


Figure 4: DNN 訓練過程。

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：從 Figure 5 可以觀察到，Neutral 和 Sad 這兩個 class 容易預測錯誤。因此，在 Figure 6 中，展示出 Neutral 和 Sad 彼此間混淆的例子（從 validation data 內所取出）。而 Table 1 列出了 CNN model 預測兩張圖片為哪個 class 的機率。這兩張圖片對我而言是有點模稜兩可的，也就是圖片同時代表著多種情緒。

從中可看到，model 雖然預測 Figure 6a 為 Neutral 的機率最高，但是其實預測 Sad 的機率為次高的，而且和其他 class 的機率相差一個數量級，可見 model 對 Figure 6a 相對其他 class 是有比較偏向 Sad 的。

至於 Figure 6b，model 預測出的機率前四名依序是，Sad、Angry、Happy、Neutral，我自己覺得看眼神比較像生氣加上難過的表情，可能是因為人在 label 時難以分辨他的表情，才會將他歸類為 Neutral，而 model 的 filter 或許跟我一樣抓到眼睛的特徵，才會預測為 Sad。

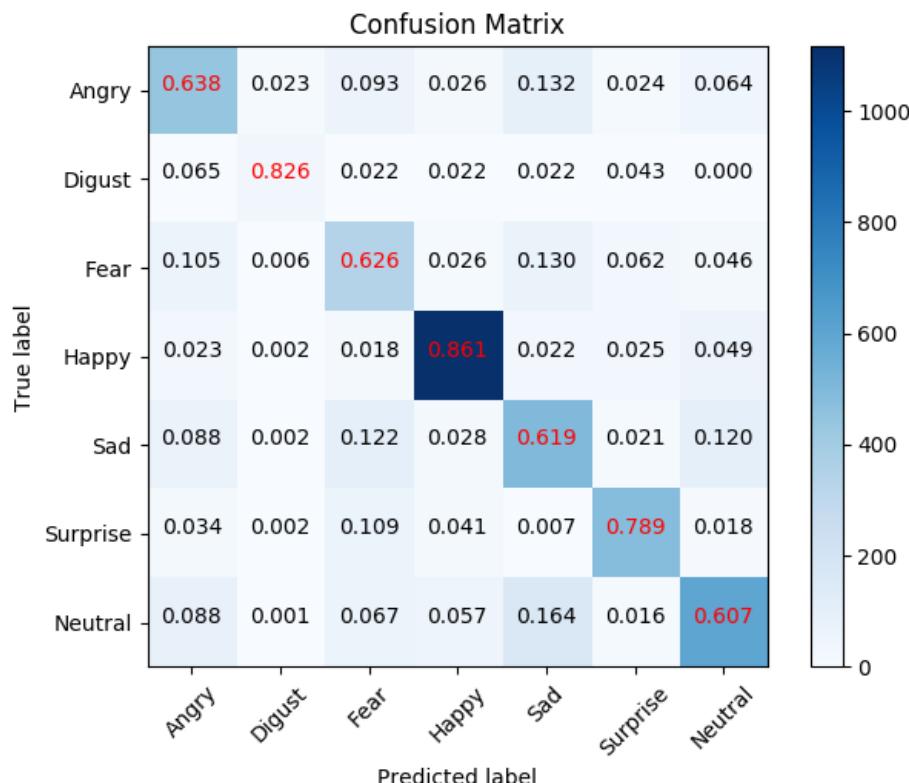


Figure 5: Confusion matrix。



(a) Truth : Sad, Prediction : Neutral。



(b) Truth : Neutral, Prediction : Sad。

Figure 6: Misclassification。

	Angry	Digust	Fear	Happy	Sad	Surprise	Neutral
Figure 6a	0.01689	0.00002	0.05177	0.01292	<b>0.26106</b>	0.00004	<b>0.65726</b>
Figure 6b	0.20673	0.01648	0.04171	0.16031	<b>0.44384</b>	0.00014	<b>0.13075</b>

Table 1: Probability distributions over 7 classes of misclassification images °

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：Figure 7、Figure 8、Figure 9的原圖是從 validation data 中選出的圖片，label 分別為 Happy、Angry、Fear，而我的 model 對這三張圖片也都預測出正確的 class。

從 Figure 7b 可以看出 heatmap 在嘴巴部份有較高的值，可見 model 在做 classification 時，是 focus 在 Figure 7a 中笑時所呈現的嘴形和牙齒等，而這確實也是我判斷這張圖為 Happy 的依據。Figure 8b 可以看出 model 是 focus 在眼睛和眉毛等部份，應該是因為眼神看起來很兇悍。至於 Figure 9b，則是 focus 在張大的嘴巴。

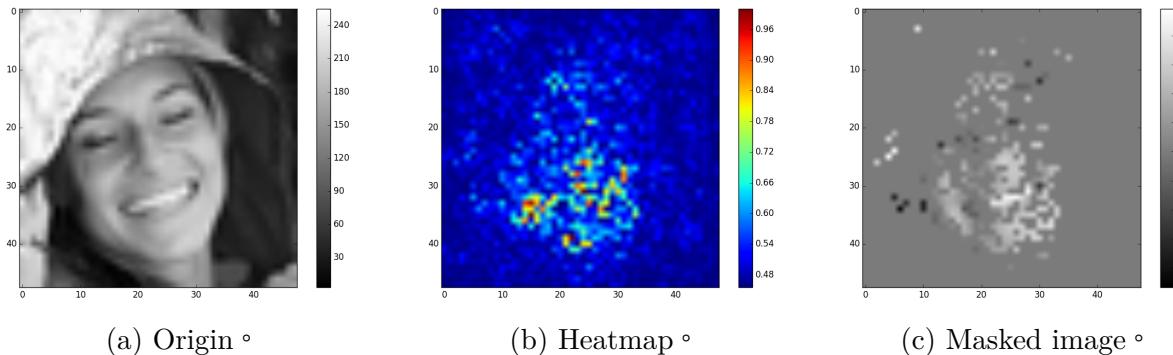


Figure 7: Image that labeled Happy and predicted Happy °

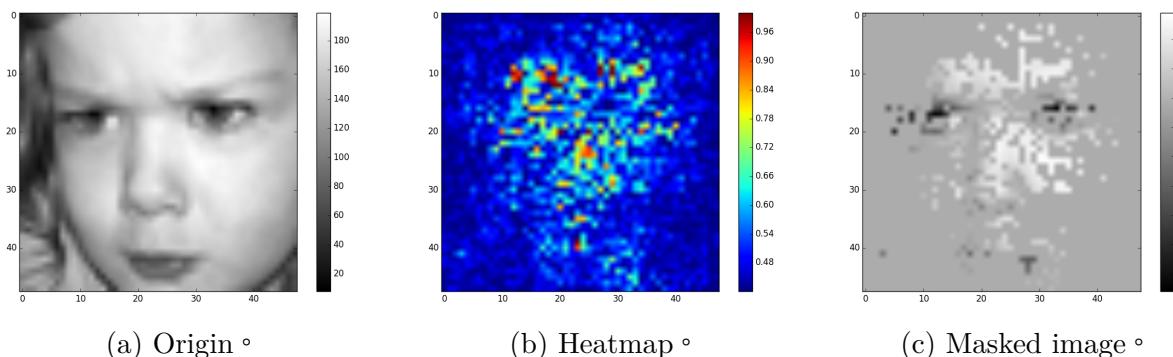


Figure 8: Image that labeled Angry and predicted Angry °

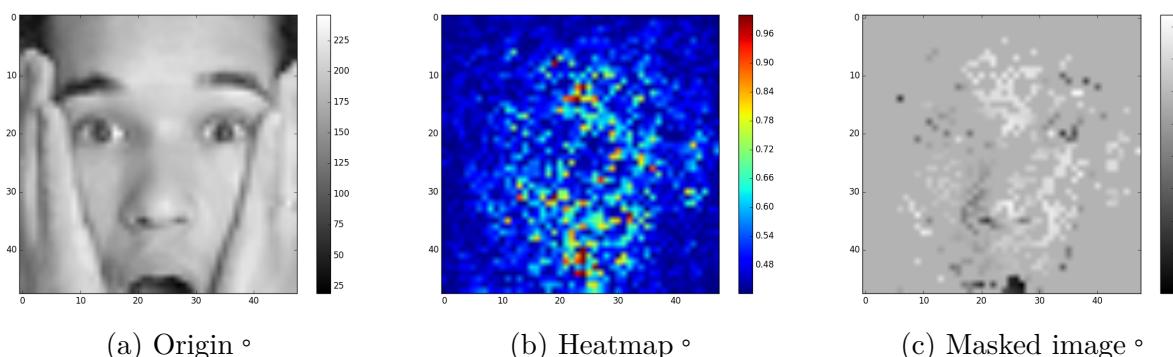


Figure 9: Image that labeled Fear and predicted Fear °

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：Figure 10 為透過 gradient ascent，畫出最大化 filter output 的圖片。

不難發現在相同 layer 中的 filter 有很多都是一樣的，只是旋轉了幾度而已，其中又以旋轉 90° 的最多，因為每張圖片可以用不同的角度拍攝，每個人的五官位置也會不一樣，所以 CNN 若要能辨識這些圖片，filter 要能以各種不同的角度抓取輸入特徵。

可以看到 Figure 10a 中，能 activate filter 的圖片多為較粗的紋理 (coarse texture)，我推測可能是因為他是第一個 Convolution Block，也就是 Figure 1 中的 C1 裡面的 layer，負責抓取較大面積的特徵，像臉部輪廓等，因此擁有粗紋理的圖片較能 activate 這一層的 filter。隨著層數的加深，如 Figure 10d 中，可觀察到 filter 辨認一些類似人臉特徵的紋理，像是眼睛（一顆顆圓圓的）、鼻子等部位。

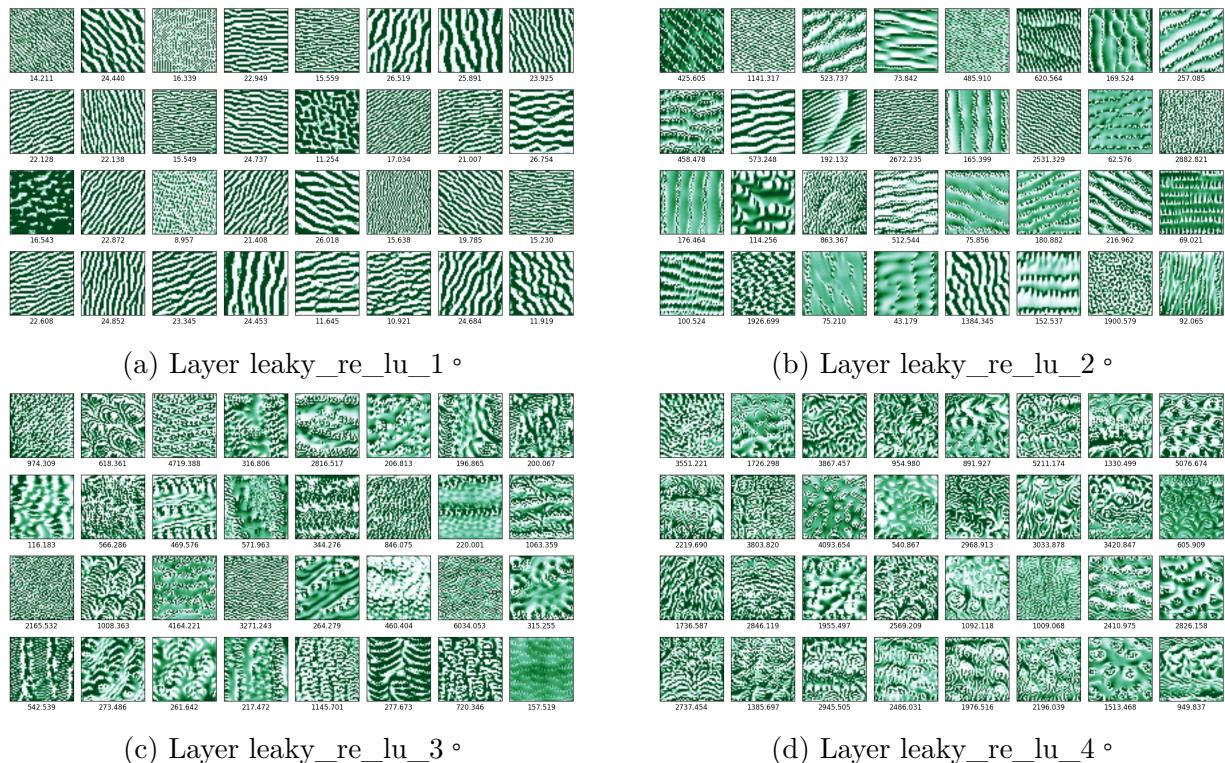
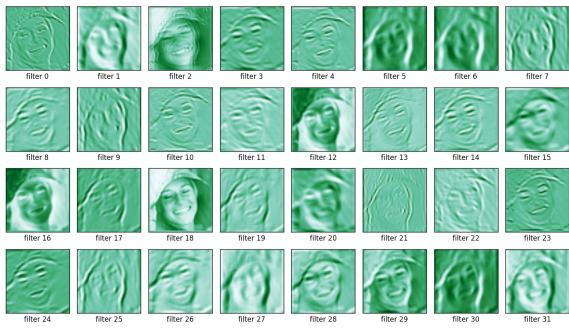


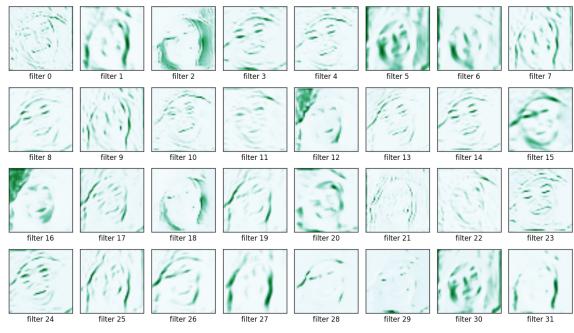
Figure 10: Gradient ascent。

而 Figure 11 為輸入 Figure 7a，某些 filter 輸出的結果。

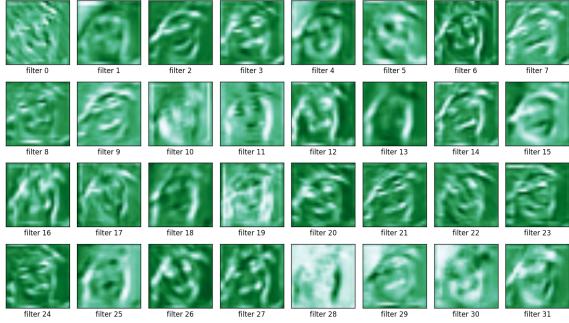
可以發現在較淺層的 layer 中，如 Figure 11a 和 Figure 11b，filter 抓到的是較大範圍的特徵，也就是整張人臉的輪廓；而到了較深層的 layer，如 Figure 11d，filter 已經能過濾出人的五官、頭髮等部位。



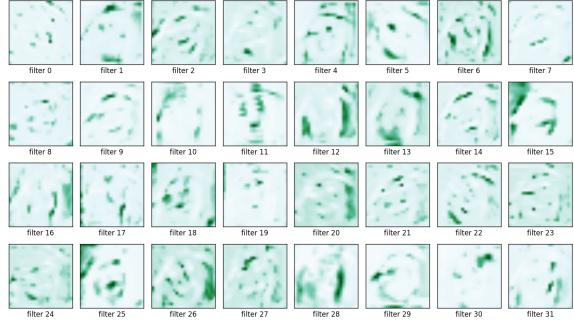
(a)  $\text{conv2d\_1} \circ$



(b)  $\text{leaky\_re\_lu\_1} \circ$



(c)  $\text{conv2d\_2} \circ$



(d)  $\text{leaky\_re\_lu\_2} \circ$

Figure 11: Filter output given image in validation data  $\circ$

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

我用了和 Figure 1 一樣的架構，用相同的 validation data，但在剩餘的 training data 中，我將其分為 labeled data  $(X_l, Y_l)$  和 unlabeled data  $X_u$ 。隨後進行 self-training，流程如下：

1. 用 labeled data  $(X_l, Y_l)$  訓練 CNN，epoch 數為 10。
2. 用 CNN 預測 unlabeled data  $X_u$  在哪個 class  $C$ ，其中落在 class  $C$  的機率為  $f(X_u)$ 。
3. 若  $f(X_u)$  大於等於 confidence (我設為 0.7)，則將  $(X_u, C)$  加入 labeled data。
4. 重複 Step 1 到 Step 3

在 Figure 12 中顯示了 self-training 的訓練過程，其中  $R$  為 unlabeled data 的比例。從訓練過程中可以發現結果並沒有比較好，且準確度隨著  $R$  的上升而下降，因為我們的 unlabeled data 是從原本已經有 label 的資料取出，和原本的模型相比之下，等於少訓練了部份的資料，因此表現沒有比較好是在預期之內。

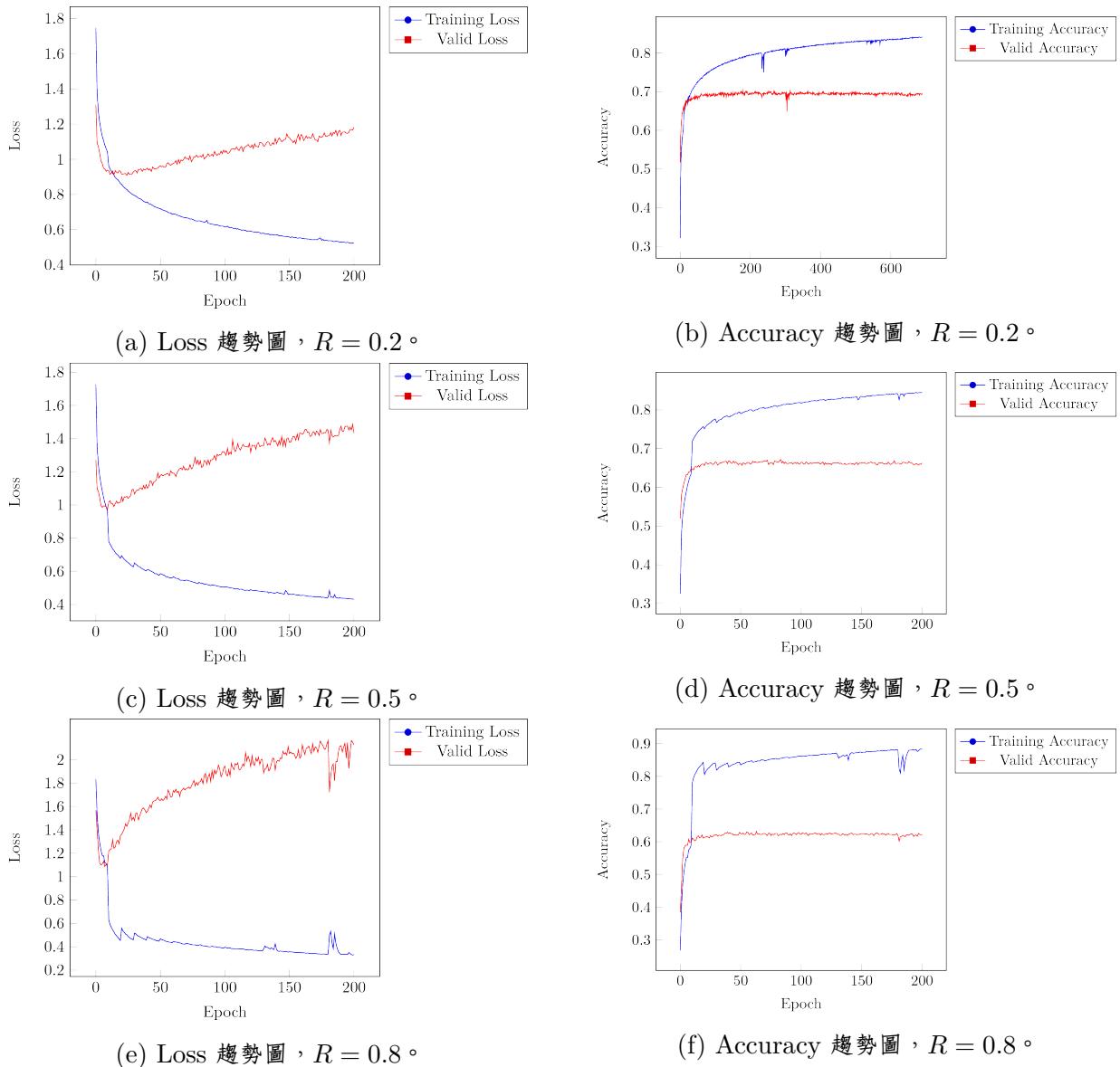


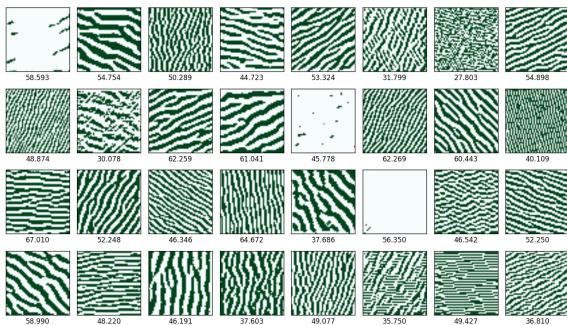
Figure 12: Self-training 訓練過程。

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%）

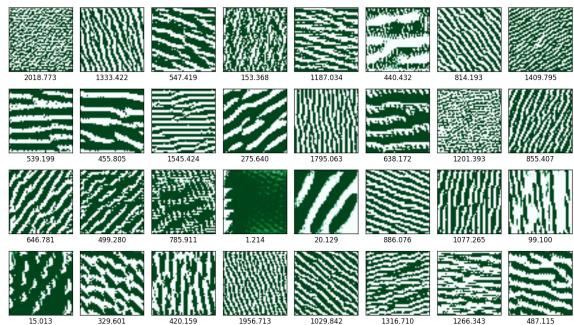
1. Use other model with poor performance to see what is the difference.

我用和 Figure 1 一樣的架構，但只訓練一個 epoch，而這個 model 在 validation data 上得到 0.51040 的準確率。

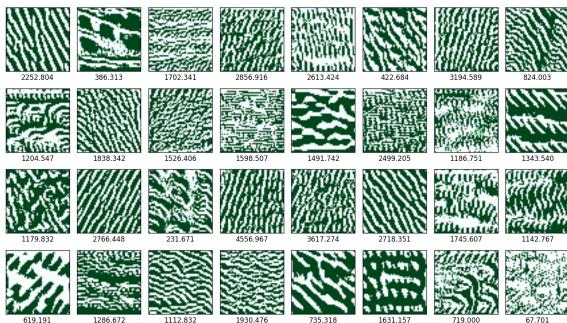
從 Figure 13 可以看到，仍然有許多 filter 是在轉了幾度後相似的，且淺層 layer 中的 filter 依然容易被粗紋理的圖片所 activate。和 Figure 10 不同的點在於，即使到了深層 layer 中，如 Figure 13c，filter 依然容易被粗紋理、直橫條紋的圖片 activate，不像 Figure 10c 中已經有接近人臉特徵的圖案了，反而是到了 Figure 13d 中才有許多非直橫條紋的圖片 activate filter。



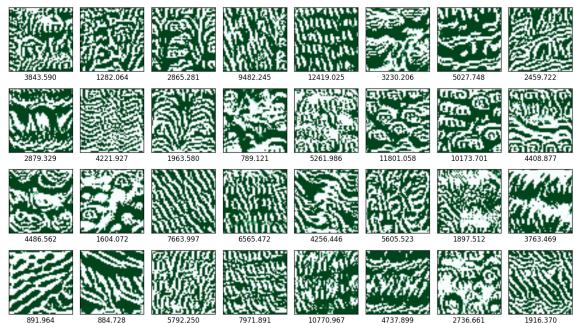
(a) Layer leaky\_re\_lu\_1。



(b) Layer leaky\_re\_lu\_2。



(c) Layer leaky\_re\_lu\_3。



(d) Layer leaky\_re\_lu\_4。

Figure 13: Gradient ascent with poor model。

而 Figure 14 是以 Figure 7a 做為輸入，某些 filter 輸出的結果。

和 Figure 11 相比下，可以發現 filter 所輸出的仍然是人臉的輪廓以及五官，但不同的是，在較深層的 layer，如 Figure 14d 中，filter 仍然有輸出明顯的人臉輪廓，反倒是 Figure 11d 中，雖然也有，但相比之下較不明顯。我因此推測這代表著這個 model 即使到了較深層的 layer，仍然有部份 filter 在過濾基本的特徵，而影響了模型的表現。

所以我想這兩個 model 最主要的差異為 Covolutional layer 的能力，訓練較多次的看過較多張的圖片，Convolutional layer 因而能從中萃取出圖片的精華，如人的五官等，而後進行分類；而訓練較少次的，還無法從訓練資料中取出特徵，準確率較低。

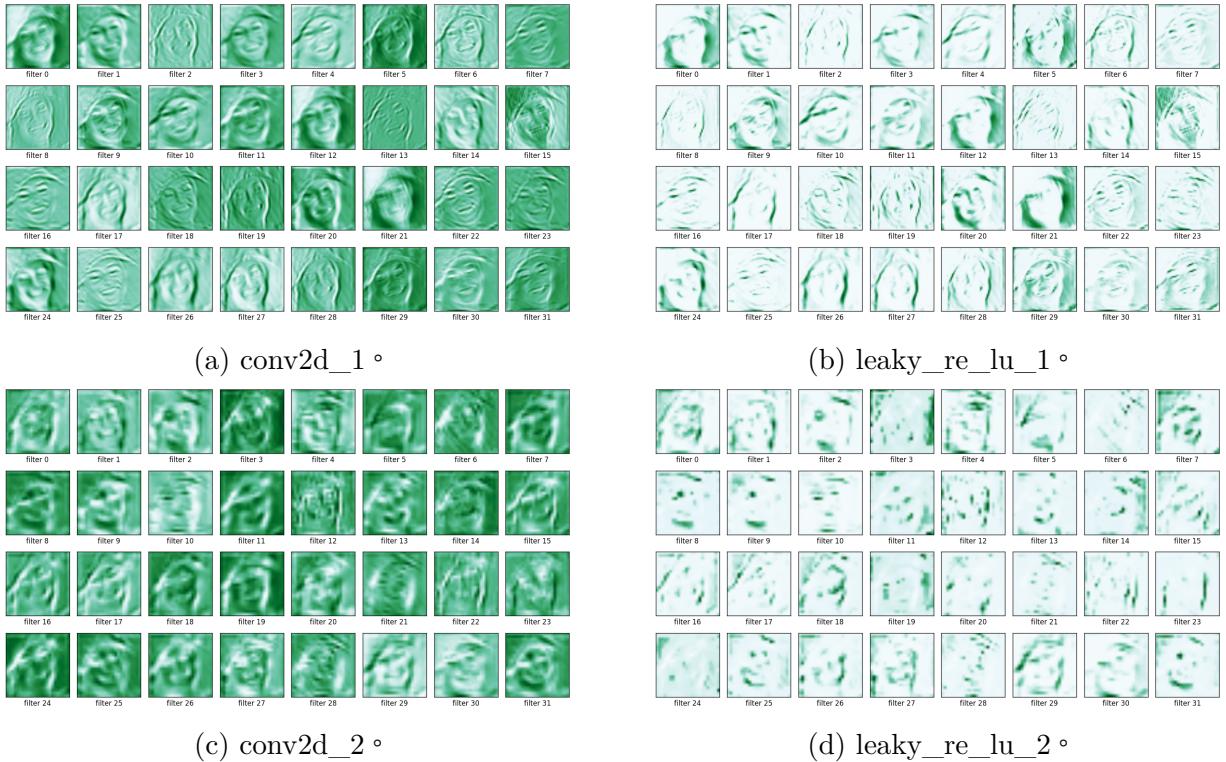


Figure 14: Filter output given image in validation data with pool model。

2. Try to find which image will activate the specific class the most.

這部份一樣是用 gradient ascent，只不過要將欲 maximize 的對象換成 model 輸出某個 class 的機率，也就是將助教的範例程式中的 target 換成以下的式子：

```
target = K.mean(emotion_classifier.output[:, class_index])
```

若單純只有這樣子改，雖然產生出的圖片 model 都會以高於 99% 的機率預測為特定的 class，但如 Figure 15 所示，這些圖片都不是人可以辨識的；於是，我在做 gradient ascent 的過程中加入了數種 regularization，包含 decay、gaussian blur、median filter，詳細的參數設定如下：

- Gradient ascent step : 5000。
- Gradient ascent learning rate : 2°。
- Decay : 每個 step decay 0.01。
- Median filter : 每 3 個 step 用  $5 \times 5$  大小的 kernel 做 median filter。
- Gaussian blur : 每 7 個 step 做 gaussian blur，其中  $(\mu_1, \mu_2, \sigma_1, \sigma_2) = (0, 0, 1, 1)$

加了 regularization 所產生的圖片，model 同樣也會以高於 99% 的機率預測為特定的 class，而且如 Figure 16 所示，明顯能看出人臉輪廓以及五官，我認為最明顯的是 Figure 16d，圖中的嘴巴部份張開，嘴角上揚，就是 Happy 這個 class 的特徵之一。

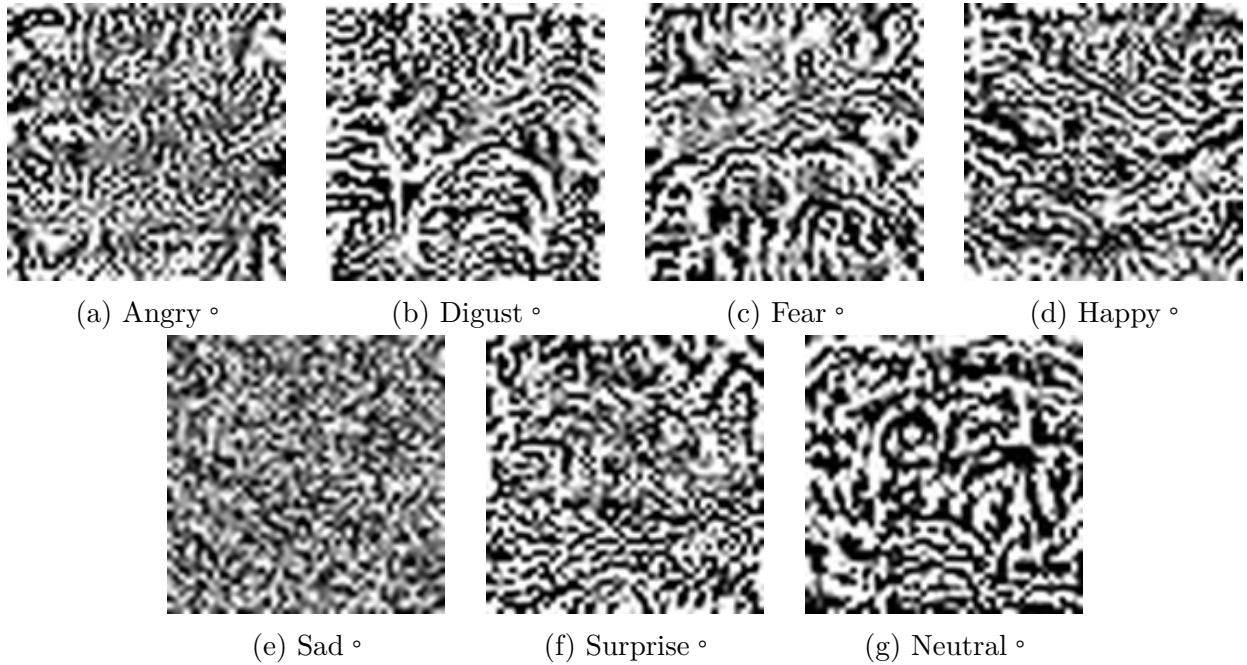


Figure 15: Image that activate the specific class the most without regularization °

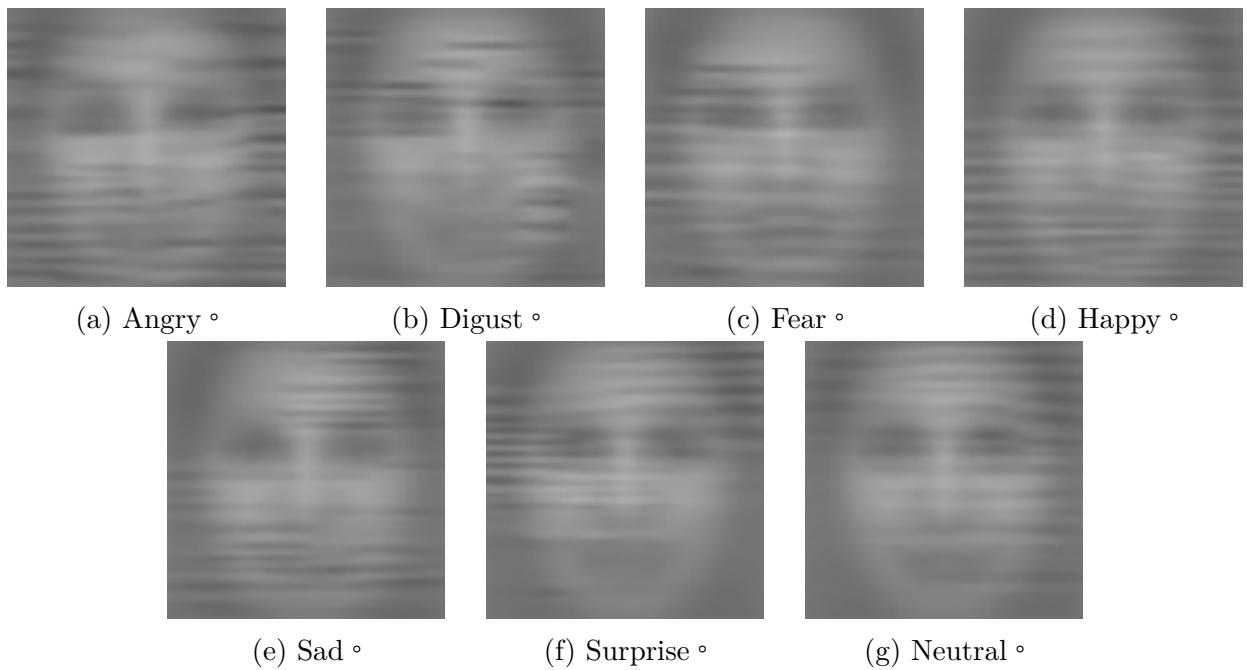


Figure 16: Image that activate the specific class the most with regularization °

3. Start from natural image (not white noise), and try to create the adversarial image.
- Figure 17a 是 validation data 中 label 為 Neutral 的圖片，而我的 model 也預測此圖片為 Neutral。我將這張圖片分別對幾個 class 做 gradient ascent，藉此產生 adversarial image。很明顯在對 class Happy 做完 gradient ascent 後，如 Figure 17b 所示，能看到嘴角的部分有些上揚；而若對 class Angry 做 gradient ascent，如 Figure 17c 所示，能看到嘴形變的有點像在鬧彆扭的嘟嘴巴。



(a) Origin ( Neutral )°



(b) Happy °



(c) Angry °

Figure 17: Adversarial images °