

Homework3 Report

姓名：陳在賢（電機四）

學號：B04505025

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？

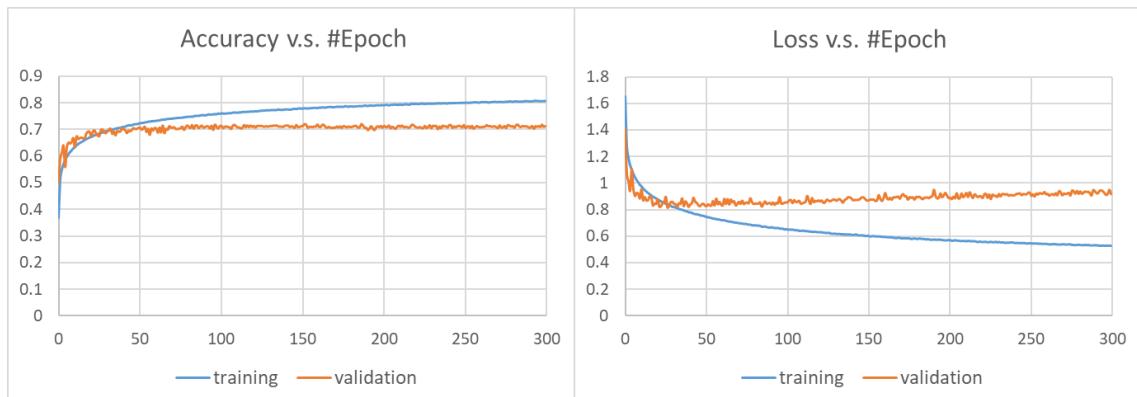
我所實作的 CNN model 在前半段共疊帶了四層 Convolution 2D layer，每層的 filters 數逐層增加，且在層跟層間加上 LeakyReLU、BatchNormalization、AveragePooling2D、Dropout 等數據處理層。詳細說明如下：

- (1) Conv2D: 在四層的 filters 數量依序為 32, 64, 128, 256（由小至大的設計緣由可參考下一段），其餘則保留 CNN 實作時常用的參數設置
- (2) LeakyReLU: 加 activation function 是希望能為模型增添非線性特徵，至於使用 LeakyReLU 是因為其普遍效能佳，且優化或計算速度相當快
- (3) BatchNormalization: 此步會把 data 的範圍調整至近似，才不易造成模型只受少數 training data 主導
- (4) AveragePooling2D: 與 max pooling 的差別在於，ave. pooling 會採納所有 pixel 的值，而不被極端 feature 所主導。但這部分沒有一定的好壞，就我實作的結果，此 task 用 ave. pooling 的效果會略好一些
- (5) Dropout: 在四層的 dropout 比例依序為 0.1, 0.2, 0.3, 0.4，會越選越大是因為前幾層只用來取特定 texture 的 feature，因此不容易出現 overfitting 或易受雜訊影響等問題；反之，越後面的模型越複雜，也就需要使用 dropout 以防止前述問題。

之後將數據 flatten 後，在後半段傳統 NN 處疊帶了三層 Dense layer，節點數呈 1024, 512, 256 逐層減少，並同樣在層與層間加上 BatchNormlization、Dropout。整個 CNN model，共用了 212 萬個參數。

從 input 至 output layer，資訊量維度、節點數、相關參數量是先遞增再遞減，此設計一來是避免中間出現 bottleneck layer，造成某部分資訊被強迫丟掉。同時希望模型先由少量的 Conv2D 取出 texture，再從中取出複雜且多元的 feature（如眼睛、嘴巴形狀或皺紋等）。之後進入 Dense layer 再逐步結合多個臉部特徵，算出跟情緒的關係，並慢慢降維至最終七個類別。

關於訓練過程，我在 train 完每個 epoch 後，有分別記錄 training 及 validation 的 accuracy 及 loss，數據圖如下：



可看出 train 前 50 個 epoch 時，training 及 validation 的 loss 及 accuracy 都明顯得增加，validation 的最高精確度更可達 0.7；尤其前 25 個 epoch，由於用到 dropout 技巧，validation 甚至比 training data 更加精確，進度幅度也相當驚人。

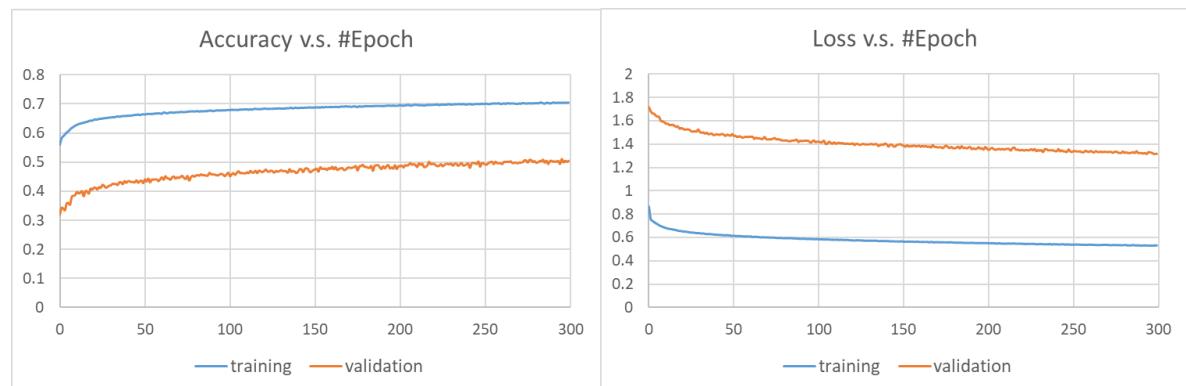
不過 50 個 epoch 後，雖然 training data 的 accuracy 持續上升、loss 持續下降，validation 的 accuracy 却趨於飽和且 loss 越來越大，因此很明顯得發生了 overfitting。不過令我意外的是，在具有超過 200 萬參數的複雜模型，overfitting 却抑制得相當好（至少 validation accuracy 不會明顯往下掉）。可見 dropout 對於 overfitting 的抑制表現相當出色（不過我還沒比較沒有 dropout 的模型到底會發生多嚴重 overfitting）！

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？

我所實作的 DNN model 共含四層 layer：一層 2304 dim input layer（為將 $48 \times 48 \text{ pixel}^2$ 影像作 flatten 所得 2304 維向量）、兩層 hidden layer (820, 256 dims)、一層 7 dim output layer。而層跟層之間一樣有加入 LeakyReLu、BatchNormalization、Dropout 等數據。整個 DNN model 共有約 210 萬個參數。

在訓練過程中，最明顯的是 DNN 較 CNN 快許多，這是因為 CNN 在 train 後半段的 NN 時，還必須先將 training data 跑完前面的 convolution 運算，得到調整過的 feature data，才能做 training。

至於準確率，我一樣有紀錄 training 及 validation 的 accuracy 及 loss 的變化過程，如下二數據圖所示：



首先關於 training accuracy，估計是因為參數量（即模型複雜度）與 CNN 近似，因此在 training data 的 fit 情形近似。然而 validation acc. 却明顯較低，且 train 完 300 個 epoch 後似乎都沒達到飽和，亦即最佳模型還需花更多時間才訓練得出來。

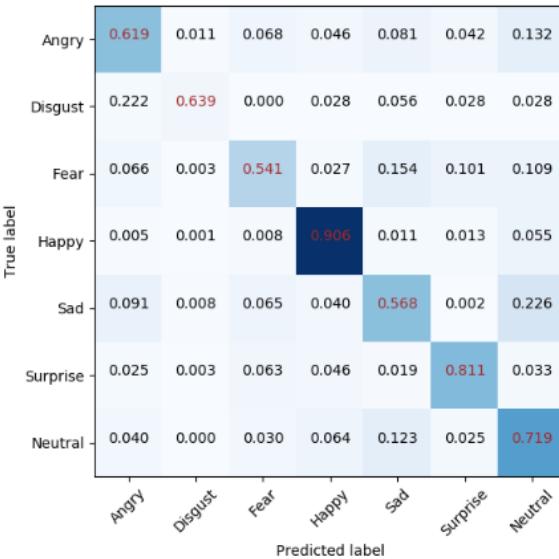
我認為這結果很合理，因為 CNN 的優點恰好在影像辨別有關的 task 都幾乎符合（例如：相同關鍵物件可能出現以各種角度出現在各個地方，卻代表相同的特性）。也就是，我們將人類的 domain knowledge 放入 CNN，使得 CNN 雖然在提煉 feature 要花較久的時間，但取出的 feature 都相當精華，也就能從這些 feature 迅速學會辨別能力。

相反得，DNN 是單以 pixel 值為基礎，慢慢得從像素值間的關聯開始 train，也就使得 train 完 300 個 epoch 後，validation acc. 仍未飽和，模型仍緩慢進步。

3. (1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？並說明你觀察到了什麼？

[繪出 confusion matrix 分析]

我畫的 confusion matrix 如下圖：



可看出準確率最高為的 Happy 類別、最差的兩個則為 Fear 跟 Sad 類別。其中，Fear 容易被誤判為 Sad，而 Sad 容易被誤判為 Neutral。

關於 Happy 準確率高，我認為蠻合理的，因為七個 class 只有 Happy 為正面情緒，因此只要取到嘴唇有上揚或眼睛睜成彎月形等正面情緒特徵，就容易判斷出。

相反得，其他幾類皆為負面或中立情緒，可以想像若臉部表情不夠誇張，就容易誤判，為了驗證此論述，我隨機找了張 Fear 被誤判為 Sad，及 Sad 被誤判為 Neutral 的照片，如下二圖：



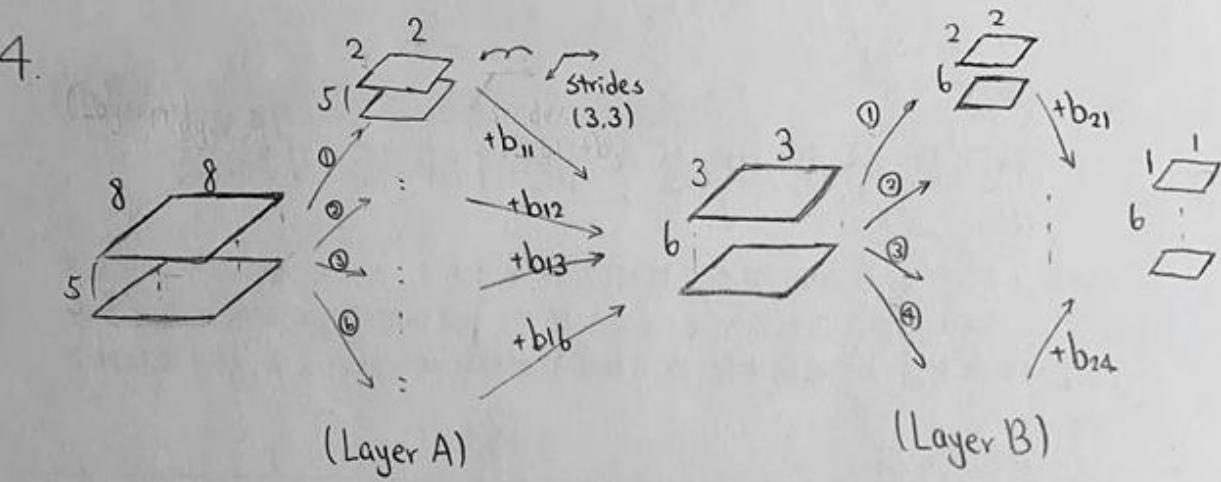
Prediction: Sad
True label: Fear



Prediction: Neutral
True label: Sad

這兩張圖皆如上所述，臉部表情不夠明顯導致誤判。而若將影像在七個類別的預測分數顯示出，也可看出在 True 及 Predicted 兩類別的分數差異並不大，證明這些表情 feature 在此二類別中，應該都給了差不多的分數而難以辨別。

但就此二圖來看，其實光要用人眼辨識對類別都很難了（甚至會覺得 True Label 實際也沒標對 XD），因此要求機器辨別正確也是太強人所難！



$$[a] \quad 2 \times 2 \times 5 \times b + b = 12b \#$$

\downarrow weight \downarrow bias

$$2 \times 2 \times 6 \times 4 + 4 = \underbrace{100}_{\#}$$

b) multi: $2 \times 2 \times 5 \times 6 \times 3^2 = \underline{1080} \#$
 add: $(2 \times 2 \times 5 - 1) \times 6 \times 3^2 = \underline{1026} \#$

$$\text{multi: } 2 \times 2 \times 6 \times 4 \times 1^2 = \underline{\underline{96}} \quad \#$$

$$\text{add: } (2 \times 2 \times 6 - 1) \times 4 \times 1^2 = \underline{\underline{92}} \quad \#$$

layer no.	time-consumption
1	$(2 \times C \times K_1^2 - 1) \times C_1 \times N_1^2$ ↗ ↑ 加法及乘法
2	$(2 \times C_1 \times K_2^2 - 1) \times C_2 \times N_2^2$
\vdots	
l	$(2 \times C_{l-1} \times K_l^2 - 1) \times C_l \times N_l^2$

k_1 = 第1層 layer 的 kernel 長寬
 C_1 = 第1層 filter 數 = 第二層 image depth
 N_1 = 第1層, kernel 在一方向須移動次數
 = 第2層 image 的長寬

$$n_{i+1} = \frac{(n_i + 2p_i) - k_i}{s_i} + 1 \quad \#$$

$$\Rightarrow O(l) = \sum_{i=1}^l (2 \times C_{i-1} \times K_i^2 - 1) \times C_i \times N_i^2 \quad \#$$

以下結果皆來自 python, 計算過程略:

$$(a) \quad I = \begin{bmatrix} -0.6166 \\ -0.5888 \\ -0.5226 \end{bmatrix}$$

$$\text{II. } \begin{bmatrix} 0.6782 \\ -0.1344 \\ 0.0213 \end{bmatrix}$$

1

```
[ 7.18658682  1.37323947]
[ 0.75871342 -0.94399334]
[-3.07034019 -4.45059025]
[ 2.60849751 -2.97853006]
[-1.82299166 -4.75401212]
[ 3.35457763  3.91896138]
[-4.41464321  2.55604371]
[ 3.46569126 -1.73131477]
[-2.31359638  6.03371503]
[-5.75249521  0.97648096]
```

(c)

平均 L2. norm:
5.472

每個 row 即為每個 data 轉換至 2D 的新數據，排列順序依照題目的數據點排