學號：B02705027 系級： 資管四 姓名：陳信豪

P1: Build Convolution Neural Network (1%)

[Accuracy] Build CNN model, and tune it to the best performance as possible as you can.

Record your model structure and training procedure.

以下是我經由 model.summary() 得出的 CNN 模型架構。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 44, 44, 64)        1664
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 44, 44, 64)        123904
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
p_re_lu_2 (PReLU)            (None, 22, 22, 64)        30976
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
p_re_lu_3 (PReLU)            (None, 22, 22, 64)        30976
_________________________________________________________________
average_pooling2d_1 (Average (None, 10, 10, 64)        0
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 128)       73856
_________________________________________________________________
p_re_lu_4 (PReLU)            (None, 10, 10, 128)       12800
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 128)       147584
_________________________________________________________________
p_re_lu_5 (PReLU)            (None, 10, 10, 128)       12800
_________________________________________________________________
zero_padding2d_6 (ZeroPaddin (None, 12, 12, 128)       0
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              3277824
_________________________________________________________________
p_re_lu_6 (PReLU)            (None, 1024)              1024
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
p_re_lu_7 (PReLU)            (None, 1024)              1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 7175
_________________________________________________________________
activation_1 (Activation)    (None, 7)                 0
=================================================================
Total params: 4,845,063.0
Trainable params: 4,845,063.0
Non-trainable params: 0.0
_________________________________________________________________
```

我的數據主要如下。

Backend: Theano

batch size：128

epoch 數：1500

early stopping patience: 100

loss: categorical_crossentropy

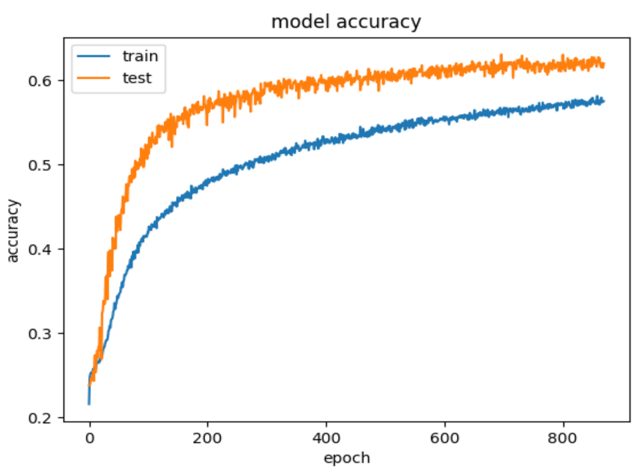
optimizer: Adadelta(lr=0.1, rho=0.95, epsilon=1e-08)

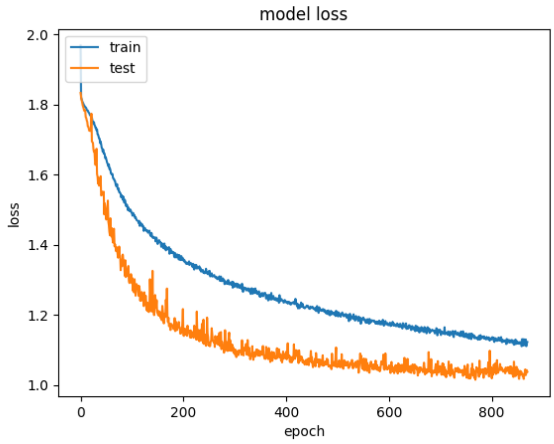
資料取 10% 獨立出來作為 validation dataset

除此之外我運用了 Keras 套件提供的 ImageDataGenerator
讓圖片可以平移翻轉和旋轉

```
datagen = ImageDataGenerator(
        width_shift_range=0.5,
        height_shift_range=0.5,
        rotation_range=40,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=False)
```

最後結果：





可以發現大約到 800 epoch 時就停止了

validation/test 的 accuracy 到達約 62%

train 的 accuracy 還沒有超過 test 的 accuracy

如果將 early stopping patience 設大一點，之後應該會超過

但 validation 的 loss 下降速度已經很慢了，時間會很長，所以就在這邊打住

P2: Build Deep Neural Network (1%)

[Accuracy] Using the same number of parameters as above CNN, build a DNN model to do this task.

Record your model structure and training procedure. Explain what you observed.

以下是我經由 model.summary() 得出的 DNN 模型架構。

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #
====================================================================================================
dense_1 (Dense)                  (None, 1024)          2360320
____________________________________________________________________________________________________
p_re_lu_1 (PReLU)                (None, 1024)          1024
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 1024)          0
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 1024)          1049600
____________________________________________________________________________________________________
p_re_lu_2 (PReLU)                (None, 1024)          1024
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 1024)          0
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 1024)          1049600
____________________________________________________________________________________________________
p_re_lu_3 (PReLU)                (None, 1024)          1024
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 1024)          0
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 7)             7175
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 7)             0
====================================================================================================
Total params: 4,469,767.0
Trainable params: 4,469,767.0
Non-trainable params: 0.0
____________________________________________________________________________________________________
```

我 CNN 的模型架構 params 數是 4845063

而 DNN 的模型架構 params 數是 4469767

兩者相近

在這邊我想說讓兩者盡量在同等的情況下去比較

所以有調整 epoch 和 patience 並將 CNN 重新 train 了一次

而且也不採用 ImageDataGenerator，來處理圖片

我的數據主要如下 (CNN 和 DNN 皆同)

Backend: Theano

batch size：128

epoch 數：100

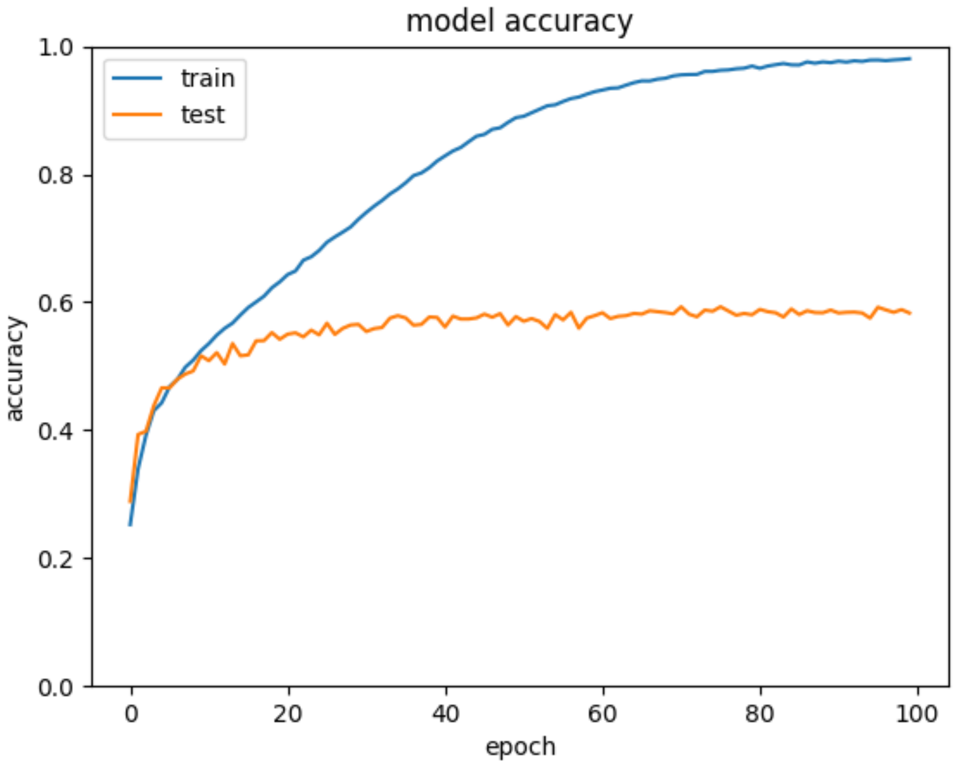
early stopping patience: 100

loss: categorical_crossentropy

optimizer: Adadelta(lr=0.1, rho=0.95, epsilon=1e-08)

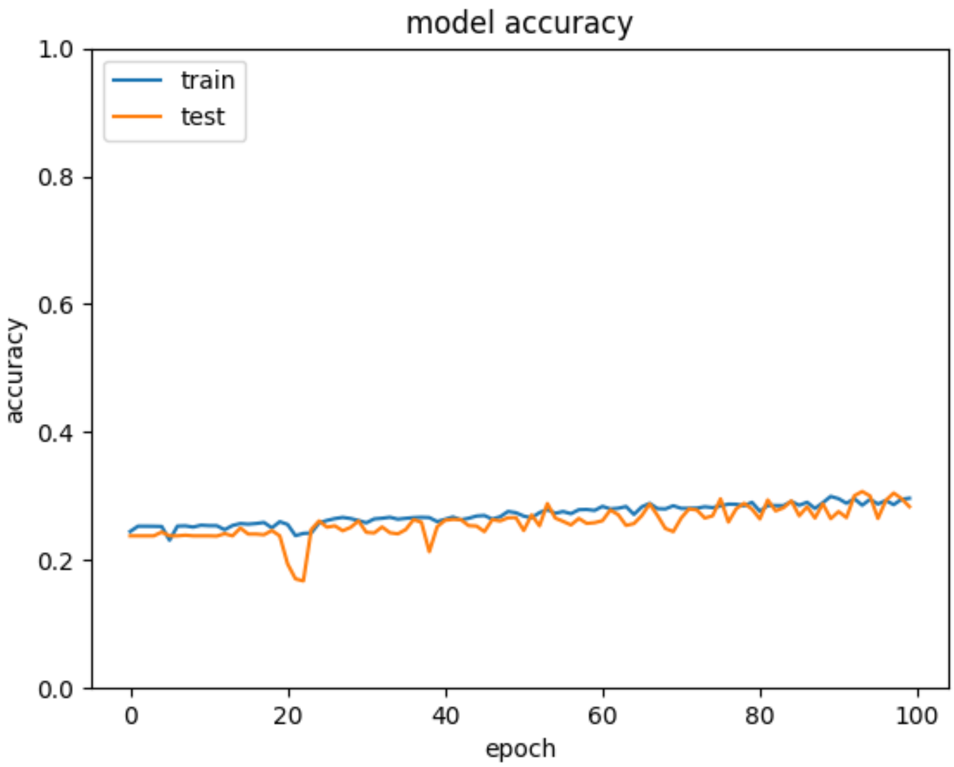
最後結果：

CNN



model accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

DNN



model accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

主要發現：
兩者雖然參數量相近，但是 DNN 的準確率卻難以提升，經過 100 個 epoch 後只到 27% 的準確率，反觀 CNN 經過 100 個 epoch 後準確率就到達了 56% 左右。

另外，CNN 跑一個 epoch 約 80 秒，DNN 一個約 20 秒，但雖然 CNN 每一個 epoch 跑得比較慢，其準確率在提升上還是比較有效率的。

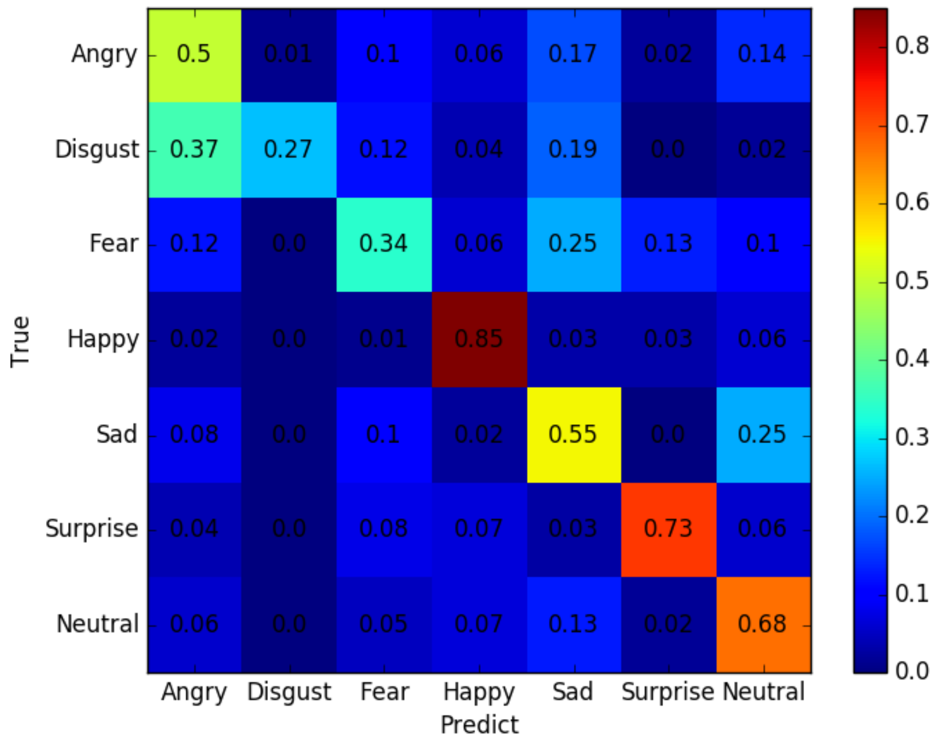
P3: Analyze the Model by Confusion Matrix (1%)

[Analysis] Observe the prediction of your validation data(10% ~ 20% of training data is OK).

Plot the prediction into confusion matrix and describe what you observed.

(from confusion_matrix.py)

取 10 % 做的 validation dataset



主要發現：

diagonal line 上的準確大致上都不錯，模型有訓練成功。

Disgust 常被誤認為 Angry，其值 37% 甚至大於 True Positive 的 27%。

Fear 有 25% 機率被誤認為 Sad，偏高。

Sad 有 25% 機率被誤認為 Neutral，偏高。

Classification Report:

	precision	recall	f1-score	support
Angry	0.56	0.50	0.53	413
Disgust	0.74	0.27	0.39	52
Fear	0.48	0.34	0.40	421
Happy	0.83	0.85	0.84	683
Sad	0.48	0.55	0.51	480
Surprise	0.73	0.73	0.73	327
Neutral	0.55	0.68	0.60	495
avg / total	0.62	0.62	0.61	2871

發現：

Happy 是準確率最高的，無論在 precision 和 recall 上都有最好的表現。

P4: Analyze the Model by Plotting the Saliency Map (1%)

[Analysis] Plot the saliency map of original image to see which part is important when classifying

P5: Analyze the Model by Visualizing Filters (1%)

[Analysis] Use Gradient Ascent method mentioned in class to find the image that activates the selected filter the most and plot them.

Bonus: Semi-supervised Learning (1%)

You can split part of training data and remove their label.

Then try semi-supervised learning techniques (self-training, clustering...) taught in class, and record its performance.