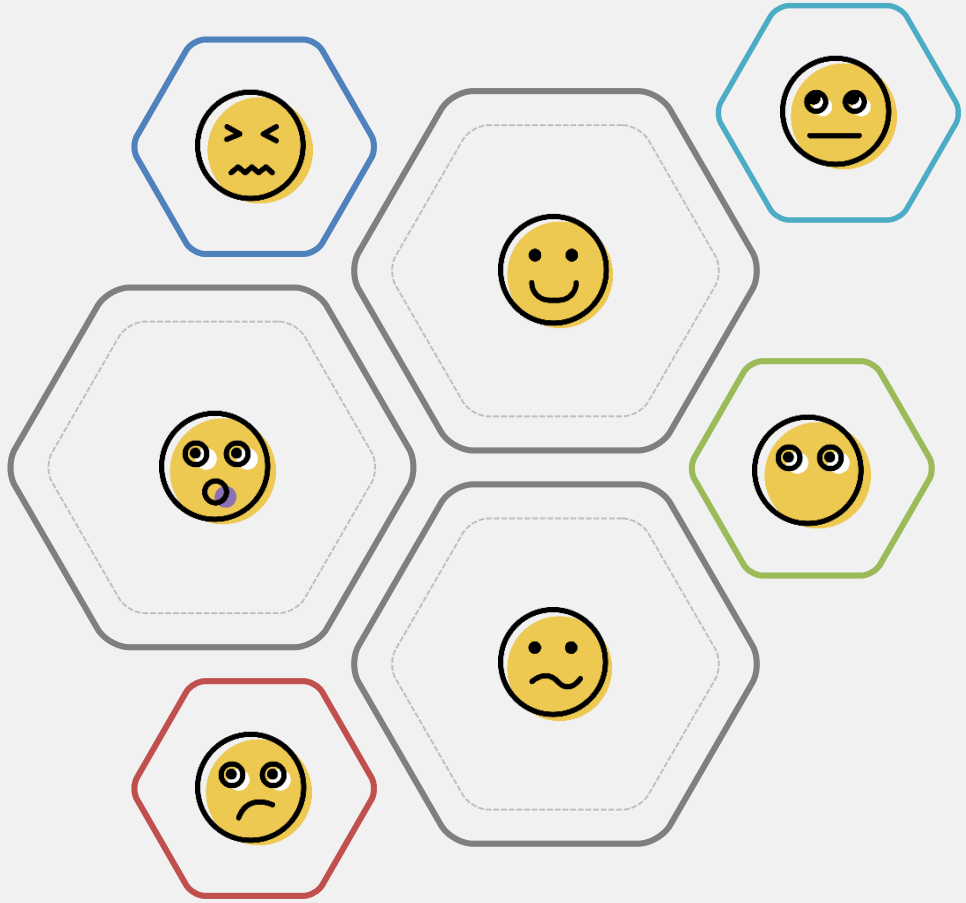
期末專案

學習情緒辨識

Learning emotion recognition

資管碩二 羅聖崴
資管碩一 邱靖詒
資管碩一 林佳緯



使用工具
Tools

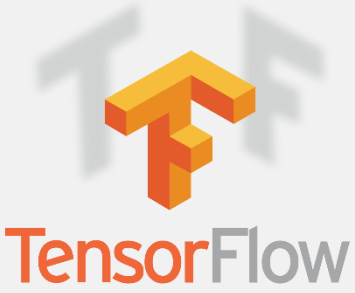
使用 Keras 作為 Frontend

In [1]:

```
1 from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
2 from keras.layers import Dense, Activation, Dropout, Flatten
3 from keras import optimizers
4 from keras.models import Sequential
5 from keras.preprocessing.image import ImageDataGenerator
6 import numpy as np
```

```
Using TensorFlow backend.
```

使用 Tensorflow 作為 Backend



資料前處理

Data preprocessing

設定圖片長寬為150px

```
In [2]:
1  img_width = 150
2  img_height = 150
3  train_data_dir = 'data/train'
4  valid_data_dir = 'data/validation'
```

設置資料夾分別存放訓練樣本和測試樣本

```
In [3]:
1  datagen = ImageDataGenerator(rescale = 1./255)
```

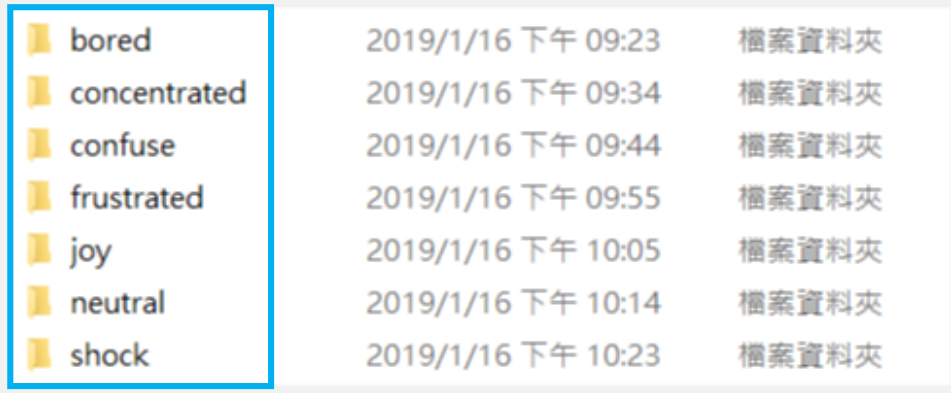
使用ImageGenerator套件進行影像預處理

為了要符合激活函數 → 影像正規化為0至1之數值 ➡ 防止overfitting的現象

收集7種情緒照片

分類
Classification

依據資料夾名稱 one hot encoding 分為情緒7個類別



- 訓練樣本數為2109張照片

In [4]:
```
train_generator = datagen.flow_from_directory(directory=train_data_dir,
                                              target_size=(img_width,img_height),
                                              classes=['frustrated','confuse','bored','joy', 'shock','concentrated','neutral'
                                              class_mode='categorical',
                                              batch_size=16)
```

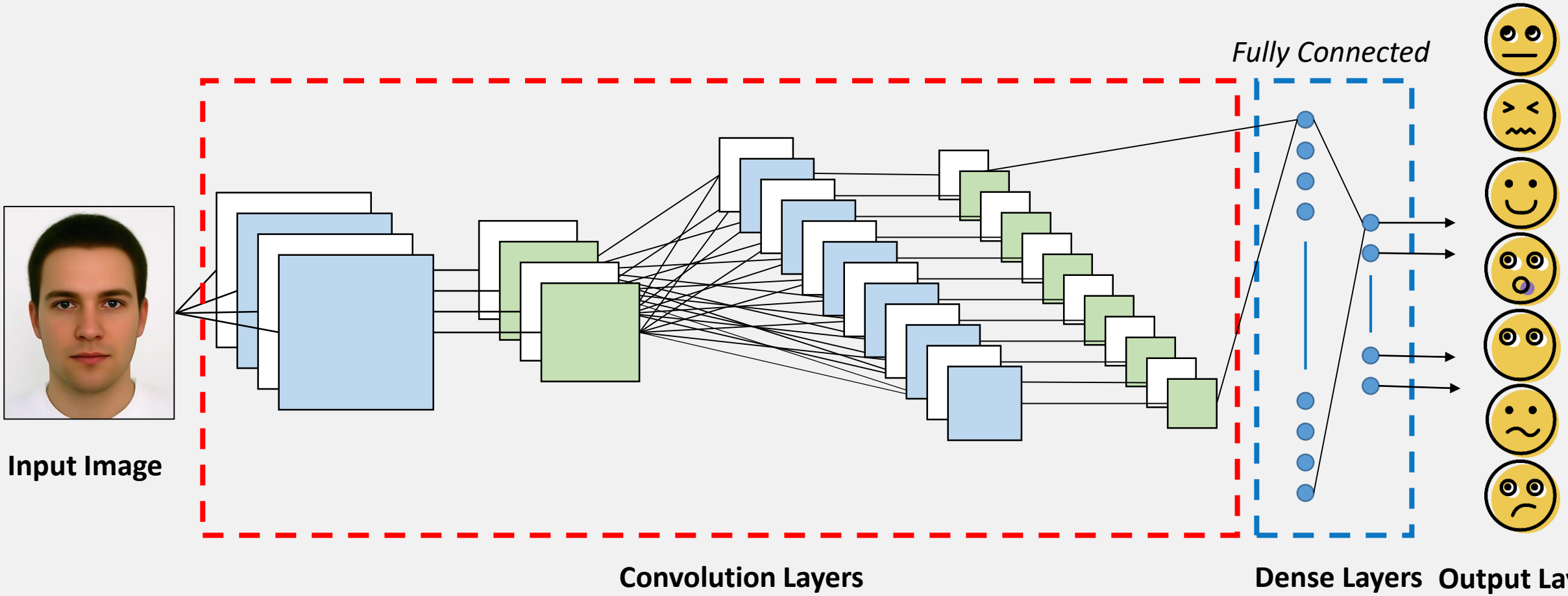
```
Found 2109 images belonging to 7 classes.
```

- 測試樣本數為745張照片

In [5]:
```
validation_generator = datagen.flow_from_directory(directory=valid_data_dir,
                                                   target_size=(img_width,img_height),
                                                   classes=['frustrated','confuse','bored','joy','shock','concentrated','neut
                                                   class_mode='categorical',
                                                   batch_size=32)
```

```
Found 745 images belonging to 7 classes.
```

網路架構
Network Framework



網路架構

Network Framework

```
12 model =Sequential()
13
14 model.add(Conv2D(32,(3,3), input_shape=(img_width, img_height, 3)))
15 model.add(Activation('relu'))
16 model.add(MaxPooling2D(pool_size=(2,2)))
17
18 model.add(Conv2D(32,(3,3), input_shape=(img_width, img_height, 3)))
19 model.add(Activation('relu'))
20 model.add(MaxPooling2D(pool_size=(2,2)))
21
22 model.add(Conv2D(64,(3,3), input_shape=(img_width, img_height, 3)))
23 model.add(Activation('relu'))
24 model.add(MaxPooling2D(pool_size=(2,2)))
25
26 model.add(Flatten())
27 model.add(Dense(128))
28 model.add(Activation('relu'))
29 model.add(Flatten())
30 model.add(Dense(64))
31 model.add(Activation('relu'))
32 model.add(Dropout(0.5))
33 model.add(Dense(7))
34 model.add(Activation('sigmoid'))
```

3層捲積

Dense層設計128→64→7
最後一層用Sigmoid函數(0~1)

```
1 model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
```

Loss function:
對七個情緒類別做cross entropy

網路架構
Network Framework

印出模型概況

```
In [7]:  1  print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2367616
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
activation_5 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 455
_________________________________________________________________
activation_6 (Activation)    (None, 7)                 0
=================================================================
Total params: 2,404,967
Trainable params: 2,404,967
Non-trainable params: 0
_________________________________________________________________
None
```

模型訓練
Model Training

```
In [8]:
1  model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
2  training = model.fit_generator(generator=train_generator, steps_per_epoch=2048 // 16,epochs=20,validation_data=validation_gen
3  model.save_weights('models/simple_CNN.h5')
```

```
Epoch 1/20
128/128 [==============================] - 403s 3s/step - loss: 1.9359 - acc: 0.2227 - val_loss: 1.7188 - val_acc: 0.3875
Epoch 2/20
128/128 [==============================] - 366s 3s/step - loss: 1.3686 - acc: 0.5169 - val_loss: 1.2885 - val_acc: 0.6409
Epoch 3/20
128/128 [==============================] - 363s 3s/step - loss: 0.7557 - acc: 0.7354 - val_loss: 1.3372 - val_acc: 0.7151
Epoch 4/20
128/128 [==============================] - 392s 3s/step - loss: 0.4434 - acc: 0.8593 - val_loss: 1.2626 - val_acc: 0.7577
Epoch 5/20
128/128 [==============================] - 362s 3s/step - loss: 0.2567 - acc: 0.9218 - val_loss: 1.3396 - val_acc: 0.7515
Epoch 6/20
128/128 [==============================] - 355s 3s/step - loss: 0.1990 - acc: 0.9408 - val_loss: 1.8091 - val_acc: 0.7517
Epoch 7/20
128/128 [==============================] - 361s 3s/step - loss: 0.1452 - acc: 0.9546 - val_loss: 1.6808 - val_acc: 0.7831
Epoch 8/20
128/128 [==============================] - 366s 3s/step - loss: 0.0910 - acc: 0.9731 - val_loss: 1.9618 - val_acc: 0.7794
Epoch 9/20
128/128 [==============================] - 356s 3s/step - loss: 0.0909 - acc: 0.9731 - val_loss: 2.1357 - val_acc: 0.7899
Epoch 10/20
128/128 [==============================] - 357s 3s/step - loss: 0.0769 - acc: 0.9766 - val_loss: 2.5581 - val_acc: 0.7707
Epoch 11/20
128/128 [==============================] - 355s 3s/step - loss: 0.0714 - acc: 0.9810 - val_loss: 2.1419 - val_acc: 0.7726
Epoch 12/20
128/128 [==============================] - 341s 3s/step - loss: 0.0422 - acc: 0.9834 - val_loss: 2.1237 - val_acc: 0.7931
Epoch 13/20
128/128 [==============================] - 335s 3s/step - loss: 0.0450 - acc: 0.9849 - val_loss: 2.0546 - val_acc: 0.7979
Epoch 14/20
128/128 [==============================] - 351s 3s/step - loss: 0.0328 - acc: 0.9902 - val_loss: 2.2051 - val_acc: 0.7967
Epoch 15/20
128/128 [==============================] - 348s 3s/step - loss: 0.0325 - acc: 0.9897 - val_loss: 2.7477 - val_acc: 0.7726
Epoch 16/20
128/128 [==============================] - 331s 3s/step - loss: 0.0447 - acc: 0.9888 - val_loss: 2.5791 - val_acc: 0.7843
Epoch 17/20
128/128 [==============================] - 334s 3s/step - loss: 0.0309 - acc: 0.9927 - val_loss: 2.2718 - val_acc: 0.7886
Epoch 18/20
128/128 [==============================] - 336s 3s/step - loss: 0.0194 - acc: 0.9946 - val_loss: 2.6559 - val_acc: 0.7824
Epoch 19/20
128/128 [==============================] - 391s 3s/step - loss: 0.0341 - acc: 0.9902 - val_loss: 2.6205 - val_acc: 0.7831
Epoch 20/20
128/128 [==============================] - 355s 3s/step - loss: 0.0243 - acc: 0.9931 - val_loss: 2.7904 - val_acc: 0.7831
```

1. 優化器**optimizer**使用ramsprop。

2. **model.fit_generator**

   Python的生成器，逐個生成資料的batch並進行訓練。

   生成器與模型將並存執行以提高效率。

3. 所有訓練資料forward+backward後更新參數的過程，

   epochs設置為20。

4. **model.save_weights(filepath)**

   將模型權重儲存到指定路徑，檔案類型是HDF5。

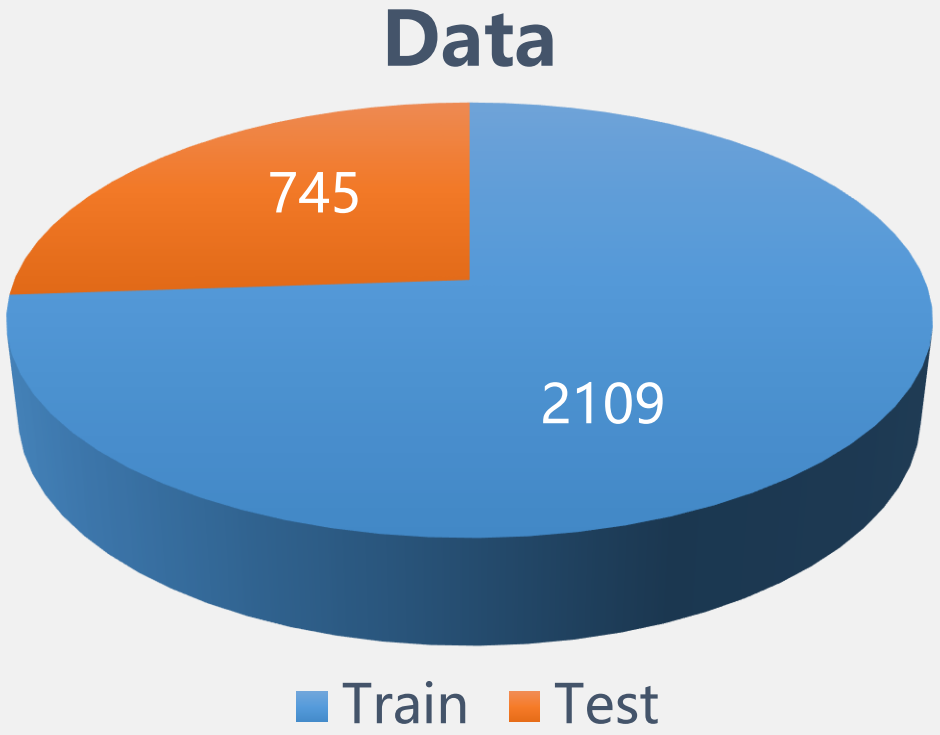
資料視覺化
Data Visualization

使用matplotlib套件製圖

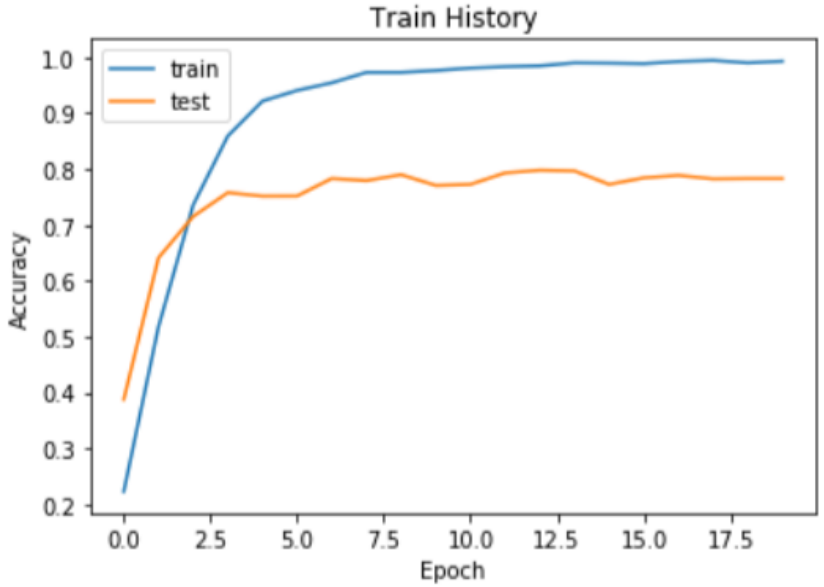
In [9]:
```
1  import matplotlib.pyplot as plt
2  def show_train_history(train_acc,test_acc):
3      plt.plot(training.history[train_acc])
4      plt.plot(training.history[test_acc])
5      plt.title('Train History')
6      plt.ylabel('Accuracy')
7      plt.xlabel('Epoch')
8      plt.legend(['train', 'test'], loc='upper left')
9      plt.show()
```

結果
Result

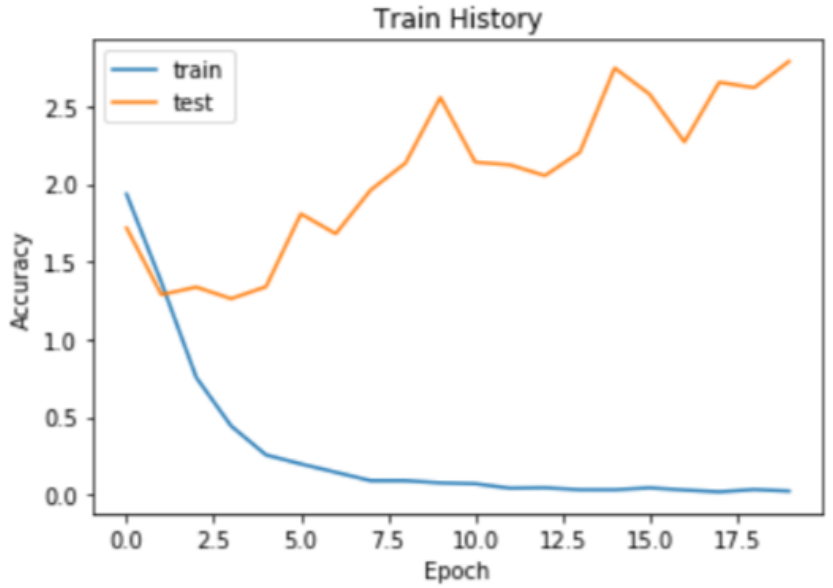
平均準確率可達74%




```
In [10]: 1 show_train_history('acc','val_acc')
```




```
In [11]: 1 show_train_history('loss','val_loss')
```



https://github.com/jennychiou/Class_AI/tree/master/Final%20Project