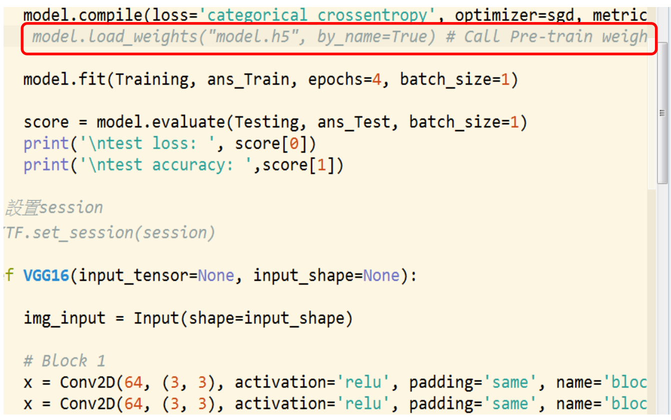
機器學習 (Project 2)

Q1: 方法描述 (Method Description)

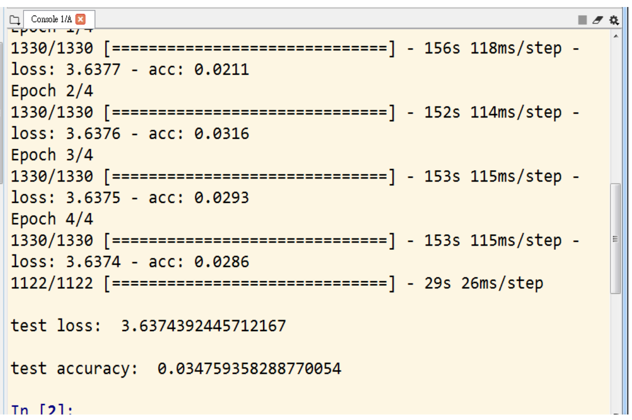
Ans: 這次的Project是利用VGG16來對於之前寫的python人臉辨識做更進一步的延伸。我的程式大致和上次一樣，是針對前35張當作Training的測試資料，並且以一個陣列來記錄那是屬於哪個資料夾(即正確答案)，剩下的則當作是Testing的資料來看準確率。其中我比較了VGG16在Fully Connected的時候使用1024、512、256的Node，去跑4個Epoch所產生不同的結果。結論和第八章所學得的知識相似，皆是說明了Model如果越廣，正確率就會越高。另外，我還有使用sgd來調整我的Learning Rate和動量等等，讓Model在收斂的時候更加精準。

Q2: 程式結果 (Experimental Results - Accuracy)

1. Keras (不含 Pre-train Model)



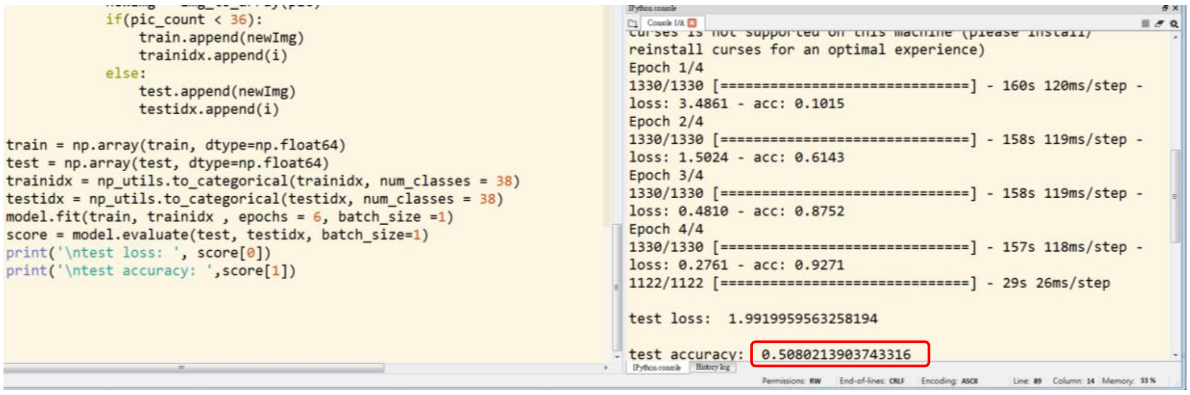
```
  model.compile(loss='categorical_crossentropy', optimizer=sgd, metric
  model.load_weights("model.h5", by_name=True) # Call Pre-train weigh

  model.fit(Training, ans_Train, epochs=4, batch_size=1)

  score = model.evaluate(Testing, ans_Test, batch_size=1)
  print('\ntest loss: ', score[0])
  print('\ntest accuracy: ',score[1])

設置session
TF.set_session(session)

f VGG16(input_tensor=None, input_shape=None):

  img_input = Input(shape=input_shape)

  # Block 1
  x = Conv2D(64, (3, 3), activation='relu', padding='same', name='bloc
  x = Conv2D(64, (3, 3), activation='relu', padding='same', name='bloc
```



```
1330/1330 [==============================] - 156s 118ms/step -
loss: 3.6377 - acc: 0.0211
Epoch 2/4
1330/1330 [==============================] - 152s 114ms/step -
loss: 3.6376 - acc: 0.0316
Epoch 3/4
1330/1330 [==============================] - 153s 115ms/step -
loss: 3.6375 - acc: 0.0293
Epoch 4/4
1330/1330 [==============================] - 153s 115ms/step -
loss: 3.6374 - acc: 0.0286
1122/1122 [==============================] - 29s 26ms/step

test loss:  3.6374392445712167

test accuracy:  0.034759358288770054

In [2]:
```

Pre-train Model 已被註解掉(紅框框處)，Train 出來的 Model 在 Training 和 Testing 的準確率都很低(約 3%)。說明了 Fine Tuning 的重要性。

2. Keras (含 Pre-train Model)

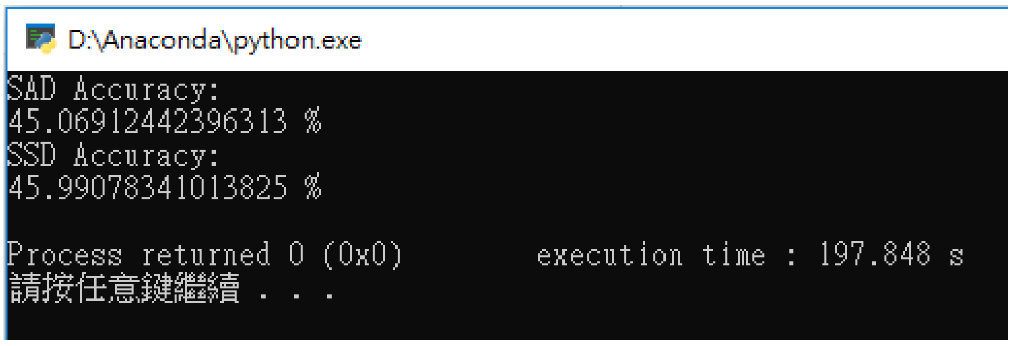


```
        if(pic_count < 36):
            train.append(newImg)
            trainidx.append(i)
        else:
            test.append(newImg)
            testidx.append(i)

train = np.array(train, dtype=np.float64)
test = np.array(test, dtype=np.float64)
trainidx = np_utils.to_categorical(trainidx, num_classes = 38)
testidx = np_utils.to_categorical(testidx, num_classes = 38)
model.fit(train, trainidx , epochs = 6, batch_size =1)
score = model.evaluate(test, testidx, batch_size=1)
print('\ntest loss: ', score[0])
print('\ntest accuracy: ',score[1])
```

```
reinstall curses for an optimal experience)
Epoch 1/4
1330/1330 [==============================] - 160s 120ms/step -
loss: 3.4861 - acc: 0.1015
Epoch 2/4
1330/1330 [==============================] - 158s 119ms/step -
loss: 1.5024 - acc: 0.6143
Epoch 3/4
1330/1330 [==============================] - 158s 119ms/step -
loss: 0.4810 - acc: 0.8752
Epoch 4/4
1330/1330 [==============================] - 157s 118ms/step -
loss: 0.2761 - acc: 0.9271
1122/1122 [==============================] - 29s 26ms/step

test loss:  1.9919959563258194

test accuracy:  0.5080213903743316
```

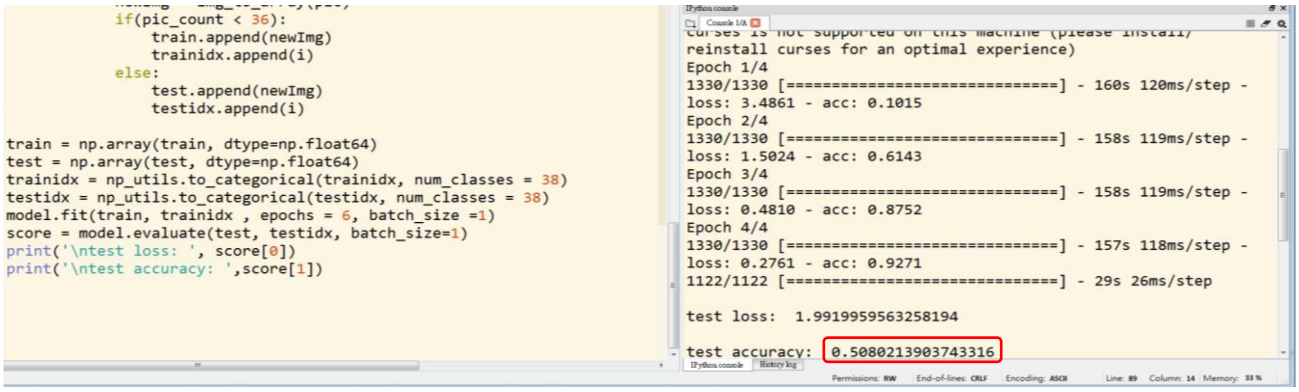
3. 與上次純 Python Code 結果比較

上次:



```
D:\Anaconda\python.exe
SAD Accuracy:
45.06912442396313 %
SSD Accuracy:
45.99078341013825 %

Process returned 0 (0x0)        execution time : 197.848 s
請按任意鍵繼續 . . .
```

這次: 1024 個 Nodes (Epoch = 4)

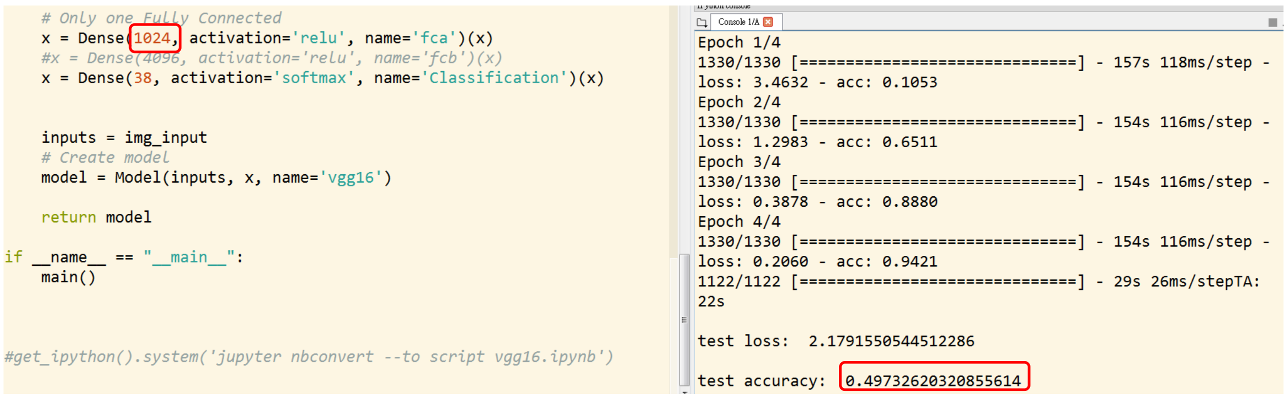


```
        if(pic_count < 36):
            train.append(newImg)
            trainidx.append(i)
        else:
            test.append(newImg)
            testidx.append(i)

train = np.array(train, dtype=np.float64)
test = np.array(test, dtype=np.float64)
trainidx = np_utils.to_categorical(trainidx, num_classes = 38)
testidx = np_utils.to_categorical(testidx, num_classes = 38)
model.fit(train, trainidx , epochs = 6, batch_size =1)
score = model.evaluate(test, testidx, batch_size=1)
print('\ntest loss: ', score[0])
print('\ntest accuracy: ',score[1])
```

```
reinstall curses for an optimal experience)
Epoch 1/4
1330/1330 [==============================] - 160s 120ms/step -
loss: 3.4861 - acc: 0.1015
Epoch 2/4
1330/1330 [==============================] - 158s 119ms/step -
loss: 1.5024 - acc: 0.6143
Epoch 3/4
1330/1330 [==============================] - 158s 119ms/step -
loss: 0.4810 - acc: 0.8752
Epoch 4/4
1330/1330 [==============================] - 157s 118ms/step -
loss: 0.2761 - acc: 0.9271
1122/1122 [==============================] - 29s 26ms/step

test loss:  1.9919959563258194

test accuracy:  0.5080213903743316
```

這次使用 Keras，利用了 Neural Network 在 VGG16 的建構之下，Accuracy 有了顯著的提升。相信若是能跑更多 Epoch 和增加 Model 的深度，能夠再把 Accuracy 創造新高。這次證明了 Train 一個 Model 會比直接用圖像轉成數字比對，再用 SAD 和 SSD 找 Nearest Neighbor 來的更加準確。

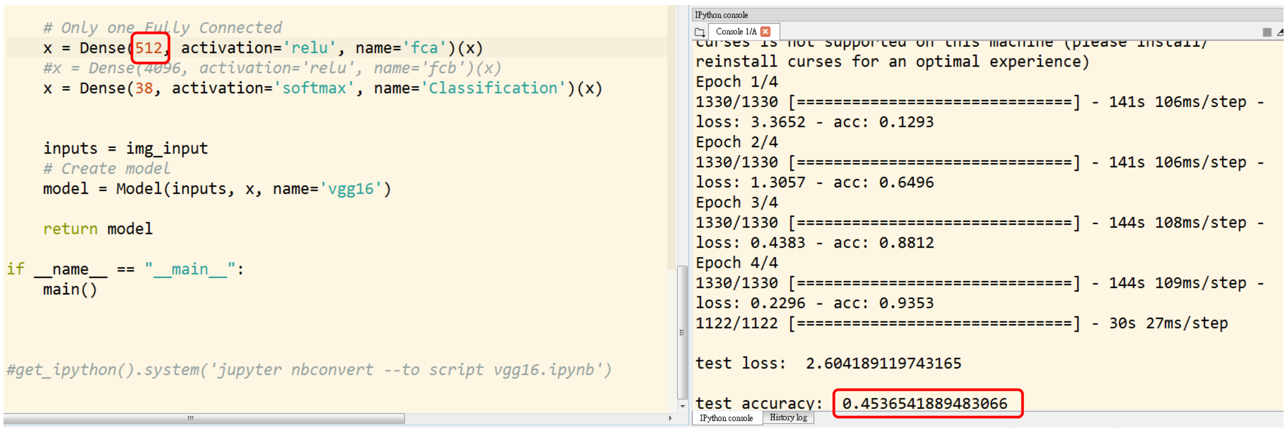
4. Fully Connected → 1024 個 Nodes (Epoch = 4)



```
    # Only one Fully Connected
    x = Dense(1024, activation='relu', name='fca')(x)
    #x = Dense(4096, activation='relu', name='fcb')(x)
    x = Dense(38, activation='softmax', name='Classification')(x)


    inputs = img_input
    # Create model
    model = Model(inputs, x, name='vgg16')

    return model

if __name__ == "__main__":
    main()


#get_ipython().system('jupyter nbconvert --to script vgg16.ipynb')
```

```
Epoch 1/4
1330/1330 [==============================] - 157s 118ms/step -
loss: 3.4632 - acc: 0.1053
Epoch 2/4
1330/1330 [==============================] - 154s 116ms/step -
loss: 1.2983 - acc: 0.6511
Epoch 3/4
1330/1330 [==============================] - 154s 116ms/step -
loss: 0.3878 - acc: 0.8880
Epoch 4/4
1330/1330 [==============================] - 154s 116ms/step -
loss: 0.2060 - acc: 0.9421
1122/1122 [==============================] - 29s 26ms/stepTA:
22s

test loss:  2.1791550544512286

test accuracy:  0.49732620320855614
```

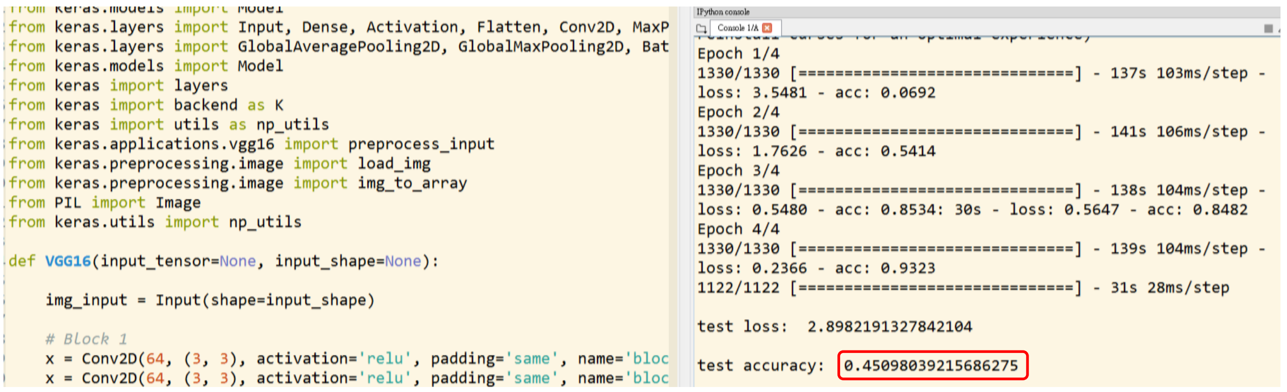
Fully Connected → 512 個 Nodes (Epoch = 4)



```
    # Only one Fully Connected
    x = Dense(512, activation='relu', name='fca')(x)
    #x = Dense(4096, activation='relu', name='fcb')(x)
    x = Dense(38, activation='softmax', name='Classification')(x)


    inputs = img_input
    # Create model
    model = Model(inputs, x, name='vgg16')

    return model

if __name__ == "__main__":
    main()


#get_ipython().system('jupyter nbconvert --to script vgg16.ipynb')
```

```
curses is not supported on this machine (please install/
reinstall curses for an optimal experience)
Epoch 1/4
1330/1330 [==============================] - 141s 106ms/step -
loss: 3.3652 - acc: 0.1293
Epoch 2/4
1330/1330 [==============================] - 141s 106ms/step -
loss: 1.3057 - acc: 0.6496
Epoch 3/4
1330/1330 [==============================] - 144s 108ms/step -
loss: 0.4383 - acc: 0.8812
Epoch 4/4
1330/1330 [==============================] - 144s 109ms/step -
loss: 0.2296 - acc: 0.9353
1122/1122 [==============================] - 30s 27ms/step

test loss:  2.604189119743165

test accuracy:  0.4536541889483066
```

Fully Connected → 256 個 Nodes (Epoch = 4)



```
from keras.models import Model
from keras.layers import Input, Dense, Activation, Flatten, Conv2D, MaxP
from keras.layers import GlobalAveragePooling2D, GlobalMaxPooling2D, Bat
from keras.models import Model
from keras import layers
from keras import backend as K
from keras import utils as np_utils
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from PIL import Image
from keras.utils import np_utils

def VGG16(input_tensor=None, input_shape=None):

    img_input = Input(shape=input_shape)

    # Block 1
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='bloc
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='bloc
```

```
reinstall curses for an optimal experience)
Epoch 1/4
1330/1330 [==============================] - 137s 103ms/step -
loss: 3.5481 - acc: 0.0692
Epoch 2/4
1330/1330 [==============================] - 141s 106ms/step -
loss: 1.7626 - acc: 0.5414
Epoch 3/4
1330/1330 [==============================] - 138s 104ms/step -
loss: 0.5480 - acc: 0.8534: 30s - loss: 0.5647 - acc: 0.8482
Epoch 4/4
1330/1330 [==============================] - 139s 104ms/step -
loss: 0.2366 - acc: 0.9323
1122/1122 [==============================] - 31s 28ms/step

test loss:  2.8982191327842104

test accuracy:  0.45098039215686275
```

由上述三種，可以比較出，若是 Model 的廣度越大(利用第八章所學來

Optimize)，準確率會提升。

Q3: 遇到的困難 (Discussion of Difficulty or Problem Encountered):

1. 由於我的GPU容量不夠，因此在跑VGG16的時候，Fully Connected最大的Node數只能設在1024。而且在Training時，Batch Size也只能訂在1。這部分的Bug是最難De(因為要try & error)，也是最耗費我時間的一環(常常會跑到一半掛掉@@)。
2. 中間有一陣子我的Accuracy一直穩定在兩趴左右，但再將預設的Adam的Optimizer改成SGD以及做完Normalized之後，Accuracy有了顯著的提升。
3. 一開始給的model.load_weights("model.h5")不能用，會有ERROR，直到問同學之後才發現後面要加上by_name=True才可以使用。
4. x = Dense(12, activation='softmax', name='Classification')(x)的12要改為38，因為這次資料有38種正確答案。