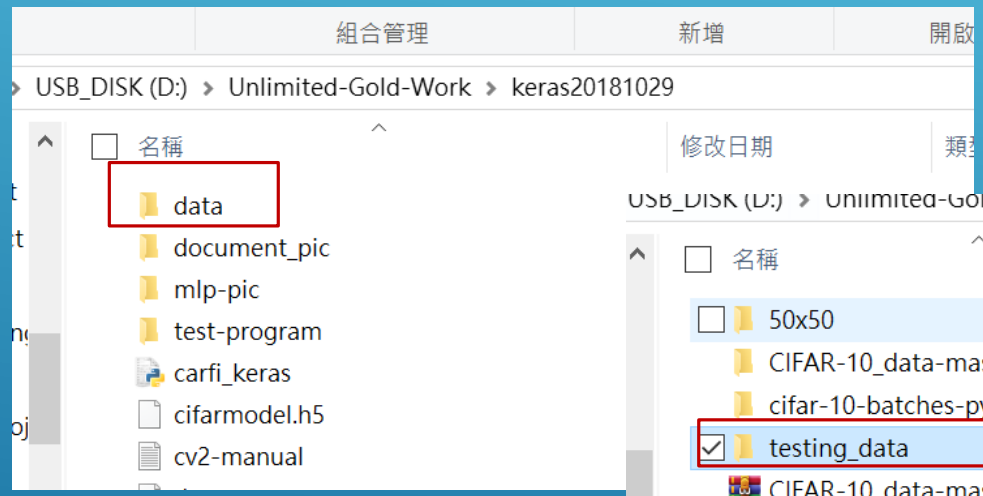
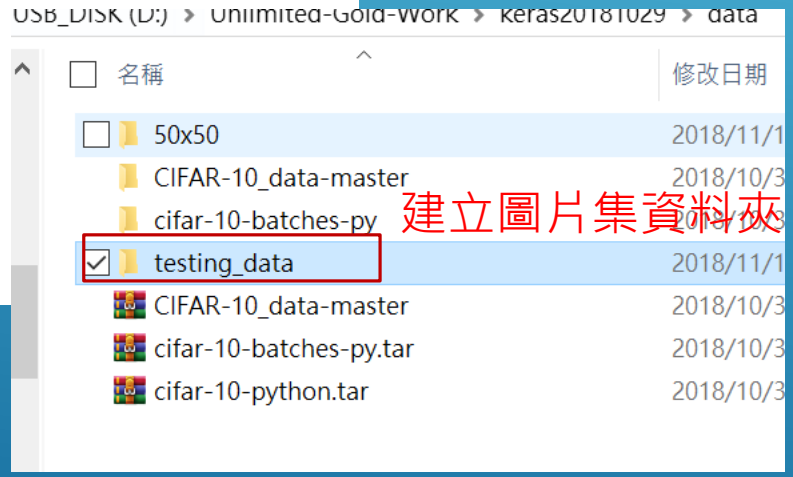
MLP KERAS

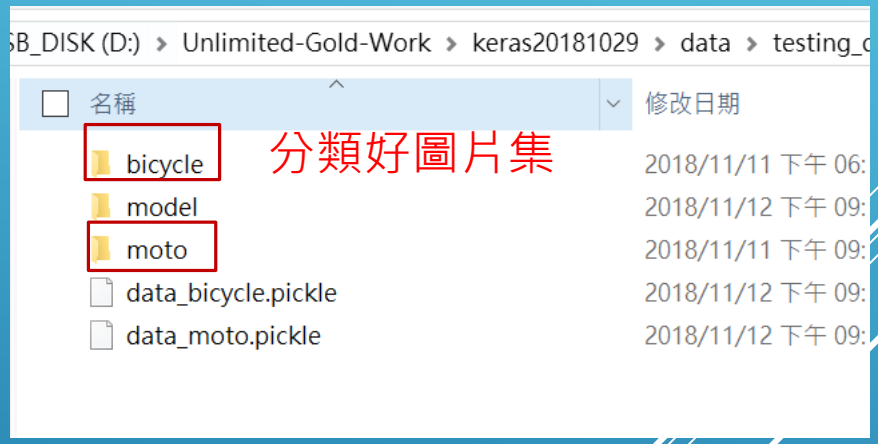
著作:110516026 鄭濠營

後面備註:以上言論只是開開玩笑,這種破程式隨便拿去,我只是想說這話過過癮

data
|_自建一個圖片集資料夾
|_ 建立資料夾且分類好圖片集







檔案路徑分類

流程介紹

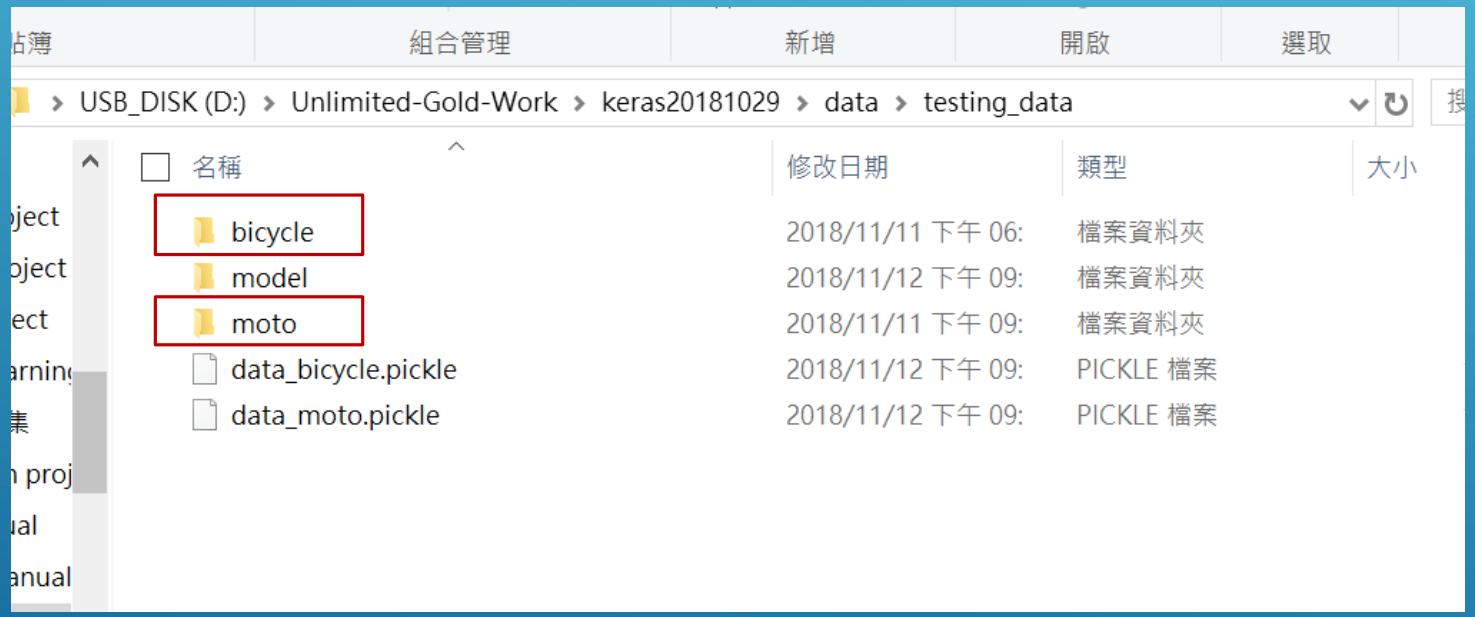
流程簡介

(1)先建立好路徑及分類好圖片集
(2)尋找圖片集路徑 → 接著讀取圖片 → 圖大小改為一致
(3)分析圖片二進位資料(0~255) → 一個個儲存於陣列變數
(4)資料預處理 → 先將資料0~255對應為0~1 → 型態轉換 float→label進行one-hot-encoding處理 → 訓練圖庫資料以及其標籤值以dictionary格式儲存→並壓縮成pickle檔
(5)建立模型[後面再談]→訓練模型
(6)儲存模型→之後就可預測模型

函式介紹

皆放在main.py之中

findfile(filename)
作用為搜尋分類好的圖片集,即是底下的 bicyble && moto



設計的函式種類

readData(filename)
圖片修改統一大小,讀取圖片資料,0~255表示,
之後儲存到陣列裡面

packingData(data)
將圖片資訊壓縮成一個檔案,格式採用
dictionary(類似json),資訊有
filename,datas,labels這三種key值

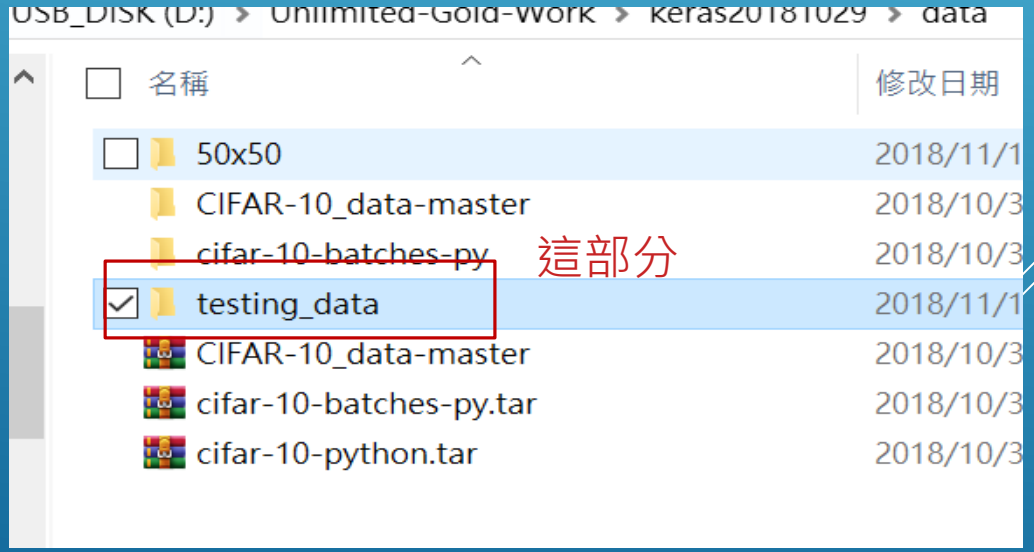
one-hot(x,n)
x指的是儲存labels的陣列,n代表輸出值數量
顧名思義進行one-hot-encoding的轉換

loadData()
負責把壓縮的pickle檔讀取,相當於讀取裡面的dictionary資料格式,並且進行0~255→0~1正規化,one-hot-encoding也在此進行,然後分成Xtrain,Ytrain陣列分別儲存,皆為二維陣列

input(filepath)
這是模型訓練完成之後,可用圖片路徑來讀取圖片並轉換輸入神經元進行預測

如果模型訓練完,且儲存完畢,可用
model_reuse.py將儲存的模型載入利用

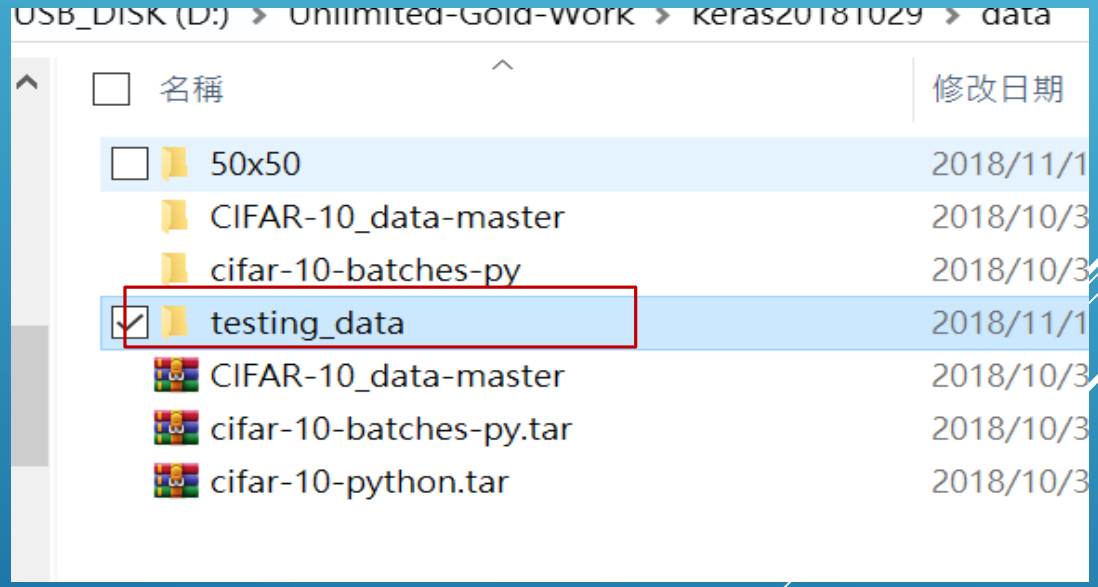
同樣有input() function可以實作
datapath(filename)
作用為找尋模型儲存路徑,filename請輸入圖片
集上層的資料夾,也就是



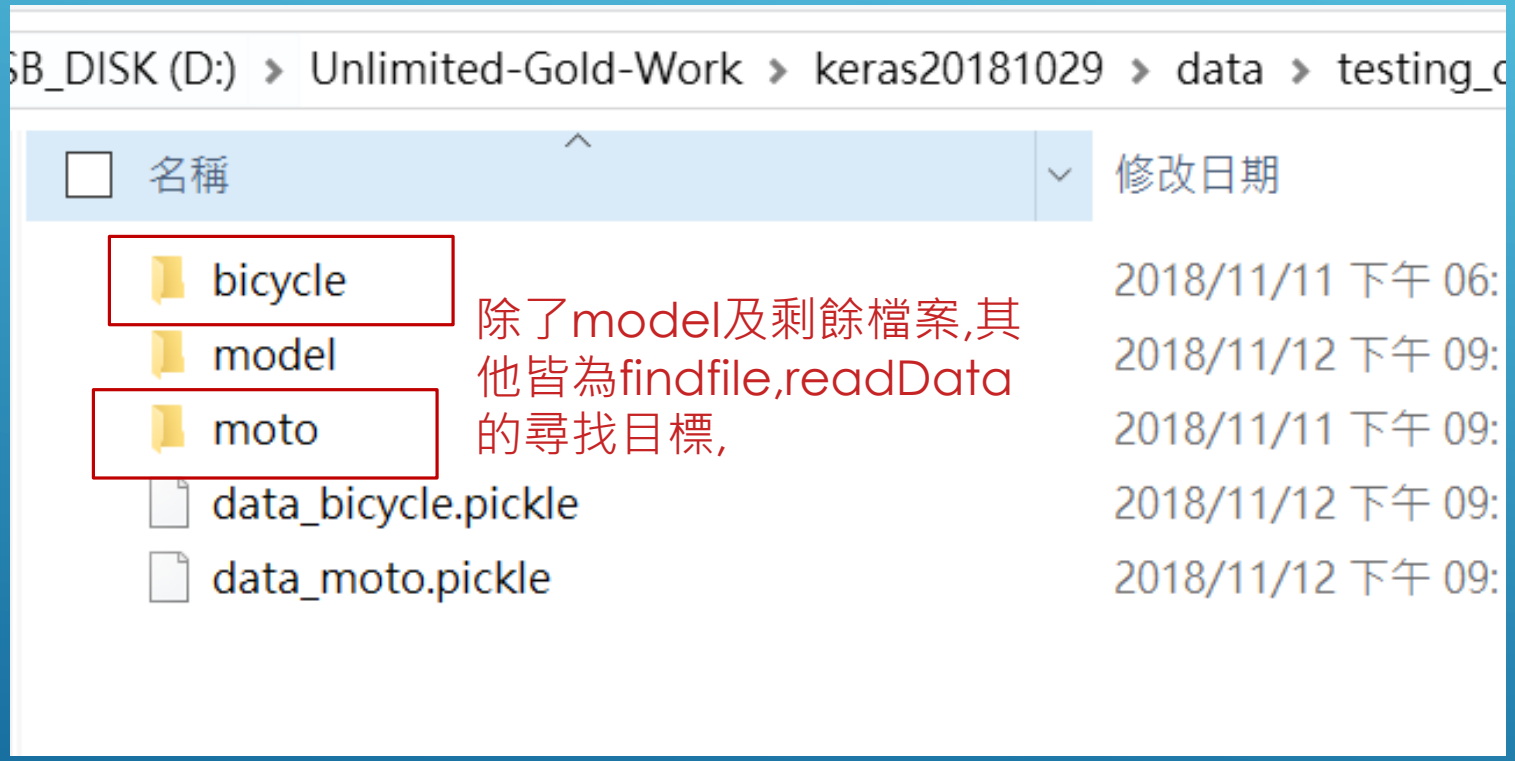
實際實作

```
findfile('testing_data')
loadData()
```

參數裡填入圖片集最上層資料夾



```
findfile: data\testing_data\bicycle
readData: data\testing_data\bicycle
```



除了model及剩餘檔案,其他皆為findfile,readData的尋找目標,

p.s. 如果你看得懂程式碼就隨便你動ㄌ@@

切記: model資料夾名稱請勿更動

以下為readData所做的事情:將圖片集全數讀入,並存成2-dim陣列

```
filelist:
data\testing_data\bicycle\cycle1.jpg
data\testing_data\bicycle\cycle2.jpg
data\testing_data\bicycle\cycle3.jpg
data\testing_data\bicycle\cycle4.jpg
data\testing_data\bicycle\cycle5.jpg
data\testing_data\bicycle\cycle6.jpg
data\testing_data\bicycle\cycle7.jpg
```

```
npimg: (1024,)
data: (7, 1024)
```

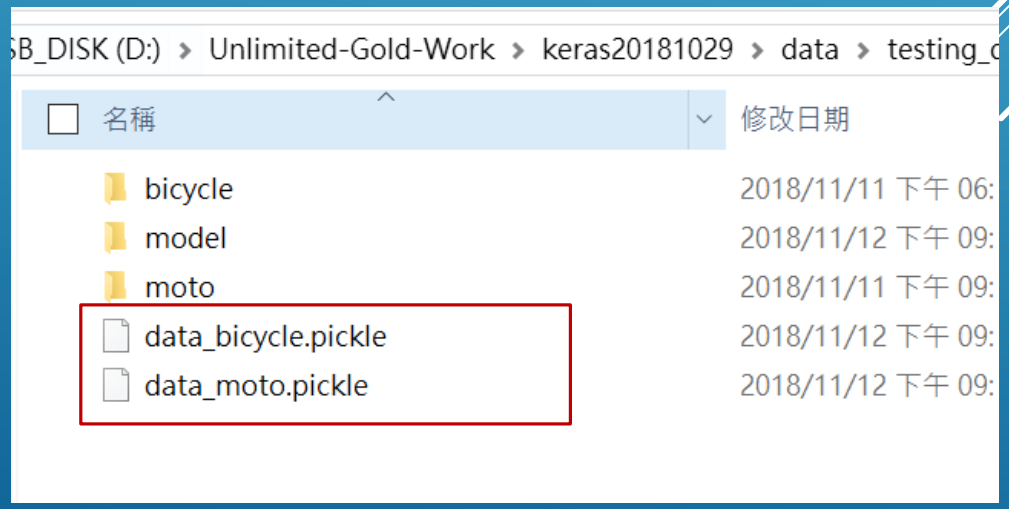
npimg是顯示單張圖片資料長度

data是顯示圖片集資料長度,(7張,1024[32x32])

```python
def packingData(data):
    global filelen
    print('data:',np.array(data).shape)
    print('datalen:',len(data))
    print('file:',currentfile)
    print('datadir:',data_dir)
    labels = [filelen]*len(data)
    dict = {'filename':"data_%s"%currentfile,'datas':data,'labels':labels}
    with open(data_dir+"\\data_%s.pickle"%currentfile, 'wb') as f:
        pickle.dump(dict, f, protocol=pickle.HIGHEST_PROTOCOL)
    filelen+=1
```

這部分就是壓縮檔,姑且設為副檔名pickle,代表其是用pickle module進行壓縮(有附api 文件檔,自行參考)

檔案名我已經定義好ㄌ,如果你看得懂且想改,不阻撓你。



```
pickleData: dict_keys(['filename', 'datas', 'labels'])
labels: [0, 0, 0, 0, 0, 0, 0]
```

這裡是loadData所顯示的部分,其就是負責載入pickle檔案
可以看到pickleData顯示的就是pickle檔內部儲存的dictionary資料型態
Labels 設置標籤分類,代表這7張屬於第0類

```
label: [[0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

←這部分就是one-hot-encoding轉換的部分,簡而言之,就是其他為0,只有一類為1的轉換方式(以索引值來代表分類號碼,然後屬於其分類號碼就設置1,其他為0

```
datas: (14, 1024)
labels: (14, 2)
```

從loadData這部分處理完後,可以看到,這是已經載入所有圖片集的資料陣列型態

datas (14張,1024[32x32]圖檔)

labels (14張,2種分類)

```
model:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 262,914
Trainable params: 262,914
Non-trainable params: 0
_________________________________________________________________
None
```

這部分就是模型的內容摘要,可以知道我建了3層
輸入層(1024神經元)→隱藏層(256神經元)→輸出層(2神經元)
Param是這模型每一層的參數總量256*1024+256

```
 - 0s - loss: 1.3251 - acc: 0.5714
Epoch 2/20
 - 0s - loss: 0.4604 - acc: 0.7143
Epoch 3/20
 - 0s - loss: 0.1844 - acc: 1.0000
Epoch 4/20
 - 0s - loss: 0.0480 - acc: 1.0000
Epoch 5/20
 - 0s - loss: 0.0479 - acc: 1.0000
Epoch 6/20
 - 0s - loss: 0.0202 - acc: 1.0000
Epoch 7/20
 - 0s - loss: 0.0158 - acc: 1.0000
Epoch 8/20
 - 0s - loss: 0.0105 - acc: 1.0000
Epoch 9/20
 - 0s - loss: 0.0088 - acc: 1.0000
Epoch 10/20
 - 0s - loss: 0.0073 - acc: 1.0000
Epoch 11/20
 - 0s - loss: 0.0066 - acc: 1.0000
Epoch 12/20
 - 0s - loss: 0.0061 - acc: 1.0000
```

這邊是訓練過程,可以看到損失率及準確率,目前訓練20次。
這邊因為我只是用於測試,圖庫少而且單一,所以訓練很快,並不會讓電腦難以分辨及計算。

```
Epoch 13/20
 - 0s - loss: 0.0054 - acc: 1.0000
Epoch 14/20
 - 0s - loss: 0.0048 - acc: 1.0000
Epoch 15/20
 - 0s - loss: 0.0045 - acc: 1.0000
Epoch 16/20
 - 0s - loss: 0.0042 - acc: 1.0000
Epoch 17/20
 - 0s - loss: 0.0038 - acc: 1.0000
Epoch 18/20
 - 0s - loss: 0.0035 - acc: 1.0000
Epoch 19/20
 - 0s - loss: 0.0033 - acc: 1.0000
Epoch 20/20
 - 0s - loss: 0.0031 - acc: 1.0000

14/14 [==============================] - 0s 2ms/step
```

```
scores: [0.00292932568117976 2, 1.0]
```

這是使用keras用於評估的函式
model.evaluate(測試用圖片集,labels)

可以看到其值 [錯誤率,準確度] 的比較

```
predicts = model.predict(input('D:\\Unlimited-Gold-Work\\keras20181029\\data\\testing_data\\moto\\moto2.jpg')
```

```
input: (1, 1024)
predicts: [0 0 0 0 0]
one_predict: [0.0, 1.0]
this is a moto
```

之後就可以爽爽的預測啦!
input先載入你要辨識的圖檔,路徑要正確(最好是用絕對路徑最穩)
這邊可以看到input 的2-dim陣列型態(1,1024)是為了與測試圖片維度需一致而做調整。
predicts是利用測試圖片前5張(我前五張都是來自同一分類)來預測(這部分可以略過)
one_predict可以清楚知道其圖片屬於哪一類(one-hot)
之後就可以看到顯示'這是摩托車'的字樣啦

THE END

備註:目前想不到要補充甚麼,想知道甚麼可以問,寫這程式,我另有寫了幾個python內部函式庫的API文件檔,有些程式不懂請自行去看API文件檔(可能寫得不清楚,請見諒)

另外我大概會把這程式上傳到github,之後要用就去github載—github名’ Unlimited-Gold-Work’。