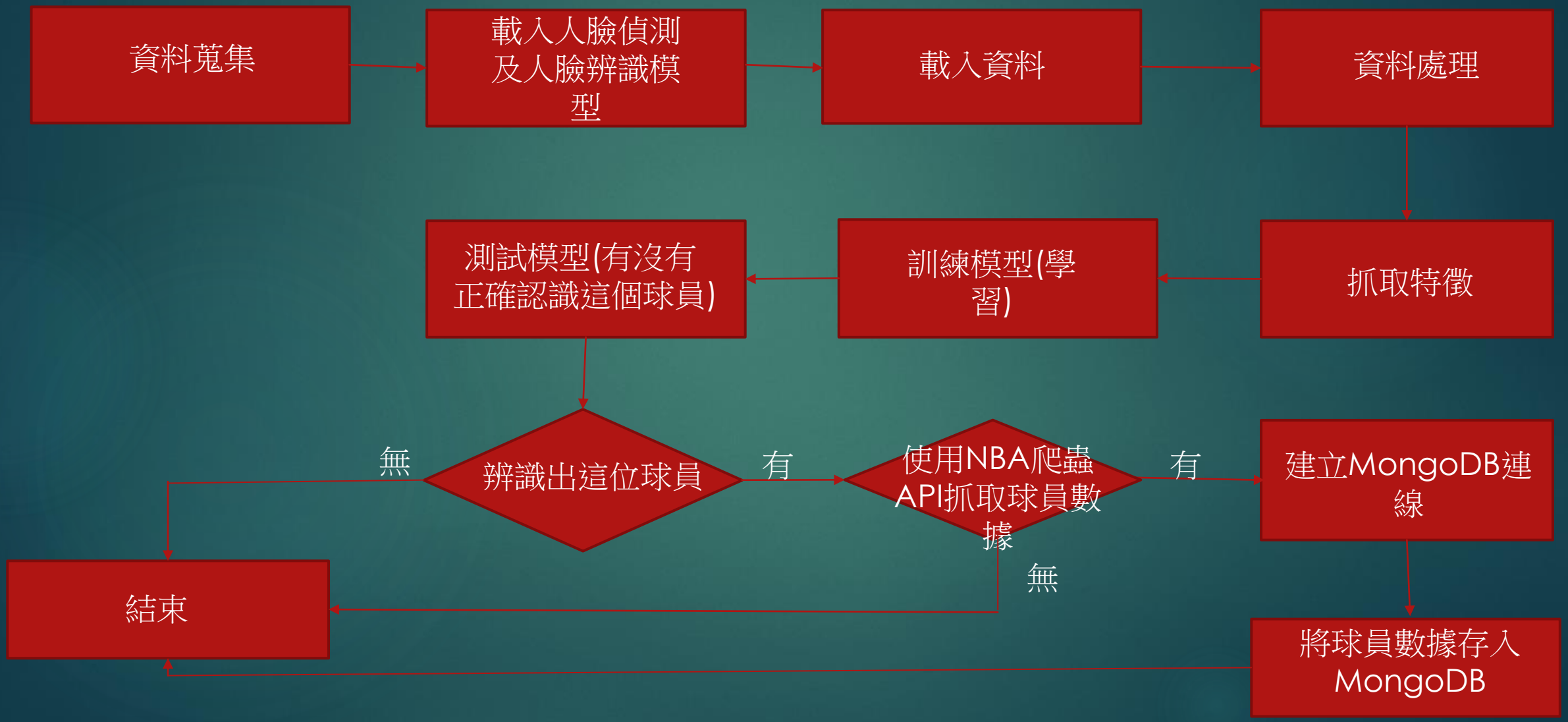
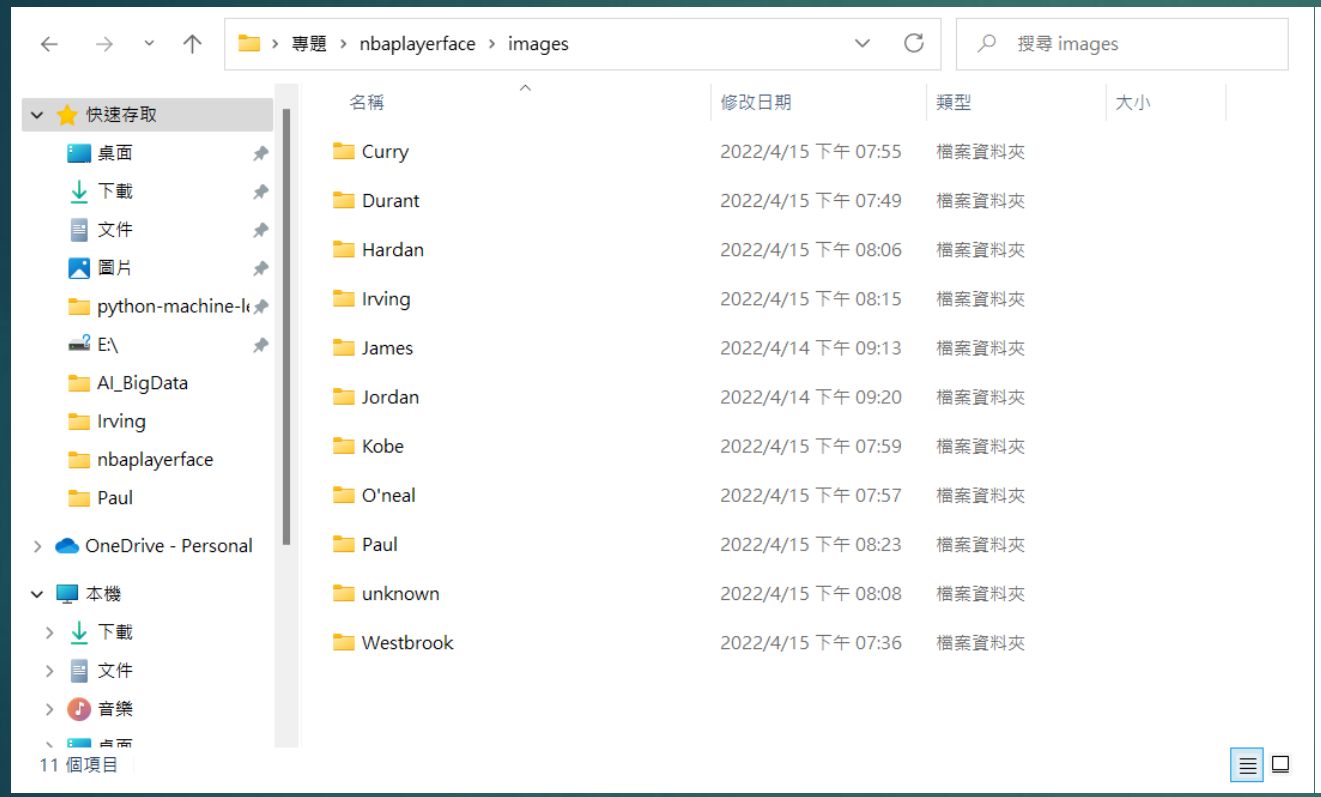
NBA人臉辨識結合MongoDB抓取球員數據

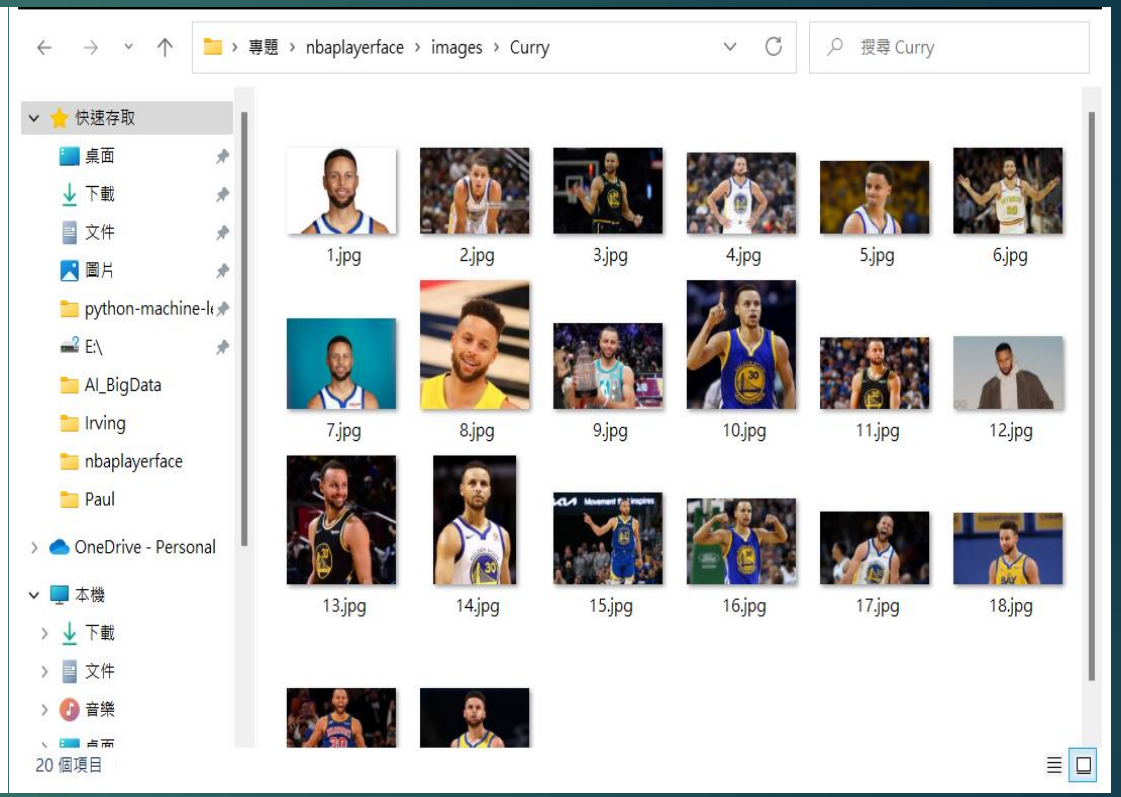
姓名:林劭謙

系統流程圖


資料蒐集
載入人臉偵測及人臉辨識模型
載入資料
資料處理
抓取特徵
訓練模型(學習)
測試模型(有沒有正確認識這個球員)
辨識出這位球員
無
有
使用NBA爬蟲API抓取球員數據
有
無
建立MongoDB連線
將球員數據存入MongoDB
結束

資料蒐集

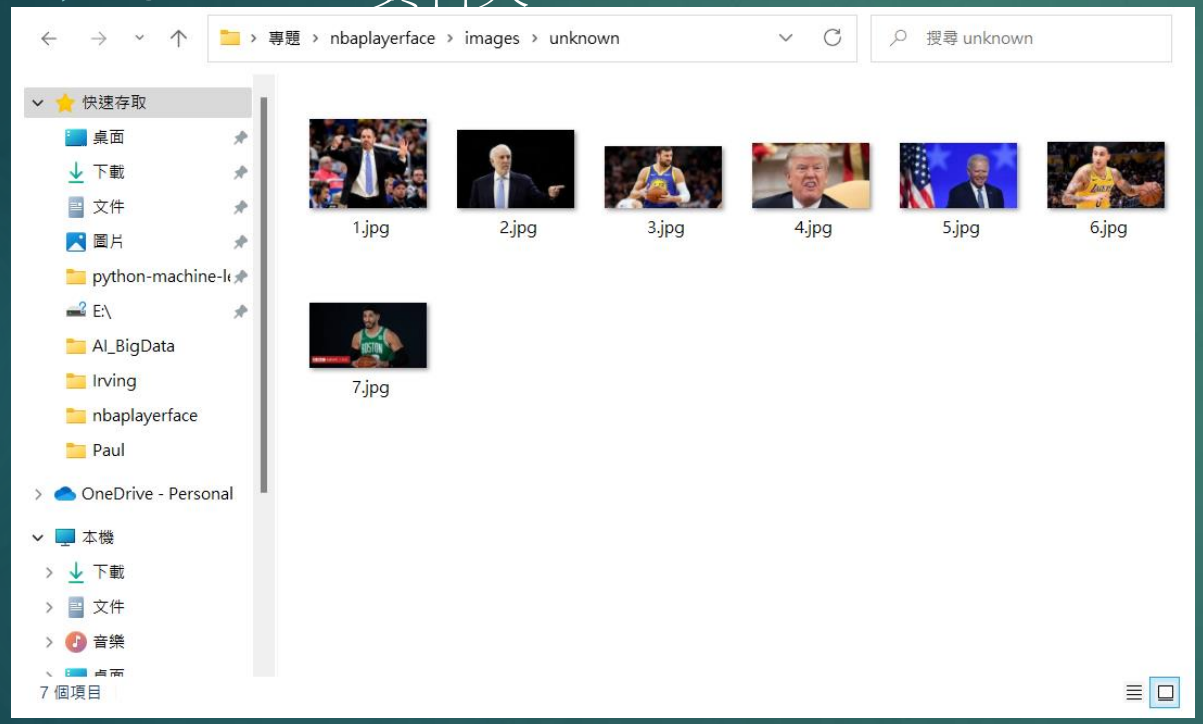




一開始進行資料蒐集的步驟，資料集裡面包含11個資料夾代表11個類別，11個資料夾中的其中10個資料夾裡面個包含了20張圖片，只有unknown這個資料夾裡面只有7張圖片，因為這個類別當模型辨識不出這個人是誰的時候模型會告訴我們這個人他不認識

資料蒐集

- Unknown資料夾



載入相關套件

```
from imutils import paths
import numpy as np
import imutils
import pickle
import cv2
import os
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
import pickle
from pymongo import MongoClient
from bson.objectid import ObjectId
from nba_api.stats.static import players
from nba_api.stats.endpoints import playergamelog
```

```
import numpy as np
import imutils
import pickle
import cv2
import os
from pymongo import MongoClient
from bson.objectid import ObjectId
from nba_api.stats.static import players
from nba_api.stats.endpoints import playergamelog
```

載入人臉偵測和人臉辨識所需的模型

```
print("[INFO] loading face detector...")
protoPath = os.path.sep.join(["face_detection_model", "deploy.prototxt"])
```

```
modelPath = os.path.sep.join(["face_detection_model",
                              "res10_300x300_ssd_iter_140000.caffemodel"])
```

```
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
```

這裡使用opencv的dnn模組裡的readNetFromCaffe()的方法載入人臉偵測模型

```
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch("openface_nn4.small2.v1.t7")
```

這裡使用opencv的dnn模組裡的readNetFromTorch()的方法載入人臉辨識模型

載入資料

```
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images("images"))
```

資料處理與抓取特徵

```
knownEmbeddings = []
knownNames = []
total = 0

for (i, imagePath) in enumerate(imagePaths):
    print("[INFO] processing image {}/{}".format(i + 1,
          len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]
    image = cv2.imread(imagePath)
    image = imutils.resize(image, width=600)
    (h, w) = image.shape[:2]
    imageBlob = cv2.dnn.blobFromImage(
        cv2.resize(image, (300, 300)), 1.0, (300, 300),
        (104.0, 177.0, 123.0), swapRB=False, crop=False)
    detector.setInput(imageBlob)
    detections = detector.forward()
```

```
if len(detections) > 0:
    i = np.argmax(detections[0, 0, :, 2])
    confidence = detections[0, 0, i, 2]
    if confidence > 0.5:
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        face = image[startY:endY, startX:endX]
        (fH, fW) = face.shape[:2]
        if fW < 20 or fH < 20:
            continue
        faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
            (96, 96), (0, 0, 0), swapRB=True, crop=False)
        embedder.setInput(faceBlob)
        vec = embedder.forward()
        knownNames.append(name)
        knownEmbeddings.append(vec.flatten())
```

```
        total += 1

print("[INFO] serializing {} encodings...".format(total))
data = {"embeddings": knownEmbeddings, "names": knownNames}
f = open("output/embeddings.pickle", "wb")
f.write(pickle.dumps(data))
f.close()
```

資料處理與抓取特徵完的結果

- 這裡顯示已經處理的圖片，從第1張圖片開始至第207張圖片為止，最後會儲存類別與圖片(這邊代表模型已偵測到人臉的特徵有哪些)

```
[INFO] processing image 1/207
[INFO] processing image 2/207
[INFO] processing image 3/207
[INFO] processing image 4/207
[INFO] processing image 5/207
[INFO] processing image 6/207
[INFO] processing image 7/207
[INFO] processing image 8/207
[INFO] processing image 9/207
[INFO] processing image 10/207
[INFO] processing image 11/207
[INFO] processing image 12/207
[INFO] processing image 13/207
[INFO] processing image 14/207
[INFO] processing image 15/207
[INFO] processing image 16/207
```

```
[INFO] processing image 196/207
[INFO] processing image 197/207
[INFO] processing image 198/207
[INFO] processing image 199/207
[INFO] processing image 200/207
[INFO] processing image 201/207
[INFO] processing image 202/207
[INFO] processing image 203/207
[INFO] processing image 204/207
[INFO] processing image 205/207
[INFO] processing image 206/207
[INFO] processing image 207/207
[INFO] serializing 207 encodings...
```

訓練模型做分類-1

- 先載入特徵與類別的pickle檔(結果如下圖)

```
print("[INFO] loading face embeddings...")
data = pickle.loads(open("output/embeddings.pickle", "rb").read())
```

```
[INFO] loading face embeddings...

{'embeddings': [array([-0.02697866, -0.01140192,  0.15159383,  0.10230408,  0.
08386499,
         0.22711067,  0.07953247, -0.05183394,  0.06916276, -0.03941859,
        -0.02437588,  0.07519679, -0.04693742, -0.15765455,  0.1082544 ,
         0.00747738, -0.08317701,  0.07181611, -0.06101048, -0.08025689,
         0.12358229,  0.01023045, -0.05819665,  0.18311031,  0.1047626 ,
        -0.05008847, -0.1716526 , -0.20552301, -0.0191577 ,  0.01746286,
         0.21363196,  0.09001563, -0.11043113,  0.07864767,  0.09523012,
         0.06341249,  0.01063039, -0.10434045,  0.04066655, -0.01615046,
         0.21859503, -0.04137753,  0.02397081, -0.05398086, -0.01446376,
        -0.13718706,  0.07092501, -0.06378015, -0.01830357, -0.10041986,
         0.05131753, -0.13457908,  0.07774848, -0.0476139 ,  0.09262847,
         0.12957424, -0.02818065,  0.11303969, -0.0341872 , -0.0367706 ,
        -0.09735327,  0.03043699,  0.00194297, -0.21455161,  0.14983563,
         0.0398602 ,  0.05475112, -0.09130552, -0.11663487,  0.12796883,
```

訓練模型做分類-2

- 載入LabelEncoder()進行類別的編碼(結果如下圖):

```
print("[INFO] encoding labels...")
le = LabelEncoder()
labels = le.fit transform(data["names"])
```

```
[INFO] encoding labels...

array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,
        4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,  5,  5,  5,
        5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
        5,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  8,  8,  8,  8,  8,  8, 10, 10, 10, 10, 10, 10, 10,
        9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        9,  9,  9], dtype=int64)
```

訓練模型做分類-3

- 使用sklearn SVC()進行分類

```
print("[INFO] training model...")
recognizer = SVC(C=1.0, kernel="linear", probability=True)
recognizer.fit(data["embeddings"], labels)
```

訓練模型做分類-4

- 將分類結果和類別分別儲存在不同的pickie檔裡(後續進行辨識新的圖片時會用到)

```
f = open("output/recognizer.pickle", "wb")
f.write(pickle.dumps(recognizer))
f.close()
```

```
f = open("output/le.pickle", "wb")
f.write(pickle.dumps(le))
f.close()
```

測試模型

- 因為辨識新圖片需要偵測人臉的特徵，所以將偵測人臉的模型載入:

```
print("[INFO] loading face detector...")
protoPath = os.path.sep.join(["face_detection_model", "deploy.prototxt"])
```

```
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
```

- 也需要辨識人臉的模型:

```
modelPath = os.path.sep.join(["face_detection_model",
                              "res10_300x300_ssd_iter_140000.caffemodel"])
```

```
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch("openface_nn4.small2.v1.t7")
```

測試模型

- 把剛剛分類的結果載入

```
recognizer = pickle.loads(open("output/recognizer.pickle", "rb").read())
le = pickle.loads(open("output/le.pickle", "rb").read())
```

- 載入要測試的圖片

```
image = cv2.imread("LeBron.jpg")
image = imutils.resize(image, width=600)
(h, w) = image.shape[:2]
```

```
imageBlob = cv2.dnn.blobFromImage(
    cv2.resize(image, (300, 300)), 1.0, (300, 300),
    (104.0, 177.0, 123.0), swapRB=False, crop=False)
```

測試模型

- 進行人臉辨識(當我信心度大於50%將圖片與辨識結果顯示出來)

```
detector.setInput(imageBlob)
detections = detector.forward()

for i in range(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.5:
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        face = image[startY:endY, startX:endX]
        (fH, fW) = face.shape[:2]

        if fW < 20 or fH < 20:
            continue
```

測試模型

- 進行人臉辨識

```
faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255, (96, 96),
    (0, 0, 0), swapRB=True, crop=False)
embedder.setInput(faceBlob)
vec = embedder.forward()

preds = recognizer.predict_proba(vec)[0]
j = np.argmax(preds)
proba = preds[j]
name = le.classes_[j]

text = "{}: {:.2f}%".format(name, confidence * 100)
y = startY - 10 if startY - 10 > 10 else startY + 10
cv2.rectangle(image, (startX, startY), (endX, endY),
    (0, 255, 0), 2)
cv2.putText(image, text, (startX, y),
    cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)
```

```
print("Show Image...")
cv2.imshow("Image", image
cv2.waitKey(0)
```

測試模型

- 人臉辨識結果(信心度達99.99%)

MongoDB連線與抓取球員資料

- 建立與MongoDB的連線(建立資料庫與資料表)

```
conn = MongoClient("mongodb://localhost")
db = conn.NBAPlayerData
collection = db.playerData
```

- 抓取球員數據(這裡使用NBA釋出的API抓取球員資料)

```
playerid = [str(player["id"]) for player in players.get_active_players()]
for ids in playerid:
    playergamelogs = playergamelog.PlayerGameLog(ids).get_dict()['resultSets'][0]
    headers = playergamelogs['headers']
    data = playergamelogs['rowSet']
    games = [game for game in data]
```

MongoDB連線與抓取球員資料

- 將抓到的球員數據更新到MongoDB裡

```
for i in range(len(headers)):
    for j in range(len(games)):
        if len(games) != 0: # 出賽場數
            myplayer = {headers[i]:games[j][i]}
            updateplayer = {"$set": {headers[i]:games[j][i]}}
            collection.update_one(myplayer, updateplayer, True)
```

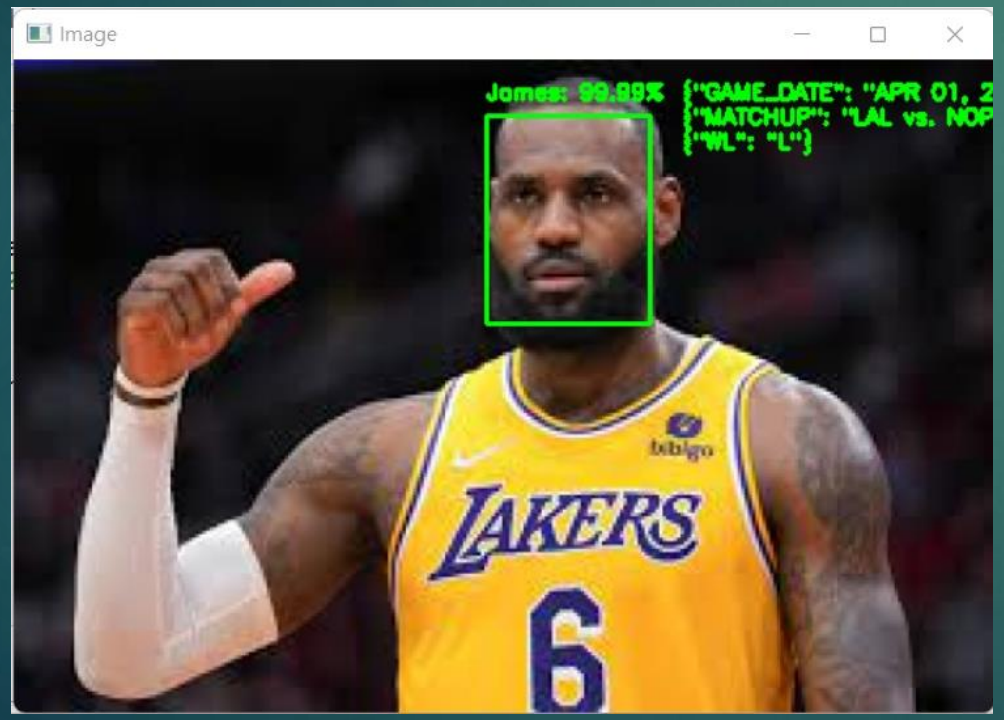
```
cursor = collection.find()
data = [d for d in cursor]
print(data)
```

- 顯示的結果(這裡的數據太多只擷取其中的部分)

```
[{'_id': ObjectId('626a8a0e752d23247653bb90'), 'SEASON_ID': '22021'}, {'_id': ObjectId('626a8a0e752d
23247653bbda'), 'Player_ID': 1630173}, {'_id': ObjectId('626a8a0e752d23247653bc24'), 'Game_ID': '002
2101226'}, {'_id': ObjectId('626a8a0e752d23247653bc26'), 'Game_ID': '0022101206'}, {'_id': ObjectId
('626a8a0e752d23247653bc28'), 'Game_ID': '0022101197'}, {'_id': ObjectId('626a8a0e752d23247653bc2
a'), 'Game_ID': '0022101182'}, {'_id': ObjectId('626a8a0e752d23247653bc2c'), 'Game_ID': '002210117
2'}, {'_id': ObjectId('626a8a0e752d23247653bc2e'), 'Game_ID': '0022101151'}, {'_id': ObjectId('626a8
a0e752d23247653bc30'), 'Game_ID': '0022101141'}, {'_id': ObjectId('626a8a0e752d23247653bc32'), 'Game
_ID': '0022101127'}, {'_id': ObjectId('626a8a0e752d23247653bc34'), 'Game_ID': '0022101109'}, {'_id':
ObjectId('626a8a0e752d23247653bc36'), 'Game_ID': '0022101095'}, {'_id': ObjectId('626a8a0e752d232476
53bc38'), 'Game_ID': '0022101076'}, {'_id': ObjectId('626a8a0e752d23247653bc3a'), 'Game_ID': '002210
1069'}, {'_id': ObjectId('626a8a0e752d23247653bc3c'), 'Game_ID': '0022101051'}, {'_id': ObjectId('62
6a8a0e752d23247653bc3e'), 'Game_ID': '0022101044'}, {'_id': ObjectId('626a8a0e752d23247653bc40'), 'G
ame_ID': '0022101028'}, {'_id': ObjectId('626a8a0e752d23247653bc42'), 'Game_ID': '0022101009'}, {'_i
d': ObjectId('626a8a0e752d23247653bc44'), 'Game_ID': '0022101003'}, {'_id': ObjectId('626a8a0e752d23
247653bc46'), 'Game_ID': '0022100989'}, {'_id': ObjectId('626a8a0e752d23247653bc48'), 'Game_ID': '00
22100962'}, {'_id': ObjectId('626a8a0e752d23247653bc4a'), 'Game_ID': '0022100458'}, {'_id': ObjectId
('626a8a0e752d23247653bc4c'), 'Game_ID': '0022100942'}, {'_id': ObjectId('626a8a0e752d23247653bc4
e'), 'Game_ID': '0022100928'}, {'_id': ObjectId('626a8a0e752d23247653bc50'), 'Game_ID': '002210072
```

MongoDB與辨識結果做結合

- 將辨識結果與MongoDB裡的資料做連結，只要辨識出這位球員時就顯示他本賽季的數據(這裡以LeBron James做測試，已退休的球員則沒有數據)




```
[{'_id': ObjectId('626a8a0e752d23247653bb90'), 'SEASON_ID': '22021'}]
[{'_id': ObjectId('626a8ba1752d23247658e4ee'), 'Player_ID': 2544}]
[{'_id': ObjectId('626a8a1c752d232476541ca6'), 'Game_ID': '0022101160'}]
[{'_id': ObjectId('626a8a0e752d23247653bcc0'), 'GAME_DATE': 'APR 01, 2022'}]
[{'_id': ObjectId('626a8a24752d232476544498b'), 'MATCHUP': 'LAL vs. NOP'}]
[{'_id': ObjectId('626a8a0e752d23247653bdc7'), 'WL': 'L'}]
[{'_id': ObjectId('626a8a0e752d23247653be16'), 'MIN': 40}]
[{'_id': ObjectId('626a8a11752d23247653cffd'), 'FGM': 13}]
[{'_id': ObjectId('626a8a20752d232476543b02'), 'FGA': 23}]
[{'_id': ObjectId('626a8a20752d232476543b94'), 'FG_PCT': 0.565}]
[{'_id': ObjectId('626a8a0e752d23247653bf91'), 'FG3M': 3}]
[{'_id': ObjectId('626a8a0e752d23247653bfe8'), 'FG3A': 8}]
[{'_id': ObjectId('626a8a0e752d23247653c03d'), 'FG3_PCT': 0.375}]
[{'_id': ObjectId('626a8a17752d23247653f7a9'), 'FTM': 9}]
[{'_id': ObjectId('626a8a12752d23247653d26b'), 'FTA': 12}]
[{'_id': ObjectId('626a8a0e752d23247653c141'), 'FT_PCT': 0.75}]
[{'_id': ObjectId('626a8a0e752d23247653c175'), 'OREB': 0}]
[{'_id': ObjectId('626a8a0e752d23247653c1d2'), 'DREB': 8}]
[{'_id': ObjectId('626a8a0e752d23247653c224'), 'REB': 8}]
```

結論與未來展望

- 本作品是將球員辨識出來後再去跟資料庫裡的資料做比對，再將該球員在資料庫裡的資料抓出來並顯示。
- MongoDB的主要用處是將NBA的爬蟲API將資料爬取下來後儲存(因為MongoDB是大數據資料庫可以儲存的資料量大，且儲存的速度快)。
- 在蒐集資料的部分可以與爬蟲做結合，先用爬蟲爬取大量的球員圖片形成資料集，再建立模型進行訓練，最後測試辨識出來的結果正不正確，然後再去跟資料庫裡的資料做比對(因為這裡再資料庫裡已有資料)，再將該球員在資料庫裡的資料抓出來並顯示。
- 整個專題只有軟體的部分，未來可以跟硬體(如樹莓派等等)做結合搭配視訊去辨識現實中的球員並將其資料顯示出來。

報告結束。