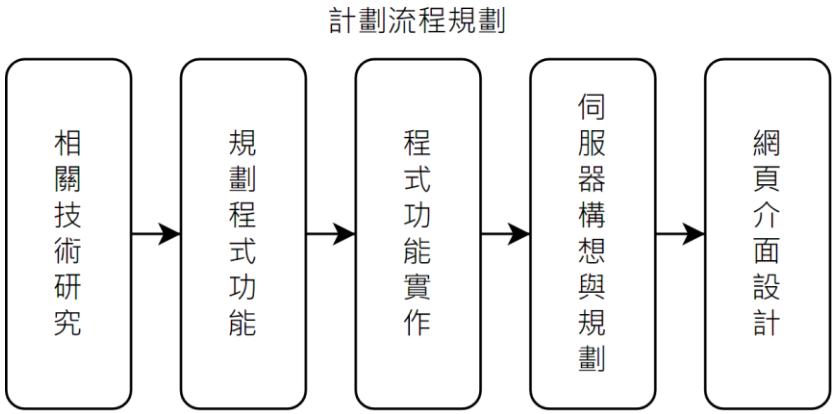
人臉識別帳號註冊及登入系統：
OpenCV 人臉識別技術及 Flask 伺服器框架

作者：
四資管一　B11109001　詹竣翔
四資管一　B11109029　林晏亘

壹、研究背景及目的

隨著近年來機器學習的理論愈發成熟，許多由機器學習發展而成的技術，如：影像辨識、自然語言等，日漸融入的人們的日常生活中。而我們生活在這個人工智慧快速發展的時代，也不能被這項趨勢拋在腦後，所以我們打算結合人臉識別技術，透過模塊化的編程模式，打造一個輕量的人臉識別及帳戶註冊及登入系統。

為了實現這個目標，我們將整個計劃分成下列幾個階段：



圖一：計劃流程規劃

貳、系統架構

「人臉識別帳號註冊及登入系統」可拆分為三個架構：處理人臉識別相關功能的「人臉識別模塊」、負責提供帳戶管理及相關 API 的「伺服器模塊」，以及負責與使用者互動的「使用者介面」。將系統分為以上架構，並開放幾種固定的方法有助於團隊的協作及系統的開發。

參、程式設計

一、使用套件

1. Flask：輕量級 Web 框架，構建 Web 應用。
2. OpenCV-Python 及 OpenCV-Contrib-Python：運用影像處理技術、物件識別功能、人臉辨識算法。在舊版本中有使用鏡頭讀取與視頻處理功能。
3. Numpy：處理多維數組和矩陣（影像像素矩陣），進行高效運算。

二、人臉識別模塊

在人臉辨識模塊方面，大致能夠分為三個階段，分別為：擷取特徵影像、影像標記與資料集建置、影像辨識。

- 擷取影像特徵階段：收取由前端獲得的影像，根據 CascadeClassifier （物件辨識分類器）讀取資源檔（決定辨識影像的哪個部分），並擷取人臉特徵部分（判斷影像中的人臉是否只有一人，若否則回傳 false，是則繼續進行），並將該影像存進以用戶命名的資料夾中。

- 影像標記與資料集建置階段：在這個階段中分為兩個功能，首先確認用戶影像資料夾後，對資料夾中的影像逐一加入準備訓練的陣列中，並且根據用戶標籤。第二功能，則是將本階段第一步功能完成的訓練陣列，使用 OpenCV 的 LBPHFaceRecognizer（人臉辨識算法）物件，訓練影像並儲存訓練數據。

- 影像辨識階段：與第一階段的裁減方式相同，擷取辨識階段時前端獲取的影像，讀取第二階段訓練完成的資料數據，進行人臉辨識。若是匹配數值小於 50，則代表成功；反之則失敗。

最後在初始化檔案中，將以上三個階段的分別檔案，集合成一個 Database 類別，簡化伺服器端功能呼叫過程。

```python
class Database:
    db: BuildDB
    re: Recognizer
    def __init__(self) -> None:
        self.db = BuildDB()
        self.re = Recognizer()

    def cut_pics(self, username, pics) -> None:
        cp = CutPics(username)
        cp.cut(pics)
        self.db.label(username)
        self.db.build()

    def recognize(self, username, image) -> bool:
        image.save(FILENAME)

        is_pass, pass_name = self.re.recognize(FILENAME)
        return is_pass and pass_name == username
```

三、伺服器模塊

在伺服器模塊方面，使用了 Flask 作為伺服器的底層架構，並依照系統需求劃分出了幾種功能：

- 使用者介面：在此系統中，使用者在正常瀏覽網頁的狀況下，使用者會利用此功能向伺服器發送 GET 請求以取得客戶端的使用者介面。

```
@app.route('/', methods=['GET'])
def default():
    '''
    Route for user interface (client side)
    '''
    return app.send_static_file('index.html')
```

- 使用者 Restful 應用程式介面：在使用者瀏覽網頁時，網頁腳本會向伺服器發送不同的訪問方式的請求，並搭配使用 JWT 權杖，以實現：登錄、註冊、查詢等功能。

JWT（JSON Web Token）是基於 JSON 標記語言以及 HMAC 雜湊加密演算法所發展出來的技術，每一個 JWT 權杖都可以被分為：標頭、承載資訊以及簽章，透過這種方式，伺服器不需要使用額外的空間以儲存每個權杖的狀態，而是保留在權杖中，以減少伺服器的負擔。而在程式碼中，會調用「get_token」函數以生成對應的 JWT 權杖，並使用「get_payload」以驗證並解析其中的承載資訊。

```
def get_token(username) -> str:
    # Function creating a credentials about users
    iat = datetime.now()
    payload = { "username": username, "iat": iat.timestamp(),
                "expires": (iat + timedelta(0, 1500)).timestamp() }

    jwt_token = jwt.generateToken(payload)
    return jwt_token

def get_playload(username, token):
    # Function for verifying credentials and getting information inside it
    if not token: return False

    if payload := jwt.decodeToken(token) and payload.get('username') == username:
        if datetime.now().timestamp() < payload.get('expires'):
            return payload
    return False
```

（一）查詢使用者：GET、HEAD 方法

若伺服器接收到了路由如：「/<username>」形式的 GET 請求，伺服器將把其認為是用於查詢指定的使用者是否存在於伺服器中。若請求的標頭中帶有「token」屬性，且其值為合法的 token 格式並通過驗證，則伺服器將會傳送完整的使用者信息；若請求的標頭不帶有「token」屬性，則伺服器將會傳送部份的使用者信息。

若伺服器接收到了路由如：「/<username>」形式的 HEAD 請求，伺服器會在確認使用者是否存在後，回傳相應的狀態：200 OK 表示該使用者存在、404 NOT FOUND 表示使用者不存在。

```
@app.route('/<username>', methods=['GET'])
def getInfo(username):
    '''
    Users Restful Apis - Get user info
    - with no token: Get user info not detailly
    - with token: Verify token and get user info in detail
    '''
    token = request.headers.get('token')
    payload = get_playload(username, token)

    if username not in users:
        return 'null', 404
    user = users[username]

    if payload:
        insert(username, 'INFOMATION', ip_address=request.remote_addr)
        return user, 200
    elif token:
        return 'null', 401

    return {key: user[key] for key in USER_SUMMARY_KEY}, 200

@app.route('/<username>', methods=['HEAD'])
def check(username):
    '''
    Users Restful Apis - Check user existence
    with image - Use Face Recognize Database to authorize
    '''
    return '', 200 if username in users else 404
```

（二）註冊使用者：POST 方法

若伺服器接收到了路由如：「/<username>」形式的 POST 請求，且該請求帶有多個名稱為「images」的圖形檔，以及以「userdata」為屬性名稱的使用者註冊資料，則會利用「人臉識別模塊」以進行影像處理，並存放於資料庫中，而伺服會利用 JWT 技術核發一份使用者權杖；若不具備所規定之條件，則會回傳狀態：401 Unauthorized 表示未能成功註冊。

```
@app.route('/<username>', methods=['POST'])
def register(username):
    '''
    Users Restful Apis - Create User
    with userdata and images - Create a new user and add the photos to the database
    '''

    images = request.files.getlist('images')
    raw_user = json.loads(__str) if (__str := request.form.get('userdata')) else 'null'

    if not raw_user or type(raw_user) != dict:
        return 'null', 401

    db.cut_pics(username, images)

    user = {'name': username, 'log': []}
    for key in USER_INFORMATION_KEY:
        user.setdefault(key, raw_user.get(key, None))

    users.setdefault(username, user)
    insert(username, 'CREATE', username=username)
    insert(username, 'LOGIN', ip_address=request.remote_addr)

    return {"token": get_token(username)}
```

（三）使用者登錄：PUT 方法

若伺服器接收到了路由如：「/<username>」形式的 PUT 請求，且該請求帶有名稱為「image」的圖形檔，則會利用「人臉識別模塊」以進行識別，若識別結果通過，則伺服會利用 JWT 技術核發一份使用者權杖；若不通過則會回傳狀態：401 Unauthorized 表示未能通過驗證。

```python
@app.route('/<username>', methods=['PUT'])
def login(username):
    '''
    Users Restful Apis - Authorize (for user login)
    with image - Use Face Recognize Database to authorize
    '''

    if username not in users:
        return 'null', 404

    image = request.files.get('image')

    if db.recognize(username, image):
        insert(username, 'LOGIN', ip_address=request.remote_addr)
        return {"token": get_token(username)}
    else:
        return 'null', 401
```

（四）註銷使用者：DELETE 方法

若伺服器收到了路由如：「/<username>」形式的 DELETE 請求，若請求的標頭中帶有「token」屬性，且其值為合法的 token 格式並通過驗證，則伺服器將會移除使用者；若不通過則會回傳狀態：401 Unauthorized 表示未能通過驗證。

```python
@app.route('/<username>', methods=['DELETE'])
def delete(username):
    '''
    Users Restful Apis - Delete user
    with token: Delete the user from storage if token is valid
    '''
    if username not in users:
        return 'null', 404

    token = request.headers.get('token')
    payload = get_playload(username, token)
    if not payload or not token:
        return 'null', 401

    del users[username]
    return {"status": "success"}, 200
```

（五）更改使用者資訊：PUT 方法

若伺服器收到了路由如：「/<username>/<key>」形式的 PUT 請求，若請求的標頭中帶有「token」屬性，且其值為合法的 token 格式並通過驗證，以及以「value」為屬性名稱的更改值，則伺服器將會修改使用者的對應屬性；若屬性名稱不應要修改，或未指定更改值，則回傳狀態：403 FORBIDDEN 表示使用者操作不當。

```
@app.route('/<username>/<key>', methods=['PUT'])
def update_information(username, key):
    '''
    Users Restful Apis - Update user information
    with image - Use Face Recognize Database to authorize
    '''
    if username not in users:
        return 'null', 404
    token = request.headers.get('token')
    payload = get_playload(username, token)

    if not payload or not token:
        return 'null', 401

    user = users[username]
    value = request.form.get('value')
    if key in USER_INFORMATION_KEY and value:
        user[key] = value
        insert(username, 'INFO_CHANGE', info_name=key, value=value)
        return {"status": "success"}, 200
    else:
        return 'null', 403
```

四、使用者介面

在使用者介面，調用了媒體截取及串流應用程式介面（Media Capture and Streams API）以取得使用權限，並利用攝像頭截取使用者的臉部圖片以提供伺服器進行註冊及登入。

為簡化開發流程，使用者介面使用了 React 函式庫來開發，透過 react-webcam 套件簡化存取媒體截取及串流應用程式介面的步驟，並結合使用本地存儲（LocalStorage）以保存由伺服器提供的使用者權杖。

在本專案中使用者介面，僅串接了「使用者登錄」、「使用者註冊」及「查詢使用者」功能，期望能夠將其他功能完善。

肆、作品展示

影片連結：https://www.youtube.com/watch?v=vFWINUgNHF4。

影片中，我們示範了：註冊、登錄的功能，為了使示範得到更好的效果，在程式中加入了部份的日誌功能，以體現登錄及註冊的過程。

伍、研究心得

在本次的計劃中，我們發現團隊非常缺乏溝通，這導致了作業初期在功能細節上面團隊初期並未商量完整，增加後續修改頻率且降低專案進度效率。

並且由於溝通問題，直到部份作業完成後才開始使用 GitHub 管理專案，僅管這部份使我們能夠依照原定計畫，快速完成自己負責的部分，但這也同樣的造成了雙方之間在協作上，造成的不同步落差，會有參數傳遞與功能修改等問題無法及時得知。

另外在辨識物件的資源檔篩選部份，根據不同的物件辨識資源檔，會產生不同的成效，有時造成無法辨識出人臉或是讀取到的人臉過多的問題。經過測試後，OpenCV 中，針對人臉提供的四個資源檔中的 haarcascade_frontalface_alt2.xml 資源檔，使用該檔能獲得最佳辨識效果。

這次的使用者介面並沒有完全的展現所有的伺服器功能，而僅僅實作了註冊、登錄以及查看使用者資訊三個部份，之後可以繼續進行完善，使其能夠搭配其他系統，以充當該系統的登入及使用者管理系統。

而經過這一次的作業，我們發現在合作進行專案時，應該事前規劃好過程中的細節，並保持團隊之間的溝通流暢，才能夠順利的完成不同的協作。

陸、參考資料

[1] OpenCV 影像創意邁向 AI 視覺王者歸來（洪錦魁，2022）

[2] haarcascade 优化参数函数 （CSDN，張良玉，2018）

[3] OpenCV: cv::face::LBPHFaceRecognizer Class Reference （OpenCV Document）

[4] Introduction to JSON Web Tokens（JWT.io）

[5] react-webcam（npm）