專案報告書

組員：B11109043 董亦浩、B11109045 謝宇翔、B11109047 曾德祥

程式主題：利用人臉辨識查詢銀行帳戶

程式說明：在選單中有三個選項：

輸入人臉：輸入用戶人臉

訓練模型：讓程式學會辨識人臉。

人臉辨識：程式辨識用戶

實作方式：一個人臉識別系統，並將結果與一個 SQLite 資料庫連接。使用 tkinter 創建了一個 GUI 介面，用於執行以下：

- 收集人臉資料：從用戶的攝像頭讀取影像，使用 OpenCV 的人臉偵測功能來找到影像中的人臉。然後，這些人臉被裁剪出來並儲存到一個指定的資料夾中。
- 訓練人臉識別模型：程式讀取已收集的人臉資料，並將這些資料與相應的標籤一起提供給 OpenCV 的 LBPH 人臉識別模型進行訓練。然後，該模型被儲存到文件中，供以後使用。
- 實施人臉識別：程式讀取從攝像頭獲得的實時影像，並使用已訓練的人臉識別模型來識別影像中的人臉。然後，識別的結果被用於查詢和更新一個 SQLite 資料庫。

使用套件：

- OpenCV (cv2)：用來讀取和寫入影像、轉換影像顏色、偵測臉部、訓練和使用人臉識別模型。
- Os：用來進行與作業系統相關的操作，如建立目錄、列出目錄內容等。
- Numpy：用 numpy 將標籤列表轉換為陣列，供 OpenCV 的人臉識別模型使用。
- Pickle：將標籤字典儲存到一個文件，以便稍後使用。
- sqlite3：建立資料庫連接、執行 SQL 查詢、更新等。
- Tkinter：標準 GUI 庫。用來建立一個具有按鈕的簡單圖形用戶界面。
- Subprocess：用它來運行其他 Python 腳本。

操作簡介：

用戶先輸入人臉→訓練自己的人臉模型→辨識人臉→如此人沒有帳戶則詢問是否新增 or 有帳戶就詢問是否修改餘額

原始碼：github.com

主程式：

```python
import tkinter as tk
import os
import subprocess

# 執行 inputFace.py 腳本的函數
def run_inputFace():
    subprocess.call(["python", "inputFace.py"])

# 執行 model.py 腳本的函數
def run_model():
    subprocess.call(["python", "model.py"])

# 執行 faceID.py 腳本的函數
def run_faceID():
    subprocess.call(["python", "faceID.py"])

# 建立一個視窗
window = tk.Tk()
window.title("程式功能")  # 設定視窗標題

# 建立按鈕並設定其命令
inputFace_button = tk.Button(window, text="輸入人臉", command=run_inputFace)  # 按鈕，執行 inputFace.py
model_button = tk.Button(window, text="訓練模型", command=run_model)  # 按鈕，執行 model.py
faceID_button = tk.Button(window, text="人臉辨識", command=run_faceID)  # 按鈕，執行 faceID.py

# 放置按鈕到視窗中
inputFace_button.pack()
model_button.pack()
faceID_button.pack()

# 開始運行視窗應用程式
window.mainloop()
```

輸入人臉：

```python
import cv2
import random

cap = cv2.VideoCapture(0)
while(not cap.isOpened()):
    cap = cv2.VideoCapture(0)

import os

# 請求用戶輸入名字
name = input("請輸入您的名字：")

# 創建名字對應的資料夾
folder_path = os.path.join('data_dir', name)
if not os.path.exists(folder_path):
    os.makedirs(folder_path)

# 載入人臉分類器
classfier = cv2.CascadeClassifier(cv2.data.haarcascades +"haarcascade_frontalface_alt2.xml")
while(True):
    # 從攝像頭讀取畫面
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 在灰度圖像中檢測人臉
    faceRects = classfier.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=3, minSize=(32, 32))

    if len(faceRects) > 0:
        # 如果檢測到人臉
        for (x, y, w, h) in faceRects:
            # 提取人臉區域
            face = frame[y - 10: y + h + 10, x - 10: x + w + 10]
            # 在原始畫面上畫出人臉矩形框
            cv2.rectangle(frame, (x - 10, y - 10), (x + w + 10, y + h + 10), (0, 255, 0), 2)
            # 生成一個隨機數字作為圖像檔案名稱
            random_number = random.randint(1000000000, 9999999999)
            # 在檢測到人臉時，將圖像保存到對應的資料夾中
            cv2.imwrite(f'data_dir/{name}/{str(random_number)}.jpg', face)

    # 顯示攝像頭畫面
    cv2.imshow('live', frame)

    if cv2.waitKey(1) == ord('q'):
        break

# 釋放攝像頭資源並關閉視窗
cap.release()
cv2.destroyAllWindows()
```

訓練模型：

```python
import cv2
import os
import numpy as np

# 函數：從指定路徑加載訓練數據集
def load_training_data(data_dir):
    images = []
    labels = []
    label_dict = {}

    # 獲取數據集中的子目錄（每個子目錄代表一個人）
    subdirs = [subdir for subdir in os.listdir(data_dir) if os.path.isdir(os.path.join(data_dir, subdir))]

    for i, subdir in enumerate(subdirs):
        label_dict[i] = subdir
        subdir_path = os.path.join(data_dir, subdir)

        # 加載每個子目錄中的圖像文件
        for filename in os.listdir(subdir_path):
            image_path = os.path.join(subdir_path, filename)

            # 讀取圖像並將其轉換為灰度
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            # 將圖像和標籤添加到訓練集中
            images.append(image)
            labels.append(i)

    return images, labels, label_dict

import pickle

def train_face_recognizer(data_dir, model_path):
    # 加載訓練數據集
    images, labels, label_dict = load_training_data(data_dir)

    # 創建LBPH人臉辨識器
    model = cv2.face.LBPHFaceRecognizer_create()

    # 訓練模型
    model.train(images, np.array(labels))

    # 將模型儲存到文件中
    model.save(model_path)

    # 將標籤字典保存到文件中
    with open('label_dict.pkl', 'wb') as f:
        pickle.dump(label_dict, f)

    print("模型訓練完成！")


# 設置數據集目錄和模型文件路徑
data_dir = './data_dir'
model_path = './model.yml'

# 訓練人臉辨識模型
train_face_recognizer(data_dir, model_path)
```

辨識人臉：

```python
import cv2
import pickle
import sqlite3

# 從OpenCV中加載預訓練的人臉級聯分類器
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# 加載用於人臉識別的預訓練模型
# 將 'path_to_model' 替換為實際模型文件的路徑
model = cv2.face.LBPHFaceRecognizer_create()
model.read('./model.yml')

# 從文件中加載標籤字典
with open('label_dict.pkl', 'rb') as f:
    label_dict = pickle.load(f)

# 創建一個空數組來存儲人臉識別結果
recognized_faces = []

def update_balance_by_name(name, new_balance):
    # 連接到數據庫
    conn = sqlite3.connect('bank.db')

    # 創建一個游標對象，用於執行SQL語句
    cursor = conn.cursor()

    # 執行更新操作，將給定姓名的賬戶餘額更新為新的餘額
    cursor.execute("UPDATE bank SET balance=? WHERE name=?", (new_balance, name))

    # 提交事務
    conn.commit()

    # 輸出更新結果
    if cursor.rowcount > 0:
        print(f"{name}賬戶更新後餘額：{new_balance}")

    # 關閉游標和數據庫連接
    cursor.close()
    conn.close()


def query_balance_by_name(name):
    # 連接到數據庫
    conn = sqlite3.connect('bank.db')

    # 創建一个游标对象，用于执行SQL语句
    cursor = conn.cursor()

    # 執行查詢，獲取與給定姓名相同的記錄
    cursor.execute("SELECT balance FROM bank WHERE name=?", (name,))
    records = cursor.fetchall()

    # 輸出查詢結果
    if len(records) > 0:
        for record in records:
            print(f"歡迎用戶: {name}, 賬戶餘額: {record[0]}")
            check = input("是否修改賬戶餘額[1確定請輸入1]")
            if check == "1":
                money = input("請輸入修改金額：")
                update_balance_by_name(name, money)
    else:
        print("無此用戶.")
        # 詢問是否添加新記錄
        check= input("是否添加用戶1[]")
        if check == "1":
            # 獲取新記錄的餘額
            money = input("請輸入餘額：")
            cursor.execute("INSERT INTO bank (name, balance) VALUES (?, ?)", (name, money))
            conn.commit()
            print(f"已添加賬戶{name},餘額為{money}.")

    # 關閉游標和數據庫連接
    cursor.close()
    conn.close()
```

```python
def recognize_faces(frame):

    flag = True
    # 將圖像轉換為灰階
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 檢測圖像中的人臉
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    name = ""

    # 遍歷檢測到的人臉
    for (x, y, w, h) in faces:

        # 提取人臉感興趣區域（ROI）
        face_roi = gray[y:y+h, x:x+w]

        # 將人臉ROI調整為人臉識別模型所需的尺寸
        face_roi = cv2.resize(face_roi, (540, 300))

        # 使用模型預測ROI的標籤和置信度
        label, confidence = model.predict(face_roi)

        # 使用標籤字典將數字標籤轉換為人名
        name = label_dict[label]

        # 在圖像中顯示人名和置信度
        if confidence < 80:
            # 在人臉周圍畫一個矩形
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            text = f'name: {name}, confidence: {int(confidence)}'
            cv2.putText(frame, text, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

            # 將人名添加到recognized_faces數組中
            recognized_faces.append(name)

            # 如果檢測到的人臉次數達到30次，輸出名稱
            if recognized_faces.count(name) >= 10:
                flag = False
    return frame,flag,name

cap = cv2.VideoCapture(0) # 默認的攝像頭編號為0，如果你有多個攝像頭，可以改變這個編號

while True:
    # 讀取一幀圖像
    ret, frame = cap.read()

    if ret:
        # 在圖像上進行人臉識別
        frame,flag,name = recognize_faces(frame)
        # 顯示處理后的圖像
        cv2.imshow('Real-time Face Recognition', frame)

    if cv2.waitKey(1) & 0xFF == ord('q') :
        # 按下 'q' 鍵退出循環
        # 釋放攝像頭資源
        cap.release()
        # 銷毀所有的窗口
        cv2.destroyAllWindows()
        break
    if flag == False:
        # 釋放攝像頭資源
        cap.release()
        # 銷毀所有的窗口
        cv2.destroyAllWindows()
        query_balance_by_name(str(name))
        break
```