KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG - UIT

BÁO CÁO THỰC HÀNH

Lab 5

CÔNG NGHỆ INTERNET OF THINGS HIỆN ĐẠI

THỰC HÀNH: BUILD AN AI APPLICATION IN IOT PLATFORM

GVHD: **Phan Trung Phát**

Lớp: **NT532.O21**

Họ và tên	MSSV
Nguyễn Thành Đăng	21520683
Nguyễn Trần Bảo Quốc	21520421

ĐÁNH GIÁ KHÁC (*):

Nội dung	Kết quả
Tổng thời gian thực hiện bài thực hành trung bình (1)	2 tuần
Link Video thực hiện (2) *(nếu có)*	Link drive demo tổng hợp
Ý kiến (3) *(nếu có)* + Khó khăn + Đề xuất ...	
Điểm tự đánh giá (4)	10/10

(*): phần (1) và (4) bắt buộc thực hiện.

Phần bên dưới là báo cáo chi tiết của nhóm/cá nhân thực hiện.

WARNING LƯU HÀNH NỘI BỘ

© Phan Trung Phát - phatpt@uit.edu.vn

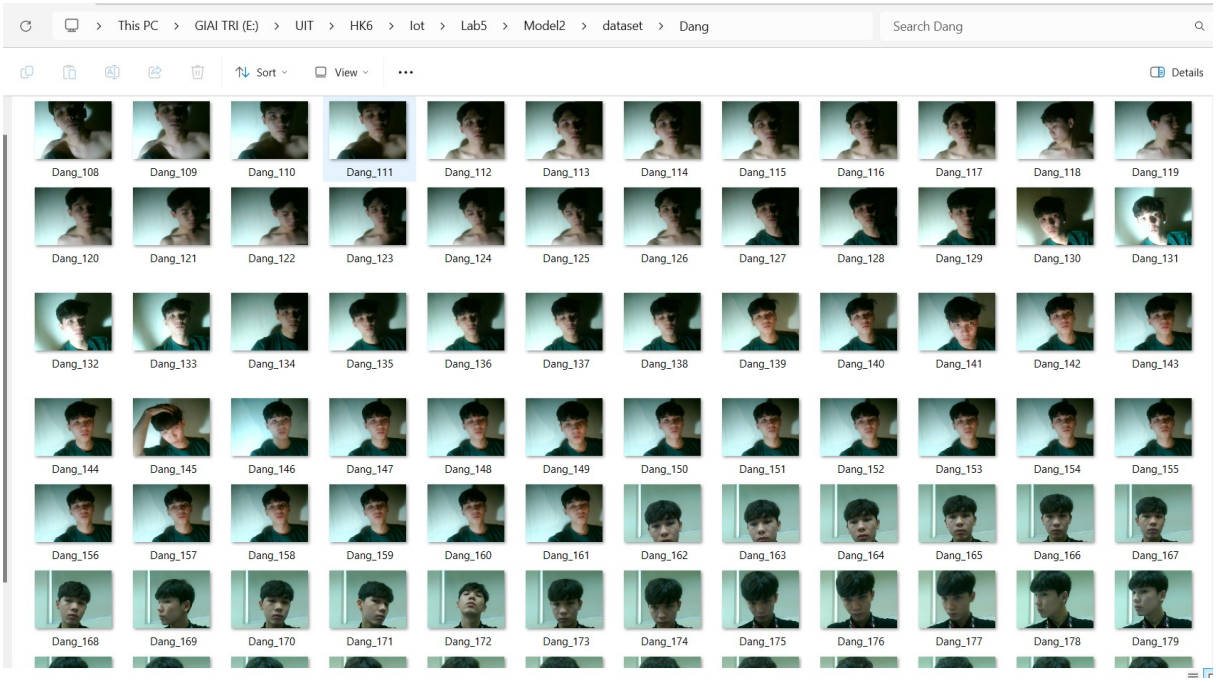
Câu hỏi 1. Thực hiện phần C, đảm bảo độ chính xác của các gương mặt đối với các thành viên trong nhóm trên 60%. Hãy nêu cách để bạn xử lý được vấn đề này. Từ đó, tìm hiểu những cách nào để tăng cường dữ liệu,... để tăng độ chính xác đối với mô hình?

1. Phương pháp tăng cường độ chính xác

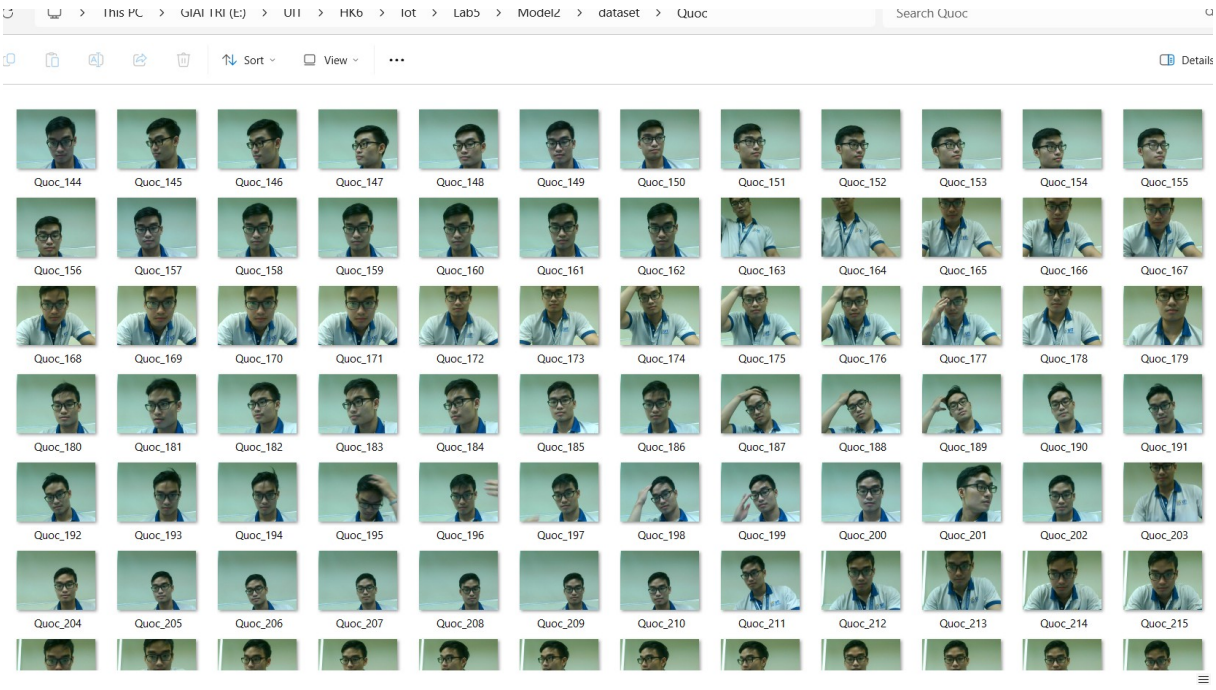
Để cải thiện độ chính xác:

1. Lựa chọn mô hình phù hợp:
- Lựa chọn kiến trúc mô hình phù hợp với bài toán và đặc điểm của dữ liệu là rất quan trọng.
- Thử nghiệm các mô hình khác nhau như Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Transformer, v.v. và so sánh hiệu suất.
- Sử dụng transfer learning bằng cách khởi tạo trọng số từ các mô hình pre-trained trên các tập dữ liệu lớn như ImageNet, COCO, v.v. Điều này giúp tăng độ chính xác với lượng dữ liệu huấn luyện ít hơn.
2. Tăng cường dữ liệu huấn luyện:
- Áp dụng kỹ thuật data augmentation để tạo ra các bản sao của dữ liệu hiện có bằng cách sử dụng các phép biến đổi ngẫu nhiên như xoay, lật, thay đổi độ sáng, v.v.
- Thu thập thêm dữ liệu từ các nguồn khác nhau như internet, các trang web chuyên ngành, v.v. Dữ liệu đa dạng sẽ giúp mô hình học tốt hơn.
- Kết hợp các kỹ thuật data augmentation và thu thập dữ liệu bổ sung sẽ giúp tăng đáng kể kích thước tập huấn luyện, từ đó cải thiện độ chính xác của mô hình.

Phương pháp lựa chọn để tăng độ nhận diện chính xác của các gương mặt đối với các thành viên trong nhóm trong bài thực hành này là **tăng dữ liệu huấn luyện model**.



Hình 1. Kho dataset (Đăng)



Hình 2. Kho dataset (Quốc)

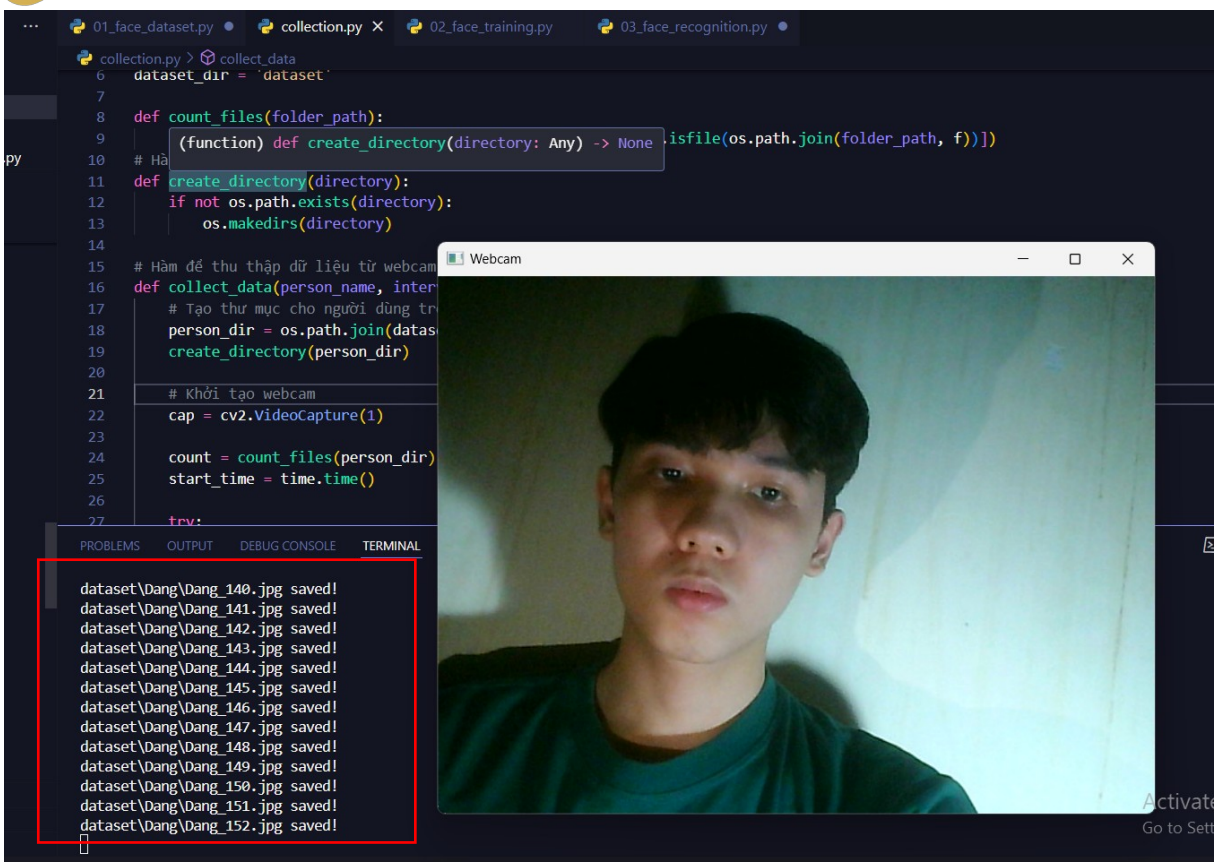
Thành viên	Số lượng ảnh
Đăng	518
Quốc	524

2. Chương trình tự động lấy dữ liệu

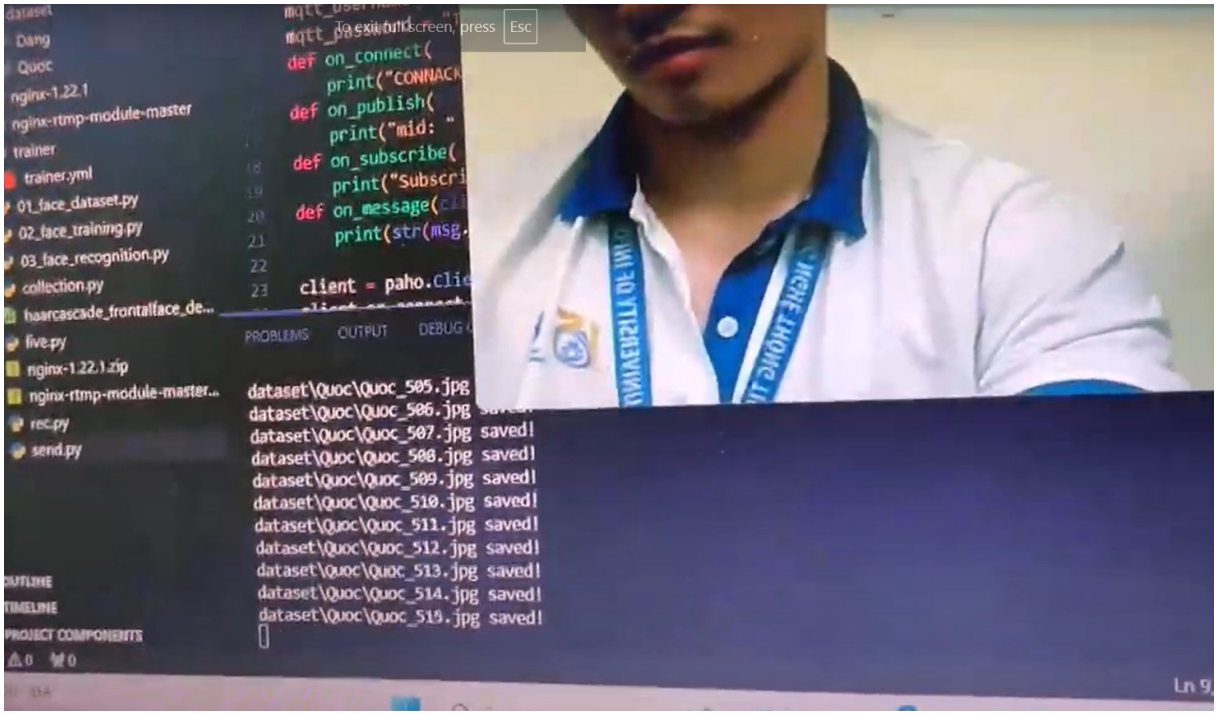
1. Minh chứng:

Demo lấy dữ liệu **Đăng**.

Demo lấy dữ liệu **Quốc**.



Hình 3. Chương trình tự động lấy dữ liệu (Đăng)



Hình 4. Chương trình tự động lấy dữ liệu (Quốc)

2. Giải thích:

```python
import cv2
import os
import time

# Thư mục gốc lưu dataset
dataset_dir = 'dataset'
```

Hình 5. Khai báo thư viện và đường dẫn parent của dữ liệu sẽ train

```python
def count_files(folder_path):
    return len([f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))])
```

Hình 6. Hàm đếm số lượng file có trong folder

Hàm **count_files** này có tác dụng đếm số lượng file có trong **<Person>** để từ đó có thể thu thập data kế tiếp thay vì ghi đè các data đã có.

```python
def create_directory(directory):
    if not os.path.exists(directory):
        os.makedirs(directory)
```

Hình 7. Hàm tạo thư mục nếu nó chưa tồn tại

```python
person_name = input('Enter the name of the person: ')
interval = int(input('Enter the interval between captures (in seconds): '))
collect_data(person_name, interval)
```

Hình 8. Nhập dữ liệu đối tượng đang lấy dữ liệu

Ở đây, **person_name** sẽ là folder của người đang thu thập dữ liệu (nơi chứa các ảnh được tự động chụp vào), **interval** là số giây trên 1 bức hình được chụp.

```python
def collect_data(person_name, interval=0.5):
    person_dir = os.path.join(dataset_dir, person_name)
    create_directory(person_dir)
    cap = cv2.VideoCapture(1)
    count = count_files(person_dir)
    start_time = time.time()
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            flipped_frame = cv2.flip(frame, 1)
            cv2.imshow('Webcam', flipped_frame)
            if time.time() - start_time >= interval:
                img_name = os.path.join(person_dir, f'{person_name}_{count}.jpg')
                cv2.imwrite(img_name, frame)
                print(f'{img_name} saved!')
                count += 1
                start_time = time.time()

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
    finally:
        cap.release()
        cv2.destroyAllWindows()
```

Hình 9. Hàm chụp tự động

Ở đây sẽ có tham số của người đang thu thập dữ liệu và khoảng thời gian chụp mỗi bức hình (được nhập vào).

Tiến hành chạy vòng lặp để chụp lần lượt các bức hình và đặt tên theo **person_name** cùng với số count (là số chỉ index của hình).

Nhấn phím ‘q’ để kết thức chương trình.

3. Chương trình huấn luyện model

Demo huấn luyện modedl **tại đây**.

```
1  import cv2
2  import numpy as np
3  from PIL import Image
4  import os
5
6  pathData = 'dataset'
```

Hình 10. Khai báo thư viện và đường dẫn tập tin huấn luyện

pathData là đường dẫn của dataset sẽ huấn luyện.

```
8  recognizer = cv2.face.LBPHFaceRecognizer_create()
9  detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml");
```

Hình 11. Bộ nhận dạng khuôn mặt của opencv

- recognizer = cv2.face.LBPHFaceRecognizer_create() là một bộ nhận dạng khuôn mặt dựa trên thuật toán LBPH.

- detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml") là một bộ phát hiện khuôn mặt dựa trên Haar Cascade.

```
11 def getImagesAndLabels(main_path):
12     faceSamples = []
13     ids = []
14     folder_index = 0
15
16     for user_folder in os.listdir(main_path):
17         user_folder_path = os.path.join(main_path, user_folder)
18         if os.path.isdir(user_folder_path):  # Check if it's a directory
19             for image_name in os.listdir(user_folder_path):
20                 image_path = os.path.join(user_folder_path, image_name)
21                 if os.path.isfile(image_path):  # Check if it's a file
22                     PIL_img = Image.open(image_path).convert('L')  # Convert it to grayscale
23                     img_numpy = np.array(PIL_img, 'uint8')
24                     id = folder_index  # Use the folder index as the ID
25                     faces = detector.detectMultiScale(img_numpy)
26                     for (x, y, w, h) in faces:
27                         faceSamples.append(img_numpy[y:y + h, x:x + w])
28                         ids.append(id)
29             folder_index += 1
30
31     return faceSamples, ids
```

Hình 12. Chương trình xử lý, huấn luyện dữ liệu

1. Khởi tạo hai danh sách faceSamples và ids để lưu trữ các mẫu khuôn mặt và nhãn tương ứng. *(Line 12 – 14)*

2. Lặp qua các thư mục con trong thư mục main_path. *(Line 16 - 29)*
   - Với mỗi thư mục con, coi đó là một người dùng và gán ID tương ứng.
   - Lặp qua các tập tin ảnh trong thư mục con.
     - Mở từng tập tin ảnh, chuyển đổi sang ảnh xám.
     - Sử dụng detector.detectMultiScale() để phát hiện các khuôn mặt trong ảnh.
     - Với mỗi khuôn mặt được phát hiện, thêm nó vào faceSamples và thêm ID tương ứng vào ids.
3. Cuối cùng, trả về faceSamples và ids. *(Line 31)*

Hàm này có thể được sử dụng để chuẩn bị dữ liệu cho việc huấn luyện một bộ nhận dạng khuôn mặt.

```
33  faces,ids = getImagesAndLabels(pathData)
34  recognizer.train(faces, np.array(ids))
35
36  recognizer.write('trainer/trainer.yml') # recognizer.save() worked on Mac, but not on Pi
37
38  print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
39
```

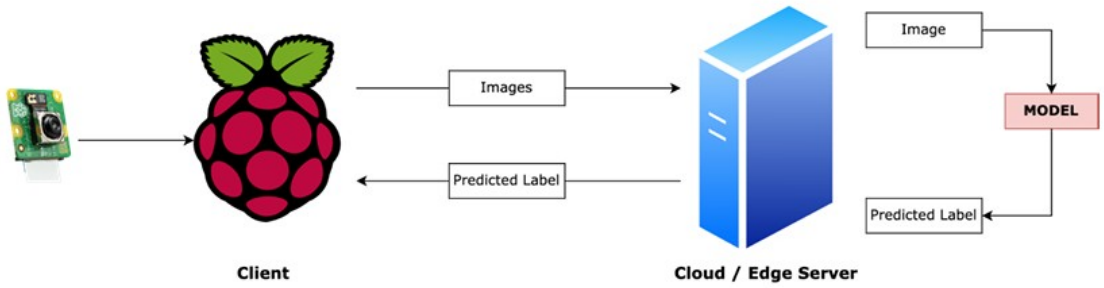
Hình 13. Lưu lại model

4. Kiểm tra độ chính xác mỗi thành viên

Demo lấy dữ liệu của **Đăng** kết quả độ tự tin nhận diện > 77%.

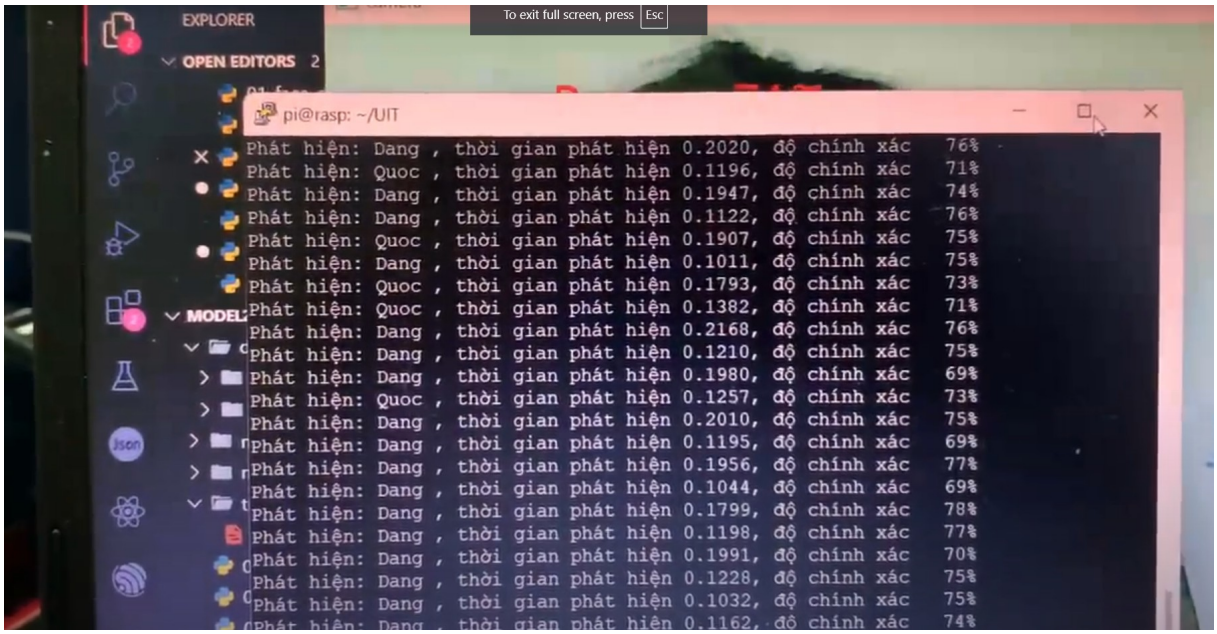
Demo lấy dữ liệu của **Quốc** kết quả độ tự tin nhận diện > 74%.

Câu hỏi 2. Tích hợp mô hình đã được huấn luyện ở câu 1 vào mô hình IoT cơ bản được thể hiện tại Hình 1. Mô hình gồm có 1 Raspberry Pi và 1 PC / laptop cá nhân. Raspberry Pi đóng vai trò như Client, được sử dụng để thu thập hình ảnh từ camera về người dùng cần nhận diện. PC / laptop cá nhân đóng vai trò như một Edge / Cloud Server dùng để xử lý hình ảnh thu được từ Raspberry Pi và nhận diện hình ảnh đó. Từ đó, trả kết quả nhận diện về cho Client và hiển thị kết quả nhận diện lên màn hình. Quá trình gửi nhận dữ liệu giữa Raspberry Pi và Edge / Cloud Server sử dụng Internet (Có thể sử dụng một trong các giao thức như: HTTP, MQTT, RPC …).

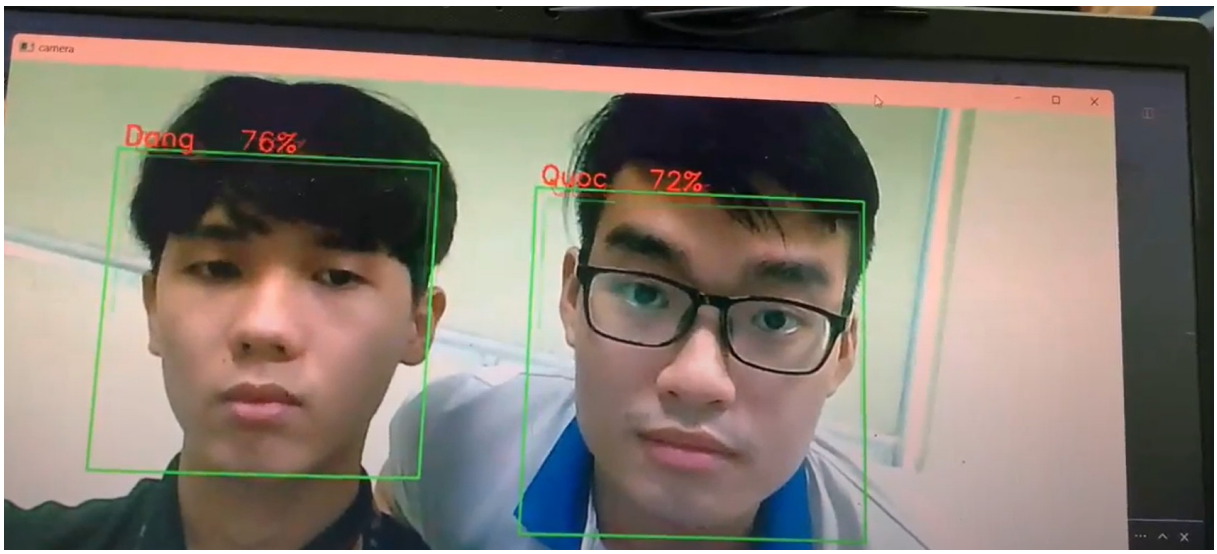


1. Minh chứng:

[Demo](#) câu 2.



Hình 14. Kết quả nhận được ở Raspberry



Hình 15. Nhận diện khuôn mặt

2. Tại client (raspberry pi)

Tiến hành tải các gói thư viện hỗ trợ python trên Raspberry.

```
pip install opencv-python
pip install zmq
```

```python
import cv2
import zmq
import base64
```

Hình 16. Khai báo thư viện

```python
context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.connect("tcp://192.168.12.119:5555")
cap = cv2.VideoCapture(1)
```

Hình 17. Thiết lập socket để stream

Ở trên thì **tcp://192.168.12.119:5555** là địa chỉ server.

```python
while True:
    ret, frame = cap.read()

    # Perform your ML processing here on 'frame'

    _, buffer = cv2.imencode('.jpg', frame)
    jpg_as_text = base64.b64encode(buffer)
    socket.send(jpg_as_text)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Hình 18. Thực hiện encode các frame thành base64 gửi cho server

3. Tại server

```python
import cv2
import numpy as np
import os
import zmq
import cv2
import base64
import paho.mqtt.client as paho
from paho import mqtt
import time
```

Hình 19. Khai báo thư viện

Sử dụng thư viện **mqtt** để gửi trả dữ liệu cho raspberry, **zmq** để tạo luồng socket 1 chiều chỉ dành cho việc nhận stream video.

```python
mqtt_broker = "269cddc81f3446da8c939add434498cf.s1.eu.hivemq.cloud"
mqtt_port = 8883
mqtt_topic = "iot/result"
mqtt_username = "hive_thanhdang"
mqtt_password = "ThanhDang123"
def on_connect(client, userdata, flags, rc, properties=None):
    print("CONNACK received with code %s." % rc)

def on_subscribe(client, userdata, mid, granted_qos, properties=None):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))

client = paho.Client(client_id="", userdata=None, protocol=paho.MQTTv311)
client.on_connect = on_connect

client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
client.username_pw_set(mqtt_username, mqtt_password)
client.connect(mqtt_broker, mqtt_port)
client.on_subscribe = on_subscribe

client.subscribe(mqtt_topic)
```

Hình 20. Thiết lập kết nối đến MQTT

Thiết lập kết nối đến các thông số MQTT chỉ định (ở đây dùng MQTT Cloud).

```python
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.bind("tcp://*:5555")
socket.setsockopt_string(zmq.SUBSCRIBE, '')
```

Hình 21. Thiết lập luồng socket zmq

Thiết lập socket của ngữ cảnh zmq để nhận stream video tốt nhất từ Raspberry. Ở đây, server sẽ lắng nghe trên port 5555.

```python
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);

font = cv2.FONT_HERSHEY_SIMPLEX

id = 0

names = [ 'Đăng', 'Quốc',"Unknown"]
```

Hình 22. Sử dụng thư việc opencv để nhận diện vị trí khuôn mặt

4. **recognizer = cv2.face.LBPHFaceRecognizer_create()**: Khởi tạo một đối tượng trình nhận dạng khuôn mặt sử dụng phương pháp Local Binary Patterns Histograms (LBPH). Đây là một trong các thuật toán nhận dạng khuôn mặt được cung cấp bởi OpenCV.
5. **recognizer.read('trainer/trainer.yml')**: Nạp mô hình đã được huấn luyện từ tệp trainer.yml trong thư mục trainer. Mô hình này chứa các thông tin về khuôn mặt đã được huấn luyện trước đó.
6. **cascadePath = "haarcascade_frontalface_default.xml":** Định nghĩa đường dẫn đến tệp phân loại Haar Cascade cho phát hiện khuôn mặt. Đây là một mô hình phát hiện khuôn mặt được cung cấp sẵn trong OpenCV.
7. faceCascade = cv2.CascadeClassifier(cascadePath): Khởi tạo một đối tượng phân loại Haar Cascade để phát hiện khuôn mặt

sử dụng mô hình được tải từ tệp haarcascade_frontalface_default.xml.

8. **font = cv2.FONT_HERSHEY_SIMPLEX**: Chọn font chữ để hiển thị văn bản trên hình ảnh.
9. **names = ['Đăng', 'Quốc', "Unknown"]**: Định nghĩa một danh sách các tên tương ứng với các ID. Trong trường hợp này, có 3 ID: 0 cho "Đăng", 1 cho "Quốc", và 2 cho "Unknown" (không xác định).

```
while True:
    jpg_as_text = socket.recv()
    start_time = time.time()
    jpg_original = base64.b64decode(jpg_as_text)
    jpg_as_np = np.frombuffer(jpg_original, dtype=np.uint8)
    img = cv2.imdecode(jpg_as_np, flags=1)
    img = cv2.flip(img, 1)

    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(0.1 * img.shape[1]), int(0.1 * img.shape[2])),
       )
    for(x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
        if (confidence < 40):
            id = names[id]
            confidence = "  {0}%".format(round(100 - confidence))

        else:
            id = "Unknown"
            confidence = "  {0}%".format(round(confidence))

        end_time = time.time()
        client.publish(mqtt_topic, payload=f"Phát hiện: {id} , thời gian phát hiện {(end_time-start_time):.4f}, độ chính xác {confidence}")
        if (id != "Unknown"):
            cv2.putText(img, str(id) +' '+ str(confidence), (x+5,y-5), font, 1, (0,0,255), 2)
        else:
            cv2.putText(img, str(id), (x+5,y-5), font, 1, (255,255,255), 2)
    cv2.imshow('camera',img)
    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27 :
        break
```

Hình 23. Xử lý

1. Nhận ảnh từ socket:
- jpg_as_text = socket.recv(): Nhận dữ liệu ảnh dạng base64 từ socket.
- jpg_original = base64.b64decode(jpg_as_text): Giải mã dữ liệu ảnh từ base64 về định dạng byte.
- jpg_as_np = np.frombuffer(jpg_original, dtype=np.uint8): Chuyển đổi dữ liệu ảnh từ byte sang numpy array.
- img = cv2.imdecode(jpg_as_np, flags=1): Giải mã numpy array thành ảnh OpenCV.
2. Xử lý ảnh:
- img = cv2.flip(img, 1): Lật ảnh theo chiều ngang (để phù hợp với camera).
- gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY): Chuyển ảnh màu sang ảnh xám.
3. Phát hiện khuôn mặt:
- faces = faceCascade.detectMultiScale(...): Sử dụng bộ phân loại Haar Cascade để phát hiện các khuôn mặt trong ảnh.

- Các tham số của detectMultiScale điều chỉnh độ nhạy và kích thước tối thiểu của khuôn mặt được phát hiện.

4. Nhận dạng danh tính:

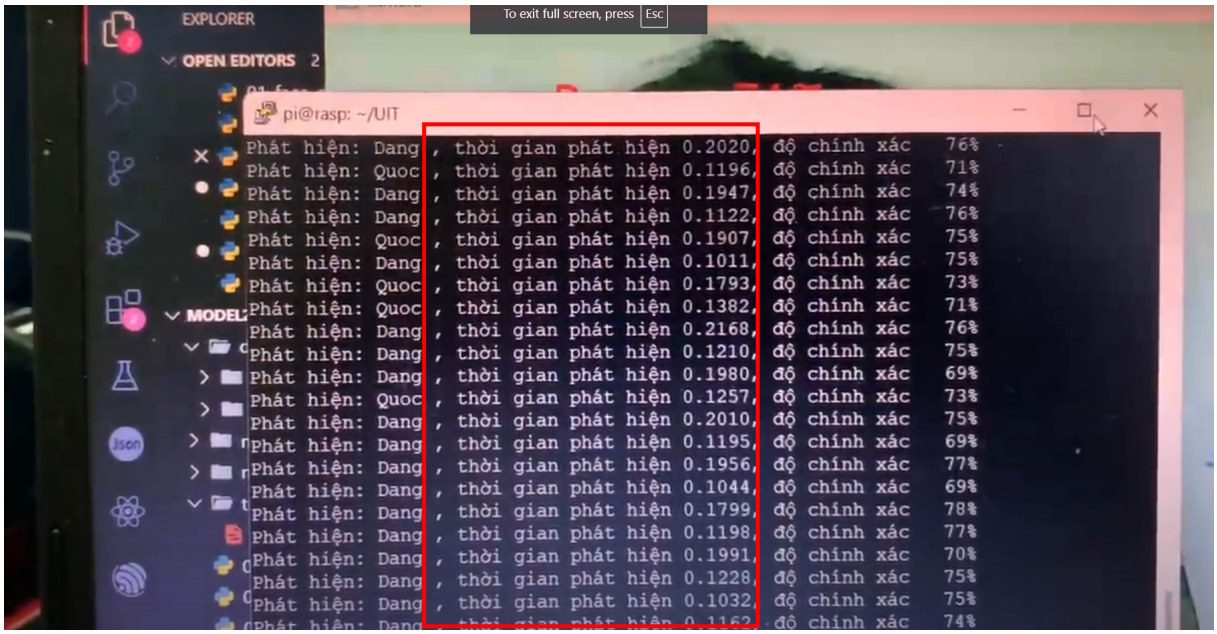
- id, confidence = recognizer.predict(gray[y:y+h,x:x+w]): Sử dụng bộ nhận dạng LBPH để xác định danh tính của người trong khuôn mặt.
- id là ID của người được nhận dạng, confidence là mức độ tin cậy.

5. Hiển thị kết quả và gửi lên MQTT:

- cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2): Vẽ một hình chữ nhật xanh lá cây quanh khuôn mặt.
- cv2.putText(img, str(id) +' '+ str(confidence), (x+5,y-5), font, 1, (0,0,255), 2): Hiển thị tên người và độ tin cậy lên ảnh.
- client.publish(mqtt_topic, payload=f"Phát hiện: {id} , thời gian phát hiện {(end_time-start_time):.4f}, độ chính xác {confidence}"): Gửi thông tin lên MQTT.

Câu hỏi 3. Viết thêm hàm để đo tổng thời gian nhận diện 1 hình ảnh. Thời gian được tính từ lúc Raspberry Pi gửi hình ảnh cho Edge / Cloud Server cho đến khi nhận được kết quả trả về.

1. Minh chứng:



2. Giải thích:

```python
while True:
    jpg_as_text = socket.recv()
    start_time = time.time()
    jpg_original = base64.b64decode(jpg_as_text)
    jpg_as_np = np.frombuffer(jpg_original, dtype=np.uint8)
    img = cv2.imdecode(jpg_as_np, flags=1)
    img = cv2.flip(img, 1)

    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(0.1 * img.shape[1]), int(0.1 * img.shape[2])),
        )
    for(x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
        if (confidence < 40):
            id = names[id]
            confidence = "  {0}%".format(round(100 - confidence))

        else:
            id = "Unknown"
            confidence = "  {0}%".format(round(confidence))

        end_time = time.time()
        client.publish(mqtt_topic, payload=f"Phát hiện: {id} , thời gian phát hiện {(end_time-start_time):.4f}, độ chính xác {confidence}")
        if (id != "Unknown"):
            cv2.putText(img, str(id) +' '+ str(confidence), (x+5,y-5), font, 1, (0,0,255), 2)
        else:
            cv2.putText(img, str(id), (x+5,y-5), font, 1, (255,255,255), 2)
    cv2.imshow('camera',img)
    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27 :
        break
```

Hình 24. Tính thời gian xử lý model

Lấy thời gian trước xử lý (**start_time**) và thời gian sau xử lý (**end_time**) và tính khoảng cách giữa 2 thời gian này, ta thu được tốc độ xử lý 1 khung hình được detect.

Câu hỏi 4: Tìm hiểu các file trong mã nguồn được cung cấp. Trình bày các quy trình, bước xử lý, mô tả về những tìm hiểu của mình về mã nguồn trên.

Demo chạy detect trên source code của thầy.

Đầu tiên chúng ta chạy lệnh:

```
pip install -r requirements.txt
```

Lệnh này giúp cài đặt cái gói cơ bản của python. Trong đó bao gồm thư viện:

1. **tensorflow**: Thư viện machine learning và deep learning mạnh mẽ, được sử dụng rộng rãi để xây dựng và triển khai các mô hình AI.
2. **keras**: Thư viện machine learning được xây dựng trên TensorFlow, cung cấp giao diện đơn giản và dễ sử dụng để phát triển mô hình neural network.
3. **scikit-learn**: Thư viện machine learning phổ biến, cung cấp các công cụ hiệu quả để thực hiện các tác vụ như phân loại, hồi quy, clustering và nhiều tác vụ khác.
4. **opencv-python**: Thư viện xử lý ảnh và video mạnh mẽ, được sử dụng rộng rãi trong các ứng dụng computer vision.
5. **h5py**: Thư viện cho phép đọc và ghi tệp dạng HDF5, thường được sử dụng để lưu trữ dữ liệu lớn.

6. **matplotlib**: Thư viện vẽ đồ thị và biểu đồ, giúp visualize các dữ liệu và kết quả trong các ứng dụng machine learning và data analysis.
7. **Pillow**: Thư viện xử lý ảnh, cung cấp các chức năng như mở, xử lý, và lưu trữ ảnh.
8. **requests**: Thư viện đơn giản và mạnh mẽ để thực hiện các yêu cầu HTTP từ Python.
9. **psutil**: Thư viện cung cấp thông tin về tiến trình và hệ thống, hữu ích trong các ứng dụng giám sát hệ thống.
10. **imageio**: Thư viện đọc và ghi nhiều định dạng ảnh và video, hữu ích trong các ứng dụng xử lý ảnh và video.
11. **flask**: Micro framework web cho Python, giúp xây dựng các ứng dụng web nhanh chóng và dễ dàng.
12. **flask_CORS**: Tiện ích bổ sung cho Flask, giúp quản lý các yêu cầu cross-origin.

```
python src/align_dataset_mtcnn.py  Dataset/raw
Dataset/processed --image_size 160 --margin 32 --random_order
--gpu_memory_fraction 0.25
```

Câu lệnh này sử dụng thư viện **MTCNN** (Multi-Task Cascaded Convolutional Networks) trong Python để tiền xử lý và căn chỉnh tập dữ liệu hình ảnh để phục vụ cho việc huấn luyện mô hình machine learning.

Cụ thể, câu lệnh này thực hiện các tác vụ sau:

- Căn chỉnh tập hình ảnh: Lấy tập hình ảnh từ thư mục "**Dataset/raw**" và căn chỉnh chúng để có kích thước **160x160** pixel, với lề xung quanh **32** pixel.
- Xáo trộn thứ tự: Xáo trộn ngẫu nhiên thứ tự của các hình ảnh sau khi căn chỉnh.
- Giới hạn bộ nhớ **GPU**: Giới hạn sử dụng bộ nhớ GPU chỉ tối đa **25**% tổng dung lượng.
- Lưu trữ kết quả: Lưu các hình ảnh đã được căn chỉnh vào thư mục "**Dataset/processed**".

```
python src/classifier.py TRAIN Dataset/processed
Models/20180402-114759.pb Models/facemodel.pkl --batch_size
1000
```

Cụ thể, câu lệnh này thực hiện các tác vụ sau:

- Chạy chế độ TRAIN: Câu lệnh chạy chế độ huấn luyện (TRAIN) để xây dựng mô hình phân loại gương mặt.
- Đường dẫn tập dữ liệu đã xử lý: Sử dụng tập dữ liệu hình ảnh đã được xử lý và lưu trữ trong thư mục "**Dataset/processed**".

- Đường dẫn mô hình pretrained: Sử dụng một mô hình pretrained được lưu trữ trong tệp "**20180402-114759.pb**" làm điểm khởi đầu cho việc huấn luyện.
- Đường dẫn lưu trữ mô hình mới: Lưu trữ mô hình mới được huấn luyện vào tệp "**facemodel.pkl**".
- Kích thước batch: Sử dụng kích thước batch là 1000 hình ảnh để thực hiện huấn luyện.

```
python src/face_rec_flask.py
```

Câu lệnh này là chạy server, ở đây server sẽ là nơi xử lý detect.

```
22  MINSIZE = 20
23  THRESHOLD = [0.6, 0.7, 0.7]
24  FACTOR = 0.709
25  IMAGE_SIZE = 182
26  INPUT_IMAGE_SIZE = 160
27  CLASSIFIER_PATH = 'Models/facemodel.pkl'
28  FACENET_MODEL_PATH = './Models/20180402-114759.pb'
29
```

Hình 25. Thiết lập các hằng số

Thiết lập các hằng số:

- MINSIZE: Kích thước tối thiểu của khuôn mặt cần phát hiện, được thiết lập là 20 pixel.
- THRESHOLD: Các giá trị ngưỡng ba giai đoạn được sử dụng trong mô hình phát hiện khuôn mặt MTCNN, được đặt là [0.6, 0.7, 0.7].
- FACTOR: Hệ số tỷ lệ được sử dụng trong phát hiện khuôn mặt đa quy mô, được đặt là 0.709.
- IMAGE_SIZE: Kích thước của hình ảnh đầu ra, được đặt là 182x182 pixel.
- INPUT_IMAGE_SIZE: Kích thước của hình ảnh đầu vào, được đặt là 160x160 pixel.
- CLASSIFIER_PATH: Đường dẫn đến mô hình phân loại tùy chỉnh đã lưu, được đặt là 'Models/facemodel.pkl'.
- FACENET_MODEL_PATH: Đường dẫn đến mô hình FaceNet được tiền huấn luyện, được đặt là './Models/20180402-114759.pb'.

```
30  # Load The Custom Classifier
31  with open(CLASSIFIER_PATH, 'rb') as file:
32      model, class_names = pickle.load(file)
33  print("Custom Classifier, Successfully loaded")
34
```

Hình 26. Phân loại mô hình

Tải mô hình phân loại tùy chỉnh:

- Đoạn code tải mô hình phân loại tùy chỉnh đã lưu và các tên lớp liên quan từ CLASSIFIER_PATH.

- Điều này cho phép sử dụng một mô hình phân loại tùy chỉnh, được huấn luyện trên một tập dữ liệu cụ thể, để nhận dạng khuôn mặt.

```
35  with tf.Graph().as_default():
36
37      # Cai dat GPU neu co
38      gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.6)
39      sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(gpu_options=gpu_options, log_device_placement=False))
40
```

Hình 27. Thiết lập mô hình

Thiết lập phiên TensorFlow:

- Đoạn code tạo một graph TensorFlow và phiên mới.
- Nó cấu hình việc sử dụng GPU, giới hạn phần trăm bộ nhớ GPU được sử dụng là 0.6.
- Điều này đảm bảo rằng GPU được sử dụng hiệu quả trong các tác vụ trích xuất đặc trưng và phân loại.

```
40
41      with sess.as_default():
42          # Load the model
43          print('Loading feature extraction model')
44          facenet.load_model(FACENET_MODEL_PATH)
```

Hình 28. Tải FacNet

Tải mô hình FaceNet:

- Đoạn code tải mô hình FaceNet được tiền huấn luyện từ FACENET_MODEL_PATH.
- FaceNet là một mô hình học sâu có thể được sử dụng để nhận dạng khuôn mặt và trích xuất đặc trưng.
- Nó trích xuất một vector đặc trưng 128 chiều (embedding) cho mỗi khuôn mặt được phát hiện.

```
46          images_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("input:0")
47          embeddings = tf.compat.v1.get_default_graph().get_tensor_by_name("embeddings:0")
48          phase_train_placeholder = tf.compat.v1.get_default_graph().get_tensor_by_name("phase_train:0")
49          embedding_size = embeddings.get_shape()[1]
50
51          pnet, rnet, onet = align.detect_face.create_mtcnn(sess, "src/align")
52
```

Hình 29. Phát hiện khuôn mặt

Tạo mô hình phát hiện khuôn mặt MTCNN:

- Đoạn code tạo mô hình phát hiện khuôn mặt MTCNN (Multi-Task Cascaded Convolutional Networks) bằng cách sử dụng hàm align.detect_face.create_mtcnn.
- MTCNN là một mô hình phát hiện khuôn mặt tiên tiến có thể định vị và căn chỉnh chính xác các khuôn mặt trong hình ảnh.
- Các biến pnet, rnet và onet đại diện cho ba giai đoạn của mô hình MTCNN.

```python
@app.route('/')
@cross_origin()
def index():
    return "OK!";

@app.route('/recog', methods=['POST'])
@cross_origin()
def upload_img_file():
    if request.method == 'POST':
        # base 64
        name="Unknown"
        f = request.form.get('image')
        print(type(f))
        w = int(request.form.get('w'))
        h = int(request.form.get('h'))

        decoded_string = base64.b64decode(f)
        frame = np.fromstring(decoded_string, dtype=np.uint8)
        #frame = frame.reshape(w,h,3)
        frame = cv2.imdecode(frame, cv2.IMREAD_ANYCOLOR)  # cv2.IMREAD_COLOR in OpenCV 3.1

        bounding_boxes, _ = align.detect_face.detect_face(frame, MINSIZE, pnet, rnet, onet, THRESHOLD, FACTOR)
```

Hình 30. Khởi tạo route và xử lý (decode) ảnh được gửi từ client

```python
        if faces_found > 0:
            det = bounding_boxes[:, 0:4]
            bb = np.zeros((faces_found, 4), dtype=np.int32)
            for i in range(faces_found):
                bb[i][0] = det[i][0]
                bb[i][1] = det[i][1]
                bb[i][2] = det[i][2]
                bb[i][3] = det[i][3]
                cropped = frame
                #cropped = frame[bb[i][1]:bb[i][3], bb[i][0]:bb[i][2], :]
                scaled = cv2.resize(cropped, (INPUT_IMAGE_SIZE, INPUT_IMAGE_SIZE),
                                    interpolation=cv2.INTER_CUBIC)
                scaled = facenet.prewhiten(scaled)
                scaled_reshape = scaled.reshape(-1, INPUT_IMAGE_SIZE, INPUT_IMAGE_SIZE, 3)
                feed_dict = {images_placeholder: scaled_reshape, phase_train_placeholder: False}
                emb_array = sess.run(embeddings, feed_dict=feed_dict)
                predictions = model.predict_proba(emb_array)
                best_class_indices = np.argmax(predictions, axis=1)
                best_class_probabilities = predictions[
                    np.arange(len(best_class_indices)), best_class_indices]
                best_name = class_names[best_class_indices[0]]
                print("Name: {}, Probability: {}".format(best_name, best_class_probabilities))

                if best_class_probabilities > 0.5:
                    name = class_names[best_class_indices[0]]
                else:
                    name = "Unknown"
```

Hình 31. Hàm đưa ra quyết định

Giải thích cách hiển thực hiện

- if faces_found > 0: - Kiểm tra xem có bao nhiêu khuôn mặt được phát hiện. Nếu có ít nhất một khuôn mặt, thì tiếp tục xử lý.
- det = bounding_boxes[:, 0:4] - Lấy ra tọa độ của các hộp giới hạn (bounding boxes) chứa các khuôn mặt. Mỗi hộp giới hạn được đại diện bởi 4 giá trị: x, y, chiều rộng, chiều cao.
- bb = np.zeros((faces_found, 4), dtype=np.int32) - Tạo ra một mảng 2D để lưu trữ các tọa độ của các hộp giới hạn. Số hàng tương ứng với số khuôn mặt được phát hiện, và mỗi hàng chứa 4 giá trị đại diện cho tọa độ x, y, chiều rộng, chiều cao của một hộp giới hạn.
- for i in range(faces_found): - Lặp qua từng khuôn mặt được phát hiện.

- bb[i][0] = det[i][0] đến bb[i][3] = det[i][3] - Gán các tọa độ của hộp giới hạn vào mảng bb.
- cropped = frame - Lấy toàn bộ khung hình làm ảnh cắt. Đây chỉ là một khai báo, chưa thực sự cắt ảnh.
- scaled = cv2.resize(cropped, (INPUT_IMAGE_SIZE, INPUT_IMAGE_SIZE), interpolation=cv2.INTER_CUBIC) - Cắt và thay đổi kích thước của khuôn mặt được phát hiện để phù hợp với kích thước đầu vào của mô hình neural network.
- scaled = facenet.prewhiten(scaled) - Tiền xử lý ảnh bằng cách chuẩn hóa các giá trị pixel.
- scaled_reshape = scaled.reshape(-1, INPUT_IMAGE_SIZE, INPUT_IMAGE_SIZE, 3) - Định dạng lại ảnh để phù hợp với đầu vào của mô hình neural network.
- feed_dict = {images_placeholder: scaled_reshape, phase_train_placeholder: False} - Tạo ra một từ điển chứa các giá trị đầu vào cho mô hình neural network.
- emb_array = sess.run(embeddings, feed_dict=feed_dict) - Chạy mô hình neural network để trích xuất các đặc trưng (embeddings) của khuôn mặt.
- predictions = model.predict_proba(emb_array) - Sử dụng các đặc trưng đó để dự đoán xác suất thuộc về các lớp (class) trong tập dữ liệu huấn luyện.
- best_class_indices = np.argmax(predictions, axis=1) - Tìm ra lớp có xác suất dự đoán cao nhất.
- best_class_probabilities = predictions[np.arange(len(best_class_indices)), best_class_indices] - Lấy ra xác suất dự đoán cao nhất.
- best_name = class_names[best_class_indices[0]] - Lấy tên tương ứng với lớp có xác suất dự đoán cao nhất.
- if best_class_probabilities > 0.5: name = class_names[best_class_indices[0]] - Nếu xác suất dự đoán cao hơn 50%, gán tên dự đoán vào biến name. Nếu không, gán "Unknown".

```python
#     return encoded_image

def encode_image_to_base64(image_frame):
    encoded_image = base64.b64encode(image_frame).decode('utf-8')
    return encoded_image

url = 'http://127.0.0.1:8000/recog'
video = cv2.VideoCapture(1)

while True:
    ret, frame = video.read()
    # encoded_image_data = encode_image_to_base64(frame)
    # encoded_image_data = base64.b64encode()
    img_encode = cv2.imencode('.png', frame)[1]
    encoded_image_data = base64.b64encode(img_encode)
    data = {
        'image': encoded_image_data,
        'w': 100,  #  width
        'h': 100   #  height
    }
    response = requests.post(url, data=data)
    print("Response from server:")
    print(response.text)
    cv2.imshow('Frame', frame)
    cv2.waitKey(1000)
    if 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Hình 32. Phía client

Bên phía client là đơn giản encode từng khung ảnh thành dạng base64 để gửi cho server (qua url được định nghĩa ở line 16 bằng phương thức POST).