**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

---------------o0o---------------



**ĐỒ ÁN MÔN HỌC**

**Lập trình ứng dụng nhận diện khuôn mặt để điểm danh học sinh qua websever và CSDL**

**GVHD: Nguyễn Khánh Lợi**

**SVTH: Đặng Lâm Tùng**

**MSSV: 1713856**

**TP. HỒ CHÍ MINH, THÁNG NĂM 20**

***LỜI CẢM ƠN***

*Tp. Hồ Chí Minh, ngày tháng năm .*

**Sinh viên**

**TÓM TẮT ĐỒ ÁN**

Đồ án này trình bày về việc thực hiện ứng dụng điểm danh học sinh ứng dụng nhận diện gương mặt (Face Recognition) bằng phương pháp Facenet trên Tensorflow, đồng thời xây dựng hệ thống server trên nền tảng Flask Python và cơ sở dữ liệu MySQL để điểm danh và quản lý việc điểm danh học sinh theo ngày.

Mục tiêu của đồ án là xây dựng được ứng dụng điểm danh học sinh trên PC có nhiệm vụ thực hiện thu ảnh và hiện thị kết quả nhận diện, với một giao diện người dùng (GUI) thân thiện, dễ sử dụng và một server thực hiện các bước xử lý nhận diện gương mặt, thêm dữ liệu gương mặt mới, điểm danh học sinh.

**MỤC LỤC**

[1. GIỚI THIỆU 1](#_Toc483413745)

[1.1 Tổng quan 1](#_Toc483413746)

[1.2 Nhiệm vụ đề tài 1](#_Toc483413747)

[2. LÝ THUYẾT 6](#_Toc483413749)

3. [THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM 2](#_Toc483413751)1

4[. KẾT QUẢ THỰC HIỆN 2](#_Toc483413752)1

5[. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 32](#_Toc483413753)

5[.1 Kết luận 32](#_Toc483413754)

5[.2 Hướng phát triển 32](#_Toc483413755)

[7. TÀI LIỆU THAM KHẢO 33](#_Toc483413756)

[8. PHỤ LỤC 33](#_Toc483413757)

DANH SÁCH HÌNH MINH HỌA

[Hình 5‑1 Kết quả thi công 3](#_Toc310380287)

[Hình 5‑2 Kết quả mô phỏng 3](#_Toc310380288)

# GIỚI THIỆU

## Tổng quan

## Bài toán face recognition

Bài toán Face Recogntion là một bài toán kinh điển của lĩnh vực Computer vision, trong đó từ ảnh gương mặt người mà ta biết được ai là ai. Trước khi deep learning phát triển thì face recognition không đủ chính xác để sử dụng trong thực tế.

## Nhiệm vụ đề tài

Nội dung 1: Tìm hiểu lý thuyết về nhận diện gương mặt

Nghiên cứu lý thuyết về việc nhận diện gương mặt (face recognition)

Thực hiện chạy demo code face recognition trên máy tính

Nội dung 2: Tìm hiểu thư viện Flask Python, cách tạo web service, các giao thức truyền nhận ảnh.

Nghiên cứu lý thuyết về thư viện Flask Python để tạo web service

Nghiên cứu lý thuyết về giao thức HTTP

Thực hiện tạo web service face recognition bằng Flask, đã thử nghiệm chạy trên local host và AWS.

Nội dung 3: Nghiên cứu hệ quản trị CSDL MySQL để quản lý việc điểm danh học sinh và xây dựng ứng dụng giao diện pyQT để thực hiện nhận diện gương mặt và điểm danh trên PC

Nghiên cứu hệ cơ sở dữ liệu MySQL

Kết nối MySQL với server xử lý face recognition.

Nghiên cứu xây dựng GUI bằng PyQT5 để thực hiện nhận diện gương mặt trên PC.

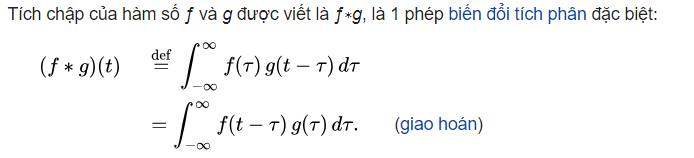
# LÝ THUYẾT

## Chương 1: Lý thuyết về nhận diện gương mặt (face recognition):

1.1 Mạng Neuron tích chập (CNN):

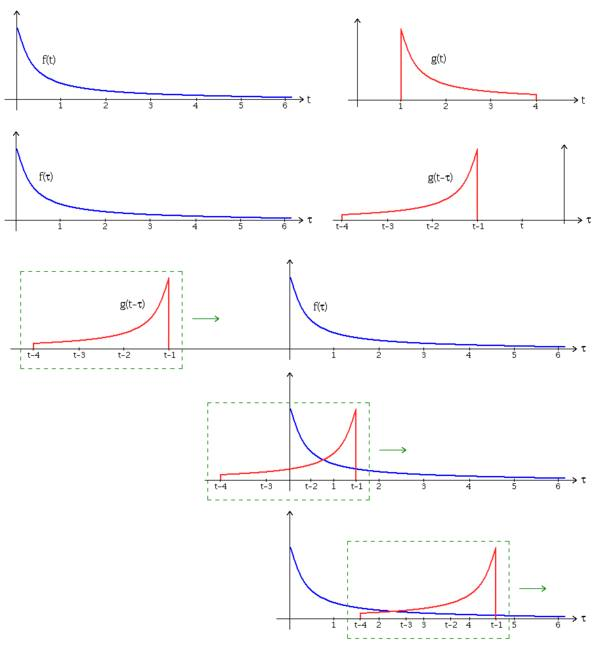
1.1.1 Tích chập (Convolution)

Trong xử lý số tín hiệu , tích chập là một khái niệm hết sức quen thuộc, nó là một phép toán cho 2 hàm số để ra một hàm số khác phép tích chập có thể thể hiện nhiều kiểu biến đổi (có thể là lọc tín hiệu, xét xem tín hiệu có theo mẫu nào không,....)



Công thức của tích chập thể hiện tính chất của tích chập đó là khi thực hiện, ta phải trừ ngược lại đáp ứng xung của hệ thống để tích chập thực hiện trước khi tín hiệu xuất hiện (thể hiện việc tín hiệu có thể xuất hiện ở trước điểm 0)

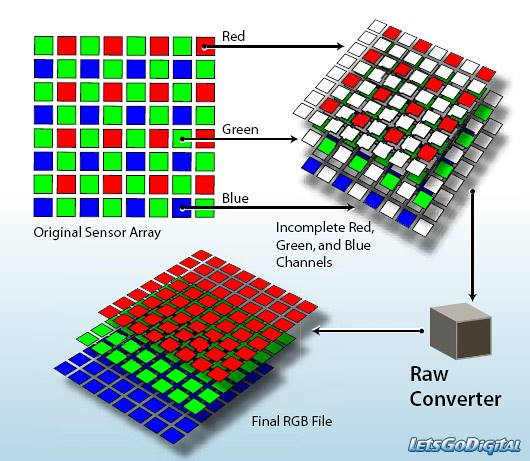
Dưới đây là hình minh họa của Wikipedia về tích chập:



Trong đó ta thực hiện trừ ngược và lật hàm g lại sau đó tích phân trên cả miền của tín hiệu

1.1.2 Tích chập 2 chiều

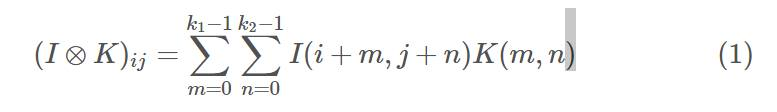
Một ảnh là một tập các ma trận các kênh màu:



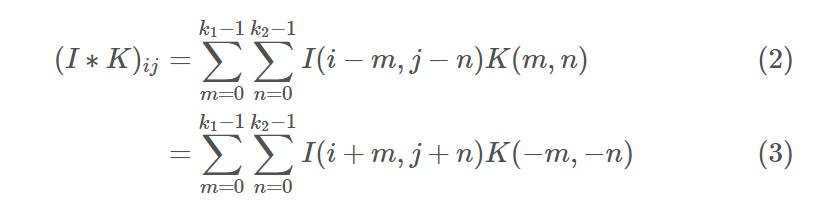
Ta hoàn toàn có thể coi đây như là tín hiệu 2 chiều vậy, chính vì vậy để lọc các tính chất của ảnh ta có thể áp dụng được các công cụ như tích chập. Chính vì thế dẫn đến các bộ lọc được dùng rất nhiều trong computer vision. Tuy nhiên ở đây tín hiệu có 2 chiều, và hơn nữa lại không có trường hợp tín hiệu xuất hiện trước t = 0 nên tích chập ở đây không cần phải lật kernel như tín hiệu 1 chiều. Chính vì vậy nó giống một khái niệm khác là tương quan chéo, mặc dù bản chất không giống nhau, phép này vẫn là tích chập.



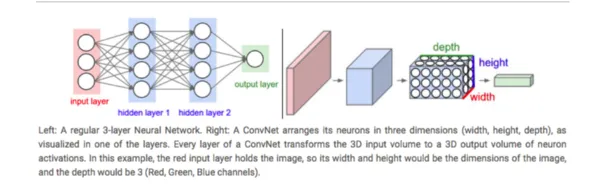
Bộ lọc Gaussian là một tích chập

Công thức tương quan chéo 2 chiều:

Công thức của tích chập 2 chiều:



Ở đây có thể thấy phép convolution được thực hiện giống với 1 chiều, và phép tích phân được thay thế bằng phép tổng, phù hợp với tín hiệu rời rạc

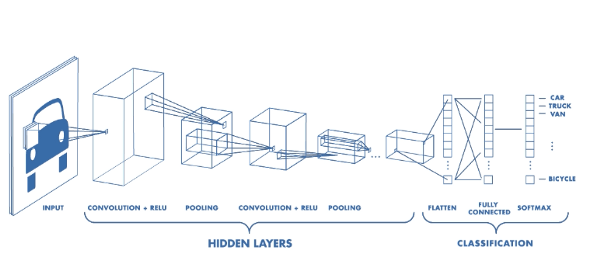
1.1.3 Mạng neuron tích chập [3]

Mạng Nơ-ron Tích Chập có kiến trúc khác với Mạng Nơ-ron thông thường. Mạng Nơ-ron bình thường chuyển đổi đầu vào thông qua hàng loạt các tầng ẩn. Mỗi tầng là một tập các nơ-ron và các tầng được liên kết đầy đủ với các nơ-ron ở tầng trước đó. Và ở tầng cuối cùng sẽ là tầng kết quả đại diện cho dự đoán của mạng.

Đầu tiên, mạng Nơ-ron Tích Chập được chia thành 3 chiều: rộng, cao, và sâu. Kế đên, các nơ ron trong mạng không liên kết hoàn toàn với toàn bộ nơ-ron kế đến nhưng chỉ liên kết tới một vùng nhỏ. Cuối cùng, một tầng đầu ra được tối giản thành véc-tơ của giá trị xác suất.

CNNs gồm hai thành phần:

* Phần tầng ẩn hay phần rút trích đặc trưng: trong phần này, mạng sẽ tiến hành tính toán hàng loạt phép tích chập và phép hợp nhất (pooling) để phát hiện các đặc trưng. Ví dụ: nếu ta có hình ảnh con ngựa vằn, thì trong phần này mạng sẽ nhận diện các sọc vằn, hai tai, và bốn chân của nó.
* Phần phân lớp: tại phần này, một lớp với các liên kết đầy đủ sẽ đóng vai trò như một bộ phân lớp các đặc trưng đã rút trích được trước đó. Tầng này sẽ đưa ra xác suất của một đối tượng trong hình.



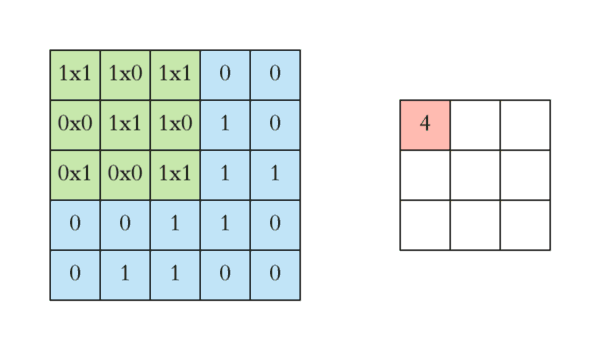
##### Rút trích đặc trưng

Tích chập là một khối quan trọng trong CNN. Thuật ngữ tích chập được dựa trên một phép hợp nhất toán học của hai hàm tạo thành hàm thứ ba. Phép toán này kết hợp hai tập thông tin khác nhau.

Trong trường hợp CNN, tích chập được thực hiện trên giá trị đầu vào của dữ liệu và kernel/filter (thuật ngữ này được sử dụng khác nhau tùy tình huống) để tạo ra một bản đồ đặc trưng (feature map).

Ta thực hiện phép tích chập bằng cách trượt kernel/filter theo dữ liệu đầu vào. Tại mỗi vị trí, ta tiến hành phép nhân ma trận và tính tổng các giá trị để đưa vào bản đồ đặc trưng.

Trong hình dưới đây, thành phần kernel/filter (màu xanh lá) trượt trên đầu vào (màu xanh dương) và kết quả được trả về bản đồ đặc trưng (màu đỏ). Kernel/filter có kích thước là 3×3 trong ví dụ này.



Trong thực tế, tích chập được thực hiện hiện trên không gian 3 chiều. Vì mỗi hình ảnh được biểu diễn dưới dạng 3 chiều: rộng, cao, và sâu. Chiều sâu ở đây chính là giá trị màu sắc của hình (RGB).

Ta thực hiện phép tích chập trên đầu vào nhiều lần khác nhau. Mỗi lần sử dụng một kernel/filter khác nhau. Kết quả ta sẽ thu được những bản đồ đặc trưng khác nhau. Cuối cùng, ta kết hợp toàn bộ bản đồ đặc trưng này thành kết quả cuối cùng của tầng tích chập.

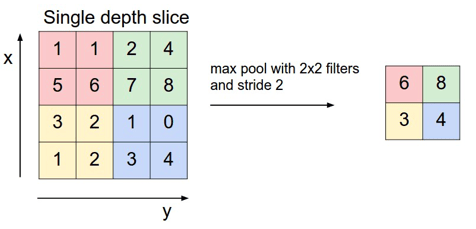
Tương tự như mạng nơ-ron thông thường, ta sử dụng một hàm kích hoạt (activate function) để có đầu ra dưới dạng phi tuyến. Trong trường hợp CNN, đầu ra của phép tích chập sẽ đi qua hàm kích hoạt nào đó ví dụ như hàm ReLU (rectified linear units).

Trong quá trình trượt kernel/filter trên dữ liệu đầu vào, ta sẽ quy định một bước nhảy (stride) với mỗi lần di chuyển. Thông thường ta lựa chọn thường chọn bước nhảy là 1. Nếu kích thước bước nhảy tăng, kernel/filter sẽ có ít ô trùng lắp.

Bởi vì kích thước đầu ra luôn nhỏ hơn đầu vào nên ta cần một phép xử lí đầu vào để đầu ra không bị co giãn. Đơn giản ta chỉ cần thêm một lề nhỏ vào đầu vào. Một lề với giá trị 0 sẽ được thêm vào xung quanh đầu vào trước khi thực hiện phép tích chập.

Thông thường, sau mỗi tầng tích chập, ta sẽ cho kết quả đi qua một tầng hợp nhất (pooling layer). Mục đích của tầng này là để nhanh chóng giảm số chiều. Việc này giúp giảm thời gian học và hạn chế việc overfitting.

Một phép hợp nhất đơn giản thường được dùng đó là max pooling, phép này lấy giá trị lớn nhất của một vùng để đại diện cho vùng đó. Kích thước của vùng sẽ được xác định trước để giảm kích thước của bản đồ đặc trưng nhanh chóng nhưng vẫn giữ được thông tin cần thiết.



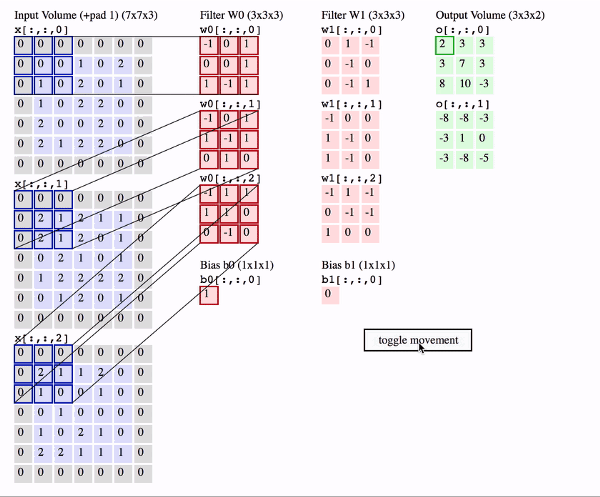
Tổng kết lại khi sử dụng CNN, ta cần chú ý đến 4 tham số quan trọng:

* Kích thước kernel/filter
* Số lượng kernel/filter
* Kích thước bước nhảy (stride)
* Kích thước lề (padding)

##### Phân lớp

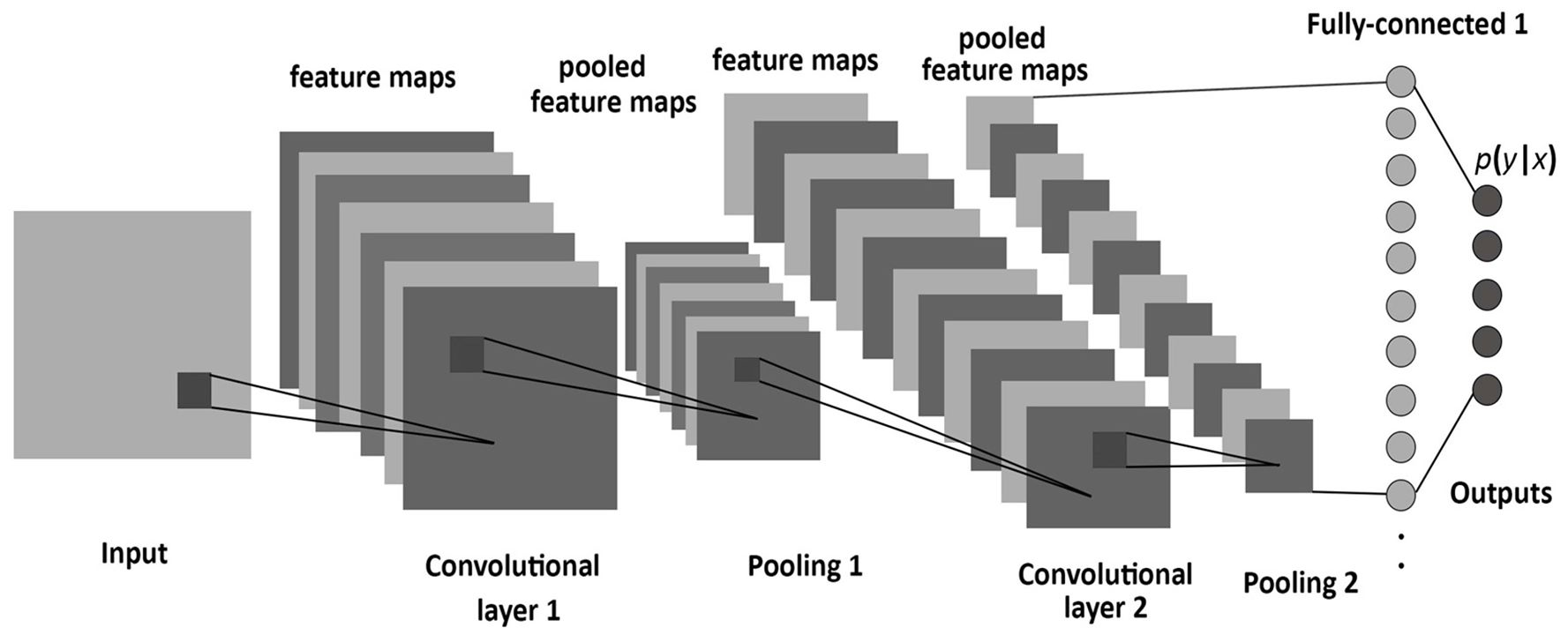
Trong phần phân lớp, ta sử dụng một vài tầng với kết nối đầy đủ để xử lí kết quả của phần tích chập. Vì đầu vào của mạng liên kết đầy đủ là 1 chiều, ta cần làm phẳng đầu vào trước khi phân lớp. Tầng cuối cùng trong mạng CNN là một tầng liên kết đầy đủ, phần này hoạt động tương tự như mạng nơ-ron thông thường.

Kết quả thu được cuối cùng cũng sẽ là một véc-tơ với các giá trị xác suất cho việc dự đoán như mạng nơ-ron thông thường.



1.1.4 Face Recognition với mạng CNN

Để hiểu cho đơn giản CNN hay Mạng neuron tích chập gồm các lớp tích chập sẽ thực hiện các thao tác tách feature của một hình ảnh ra và sau đó sử dụng một mô hình máy học khác như kNN hoặc SVM để phân biệt người này với người khác. Cách thực hiện này là one-shot learning, CNN chỉ đóng vai trò extract feature, việc huấn luyện CNN ở đây là khiến nó tách đặc trưng tốt hơn. Ta cũng có thể thực hiện huấn luyện một CNN với số đầu ra là số người, nhưng điều đó rất khó khăn, đòi hỏi thay đổi cấu trúc mạng và việc thu thập dataset cũng mất thời gian hơn, còn có thể mất cân bằng bộ dataset.[4]



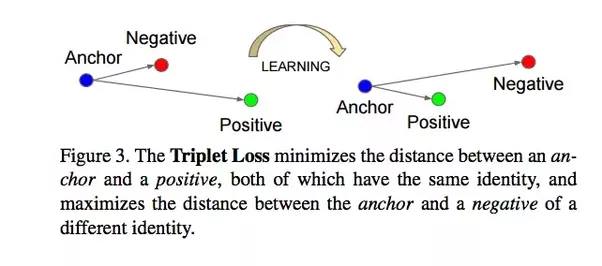
Facenet một mạng CNN có nhiệm vụ tách các đặc trưng của một ảnh mặt. Điểm đặc biệt tạo nên sự khác biệt của Facenet là nó sử dụng hàm lỗi Triplet để tối thiểu hóa khoảng cách giữa các gương mặt tương đồng và tối đa hóa khoảng cách đến những gương mặt không tương đồng, vì vậy facenet có thể phân biệt rất chính xác người với người. Quá trình train CNN được thực hiện trên các bộ dataset lớn (ví dụ như vggface hoặc MS 1 million), kết quả ta muốn là tập embedding vector ra phải tách nhau ra thành từng cụm để thực hiện phân loại bằng kNN hoặc SVM.

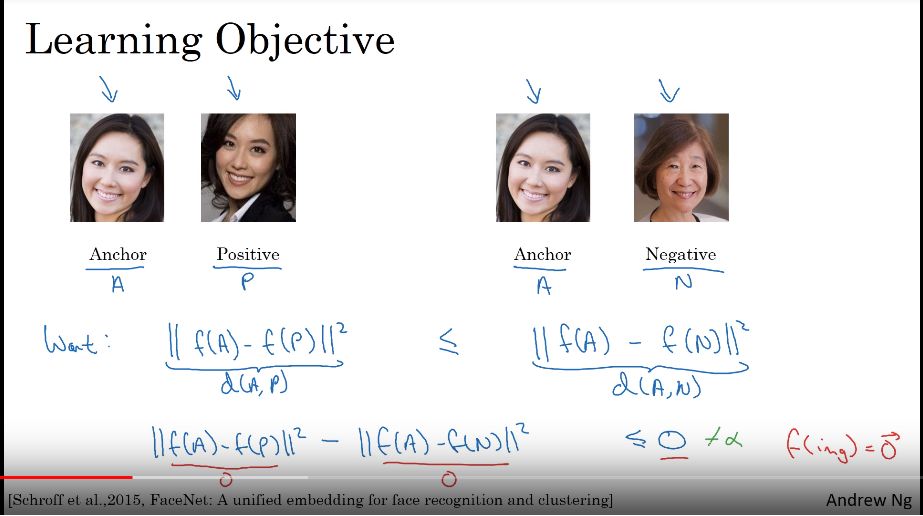
B1: Sử dung một bộ face detector để tách phần ảnh có gương mặt người ra. Các phương pháp hiện đại sử dụng MTCNN (Multi-task Cascaded Convolutional Networks), ưu điểm là mô hình này detect được gương mặt ở nhiều góc độ, và có thể nhận diện cả face landmark với thời gian xử lý khá tốt. Sau đó ta resize hình lại một kích cỡ xác định (tùy chọn), facenet thì là 160x160.

B2: Thực hiện các bước “lọc” bộ dataset. Thực tế là các bộ dataset lớn như vggface hay ms 1m đều có nhiều “nhiễu”, tức các hình không chứa gương mặt hoặc gương mặt khác với người cần xét. Vì vậy có thể dùng một mô hình nhận diện gương mặt khác để tìm và loại bỏ những hình “nhiễu” ấy. Hoặc đơn giản là ta có thể lấy clean list (danh sách các hình chuẩn) đã được lọc sẵn.

B3: Thực hiện bước normalize cho tập dataset đã chuẩn bị. Cách normalize cũng giống như cách normalize cho CNN bình thường. Các bước train: Đầu vào là tập dataset đã chuẩn bị, CNN có thể là các CNN như inception, mobilenet,… với đầu ra là embbeding vector như đã nói, embedding vector này có thể có 128 chiều hoặc 512 chiều,… tùy ý. Tóm lại ta chỉ cần lấy cấu trúc inception hoặc mobilenet, bỏ lớp cuối và thêm vào lớp embedding vector (không có activation)

B4: Forward pass trên tập data, lưu các embedding vector đã tính trên minibatch lại.  
  
B5: Chia minibatch thành 3 phần là positive, negative và anchor, trong đó anchor là phần “neo” giống như “tâm” của cụm embedding vector, sau đó chọn phần positive là những embedding vector là cùng người với anchor, negative là khác người  
  
B6: Tính Triplet loss là khoảng cách giữa các embbeding vector anchor và positive trừ cho khoảng cách giữa các embbeding vector anchor và negative (tức ta tìm cách làm giảm khoẳng cách từ các vector positive tới anchor và ngược lại đẩy các vector negative đi xa khỏi anchor)

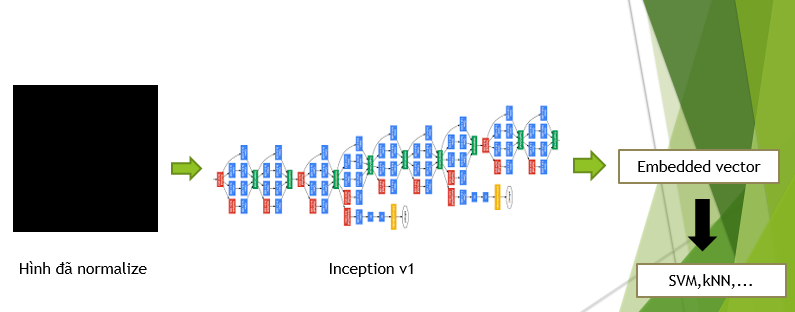




Thực hiện trong thực tế:

1. Sử dụng một model face detector nào đó để tìm bounding box của gương mặt (các mô hình hiện đại sử dụng MTCNN, còn OpenFace dùng Dlib detector) chú ý là face detector phải giống với face detector được dùng cho quá trình train mới cho kết quả chính xác
2. Cắt phần gương mặt đã tìm được ra, resize lại và thực hiện prewhiten (bước normalize data)
3. Đưa vào CNN đã train để tính ra được embbeding vector.
4. Dùng kNN, SVM, hoặc so sánh khoảng cách để tìm “cụm” mà embedding vector đó thuộc về, từ đó suy ra danh tính người được chọn





## Chương 2: Giao thức HTTP và thư viện Flask Python

### Giao thức HTTP:

Http (HyperText Transfer Protocol) là giao thức truyền tải siêu văn bản được sử dụng trong www dùng để truyền tải dữ liệu giữa Web server đến các trình duyệt Web và ngược lại. [5]

HTTP sử dụng các request như GET, POST, HEAD, PUT,.. . để thực hiện truyền thông tin

### Phương thức POST

Phương thức POST truyền thông tin thông qua HTTP header, thông tin này được mã hóa như phương thức GET. Dữ liệu đươc gửi bởi phương thức POST rất bảo mật vì dữ liệu được gửi ngầm, không đưa lên URL, bằng việc sử dụng Secure HTTP, bạn có thể chắc chắn rằng thông tin của mình là an toàn. Parameters được truyền trong request body nên có thể truyền dữ liệu lớn, hạn chế tùy thuộc vào cấu hình của Server. Không cache và bookmark được cũng như không được lưu lại trong browser history. POST không có bất kì hạn chế nào về kích thước dữ liệu sẽ gửi, có thể gửi dữ liệu nhị phân, hình ảnh.[6]

Để thực hiện truyền ảnh lên server để thực hiện nhận diện gương mặt em sử dụng giao thức POST do POST có thể send file ảnh lên server và để xử lý POST request lên server em sử dụng thư viện Flask của Python.

### Phương thức Get

Phương thức GET gửi thông tin người dùng đã được mã hóa thêm vào trên yêu cầu trang:

http://www.example.com/index.htm?name=value1&name1=value1

Chúng ta thấy rằng GET lộ thông tin trên đường dẫn URL. Băng thông của nó chỉ khoảng 1024 kí tự vì vây GET hạn chế về số kí tự được gửi đi. GET không thể gửi dữ liệu nhị phân , hình ảnh ... Có thể cached và được bookmark (đánh dấu trên trình duyệt). Lưu trong browser history.

### Thư viện Flask

Flask là một web frameworks, nó thuộc loại micro-framework được xây dựng bằng ngôn ngữ lập trình Python.



Đặc điểm của thư viện Flask là đơn giản, rất dễ hiểu và dễ sử dụng.

Thời gian phát triển ứng dụng Flask rất nhanh

Ví dụ về một chương trình Flask đơn giản:

from flask import Flask # Gọi thư viện Flask

app = Flask(\_\_name\_\_) # Tạo đối tượng Flask để tạo web service

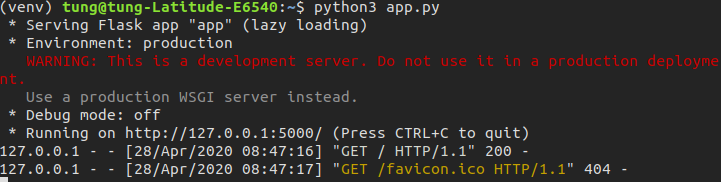
@app.route('/') # Quy định đường html là localhost/ sẽ do hàm hello\_world quản lý

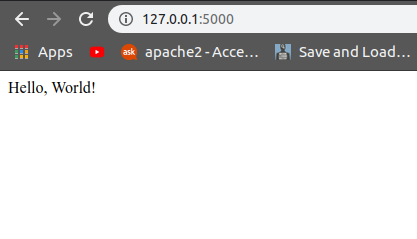
def hello\_world(): # Ta có thể chọn đường dẫn cũng như giao thức (POST hay GET)

return 'Hello, World!' # Trả lời lại request của user

if \_\_name\_\_ == '\_\_main\_\_': # Hàm main để thực chạy app

app.run()





## Chương 3: Hệ quản trị dữ liệu MySQL:

### Hệ quản trị dữ liệu MySQL và ngôn ngữ SQL

+ SQL là viết tắt của Structured Query Language là ngôn ngữ truy vấn có cấu trúc. Nó là một ngôn ngữ, là tập hợp các lệnh để tương tác với cơ sở dữ liệu. Dùng để lưu trữ, thao tác và truy xuất dữ liệu được lưu trữ trong một cơ sở dữ liệu quan hệ. [6]

+ SQL giúp giúp quản lý hiệu quả và truy vấn thông tin nhanh hơn, giúp bảo trì, bảo mật thông tin dễ dàng hơn.

+ SQL quản lý data thành từng bảng (table)

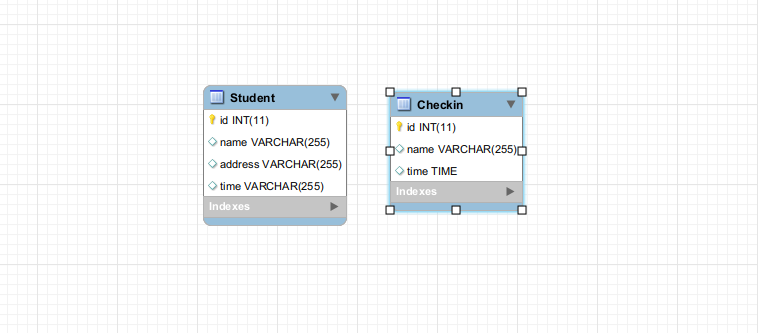
Data trong table có thể liên kết với nhau.

Ví dụ tạo Table SQL:

* CREATE TABLE if not exists Student (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255), time VARCHAR(255))

Để chọn một dữ liệu ta dùng câu lệnh SELECT:

* SELECT \* FROM Checkin JOIN Student USING(name)



Để thêm data vào TABLE:

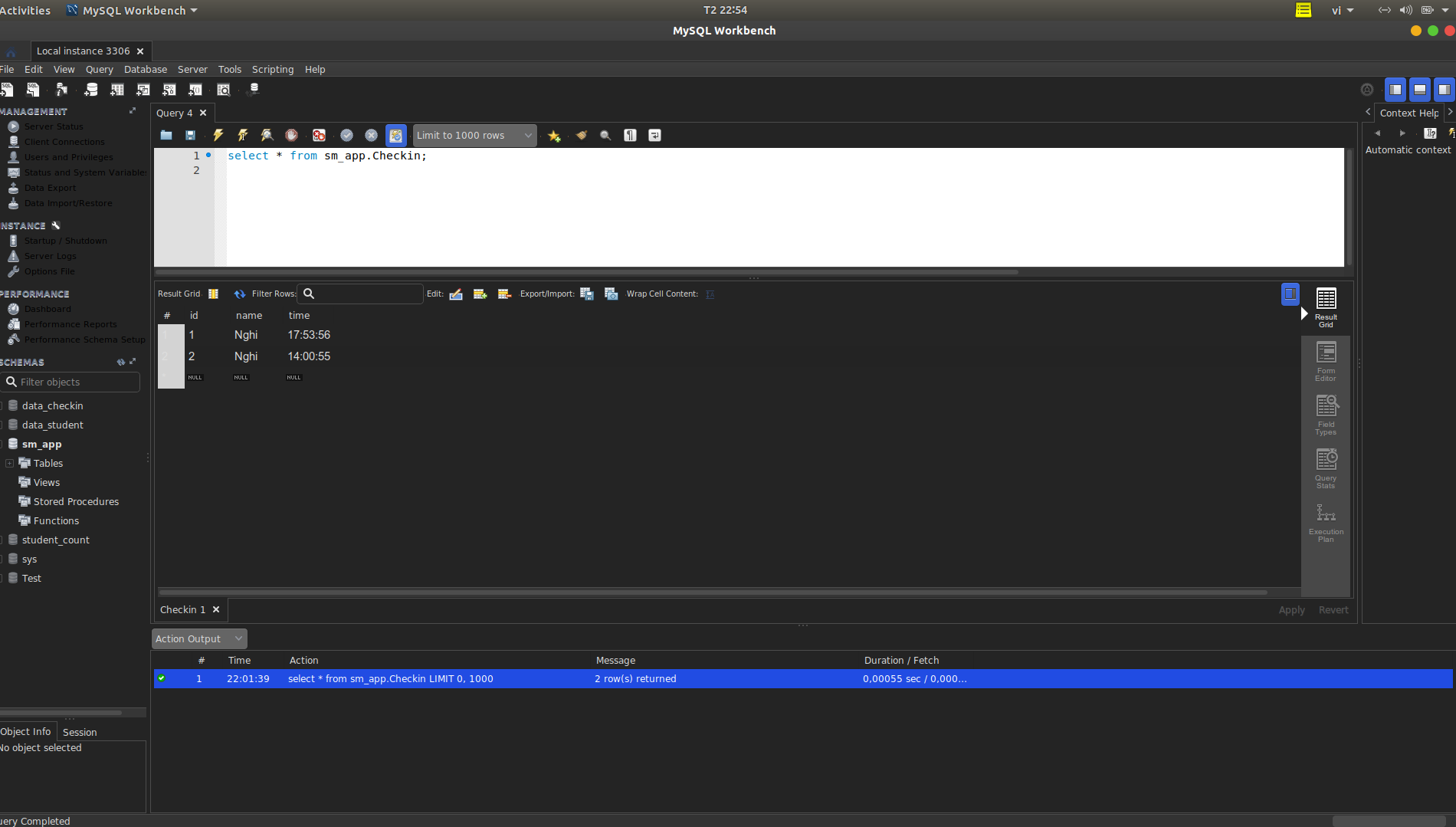
* INSERT INTO Student(name, address) VALUES (%s,%s)

MySQL là một hệ thống quản trị cơ sở dữ liệu mã nguồn mở (Relational Database Management System, viết tắt là RDBMS)

RDBMS là một phần mềm hay dịch vụ dùng để tạo và quản lý các cơ sở dữ liệu (Database) theo hình thức quản lý các mối liên hệ giữa chúng.

Điểm hiệu qủa của MySQL là nó có chức năng phân quyền và cách kết nối dễ dàng.

Giao diện MySQL Workbench, phía trên là câu lệnh Query dữ liệu từ database sm\_app, bảng Checkin.

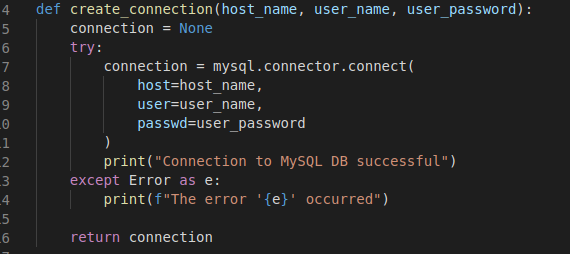


Để kết nối Python và MySQL, ta sử dụng thư viện mysql của Python:

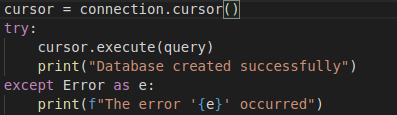
Thư viện này cung cấp phương pháp thực hiện Query data từ MySQL với các câu lệnh SQL

Để truy cập database, ta cần tạo một kết nối tới MySQL, sau đó tạo đối tượng con trỏ (Cursor) để thực hiện các câu lệnh Query data

Hàm tạo connection, hàm này trả về đối tượng mysql.connector có nhiệm vụ tạo kết nối giữa python và My SQL



Tạo đối tượng cursor và thực thi câu lệnh query



Bằng cách đưa các câu lệnh SQL dưới dạng String vào method execute của đối tượng cursor, ta có thể thực hiện truy cập CSDL MySQL

# THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM

3.1 Yêu cầu đặt ra cho phần mềm

Các chức năng cần thực hiện:

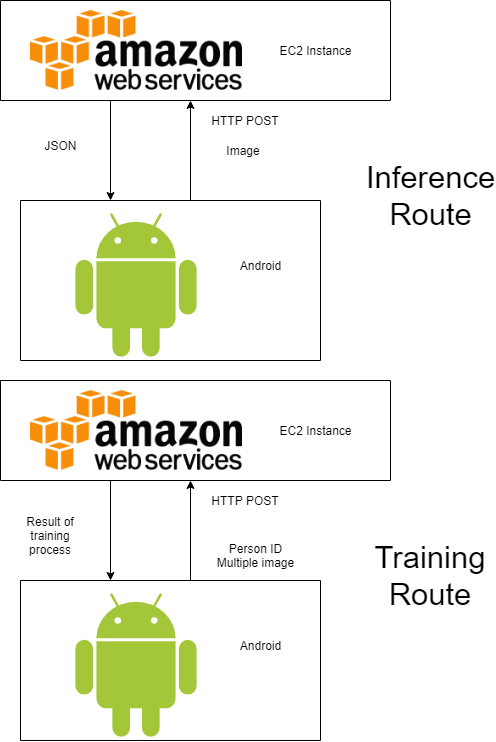
+ Thực hiện được việc nhận diện gương mặt với độ chính xác cao, ổn định (~ 80%) các gương mặt được nhận diện chính xác.

+ Ứng dụng server chạy ổn định, thời gian xử lý nhanh, ổn định.

+ Ứng dụng điểm danh trên PC có giao diện đơn giản, dễ sử dụng. Ứng dụng có thể lưu lại thông tin về các lần điểm danh, thời gian điểm danh theo ngày.

3.2 Phân tích

* Mô hình cần thực hiện ở quy mô nhỏ (một lớp học) nên database không cần quá lớn
* Để dễ dàng triển khai lên server và Cloud thì frameworks Flask được lựa chọn vì tính đơn giản và dễ triển khai của nó.
* Với phần xử lý bằng dưới PC, một giao diện GUI viết bằng PyQT hoặc giao diện web sẽ có lợi hơn, ở đây em chọn PyQT làm giao diện chính.
* Để đạt hiệu quả nhận diện chính xác nhất, em sử dụng mô hình nhận diện gương mặt facenet để lấy feature và SVM để phân loại từng người với nhau. Tuy nhiên cách làm này lại yêu cầu GPU để mang lại hiệu quả cao nhất, nhưng với điều kiện có hạn em chỉ có thể sử dụng CPU.
* Về cơ sở dữ liệu, MySQL được chọn vì tính bảo mật và tốc độ xử lý khá tốt của nó. MySQL có đặc điểm là có tính tổ chức và phân quyền người dùng cao.
* Để việc gửi ảnh được nhanh chóng và thuận tiện, em sử dụng giao thức HTTP như sau

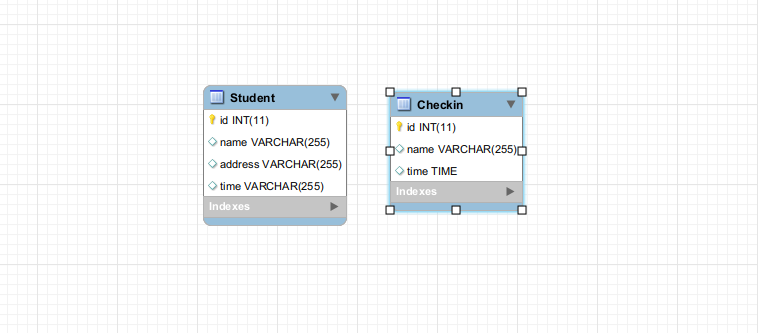


Sơ đồ khối hệ thống của việc nhận diện gương mặt [7]

Để thực hiện được việc nhận diện bằng tensorflow kết hợp Flask, ta cần phải tạo một graph Tensorflow là biến global để có thể sử dụng trong từng route của chương trình:

|  |  |
| --- | --- |
|  | |
|  | | SVM\_PATH = 'svm\_classifier.pkl' | | |
|  | | minsize = 20 | | |
|  | | threshold = [0.6, 0.7, 0.7] | | |
|  | | factor = 0.709 | | |
|  | | margin = 44 | | |
|  | | image\_size = 160 | | |
|  | | input\_map=None | | |
|  | | embedding\_size = 128 | | |
|  | | (model, class\_names) = pickle.load(open(SVM\_PATH, 'rb')) | | |
|  | | sess = tf.Session() | | |
|  | |  | | |
|  | | #model\_api = api.get\_model\_api() | | |
|  | | pnet, rnet, onet = mtcnn\_detect.create\_mtcnn(sess, 'models') | | |
|  | | def load\_graph(): | | |
|  | | with gfile.FastGFile(MODEL\_PATH,'rb') as f: | | |
|  | | graph\_def = tf.GraphDef() | | |
|  | | graph\_def.ParseFromString(f.read()) | | |
|  | | tf.import\_graph\_def(graph\_def, input\_map=input\_map, name='') | | |
|  | | with tf.Graph().as\_default() as graph: | | |
|  | | tf.import\_graph\_def( | | |
|  | | graph\_def, | | |
|  | | input\_map=None, | | |
|  | | return\_elements=None, | | |
|  | | name="" | | |
|  | | ) | | |
|  | |  | | |
|  | | return graph | | |
|  | |
| if \_\_name\_\_ == '\_\_main\_\_': | |
|  | | graph=load\_graph() |
|  | | images\_placeholder = graph.get\_tensor\_by\_name("input:0") |
|  | | embeddings = graph.get\_tensor\_by\_name("embeddings:0") |
|  | | phase\_train\_placeholder = graph.get\_tensor\_by\_name("phase\_train:0") |
|  | | embedding\_size = embeddings.get\_shape()[1] |
|  | |  |

Về thiết kế database, do ở quy mô thử nghiệm nên em thiết kế database khá đơn giản, database gồm 2 bảng, bảng thứ nhất là bảng tên người (Student), chỉ bao gồm tên và địa chỉ của một người được điểm danh, bảng thứ 2 là bảng checkin bảng này có trường tên và id người liên kết (CONSTRAINT) với bảng Student, bảng này sẽ được tạo mới theo ngày và tự động cập nhật sau khi có người được điểm danh, đồng thời để tránh điểm danh lặp, bảng này có thuộc tính không lặp lại tên.



# KẾT QUẢ THỰC HIỆN

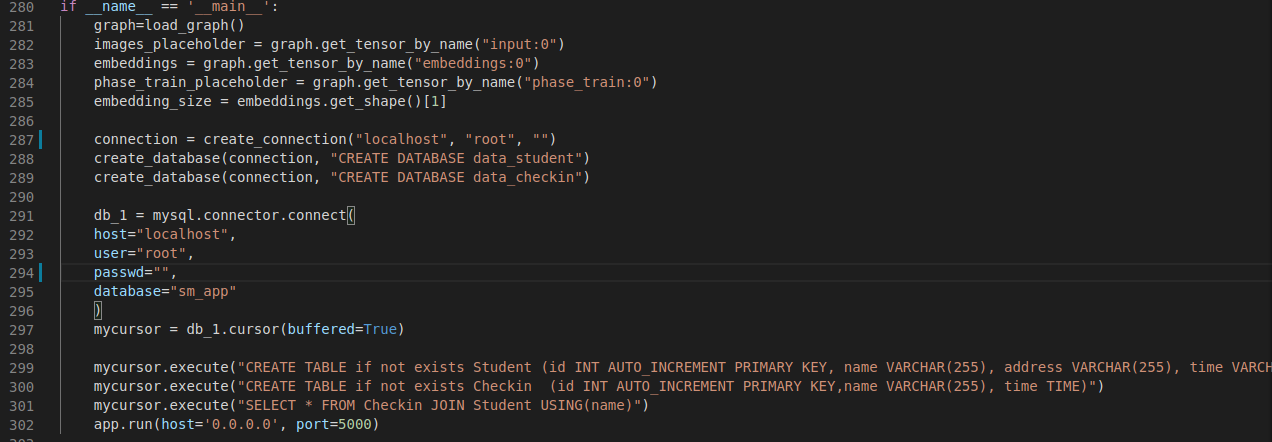
Thiết bị sử dụng trong đồ án: Laptop Dell E6540, CPU Intel i7 4800 MQ

Các thư viện Python sử dụng: Tensorflow, OpenCV, Flask,….

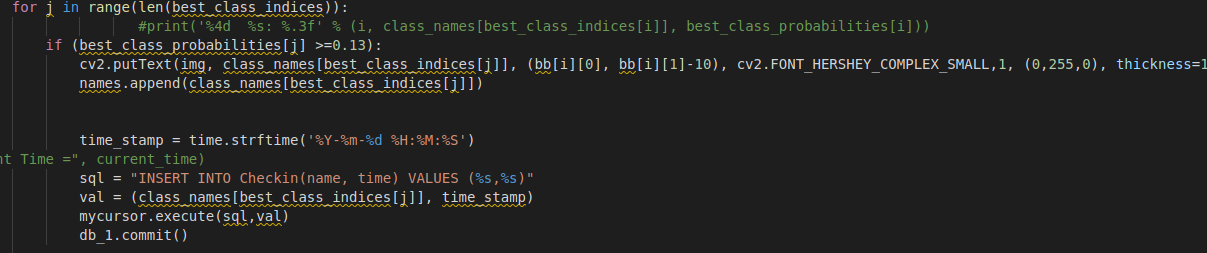
Để thực hiện các lệnh gọi HTTP, phần mềm Postman được sử dụng

4.1 Kết quả thực hiện phầm mềm

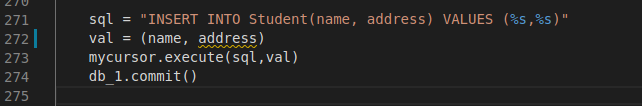
Để sử dụng SQL trong ứng dụng ta cần khởi tạo đối tượng connector trong hàm main và thực hiện tạo bảng nếu cần thiết



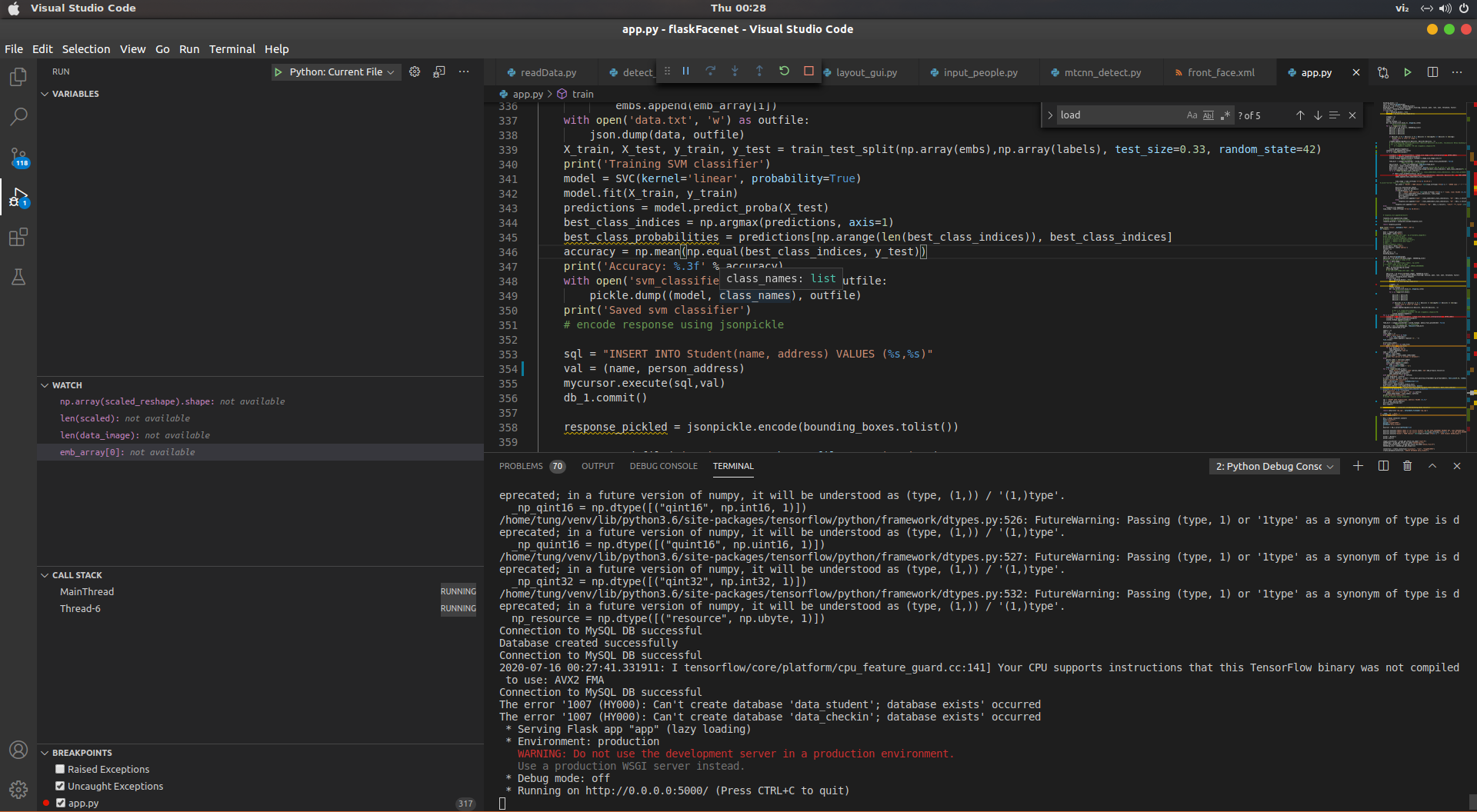
Trong route detect của app, dữ liệu về thời gian ra vào và tên được lưu vào bảng Checkin của database

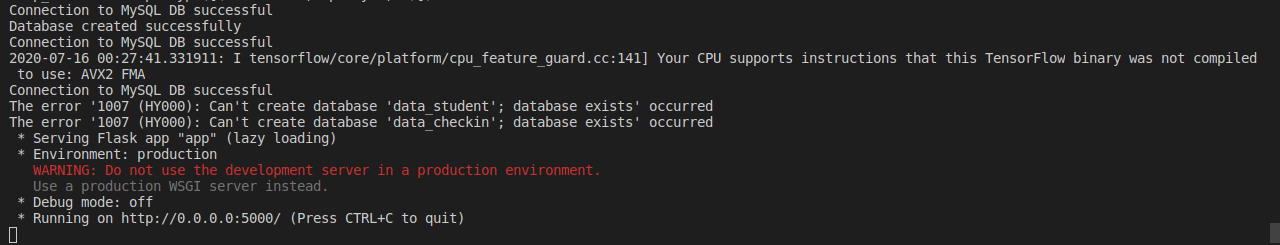


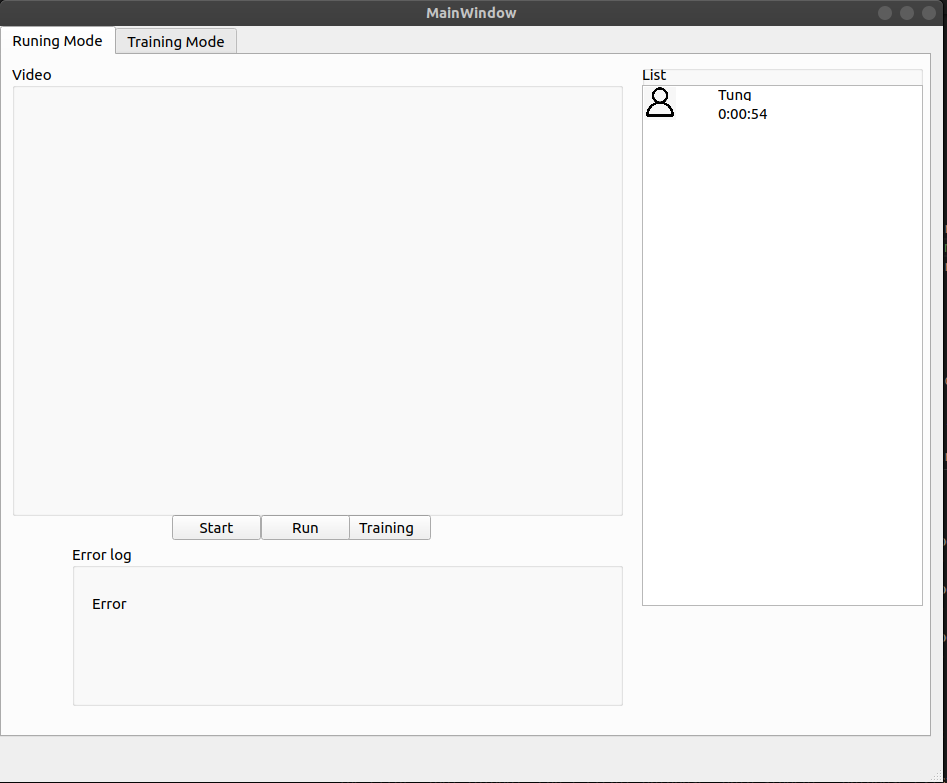
Trong route train của app, dữ liệu về Tên và địa chỉ của người được train sẽ được lưu lại vào database, ảnh của người được train sẽ được tính embedding vector và lưu lại thay vì lưu ảnh của họ



Server được bật ở localhost port 5000





GUI được viết bằng PyQT:

Các thành phần của GUI:

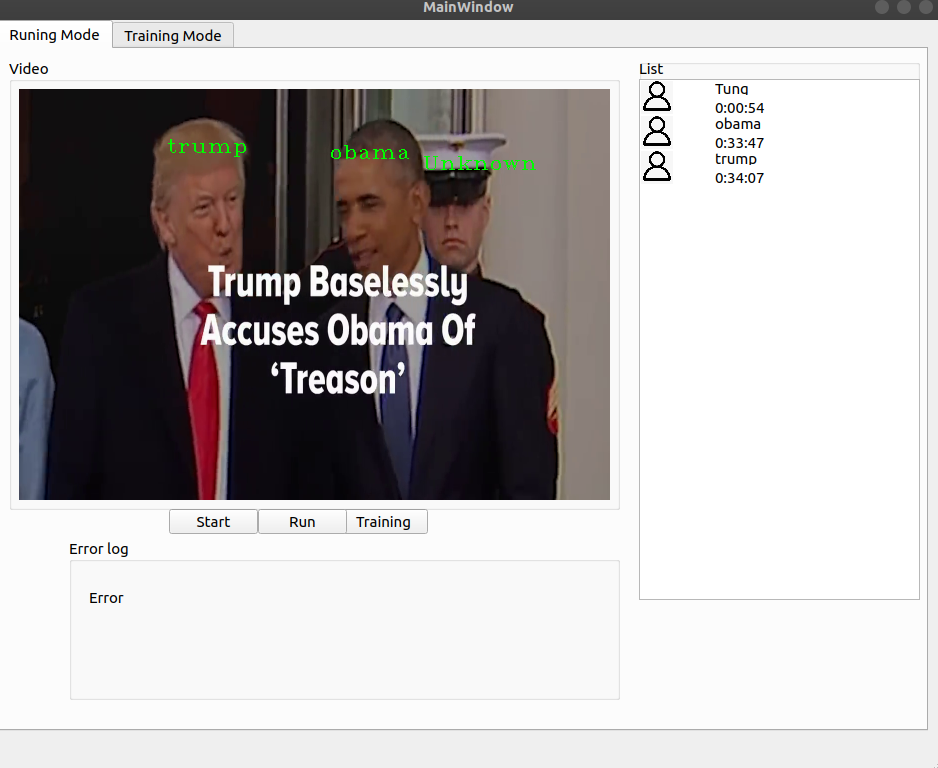
1. Phần video hiển thị video (có thể lấy từ webcam hoặc video).
2. Các nút bấm:

+ Start: Bật/tắt webcam hoặc video.

+ Run: Gửi data lên server để nhận diện gương mặt.

+ Training: Thu thập data của người đang ở trước webcam/video, đồng thời lưu dữ liệu về người đó trên CSDL.

1. Danh sách hiển thị tên từng người được điểm danh, cùng avatar của họ.
2. Error log hiển thị các lỗi xảy ra trong quá trình thực hiện chương trình.

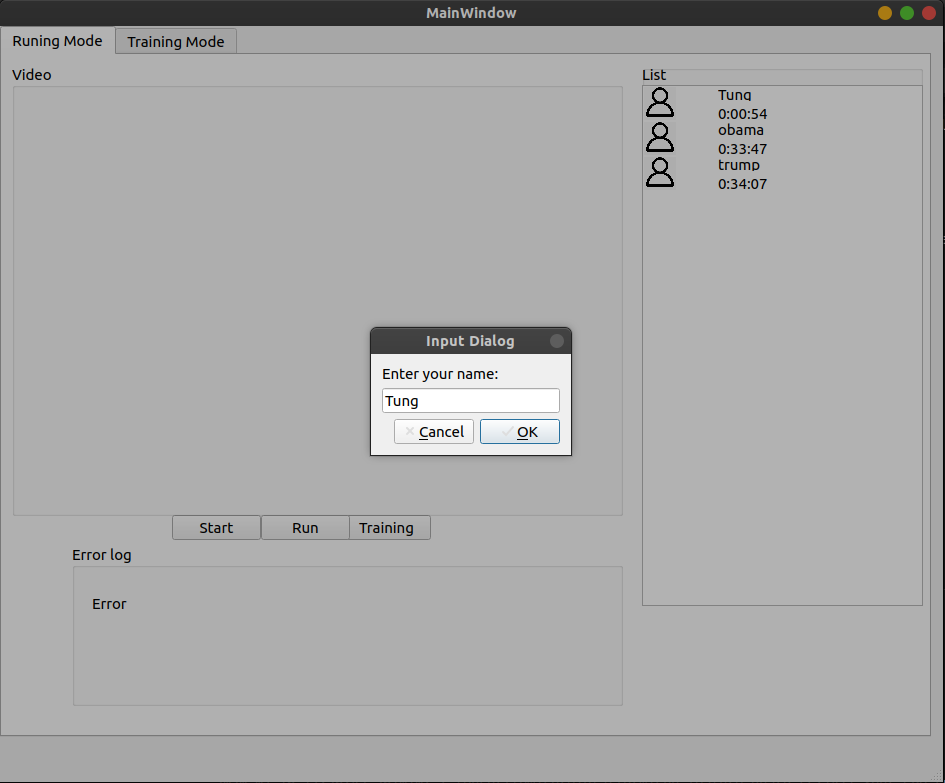


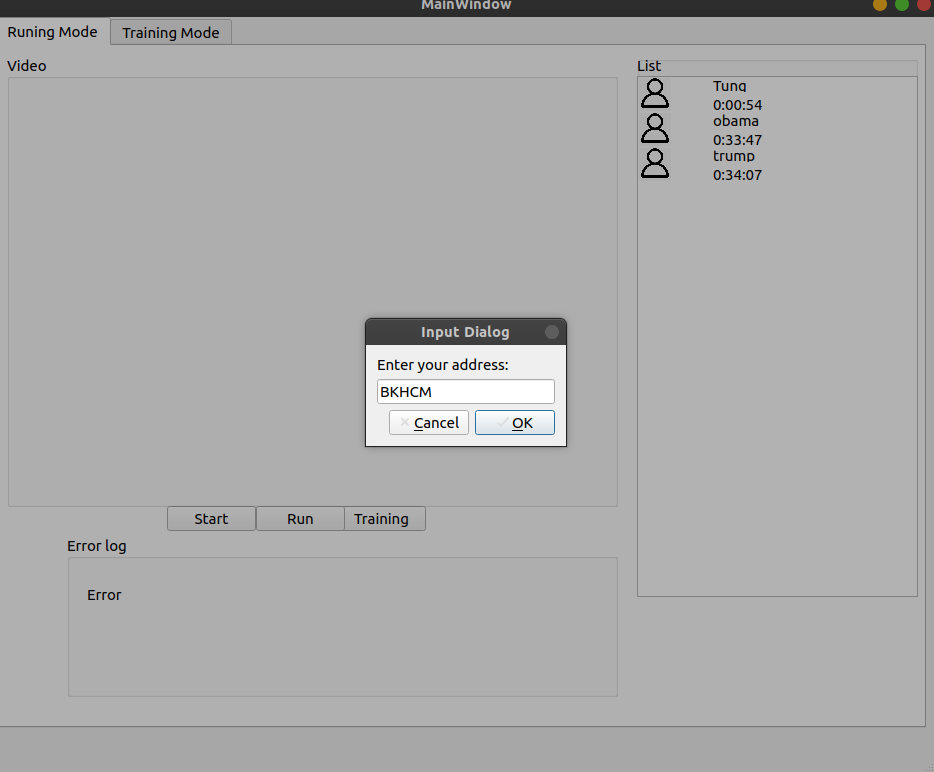
Ví dụ thực hiện chương trình với một video:

Những gương mặt đã được huấn luyện sẽ được nhận diện, trong khi những người chưa được huấn luyện sẽ hiển thị Unknown.

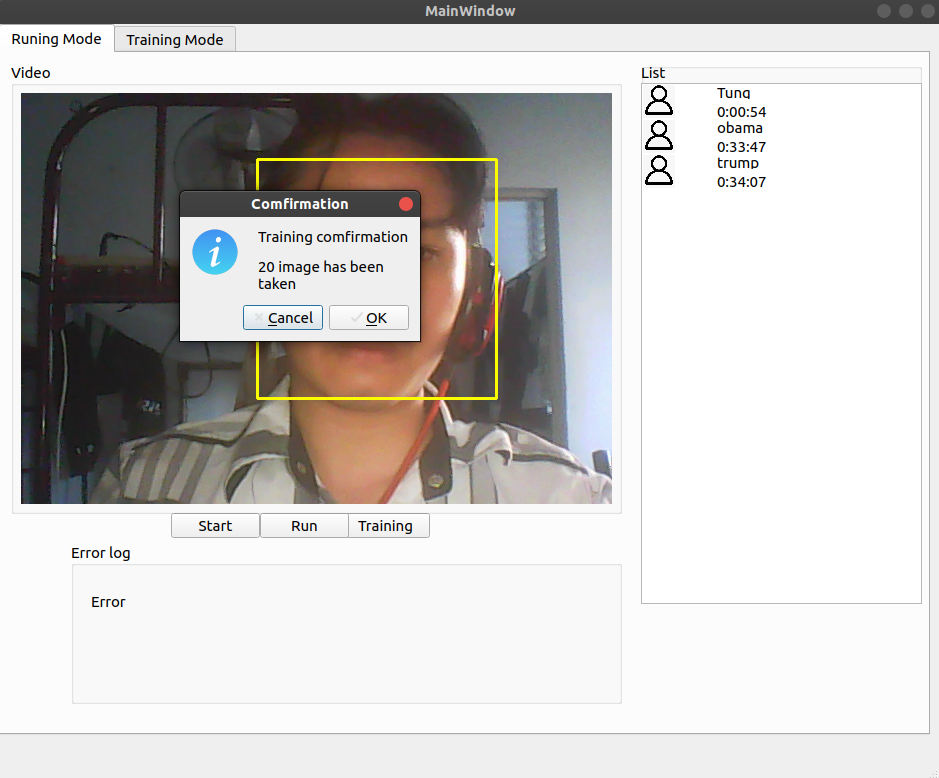
Danh sách điểm danh cũng sẽ được cập nhật ngày giờ mà gương mặt được phát hiện, trong khi đó CSDL được cập nhật.

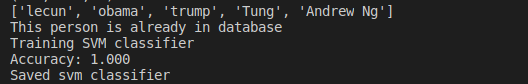
Quá trình training: Sau khi nhấn nút train, một dialog box sẽ xuất hiện để cập nhật tên và địa chỉ của người được thêm vào mô hình:





Sau khi đã nhấn xong training, ta nhấn start để bắt đầu chạy webcam. Quá trình training sẽ lấy đủ 20 ảnh có 1 gương mặt để training trên server.

-Nếu gương mặt đã có trong database, json trả về sẽ như sau:



Nếu người mới thêm vào không thuộc database, dòng This person is already in database sẽ không được thêm vào gói tin gửi trở lại.

Quá trình chạy chương trình:

Hình ảnh dưới dạng mảng numpy sẽ được ghi lại dưới dạng encode utf8 để truyền đi tròn body của một HTTP request lên server. Server sẽ thực hiện các bước nhận diện của Facenet:

1. Detect Face
2. Extract Feature
3. Recognize face
4. Đánh dấu vào Database
5. Gửi lại kết quả về Client.

Cấu trúc Json trả về như sau:

{ số gương mặt phát hiện được

{ “name”: Tên người

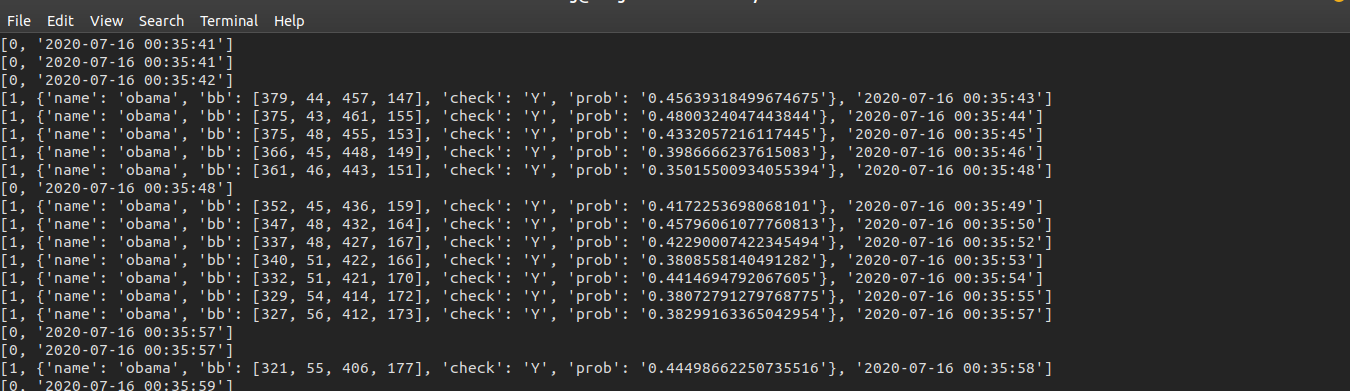
“bb”: Bounding box bao quanh gương mặt

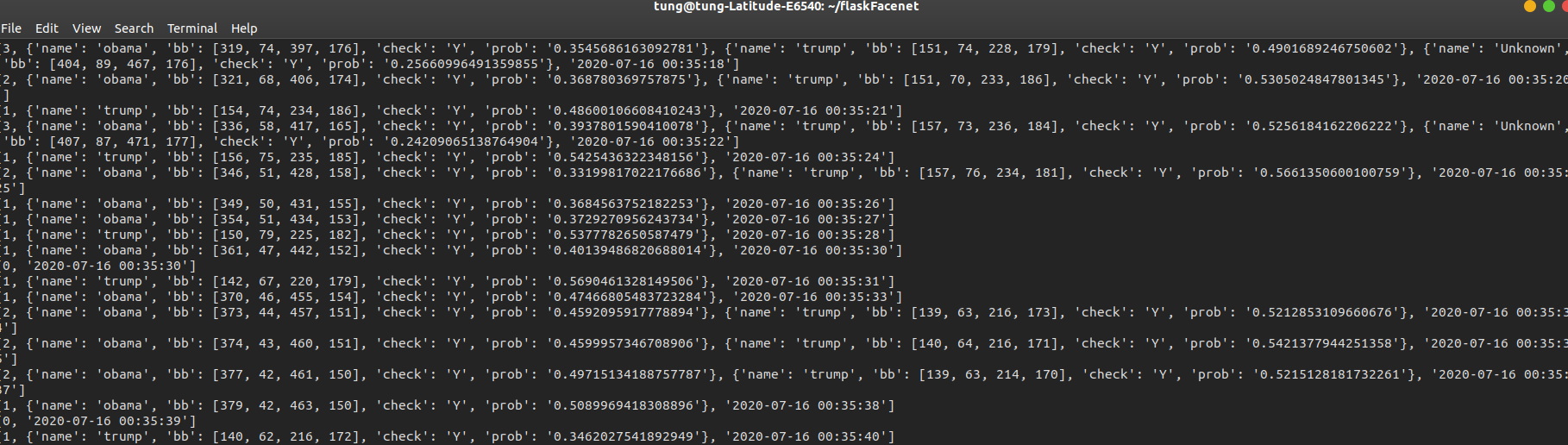
“check”: Biến kiếm tra người này đã được điểm danh chưa

“prob”: Độ chính xác của phép xác định này

}

‘Thời gian phát hiện’}

Trong đó chiều dài json này tùy thuộc vào số gương mặt phát hiện được.

Ví dụ trường hợp phát hiện được 2 gương mặt trong ảnh:

4.2 Đánh giá tốc độ chương trình

Thuật toán Face Recognition Facenet như đã trình bày gồm 3 bước:

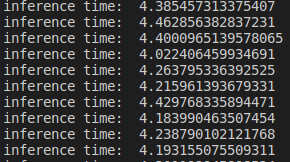
1) Face detection

2) Feature extraction

3) Face recognition

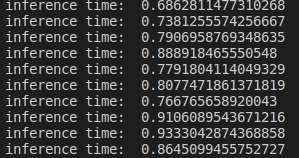
Do đó thời gian xử lý của thuật toán phụ thuộc vào số lượng gương mặt trong hình.

Trong trường hợp không detect được gương mặt thì thuật toán tương đương với thuật toán face detection với MTCNN, số fps trong 10 frame liên tiếp như sau:



=> Thời gian chạy thụât toán MTCNN trung bình là khoảng 1/(4.2) = 0.238s.

Trong trường hợp có 1 gương mặt, thời gian xử lý của server như sau:



=> Thời gian chạy này bao gồm thời gian detect face, thời gian chạy CNN và thời gian chạy SVM để phân loại từng gương mặt. Số fps xấp xỉ 0.9

=> Thời gian xử lý phần này là 1/0.9 = 1.11s

Như vậy thời gian xử lý CNN và SVM là 1.11 – 0.238 = 0.872 s

Tuy nhiên có thể nhận xét là thời gian chạy CNN đóng vai trò chủ yếu trong việc thực thi chương trình.

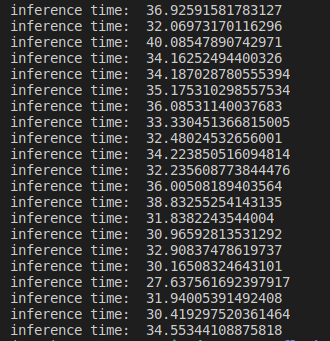
Thực hiện thí nghiệm so sánh thời gian truyền ảnh qua 2 giao thức HTTP và thư viện ZMQ

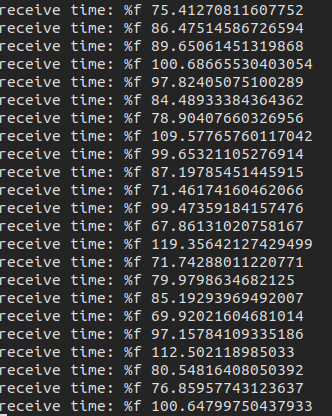
Thư viện ZMQ là thư viện chuyên cho việc xử lý truyền ảnh qua mạng, dựa trên nền tảng HTTP, thư viện ZMQ có thể được sử dụng với framework Flask của python.

Để chứng minh nghẽn cổ chai trong hệ thống là từ phần xử lý nhận diện, em setup thí nghiệm đối chứng 2 giao thức HTTP được sử dụng trong đồ án và việc truyền nhận bằng thư viện ZMQ trên local host:

Thí nghiệm được đặt ra là truyền 2 video có kích thước 640\*480 qua local host bằng 2 giao thức đang so sánh:

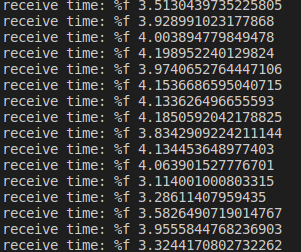
Số FPS được truyền qua giao thức HTTP POST:



Số FPS của một video được truyền qua ZMQ:

Có thể thấy tốc độ truyền ảnh qua ZMQ nhanh hơn HTTP từ 2 – 3 lần.

Tuy nhiên khi sử dụng ZMQ cho ứng dụng face detection với MTCNN thì số fps tương đương so với HTTP



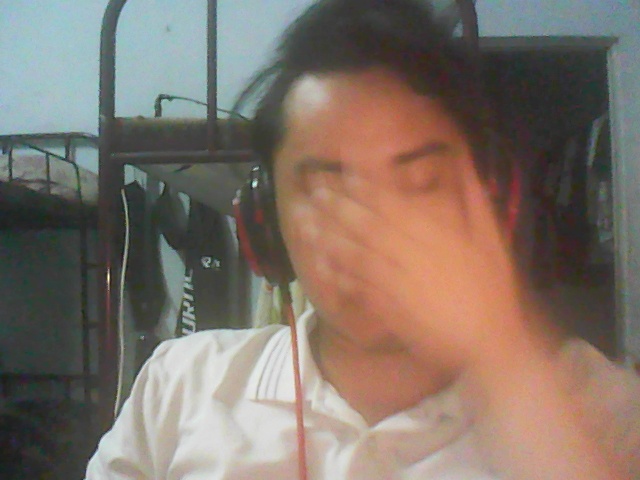
Điều này chứng tỏ rằng trong điều kiện thử nghiệm trên localhost, phương pháp sử dụng giao thức HTTP và ZMQ là có hiệu quả tương đương, nhưng xét về tốc độ truyền nhận data trên server, ZMQ chắc chắn là lựa chọn tốt hơn.

4.2 Đánh giá độ chính xác

Do bài toán nhận diện gương mặt bao gồm khá nhiều bước phức tạp, em chỉ có thể thực hiện đo đạc độ chính xác trên 100 hình nhận diện của bản thân như sau:

|  |  |  |
| --- | --- | --- |
| Lần thử | Số frame phát hiện được guơng mặt | Số frame nhận diện được guơng mặt |
| 1 | 77 | 77 |
| 2 | 90 | 89 |
| 3 | 98 | 97 |

Một số trường hợp không detect được gương mặt:







Ta có thể thấy độ chính xác của mô hình facenet khá cao, lỗi xuất hiện chủ yếu ở phần MTCNN, đồng thời MTCNN gặp lỗi khá nhiêu khi detect guơng mặt quay góc

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Ưu điểm:

Phần mềm thực hiện được các chức năng yêu cầu (nhận diện gương mặt, điểm danh, lưu vào CSDL, có GUI để chạy trên PC)

Nhược điểm:

Do giới hạn của phần cứng, kết quả thực hiện nhận diện gương mặt còn khá chậm

Phương pháp lấy mẫu data để train còn thiếu sót các trường hợp khó. các góc của gương mặt

Giao diện đồ họa (GUI) còn khá sơ sài, thiếu các chức năng tra cứu CSDL người dùng.

## Hướng phát triển

Các hướng phát triển cho đề tài:

+ Sử dụng phần cứng tốt hơn (GPU) để tăng hiệu quả nhận diện gương mặt.

+ Chỉnh sửa giao diện người dùng đẹp hơn, chuyên nghiệp hơn.

+ Thực hiện các bước lấy data training tốt hơn để đạt độ chính xác cao hơn.

# TÀI LIỆU THAM KHẢO

[1] Facenet, <https://github.com/davidsandberg/facenet>

[2] FaceNet: A Unified Embedding for Face Recognition and Clustering, <https://arxiv.org/abs/1503.03832>

[3] towardsdatascience, Understanding CNN (Convolutional Neural Network), <https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4>

[4] Đặng Lâm Tùng, Deep learning và bài toán face recognition, <https://forum.machinelearningcoban.com/t/deep-learning-va-bai-toan-face-recognition/1776>

[5]Wikipedia, Hypertext Transfer Protocol, <https://vi.wikipedia.org/wiki/Hypertext_Transfer_Protocol>

[6] viblo, Phương thức GET và POST, <https://viblo.asia/p/phuong-thuc-get-va-post-aWj53VBYl6m>

[7] Tung Dang, An Efficient Deep Learning Pipeline for Face Recognition on Mobile Environment, *ICAST, 2018*

# PHỤ LỤC

Mã nguồn chương trình:

Server:

|  |
| --- |
| from flask import Flask,request, jsonify, Response,send\_file |
|  | from tensorflow.python.platform import gfile |
|  | import numpy as np |
|  | import cv2 |
|  | import tensorflow as tf |
|  | import mtcnn\_detect |
|  | import jsonpickle |
|  | import facenet |
|  | import pickle |
|  | import time |
|  | import json |
|  | import os |
|  | import sql |
|  | from sklearn.svm import SVC |
|  | from sklearn.model\_selection import train\_test\_split |
|  | import time |
|  |  |
|  | import mysql.connector |
|  | from mysql.connector import Error |
|  |  |
|  | def create\_connection(host\_name, user\_name, user\_password): |
|  | connection = None |
|  | try: |
|  | connection = mysql.connector.connect( |
|  | host=host\_name, |
|  | user=user\_name, |
|  | passwd=user\_password |
|  | ) |
|  | print("Connection to MySQL DB successful") |
|  | except Error as e: |
|  | print(f"The error '{e}' occurred") |
|  |  |
|  | return connection |
|  |  |
|  | connection = create\_connection("localhost", "root", "Tung3071999%") |
|  |  |
|  | def create\_database(connection, query): |
|  | cursor = connection.cursor() |
|  | try: |
|  | cursor.execute(query) |
|  | print("Database created successfully") |
|  | except Error as e: |
|  | print(f"The error '{e}' occurred") |
|  |  |
|  | app = Flask(\_\_name\_\_) |
|  |  |
|  | #IMG\_FOLDER = os.path.join('static') |
|  |  |
|  | MODEL\_PATH = './models/frozen.pb' |
|  | SVM\_PATH = 'svm\_classifier.pkl' |
|  | minsize = 20 |
|  | threshold = [0.6, 0.7, 0.7] |
|  | factor = 0.709 |
|  | margin = 44 |
|  | image\_size = 160 |
|  | input\_map=None |
|  | embedding\_size = 128 |
|  | (model, class\_names) = pickle.load(open(SVM\_PATH, 'rb')) |
|  | sess = tf.Session() |
|  |  |
|  | #model\_api = api.get\_model\_api() |
|  | pnet, rnet, onet = mtcnn\_detect.create\_mtcnn(sess, 'models') |
|  | def load\_graph(): |
|  | with gfile.FastGFile(MODEL\_PATH,'rb') as f: |
|  | graph\_def = tf.GraphDef() |
|  | graph\_def.ParseFromString(f.read()) |
|  | tf.import\_graph\_def(graph\_def, input\_map=input\_map, name='') |
|  | with tf.Graph().as\_default() as graph: |
|  | tf.import\_graph\_def( |
|  | graph\_def, |
|  | input\_map=None, |
|  | return\_elements=None, |
|  | name="" |
|  | ) |
|  |  |
|  | return graph |
|  | @app.route("/test\_post", methods=['POST','GET']) |
|  | def index\_post(): |
|  | """ |
|  | this is a root dir of my server |
|  | :return: str |
|  | """ |
|  | name = request.get\_data("name",as\_text = True) |
|  | return str(name) |
|  |  |
|  | @app.route("/") |
|  | def index(): |
|  | """ |
|  | this is a root dir of my server |
|  | :return: str |
|  | """ |
|  |  |
|  | return "This is root!!!!" |
|  | graph=load\_graph() |
|  | @app.route('/post', methods=['POST','GET']) |
|  | def test(): |
|  | #read image file string data |
|  | filestr = request.files['file'].read() |
|  | #convert string data to numpy array |
|  | npimg = np.fromstring(filestr, np.uint8) |
|  | # convert numpy array to image |
|  | img = cv2.imdecode(npimg, cv2.IMREAD\_UNCHANGED) |
|  |  |
|  | bounding\_boxes = [] |
|  | sess= tf.Session(graph=graph) |
|  | emb\_array = np.zeros((1, embedding\_size)) |
|  | bounding\_boxes, \_ = mtcnn\_detect.detect\_face(img, minsize, pnet, rnet, onet, threshold, factor) |
|  | nrof\_faces = bounding\_boxes.shape[0] |
|  | if nrof\_faces > 0: |
|  | det = bounding\_boxes[:, 0:4] |
|  | img\_size = np.asarray(img.shape)[0:2] |
|  |  |
|  | cropped = [] |
|  | scaled = [] |
|  | names = [] |
|  | scaled\_reshape = [] |
|  | bb = np.zeros((nrof\_faces,4), dtype=np.int32) |
|  | #print(nrof\_faces) |
|  | for i in range(nrof\_faces): |
|  | emb\_array = np.zeros((1, embedding\_size)) |
|  | bb[i][0] = det[i][0] |
|  | bb[i][1] = det[i][1] |
|  | bb[i][2] = det[i][2] |
|  | bb[i][3] = det[i][3] |
|  |  |
|  | if bb[i][0] <= 0 or bb[i][1] <= 0 or bb[i][2] >= len(img[0]) or bb[i][3] >= len(img): |
|  | print('face is inner of range!') |
|  | continue |
|  | cropped.append(img[bb[i][1]:bb[i][3], bb[i][0]:bb[i][2], :]) |
|  | cv2.rectangle(img,(bb[i][0],bb[i][1]),(bb[i][2],bb[i][3]),(0,0,255), thickness=2) #draw bounding box for the face |
|  | # for i in range(len(cropped)): |
|  | # if cropped[i].shape[0] >70 and cropped[i].shape[1]>70: |
|  | scaled.append(cropped[i]) |
|  | for i in range(len(scaled)): |
|  | scaled[i] = cv2.resize(scaled[i], (image\_size,image\_size),interpolation=cv2.INTER\_CUBIC) |
|  | scaled[i] = facenet.prewhiten(scaled[i]) |
|  | scaled\_reshape.append(scaled[i].reshape(-1,image\_size,image\_size,3)) |
|  | #Feed forward |
|  | feed\_dict = {images\_placeholder: scaled\_reshape[i], phase\_train\_placeholder: False} |
|  | #Đưa vector emb vào classifier |
|  | emb\_array[0, :] = sess.run(embeddings, feed\_dict=feed\_dict) |
|  | predictions = model.predict\_proba(emb\_array) |
|  | best\_class\_indices = np.argmax(predictions, axis=1) #Class đạt độ chính xác cao nhất |
|  | best\_class\_probabilities = predictions[np.arange(len(best\_class\_indices)), best\_class\_indices]#Độ chính xác |
|  | for j in range(len(best\_class\_indices)): |
|  | #print('%4d %s: %.3f' % (i, class\_names[best\_class\_indices[i]], best\_class\_probabilities[i])) |
|  | if (best\_class\_probabilities[j] >=0.13): |
|  | cv2.putText(img, class\_names[best\_class\_indices[j]], (bb[i][0], bb[i][1]-10), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,1, (0,255,0), thickness=1, lineType=2) |
|  | names.append(class\_names[best\_class\_indices[j]]) |
|  |  |
|  |  |
|  | time\_stamp = time.strftime('%Y-%m-%d %H:%M:%S') |
|  | # print("Current Time =", current\_time) |
|  | sql = "INSERT INTO Checkin(name, time) VALUES (%s,%s)" |
|  | val = (class\_names[best\_class\_indices[j]], time\_stamp) |
|  | mycursor.execute(sql,val) |
|  | db\_1.commit() |
|  |  |
|  | else: |
|  | cv2.putText(img, "Unknown", (bb[i][0], bb[i][1]-10), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,1, (0,255,0), thickness=1, lineType=2) |
|  | cv2.imwrite('img.jpg',img) |
|  | # encode response using jsonpickle |
|  | response\_pickled = jsonpickle.encode(bounding\_boxes.tolist()) |
|  |  |
|  | return send\_file('img.jpg', attachment\_filename='img.jpg') |
|  |  |
|  | @app.route('/train', methods=['POST','GET']) |
|  | def train(): |
|  |  |
|  | #read image file string data |
|  | filestr = request.files.getlist("file") |
|  | namestr = request.files.get("name") |
|  | name = [] |
|  | # print(namestr.read()) |
|  | person\_data = (namestr.read().decode("utf-8")).split("\n") |
|  | npimg = [] |
|  | img = [] |
|  | data\_person = [] |
|  | bounding\_boxes = [] |
|  |  |
|  | sess= tf.Session(graph=graph) |
|  | emb\_array = np.zeros((1, embedding\_size)) |
|  | #convert string data to numpy array |
|  | for i in filestr: |
|  | npimg = np.fromstring(i.read(), np.uint8) |
|  | # convert numpy array to image |
|  | img = cv2.imdecode(npimg, cv2.IMREAD\_UNCHANGED) |
|  | emb\_array = np.zeros((1, embedding\_size)) |
|  | bounding\_boxes, \_ = mtcnn\_detect.detect\_face(img, minsize, pnet, rnet, onet, threshold, factor) |
|  | nrof\_faces = bounding\_boxes.shape[0] |
|  | if nrof\_faces > 0: |
|  | det = bounding\_boxes[:, 0:4] |
|  | img\_size = np.asarray(img.shape)[0:2] |
|  |  |
|  | cropped = [] |
|  | scaled = [] |
|  | names = [] |
|  | scaled\_reshape = [] |
|  | bb = np.zeros((nrof\_faces,4), dtype=np.int32) |
|  | #print(nrof\_faces) |
|  | for i in range(nrof\_faces): |
|  | emb\_array = np.zeros((1, embedding\_size)) |
|  | bb[i][0] = det[i][0] |
|  | bb[i][1] = det[i][1] |
|  | bb[i][2] = det[i][2] |
|  | bb[i][3] = det[i][3] |
|  |  |
|  | if bb[i][0] <= 0 or bb[i][1] <= 0 or bb[i][2] >= len(img[0]) or bb[i][3] >= len(img): |
|  | print('face is inner of range!') |
|  | continue |
|  | cropped.append(img[bb[i][1]:bb[i][3], bb[i][0]:bb[i][2], :]) |
|  | cv2.rectangle(img,(bb[i][0],bb[i][1]),(bb[i][2],bb[i][3]),(0,0,255), thickness=2) #draw bounding box for the face |
|  | # for i in range(len(cropped)): |
|  | # if cropped[i].shape[0] >70 and cropped[i].shape[1]>70: |
|  | scaled.append(cropped[i]) |
|  | for i in range(len(scaled)): |
|  | scaled[i] = cv2.resize(scaled[i], (image\_size,image\_size),interpolation=cv2.INTER\_CUBIC) |
|  | scaled[i] = facenet.prewhiten(scaled[i]) |
|  | scaled\_reshape.append(scaled[i].reshape(-1,image\_size,image\_size,3)) |
|  | #Feed forward |
|  | feed\_dict = {images\_placeholder: scaled\_reshape[i], phase\_train\_placeholder: False} |
|  | #Đưa vector emb vào classifier |
|  | emb\_array[0, :] = sess.run(embeddings, feed\_dict=feed\_dict) |
|  | data\_person.append(emb\_array) |
|  |  |
|  | labels = [] |
|  | embs = [] |
|  | class\_names = [] |
|  | with open('class.txt') as file: |
|  | for l in file.readlines(): |
|  | class\_names.append(l.replace('\n', '')) |
|  | file.close() |
|  |  |
|  | print(class\_names) |
|  | with open('data.txt') as json\_file: |
|  | data = json.load(json\_file) |
|  | for p in data['person']: |
|  | embs.append(p['emb']) |
|  | labels.append(p['name']) |
|  | name = person\_data[0] |
|  | if name in class\_names: |
|  | person\_label = class\_names.index(name) |
|  | print('This person is already in database') |
|  | else: |
|  | person\_label = len(class\_names) |
|  | file = open('class.txt','w') |
|  | class\_names.append(str(name)) |
|  | for name in class\_names: |
|  | file.write(str(name) + '\n') |
|  | file.close() |
|  | for i in range(len(emb\_array)): |
|  | data['person'].append({'name':person\_label,'emb':emb\_array[i].tolist()}) |
|  | labels.append(person\_label) |
|  | embs.append(emb\_array[i]) |
|  | with open('data.txt', 'w') as outfile: |
|  | json.dump(data, outfile) |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(np.array(embs),np.array(labels), test\_size=0.33, random\_state=42) |
|  | print('Training SVM classifier') |
|  | model = SVC(kernel='linear', probability=True) |
|  | model.fit(X\_train, y\_train) |
|  | predictions = model.predict\_proba(X\_test) |
|  | best\_class\_indices = np.argmax(predictions, axis=1) |
|  | best\_class\_probabilities = predictions[np.arange(len(best\_class\_indices)), best\_class\_indices] |
|  | accuracy = np.mean(np.equal(best\_class\_indices, y\_test)) |
|  | print('Accuracy: %.3f' % accuracy) |
|  | with open('svm\_classifier1.pkl', 'wb') as outfile: |
|  | pickle.dump((model, class\_names), outfile) |
|  | print('Saved svm classifier') |
|  | # encode response using jsonpickle |
|  |  |
|  | sql = "INSERT INTO Student(name, address) VALUES (%s,%s)" |
|  | val = (name, address) |
|  | mycursor.execute(sql,val) |
|  | db\_1.commit() |
|  |  |
|  | response\_pickled = jsonpickle.encode(bounding\_boxes.tolist()) |
|  |  |
|  | return send\_file('img.jpg', attachment\_filename='img.jpg') |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | graph=load\_graph() |
|  | images\_placeholder = graph.get\_tensor\_by\_name("input:0") |
|  | embeddings = graph.get\_tensor\_by\_name("embeddings:0") |
|  | phase\_train\_placeholder = graph.get\_tensor\_by\_name("phase\_train:0") |
|  | embedding\_size = embeddings.get\_shape()[1] |
|  |  |
|  | connection = create\_connection("localhost", "root", "Tung3071999%") |
|  | create\_database(connection, "CREATE DATABASE data\_student") |
|  | create\_database(connection, "CREATE DATABASE data\_checkin") |
|  |  |
|  | db\_1 = mysql.connector.connect( |
|  | host="localhost", |
|  | user="root", |
|  | passwd="Tung3071999%", |
|  | database="sm\_app" |
|  | ) |
|  | mycursor = db\_1.cursor(buffered=True) |
|  |  |
|  | mycursor.execute("CREATE TABLE if not exists Student (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255), time VARCHAR(255))") |
|  | mycursor.execute("CREATE TABLE if not exists Checkin (id INT AUTO\_INCREMENT PRIMARY KEY,name VARCHAR(255), time TIME)") |
|  | mycursor.execute("SELECT \* FROM Checkin JOIN Student USING(name)") |
|  | app.run(host='0.0.0.0', port=5000) |