

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**  
**KHOA ĐIỆN - ĐIỆN TỬ**  
**Bộ Môn Điều Khiển và Tự Động**

-----o0o-----



**BÁO CÁO ĐỒ ÁN 1**

**NHẬN DIỆN KHUÔN MẶT BẰNG BẰNG MTCNN VÀ FACENET**  
**LỚP L01 --- HK 222**

**GVHD : Cô Hồ Thanh Phương**

**SVTH : Nguyễn Văn Mạnh MSSV : 2013737**  
**Trần Trung Tín MSSV : 2014757**

TP.Hồ Chí Minh, tháng 05 năm 2023

## **Lời cảm ơn**

Để hoàn thành đề tài này, chúng em xin gửi lời cảm ơn đến các Quý Thầy cô Khoa Điện – Điện tử, Trường Đại học Bách Khoa – Đại học Quốc Gia Thành phố Hồ Chí Minh đã tạo cơ hội được học tập, rèn luyện và tích lũy kiến thức, kỹ năng để thực hiện đề tài. Đặc biệt, chúng em xin gửi lời cảm ơn đến Cô Hồ Thanh Phương đã tận tình giúp đỡ trong quá trình chọn đề tài, chỉ dẫn, theo dõi và đưa ra những lời khuyên bổ ích giúp chúng em giải quyết được các vấn đề gặp phải trong quá trình nghiên cứu và hoàn thành đề tài một cách tốt nhất.

Một lần nữa chân thành cảm ơn thầy. Do kiến thức của bản thân còn hạn chế và thiếu kinh nghiệm thực tiễn nên nội dung khóa luận khó tránh những thiếu sót. Chúng em rất mong nhận sự góp ý, chỉ dạy thêm từ Quý Thầy cô. Cuối cùng, chúng em xin chúc Quý Thầy cô luôn thật nhiều sức khỏe và đạt được nhiều thành công trong công việc.

Trân trọng cảm ơn!

*TP.Hồ Chí Minh, tháng 05 năm 2023*  
Nhóm sinh viên thực hiện

**Trần Trung Tín**  
**Nguyễn Văn Mạnh**

# MỤC LỤC

MỤC LỤC .....	i
CHƯƠNG 1 Giới thiệu tổng quan.....	1
1.1 Tổng quan về nhận diện khuôn mặt .....	1
1.2 Phân tích bài toán .....	1
1.3 Các công cụ .....	1
1.3.1 Python (sử dụng pycharm) .....	1
1.3.2 Thư viện OpenCV .....	2
1.3.3 Các thư viện dùng trong pycharm .....	3
CHƯƠNG 2 Giới thiệu thuật toán.....	3
2.1 Giới thiệu cấu trúc mạng CNN .....	4
2.1.1 Giới thiệu .....	4
2.1.2 Lớp tích chập - Convolution Layer .....	5
2.1.3 Bước nhảy – Stride .....	7
2.1.4 Đường viền – Padding .....	8
2.1.5 Hàm phi tuyến – ReLU .....	8
2.1.6 Lớp gộp - Pooling Layer .....	9
2.1.7 Tóm tắt.....	9
2.2 Tổng quan về MTCNN .....	10
2.2.1 Stage 1: P-Net.....	10
2.2.2 Stage 2: R-Net .....	12
2.2.3 Stage 3: O-Net .....	12
2.3 Tổng quan về FaceNet .....	13
2.3.1 Các khái niệm cơ bản .....	13
CHƯƠNG 3 Quy trình thực hiện và kết quả.....	14

3.1	Tạo môi trường ảo bằng Anaconda và cài các thư viện hỗ trợ .....	14
3.2	Face Detection với MTCNN .....	14
3.2.1	Detect Face bằng OpenCV và MTCNN .....	14
3.2.2	Capture Face: .....	16
3.3	Cập nhật FaceList và Face Recognition với FaceNet: .....	18
3.3.1	Cập nhật FaceList: .....	18
3.3.2	FaceRecognition dựa theo FaceList: .....	21
3.4	Kết quả .....	25
CHƯƠNG 4 Kết luận và hướng phát triển.....		26
4.1	Kết luận .....	26
4.2	Hướng Phát triển .....	27
Tài liệu tham khảo .....		27

# CHƯƠNG 1 Giới thiệu tổng quan

## 1.1 Tổng quan về nhận diện khuôn mặt

Nhận dạng khuôn mặt là bài toán phổ biến hiện nay vì nó có thể được sử dụng trong rất nhiều ứng dụng, đặc biệt là các ứng dụng chăm công nhân viên. Do đó, chúng em quyết định áp dụng bài toán nhận dạng khuôn mặt vào việc điểm danh sinh viên. Theo cách truyền thống, khi muốn điểm danh sinh viên, giáo viên thường điểm danh bằng cách gọi tên sinh viên trong danh sách. Phương pháp điểm danh này làm mất rất nhiều thời gian của giáo viên và sinh viên. Hiện nay, một số giáo viên có thể sử dụng thêm phương pháp điểm danh qua các nền tảng hỗ trợ quản lý học tập. Cụ thể, sinh viên phải đăng nhập tài khoản của mình và tham gia điểm danh khi giáo viên yêu cầu. Phương pháp này nhanh hơn rất nhiều so với điểm danh đọc tên truyền thống, nhưng cũng có những bất cập dẫn đến việc sinh viên không điểm danh được, chẳng hạn như sinh viên không có các thiết bị thông minh để truy cập vào hệ thống, đường truyền mạng bất ổn định hay máy chủ Web bị quá tải dẫn đến không đăng nhập vào hệ thống được. Bên cạnh đó, sinh viên vẫn có thể điểm danh hộ hoặc vẫn điểm danh được dù không có mặt tại lớp. Trong những năm gần đây, nhờ sự phát triển cao của công nghệ nhận dạng sinh trắc học, trong đó bao gồm công nghệ nhận diện khuôn mặt, chúng ta có thể hoàn toàn thay thế phương pháp điểm danh truyền thống bằng phương pháp điểm danh tự động và thông minh sử dụng công nghệ nhận diện khuôn mặt từ ảnh hoặc camera. Rõ ràng, điểm danh bằng cách nhận diện khuôn mặt bắt buộc người muốn điểm danh phải xuất hiện trong khung hình chụp hoặc trong khung hình camera và đương nhiên không ai có thể điểm danh hộ được. Ứng dụng điểm danh này sẽ tự động nhận diện và định danh sinh viên, lưu vào danh sách điểm danh cho nên giáo viên tiết kiệm được rất nhiều thời gian và công sức.

## 1.2 Phân tích bài toán

Một hệ thống nhận diện mặt người thông thường bao gồm bốn bước xử lý sau:

1. Phát hiện khuôn mặt (Face Detection).
2. Phân đoạn khuôn mặt (Face Alignment hay Segmentation).
3. Trích chọn đặc trưng (Feature Extraction).
4. Nhận diện (Recognition) hay Phân lớp khuôn mặt (Face Classification)

## 1.3 Các công cụ

### 1.3.1 Python (sử dụng pycharm)

Python là một ngôn ngữ lập trình thông dịch (interpreted), hướng đối tượng (object-oriented), và là một ngôn ngữ bậc cao (high-level) ngữ nghĩa động (dynamic semantics). Python hỗ trợ các module và gói (packages), khuyến khích chương trình module hóa và tái sử

dụng mã. Trình thông dịch Python và thư viện chuẩn mở rộng có sẵn dưới dạng mã nguồn hoặc dạng nhị phân miễn phí cho tất cả các nền tảng chính và có thể được phân phối tự do

Các đặc điểm của Python:

- Ngữ pháp đơn giản, dễ đọc.
- Vừa hướng thủ tục (procedural-oriented), vừa hướng đối tượng (object-oriented)
- Hỗ trợ module và hỗ trợ gói (package)
- Xử lý lỗi bằng ngoại lệ (Exception)
- Kiểu dữ liệu động ở mức cao.
- Có các bộ thư viện chuẩn và các module ngoài, đáp ứng tất cả các nhu cầu lập trình.
- Có khả năng tương tác với các module khác viết trên C/C++ (Hoặc Java cho Jython, hoặc .Net cho IronPython).
- Có thể nhúng vào ứng dụng như một giao tiếp kịch bản (scripting interface).

### 1.3.2 Thư viện OpenCV

Opencv (Open Computer Vision library) do Intel phát triển, được giới thiệu năm 1999 và hoàn thiện thành phiên bản 1.0 năm 2006. Thư viện opencv – gồm khoảng 500 hàm – được viết bằng ngôn ngữ lập trình C và tương thích với các hệ điều hành Windows, Linux, Mac OS... đóng vai trò xác lập chuẩn giao tiếp, dữ liệu, thuật toán cho lĩnh vực CV và tạo điều kiện cho mọi người tham gia nghiên cứu và phát triển ứng dụng.

Trước Opencv không có một công cụ chuẩn nào cho lĩnh vực xử lý ảnh. Các đoạn code đơn lẻ do các nhà nghiên cứu tự viết thường không thống nhất và không ổn định. Các bộ công cụ thương mại như Matlab, Simulink,...v.v.. lại có giá cao chỉ thích hợp cho các công ty phát triển các ứng dụng lớn. Ngoài ra còn có các giải pháp kèm theo thiết bị phần cứng mà phần lớn là mã đóng và được thiết kế riêng cho từng thiết bị, rất khó khăn cho việc mở rộng ứng dụng.

OpenCV là công cụ hữu ích cho những người bước đầu làm quen với xử lý ảnh số vì các ưu điểm sau:

- OpenCV là công cụ chuyên dụng: được Intel phát triển theo hướng tối ưu hóa cho các ứng dụng xử lý và phân tích ảnh, với cấu trúc dữ liệu hợp lý, thư viện tạo giao diện, truy xuất thiết bị phần cứng được tích hợp sẵn. OpenCV thích hợp để phát triển nhanh ứng dụng
- OpenCV là công cụ mã nguồn mở: Không chỉ là công cụ miễn phí, việc được xây dựng trên mã nguồn mở giúp OpenCV trở thành công cụ thích hợp cho nghiên cứu và phát triển, với khả năng thay đổi và mở rộng các mô hình, thuật toán

- OpenCV đã được sử dụng rộng rãi: Từ năm 1999 đến nay, OpenCV đã thu hút được một lượng lớn người dùng, trong đó có các công ty lớn như Microsoft, IBM, Sony, Siemens, Google và các nhóm nghiên cứu ở Stanford, MIT, CMU, Cambridge,... Nhiều forum hỗ trợ và cộng đồng người dùng đã được thành lập, tạo nên kênh thông tin rộng lớn, hữu ích cho việc tham khảo, tra cứu

### 1.3.3 Các thư viện dùng trong pycharm

```
tensorflow  
keras  
scikit-learn  
h5py  
matplotlib  
Pillow  
requests  
psutil
```

Mình cần quan tâm đến thư viện tensorflow

*Thư viện TensorFlow:* Với sự bùng nổ của lĩnh vực Trí Tuệ Nhân Tạo – A.I. trong thập kỷ vừa qua, machine learning và deep learning rõ ràng cũng phát triển theo cùng. Và ở thời điểm hiện tại, TensorFlow chính là thư viện mã nguồn mở cho machine learning nổi tiếng nhất thế giới, được phát triển bởi các nhà nghiên cứu từ Google. Việc hỗ trợ mạnh mẽ các phép toán học để tính toán trong machine learning và deep learning đã giúp việc tiếp cận các bài toán trở nên đơn giản, nhanh chóng và tiện lợi hơn nhiều.

Các hàm được dựng sẵn trong thư viện cho từng bài toán cho phép TensorFlow xây dựng được nhiều neural network. Nó còn cho phép bạn tính toán song song trên nhiều máy tính khác nhau, thậm chí trên nhiều CPU, GPU trong cùng 1 máy hay tạo ra các dataflow graph – đồ thị luồng dữ liệu để dựng nên các model. Nếu bạn muốn chọn con đường sự nghiệp trong lĩnh vực A.I. này, nắm rõ những điều cơ bản của TensorFlow thực sự rất quan trọng.

Được viết bằng C++ và thao tác interface bằng Python nên phần performance của TensorFlow cực kỳ tốt. Đối tượng sử dụng nó cũng đa dạng không kém: từ các nhà nghiên cứu, nhà khoa học dữ liệu và dĩ nhiên không thể thiếu các lập trình viên.

## CHƯƠNG 2 Giới thiệu thuật toán

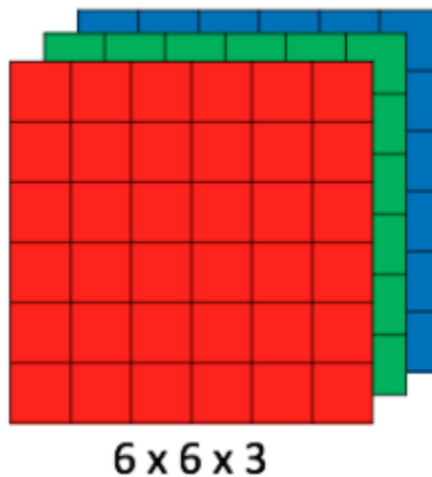
## 2.1 Giới thiệu cấu trúc mạng CNN

### 2.1.1 Giới thiệu

Convolutional Neural Network (CNN – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh. Để tìm hiểu tại sao thuật toán này được sử dụng rộng rãi cho việc nhận dạng (detection).

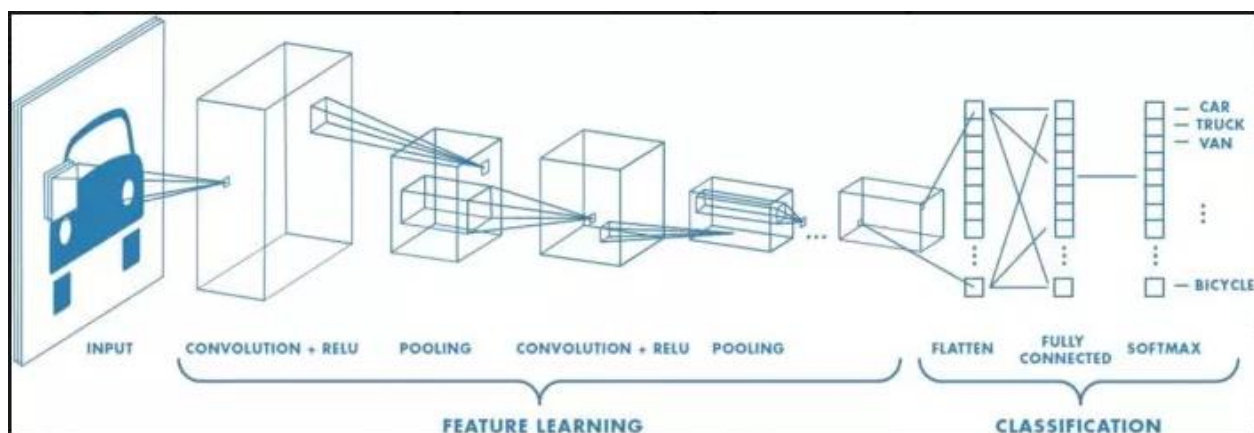
Trong mạng neural, mô hình mạng neural tích chập (CNN) là 1 trong những mô hình để nhận dạng và phân loại hình ảnh. Trong đó, xác định đối tượng và nhận dạng khuôn mặt là 1 trong số những lĩnh vực mà CNN được sử dụng rộng rãi.

CNN phân loại hình ảnh bằng cách lấy 1 hình ảnh đầu vào, xử lý và phân loại nó theo các hạng mục nhất định (Ví dụ: Chó, Mèo, Hổ, ...). Máy tính coi hình ảnh đầu vào là 1 mảng pixel và nó phụ thuộc vào độ phân giải của hình ảnh. Dựa trên độ phân giải hình ảnh, máy tính sẽ thấy  $H \times W \times D$  (H: Chiều cao, W: Chiều rộng, D: Độ dày). Ví dụ: Hình ảnh là mảng ma trận RGB  $6 \times 6 \times 3$  (3 ở đây là giá trị RGB).



Về kỹ thuật, mô hình CNN để training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernels), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất giữa 0 và 1. Hình dưới đây là toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.

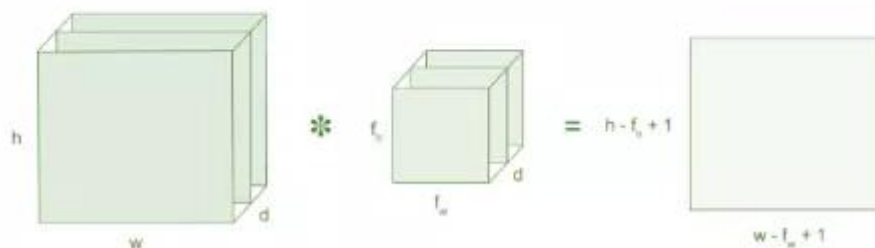




## 2.1.2 Lớp tích chập - Convolution Layer

Tích chập là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vuông nhỏ của dữ liệu đầu vào. Nó là 1 phép toán có 2 đầu vào như ma trận hình ảnh và 1 bộ lọc hoặc hạt nhân.

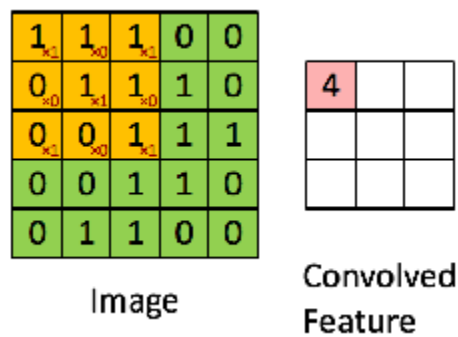
- An image matrix (volume) of dimension  **$(h \times w \times d)$**
- A filter  **$(f_h \times f_w \times d)$**
- Outputs a volume dimension  **$(h - f_h + 1) \times (w - f_w + 1) \times 1$**





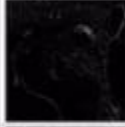




Ví dụ: Xem xét 1 ma trận  $5 \times 5$  có giá trị pixel là 0 và 1. Ma trận bộ lọc  $3 \times 3$  như hình bên dưới



Sau đó, lớp tích chập của ma trận hình ảnh 5 x 5 nhân với ma trận bộ lọc 3 x 3 gọi là 'Feature Map' như hình bên dưới.

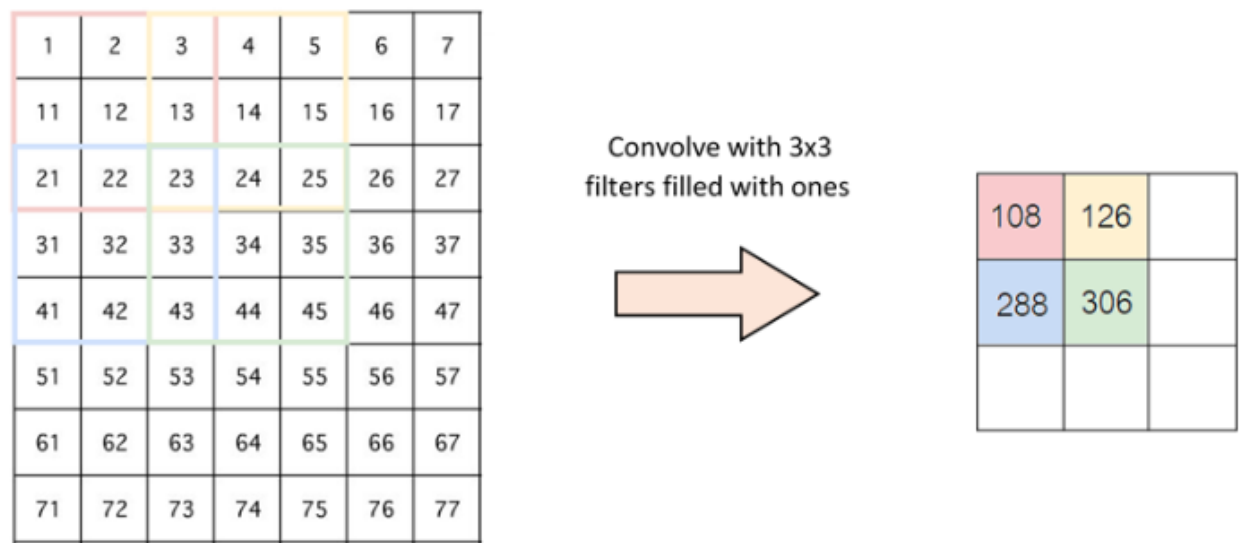


Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

### 2.1.3 Bước nhảy – Stride

Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride là 2.



### 2.1.4 Đường viền – Padding

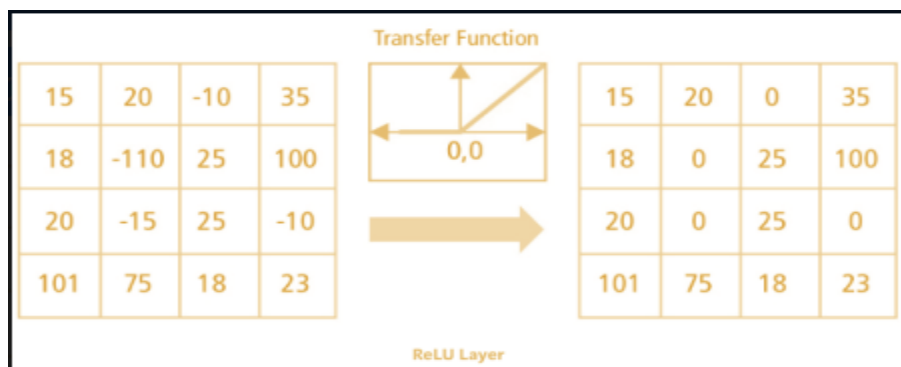
Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có hai lựa chọn:

1. Chèn thêm các số 0 vào 4 đường biên của hình ảnh ( padding)
2. Cắt bớt hình ảnh tại những điểm không phù hợp với kernel

### 2.1.5 Hàm phi tuyến – ReLU

ReLU viết tắt của Rectified Linear Unit, là 1 hàm phi tuyến. Với đầu ra là:  $f(x) = \max(0, x)$ .

Tại sao ReLU lại quan trọng: ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.



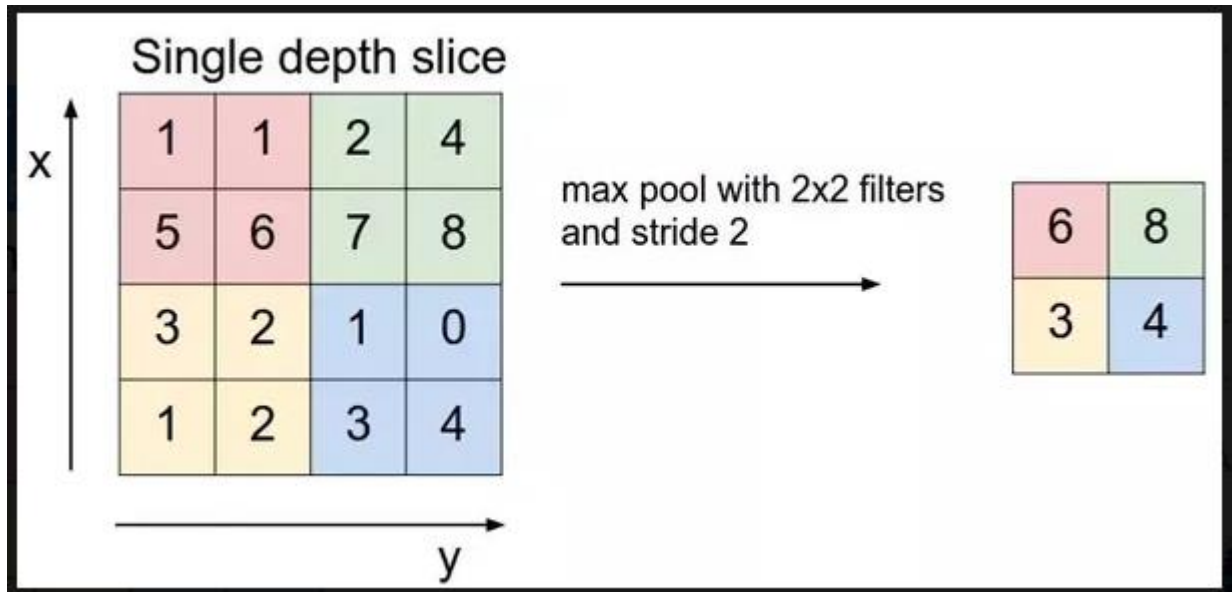
Có 1 số hàm phi tuyến khác như tanh, sigmoid cũng có thể được sử dụng thay cho ReLU. Hầu hết người ta thường dùng ReLU vì nó có hiệu suất tốt.

### 2.1.6 Lớp gộp - Pooling Layer

Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau:

1. Max Pooling
2. Average Pooling
3. Sum Pooling

Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình. Tổng tất cả các phần tử trong map gọi là sum pooling



### 2.1.7 Tóm tắt

1. Đầu vào của lớp tích chập là hình ảnh
2. Chọn đối số, áp dụng các bộ lọc với các bước nhảy, padding nếu cần. Thực hiện tích chập cho hình ảnh và áp dụng hàm kích hoạt ReLU cho ma trận hình ảnh.

3. Thực hiện Pooling để giảm kích thước cho hình ảnh.
4. Thêm nhiều lớp tích chập sao cho phù hợp
5. Xây dựng đầu ra và dữ liệu đầu vào thành 1 lớp được kết nối đầy đủ (Full Connected)
6. Sử dụng hàm kích hoạt để tìm đối số phù hợp và phân loại hình ảnh.

## 2.2 Tổng quan về MTCNN

Phần đầu tiên sẽ là về Face Detection, một bài toán với nhiệm vụ phát hiện các khuôn mặt có trong ảnh hoặc frame trong Video. Mạng MTCNN sẽ giúp chúng ta điều này với 3 lớp mạng khác biệt, tượng trưng cho 3 stage chính là P-Net, R-Net và O-Net (PRO!)

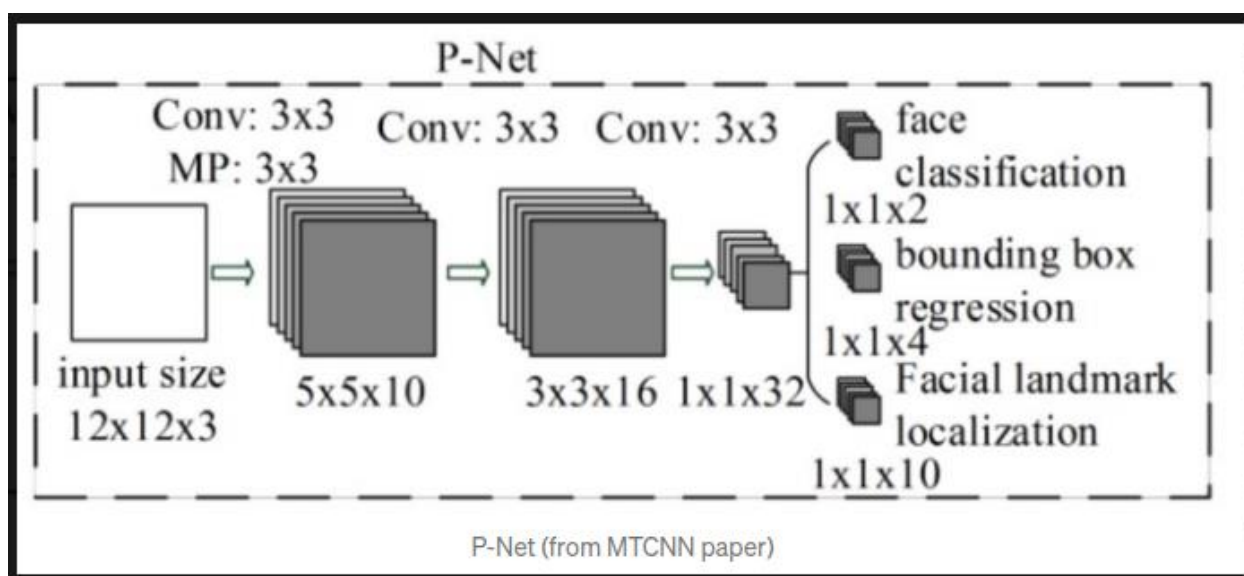
### 2.2.1 Stage 1: P-Net

Trước hết, một bức ảnh thường sẽ có nhiều hơn một người - một khuôn mặt. Ngoài ra, những khuôn mặt thường sẽ có kích thước khác nhau. Ta cần một phương thức để có thể nhận dạng toàn bộ số khuôn mặt đó, ở các kích thước khác nhau. MTCNN đưa cho chúng ta một giải pháp, bằng cách sử dụng phép Resize ảnh, để tạo một loạt các bản copy từ ảnh gốc với kích cỡ khác nhau, từ to đến nhỏ, tạo thành 1 Image Pyramid.

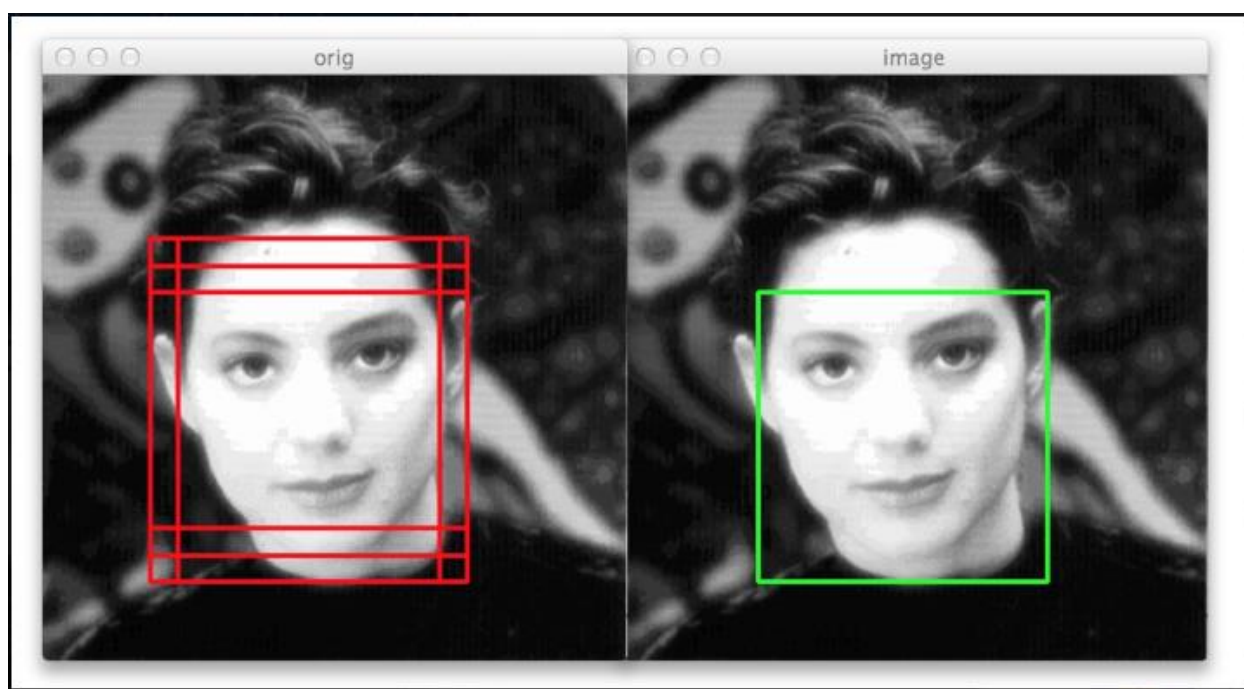


**Image pyramid**

Với mỗi một phiên bản copy-resize của ảnh gốc, ta sử dụng kernel  $12 \times 12$  pixel và stride  $= 2$  để đi qua toàn bộ bức ảnh, dò tìm khuôn mặt. Vì các bản copies của ảnh gốc có kích thước khác nhau, cho nên mạng có thể dễ dàng nhận biết được các khuôn mặt với kích thước khác nhau, mặc dù chỉ dùng 1 kernel với kích thước cố định (Ảnh to hơn, mặt to hơn; Ảnh nhỏ hơn, mặt nhỏ hơn). Sau đó, ta sẽ đưa những kernels được cắt ra từ trên và truyền qua mạng P-Net (Proposal Network). Kết quả của mạng cho ra một loạt các bounding boxes nằm trong mỗi kernel, mỗi bounding boxes sẽ chứa tọa độ 4 góc để xác định vị trí trong kernel chứa nó (đã được normalize về khoảng từ  $(0,1)$ ) và điểm confident (Điểm tự tin) tương ứng.



Để loại trừ bớt các bounding boxes trên các bức ảnh và các kernels, ta sử dụng 2 phương pháp chính là lập mức Threshold confident - nhằm xóa đi các box có mức confident thấp và sử dụng NMS (Non-Maximum Suppression) để xóa các box có tỷ lệ trùng nhau (Intersection Over Union) vượt qua 1 mức threshold tự đặt nào đó. Hình ảnh dưới đây là minh họa cho phép NMS, những box bị trùng nhau sẽ bị loại bỏ và giữ lại 1 box có mức confident cao nhất.

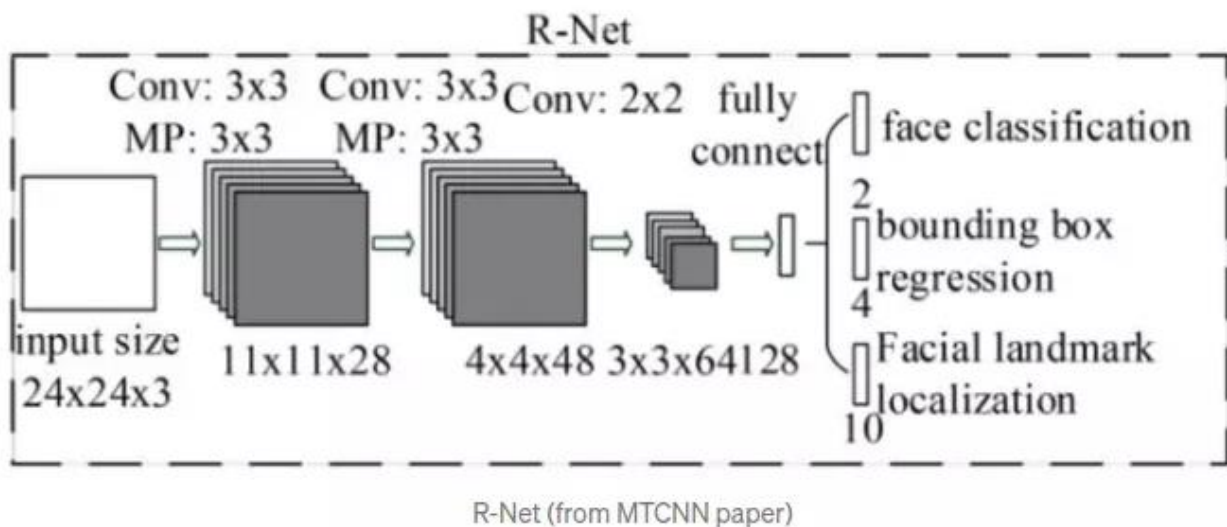


Sau khi đã xóa bớt các box không hợp lý, ta sẽ chuyển các tọa độ của các box về với tọa độ gốc của bức ảnh thật. Do tọa độ của box đã được normalize về khoảng  $(0,1)$  tương ứng như kernel, cho nên công việc lúc này chỉ là tính toán độ dài và rộng của kernel dựa theo ảnh gốc, sau đó nhân tọa độ đã được normalize của box với kích thước của kernel và cộng với



tọa độ của các góc kernel tương ứng. Kết quả của quá trình trên sẽ là những tọa độ của box tương ứng ở trên ảnh kích thước ban đầu. Cuối cùng, ta sẽ resize lại các box về dạng hình vuông, lấy tọa độ mới của các box và feed vào mạng tiếp theo, mạng R.

### 2.2.2 Stage 2: R-Net



Mạng R (Refine Network) thực hiện các bước như mạng P. Tuy nhiên, mạng còn sử dụng một phương pháp tên là padding, nhằm thực hiện việc chèn thêm các zero-pixels vào các phần thiếu của bounding box nếu bounding box bị vượt quá biên của ảnh. Tất cả các bounding box lúc này sẽ được resize về kích thước 24x24, được coi như 1 kernel và feed vào mạng R. Kết quả sau cũng là những tọa độ mới của các box còn lại và được đưa vào mạng tiếp theo, mạng O.

### 2.2.3 Stage 3: O-Net

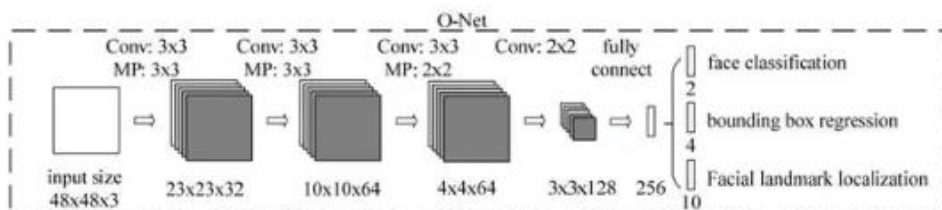


Fig. 2. The architectures of P-Net, R-Net, and O-Net, where "MP" means max pooling and "Conv" means convolution. The step size in convolution and pooling is 1 and 2, respectively.

Cuối cùng là mạng O (Output Network), mạng cũng thực hiện tương tự như việc trong mạng R, thay đổi kích thước thành 48x48. Tuy nhiên, kết quả đầu ra của mạng lúc này không còn chỉ là các tọa độ của các box nữa, mà trả về 3 giá trị bao gồm: 4 tọa độ của bounding box (out[0]), tọa độ 5 điểm landmark trên mặt, bao gồm 2 mắt, 1 mũi, 2 bên cánh môi (out[1]) và điểm confident của mỗi box (out[2]). Tất cả sẽ được lưu vào thành 1 dictionary với 3 keys kể trên.



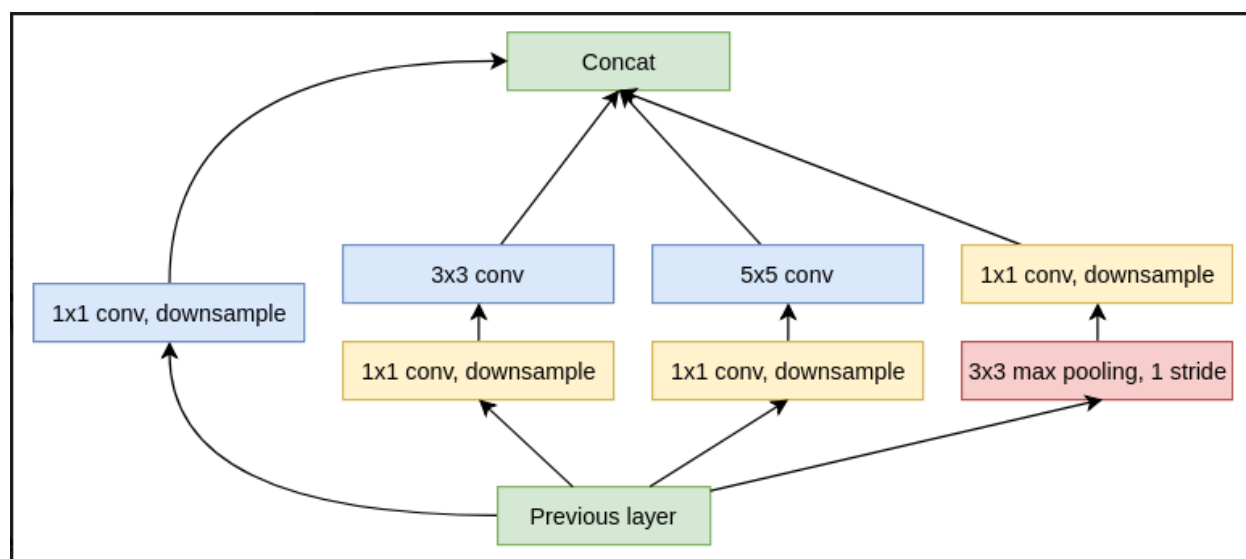
## 2.3 Tổng quan về FaceNet

Phần tiếp theo là về Face Verification. Nhiệm vụ chính của bài toán này là đánh giá xem ảnh mặt hiện tại có đúng với thông tin, mặt của một người khác đã có trong hệ thống không.

### 2.3.1 Các khái niệm cơ bản

*Embedding Vector*: Là một vector với dimension cố định (thường có chiều nhỏ hơn các Feature Vector bình thường), đã được học trong quá trình train và đại diện cho một tập các feature có trách nhiệm trong việc phân loại các đối tượng trong chiều không gian đã được biến đổi. Embedding rất hữu dụng trong việc tìm các Nearest Neighbor trong 1 Cluster cho sẵn, dựa theo khoảng cách-mối quan hệ giữa các embedding với nhau.

*Inception V1*: Một cấu trúc mạng CNN được giới thiệu vào năm 2014 của Google, với đặc trưng là các khối Inception. Khối này cho phép mạng được học theo cấu trúc song song, nghĩa là với 1 đầu vào có thể được đưa vào nhiều các lớp Convolution khác nhau để đưa ra các kết quả khác nhau, sau đó sẽ được Concatenate vào thành 1 output. Việc học song song này giúp mạng có thể học được nhiều chi tiết hơn, lấy được nhiều feature hơn so với mạng CNN truyền thống. Ngoài ra, mạng cũng áp dụng các khối Convolution 1x1 nhằm giảm kích thước của mạng, khiến việc train trở nên nhanh hơn.



*Triplet Loss* : Triplet Loss giúp học một embedding space (không gian nhúng) trong đó các khuôn mặt của cùng một người được gom vào gần nhau hơn và các khuôn mặt của các người khác được đẩy xa nhau hơn.

Triplet Loss sử dụng ba hình ảnh: hình ảnh anchor, hình ảnh positive và hình ảnh negative. Hình ảnh anchor là hình ảnh của một người bất kỳ, hình ảnh positive là hình ảnh của cùng một người với hình ảnh anchor và hình ảnh negative là hình ảnh của một người khác.

Mục tiêu của Triplet Loss là tối thiểu hóa khoảng cách giữa hình ảnh anchor và hình ảnh positive trong khi đồng thời tối đa hóa khoảng cách giữa hình ảnh anchor và hình ảnh negative. Khoảng cách này được tính toán bằng cách sử dụng hàm Euclidean distance hoặc cosine similarity. Quá trình này giúp mô hình học được các đặc trưng của khuôn mặt và tạo ra một embedding space tốt hơn để phân biệt các khuôn mặt của các người khác nhau.

## CHƯƠNG 3 Quy trình thực hiện và kết quả

Trong đồ án 1 chúng em chỉ inference lại model đã được pretrain, không xây dựng và train model.

### 3.1 Tạo môi trường ảo bằng Anaconda và cài các thư viện hỗ trợ

Việc cài đặt và chạy trên môi trường ảo Anaconda để tránh các vấn đề xung đột với các thư viện có sẵn trong máy. Sau khi kích hoạt được môi trường ảo tiến hành cài các thư viện hỗ trợ bằng pip install

```
PS C:\Users\PC\Desktop\doan> conda activate doan1
PS C:\Users\PC\Desktop\doan> pip install -r c:\Users\PC\Desktop\doan\FaceNet-Infer\requirements.txt
```

requirements.txt - Notepad

File Edit Format View Help

```
torch==1.8.0
torchvision==0.9.0
numpy==1.19.2
opencv-python==4.5.1.48
facenet_pytorch==2.5.2
```

### 3.2 Face Detection với MTCNN

#### 3.2.1 Dectec Face bằng OpenCV và MTCNN

Gọi ra 1 object từ class MTCNN đi kèm một số cấu hình:

```
mtcnn = MTCNN(thresholds= [0.7, 0.7, 0.8] ,keep_all=True, device = device)
```

Thresholds chính là mức thresholds cho 3 lớp mạng P, R và O. Mặc định sẽ là [0.6, 0.7, 0.7] nhưng để tăng độ chính xác nên truyền vào 3 mức cao hơn như trên. Keep\_all để xác định việc chúng ta có detect và trả về tất cả các mặt có thể trong bức hình hay không, mình muốn nên để giá trị True.

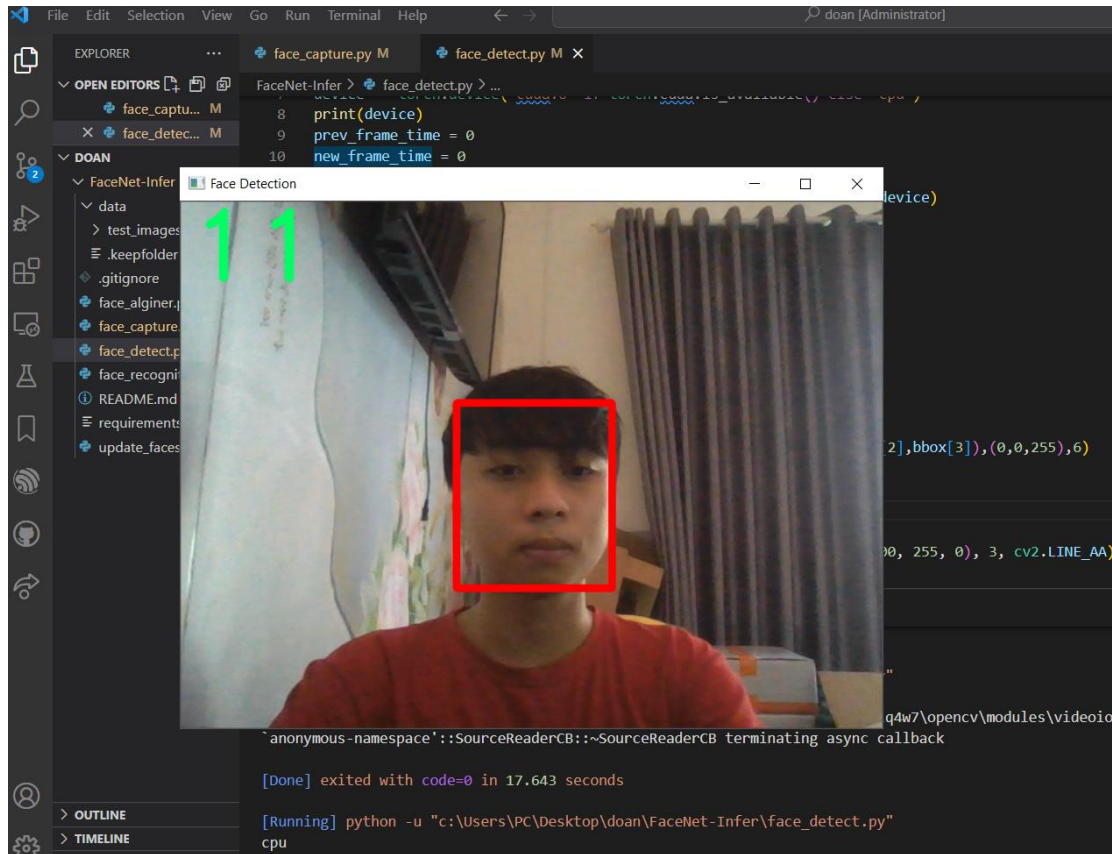
Sử dụng hàm cv2.VideoCapture() của OpenCV để gọi webcam và ghi lại từng frame ảnh. Set cho kích thước của webcam về theo kích thước mong muốn 640x480.

```
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH,640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
while cap.isOpened():
    isSuccess, frame = cap.read()
    if isSuccess:
        boxes, _, points_list = mtcnn.detect(frame, landmarks=True)
        if boxes is not None:
            for box in boxes:
                bbox = list(map(int,box.tolist()))
                cv2.rectangle(frame,(bbox[0],bbox[1]),(bbox[2],bbox[3]),(0,0,255),6)
            new_frame_time = time.time()
            fps = 1/(new_frame_time-prev_frame_time)
            prev_frame_time = new_frame_time
            fps = str(int(fps))
            cv2.putText(frame, fps, (7, 70), cv2.FONT_HERSHEY_DUPLEX, 3, (100, 255, 0),
3, cv2.LINE_AA)
            cv2.imshow('Face Detection', frame)
            if cv2.waitKey(1)&0xFF == 27:
                break
    cap.release()
cv2.destroyAllWindows()
```

2 câu lệnh while và if trên dùng để khẳng định rằng sẽ thu được 1 frame từ camera, tránh gây ra lỗi khi thực hiện detect phía dưới. Class MTCNN cung cấp cho ta một hàm tên detect, giúp trả về một list các boxes hình chữ nhật, mỗi box bao gồm 2 tọa độ tương ứng 2 góc của box(bbox[0], bbox[1] là cặp tọa độ (x,y) của điểm góc trái trên của hình chữ nhật, góc phải

dưới tương tự). Khi đó, công việc của chúng ta chỉ còn là kiểm tra xem có box nào được trả về không và dùng lệnh `cv2.rectangle` để tạo ra một hình hộp chữ nhật với 2 góc đã cho ở trên.

### Kết quả:



Nhận xét: Trong môi trường ánh sáng thuận lợi mô hình hoạt động tốt, trong trường hợp bị ngược sáng hoặc ánh sáng yếu hiệu suất nhận diện khuôn mặt giảm không còn phát hiện được. Khi đó ta cần sử dụng các kỹ thuật xử lý ảnh như bộ lọc Homomorphic để giảm độ sáng và khử nhiễu cho các frame ảnh.

### 3.2.2 Capture Face

Sau khi đã detect được khuôn mặt ta cần tạo folder ảnh khuôn mặt được cắt sát để sử dụng train cho mạng facenet để phân biệt khuôn mặt đó.

Source code:

```
import cv2
from facenet_pytorch import MTCNN
import torch
```

```

from datetime import datetime
import os

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)

IMG_PATH = r'C:\Users\PC\Desktop\doan\FaceNet-Infer\data\test_images'
count = 50
usr_name = "tin"
USR_PATH = os.path.join(IMG_PATH, usr_name)
leap = 1

mtcnn = MTCNN(margin = 20, keep_all=False, select_largest = True,
post_process=False, device = device)

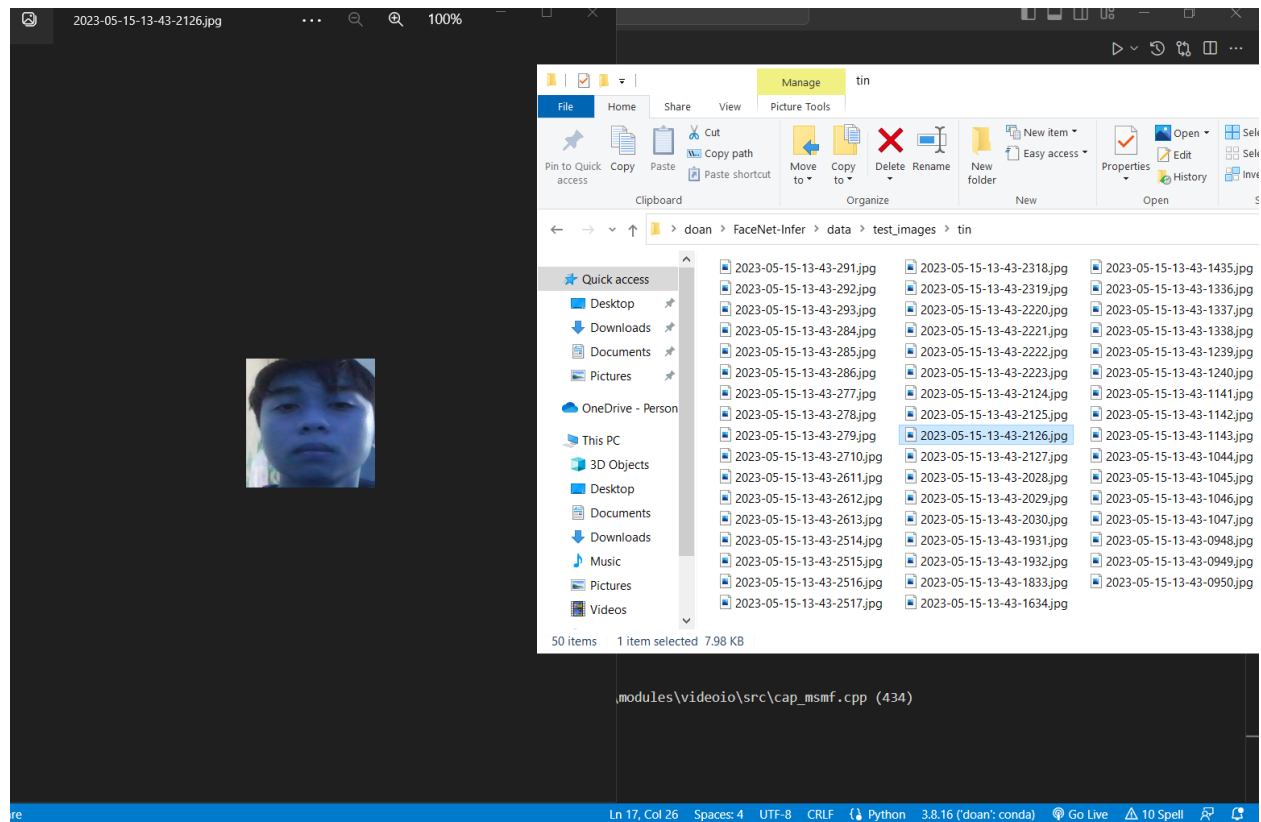
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH,640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
while cap.isOpened() and count:
    isSuccess, frame = cap.read()
    if mtcnn(frame) is not None and leap%2:
        path = str(USR_PATH+'/{}.jpg'.format(str(datetime.now())[:-7].replace(":", "-").replace(" ", "-")+str(count)))
        face_img = mtcnn(frame, save_path = path)
        count-=1
    leap+=1
    cv2.imshow('Face Capturing', frame)
    if cv2.waitKey(1)&0xFF == 27:
        break
cap.release()
cv2.destroyAllWindows()

```

Phần code của Capture sẽ không khác nhiều so với phần Detect, chỉ thêm vào một số biến như count - dùng để đếm số lượng ảnh; leap - bước nhảy, tức máy sẽ lấy ảnh sau mỗi leap frame.

Tổng cộng sẽ có 50 ảnh được lưu vào máy theo đường dẫn như IMG\_PATH. Tên của file ảnh sẽ theo định dạng "năm-tháng-ngày-giờ-phút-giây-số thứ tự ảnh.jpg".

## Kết quả:



## 3.3 Cập nhật FaceList và Face Recognition với FaceNet

### 3.3.1 Cập nhật FaceList:

Khai báo thư viện và khai báo các biến toàn cục

```
import glob
import torch
from torchvision import transforms
from facenet_pytorch import InceptionResnetV1, fixed_image_standardization
import os
```

```

from PIL import Image
import numpy as np

IMG_PATH = r'C:\Users\PC\Desktop\doan\FaceNet-Infer\data\test_images'
DATA_PATH = r'C:\Users\PC\Desktop\doan\FaceNet-Infer\data'

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)

```

Tiến hành chuẩn hóa dữ liệu (normalize): Khi dữ liệu được chuẩn hóa về 1 khoảng cố định, thuật toán tối ưu Gradient Descent sẽ đưa ra được kết quả converge nhanh chóng hơn và tránh bị các vấn đề liên quan tới Vanishing/Exploding Gradient. Hàm sau là một hàm normalize đơn giản, giúp đưa pixel ảnh về trong khoảng  $[-1,1]$  mà vẫn giữ được distribution gốc của nó.

```

def trans(img):
    transform = transforms.Compose([
        transforms.ToTensor(),
        fixed_image_standardization
    ])
    return transform(img)

```

hàm fixed\_image\_standardization():

```

def fixed_image_standardization(image_tensor):
    processed_tensor = (image_tensor - 127.5) / 128.0
    return processed_tensor

model.eval()

```

Khai báo model sẽ sử dụng, FaceNet sử dụng cấu trúc mạng InceptionV1 và đầu ra là một Feature Vector 128 chiều tuy nhiên do cấu trúc mạng InceptionV1 đã cũ, tỷ lệ chính xác đã bị các mạng bây giờ vượt xa cũng như tốc độ tính toán của máy tính đã có sự cải thiện đáng kể so với năm 2015, tác giả của repository trên đã sử dụng kiến trúc mạng mới là InceptionResnetV1 với đầu ra là một Feature Vector 512 chiều. Mạng được pretrained trên tập dữ liệu CASIA-WebFace.

```

model = InceptionResnetV1(
    classify=False,
    pretrained="casia-webface"
).to(device)

```

Tham số `classify` được gán bằng `False`, nhằm đảm bảo đầu ra sẽ là 1 Feature Vector chứ không phải kết quả đã được classify theo tập pretrained. Lệnh `model.eval()` để khai báo cho PyTorch rằng mình đang evaluation, không phải training

```

for usr in os.listdir(IMG_PATH):
    embeds = []
    for file in glob.glob(os.path.join(IMG_PATH, usr)+'/*.jpg'):
        # print(usr)
        try:
            img = Image.open(file)
        except:
            continue
        with torch.no_grad():
            # print('smt')
            embeds.append(model(trans(img).to(device).unsqueeze(0))) #1 anh, kích
thuoc [1,512]
    if len(embeds) == 0:
        continue
    embedding = torch.cat(embeds).mean(0, keepdim=True) #dua ra trung binh cua
30 anh, kích thuoc [1,512]
    embeddings.append(embedding) # 1 cai list n cai [1,512]
    # print(embedding)
    names.append(usr)

embeddings = torch.cat(embeddings) #[n,512]
names = np.array(names)

```

*user* là tên của các folder trong `IMG_PATH`, đồng nghĩa là tên của *user*. Khi đó, list `names` sẽ là danh sách tên của các user trong folder. Với mỗi folder (user), ta xét toàn bộ ảnh, mỗi ảnh sẽ dùng model đã được load pretrained trên để sinh ra một embed với kích thước



[1,512]. Quay trở lại lúc này, chúng ta đã lấy tổng cộng là 50 ảnh, vậy ta sẽ có tổng cộng là 50 embeds cho 1 cá nhân. Tuy nhiên, ta chỉ cần 1 embeds, đại diện cho 1 cá nhân. Vì vậy, ta sẽ sử dụng hàm `torch.cat()` để đưa list về 1 tensor 2 chiều và sử dụng hàm `torch.mean()` để lấy giá trị trung bình cho toàn bộ embeds. Kết quả cuối cùng sẽ được thêm vào list embeddings, đại diện cho tập hợp các embedding của các user, ứng với đó là 1 giá trị name của user với cùng index trên tập names.

Sau cùng, ta sẽ một lần nữa concatenate list embeddings lại dưới dạng tensor và lưu thành một file .pth, còn tập hợp tên user sẽ được lưu dưới dạng .npy

```
embeddings = torch.cat(embeddings) #[n,512]
names = np.array(names)

if device == 'cpu':
    torch.save(embeddings, DATA_PATH+"/faceslistCPU.pth")
else:
    torch.save(embeddings, DATA_PATH+"/faceslist.pth")
```

Kết quả:

	18/05/2023 15:47	PTH File	3 KB
faceslist.pth	18/05/2023 15:47	PTH File	3 KB
usernames.npy	18/05/2023 15:47	NPY File	1 KB

### 3.3.2 FaceRecognition dựa theo FaceList

Lấy frame từ camera, lấy tọa độ các box khuôn mặt và vẽ ô chữ nhật:

```
embeddings, names = load_faceslist()
while cap.isOpened():
    isSuccess, frame = cap.read()
    if isSuccess:
        boxes, _ = mtcnn.detect(frame)
        if boxes is not None:
            for box in boxes:
                bbox = list(map(int, box.tolist()))
```

```

        face = extract_face(bbox, frame)
        idx, score = inference(model, face, embeddings)
        if idx != -1:
            frame = cv2.rectangle(frame, (bbox[0],bbox[1]),
(bbox[2],bbox[3]), (0,0,255), 6)
            score = torch.Tensor.cpu(score[0]).detach().numpy()*power
            frame = cv2.putText(frame, names[idx] +
'_{:.2f}'.format(score), (bbox[0],bbox[1]), cv2.FONT_HERSHEY_DUPLEX, 2, (0,255,0),
2, cv2.LINE_8)
        else:
            frame = cv2.rectangle(frame, (bbox[0],bbox[1]),
(bbox[2],bbox[3]), (0,0,255), 6)
            frame = cv2.putText(frame, 'Unknown', (bbox[0],bbox[1]),
cv2.FONT_HERSHEY_DUPLEX, 2, (0,255,0), 2, cv2.LINE_8)

        new_frame_time = time.time()
        fps = 1/(new_frame_time-prev_frame_time)
        prev_frame_time = new_frame_time
        fps = str(int(fps))
        cv2.putText(frame, fps, (7, 70), cv2.FONT_HERSHEY_DUPLEX, 3, (100,
255, 0), 3, cv2.LINE_AA)

        cv2.imshow('Face Recognition', frame)
        if cv2.waitKey(1)&0xFF == 27:
            break

```

load lại FaceList và usernames từ 2 file lưu ở trên bằng hàm load\_facelist(). Hàm này sẽ trả về 2 tập là embeddings và usernames:

```

def load_faceslist():
    if device == 'cpu':
        embeds = torch.load(DATA_PATH+'/faceslistCPU.pth')
    else:
        embeds = torch.load(DATA_PATH+'/faceslist.pth')

```

```
names = np.load(DATA_PATH+'usernames.npy')  
return embeds, names
```

Tạo hàm `extract_face(bbox, frame)` để trích xuất khuôn mặt từ những tọa độ box nhận được

```
def extract_face(box, img, margin=20):  
    face_size = 160  
    img_size = frame_size  
    margin = [  
        margin * (box[2] - box[0]) / (face_size - margin),  
        margin * (box[3] - box[1]) / (face_size - margin),  
    ] #tạo margin bao quanh box cũ  
    box = [  
        int(max(box[0] - margin[0] / 2, 0)),  
        int(max(box[1] - margin[1] / 2, 0)),  
        int(min(box[2] + margin[0] / 2, img_size[0])),  
        int(min(box[3] + margin[1] / 2, img_size[1])),  
    ]  
    img = img[box[1]:box[3], box[0]:box[2]]  
    face = cv2.resize(img, (face_size, face_size), interpolation=cv2.INTER_AREA)  
    face = Image.fromarray(face)  
    return face
```

`frame_size` chính là kích thước frame đầu vào lấy từ webcam (640x480) `margin` sẽ có giá trị = 20 để tương ứng như `margin` khi capturing face.

$\text{margin} * (\text{box}[2] - \text{box}[0]) / (\text{face\_size} - \text{margin})$ , đặt  $\text{box}[2] - \text{box}[0]$  là chiều rộng của bounding box, ta giả sử rằng chiều rộng đó bằng đúng  $\text{img\_size} - \text{margin}$ , tức 140. Khi đó,  $\text{margin}[0]$  mới sẽ trở thành  $20 * 140 / 140 = 20$ , tức là giữ nguyên  $\text{margin} = 20$  do chiều rộng sau khi cộng với `margin` mới sẽ có kích thước =  $140 + 20 = 160$ , tức bằng chính `img_size`. Ngược lại, nếu chiều rộng nhỏ hơn, vào khoảng 130, khi đấy ta sẽ cần `margin` một khoảng nhỏ hơn 20, vào khoảng 18,5px.

hàm inference để kết xuất embedding cho từng ảnh mặt - thứ đã được extract từ hàm extract\_face() ở trên. Đầu vào có một thuộc tính là threshold, thuộc tính này sẽ quyết định việc mạng có lấy mặt hay không.

```
def inference(model, face, local_embeds, threshold = 3):
    #local: [n,512] voi n la so nguoi trong faceslist
    embeds = []
    # print(trans(face).unsqueeze(0).shape)
    embeds.append(model(trans(face).to(device).unsqueeze(0)))
    detect_embeds = torch.cat(embeds) #[1,512]
    # print(detect_embeds.shape)
    # [1,512,1] [1,512,n]
    norm_diff = detect_embeds.unsqueeze(-1) - torch.transpose(local_embeds, 0,
1).unsqueeze(0)
    # print(norm_diff)
    norm_score = torch.sum(torch.pow(norm_diff, 2), dim=1) #(1,n), moi cot la
tong khoang cach euclide so vs embed moi

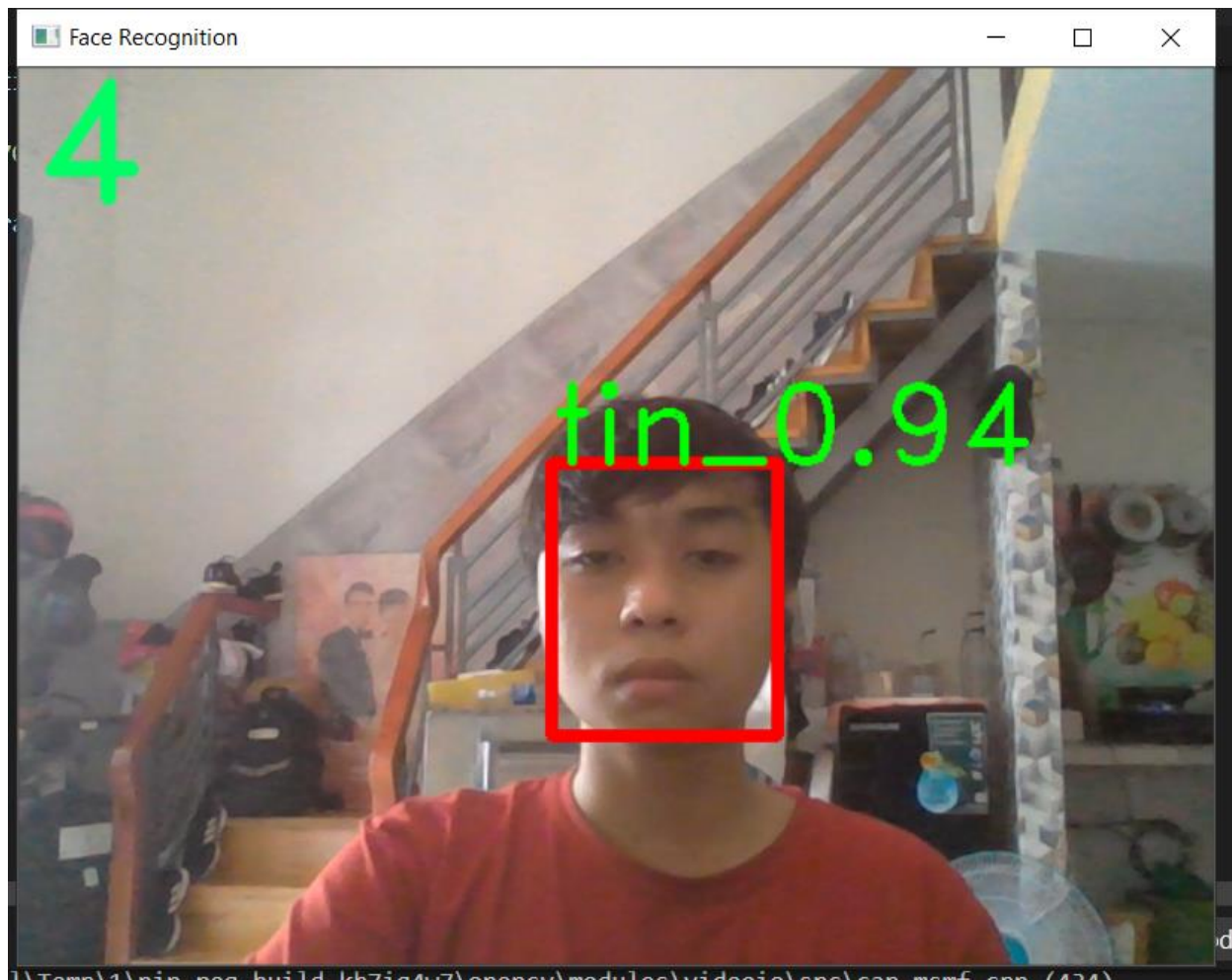
    min_dist, embed_idx = torch.min(norm_score, dim = 1)
    print(min_dist*power, names[embed_idx])
    # print(min_dist.shape)
    if min_dist*power > threshold:
        return -1, -1
    else:
        return embed_idx, min_dist.double()
```

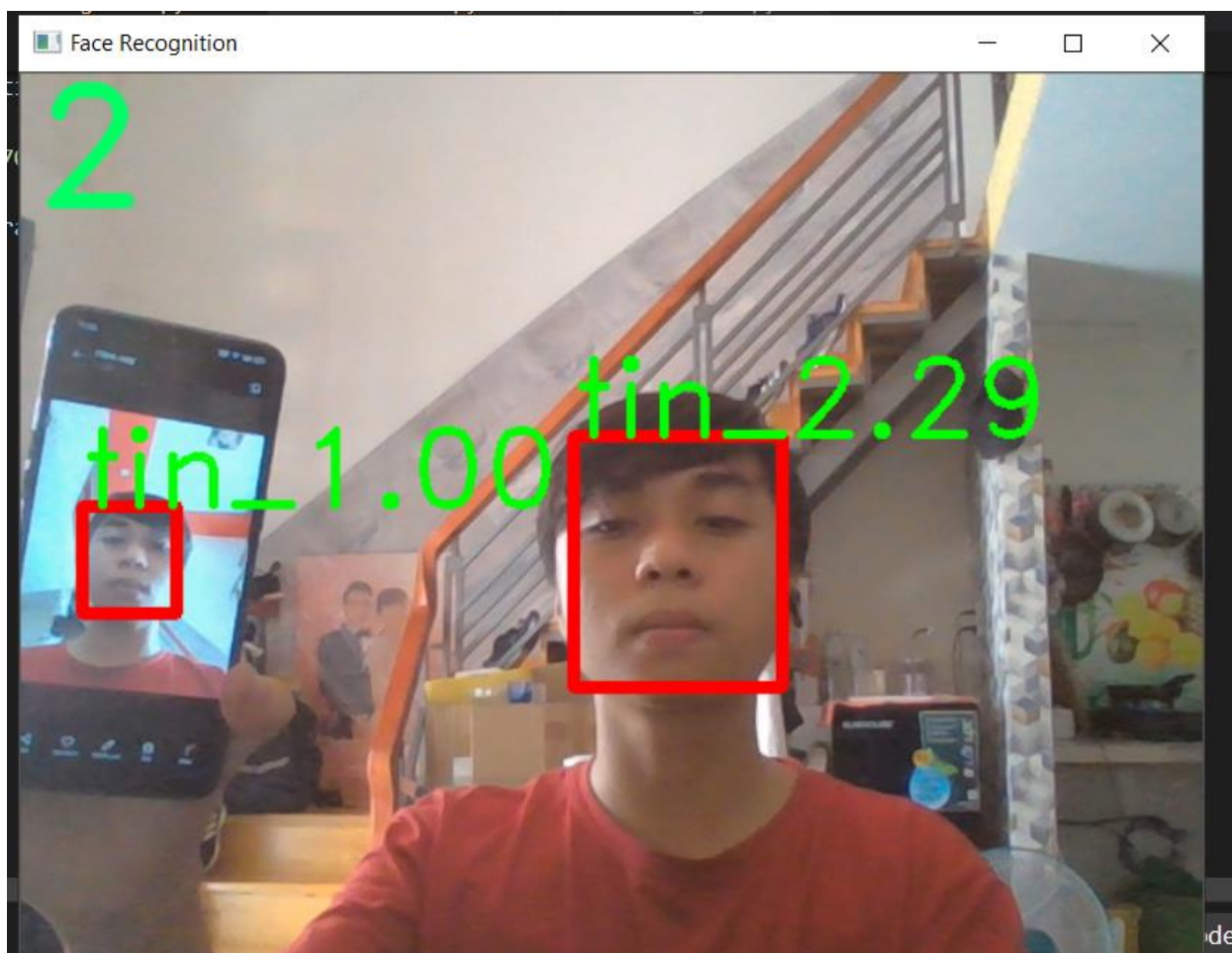
Phần kết xuất embedding sẽ tương tự như trong phần Update Facelist. Tuy nhiên, ta không lưu lại vào FaceList mà sẽ tiến hành so sánh khoảng cách giữa embedding vừa nhận được với các embeddings khác có trong FaceList. Câu lệnh norm\_diff sẽ tính toán ra một ma trận với kích thước [1,512,n], là khoảng cách (hiệu) của từng chiều trong 512 chiều của embedding nhận được với các embeddings trong FaceList. Sau đó, norm\_score sẽ là tổng bình phương (vì chúng ta quan tâm khoảng cách, không quan tâm giá trị là âm hay dương) các khoảng cách của n tập hiệu khoảng cách trên.

Kết quả cuối cùng là một dãy các score, mỗi cột là score điểm khác biệt giữa face nhận được với faces trong FaceList. Điểm khác biệt càng nhỏ, độ tương đồng giữa 2 face càng lớn. Vì vậy chúng ta sẽ trả về index của khuôn mặt trong FaceList có score nhỏ nhất so với face cần inference. Đặt thêm 1 mức threshold - nếu min\_score > threshold thì ta có thể đánh giá

rằng khuôn mặt này chưa có trong FaceList. Khi đó, giá trị index trả về sẽ là -1, tức sẽ không nhận dạng khi gặp giá trị này và trả về box Unknown. Sử dụng thêm `cv2.putText()` ở rìa ngoài của box.

### 3.4 Kết quả





## CHƯƠNG 4 Kết luận và hướng phát triển

### 4.1 Kết luận

Mặc dù kết quả thu được là khá khả quan, tuy nhiên vẫn còn nhiều hạn chế:

Phần Classifier dựa chủ yếu vào việc tính khoảng cách giữa các embeddings (khá giống thuật toán k-NN với  $k = 1$ ) cho nên kết quả chưa thực sự quá tốt. Để khắc phục, chúng ta có thể xây dựng một đầu Classifier đơn giản mà mạnh mẽ hơn như SVM hoặc một mạng FCN nhỏ và áp dụng phương pháp Online Training.

Khi gặp những điều kiện bất lợi về độ sáng (đặc biệt là trời tối, thiếu ánh sáng) và các yếu tố ngoại cảnh, độ chính xác của mạng sẽ giảm khá đáng kể. Ta có thể tăng mức threshold để chấp nhận score cao, hoặc thêm dữ liệu ở các điều kiện đó.

Sử dụng một bức hình từ điện thoại và mạng vẫn có thể nhận diện được. Kết quả như vậy là do MTCNN chưa có khả năng nhận diện vật thể sống (Liveness Detection).

## 4.2 Hướng Phát triển

Trong thời gian tới, tụi em hướng tới khắc phục những hạn chế mình gặp phải trong đồ án 1 lần này. Cụ thể là tạo GUI với Python để dễ dàng hơn trong việc điểm danh, tìm hiểu về hệ quản trị cơ sở dữ liệu (SQL)

Khắc phục được hạn chế đó là đưa điện thoại vào vẫn nhận diện được khuôn mặt của mình trong đó.

So sánh độ hiệu quả giữa các mô hình mang lại ( tụi em chưa tìm hiểu được nhiều mô hình )

Kết hợp với xe tự hành phát hiện ngủ gật trong lúc lái xe...

### Tài liệu tham khảo

Tài liệu tham khảo:

<https://www.mathworks.com/discovery/convolutional-neural-network.html>

<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

<https://blog.datawow.io/interns-explain-cnn-8a669d053f8b>

<https://arxiv.org/pdf/1503.03832.pdf>

<https://arxiv.org/ftp/arxiv/papers/1604/1604.02878.pdf>

<https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/>

<https://viblo.asia/p/nhan-dien-khuon-mat-voi-mang-mtcnn-va-facenet-phan-2-bJzKmrVXZ9N?fbclid=IwAR05F1zMDxpu9HEEaSKqh3717g8b2VDZWVUnWCdMz4yBcwZbb3pevK2Ohfo>

