**LỜI CẢM ƠN**

Sau quá trình tìm hiểu, nghiên cứu và thực hiện đề tài khoá luận tốt nghiệp và hoàn thành đúng tiến độ, chúng em xin chân thành gửi lời cảm ơn sâu sắc nhất đến quý thầy cô, gia đình và những người anh, người em, người bạn đã giúp đỡ cho chúng em.

Chúng em xin gửi lời cảm ơn đến thầy…, giảng viên trường Đại Học Sư Phạm Kỹ Thuật Tp.HCM đã hướng dẫn và tạo điều kiện để nhóm có thể hoàn thành được đề tài của mình.

Một lần nữa chúng em xin cảm ơn tất cả mọi người đã luôn giúp đỡ, hỗ trợ tận tình cho chúng em để chúng em hoàn thành tốt đề tài này.

Xin chân thành cảm ơn!

Người thực hiện đề tài

(Ký tên)

**MỤC LỤC**

**DANH MỤC HÌNH**

**CÁC TỪ VIẾT TẮT**

**TÓM TẮT**

Trong đề tài này, ở lĩnh vực nhận diện khuôn mặt chúng tôi sẽ tập trung vào tìm hiểu cấu trúc, điều chỉnh thông số, và tìm ra các giải pháp tối ưu mô hình nhận diện khuôn mặt đã được huấn luyện sẵn thay vì xây dựng và huấn luyện mọi thứ từ đầu. Bằng việc sử dụng các mô hình MTCNN và FaceNet, chúng tôi sẽ tạo ra một hệ thống nhận diện khuôn mặt trong video.

Sau khi có một hệ thống nhận diện khuôn mặt hoàn chỉnh, chúng tôi sẽ tiến hành tìm ra phương pháp tối ưu để giảm bớt độ nặng của mô hình nhằm triển khai hệ thống này lên phần cứng, cụ thể chúng tôi đã chọn cho mình một Jetson Nano

…

**Chương 1: Tổng quan**

* 1. **Giới thiệu**

Sự phát triển về công nghệ đã kéo theo sự phát triển trong nhiều lĩnh vực hiện này, đặc biệt thế giới hiện nay càng chú trọng vào sự phát triển của “Trí tuệ nhân tạo” để giải quyết nhiều vấn đề một cách tự động mà không cần tới công sức của con người hơn hết là cải thiện rất nhiều hiệu suất. Một số lĩnh vực ta có thể kể như là nhận dạng chữ viết, nhận dạng khuôn mặt, máy dịch,… Chính vì thế cuộc sống con người ngày càng cải thiện rất nhiều.

Nhận dạng khuôn mặt là vấn đề được quan tâm nhất vì được sử dụng rộng rãi trong các hệ thống an ninh. Với sự đam mê và thích thú về lĩnh vực “Thị giác máy tính” (một lĩnh vực nằm trong “Trí tuệ nhân tạo”), nhóm chúng tôi đã chọn cho mình một mô hình nhận dạng khuôn mặt và triển khai lên một cánh cửa tự động với mục đích tìm hiểu sâu về lĩnh vực này và tìm ra phương pháp triển khai lên các phần cứng cụ thể.

Ở đề tài này chúng tôi sẽ ứng dụng cấu trúc MTCNN kết hợp FaceNet để xây dựng một hệ thống nhận diện khuôn mặt và triển khai chúng lên phần cứng, cụ thể là Jetson Nano. Thông qua đề tài này chúng tôi sẽ có những cốt lõi về nền tảng để tương lai có thể tiến hành ứng dụng các lĩnh vực khác như hệ thống chấm công, nhận diện nhân viên,…

* 1. **Đối tượng nghiên cứu và phạm vi nghiên cứu**

Đối tượng nghiên cứu: Chúng tôi sẽ nghiên cứu thuật toán MTCNN với FaceNet, tìm hiểu cấu trúc của mô hình và sử dụng pretrained model để thiết kế và xây dựng hệ thống nhận diện, tìm hiểu về “Network Optimization” để có thể triển khai mô hình lên hệ thống có giới hạn về phần cứng để có thể tăng FPS và giữ độ chính xác, và khi hệ thống nhận diện đã xong, chúng tôi sẽ tiếp tục tìm hiểu cụ thể về Jetson Nano, cách cài đặt hệ điều hành và các thư viện cần thiết sau đó triển khai hệ thống lên phần cứng. Ngoài Jetson Nano ra, chúng tôi sẽ tiến hành nghiên cứu thêm một số ngoại vi thích hợp để có thể tạo ra một “Cánh cửa tự động” hoàn chỉnh.

Phạm vi nghiên cứu: Thiết kế hệ thống nhận diện, mô phỏng, thực thi trên máy vi tính cá nhân. Sau đó triển khai và kiểm thử hiệu suất mô hình lên Jetson Nano. Sử dụng những phương pháp, thư viện hỗ trợ, kết hợp, bổ trợ cho nhau để hoàn thành hệ thống. Tham khao một số bài báo về các lĩnh vực liên quan tới vấn đề.

* 1. **Mục tiêu và giới hạn của đề tài**
     1. **Mục tiêu**

- Tìm hiểu các lí thuyết về mô hình MTCNN và FaceNet.

- Liên kết MTCNN và FaceNet tạo một hệ thống phát hiện và nhận diện gương mặt hoàn chỉnh thông qua việc sử dụng Pytorch framework.

- Xây dựng và ứng dụng để thu thập thông tin người dùng để phục vụ model và mô phỏng quy trình hoạt động hệ thống.

- Tìm hiểu về phần cứng Jetson Nano.

- Cài đặt hệ điều hành, thư viện cần thiết và triển khai hệ thống nhận diện lên phần cứng Jetson Nano.

- Kiểm chứng độ hiệu quả mô hình và tiến hành tối ưu mô hình.

* + 1. **Giới hạn**
  1. **Phương pháp nghiên cứu**

-

* 1. **Bố cục báo cáo**

Chương 1: Tổng quan

Chương 2:

Chương 3:

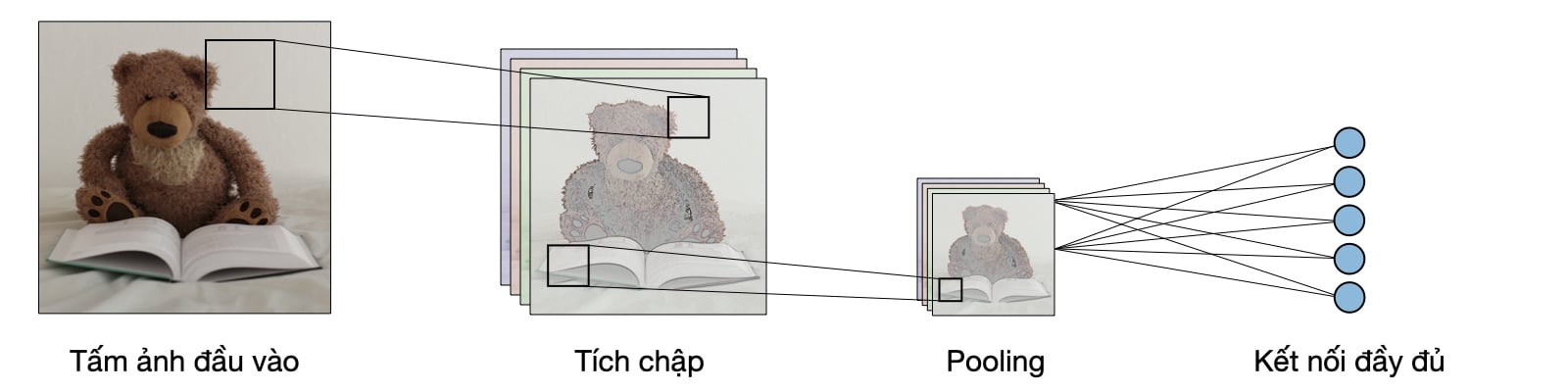
Chương 4

**Chương 2: Cơ sở lý thuyết**

**2.1. Khái niệm về CNN**

Mô hình mạng CNN là một trong mô hình có vai trò cực kì chủ chốt trong nhiều lĩnh vực liên quan nhận dạng mẫu; từ xử lí ảnh tới nhận diện giọng nói.

Đầu vào của mạng CNN sẽ được chuyển thành một loạt các lớp tích chập với các bộ lọc đi kèm (còn được gọi là kernels hoặc filter) và các lớp pooling để trích xuất các đặc trưng đối tượng, sau đó sẽ cho qua các lớp Fully Connected để cho ra các đầu ra. Tầng tích chập và tầng pooling có thể được hiệu chỉnh theo các siêu tham số (hyperparameters).



Hình 2.1. Kiến trúc truyền thống của CNN

**2.1.1. Convolutional Layer**

Đây là lớp quan trọng cũng là lớp đầu tiên trong kiến trúc CNN. Lớp này sẽ có nhiệm vụ phát hiện các đặc trưng có tính không gian hiệu quả. Những yếu tố quan trọng của convolutional layer là filter map (hay còn gọi là kernel), stride, padding, feature map.

Tầng này sẽ sử dụng các filter map để thực hiện phép toán tích chập khi đưa chúng đi qua các đầu vào. Những filter map này thường sẽ là một ma trận 3x3 hoặc 5x5.

Stride là khoảng cách giữa 2 kernel khi quét. Giả sử nếu stride là 1 thì kernel sẽ quét 2 ô ngay cạnh, với stride bằng 2 thì sẽ quét ô 1 với ô 3 và bỏ qua ô giữa. Việc thay đổi stride sẽ ảnh hưởng đến việc tính toán nhanh hay chậm, độ chính xác lớn hay thấp… tuỳ vào bài toán, nghiên cứu khác nhau thì stride sẽ được chọn khác nhau.

Padding là thêm các giá trị 0 vào đường biên của hình ảnh đầu vào nhằm phù hợp với kernel để tính toán.

Feature map là một ma trận thể hiện kết quả của mỗi lần filter map quét qua các đầu vào. Mỗi lần quét là một quá trình tính toán tích chập ma trận. Đây là đặc trưng của từng ảnh.

Chúng ta có thể kể đến hàm kích hoạt cũng có vai trò không nhỏ đến kết quả của mô hình. Hiện nay hai hàm kích hoạt phổ biến nhất là ReLU và Softmax. Có thể tham khảo chi tiết và các hàm kích hoạt khác ở [1]

**2.1.2. Pooling Layer**

Vì những đầu vào của chúng ta là những ma trận ảnh, kích cỡ không hề nhỏ, từ đó các thông số quá nhiều, khiến cho việc tính toán của mô hình trở nên bị ảnh hưởng. Pooling layer sẽ đảm nhiệm vai trò giảm các thông số của mô hình lại mà không làm ảnh hưởng tới kết quả mô hình. Từ đó việc triển khai mô hình với tốc độ khả thi hơn. Hiện nay có 2 loại chủ yếu là Max Pooling và Average Pooling.

**2.1.3. Fully Connected Layer**

Lớp này có nhiệm vụ đưa ra kết quả sau khi Convolutional Layer và Pooling Layer đã xử lý, rút trích được đặc trưng.

Nó sẽ tổng hợp những đặc trưng lại, kết nối và phân loại chúng thành những output theo yêu cầu bài toán đưa ra. Thường lớp sẽ có cấu trúc của một ANN [2].

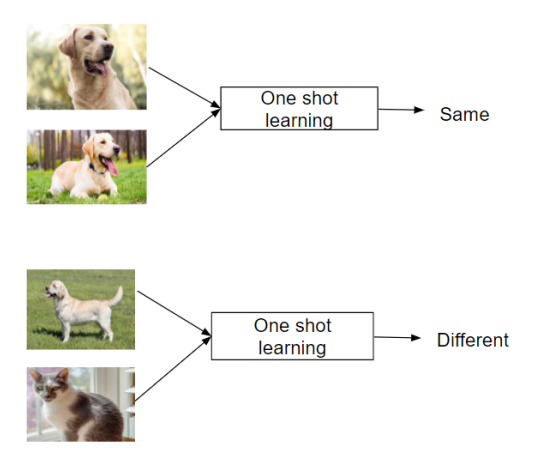
**2.2. Các thuật toán nhận dạng khuôn mặt**

2.2.1. Giới thiệu về One-shot Learning

One-shot Learning là một phương pháp phân loại mà mô hình sẽ dự đoán class trên một bức ảnh chỉ bằng một hay vài lần huấn luyện mẫu mỗi class. Con người rất giỏi trong One-shot Learning. Ta có thể lấy một ví dụ về một đứa bé có thể phân loại đâu là chó, đâu là mèo chỉ bằng một vài lần nhìn bức ảnh. Vậy nên chúng ta sẽ triển khai một hoạt động tương tự lên máy tính để có nhận dạng được đối tượng.

Phương pháp này sử dụng một model đặc trưng để huấn luyện phân loại đối tượng, thường sẽ là KNN hay SVM [3].

One-shot Learning cực kì hữu dụng trong các lĩnh vực như xác minh chữ kí và nhận dạng khuôn mặt, khi mà chúng ta có rất nhiều class mà chỉ có vài mẫu. Vấn đề bất cập ở phương pháp này đó là khi chúng ta có một class mới, chúng ta sẽ cần phải huấn luyện lại toàn bộ, việc này sẽ mất thời gian nếu số lượng dữ liệu cần huấn luyện lớn.



Hình 2.2.

2.2.2. Siamese Neural Network (SNN)

Là một kiến trúc neural có hai hoặc nhiều mạng con giống nhau (cùng một cấu hình và cùng thông số với trọng số). Việc cập nhật thông số trên kiến trúc này sẽ được phản ánh đồng thời ở các mạng con của nó.

Một số ứng dụng kể đến: Face Verification, Signature Verification… và FaceNet đã sử dụng kiến trúc này để mã hoá đặc trưng các class thành các vector.

Ưu điểm của kiến trúc này

* Cần rất ít Sample để huấn luyện SNN mà không cần lo lắng về Image Imbalance
* Có thể linh hoạt kết hợp các cấu trúc khác để tối ưu hoá kết quả hơn.

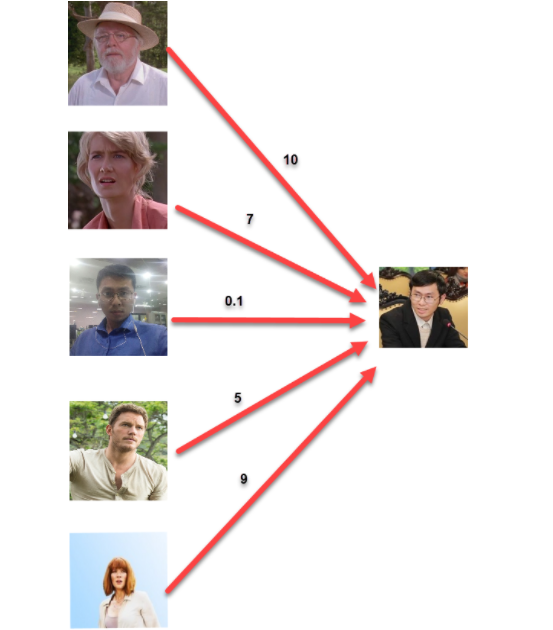
Bù lại vẫn có những khuyết điểm như

* Thời gian huấn luyện lâu hơn do phải cập nhật từng cặp đôi một với nhau, khiến cho việc đáp ứng các bài toán thời gian thực không khả thi nhiều.
* Chỉ đưa ra sự tương đồng giữa 2 vector chứ không thể hiện xác suất dự đoán mỗi lớp Output như phương pháp khác.

2.2.3. Similarity Learning

Tương tự với các bài toán cluster, hiểu đơn giản thì hai ảnh càng giống nhau (cùng một người), thì khoảng cách chúng càng bé và ngược lại thì khoảng cách xa hơn.

Thường ở bài toán này ta sẽ định nghĩa một ngưỡng Threshold. Ta sẽ so sánh input với tất cả các ảnh trong dataset, nếu khoảng cách nằm dưới ngưỡng cho phép thì ta sẽ lấy đó làm output.



Hình 2.3. Phương pháp Similarity Learning

So với One-shot learning ở trên, phương pháp này có điểm nổi trội hơn là không phụ thuộc vào số lượng class, cũng như không cần phải huấn luyện lại mỗi khi có một class mới.

Và ở thuật toán này nó sẽ đi kèm với SNN dùng để embed các ảnh thành 1 vector. Thông thường là ta sẽ embed 1 ảnh xuống còn 1 vector 512 hoặc 128 chiều, vừa đủ nhỏ để tính toán lại không làm mất thông tin của hình ảnh.

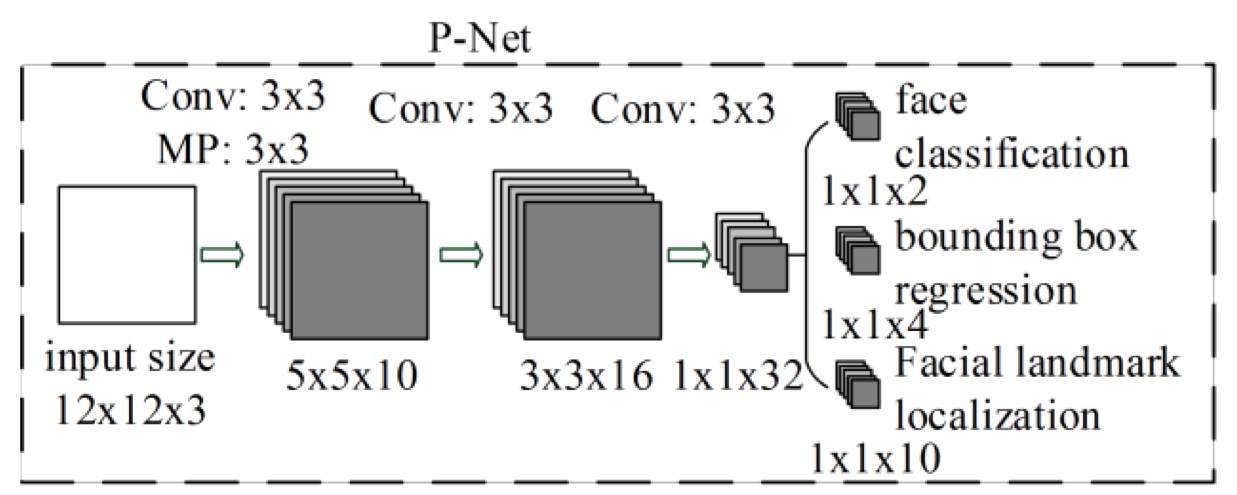
**2.3. MTCNN**

2.3.1. Giới thiệu về MTCNN

Khi các phương pháp Deep learning, cụ thể là các kiến trúc Convolutional Neural Network (CNN) hay You Only Look Once (YOLO), đã phát triển vượt bậc và dùng rộng rãi trong các lĩnh vực nhận dạng đối tượng, thậm chí còn giải quyết được nhiều vấn đề khó khăn trong phân loại ảnh. Nhưng chúng lại không được thiết kế cụ thể cho bài toán phân loại ảnh, chưa kể lại đòi hỏi rất nhiều tài nguyên, tính toán nhiều và phức tạp. Vào năm 2016, “Multi – task Cascaded Convolutional Networks” [3], hay viết tắt là MTCNN, ra đời nhằm giải quyết đồng thời hai bài toán face detection and face alignment và thậm chí giải quyết bài toán real-time. Cấu trúc chính của MTCNN bao gồm 3 lớp CNN (P-Net, R-Net, O-net).

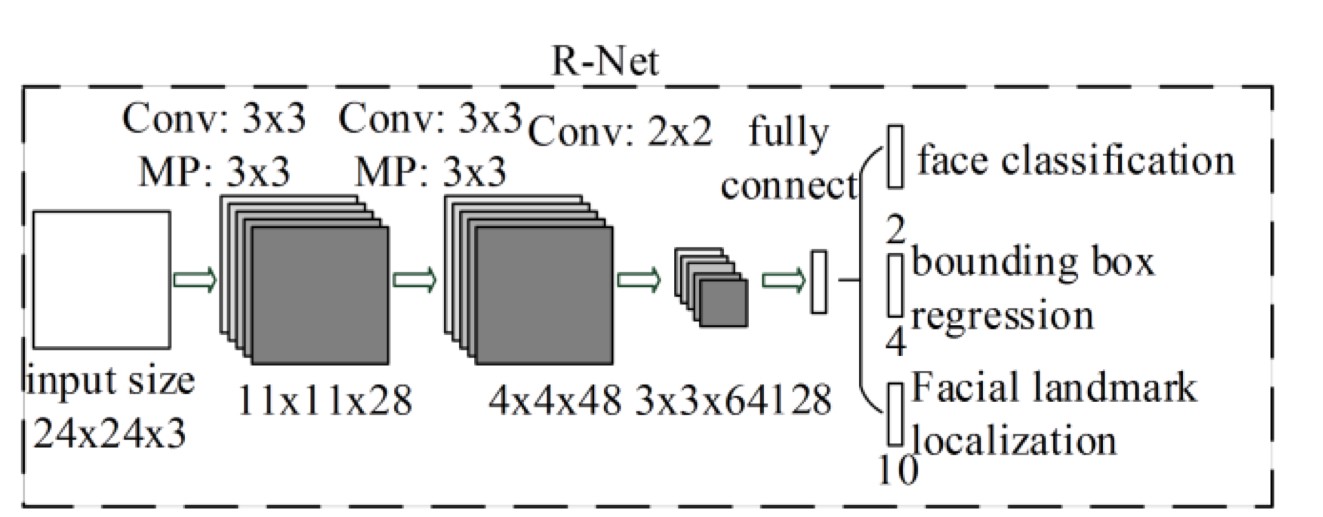
2.3.2. Cấu trúc mạng MTCNN

Lớp đầu tiên là The Proposal Network (P-Net), là một fully convolutional network (FCN) dùng để trích xuất ra khung hình khuôn mặt ứng viên và lấy ra các vector hồi quy bounding box (bounding box regression vectors). Sau đó các ứng viên sẽ được hiệu chỉnh dựa trên bounding box. Tiếp đó là sử dụng phương pháp NMS (non-maximum suppression) để hợp nhất các ứng viên có khả năng trùng lặp cao.



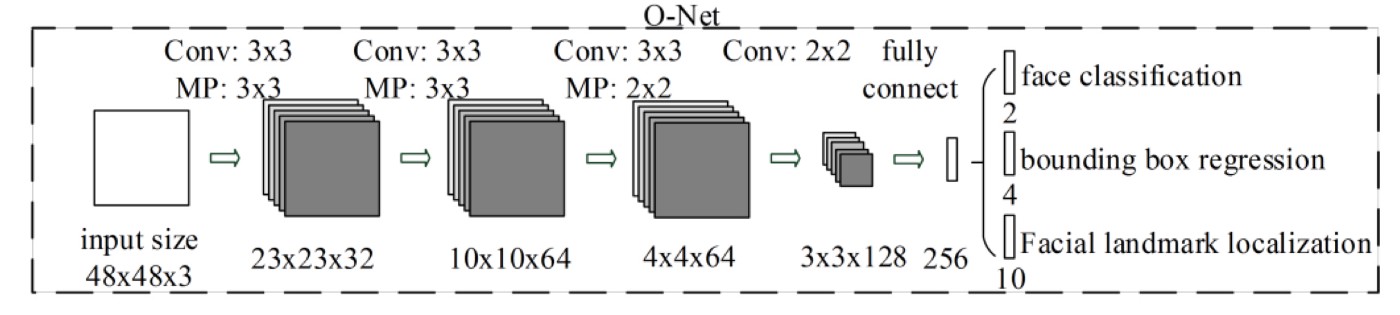
Hình 2.4

Lớp tiếp theo là The Refine Network (R-Net) là một mạng lưới CNN. Bằng cách đưa các ứng cử viên ở lớp P-Net, mạng này sẽ tiếp tục lọc một lượng lớn các ứng viên có tỷ lệ sai lệch cao. Tương tự như phương pháp P-Net sẽ dùng NMS và bounding box.

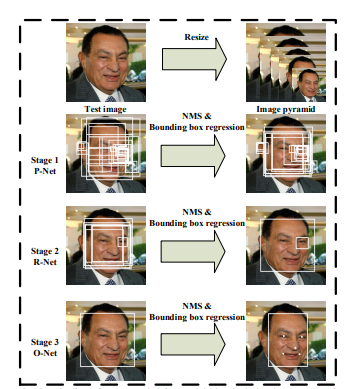


Hình 2.5

Lớp cuối cùng là The Output Network (O-net) có kiến trúc tương tự như lớp R-Net, nhưng lớp này sẽ có mục tiêu mô tả khuôn mặt chi tiết hơn và cho ra 5 điểm trên gương mặt để xác định mắt, mũi, miệng trên gương mặt.



Hình 2.6



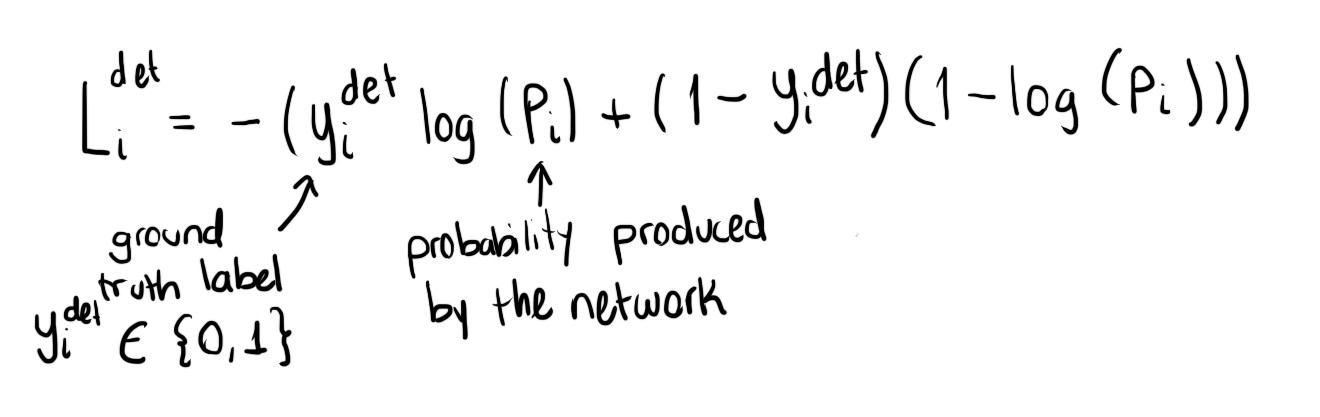
Hình 2.7 Tổng thể cách thức hoạt động của MTCNN

2.3.3. Quá trình huấn luyện

Theo bài báo [4], tại mục II.C, tác giả chia ra 3 nhiệm vụ chính để huấn luyện mô hình đó là: Face classification, Bounding box regression và Facial landmark localization

1. Face classification

Mục tiêu của nhiệm vụ này là bài toán phân loại 2 lớp, tác giả đã sử dụng Entropy Loss Function với mỗi x­i:

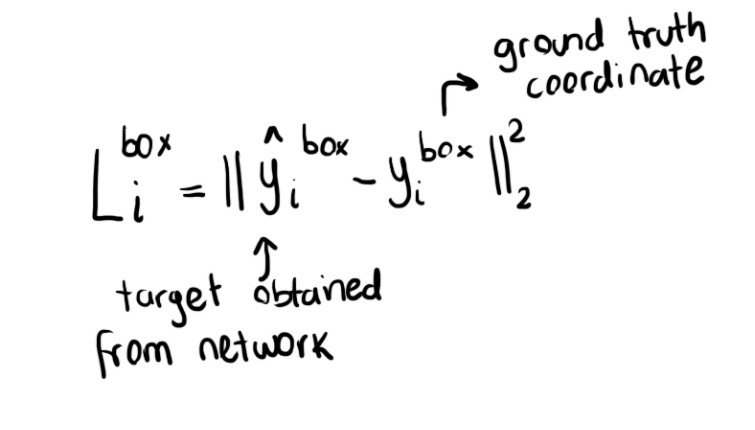


Hình 2.8. Cross-entropy loss

pi là xác suất được tạo bởi mạng cho biết mẫu xi có phải là face hay không. {0,1} biểu thị the ground-truth label.

1. Bounding box regression

Tác giả sử dụng Euclidean Loss cho mỗi mẫu ứng viên xi để dự đoán sự chênh lệch giữa nó và “ground-truth” gần nhất:

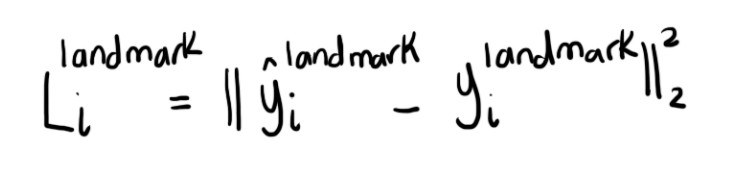


Hình 2.9 Euclidean loss

Trong đó là mục tiêu hồi quy thu được từ mạng và

là tọa độ “ground-truth” của mẫu thứ i. Box sẽ có 4 thông số bao gồm: trái, trên, chiều dài và chiều cao.

1. Facial landmark localization

Giống như Bounding box regression tác giả cũng sử dụng Euclidean Loss: 

Hình 2.10. Euclidean loss

Trong đó: là tọa độ landmark thu được từ mạng và là tọa độ “ground-truth” của mẫu thứ I. Sẽ có 5 landmark được trả về bao gồm: mắt trái, mắt phải, mũi, khóe miệng trái, khóe miệng phải.

**2.4. Khái quát FaceNet**

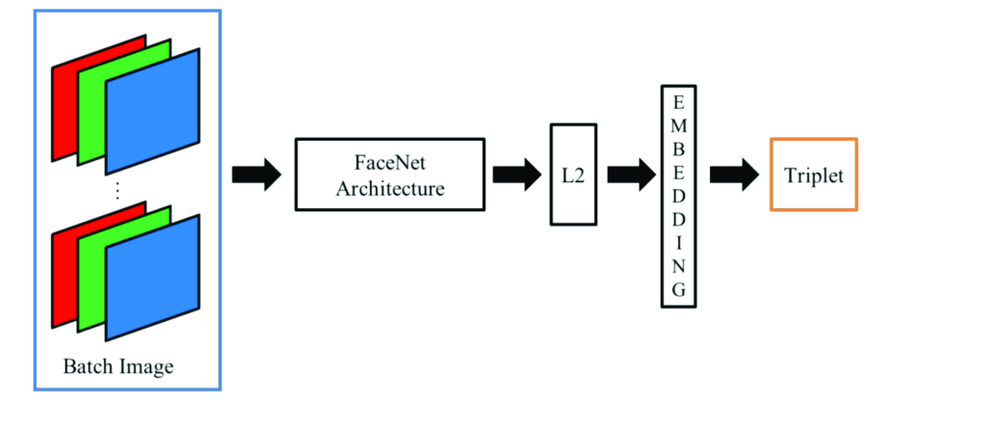
2.4.1. Giới thiệu FaceNet

FaceNet được giới thiệu vào năm 2016 với nhóm tác giả Florian Schroff, Dmitry Kalenichenko và James Phibin. Sự ra đời của FaceNet đã cải thiện rất nhiều trong lĩnh vực bài toán nhận dạng gương mặt và các bài toán liên quan đến phân loại.

FaceNet chính là một dạng SNN, có tác dụng biểu diễn các bức ảnh thành một không gian Euclidean n chiều (thường là 128) sao cho khoảng cách giữa các vector embedding càng nhỏ, mức tương đồng giữa chúng càng lớn.

2.4.2. Cấu trúc FaceNet

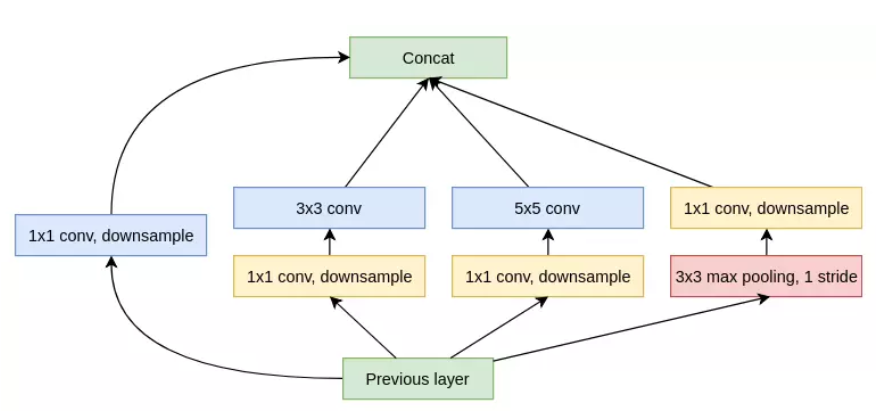
Dựa vào bài báo [5], cấu trúc FaceNet được mô tả như sau:



Hình 2.11. Cấu trúc FaceNet

Đầu vào là tập dữ liệu của đối tượng cần nhận dạng, tập này sẽ đi qua một kiến trúc mà FaceNet sử dụng (dùng nhiều nhất là Inception V1 hay còn được gọi là Google Net). Sau đó dữ liệu này sẽ được chuẩn hoá theo chuẩn Norm L2, quá trình này sẽ cho ra 1 vector gồm 128 điểm được gọi là embedding. Output này sẽ được đưa qua Triplet, một dạng Loss Function trong kiến trúc FaceNet để huấn luyện.

2.4.3. Sơ lược về kiến trúc Inception V1 (Google Net)



Hình 2.12. Cấu trúc mạng Inception V1 [6]

Inception V1 là một cấu trúc mạng CNN được Google giới thiệu vào năm 2014. Mạng này cho phép input được huấn luyện dưới dạng kiến trúc song song, nghĩa là input được đưa vào kiến trúc mạng CNN khác nhau để đưa ra kết quả khác nhau, sau đó sẽ gộp lại và tạo ra output. Việc học song song như này cho phép đạt được hiệu quả nhiều hơn, giúp trích xuất các đối tượng đặc trưng so với CNN thông thường.

2.4.4. Triple Loss

**a. Khái niệm**

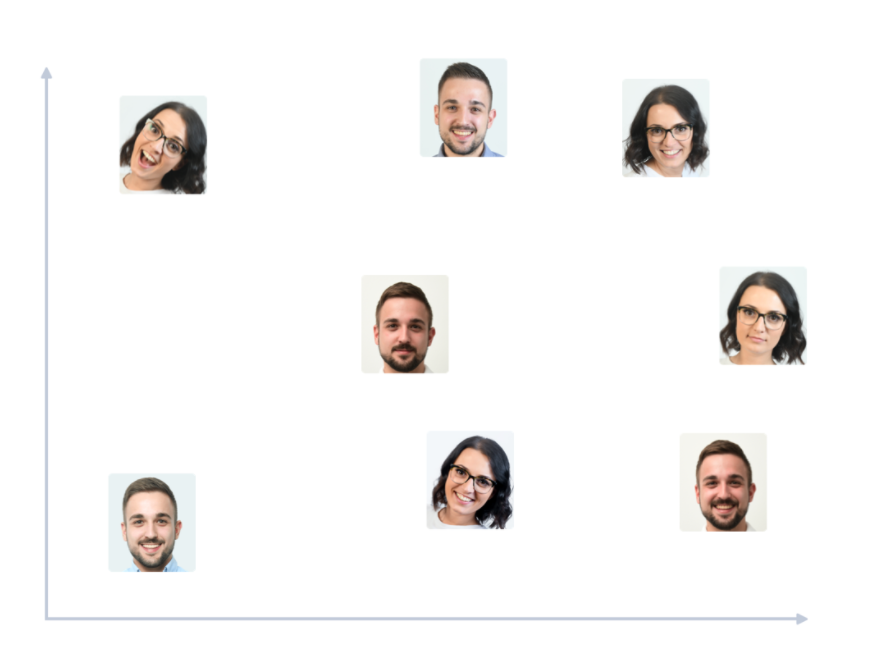
Sau khi có được embedding vector 128 điểm, đây chính là các vector chứa những thông tin quan trọng về các tấm ảnh, cụ thể ở đây là những khía cạnh của gương mặt. Những vectors này sẽ thành đầu vào cho hàm loss function để tính toán khoảng cách giữa các vector.

Thuật toán Triple Loss yêu cầu cần 3 tấm ảnh trong đó có 1 tấm ảnh được gọi là anchor. Anchor image sẽ được chọn ngẫu nhiên trong quá trình học. 2 tấm ảnh tiếp theo sẽ là Positive và Negative Image, positive là ảnh giống với ảnh Anchor và negative là ảnh khác với ảnh Anchor.

**b. Cách chọn Image Input và công thức hàm loss**

Quá trình triển khai thuật toán Triple Loss được diễn ra như sau:

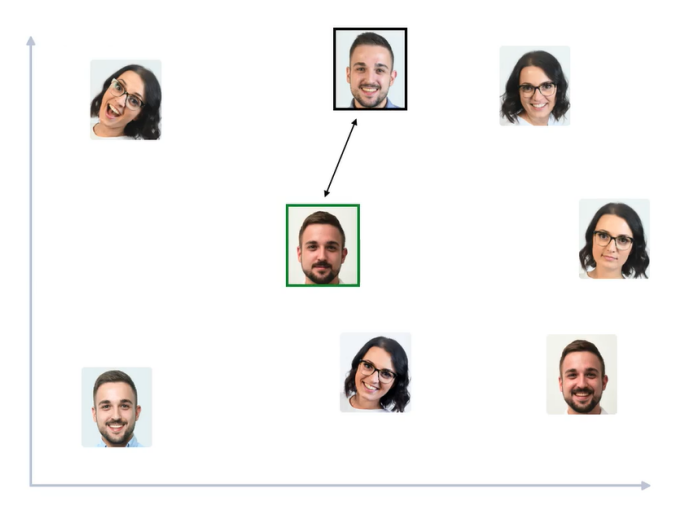
Bước 1: Từ những embedding vectors, thuật toán sẽ tiến hành plot ra ảnh của các đối tượng trên một mặt phẳng 2D một cách ngẫu nhiên.



Hình 2.12. Các đối tượng được plot ngẫu nhiên

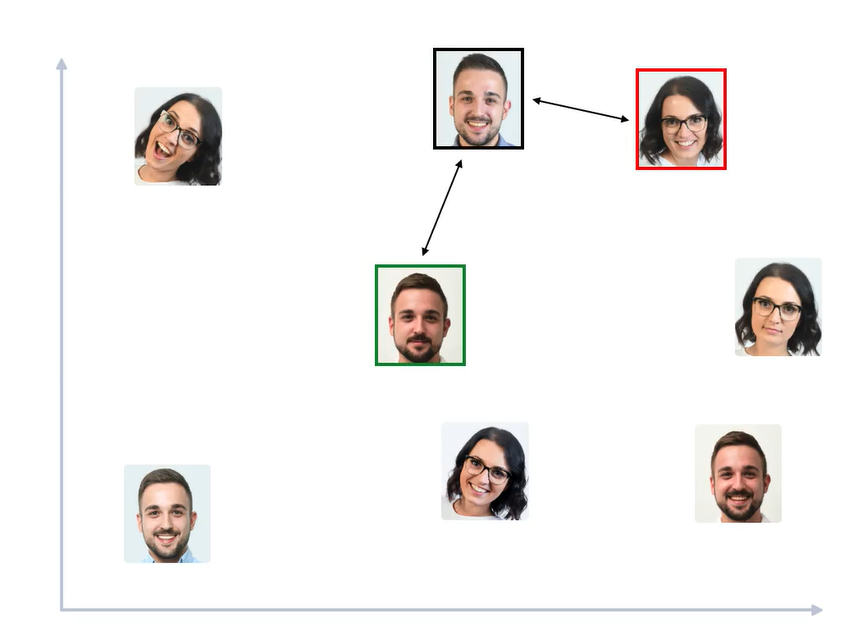
Bước 2: Tiến hành chọn ngẫu nhiên 1 ảnh thành anchor image

Bước 3: Chọn ngẫu nhiên 1 tấm ảnh giống ảnh anchor (positive example)



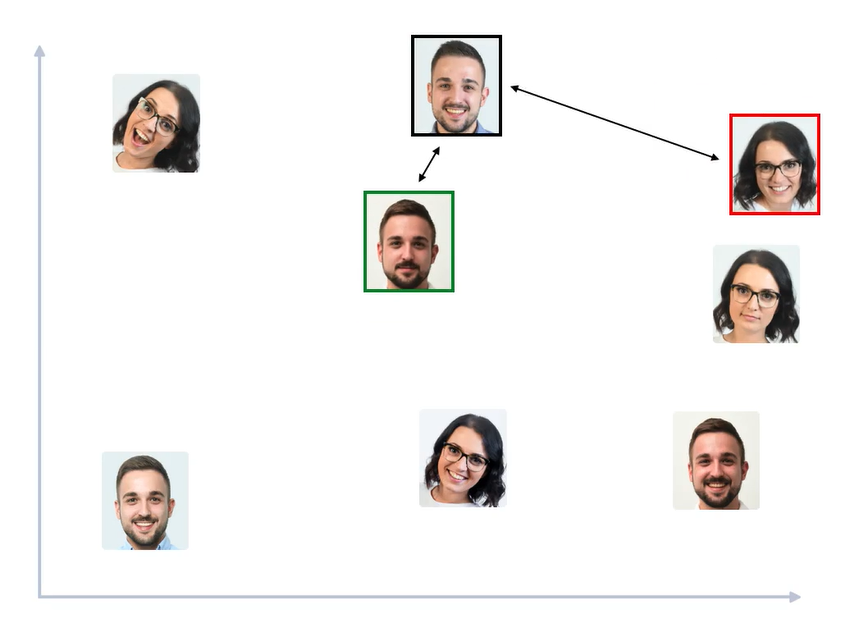
Hình 2.13. Positive example

Bước 4: Chọn ngẫu nhiên 1 tấm ảnh khác ảnh anchor (negative example)



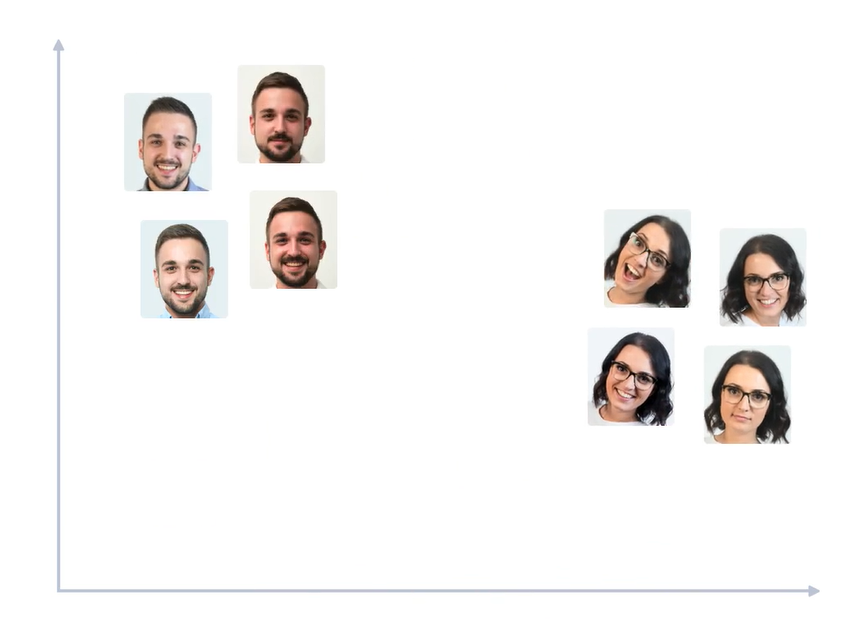
Hình 2.14. Negative Example

Bước 5: Điều chỉnh thông số mạng FaceNet để ảnh positive gần với anchor hơn so với ảnh negative



Hình 2.15. Điều chỉnh thông số

Bước 6: Tiến hành lập lại cho đến khi nhóm được các ảnh



Hình 2.16. Kết quả

Công thức tính toán hàm Loss: Công thức tổng quát được cung cấp trong bài báo viết về Triple Loss:

Trong đó:

: ảnh anchor

: ảnh cùng là 1 người với anchor image

: ảnh khác với người trong anchor image

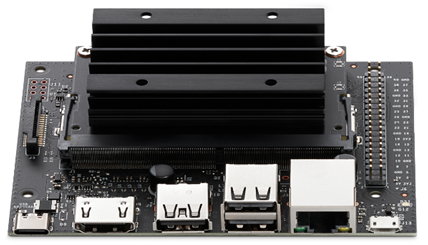
: độ sai lệch giữa 2 miền giá trị

: embedding của ảnh anchor.

**2.5.Giới thiệu phần cứng**

**2.5.1 Giới thiệu NVIDIA Jetson Nano 2GB Developer Kit**

Nvidia Jetson Nano là lý tưởng để học, nghiên cứu , và phát triển trong AI và Robot. Với một cộng đồng phát triển và nhiều dự án mở, phù hợp để học tập và nghiên cứu. Hiệu năng nhận diện , AI với giá thành thấp, phù hợp với mọi người để đi theo ngành này.



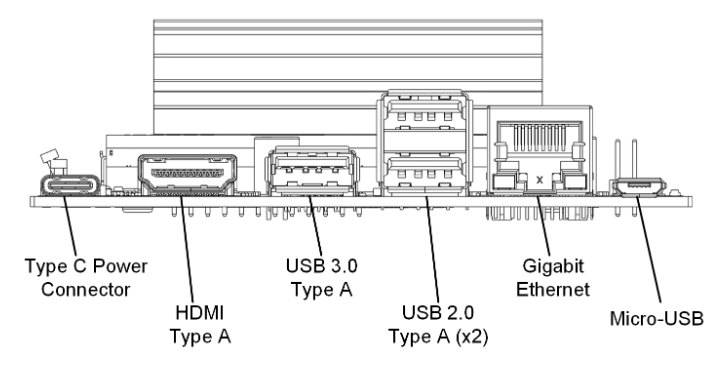
Hình 1: Jetson Nano 2GB.

2.5.1.1. Các thông số của Jetson Nano.

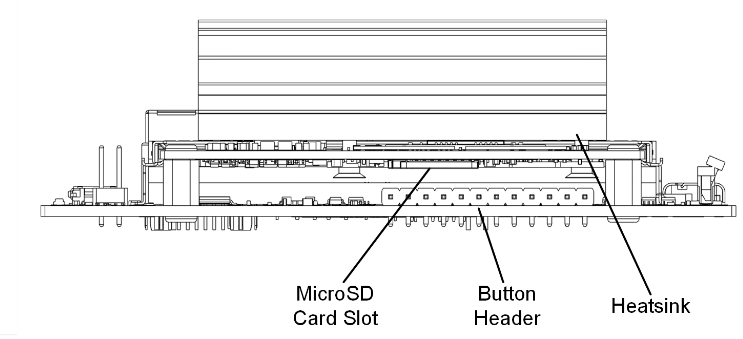
|  |  |
| --- | --- |
| **Processing** |  |
| GPU | 128-core NVIDIA Maxwell™ |
| CPU | 64-bit Quad-core ARM A57 (1.43 GHz) |
| Memory | 2 GB 64-bit LPDDR4 (25.6 GB/s bandwidth) |
| Video Encoder | 4Kp30 | (4x) 1080p30 | (2x) 1080p60 |
| Video Decoder | 4Kp60 | (2x) 4Kp30 | (8x) 1080p30 | (4x) 1080p60 |
| **Interfaces** |  |
| Networking | 10/100/1000 Base-T Ethernet |
| Wi-Fi | Available via included 802.11ac wireless adaptor |
| USB | 1 USB 3.0 Type A port.  2 USB 2.0 Type A ports  1 USB 2.0 Micro-B (Device Mode) |
| Display | HDMI 2.0 |
| 40-Pin Header | GPIOs, I2C, I2S, SPI, PWM, UART |
| Camera | 1x  MIPI CSI-2 connector |
| Storage | microSD (Card not included) |
| Other IO | 12-pin header (Power and related signals, UART) 4-pin fan header |
| Power | USB-C 5V 3A |

Bảng 1. Module Jetson Nano 2GB Developer Kit.

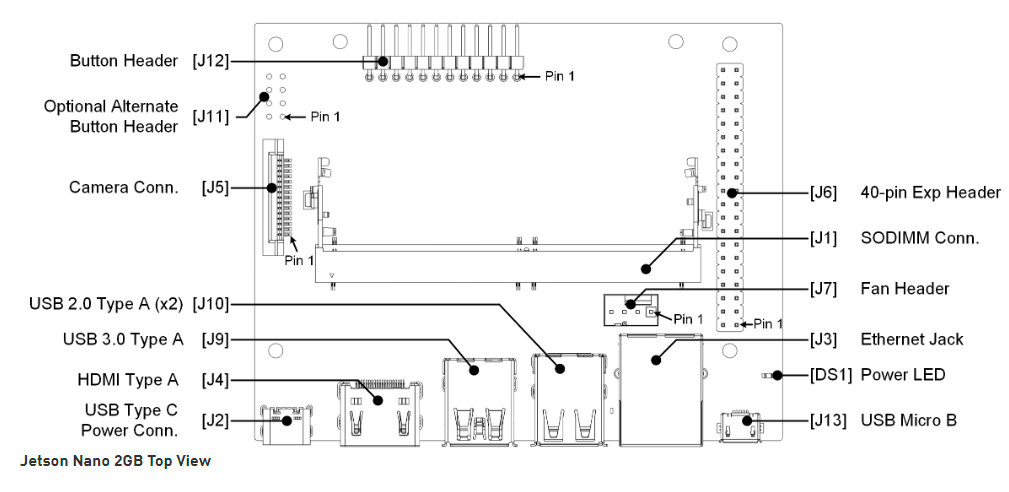
2.5.1.2. Layout của mạch Jetson Nano

****

Hình: Các cổng giao tiếp.

****

Hình: Các khe cắm thẻ nhớ

****

Hình: Tổng quan về các cổng giao tiếp, chân của Jetson Nano.

Chức năng từng cổng:

[DS1] LED tín hiệu: Dùng để báo hiệu là có nguồn cấp vào Jetson Nano.

[J2] USB Type C: Dùng cho giao tiếp cổng USB Type C, là nơi chúng ta cấp nguồn cho Jetson Nano.

[J3] Jack RJ45: Dùng để giao tiếp chuẩn Ethernet, có thể chia sẻ mạng Internet từ Host PC sang Jetson Nano.

[J4] Cổng HDMI: Dùng để hiển thị giao diện người dùng lên màn hình.

[J5] Cổng kết nối với camera.

[J6] 40-pin header: Các Pin với nguồn cấp( +5V/+3.3V) và các Pin giao tiếp I2C (2x), UART, SPI (2x), I2S và GPIOs.

[J7] 4-pin điều khiển quạt .

[J9] USB 3.0 type A.

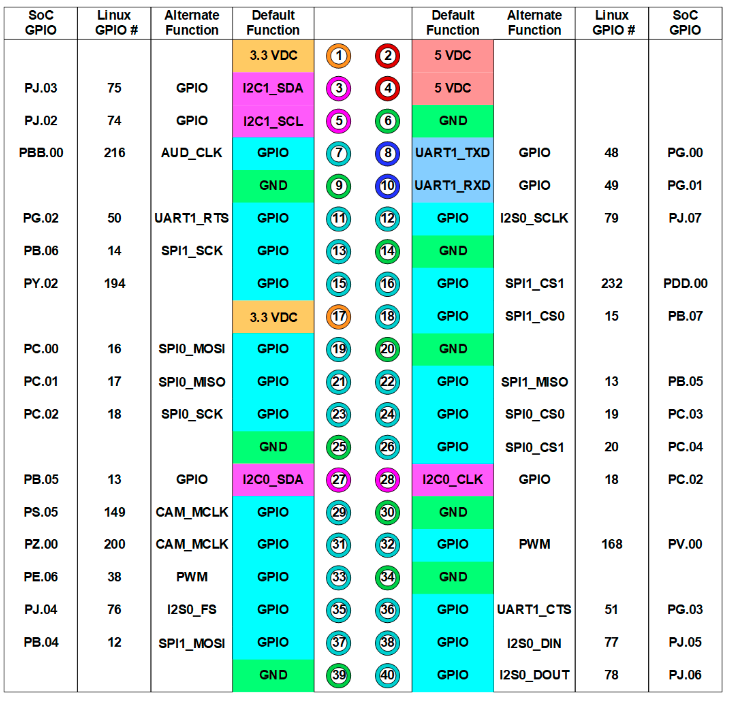
[J10] 2 USB 2.0 type A chỉ kết nối hỗ trợ Host Model.

[J11] Optional nút nhấn (2x4); Bao gồm kết nối cho Reset/Force Recovery/Power Buttons, và Auto-power-on disable.

[J12]Nút nhấn header (1x12): Kết nối tới LED, Reset/Force Recovery/Power Buttons, UART và Auto-power-on disable.

[J13] Micro-USB 2.0.

2.5.1.3. 40 chân giao tiếp (J6)

****

Hình: 40 chân kết nối (J6)

40 chân này sẽ cung cấp kết nối tới nguồn, đất và các ngoại vi.

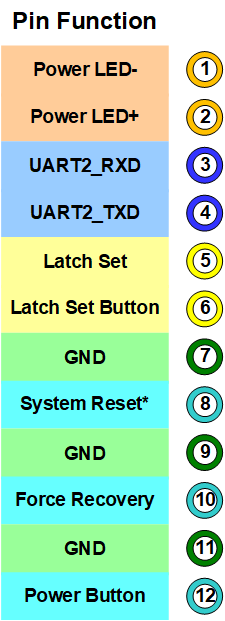
Chân nguồn: có 2 nguồn là 3.3V và 5V. Không thể chuyển đổi, nguồn được cấp liên tục khi kết nối.

Chân tín hiệu:

Tất cả các chân sử dụng mức điện áp 3.3V.

Mặc định các chân này là chân GPIO, ngoại trừ các chân được hỗ trợ I2C, UART.

2.5.1.4. 12 chân Button (J12)



Những chân này cung cấp GPIO để điều khiển trạng thái của mạch.

|  |  |  |  |
| --- | --- | --- | --- |
| **Chân** | **Tên** | **Miêu Tả** | **Điện áp** |
| 1 | Led tín hiệu(-) | Kết nối tới LED Cathode để hiển thị hệ thống Sleep/Wake (Off khi chế độ sleep) | +5V |
| 2 | Led tín hiệu (+) | Kết nối tới LED Anode | +5V |
| 3 | UART2\_RXD | Chân Receiver của UART2 | +3.3V |
| 4 | UART2\_TXD | Chân Transmit của UART2 | +3.3V |
| 5 | Latch Set | Đấu chân 5 và chân 6 để ngắt Auto-Power-On và chuyển sang mở nguồn bằng nút nhấn. | +5V |
| 6 | Latch Set Button | Đấu chân 5 và chân 6 để ngắt Auto-Power-On và chuyển sang mở nguồn bằng nút nhấn | +5V |
| 7 | Ground |  |  |
| 8 | Reset Button | Thường đấu với chân 7 và chân 8. Dùng để khởi tạo lại hệ thống | +1.8V |
| 9 | Ground |  |  |
| 10 | Recovery Mode Button | Thường đấu với chân số 9 và chân số 10. Nhấn giữ nút nguồn trong lúc cấp nguồn hệ thống để đưa vào chế độ USB Force Recovery | +1.8V |
| 11 | Ground |  |  |
| 12 | Power Button | Thường đấu với chân 11 và chân 12. Dùng để bật nguồn khi ta ngắt chế độ Auto-Power-On | +5V |

**2.5.2 Các module ngoại vi**

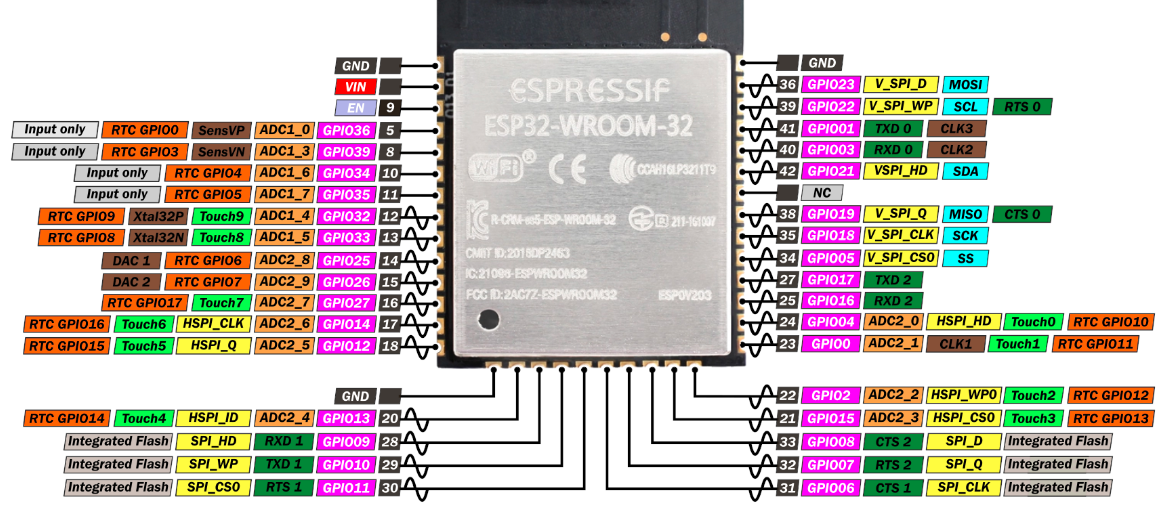
2.5.2.1 Khoá điện



Hình: Khoá điện LY-03.

* Vật liệu: Thép không gỉ
* Nguồn điện: 12V DC
* Dòng điện làm việc: 0.8A
* Công suất: 9.6W
* Yêu cầu nguồn cấp: 12VDC/1A
* Kích thước: L54xD38xH28

2.5.2.2 Module ESP32



Hình: ESP32 và interface.

* 30 chân GPIO.
* Điện áp logic là 2,7 đến 3,3 V.
* 1 Enable Button (Chân reset) và 1 User Button(GPIO 0).
* Led báo nguồn và User Led (GPIO 2).
* 3 UART: Serial Debug mặc định là UART 0.
* Clock: 240MHz

**Input Only Pins:**

GPIO 34 đến 39 – Input Only Pins. Các chân này không có nội trở kéo lên hoặc kéo xuống. Chúng không thể được sử dụng làm output, vì vậy chỉ sử dụng các chân này làm input:

* GPIO 34
* GPIO 35
* GPIO 36
* GPIO 39

**Analog to Digital Converter (ADC)**

ESP32 có 18 kênh input ADC và độ phân giải 12 bit (trong khi ESP8266 chỉ có 1 kênh ADC độ phân giải 10 bit). Đây là các GPIO có thể được sử dụng làm ADC và các kênh tương ứng:

* ADC1\_CH0 (GPIO 36)
* ADC1\_CH1 (GPIO 37)
* ADC1\_CH2 (GPIO 38)
* ADC1\_CH3 (GPIO 39)
* ADC1\_CH4 (GPIO 32)
* ADC1\_CH5 (GPIO 33)
* ADC1\_CH6 (GPIO 34)
* ADC1\_CH7 (GPIO 35)
* ADC2\_CH0 (GPIO 4)
* ADC2\_CH1 (GPIO 0)
* ADC2\_CH2 (GPIO 2)
* ADC2\_CH3 (GPIO 15)
* ADC2\_CH4 (GPIO 13)
* ADC2\_CH5 (GPIO 12)
* ADC2\_CH6 (GPIO 14)
* ADC2\_CH7 (GPIO 27)
* ADC2\_CH8 (GPIO 25)
* ADC2\_CH9 (GPIO 26)

**Digital to Analog Converter (DAC)**

* DAC1 (GPIO25)
* DAC2 (GPIO26)

**PWM**

Bộ điều khiển ESP32 LED PWM có 16 kênh độc lập có thể được cấu hình để tạo tín hiệu PWM với các đặc tính khác nhau. Tất cả các chân có thể hoạt động như output đều có thể được sử dụng làm chân PWM (GPIO từ 34 đến 39 không thể tạo PWM). Các thông số cấu hình:

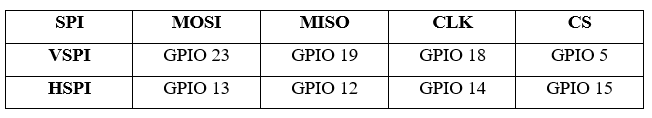
* Tần số tín hiệu.
* Duty cycle.
* Kênh PWM.
* Chân GPIO xuất tín hiệu.

**I2C**

ESP32 có hai kênh I2C và bất kì chân nào cũng có thể cấu hình làm chân SDA và SCL. Khi sử dụng ESP32 với Arduino IDE, chân I2C mặc định là:

* *GPIO 21 (****SDA****)*
* *GPIO 22 (****SCL****)*

**SPI**



* ESP32 có module giao tiếp SPI với thiết bị ngoại vi, gọi là SPI0, SPI1, HSPI, VSPI. SPI0 chỉ dành riêng để kết nối bộ nhớ flash của ESP32 với các thiết bị bố nhớ flash khác bên ngoài.
* SPI1 được kết nối cũng tương tự như SPI0 nhưng nó dùng để ghi dữ liệu cho bộ nhớ Flash của chip.
* HSPI và VSPI và sử dụng tự do. SPI1 và HSPI và VSPI đều có 3 cổng kết với chip, giúp chúng ta dễ dàng kết nối đồng thời với 3 slave bằng giao tiếp SPI mà ESP32 sẽ là thiết bị master.

**Interrupts**: Tất cả các GPIO có thể được cấu hình như ngắt.

2.7. Các thư viện cần thiết cho mô hình nhận dạng khuôn mặt

Trong phần này, nhóm chúng tôi xin giới thiệu một số thư viện chính cho đề tài này.

2.7.1. Numpy

Numpy là một thư viện của Python phục vụ cho các lĩnh vực khoá máy tính. Thư viện này cung cấp các hàm liên quan đến việc tính toán các mảng nhiều chiều có kích thước lớn. Numpy rất quan trọng vì tính hữu dụng của các hàm liên quan đến Đại Số Tuyến Tính. [7]

2.7.2. OpenCV

OpenCV, viết tắt của Open Source Computer Vision Library, được hiểu là một thư viện mã nguồn mở cho việc xử lí computer vision. Cụ thể hơn là một kho lưu trữ các mã nguồn mở được dùng để xử lý hình ảnh, phát triển các ứng dụng đồ hoạ trong thời gian thực.

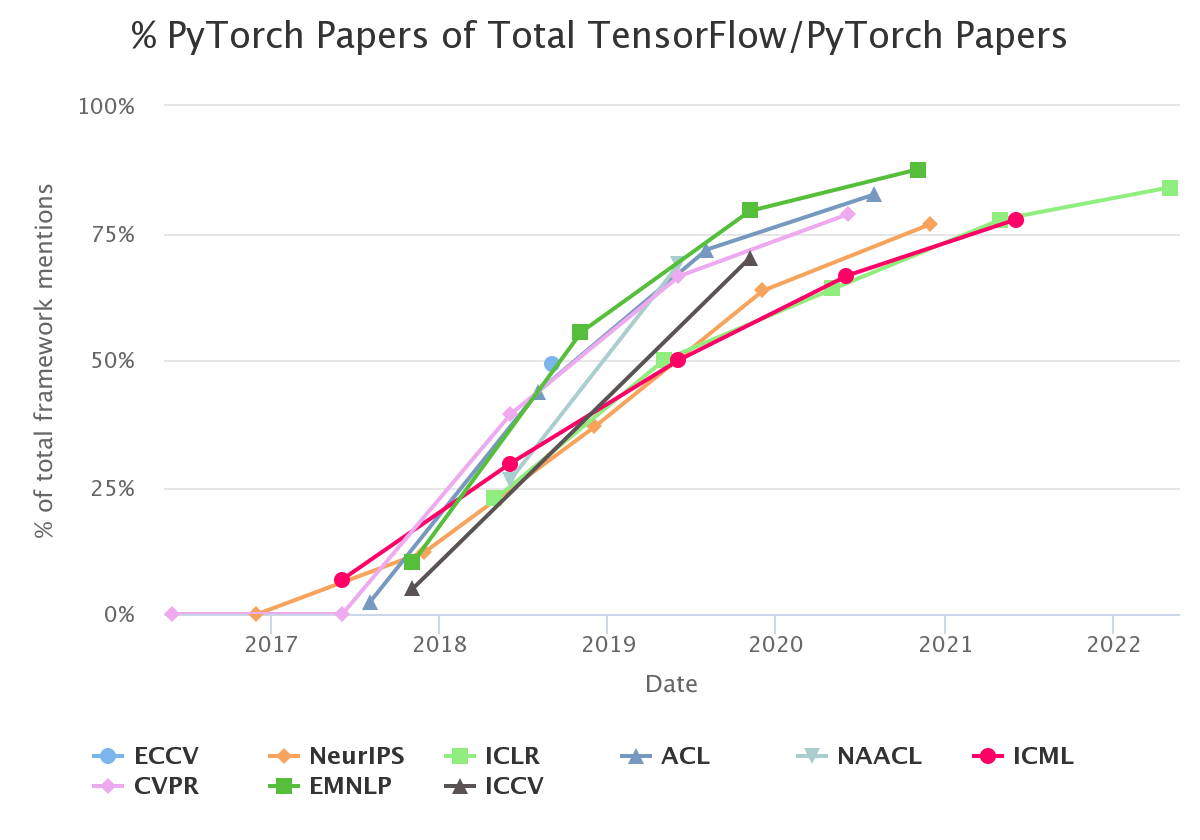
OpenCV kể từ lần đầu xuất hiện từ năm 1999, đã có con số người dùng rất lớn lên tới 47.000 người. Tất cả là nhờ những ưu điểm vượt trội của nó.

OpenCV cho phép cải thiện tốc độ của CPU khi thực hiện các bài toán real-time. Ngoài ra còn cung cấp một số lượng lớn các Service cho bài toán Computer Vision hay là Machine Learning.

Cùng với giấy phép BSD, đây là một thư viện được dùng miễn phí cho cả mục đích thương mại lẫn phi thương mại. OpenCV còn thân thiện với mọi loại ngôn ngữ, ví dụ như C/C++, Python hay Java… Ngoài ra còn tương thích với các hệ điều hành khác nhau như Linux, MacOS, iOS, Android, Window. [8]

2.7.3. Pytorch

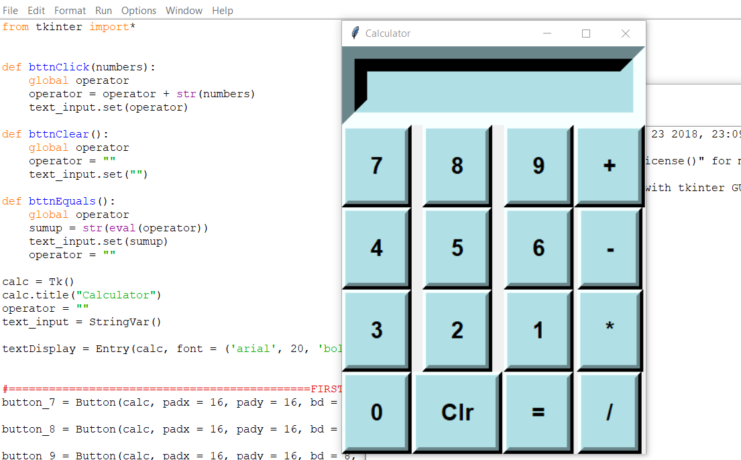
Pytorch là một Deep Learning framework được phát triển bởi Facebook. Chính vì được phát triển với giấy phép mã nguồn mở, Pytorch đã tạo cho mình một cộng đồng rất lớn. Chính vì cộng đồng lớn, mọi vấn đề ta đang thắc mắc có thể được giải quyết và được chia sẻ lên cộng đồng. Pytorch cùng với Tensorflow và Keras là một trong những framework được phổ biến được sử dụng trong lĩnh vực Deep Learning hiện nay. Hầu hết các Paper liên quan đến lĩnh vực này, các code public thường được viết dưới dạng Pytorch hoặc Tensorflow. Có thể tham khảo tỉ lệ số lượng Paper sử dụng Pytorch và Tensorflow ở dưới:



Hình: So sánh tỉ lệ số lượng paper sử dụng Pytorch và Tensorflow [[nguồn](http://horace.io/pytorch-vs-tensorflow/" \t "_blank)]

2.7.4. Tkinter

Tkinter là một thư viện GUI dành cho ngôn ngữ Python. Tkinter chứa những module Tk hỗ trợ cho việc lập trình giao diện hướng đối tượng.



Hình. Một giao diện Calculator tạo bởi thư viện tkinter

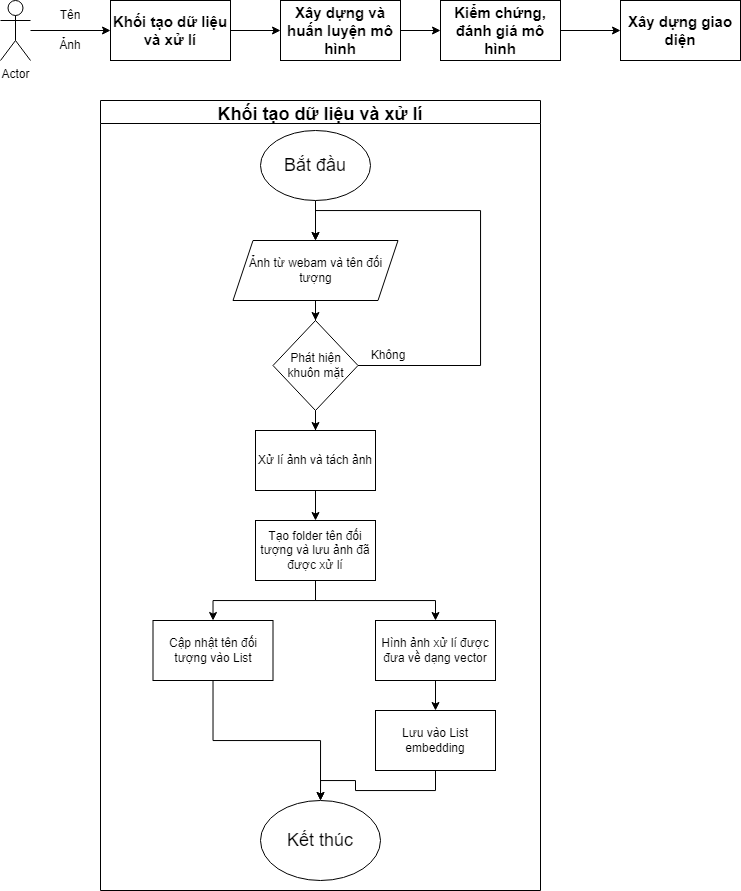
CHƯƠNG 3: THIẾT KẾ HỆ THỐNG NHẬN DIỆN KHUÔN MẶT

3.1. Yêu cầu của hệ thống

Chương trình được triển khai lên hệ thống bằng những phần cứng có giới hạn về tài nguyên có các chức năng như sau:

* Chương trình tự tạo tập Dataset từ những khuôn mặt được ghi hình từ camera và được gán tên đối tượng tương ứng.
* Mô hình nhận dạng được khuôn mặt và xác định đó là đối tượng nào từ những tập Dataset đã được tạo ở trên.
* Đóng khung khuôn mặt đó và hiển thị tên lên chương trình.

3.2. Quy trình tạo hệ thống



Theo quy trình chúng tôi đề ra, việc tạo ra hệ thống sẽ có các bước chính:

* Khối tạo dữ liệu và xử lí: Khối này có đầu vào là những frame từ camera và tên của đối tượng trong tấm ảnh đó. Dùng kỹ thuật phát hiện khuôn mặt từ những frame, tiến hành cắt lấy phần khuôn mặt và thay đổi kích thước để phù hợp với kích thước đầu vào mô hình đồng thời lấy phần toạ độ tương ứng. Sau đó tạo Folder để chứa những bức ảnh xử lí, đồng thời cập nhật tên đối tượng cùng với toạ độ thu được vào List, một danh sách chứa tên các đối tượng nhận diện cho mô hình.
* Xây dựng và huấn luyện mô hình: Xây dựng các mô hình FaceNet và MTCNN và chọn các thông số cần thiết cho bài toán.
* Kiểm chứng, đánh giá mô hình: Tiến hành kiểm tra chất lượng mô hình, nếu quá trình nhận diện đạt yêu cầu tiến hành bước tiếp theo. Nếu mô hình không đạt, quay lại bước trước đó, tiến hành kiểm tra và chỉnh sửa lại thông số.
* Xây dựng ứng dụng: Xây dựng phầm mềm trên Linux để tích hợp mô hình nhận diện, giúp cho việc tương tác với người dùng dễ dàng hơn. Phần này sẽ được đề cập ở chương sau.

3.3. Thiết kế hệ thống

3.3.1. Khối tạo dữ liệu và xử lí

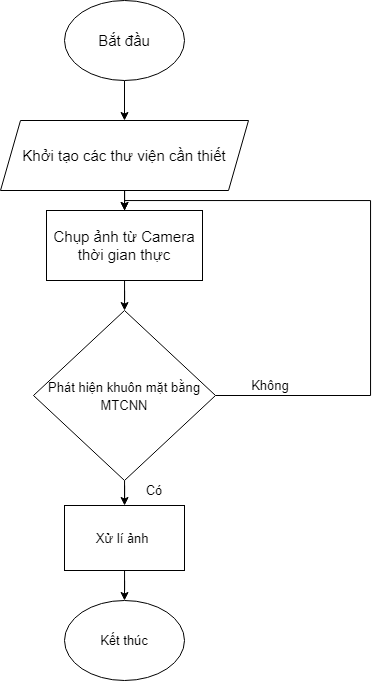
3.3.1.1. Phát hiện khuôn mặt

Để có thể phát hiện khuôn mặt từ những frame thu được từ camera, chúng ta cần một mô hình nhận diện đã được train trước (pre-trained models). Tác giả của FaceNet đã cung cấp chúng ta 2 mô hình với 2 tập Dataset khác nhau là CASIA-WebFace và VGGFace2 đã được huấn luyện với kiến trúc Inception ResNet V1. Nhóm chúng tôi sẽ chọn mô hình được huấn luyến từ tập VGGFace2 để phục vụ cho bài toán, bởi vì độ chính xác của nó lên tới 0.9965.

Tập VGGFace2 có tổng số 3.31 triệu ảnh của 9131 đối tượng khác nhau, tức trung bình mỗi đối tượng có 362 ảnh trong bộ Dataset. Phần lớn những ảnh này thu được từ công cụ tìm kiếm hình ảnh của Google với sự đa dạng về giới tính, tuổi tác, màu da, tư thế, môi trường,… Những tập này đi qua kiến trúc ResNet-50 hoặc SqueezeNet-ResNet-50 tạo ra những model, chúng đều được đánh giá trên tập dữ liệu chuẩn và đều đạt được state-of-the-art. [9]

3.3.1.2. Xử lí và tách ảnh

Những frame ta thu được từ camera không thể nào đem làm đầu vào của mô hình dự đoán được. Vì vậy khối này sẽ trích xuất đối tượng đặc trưng cần thiết, xử lý ảnh để phù hợp với đầu vào của mô hình dự đoán. Chương trình này có thể thực hiện theo lưu đồ sau



Sau khi khởi tạo các thư viện cần thiết, hệ thống sẽ khởi động Camera để chụp ảnh đầu vào. Sau đó bằng phương pháp MTCNN để xác định xem trong frame này có khuôn mặt của một người không, nếu có sẽ tiến hành bước xử lí ảnh bao gồm cắt ảnh phần đặc trưng cần thiết, thay đổi kích thước phù hợp mô hình , đổi dạng ảnh RGB sang BGR; Trường hợp frame không có khuôn mặt nào, các frame đó sẽ bỏ qua và chờ một frame khác. Sau khi xử lí ảnh xong chương trình sẽ quay lại bước chụp ảnh đầu vào.

3.3.1.3.Hình ảnh

3.3.2. Xây dựng và huấn luyện mô hình

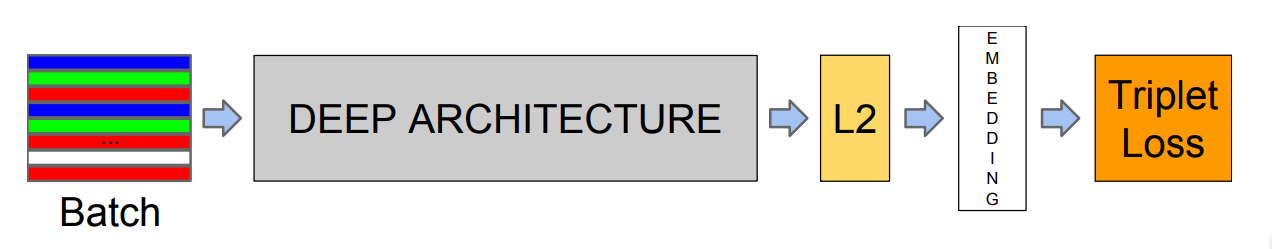
3.3.2.1. Quy trình Training:

+ Với mỗi đối tượng ta sẽ có một tập Dataset với một lượng ảnh nhất định.

+ Xây dựng một mạng DNN dùng để Feature Extractor cho Dataset trên, kết quả là thu được một Embedding 128-Dimensions. Trong paper [5] có hai mạng đặc trưng là Zeiler&Fergus và InceptionV1.

+ Huấn luyện mạng DNN để kết quả Embedding có khả năng nhận diện tốt, bao gồm việc sử dụng l2 normalization (Khoảng cách Euclide) cho các embedding đầu ra và tối ưu lại các thông số trong mạng bằng Triplet Loss.

+ Hàm Triplet Loss sẽ sử dụng phương pháp Triplet Selection, lựa chọn các embedding sao cho việc học diễn ra tốt nhất.



Hình : FaceNet architecture

3.3.2.2. Quy trình tạo Interface

+Truyền ảnh mặt cần classify vào trong mạng Feature Extractor, thu được 1 embedding.

+Tiến hành sử dụng hàm l2 và so sánh với các embedding khác trong tập embeddings đã có. Việc classify sẽ giống như thuật toán k-NN với k = 1.

3.4 Tiến hành tạo hệ thống

Đây là quy trình chúng tôi thực hiện để tạo ra hai khối “Khối tạo dữ liệu và xử lí” và “Xây dựng và huấn luyện mô hình”. Toàn bộ quá trình tạo ra chương trình đều được viết bằng ngôn ngữ Python và dựa trên framework chính là Pytorch. Vì Pytorch đã hỗ trợ những module cần thiết từ paper của MTCNN lẫn FaceNet và cung cấp pre-trained model trên các tập dữ liệu lớn. Cho nên nhóm chúng tôi sẽ tiến hành inference một mô hình hoàn chỉnh, không tập trung vào xây dựng và train từ đầu những kiến trúc này.

Ở thời điểm hiện tại làm đề tài này, chúng tôi sử dụng Python 3.8 cùng với các thư viện cần thiết như sau:

torch==1.8.0

torchvision==0.9.0

numpy==1.19.2

opencv-python==4.5.1.48

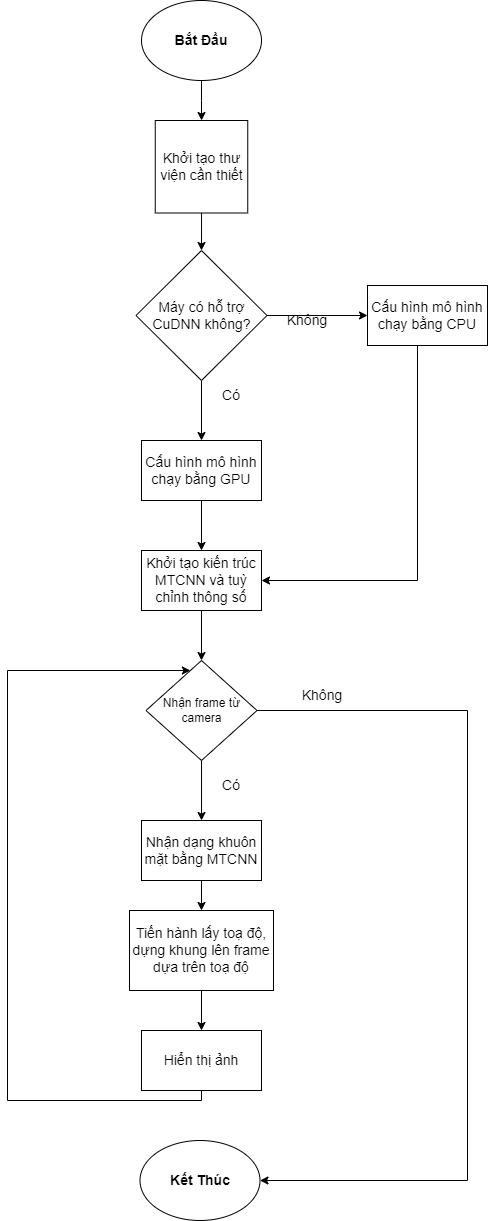
Tiếp theo, chúng ta cần một library đã implement từ Paper của cả MTCNN lẫn FaceNet, và cung cấp cho chúng ta file pre-trained model. Ở đây, nhóm chúng tôi đã sử dụng một library được viết dựa trên framework PyTorch. Có thể cài thông qua câu lệnh:

pip install facenet-pytorch

3.4.1. Khối tạo dữ liệu và xử lí

3.4.1.1. Chương trình Detect Face

Đầu tiên nhóm chúng tôi tiến hành xây dựng một chương trình có thể phát hiện khuôn mặt đối tượng bất kỳ thu được từ camera. Bằng cách dựa vào 2 library là OpenCV cùng với MTCNN từ facenet-pytorch. Chương trình này sẽ chạy theo lưu đồ thuật toán sau đây:



Trước hết ta khai báo thư viện cần thiết cho yêu cầu trên và đồng thời dùng PyTorch để kiểm tra thiết bị trên máy để quyết định mô hình sẽ chạy trên CPU hay là GPU với hỗ trợ CUDA.

import cv2

from facenet\_pytorch import MTCNN

import torch

import numpy as np

import time

device =  torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')

print(device)

Tiếp theo, ta sẽ gọi 1 object từ class MTCNN cùng với thiết lập một số thông số như sau

mtcnn = MTCNN(thresholds= [0.7, 0.7, 0.8] ,keep\_all=True, device = device)

Parameter “thresholds” là mức ngưỡng cho 3 lớp mạng tương ứng là P, R và O. Mặc định của 3 thông số này là [0.6, 0.7, 0.7], để tăng độ chính xác thì phải truyền giá trị cao hơn như ở trên. Parameter “keep\_all” để kích hoạt việc phát hiện và trả về tất cả các khuôn mặt có thể có trong hình. Parameter “device” để xác định thiết bị CPU hay GPU chạy mô hình.

Tiếp theo, chúng ta sẽ tiến hành mở webcam bằng cách dùng thư viện OpenCV. Chúng ta sẽ tiến hành phát hiện khuôn mặt và tiến hành khoanh box cho từng frame. Trong class MTCNN cung cấp cho chúng ta một hàm tên là detect,giúp trả về là một list các box hình chữ nhật, mỗi box sẽ gồm 2 tọa độ tương ứng 2 góc của box (điểm góc trái trên và điểm góc phải dưới của hình chữ nhật). Dựa vào list các box thu được, ta sẽ dùng Op enCV để dựng nên hình chữ nhật dựa theo tọa độ đó để khoanh vùng khuôn mặt.

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, \_ = mtcnn.detect(frame)

if boxes is not None:

for box in boxes:

bbox = list(map(int,box.tolist()))

frame = cv2.rectangle(frame,(bbox[0],bbox[1]),(bbox[2],bbox[3]),(0,0,255),6)

cv2.imshow('Real-time faceDetect', frame)

if cv2.waitKey(1)&0xFF == 27:

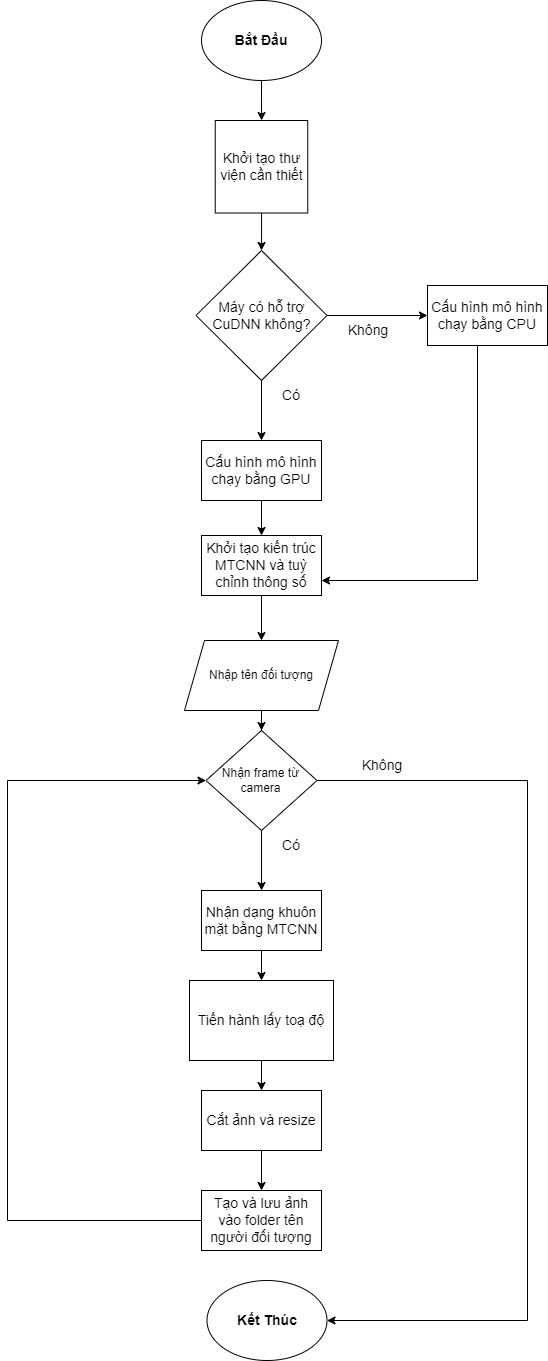
break

cap.release()

cv2.destroyAllWindows()

3.4.1.2. Chương trình Capture Face

Tiếp theo, chúng ta sẽ tiến hành viết một chương trình tạo ra Folder để chứa ảnh khuôn mặt của đối tượng. Chương trình này tương tự với chương trình Detect đã đề cập ở trên nhưng một số Parameter sẽ thay đổi để hợp với yêu cầu bài toán. Sau đây là giải thuật của chương trình này:



count = 50

usr\_name = input("Tên của đối tượng: ")

USR\_PATH = os.path.join(IMG\_PATH, usr\_name)

step = 1

count được dùng để giới hạn số lượng ảnh cho một đối tượng. step là bước nhảy, tức là thay vì ta sẽ lấy từng frame một về thì sau step frame ta sẽ lấy một ảnh.

mtcnn = MTCNN(margin = 20, keep\_all=False, post\_process=False, device = device)

Tham số truyền vào class MTCNN sẽ thay đổi so với chương trình Detect, như việc thêm vào margin nhằm để lấy box to hơn, post\_process trả về False để giữ cho những pixel ảnh được lưu không bị normalization về giá trị [-1,1].

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)

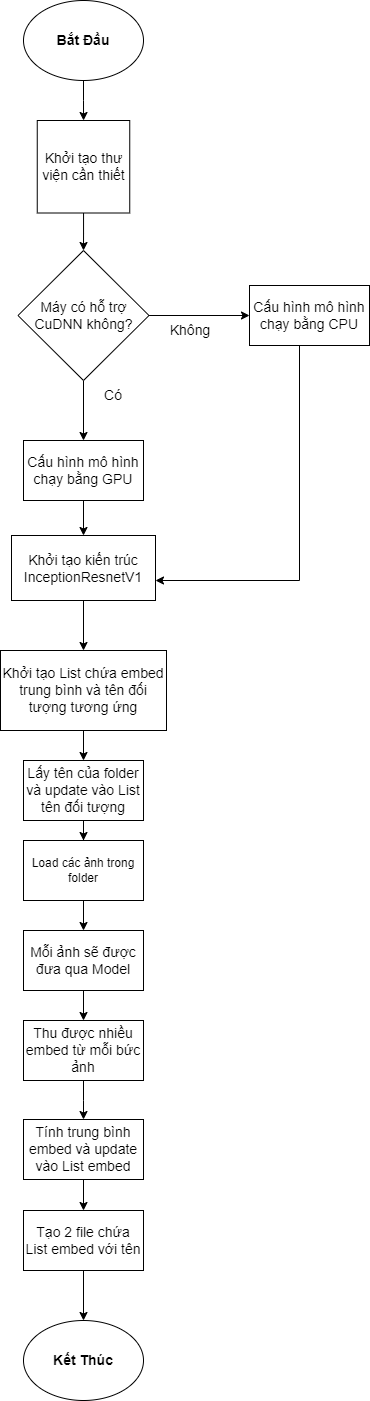
if cv2.waitKey(1)&0xFF == 27:

break

Thay vì dùng detect như chương trình trên, chúng ta sẽ dùng hàm forward thẳng qua class. Hàm này sẽ trả về ảnh dựa trên 2 tọa độ góc của box chứa mặt người sau khi cắt và resize ảnh về đúng kích thước (160x160).

3.4.1.3. Chương trình cập nhật Face List

Chương trình này sẽ dựa vào khuôn mặt, để dự đoán xem đây là khuôn mặt của đối tượng nào. Được hoạt động theo giải thuật sau:



Đầu tiên ta viết một hàm normalize để đưa pixel ảnh về trong khoảng [-1,1] mà vẫn giữ được tính chất của nó. Việc normalize rất quan trọng vì khi dữ liệu được chuẩn hóa về một khoảng nhất định, thuật toán Gradient Descent sẽ cho kết quả nhanh hơn và tránh các Vanishing/Exploding Gradient.

def trans(img):

transform = transforms.Compose([

transforms.ToTensor(),

fixed\_image\_standardization

])

return transform(img)

Tiếp theo ta sẽ khai báo model được sử dụng. Trong paper [5] đã đề cập rằng FaceNet sử dụng InceptionV1 cho ra một Feature Vector có 128 chiều. Nhưng vì kiến trúc đã lỗi thời, nên đã thay vào đó là một kiến trúc InceptionResnetV1 với đầu ra là vector 512 chiều để cải thiện hiệu suất. Và ta sẽ sử dụng model pre-trained với tập Dataset VGGFace2 bằng cách khai báo

model = InceptionResnetV1(

classify=False,

pretrained="casia-webface"

).to(device)

model.eval()

Và cuối cùng chúng ta sẽ tạo lập embed từ các ảnh khuôn mặt

for usr in os.listdir(IMG\_PATH):

embeds = []

for file in glob.glob(os.path.join(IMG\_PATH, usr)+'/\*.jpg'):

try:

img = Image.open(file)

except:

continue

with torch.no\_grad():

embed = model(trans(img).to(device))

embeds.append(embed.unsqueeze(0))) #1 anh, kich thuoc [1,512]

if len(embeds) == 0:

continue

embedding = torch.cat(embeds).mean(0, keepdim=True) #dua ra trung binh cua 50 anh, kich thuoc [1,512]

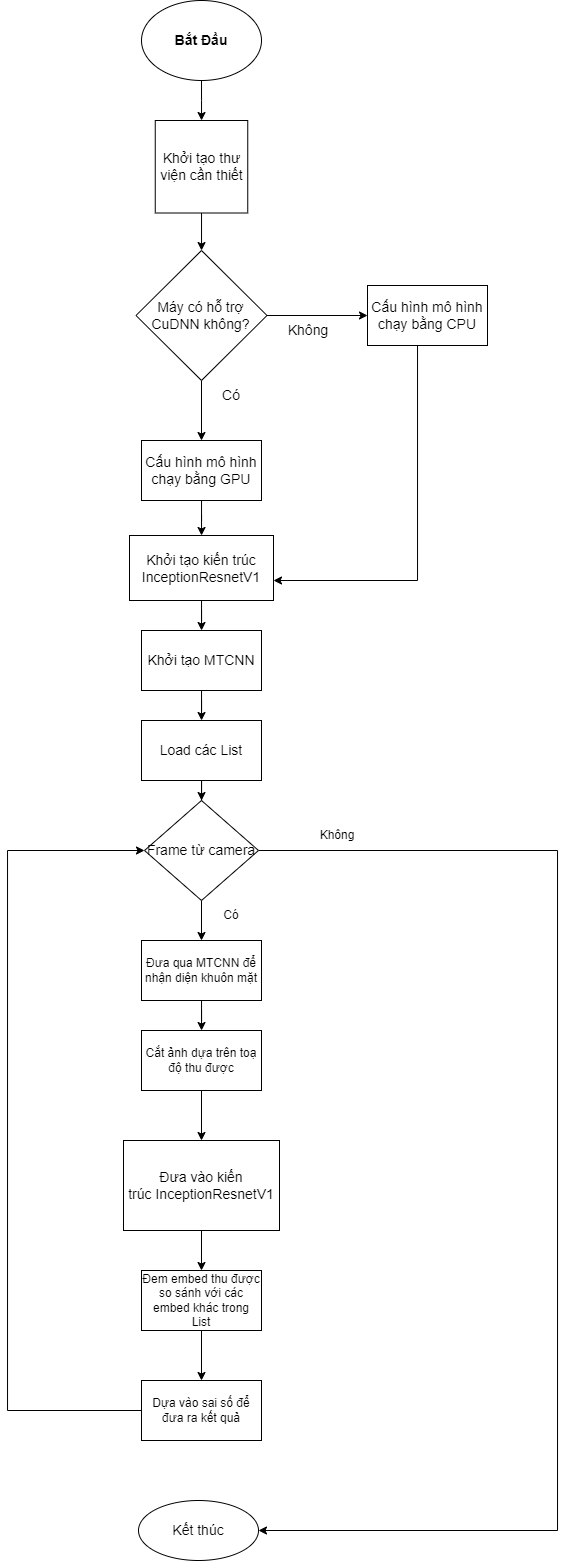
embeddings.append(embedding) # 1 cai list n cai [1,512]

names.append(usr)

List names sẽ là danh sách tên của các đối tượng có trong folder. Với mỗi folder (đối tượng), ta xét toàn bộ ảnh, mỗi ảnh sẽ dùng model đã được pre-trained trên để sinh ra một vector [1,512]. Tức là 50 ảnh sẽ thu được 50 embed cho một đối tượng. Tuy nhiên ta chỉ cần một embed đại diện cho một đối tượng là đủ, vậy nên 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 embed. 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.

3.4.2. Khối xây dựng và huấn luyện mô hình

Khối này sẽ tiến hành tạo ra một chương trình có thể đưa bất kì khuôn mặt nào để nhận dạng xem người đó có trong List đã được huấn luyện không và đưa ra kết quả. Lưu đồ thuật toán ta có như sau:



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)

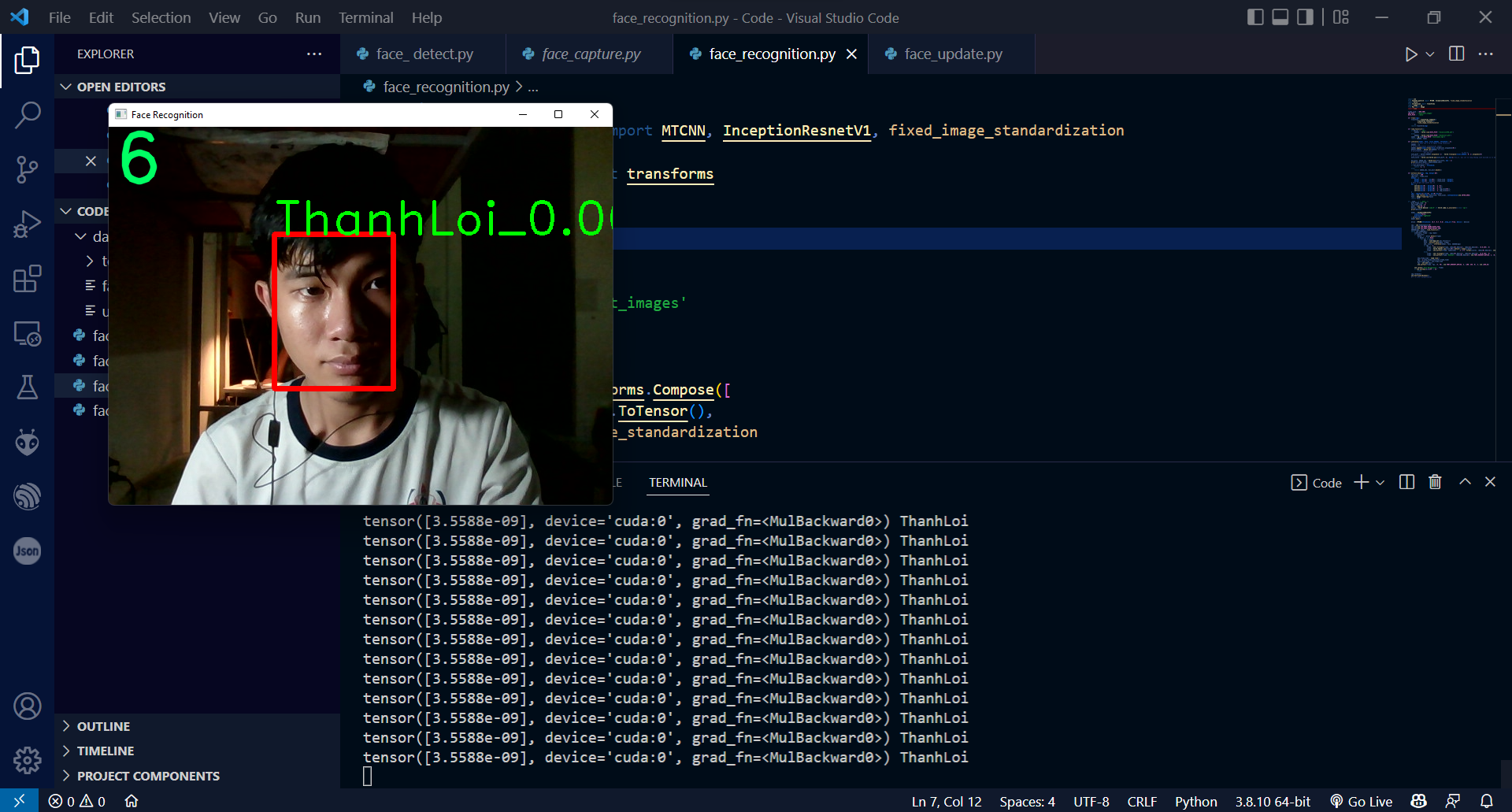
frame = cv2.putText(frame,'Unknown', (bbox[0],bbox[1]), cv2.FONT\_HERSHEY\_DUPLEX, 2, (0,255,0), 2, cv2.LINE\_8)

Đầu tiên, ta sẽ cần load lại FaceList và usernames đã được tạo và lưu ở chương trình Face Capture thông qua hàm load\_faceslist(). Sau đó ta sẽ cho frame vào hàm detect của MTCNN để thu được list tọa độ các box có chứa mặt. Ta sẽ tạo một hàm extract\_face() để cho list tọa độ thu được đó để mình có được tọa độ bounding box để có thể khoanh vùng trên frame ảnh cùng với ảnh mặt để tạo embedding để tìm ra sự tương đồng với embedding khác trong FaceList.

Và cuối cùng là hàm inference() sẽ có công dụng kết xuất embedding cho từng ảnh mặt thu được từ hàm extract\_face(), sau đó so sánh khoảng cách giữa embedding vừa nhận được với các embedding khác có trong FaceList để thu được kết quả là những score. Dựa vào đó ta sẽ biết được tên đối tượng là ai và hiển thị lên frame thông qua câu lệnh cv2.putText().

3.5. Kiểm chứng, đánh giá mô hình

-Kết quả thu được





-Đánh giá:

+ Độ chính xác của mạng sẽ giảm rất đáng kể khi gặp phải vấn đề về bất lợi độ sáng. Ta có thể khắc phục bằng cách tăng threshold để chấp nhận score cao, hoặc thêm dataset ở điều kiện ánh sáng đó.

+ Chưa triển khai được Align mặt như Paper gốc đề cập, khiến cho việc khi người dùng có những góc mặt khác nhau, embedding thu được cũng sai lệch rất nhiều.

+ Phần Classifier nhóm chỉ dựa vào khoảng cách giữa embedding nên kết quả thu được chưa tốt. Đề xuất khắc phục có thể xây dựng một Classifier đơn giản mà mạnh hơn như SVM hoặc một mạng FCN.

CHƯƠNG 4: PHÂN TÍCH VÀ THIẾT KẾ GIAO DIỆN CHO CHƯƠNG TRÌNH

4.1.

CHƯƠNG 5: PHÂN TÍCH VÀ THIẾT KẾ CÁNH CỬA TỰ ĐỘNG

5.1. Yêu cầu bài toán

Khi Jetson Nano có nhiệm vụ ghi hình và thực hiện thuật toán nhận dạng khuôn mặt, nhằm giảm bớt khối lượng công việc trên Jetson Nano nhóm chúng tôi chọn một microcontroller ít tiêu thụ năng lượng là ESP32, hơn nữa nó còn tích hợp Wi-Fi cùng Bluetooth để có thể mở rộng ứng dụng. Nhiệm vụ của ESP32 sẽ có nhiệm vụ nhận tín hiệu hoặc truyền tín hiệu Jetson Nano cùng với điều khiển các ngoại vi để mở/đóng cửa.

5.2. Lưu đồ giải thuật

Tài liệu tham khảo:

Hình 2.1. => <https://stanford.edu/~shervine/l/vi/teaching/cs-230/cheatsheet-convolutional-neural-networks>

Hình 2.2 => <https://arxiv.org/abs/2011.07747>

Hình 2.3 => [https://phamdinhkhanh.github.io/2020/03/12/FaceNetAlgorithm.html#4-thu%E1%BA%ADt-to%C3%A1n-FaceNet](https://phamdinhkhanh.github.io/2020/03/12/faceNetAlgorithm.html#4-thu%E1%BA%ADt-to%C3%A1n-facenet)

Hình 2.4, 2.5, 2.6 ,2.7=> [4]

Hình 2.8, 2.9, 2.10 => <https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-face-detection-and-facial-landmark-alignment-7c21e8007923>

Hình jetson: <https://developer.nvidia.com/embedded/learn/jetson-nano-2gb-devkit-user-guide#id-.JetsonNano2GBDeveloperKitUserGuidevbatuu_v1.0-Introduction>

Hình esp32: https://deviot.vn/tutorials/esp32.66047996/gioi-thieu-ve-esp32-devboard.60785539

# Bibliography

|  |  |
| --- | --- |
| [1] | Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall, *"Activation Functions: Comparison of Trends in Practice and Research for Deep Learning",* 8 Nov 2018. |
| [2] | Enzo Grossi and Massimo Buscema, *"Introduction to artificial neural networks",* 29 August 2007. |
| [3] | Ratna Astuti Nugrahaeni, Kusprasapta Mutijarsa, *"Comparative Analysis of Machine Learning KNN, SVM, and Random Forests Algorithm for Facial Expression Classification",* IEEE, March 2017. |
| [4] | Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, Yu Qiao, *"Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks",* arXiv, 11 Apr 2016. |
| [5] | Florian Schroff, Dmitry Kalenichenko, James Philbin, *"FaceNet: A Unified Embedding for Face Recognition and Clustering",* arXiv, 17 Jun 2015. |
| [6] | *Going Deeper with Convolutions,* arXiv, 17 Sep 2014. |
| [7] | "NumPy User Guide 1.22.4," May 20 2022. [Online]. Available: https://numpy.org/doc/stable/index.html. |
| [8] | "OpenCV modules," Jun 5 2022. [Online]. Available: https://docs.opencv.org/4.6.0/. |
| [9] | Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi and Andrew Zisserman, "VGGFace2: A dataset for recognising faces across pose and age," arXiv:1710.08092v2 [cs.CV] , 13 May 2018. |