TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA ĐIỆN ĐIỆN TỬ

BỘ MÔN KỸ THUẬT MÁY TÍNH – VIỄN THÔNG

NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH

ĐỒ ÁN 2

THIẾT KẾ HỆ THỐNG NHẬN DIỆN NGƯỜI ĐEO KHẨU TRANG

SVTH:	MSSV
Trương Thành Lợi	18119169
Cù Khắc Lực	18119172

Tp. Hồ Chí Minh, tháng 12 năm 2021

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA ĐIỆN ĐIỆN TỬ

BỘ MÔN KỸ THUẬT MÁY TÍNH – VIỄN THÔNG

NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH

ĐỒ ÁN 2

THIẾT KẾ HỆ THỐNG NHẬN DIỆN NGƯỜI ĐEO KHẨU TRANG

SVTH:	MSSV
Trương Thành Lợi	18119169
Cù Khắc Lực	18119172

GVHD: PHAN VĂN CA

Tp. Hồ Chí Minh, tháng 12 năm 2021

LỜI CẢM ƠN

Đầu tiên cho em xin trân trọng gửi lời cảm ơn chân thành và sự kính trọng tới thầy Phan Văn Ca, giảng viên trường Đại học Sư phạm Kỹ thuật TPHCM đã tận tình hướng dẫn nhóm em trong suốt quá trình thực hiện đồ án này.

Do kiến thức còn hạn hẹp nên không tránh khỏi những thiếu sót trong cách hiểu, lỗi trình bày. Em rất mong nhận được sự đóng góp ý kiến của thầy để báo cáo tiểu luận đạt được kết quả tốt hơn.

Nhóm em xin chân thành cảm ơn!

ĐIỂM SỐ

TIÊU CHÍ	NỘI DUNG	BỐ CỤC	TRÌNH BÀY	TỔNG
ĐIỂM				

NHẬN XÉT

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Ký tên

Phan Văn Ca

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

CÁC TỪ VIẾT TẮT

1. **mAP: Mean Average Precision**
2. **FC: Fully Connected**
3. **IoU: Intersection over Union**
4. **CNN: Convolutional Neural Network**
5. **TTL: Transistor-transistor Logic**
6. **AP: Average Precision**
7. **SoM: System on Module**
8. **URL: Uniform Resource Locator**

CHƯƠNG 1:TỔNG QUAN

1.1. ĐẶT VẤN ĐỀ

Đại dịch COVID-19 là một thách thức đối với nền y tế của tất cả toàn cầu. Theo Trung tâm kiểm soát và phòng ngừa dịch bệnh (CDC), COVID-19 lây lan khi người nhiễm bệnh thở ra các giọt bắn và các hạt rất nhỏ có chứa vi-rút. Những giọt bắn và hạt này có thể bị người khác hít vào hoặc rơi vào mắt, mũi hoặc miệng của họ. Trong một số trường hợp, họ có thể gây ô nhiễm các bề mặt họ chạm vào. Việc chúng ta đeo khẩu trang là một điều cấp thiết vì sự hiệu quả việc này ngăn chặn được 80% tất cả các bệnh liên quan đường hô hấp. Do đó, nhiều hệ thống giám sát và phát hiện ra hành vi đeo mặt nạ đã được phát triển để phục vụ giám sát hiệu quả ở các bệnh viện, sân bay, địa điểm buôn bán,… Tuy nhiên, các hệ thống phát hiện hành vi đeo mặt nạ thường trở nên rất đắt đỏ, hay các phầm mềm hay phần cứng bị thì bó buộc khiến cho việc tiếp cận vấn đề cũng rất khó khăn. Trong đồ án lần này, chúng tôi sẽ triển khai một mô hình AI để nhận dạng người đeo khẩu trang. Từ đó, chúng tôi sẽ nghiên cứu cách triển khai mô hình này lên một phần cứng cụ thể tạo ra một hệ thống hoàn chỉnh.

1.2. MỤC TIÊU

- Tìm hiểu về Deep Learning và các ứng dụng .

- Hiểu rõ được cơ sở lí thuyết, kiến trúc của mô hình CNN và các mô hình bài toán nhận diện vật thể.

- Sử dụng các thư viện hỗ trợ, môi trường ảo để thực thi mô hình.

- Tiến hành tối ưu mô hình và nghiên cứu các triển khai trên phần cứng.

- Nghiên cứu vi điều khiển và chọn ra một vi điều khiển phù hợp cho vấn đề đặt ra.

- Tìm hiểu các giao thức giao tiếp IoT.

1.3. NỘI DUNG NGHIÊN CỨU

Ở báo cáo này,nhóm sẽ nghiên cứu về Deep Learning, và các thuật toán về nhận dạng vật thể Yolo,…Đồng thời sẽ thực hiện một số bài toán tối ưu cho mô hình để đạt được FPS cao và độ chính xác ở mức trung bình. Nếu được nhóm sẽ tiến hành một số chức năng cho hệ thống này.

1.4. BỐ CỤC

Chương 1: Tổng quan.

Chương 2: Cơ sở lý thuyết.

Chương 3: Thiết kệ mô hình nhận diện đeo khẩu trang.

Chương 4: Thiết kế hệ thống nhận diện đeo khẩu trang.

Chương 5: Kết luận và hướng phát triển.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. TỔNG QUAN VỀ THUẬT TOÁN OBJECT DETECTION

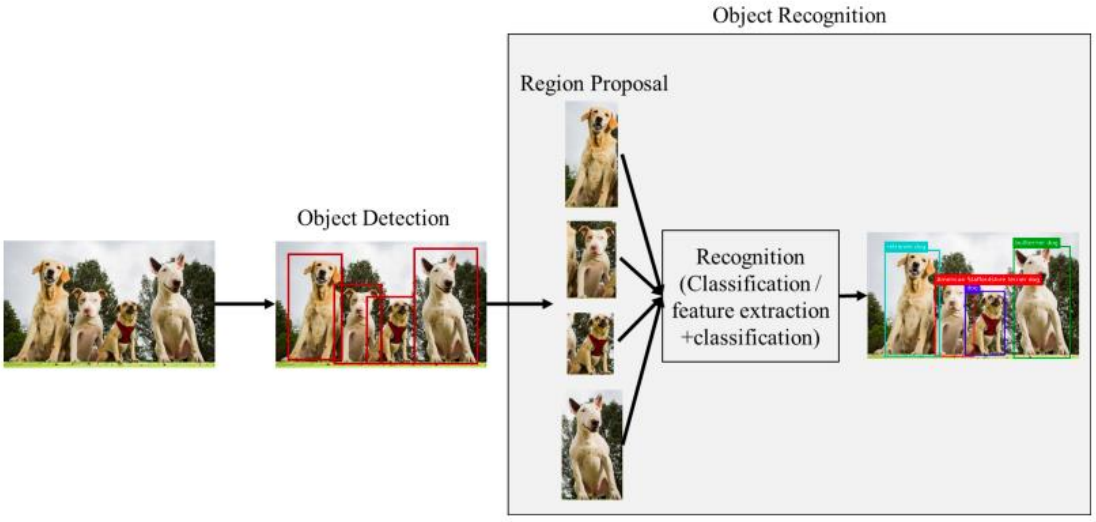
2.1.1. Khái niệm

Object detection là một kỹ thuật của lĩnh vực thị giác máy tính và xử lí ảnh dùng để xác định cụ thể một đối tượng định rõ(ví dụ như là người, động vật, xe, …) trong ảnh số và video. Các bài toán trong kỹ thuật object detection bao gồm multi-categories detection, edge detection, salient object detection, pose detection, scene text detection, face detection, pedestrian detection, …

2.1.2. Phân loại thuật toán Object detection

Object Detection có thể phân ra làm hai loại, loại thứ nhất được gọi là two-stage detector với các thuật toán tiêu biểu RCNN, Fast-RCNN, Faster-RCNN, Mask-RCNN và loại thứ hai gọi là one-stage detector với 1 số model điển hình như: SSD, Yolo, RetinaNet,…

Các thuật toán two-stage có hai giai đoạn. Giai đoạn đầu tiên, mô hình sẽ trích xuất các vùng đặc trưng (các vùng có khả năng chứa các đối tượng). Sau khi hoàn tất giai đoạn một, các vùng trích xuất thu được sẽ được mô hình thực hiện tiếp để đưa ra kết quả phân loại đối tượng và xác định vị trí



Hình 1: Hình ảnh minh hoạ hai giai đoạn của two-stage detector.

Ở thuật toán one-stage, chúng ta không cần tới giai đoạn rút trích các vùng đặc trưng mà thay vào đó chúng ta sẽ tự định nghĩa các vùng đặc trưng các đối tượng gọi

là anchor. Sau đó mô hình sẽ dựa trên các anchor để bắt đầu dự đoán, có thể nói đây là bài toán regression (với 4 tọa độ offset, ví dụ x, y, w, h). Tuy mô hình này thường đánh đổi độ chính xác để tốc độ nhanh hơn so với two-stage, nhưng một số mô hình one-stage vẫn tỏ ra nổi trội hơn 1 chút như là Retina-Net.

2.1.3. Kiến trúc của Object Detection

Backbone network có vai trò rút trích các đặc trưng cơ bản từ bức ảnh đầu vào. Hầu hết các Backbone network cho các bài toán nhận dạng đều cho ra một fully connected layers.

Khi hướng đến yêu cầu về chính xác hoặc hiệu suất, chúng ta sẽ chọn các backbone khác nhau để đáp ứng. Những Backbone có nhiều lớp dày đặc và phức tạp như ResNet [1], ResNeXt [2], AmoebaNet [3] hay các Backbone có các trọng số nhẹ hơn (lightweight backbones) như là MobileNet [4], MobileNetV2 [5], ShuffleNet [6],… Đối với các thiết bị di động, lightweight backbones gần như đáp ứng được yêu cầu. Còn đối với nhu cầu về độ chính xác cao, Backbone cần phải phức tạp hơn. Tức là, việc phát hiện thời gian thực thông qua video hoặc webcam không chỉ đòi hỏi tốc độ xử lí cao mà còn phải có độ chính xác lớn [7] , muốn đáp ứng được thì phải có được một backbone được thiết kế rất tỉ mĩ và rất tốt để thích ứng với hệ thống cũng như tạo ra sự cân bằng giữa độ chính xác và tốc độ.

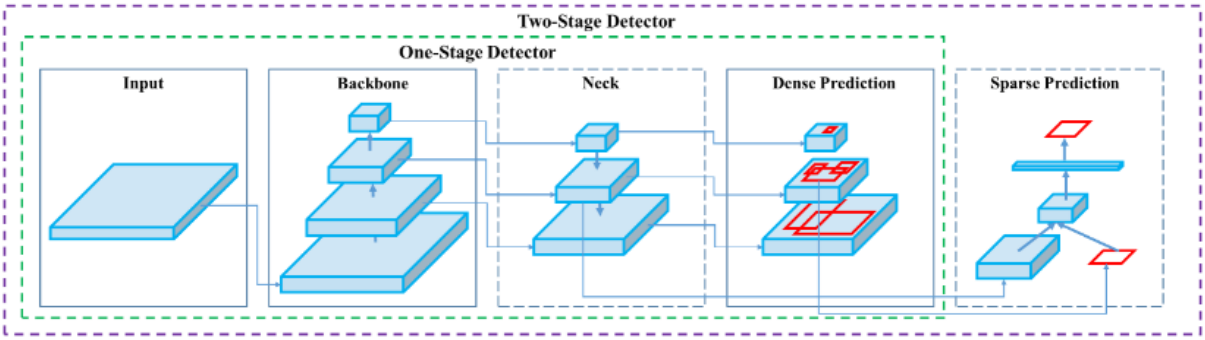
Các mạng phân loại có độ hiệu quả cao ngày nay có thể làm giảm đi độ phức tạp của việc phát hiện đối tượng cũng như cải thiện được độ chính xác. Đây là một cách hiệu quả để cải thiện chất lượng của mạng phân loại vì backbone có vai trò rút trích đặc trưng của đối tượng. Như đã biết, chất lượng của phân loại quyết định chủ yếu dựa trên hiệu suất của mạng mà ta có, do đó đây là một bước quan trọng chúng ta có thể khám phá. Chúng ta có thể tham khảo ở [8].

Head: Phần head được sử dụng để tăng khả năng phân biệt đặc trưng để dự đoán class và bounding-box. Ở phần head có thể áp dụng 1 tầng hoặc 2 tầng:

Tầng 1: Dense Prediction, dự đoán trên toàn bộ hình với các mô hình RPN, YOLO, SSD,...

Tầng 2: Sparse Prediction dự đoán với từng mảng được dự đoán có vật thể với các mô hình R-CNN series,..

Neck: Ở phần giữa Backbone và Head, thường có thêm một phần Neck. Neck thường được dùng để làm giàu thông tin bằng cách kết hợp thông tin giữa quá trình bottom-up và quá trình top-down (do có một số thông tin quá nhỏ khi đi qua quá trình bottom-up bị mất mát nên quá trình top-down không tái tạo lại được).



Hình 2: Minh hoạ kiến trúc các thuật toán Object Detection.

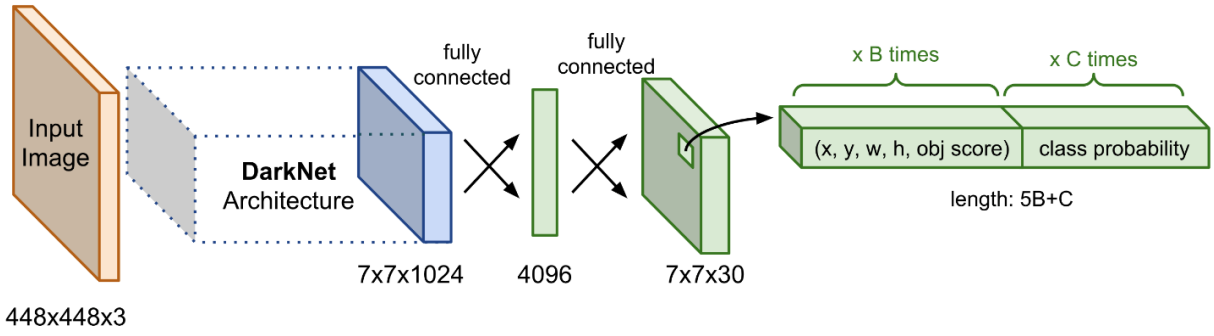
2.1.4. Đánh giá mô hình bằng mAP

Đối với mỗi model Object detection sau khi đào tạo, cần có những thang điểm để đánh giá sự chính xác của nó. Hầu như các bài báo cũng như các trang web lớn thường sử dụng mAP làm thước đo chính.

2.2. LÝ THUYẾT VỀ YOLO

2.2.1. Kiến trúc chung mạng YOLO

Kiến trúc YOLO bao gồm: Một lớp Backbone với phần phía sau là những Extra Layers được áp dụng để phát hiện vật thể trên feature map của Backbone network. Backbone network của YOLO sử dụng chủ yếu là các convolutional layer và các fully conntected layer. Các kiến trúc YOLO cũng khá đa dạng và có thể tùy biến thành các version cho nhiều input shape khác nhau.



Hình 3: Sơ đồ kiến trúc mạng YOLO [9].

Thành phần Darknet Architechture được gọi là Backbone network có tác dụng trích suất đặc trưng. Output của Backbone network là một feature map có kích thước 7x7x1024 sẽ được sử dụng làm input cho các Extra layers có tác dụng dự đoán nhãn và tọa độ bounding box của vật thể.

Trong YOLOv3, tác giả áp dụng một mạng feature extractor là darknet-53. Mạng này gồm 53 convolutional layers kết nối liên tiếp, mỗi layer được theo sau bởi một batch normalization và một activation Leaky Relu. Để giảm kích thước của output sau mỗi convolution layer, tác giả down sample bằng các filter với kích thước là 2. Mẹo này có tác dụng giảm thiểu số lượng tham số cho mô hình.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Hình 4: Các layer trong mạng Darknet-53 [10].

Các bức ảnh khi được đưa vào mô hình sẽ được scale để về chung một kích thước phù hợp với input shape của mô hình và sau đó được gom lại thành batch đưa vào huấn luyện.

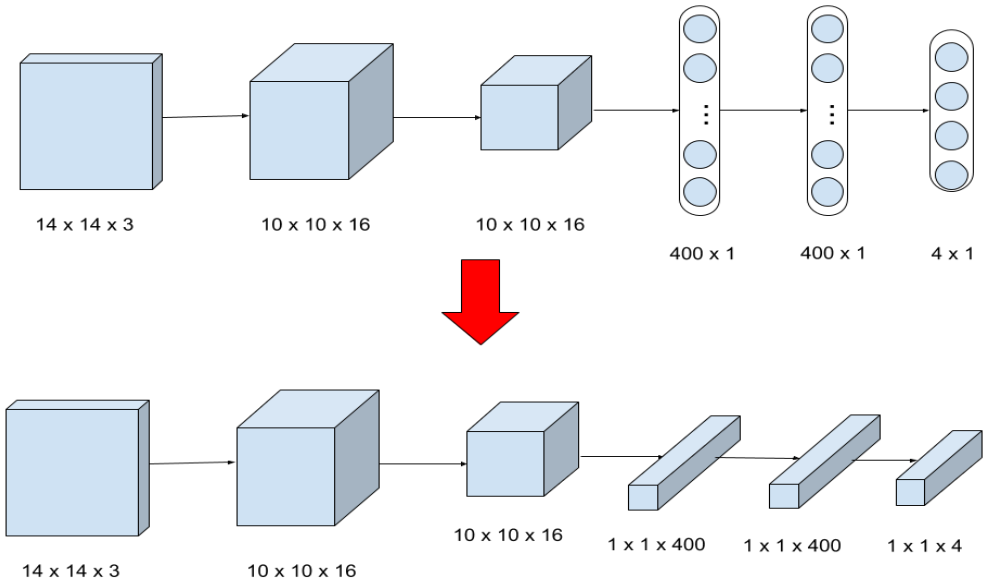
Hiện tại YOLO đang hỗ trợ 2 đầu vào chính là 416x416 và 608x608. Mỗi một đầu vào sẽ có một thiết kế các layer riêng phù hợp với shape của input. Sau khi đi qua các layer convolutional thì shape giảm dần theo cấp số nhân là 2. Cuối cùng ta thu được một feature map có kích thước tương đối nhỏ để dự báo vật thể trên từng ô của feature map.

Kích thước của feature map sẽ phụ thuộc vào đầu vào. Đối với input 416x416 thì feature map có các kích thước là 13x13, 26x26 và 52x52. Và khi input là 608x608 sẽ tạo ra feature map 19x19, 38x38, 72x72

2.2.2. Grid System

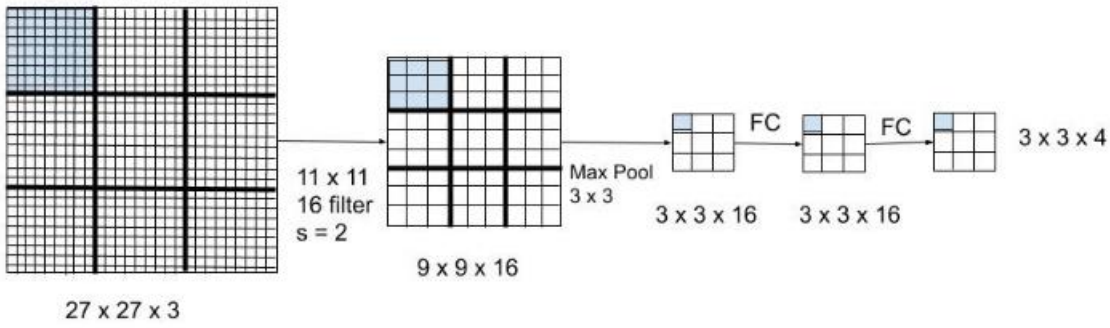
Ý tưởng của việc thực hiện Grid System xuất phát từ việc thay vì thực hiện các lớp Fully Connected ở cuối mô hình thì các lớp này sẽ được chuyển thành các lớp tích chập như các lớp phía trước trong bài toán phân loại vật thể. Khi đó ở lớp cuối cùng thay cho lớp fully connected có kích thước (số classes, 1) thành một lớp tích chập có kích thước 3 chiều (1,1,số classes)

Việc chuyển đổi này không làm ảnh hưởng đến kết quả dự đoán mà còn thể hiện được chúng ta hoàn toàn có thể phân loại vật thể bằng các phép toán tích chập với kết quả dự đoán nằm ở lớp tích chập cuối cùng và còn ưu việt hơn khi vẫn giữ được vị trí của vật thể. Việc chuyển đổi diễn ra như sau, ví dụ, hình ảnh chứa vật cần phân loại có kích thước 14 x 14 x 3, sau khi thực hiện các bước tích chập nhận được lớp tích chập cuối cùng có kích thước 1 x 1 x 4, chính lớp tích chập mang kết quả phân loại vật thể.



Hình 5: Chuyển các lớp Fully Connected cuối thành các lớp tích chập.

Để chứng minh vị trí vật thể không thay đổi và vẫn có thể xác định được qua lớp cuối ta giả sử hình ảnh đang xét có kích thước 27 x 27 x 3 được chia thành 3 x 3 grid như hình dưới, vị trí của vật thể nằm ở ô có đánh màu, sau khi thực hiện các bước tích chập, ta thu được lớp tích chập cuối cùng có kích thước 3 x 3 x 3.



Hình 6: Vị trí và thông tin của vật thể được duy trì đến lúc cuối cùng.

Ta có thể thấy sau khi thực hiện tích chập dữ liệu và vị trí của vật thể được duy trì cho đến lớp cuối cùng, và ở ô màu tương ứng mang kết quả phân loại của vật thể. Như vậy ta vừa có thể phân loại vật thể vừa xác định được vị trí của vật thể.

Qua đó, grid system sẽ chia hình gốc thành số grid tương đương với kích thước của lớp cuối (không đề cập đến số chiều), như ví dụ trên lớp cuối có kích thước 3 x 3 vậy ta sẽ chia hình gốc thành 3 x 3 grid (đường kẻ đậm). Tại đó mỗi grid sẽ mang 3

thông tin chính: Grid có đang chứa vật thể hay không, tọa độ của các bounding box (gồm tọa độ x,y của góc trên bên trái và chiều dài, chiều rộng của bounding box), xác suất phân loại vật thể. Xét hình 2.15, giả sử chúng ta cần nhận diện biển báo cấm vượt quá 30km/h, mặc dù biển báo không nằm trọn trong một grid nhưng thuật toán sẽ chỉ xác định tâm vật thể (hình tròn trong hình dưới) và tâm vật thể nằm ở grid nào thì grid đó sẽ được xác định tồn tại vật thể. Ma trận của 1 grid như sau:

$$[\ p_c\ ,\ b_x\ ,\ b_y\ ,\ b_h\ ,\ b_w\ ,\ c_1\ ,\ c_2\ ,\ c_3\ ,\ ...\ ]$$

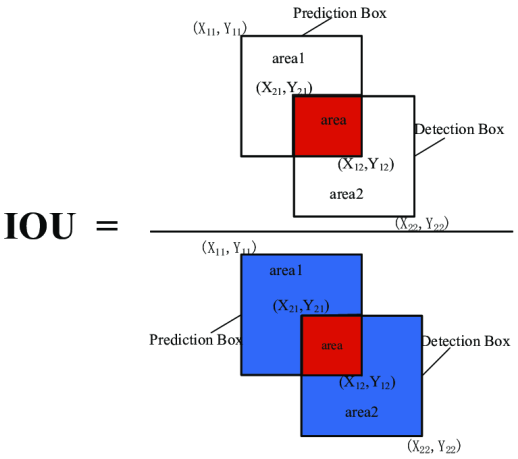
Hình 7: Mô tả grid system trong bài toán nhận diện biển báo.

2.2.3. Khái niệm về chỉ số IoU và thuật toán Non-max suppression

Chỉ số IoU (Intersection over Union) cho ta biết tỉ lệ trùng vào nhau của 2 box.

$$IoU = (box\ A \cap box\ B)/(box\ A \cup box\ B)$$

Trong đó $A \cap B$ là phần giao nhau (Intersection) của box A và box B, $A \cup B$ là phần chung của 2 box (Union) bằng tổng diện tích của 2 box trừ đi phần giao nhau. Việc xác định IoU giúp tính toán khả năng phát hiện chính xác vật thể, trong đó box A thường là các anchor box (hay groundtruth bounding box) được gán nhãn ở pha huấn luyện và box B là bounding box của hệ thống xác định ở pha kiểm tra. Tính toán IoU để đánh giá mô hình đã phát hiện vật thể đúng hay chưa.



Hình 8: Phần giao nhau giữa 2 box A và B.

Trong pha kiểm tra, hệ thống sẽ đưa ra nhiều bounding box khác nhau với các xác suất dự đoán khác nhau và tỉ số IoU khác nhau, vì vậy thuật toán Non-max suppression giúp loại bỏ các bounding box có tỷ lệ dự đoán thấp và chỉ giữ lại 1 bounding box cuối cùng có tỷ lệ dự đoán cao nhất. Thuật toán Non-max suppression diễn ra như sau:

Bước 1: Loại bỏ tất cả các bounding box có xác suất xuất hiện của vật thể p thấp hơn ngưỡng. Việc loại bỏ như vậy để các grid không chứa vật thể có xác suất xuất hiện của vật thể thấp sẽ không hiển thị bounding box.

Bước 2: Chọn các bounding box có xác suất xuất hiện vật thể cao nhất.

Bước 3: Nếu có nhiều bounding box có cùng xác suất xuất hiện vật thể cao nhất thì ta sẽ loại bỏ bằng IoU, bounding box nào có chỉ số IoU thấp hơn ngưỡng sẽ bị loại bỏ. Kết thúc bước ba, ta sẽ nhận được bounding box có tỉ lệ nhận diện vật thể tốt nhất.

2.2.4. Gán nhãn cho các mẫu

Tương tự các bài toán "Máy học"/"Học sâu", bài toán nhận diện vật thể cũng gồm có 2 pha, pha kiểm tra và pha huấn luyện. Trong pha học của thuật toán YOLO, chúng ta sẽ thực hiện label vị trí và phân loại cho vật thể, khi đó y có dạng như sau:

$y = [p_c, bx, by, bh, bw, c1, c2, c3, ...]$

Ví dụ đối với hình 7, giả sử bài toán cần tìm vị trí vật thể và phân loại với 3 nhãn, trong đó biển báo của hình thuộc nhãn thứ nhất. Khi đó, ở grid phát hiện được biển báo giao thông, y sẽ được gán như sau y = [1 , 230 , 120 , 20 , 30 , 1 , 0 , 0], tức là grid

đó có vật thể, tọa độ (x,y) ở phía trên bên trái của anchor box là (230,120), anchor box có chiều dài là 20 chiều rộng là 30, biển thuộc nhãn 1 và không thuộc hai nhãn còn lại. Nếu xét grid không chứa vật thể y sẽ được gán y = [0 , x , x , x , x , x , x , x] tức là không có vật thể ở grid đó và các giá trị còn lại không cần quan tâm.

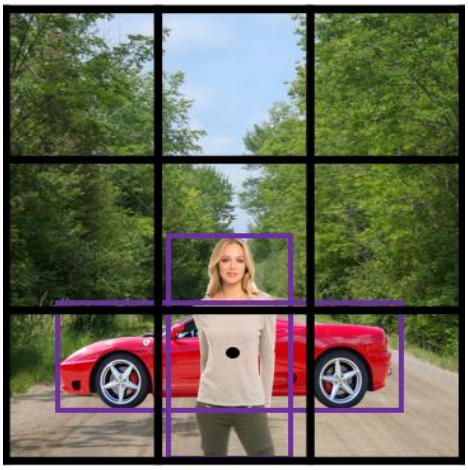
Tuy nhiên, nếu có cùng 2 vật thể cùng xuất hiện trong 1 grid thì không thể thực hiện gán y như trên. Với vấn đề đó, chúng ta có thể xử lý theo 2 cách:

Cách 1: Chia nhỏ grid ra đến khi nào 2 vật thể nằm ở 2 grid khác nhau. Tuy nhiên nếu chia càng nhỏ grid, việc học không thể diễn ra sâu khiến việc phân loại vật thể khó chính xác do các lớp sau không học được các đặc trưng cao.Và nếu tâm của 2 vật thể gần như trùng nhau cũng không thể giải quyết được như trên. Khi đó, ta phải thực hiện theo cách thứ 2.

Cách 2: Thay vì y chỉ được gán cho 1 vật thể, y sẽ được mở rộng ra với nhiều vật thể như sau:

y =[p_{c1}, bx1 , by1 , bh1 , bw1 , c1 , c2 , c3,..., p_{c2}, bx2 , by2 , bh2 , bw2 , c1 , c2 , c3,...]

Xét ví dụ với hình 9 bên dưới, cô gái và chiếc xe đều nằm cùng chung 1 grid. Khi đó, y sẽ được gán như sau y = [1, 120, 20, 20 , 120 , 1 , 0 , 1 , 90, 50 , 90 , 20 , 0 , 1] . Tức là ở grid này có xuất hiện ô tô (y[0] = 1), (120,20,20,120) là 4 thông số anchor box của xe ô tô, (y[5]=1,y[6]=0) để phân loại cho anchor box này là xe ô tô chứ không phải cô gái, tương tự y[7]=1 có nghĩa grid này cũng có cô gái, 4 thông số tiếp theo để xác định tọa độ anchor box cho cô gái cà (y[12]=0,y[13]=1) để phân loại anchor box này là cô gái chứ không phải xe ô tô. Như vậy, với cách 2 ta sẽ ghép y của 2 hay nhiều vật thể nằm trong cùng 1 grid lại thành một, y[0:6] để xác định cho việc phát hiện chiếc xe, và y[7:13] để xác định cho cô gái. Tuy nhiên với cách thứ 2, nếu ta ghép càng nhiều tốc độ xử lý càng lâu do càng có nhiều phép toán phép thực hiện, vì vậy không nên quá lạm dụng cách này mà nên phối hợp hài hòa với cách thứ 1, tăng số grid phải chia lên.



Hình 9: Tâm của 2 vật thể trùng nhau và cùng nằm trong 1 grid.

2.2.5. Hàm mất mát

Sau khi gán nhãn cho toàn bộ tập dữ liệu, các mẫu dữ liệu sẽ được đưa qua mạng CNN (toàn bộ ảnh sẽ đưa vào mạng CNN) để thực hiện việc học các tham số. Trong quá trình này, các hàm mất mát sẽ được tính toán. Đối với bài toán nhận diện vật thể sẽ cần tính toán 3 hàm mất mát.

Hàm mất mát phân loại (Classification Loss Function):

$$L_{classification} = \sum_{i=0}^{S^2} \prod_{i}^{obj} \sum_{c \in class} (p_i(c) - \hat{p_i}\,(c))^2$$

Hàm mất mát vị trí (Localization Loss Function): Được sử dụng để tính toán độ sai số giữa các bounding box dự đoán với các anchor box, cải thiện Localization Loss sẽ giúp việc phát hiện vật thể trở nên chính xác hơn:

$$L_{localization} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \prod_{ij}^{obj} [(offsetx_i - \hat{offsetx_i})^2 + (offsety_i - \hat{offsety_i})^2 + (width_i - \hat{width_i})^2 + (height_i - \hat{height_i})^2$$

Hàm mất mát dự đoán (Confidence Loss Function): thể hiện sai số giữa dự đoán của bounding box với nhãn thực tế:

$$L_{confidence} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \prod_{ij}^{obj} (C_i - \hat{C_i})^2 + \lambda_{noobject} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \prod_{ij}^{noobj} (C_i - \hat{C_i})^2$$

Hàm mất mát tổng quát sẽ bằng tổng 3 hàm mất mát phía trên:

$$L_{total} = L_{classification} + L_{localization} + L_{confidence}$$

2.2.6. Ngõ ra

Ở ngõ ra, mỗi grid sẽ thực hiện dự đoán 2 bounding box có p_c cao nhất. Loại bỏ tất cả bounding box có p_c thấp trên toàn bộ bức ảnh. Và cuối cùng thuật toán Non-max suppression sẽ được thực hiện để giữ lại bounding box chính xác nhất cho từng vật thể.

Hình 10: Thuật toán được áp dụng ở ngõ ra.

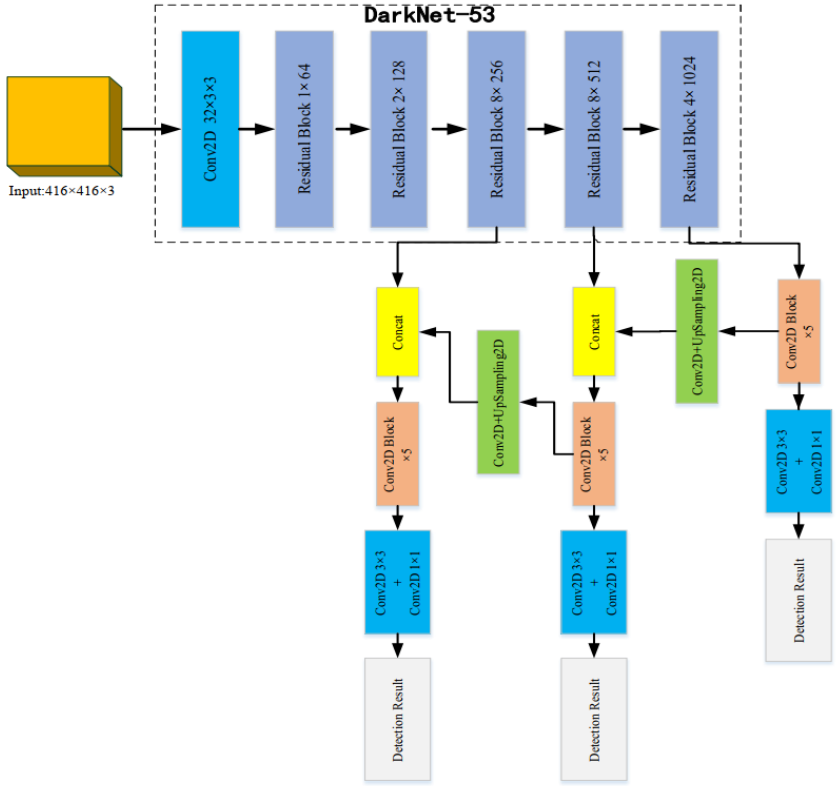
(a) (b) (c)

(a): Với mỗi grid, lấy 2 bounding box có p_c cao nhất.

(b): Loại bỏ tất cả các bounding box có p_c thấp trên toàn bộ bức ảnh, bước này giúp loại bỏ các grid không chứa vật thể.

(c): Áp dụng non-max suppression để chỉ chọn ra bounding-box cuối cùng có xác suất dự đoán cao nhất.

2.2.7. Kiến trúc YOLO V3 và các vận hành



Hình 11: Hình ảnh minh hoạ kiến trúc YOLO V3 [11].

YOLO V3 sử dụng kiến trúc DarkNet-53 làm backbone (có 53 lớp tích chập), và bản thân YOLO V3 có thêm 53 lớp tích chập nữa. Tổng cộng chúng ta có 106 lớp tích chập.

YOLOV3 sẽ đưa ra dự đoán theo các bước sau:

Đầu tiên, bức ảnh có kích thước 416x416 sẽ là đầu vào của kiến trúc DarkNet-53. Trải qua nhiều xử lí tích chập, một feature map ta thu được có kích thước 13x13, và qua 7 lần với 2 convolution kernels có kích thước 1x1 và 3x3, ta thu được những dự đoán đầu tiên.

Feature map với kích thước 13x13 sẽ được trải qua 5 lần với 2 convolution kernels có kích thước là 1x1 và 3x3, sau đó thực hiện 1 lần convolution kernels có kích thước 1x1 theo sau đó là 2 lần Upsampling layer, thu được một feature map được khuếch đại lên 26x26. Sau đó lớp thu được sẽ đi qua 7 lần với 2 convolution kernels như ở ý trên thu được dự đoán thứ 2.

Feature map có kích thước 26x26 sẽ qua 5 lần với 2 lớp convolution kernels có kích thước là 1x1 và 3x3, sau đó 2 lần Upsampling layer, thu được kích thước 52x52. Sau đó lại trải qua 7 lần với 2 convolution kernels để thu được kết quả lần thứ 3.

2.2.8. Đánh giá YOLOv3

Dựa trên bài báo [12] đã đưa ra một số so sánh về cấu trúc YOLO v3 cùng với một số thuật toán Object Detection khác như là SSD513, RetinaNet, Faster RCNN ở Bảng 3 cùng với đồ thị 3, ta có thể nói YOLO v3 là một mô hình phát hiện ổn. Cân bằng giữa độ chính xác cùng với tốc độ phát hiện. Tuy ở tập dữ liệu COCO không đạt được kết quả cao ở thang điểm AP giữa .5 và .95 chỉ số IOU. Nhưng dù vậy mô hình có thể sánh ngang với SSD với số điểm gấp 3 lần và thua thiệt một chút so với mô hình RetinaNet. Nhưng nếu ta nhìn vào thời gian thực thi thì YOLO v3 lại vượt trội hơn hẳn các mô hình khác. Điều này cho thấy YOLO v3 là một mô hình rất mạnh về tác vụ có thể phát hiện và cho các đối tượng vào box để xuất kết quả.

2.3. GIỚI THIỆU PHẦN CỨNG

2.3.1. Giới thiệu ESP32 CAM AI THINKER

ESP32-CAM AI-Thinker là phiên bản nâng cao của ESP8266-01 được Espressif sản xuất ra với nhiều tính năng. ESP32-CAM đã tích hợp với Wi-Fi, Bluetooth và có thể được sử dụng với máy ảnh OV2640 hoặc OV7670. IC ESP32 có các giao thức ADC, SPI, I2C và UART độ phân giải cao để giao tiếp thông tin. Mô-đun có cảm biến Hall, cảm biến nhiệt độ, cảm biến cảm ứng và bộ hẹn giờ cơ quan giám sát. RTC có thể hoạt động ở các chế độ khác nhau. Mô-đun có tần số xung nhịp tối đa là 160 MHz có nghĩa là khả năng tính toán lên đến 600 DMPIS. Hơn nữa, nó khá bền và đáng tin cậy khi nói đến kết nối internet.

2.3.2. Các thành phần AI-Thinker của ESP32-CAM

Mô-đun ESP32-CAM AI-Thinker bao gồm các phần khác nhau. Các khu vực này được mô tả bên dưới:

ESP32-S Chip : Module này là một con chip chính chứa hai hiệu suất cao 32-bit CPU LX6 với một kiến trúc đường ống 7 giai đoạn và sử dụng cho tất cả các quá trình xử lý và hoạt động.

Đầu ra khối IPEX : IPEX được in kết nối ăng-ten GSM để truyền tín hiệu.

Tụ tantali : Tụ tantali chủ yếu được sử dụng trên các mô-đun kích thước nhỏ. Chúng bền và cung cấp khả năng lọc nguồn điện cho số lượng tín hiệu tốt.

Nút đặt lại : Khi được nhấn, nút đặt lại sẽ khởi động lại mã được thực thi trên mô-đun.

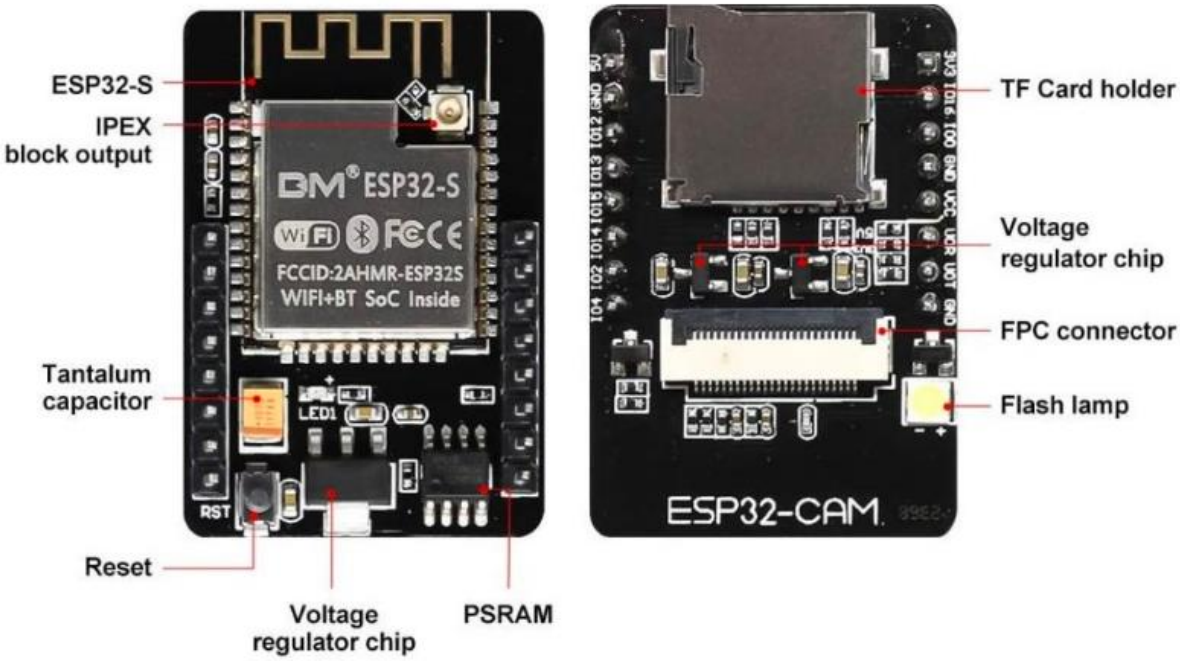
Chip điều chỉnh điện áp : Chip điều chỉnh điện áp trên mô-đun duy trì điện áp đầu ra bất chấp sự biến động của nguồn cung cấp đầu vào. Nó điều chỉnh điện áp đến 3,3 volt.

PSRAM : Bộ nhớ truy cập giả ngẫu nhiên công suất thấp 4MB được tích hợp trong mô-đun để xử lý nhanh các hướng dẫn. Nó giúp máy ảnh chạy mượt mà.

Giá đỡ thẻ TF : Dòng ESP32 được nhúng với một giá đỡ thẻ micro-SD để lưu trữ dữ liệu. Tất cả quá trình truyền diễn ra thông qua Giao diện ngoại vi nối tiếp.

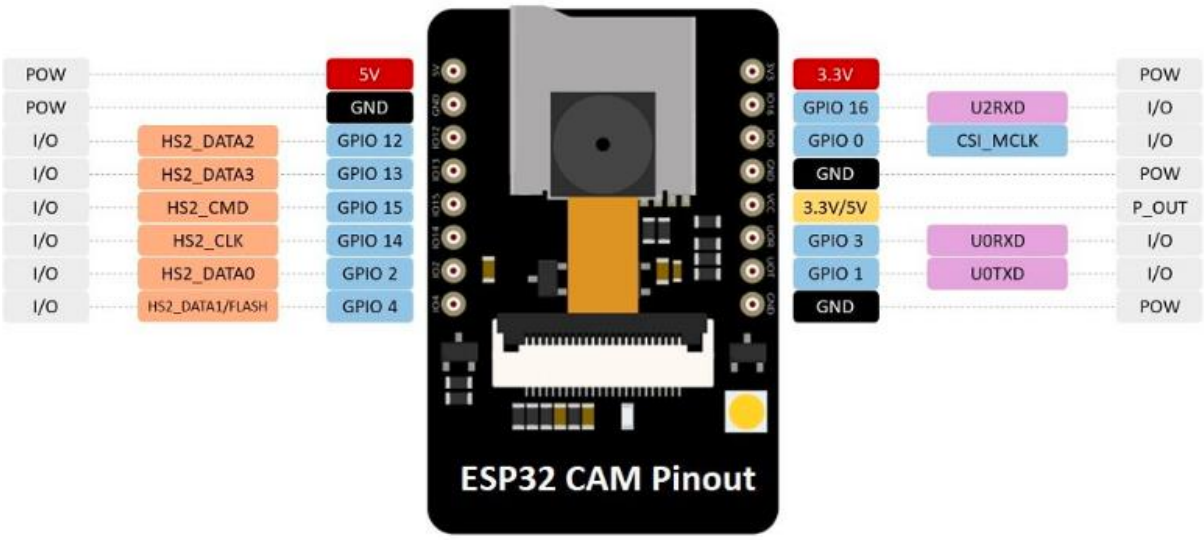
Đầu nối FPC : Để gắn máy ảnh, mô-đun ESP32 chứa một đầu nối mạch in linh hoạt. Cao độ tốt của chúng chịu trách nhiệm về độ tin cậy của tín hiệu.

Đèn flash : Đèn flash tạo ra các xung điện hoạt động như một đèn flash cho máy ảnh để máy ảnh có thể chụp được những hình ảnh rõ nét.



Hình 12: Các thành phần của ESP32-CAM.

2.3.3. Các chân trong ESP32 CAM AI THINKER



Hình 13: Hình minh hoạ các chân của ESP32.

Chân nguồn: Mô-đun có ba chân nối đất và hai chân cấp nguồn dương như chân 5V và 3,3V. Các chân này có thể được sử dụng để cấp nguồn cho mô-đun ESP32-CAM AI-Thinker. Nhưng không nên cấp nguồn cho sự phát triển này giáp với chân 3,3V, nó không cung cấp nguồn ổn định cho bo mạch.

Pin đầu ra nguồn: ESP32-CAM cũng cung cấp một chân đầu ra nguồn như được hiển thị bằng màu vàng trong sơ đồ sơ đồ chân ở trên. Đây là chân VCC có thể xuất ra 5V hoặc 3.3V. Theo kết nối Jumper trên ESP32-CAM, chân VCC cung cấp đầu ra 3.3V.

Chân UART: Hầu như tất cả các chân GPIO của ESP32-CAM đều là chân đa năng. GPIO1 và GPIO3 lần lượt có các chức năng thay thế cho việc truyền và nhận dữ liệu nối tiếp cho cổng UART. Bảng AI-Thinker không đi kèm với bộ lập trình tích hợp. Do đó, các chân UART này được sử dụng để lập trình và giao tiếp với PC để tải mã lên.

Tên ghim	Hàm số
GPIO1	U0TXD (Chân truyền UART)
GPIO3	UORXD (Chân tiếp nhận UART)

Bảng 1: Các chân UART.

Chúng ta có thể sử dụng cáp FTDI để flash mã tới ESP32-CAM bằng cách sử dụng các chân UART

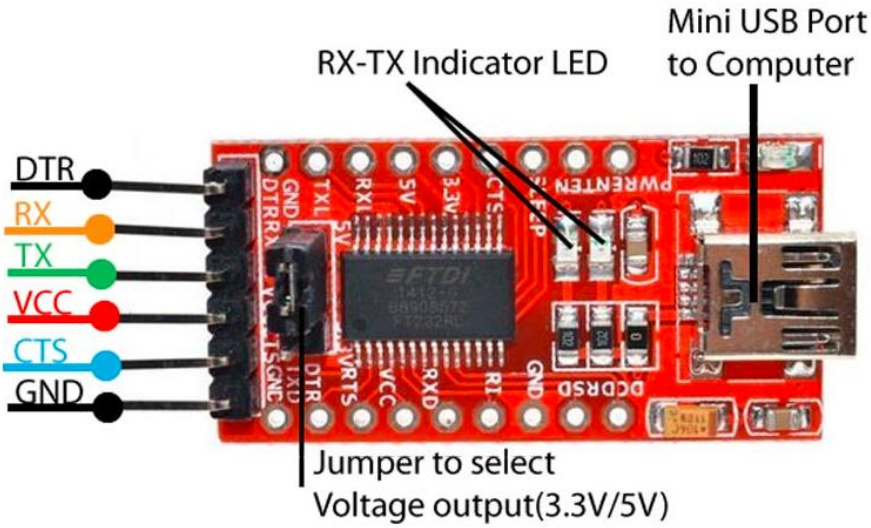
Chân kết nối máy ảnh

Bảng sau liệt kê các kết nối chân GPIO với Camera OV2640 và tên biến tương ứng của chúng mà chúng ta sẽ sử dụng trong chương trình của mình.

MÁY ẢNH OV2640	ESP32	TÊN BIẾN
D0	GPIO5	Y2_GPIO_NUM
D1	GPIO18	Y3_GPIO_NUM
D2	GPIO19	Y4_GPIO_NUM
D3	GPIO21	Y5_GPIO_NUM
D4	GPIO36	Y6_GPIO_NUM
D5	GPIO39	Y7_GPIO_NUM
D6	GPIO34	Y8_GPIO_NUM
D7	GPIO35	Y9_GPIO_NUM
XCLK	GPIO0	XCLK_GPIO_NUM
PCLK	GPIO22	PCLK_GPIO_NUM
VSYNC	GPIO25	VSYNC_GPIO_NUM
HREF	GPIO23	HREF_GPIO_NUM
SDA	GPIO 26	SIOD_GPIO_NUM
SCL	GPIO 27	SIOC_GPIO_NUM
PIN NGUỒN	GPIO 32	PWDN_GPIO_NUM

Bảng 2: Chân kết nối máy ảnh.

2.3.4. FT232RL USB TO TTL Converter



Hình 14: Module FT232RL.

FT232RL USB để TTL 3.3V / 5V FTDI Serial Adapter Module là một mô-đun phổ biến được sử dụng để kết nối một sê-ri TTL giao tiếp thiết bị với máy tính qua cổng USB mini. Mô-đun chuyển đổi này có các tùy chọn cho các tùy chọn điện áp đầu ra khác nhau được thiết lập bởi jumper trên bo mạch

Các tính năng và thông số kỹ thuật của FT232RL USB TO TTL Converter

Phần này đề cập đến một số tính năng và thông số kỹ thuật của FT232RL USB to TTL Converter:

1. Điện áp hoạt động: 5V / 3.3V DC
2. Dòng điện tối đa: 5V - 500mA; 3.3V - 50mA
3. Kết nối: Mini USB
4. Tích hợp đầy đủ bộ mô tả thiết bị lưu trữ EEPROM 1024 bit và cấu hình CBUS I / O
5. Tốc độ truyền dữ liệu từ 300 baud đến 3 Mbaud (RS422, RS485, RS232) ở mức TTL
6. Bộ đệm nhận 128 byte và bộ đệm truyền 256 byte
7. Truyền và nhận tín hiệu ổ đĩa LED

8. Thế hệ đồng hồ tích hợp đầy đủ mà không cần pha lê bên ngoài

Cấu hình chân của FT232RL USB TO TTL Converter

FT232RL USB to TTL Converter có 6 chân. Bảng dưới đây mô tả tất cả các loại pin cùng với chức năng của từng loại pin.

Loại pin	**Sự miêu tả**
DTR	Sẵn sàng đầu cuối dữ liệu (Đầu ra được sử dụng để điều khiển luồng)
RX	Nhận dữ liệu nối tiếp
TX	Truyền dữ liệu nối tiếp
VCC	Nguồn điện đầu vào
CTS	Xóa để gửi (Đầu vào được sử dụng để kiểm soát luồng)
GND	Đất

Bảng 3: Miêu tả các Pin và chức năng của từng lại Pin.

2.4. GIAO THỨC HTTP

2.4.1. Khái niệm

HTTP viết tắt của từ Hypertext Transfer Protocol được thiết kế để cho phép giao tiếp giữa client và server. Là công nghệ được sử dụng để truyền gửi files, đồ họa hoặc văn bản trên internet và về cơ bản là nền tảng của trao đổi dữ liệu cho World Wide Web. HTTP hoạt động như là một giao thức request-response giữa client và server.

2.4.2. Các phương thức HTTP

Các phương thức HTTP:

- GET
- POST
- PUT

- HEAD
- DELETE
- PATCH
- OPTIONS

Có 2 phương thức HTTP được sử dụng phổ biến nhất và nhóm sẽ đi vào phân tích những phương thức này cũng như sử dụng trong đề tài lần này là: GET và POST

2.4.2.1. Phương thức GET

GET được sử dụng để request dữ liệu từ một nguồn dữ liệu cụ thể.

Ví dụ sử dụng method GET khi bạn tạo một request tới một trang web với tham số đi kèm:

/myhttp/formtest.php?name1=value1&name2=value2

Một số lưu ý khi sử dụng method GET:

- Truy vấn GET có thể bị cache lại

- Truy vấn GET vẫn duy trì trong lịch sử trình duyệt

- Truy vấn GET có thể được đánh dấu bookmark

- Truy vấn GET không bao giờ nên được sử dụng khi xử lý các dữ liệu nhạy cảm

- Truy vấn GET có độ dài giới hạn

- Truy vấn GET chỉ có thể được sử dụng để yêu cầu dữ liệu

2.4.2.2. Phương thức POST

POST được sử dụng để gửi dữ liệu tới một server để tạo hoặc cập nhật tài nguyên trên đó.

Dữ liệu được gửi tới server với method POST được lưu trữ trong request body của truy vấn HTTP.

Một số lưu ý khi sử dụng POST:

- Truy vấn POST không bao giờ bị cached lại

- Truy vấn POST không được duy trì trong lịch sử trình duyệt

- Truy vấn POST không thể được đánh dấu bookmark

- Truy vấn POST không có giới hạn độ dài

CHƯƠNG 3:THIẾT KẾ MÔ HÌNH NHẬN DẠNG KHẨU TRANG

3.1. SƠ ĐỒ KHỐI

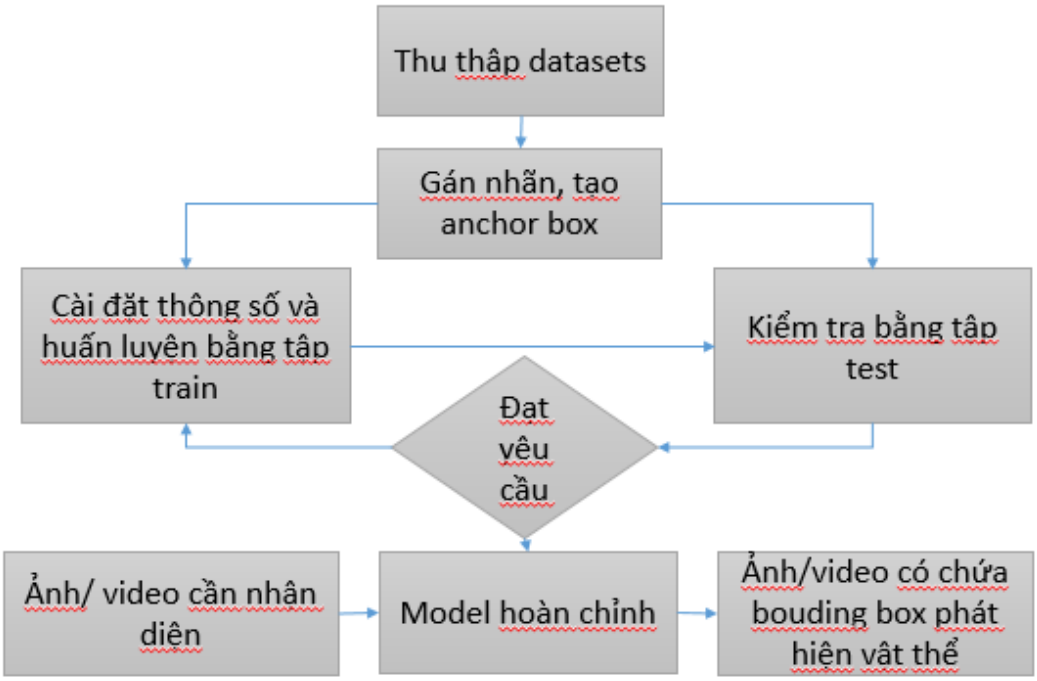
Bước 1: Tìm kiếm và thu thập datasets về các hình ảnh người đeo khẩu trang và không đeo khẩu trang, tiền xử lý và gán nhãn cho từng bức ảnh.

Bước 2: Phân chia dữ liệu thành tập train và tập test.

Bước 3: Huấn luyện mô hình

Bước 4: Đánh giá mô hình

Sơ đồ khối mô hình:



Hình 15: Sơ đồ khối mô hình nhận diện đeo khẩu trang.

3.2. DATASETS

Để huấn luyện một mô hình Deep Learning trước tiên ta cần chuẩn bị một tập dữ liệu. Tập dữ liệu là yếu tố vô cùng quan trọng trong việc huấn luyện mô hình. Vì dữ liệu ở các môi trường khác nhau (như là cường độ ánh sáng, vật thể bị méo dạng,…)

thì các pixel của các ảnh mang giá trị khác nhau. Do đó, tập dữ liệu lớn sẽ giúp mô hình học được đối tượng ở nhiều trạng thái môi trường khác nhau, từ đó tăng khả năng chính xác khi dự đoán ngoài thực tế.

Tập dữ liệu được chia làm 3 tập:

Tập training: Tập training thường có kích thước lớn nhất. Chúng thường được gán nhãn và cho trước nhãn để tính toán các loss function và cập nhật tham số để mô hình tăng độ chính xác.

Tập validation: Tập validation thường được chia ra từ tập training. Tập validation không được dùng để cập nhật các tham số của mô hình, mà nó dùng để đánh giá xem mô hình có cần điều chỉnh thông số lại hay không, có bị high bias hay high variance hay không, có thể tìm hiểu khái niệm và các giải pháp tại [13]

Tập test: Tập test được dùng để đánh giá lại mô hình có hiệu quả hay không. Dữ liệu của tập test phải chưa từng xuất hiện trong tập training để có thể đánh giá mô hình ngoài thực tế.

Ở đề tài này, chúng tôi sẽ sử dụng tập Dataset của RMFD [14]. Tập dữ liệu này chứa rất nhiều ảnh những người đeo khẩu trang được tổng hợp từ những tập Dataset khác, bao gồm:

Masked Face Detection Dataset (MFDD): Tập này chứa 24,771 ảnh người đeo khẩu trang.

Real-world Masked Face Recognition Dataset (RMFRD): Tập này chứa 5,000 ảnh của 525 người đang đeo khẩu trang, và 90,000 bức ảnh cho 525 đối tượng không đeo khẩu trang. Đây là một tập Dataset lớn nhất trong lĩnh vực nhận diện khẩu trang hiện tại.

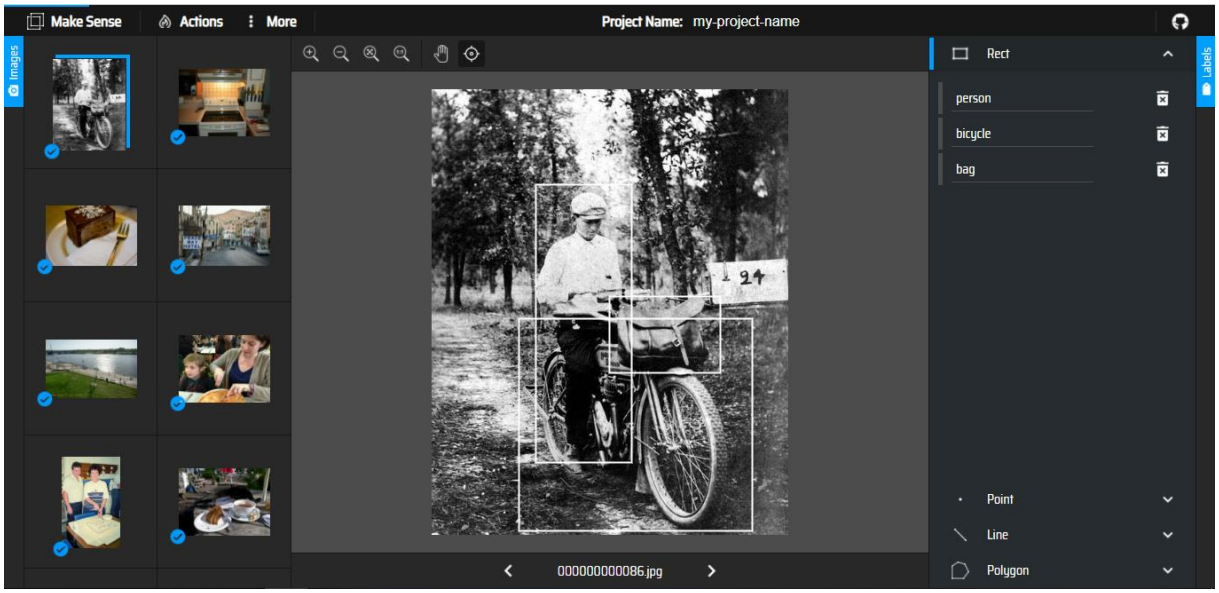
Simulated Masked Face Recognition Dataset (SMFRD): Tập này chứa 50,000 ảnh khuôn mặt từ 10,000 đối tượng bằng cách sử dụng những công cụ để tạo ra ảnh đeo khẩu trang. Ví dụ một số tấm ảnh ở tập này:

Hình 16: Một vài ảnh của tập SMFRD.

Hình 17: Một vài tập ảnh RMFD.

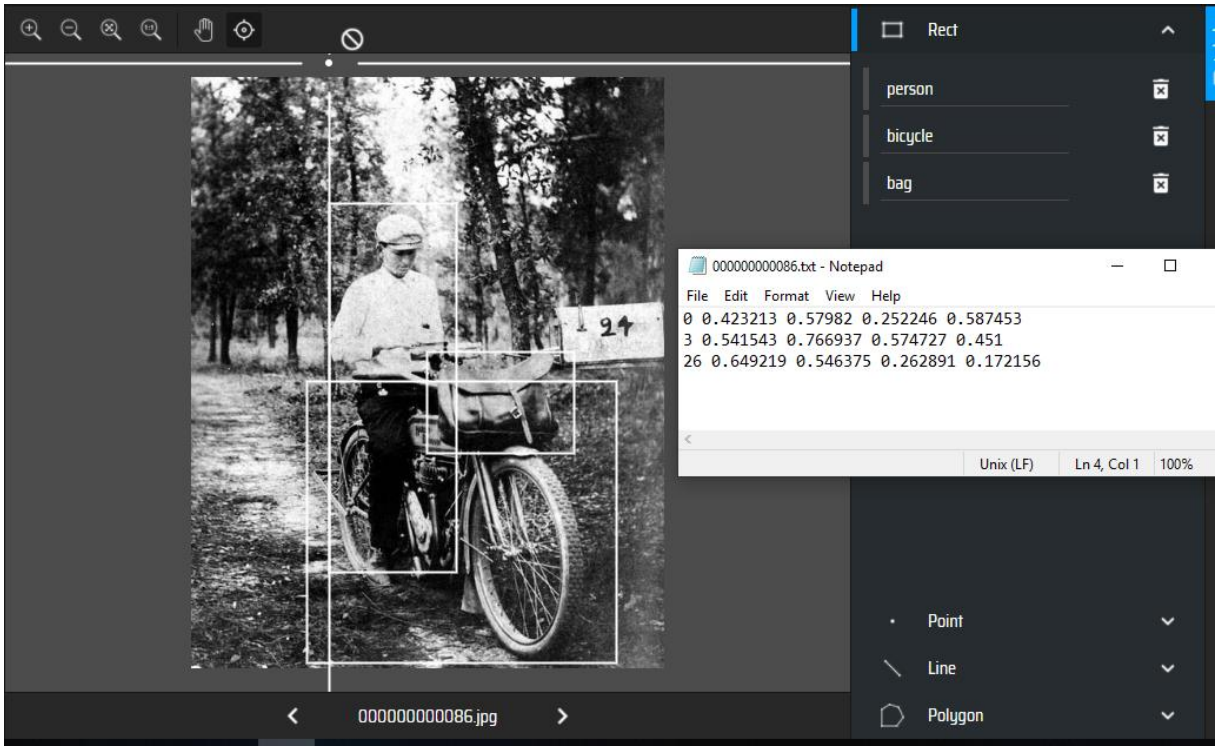
3.3. GÁN NHÃN, TẠO ANCHOR BOX

Sau khi có bộ datasets, các ảnh sẽ được xác định tọa độ của anchor box và gán nhãn bằng tools tại website: makesense.ai. Kết quả trả về sau khi gán nhãn là một file txt chứa nhãn và các tọa độ cần thiết của một anchor box.



Hình 18: Tiến hành tạo anchor box và gán nhãn cho vật thể.

Kết quả trả về sẽ là 1 file txt chứa class và tọa độ của anchor box



Hình 19: Kết quả sau khi tiến hành gán nhãn và xác định toạ độ anchor box.

3.4. CHUẨN BỊ TẬP DỮ LIỆU HUẤN LUYỆN

Các tập dữ liệu hình ảnh dùng cho đề tài này đã được để trong thư mục "data", trong thư mục này chúng ta có những thư mục và tập tin cần lưu ý như sau:

"/images" : thư mục này sẽ chứa 9,000 ảnh từ tập Dataset, bao gồm những ảnh đeo và không đeo khẩu trang.

"/label" : thư mục này sẽ chứa các tệp .txt , chứa các thông tin anchor box, phần tạo anchor box đã được giải thích ở 3.3.

2 tập tin "train.txt" và "valid.txt" được dùng để chia các hình ảnh ra làm hai tập Training và tập Validation

Các tập tin "face_mask.data" , "face_mask.names", "label.npy", train.shapes" và "valid.shapes" được dùng để cung cấp các thông tin cho thuật toán YOLO để học tập. Nếu chúng ta có bổ sung hay thay đổi về tập Dataset, chúng ta sẽ chạy tập tin "10-preparation-process.ipynb" thông qua Google Colab, hoặc Jupyter Notebook để tạo lại chúng.

3.5. HUẤN LUYỆN MÔ HÌNH

3.5.1. Kiến trúc Yolo-Fastest

Sau khi tập dataset hoàn chỉnh, chúng ta sẽ tiến hành huấn luyện mô hình. Nhóm sẽ sử dụng một thuật toán được lấy ý tưởng từ YOLO v3 nhưng được tối ưu các kiến trúc bên trong để mô hình trở nên nhanh và nhẹ hơn, đó là YOLO-Fastest của dog-qiuqiu [15]. Cụ thể, là đối với các GPU đời cũ như 1050 ti thì so với kiến trúc gốc của AlexeyAB/Darknet thì đạt được kết quả là 4ms so với 40ms, tức là hiệu năng gấp 10 lần. Và hơn hết, kiến trúc gốc của Darket chủ yếu vào các GPU nên không tối ưu cho CPU , còn kiến trúc của dog-qiuqiu đã được thay đổi để phù hợp.

Network	COCO mAP(0.5)	Resolution	Run Time(Ncnn 4xCore)	Run Time(Ncnn 1xCore)	FLOPS	Params	Weight size
Yolo-Fastest-1.1	24.40 %	320X320	5.59 ms	7.52 ms	0.252BFlops	0.35M	1.4M
Yolo-Fastest-1.1-xl	34.33 %	320X320	9.27ms	15.72ms	0.725BFlops	0.925M	3.7M
Yolov3-Tiny-Prn	33.1%	416X416	%ms	%ms	3.5BFlops	4.7M	18.8M
Yolov4-Tiny	40.2%	416X416	23.67ms	40.14ms	6.9 BFlops	5.77M	23.1M

Bảng 4: Hình ảnh đánh giá mô hình YOLO.

Dựa vào hình 20, ta có thể thấy mô hình chỉ số mAP của Yolo-Fastest nằm giữa hai mô hình Yolo v3 và Yolo v4. Nhưng mấu chốt ta thấy mô hình của Yolo-Fastest lại vượt trội hơn hai mô hình còn lại ở những điểm:

Đầu tiên xét về thời gian thực thi trên các phần cứng (tác giả đã test trên platform Mi 11 Snapdragon 888 CPU), mô hình đã dự đoán và đưa ra kết quả trong thời gian rất ngắn, tiết kiệm được xấp xỉ gấp 3 lần thời gian Yolov4 đưa ra kết quả

FLOPS (FLoating-point Operations Per Second): thời gian thực thi các phép tính rất ít, phù hợp cho các phần cứng bị hạn chế về mặt tài nguyên. Ngoài hai mô hình trên, ta có thể so sánh với một số thuật toán khác phía dưới:

Các thông số và khối lượng mô hình: đã giảm đi rất nhiều, tiết kiệm được rất nhiều bộ nhớ cho phần cứng.

Vậy nên mô hình này rất phù hợp cho những bài toán nhận diện vật thể đơn giản đối với những phần cứng hạn chế. Tiếp theo, chúng ta sẽ xem thời gian thực thi khi mô hình này được triển lên những phần cứng khác nhau.

Equipment	Computing backend	System	Framework	Run time
Mi 11	Snapdragon 888	Android(arm64)	ncnn	5.59ms
Mate 30	Kirin 990	Android(arm64)	ncnn	6.12ms
Meizu 16	Snapdragon 845	Android(arm64)	ncnn	7.72ms
Development board	Snapdragon 835(Monkey version)	Android(arm64)	ncnn	20.52ms
Development board	RK3399	Linux(arm64)	ncnn	35.04ms
Raspberrypi 3B	4xCortex-A53	Linux(arm64)	ncnn	62.31ms
Orangepi Zero Lts	H2+ 4xCortex-A7	Linux(armv7)	ncnn	550ms
Nvidia	Gtx 1050ti	Ubuntu(x64)	darknet	4.73ms
Intel	i7-8700	Ubuntu(x64)	ncnn	5.78ms

Bảng 5: Thời gian thực thi trên nhiều phần cứng khác nhau.

Ngoài ra, chúng ta có thể xem qua sự so sánh một số thuật toán nhận diện khác ở phía dưới:

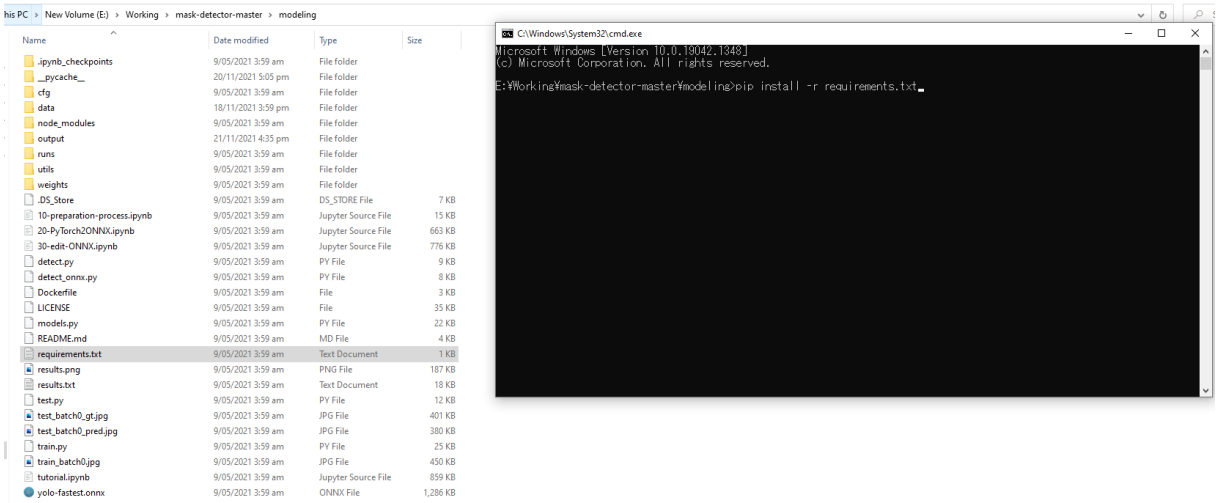
Network	Model Size	mAP(VOC 2007)	FLOPS
Tiny YOLOv2	60.5MB	57.1%	6.97BFlops
Tiny YOLOv3	33.4MB	58.4%	5.52BFlops
YOLO Nano	4.0MB	69.1%	4.51Bflops
MobileNetv2-SSD-Lite	13.8MB	68.6%	&Bflops
MobileNetV2-YOLOv3	11.52MB	70.20%	2.02Bflos
Pelee-SSD	21.68MB	70.09%	2.40Bflos
Yolo Fastest	1.3MB	61.02%	0.23Bflops
Yolo Fastest-XL	3.5MB	69.43%	0.70Bflops
MobileNetv2-Yolo-Lite	8.0MB	73.26%	1.80Bflops

Bảng 6: Đánh giá mô hình YOLO Fastest và mô hình nhận diện khác.

3.5.2. Thực hiện huấn luyện

Trước khi tiến hành huấn luyện mô hình, chúng ta sẽ tiến hành cài Python 3.8 và các thư viện cần thiết. Để cài thư viện một cách tự động hãy mở Command và dẫn tới thư mục chứa các tập tin cần thiết và chương trình, và chúng ta sẽ gõ lệnh trên Command:

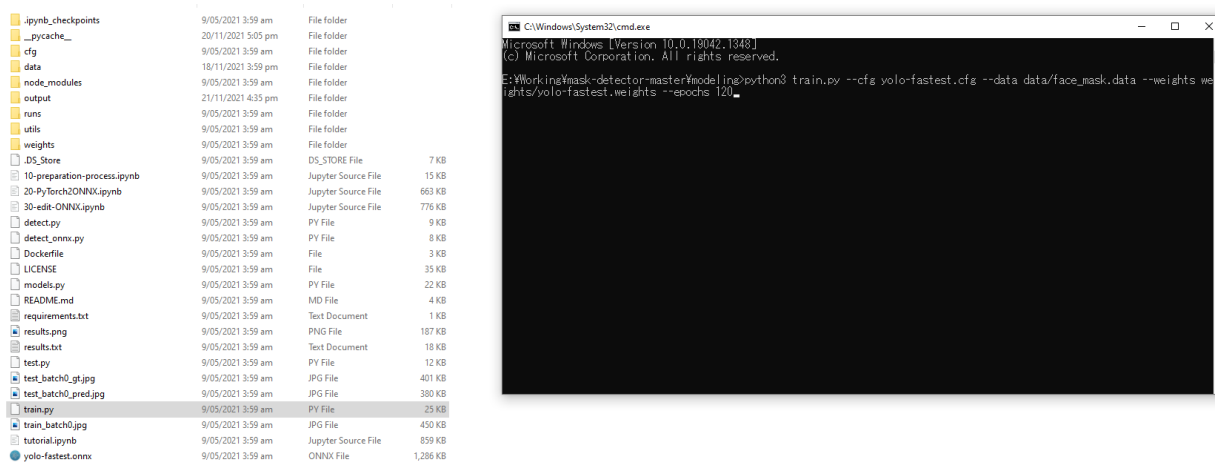
$ pip install -r requirements.txt



Hình 20: Tiến hành cài đặt thư viện.

Sau khi cài tất cả thư viện cần thiết, chúng ta sẽ tiếp tục gõ lệnh sau để tiến hành huấn luyện mô hình dựa vào tập Datasets chúng ta đã có:

$ python3 train.py --cfg yolo-fastest.cfg --data data/face_mask.data --weights weights/yolo-fastest.weights --epochs 120



Hình 21: Tiến hành huấn luyện mô hình.

Trong đó: --cfg: nơi dẫn tới tập tin yolo-fatest.cfg, tập tin này chứa thông tin của kiến trúc của Yolo-Fastest.

--data : dẫn tới tập tin face_mask.data, tập tin này sẽ chứa các thông tin các tập dataset cần cho việc huấn luyện.

--weights: nơi dẫn đến yolo-fastest.weights, đây là mô hình đã được huấn luyện trước để nhận dạng nhiều đối tượng được cung cấp bởi chính tác giả.

--epochs: số lần huấn luyện.

Sau khi huấn luyện hoàn thành, chúng ta thu được một mô hình "best.pt" dựa vào kiến trúc của yolo-fastest. Chúng ta có thể dùng lệnh như sau để chuyển thanh định dạng gốc của Darknet (đuôi .cfg) và dùng nó cho kiến trúc YOLO:

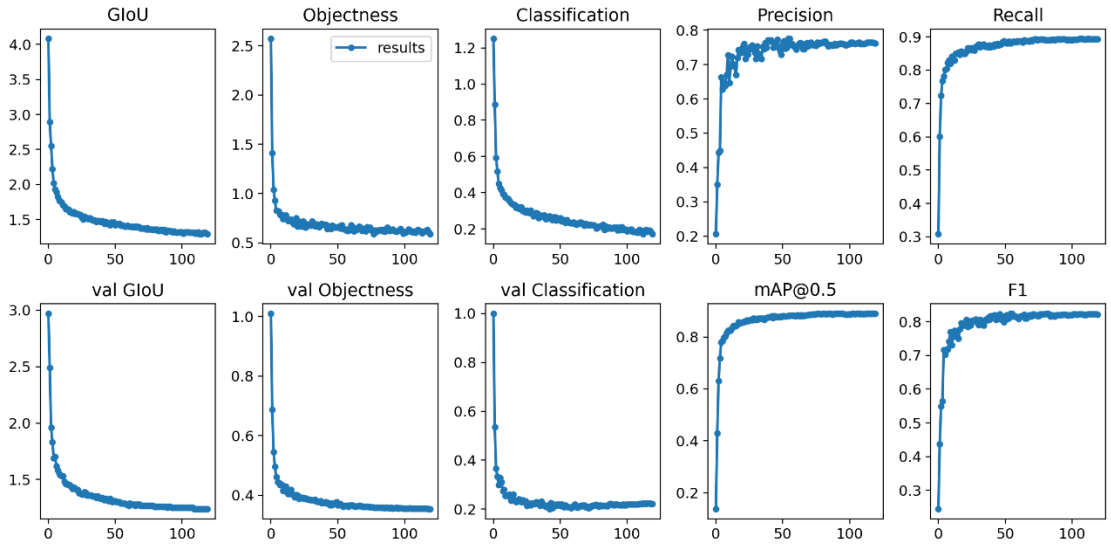
```
$ python3  -c "from models import *; convert('cfg/yolo-fastest.cfg', 'weights/best.pt')"
```

Sau khi huấn luyện hoàn tất chúng ta có đầy đủ các tập tin như sau: "yolo-fastest.cfg", "best.weights" và "face_mask.names" (thu được từ khi chuẩn bị ở tập Dataset, xem mục 3.4)

3.5.3. Đồ thị chỉ số và chương trình nhận diện

Đồ thị ta thu được trong quá trình huấn luyện mô hình bằng cách dùng lệnh:

```
from utils import utils; utils.plot_results()
```

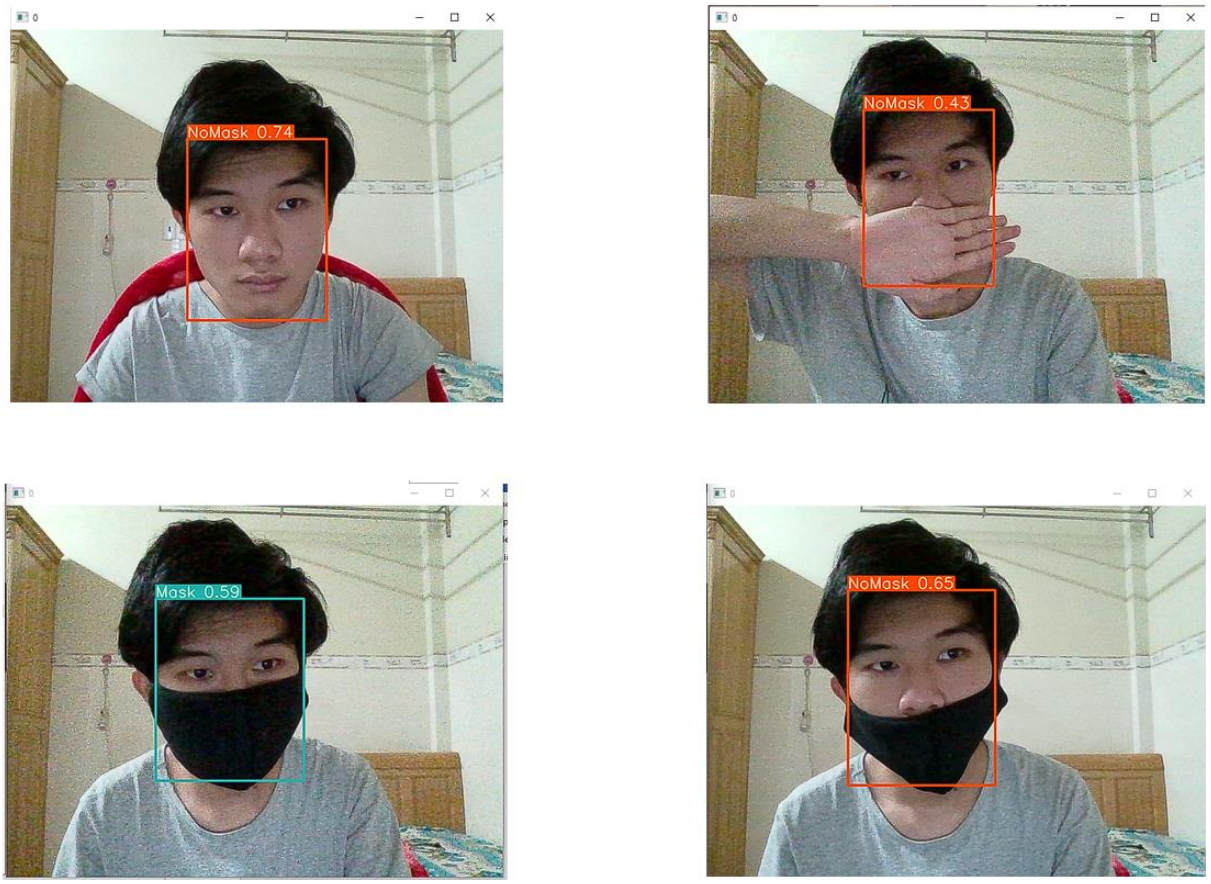


Hình 22: Đồ thị chỉ số của mô hình.

Để tiến hành kiểm tra mô hình huấn luyện được, chúng ta sẽ tiến hành chạy một trong các chương trình sau đây:

```
$ python3 detect.py --source 0
```

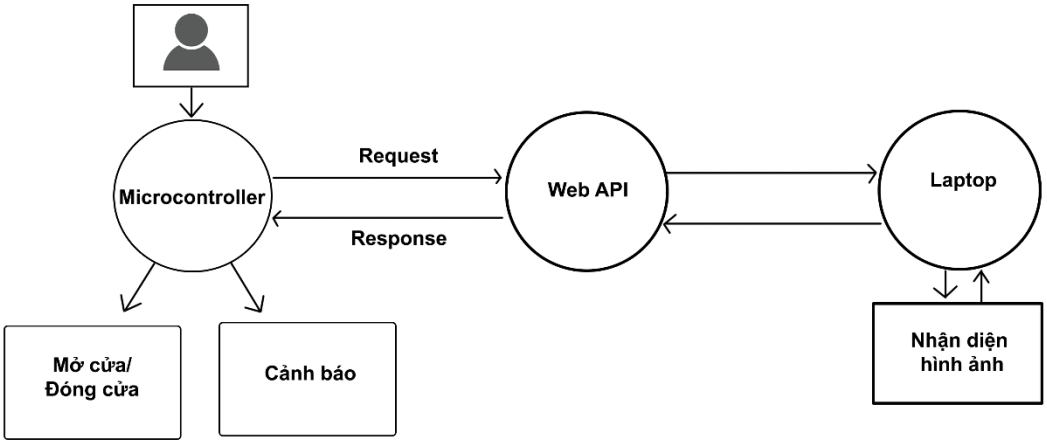
Chương trình sẽ tự động dựa vào khung hình trên camera để nhận diện.



Hình 23: Kết quả mô hình nhận diện.

CHƯƠNG 4:THIẾT KẾ HỆ THỐNG NHẬN DIỆN KHẨU TRANG

4.1. SƠ ĐỒ KHỐI TỔNG QUAN HỆ THỐNG



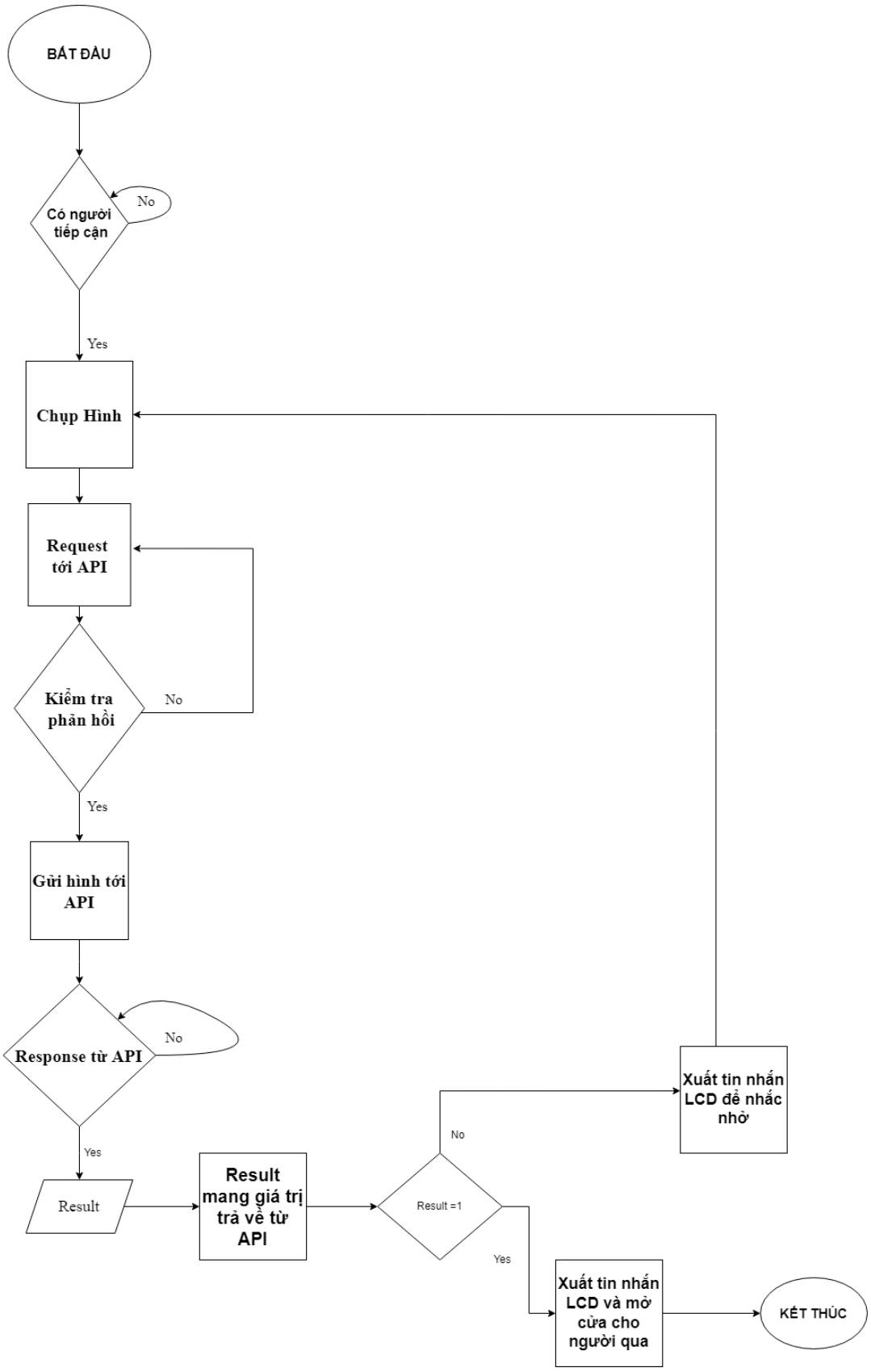
Hình 24: Sơ đồ khối tổng quan về hệ thống nhận diện khẩu trang.

Trong sơ khối, chúng ta sẽ phân bổ thành 2 tầng: Tầng đầu tiên được gọi là tầng giao tiếp Microcontroller-API và tầng thứ hai được gọi là Tầng giao tiếp API-Laptop

Ở tầng thứ nhất, Microcontroller có trọng trách ghi hình liên tục mỗi khi có người tiếp cận Camera, tương ứng mỗi Frame mà ta thu được từ Microcontroller sẽ được truyền tới Web API để frame được xử lí quá trình nhận dạng ở tầng hai. Ngay sau khi Web API trả kết quả được xử lí về, Microcontroller có trọng trách sẽ đưa lệnh điều khiển tới các ngoại vi để mở cửa/ đóng cửa, cảnh báo đeo khẩu trang.

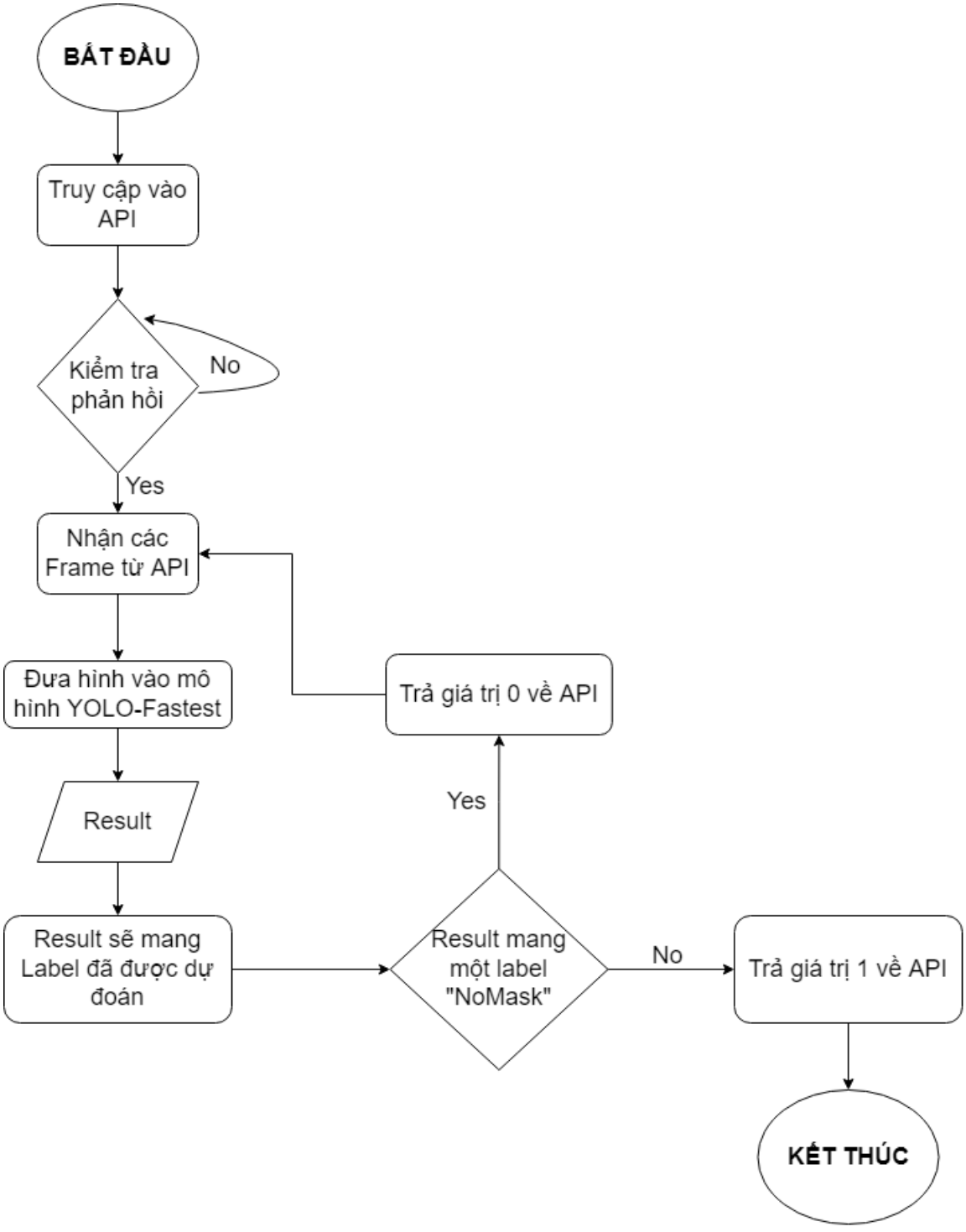
Ở tầng thứ hai, Laptop sẽ được nhận những dữ liệu từ Web API, với mỗi một frame nhận được sẽ tiến thành thuật toán nhận diện, sau đó trả kết quả đeo hay không đeo tới Web API.

4.2. LƯU ĐỒ THUẬT TOÁN TẦNG MICROCONTROLLER-API



Hình 25: Lưu đồ thuật toán tầng Microcontroller- API.

4.3. LƯU ĐỒ THUẬT TOÁN API-LAPTOP



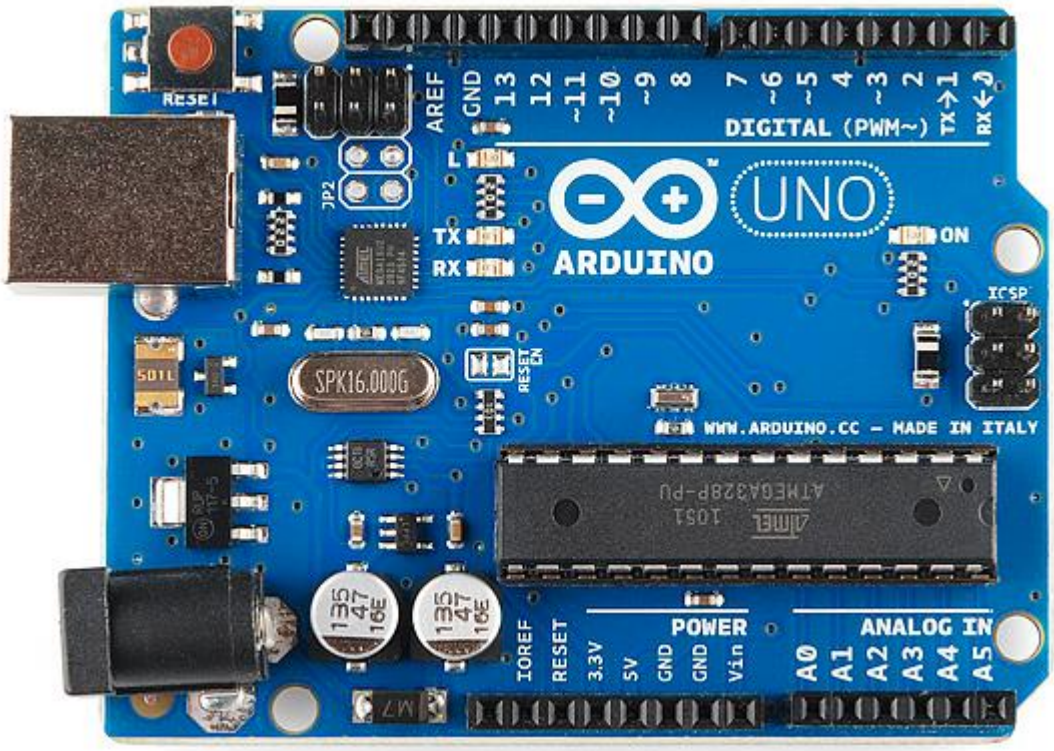
Hình 26: Lưu đồ thuật toán tầng API-Laptop.

4.4. SƠ ĐỒ MẠCH HỆ THỐNG

4.4.1. Các linh kiện sử dụng

• Trong đề tài này, chúng ta sẽ sử dụng 2 SoM với mỗi một board sẽ có những chức năng riêng biệt. SoM đầu tiên chúng ta sẽ sử dụng là ESP32-CAM, một SoM có

tích hợp một camera phù hợp cho việc ghi hình và gửi dữ liệu tới API để xử lí, và SoM còn lại sẽ là Arduino Uno xử lí các tác vụ bên ngoài như là mở cửa, xuất thông báo trên LCD,…



Hình 27: Adruino Uno.

- LCD 16x2: LCD sẽ được dùng để xuất ra nội dung để nhắc nhở đeo khẩu trang hoặc là mời vào khi đã đeo khẩu trang.

Hình 28: LCD 16x2.

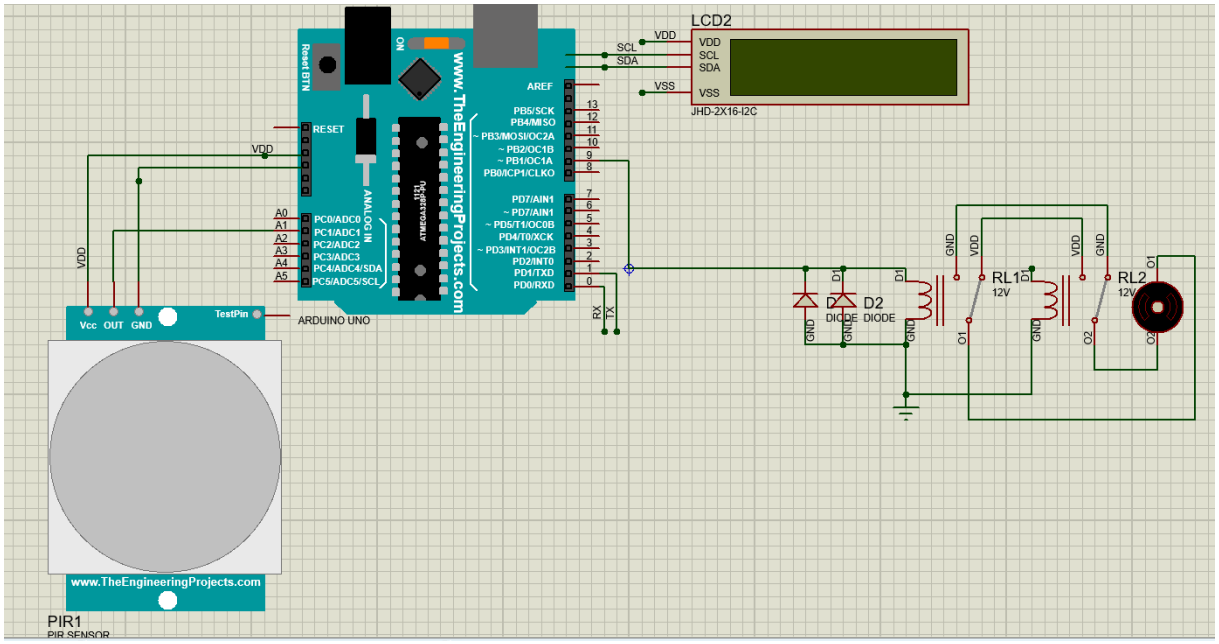
- Để tránh ESP32 phải liên tục làm việc nhằm tăng tuổi thọ cũng như tiết kiệm điện, chúng ta sẽ cần một cảm biến vật cản hồng ngoại E18-D80NK. Cảm biến này sẽ xác định được có người phía trước cảm biến, ngoài ra cảm biến phát ra tia hồng ngoại với dải tần số chuyên biệt cho khả năng chống nhiễu tốt kể cả ở điều khiện ánh sáng ngoài trời.

Hình 29: Sensor E18-D80NK.

- Động cơ Servo MG995 sẽ được dùng để đóng/mở cửa.

Hình 30: Động cơ Servo.

4.4.2. Sơ đồ mạch hệ thống



Hình 31: Sơ đồ mạch hệ thống.

Sau khi cảm biến hồng ngoại PIR phát hiện có người đang tới, lúc này Arduino Uno sẽ nhận được tín hiệu, và sau đó Arduino Uno sẽ gửi lệnh cho ESP32 tiến hành chụp hình và gửi tới API để Laptop triển khai thuật toán nhận dạng và trả kết quả lên

API. Sau khi ESP32 đã thiết lập mọi thứ cần thiết, sẽ gửi tín hiệu lại cho Arduino Uno. Arduino Uno sẽ bắt đầu truy cập vào API và đọc kết quả, phụ thuộc vào kết quả mà Arduino sẽ đưa ra lệnh phù hợp cho các ngoại vi.

4.5. THIẾT KẾ WEB API GIAO TIẾP

4.5.1. Giới thiệu Flask

Hình 32: Logo của Flask.

Flask là một framework của Python, còn được gọi là microframework bởi nó không yêu cầu các công cụ hay thư viện cụ thể nào.

Flask cung cấp các công cụ, thư viện và công nghệ cho phép bạn xây dựng một trang web. Đặc biệt, Flask rất dễ cài đặt chỉ với một lệnh: pip install Flask và triển khai chỉ bằng vài câu lệnh đơn giản, ở dưới là ví dụ một trang web có nội dung Hello World!:

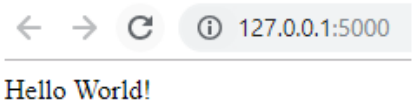
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(debug=True)
```

Hình 33: Chương trình Web có nội dung Hello World!.

Và chỉ cần chạy chương trình và mở một trình duyệt bất kì sau đó truy cập vào địa chỉ IP mà Flask cung cấp ta sẽ được một trang web hoàn chỉnh:



Hình 34: Web “Hello World!”.

Lí do nhóm chọn Framework này cho đồ án bởi vì:

Kiến trúc nhỏ, gọn, không bị bó buộc bởi bộ khung cồng kềnh, không gặp bất cứ khó khăn nào khi cấu hình hay tổ chức ứng dụng.

Flask có những ưu điểm: cực kì linh hoạt, tối giản, dễ tìm hiểu và sử dụng, định tuyến dễ dàng và rất dễ mở rộng.

Cung cấp cho người dùng các thành phần cốt lõi thường được sử dụng nhất của khung ứng dụng web như URL Routing, Request & Response Object, Template,…

Nguồn tài liệu tham khảo về Flask phong phú. Ví dụ một số cuốn sách tham khảo bên dưới:

Hình 35: Một số cuốn sách tham khảo về Flask.

Chính vì những ưu điểm trên, công việc chính của chúng ta chỉ cần xác định ý tưởng, mục tiêu và tập trung vào việc xây dựng các ứng dụng web.

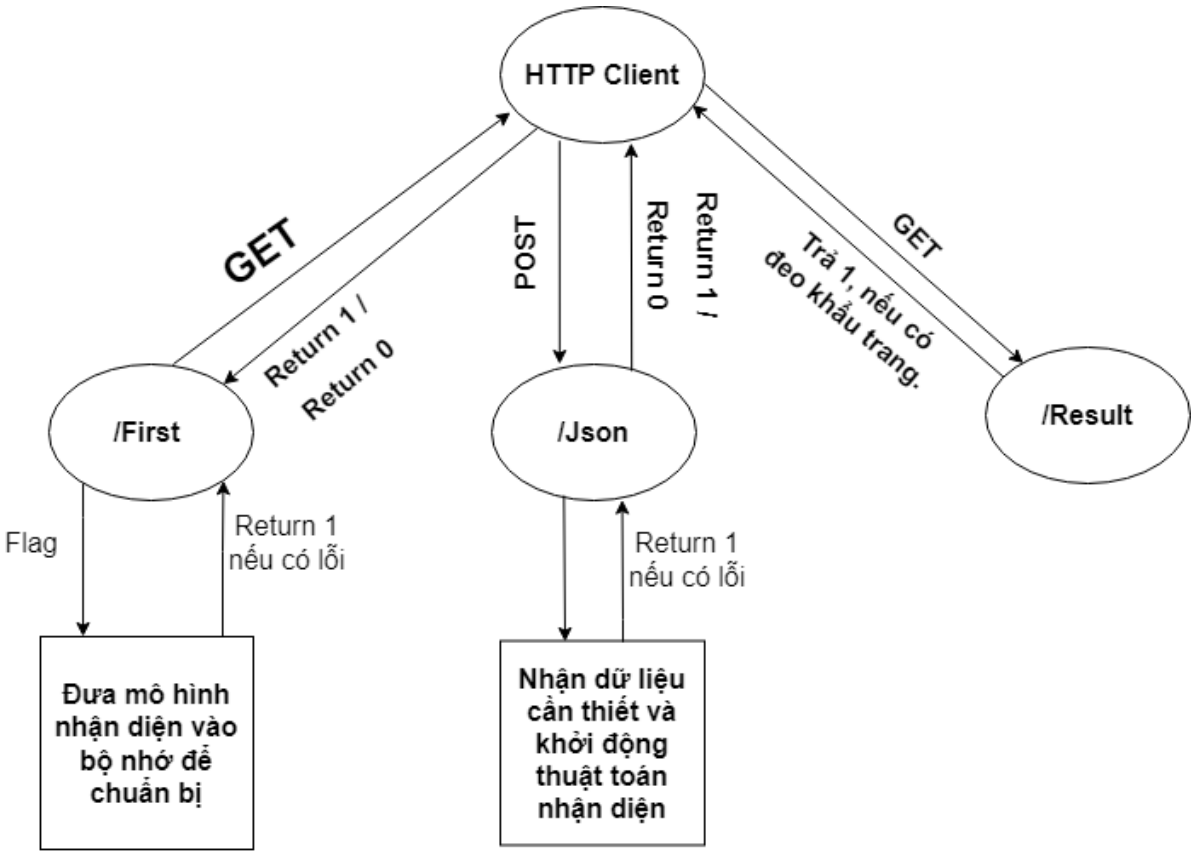
4.5.2. Sơ đồ chung và triển khai Web API

Để triển khai một Web API phù hợp cho hệ thống nhận diện khẩu trang của nhóm, giả định địa chỉ IP ta tạm thời gọi là localip và Port sẽ là 2222 (Địa chỉ IP và port cho Web API có thể thay đổi), chúng ta sẽ thiết kế 3 URL với tương ứng ba nhiệm vụ khác nhau.:

Đầu tiên là URL "http://localip:2222/First" , khi Vi điều khiển chúng ta sử dụng phương thức GET vào URL này, lập tức Web API sẽ tiến hành gửi Flag cho để kích hoạt chương trình có nhiệm vụ thiết lập mô hình nhận diện vào các bộ nhớ (ví dụ: Nếu chúng ta sử dụng một Laptop có VGA, các phép tính sẽ được thực hiện song song trên VRAM của VGA cũng như đưa các ma trận liên quan vào VRAM).

Tiếp theo là URL "http://localip:2222/Json", khi Vi điều khiển sử dụng phương thức POST dùng để gửi dữ liệu tới Web API, ở đây chúng ta sẽ gửi một tập tin có định dạng JSON (JavaScript Object Notation) có chứa các cặp key-value để truyền dữ liệu tới Web API để xử lí.

Cuối cùng là URL "http://localip:2222/Result", Vi điều khiển sẽ sử dụng phương thức GET để nhận giá trị trả về. Căn cứ vào giá trị trả về, vi điều khiển sẽ cấp phát nhiệm vụ cho các ngoại vi. Cụ thể nếu giá trị trả về 1 thì chúng ta sẽ tiến hành mở cửa và xuất ra màn hình LCD dòng chữ "Xin Mời Quý Khách", hoặc ngược lại sẽ tiếp tục đóng cửa và xuất ra dòng LCD "Xin Hãy Đeo Khẩu Trang" .



Hình 36: Sơ đồ mô tả các URL của Web API.

Để tiến hành chạy Web API, nếu có IDE của Python hãy mở file "predict_server.py" để tiến hành khởi động. Nếu không có, chúng ta sẽ dùng Command Prompt để chạy chương trình với lệnh:

python <Địa chỉ tới file predict_server.py>

```
PS E:\Working\mask-detector-master\mask-detector-master\deployment> & C:/Python/Python38/python.exe e:/Working/mask-detector-master/mask-detec
tor-master/deployment/predict_server.py
```

Sau khi chạy chương trình, thư viện Flask sẽ tiến hành khởi tạo các URL và cấp địa chỉ IP của Web API để các Client truy cập.

```
* Serving Flask app 'predict_server' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.1.3:2222/ (Press CTRL+C to quit)
```

Hình 37: Các thông tin về Web API sau khi chạy chương trình.

Ta có thể thấy IP được cấp là : 192.168.1.3, cùng với Port là 2222. Và Ctrl+ C để ngừng Web API

4.6. THIẾT KẾ CHƯƠNG TRÌNH CHO ESP32

4.6.1. Giới thiệu Arduino IDE

Arduino IDE là một phầm mềm mã nguồn mở chủ yếu được sử dụng để viết các chương trinh và biên dịch mã vào các module Arduino. Khi người dùng viết mã và biên dịch, IDE sẽ tạo file Hex cho mã. Những file Hex này sẽ được nạp vào các board bằng cáp USB.

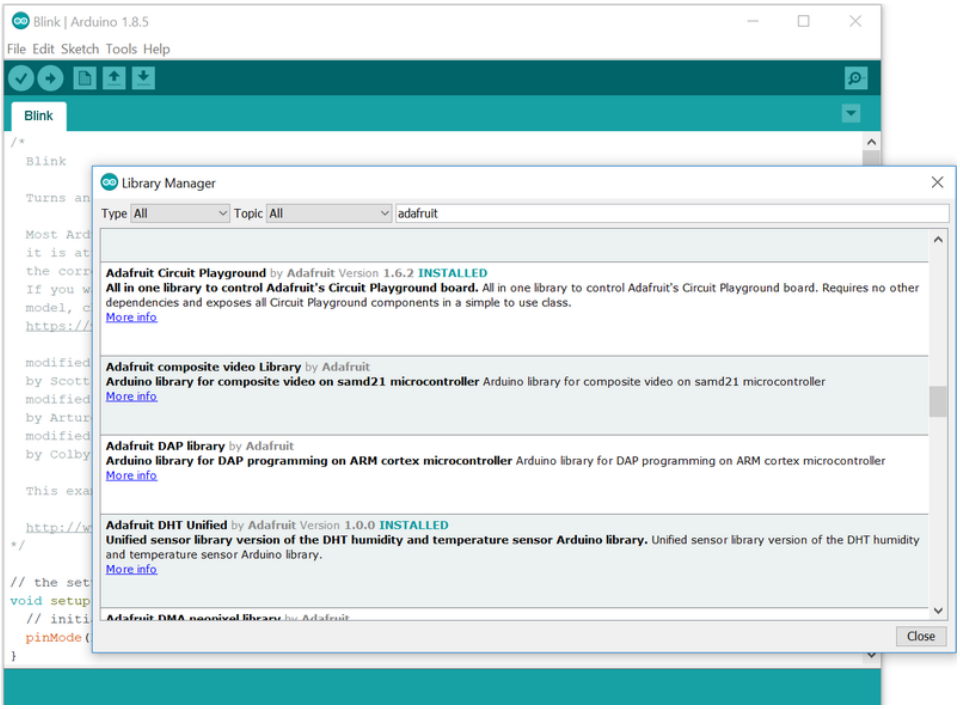
Ngoài ra, Arduino IDE có những ưu điểm sau đây:

Phầm mềm có mã nguồn mở miễn phí, cùng với việc mỗi khi phát hiện các lỗi về bảo mật, nhà phát hành sẽ tiến hành cập nhật bản vá rất nhanh nhằm tăng khả năng bảo mật thông tin.

Sử dụng ngôn ngữ C/C++ thân thiện để lập trình.

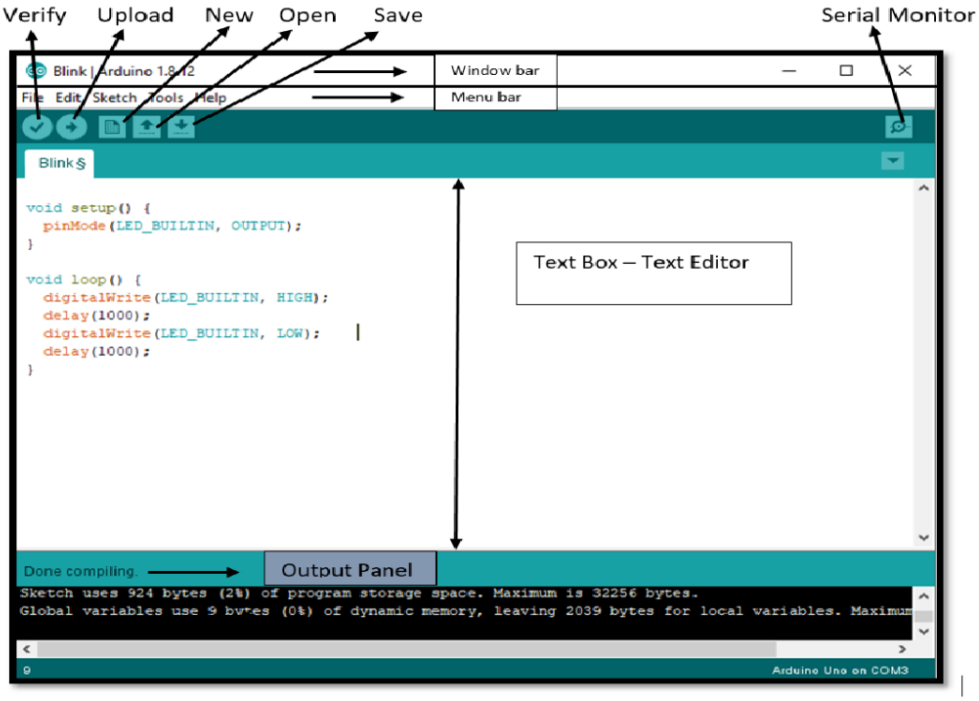
Hỗ trợ tốt cho nhiều board mạch như Arduino, ESP32,…

Thư viện hỗ trợ phong phú với hơn 700 thư viện, được viết và chia sẽ bởi các nhà phát hành Arduino Software và các thành viên trong cộng đồng Arduino. Chúng ta có thể tận dụng chúng mà không cần phải trả bất kì chi phí nào.



Hình 38: Thư viện Library Manager vô cùng đa dạng.

Giao diện đơn giản, dễ sử dụng



Hình 39: Một số tính năng thường được sử dụng.

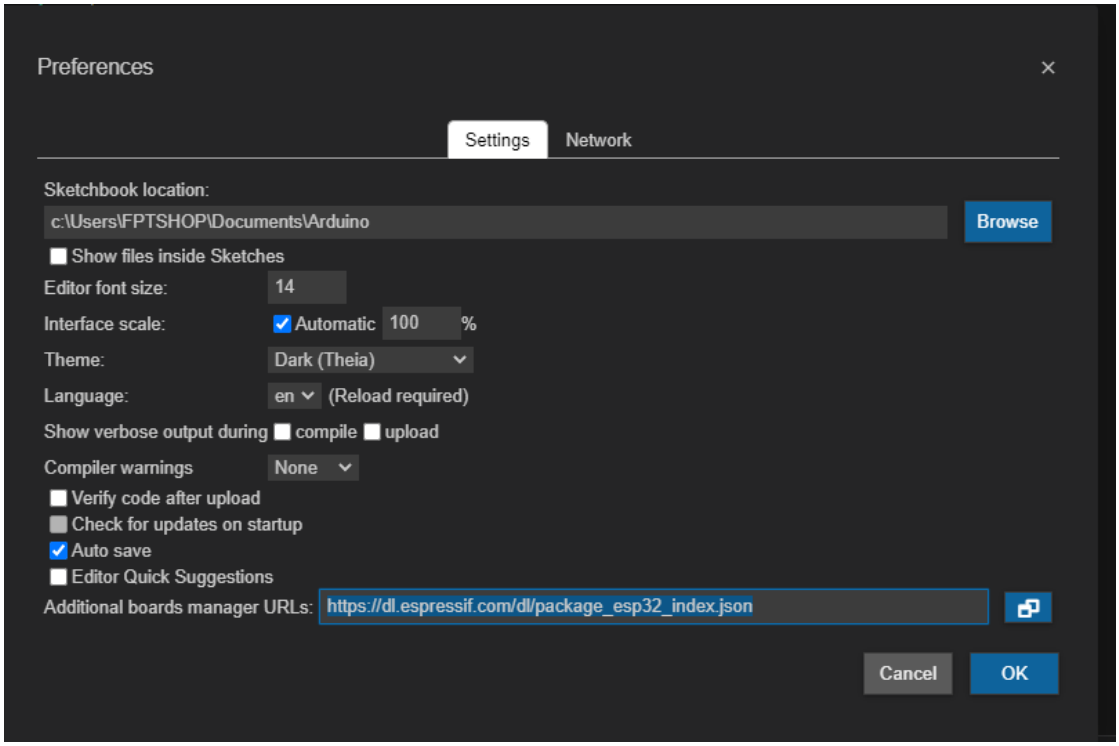
Hỗ trợ đa nền tảng như Window, MacOS, Linux. Chính vì thế người dùng có thể truy cập vào phầm mềm bất cứ đâu, bất cứ khi nào. Ngoài ra, người dùng còn có thể truy cập vào công cụ từ đám mây. Điều này cho phép các nhà lập trình lựa chọn tạo và

lưu dự án của mình trên đám mây. Hoặc họ có thể xây dựng chương trình trên máy tính và upload nó vào board mạch.

4.6.2. Cài đặt board esp32 và thư viện cần thiết

Để tiến hành viết chương trình và nạp vào board ESP32, đầu tiên chúng ta sẽ tiến hành cài các Board hỗ trợ ESP32 lên Arduino. Đầu tiên chúng ta sẽ mở phầm mềm Arduino IDE lên, tiếp theo chúng ta sẽ vào File => Preferences. Khi hộp thoại Preferences xuất hiện. Chúng ta sẽ copy và paste địa chỉ này vào tab Settings, tại mục "Additional boards managers URLs" và bấm OK :

https://dl.espressif.com/dl/package_esp32_index.json



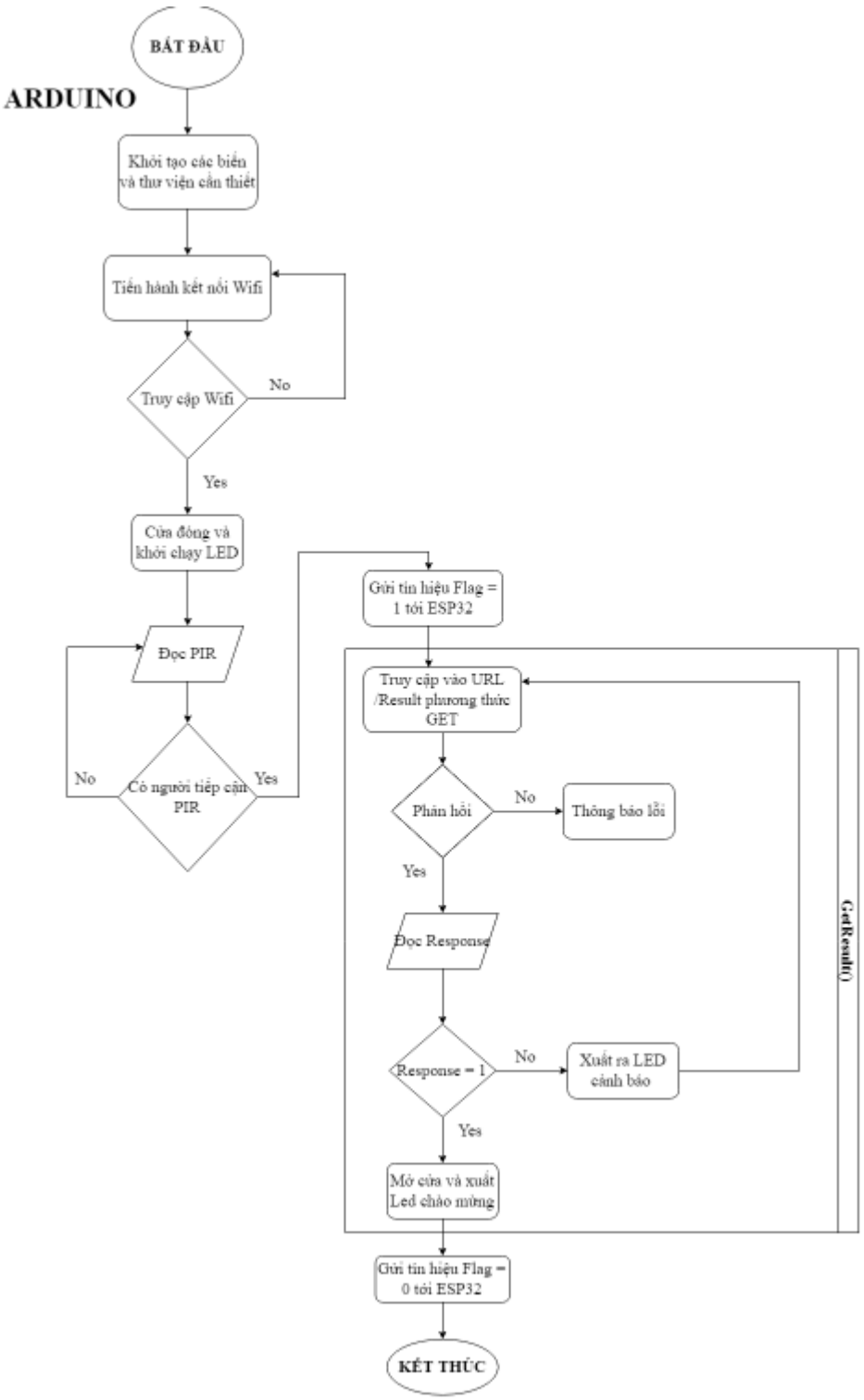
Sau đó, chúng ta vào Tools => Board => Board Manager… Tại ô tìm kiếm, gõ "Esp32", và ta sẽ thấy xuất hiện esp32 by Espressif Systems, cung cấp và hỗ trợ rất nhiều board liên quan của hãng Espressif Systems, và bấm INSTALL để tiến hành cài đặt.

Sau đó, vào lại Tools => Board, bạn sẽ thấy ngoài Arduino ra sẽ xuất hiện thêm esp32, ở đây chúng ta sẽ chọn board AI Thinker ESP32-CAM để biên dịch và nạp chương trình vào board chúng ta sử dụng cho đề tài.

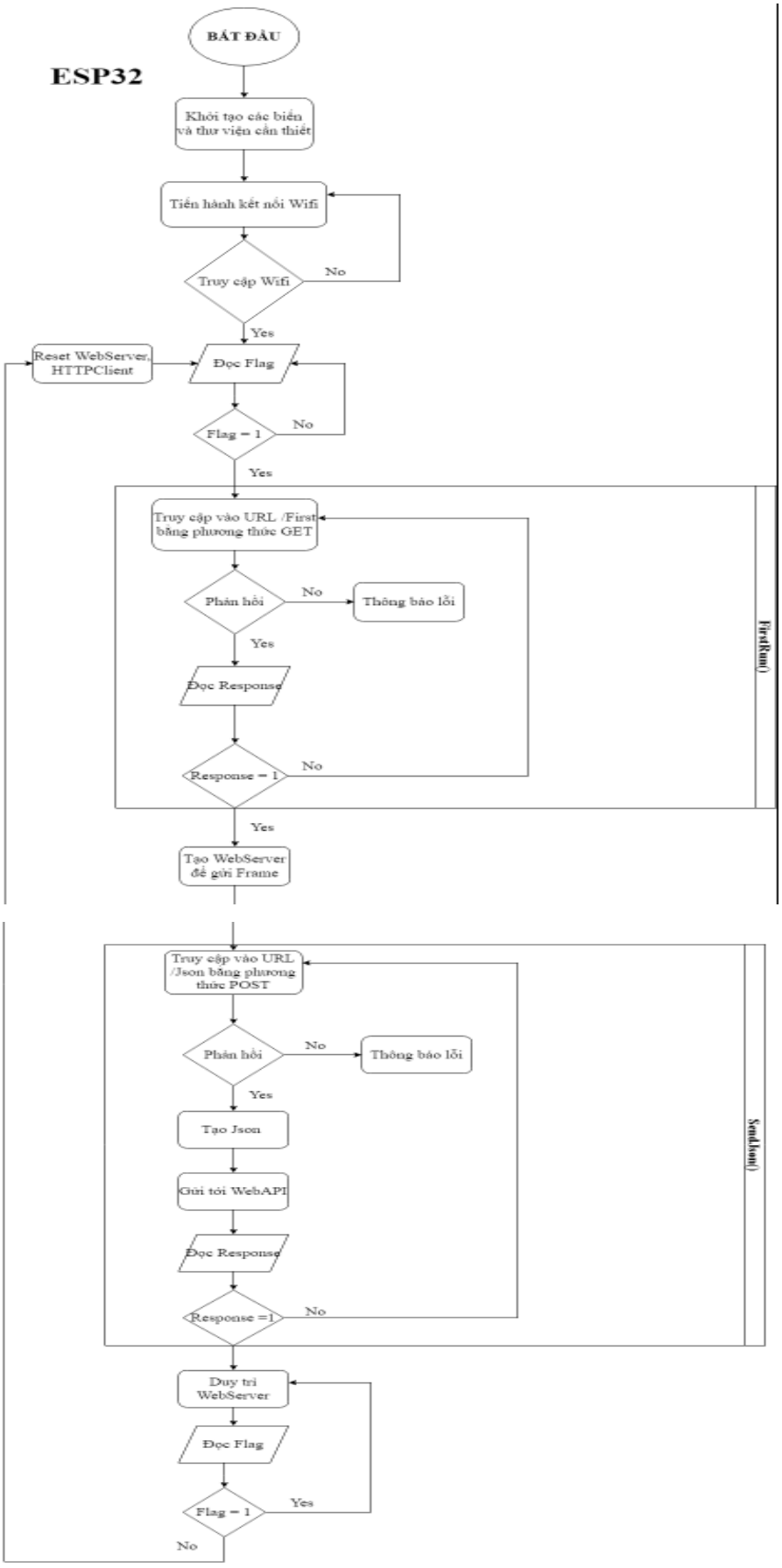
Tiếp theo, chúng ta sẽ tiến hành cài một số thư viện cần thiết. Đầu tiên hãy vào Sketch => Include Library => Manager Libraries… Search " Arduino Json by Benoit Blanchon" và tiến hành cài. Sau đó vào đường dẫn: https://github.com/yoursunny/esp32cam để tải thư viện và lưu về dưới dạng đuôi .zip,

sau đó vào Sketch => Include Library => Add .ZIP Library… Chọn đường dẫn dẫn tới tập tin ta vừa tải để load thư viện.

4.6.3.Thuật toán toàn bộ hệ thống nhận diện khẩu trang



Hình 40: Lưu đồ giải thuật của hệ thống của Adruino.



Hình 41: Lưu đồ giải thuật của hệ thống của ESP32.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. ĐÁNH GIÁ ĐỀ TÀI

5.1.1. Những công việc đã làm được

Tìm hiểu các kiến thức về các thuật toán được dùng cho Object Detection, cụ thể là thuật toán YOLO.

Xây dựng các chương trình liên quan, tạo ra tập dataset để huấn luyện, đánh giá, triển khai thuật toán Object Detection.

Tìm hiểu và sử dung được Framework Flask để tạo ra một API để hệ thống có thể mượn sức mạnh của VGA Laptop để xử lí thuật toán nhanh hơn, giảm thiểu áp lực lên phần cứng.

Vẽ được sơ đồ hệ thống, lưu đồ giải thuật

Nghiên cứu và lập trình được trên board Arduino và ESP32 thông qua IDE Arduino

5.1.2. Khuyết điểm của đề tài

Vẫn chưa thể hoàn thiện toàn bộ sản phẩm vì thiếu linh kiện.

Thuật toán nhận diện rất kém khi thu được hình ảnh ở ánh sáng thấp.

Hệ thống hiện tại chưa đề cập đến giải pháp nâng cao độ chính xác của mô hình YOLO.

Chưa thể triển khai nhiều mô hình khác để so sánh cụ thể hiệu suất và tốc độ các mô hình.

Vẫn chưa tạo ra giao diện để người dùng có thể dễ sử dụng và theo dõi.

5.2. HƯỚNG PHÁT TRIỂN CỦA ĐỀ TÀI

Đề tài này không dừng ở mỗi hệ thống nhận diện đeo khẩu trang, với kiến trúc và những đoạn chương trình này chúng ta có thể mở rộng trong nhiều đối tượng nhận diện khác tuỳ theo nhu cầu miễn là chúng ta có một tập Dataset phục vụ cho huấn luyện, ví dụ như nhận diện nhân viên.

Việc sử dụng API giao tiếp có thể mở rộng cho nhiều thiết bị khác không chỉ ép buộc ở mỗi ESP32 như là Android, IOS hoặc các máy tính khác.

Thuật toán có thể triển khai không chỉ ở Laptop, chúng ta có thể triển khai lên các SoM (System on Module) như Nvidia Jetson, Raspberry Pi, BeagleBone,…

TÀI LIỆU THAM KHẢO

[1] K. He, G. Gkioxari, P. Dollr, and R. Girshick, "Mask r-cnn," 2017 IEEE International Conference on Computer Vision (ICCV), p. 2980–2988, Oct, 2017.

[2] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1492-1500, 2017.

[3] G. Ghiasi, T.-Y. Lin, R. Pang, and Q. V. Le, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," arXiv preprint, 2019.

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510-4520, 2018.

[6] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6848-6856, 2018.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788, June 2016.

[8] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," Neural computation, vol. 29, no. 9, pp. 2352-2449, 2017.

[9] L. Weng, "Object Detection Part 4: Fast Detection Models," https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html, 2018.

[10] V. H. Tiệp, Machine Learning cơ bản, Nhà xuất bản khoa học và kỹ thuật, 2016.

[11] Yi-Qi Huang , Jia-Chun Zheng , Shi-Dan Sun, Cheng-Fu Yang, and Jing Liu, "Optimized YOLOv3 Algorithm and Its Application in Traffic Flow Detections," p. 4.

[12] Joseph Redmon, Ali Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767, 2018.

[13] T. Nguyen, 28 April 2019. [Online]. Available: https://nttuan8.com/bai-10-cac-ky-thuat-co-ban-trong-deep-learning/.

[14] [Online]. Available: https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset.

[15] dog-qiuqiu, [Online]. Available: https://github.com/dog-qiuqiu/Yolo-Fastest.

[16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft coco: Common objects in context," Computer Vision – ECCV 2014, pp. 740-755, 2014.

[17] Licheng Jiao, Fellow, IEEE, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng and Rong Qu, "A Survey of Deep Learning-based Object Detection," IEEE Access, vol. 7, p. 10, 2019.