

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ
KHOA ĐIỆN TỬ



BÁO CÁO ĐỒ ÁN I

*Đề tài: NHẬN DIỆN BIỂN BÁO GIAO THÔNG VÀ
ĐÈN GIAO THÔNG*

Giảng viên hướng dẫn: Nguyễn Tiến Hòa

Mã lớp: ET3290Q

Nhóm sinh viên thực hiện: Nguyễn Trần Dũng - 20207552

Hà Nội, 7-2023

LỜI MỞ ĐẦU

Trong thời đại của kỷ nguyên cách mạng khoa học 4.0, công nghệ đang tiến bộ không ngừng và trở thành một phần không thể thiếu trong cuộc sống. Nhờ sự phát triển của công nghệ, con người đã tiết kiệm được nguyên liệu, nhiên liệu, thời gian và năng lượng trong công việc hàng ngày. Trí tuệ nhân tạo (AI) đã xâm nhập vào mọi lĩnh vực cuộc sống, và trong đó, công nghệ phần mềm đóng vai trò quan trọng nhất, dẫn đầu trong quá trình này và được ứng dụng ngày càng rộng rãi để thay đổi thói quen, phong cách làm việc và tăng cường hiệu suất công việc của con người.

Với việc nhận thức được xu hướng thời đại và để đáp ứng yêu cầu ngày càng cao của Đồ án I, áp dụng công nghệ vào các khía cạnh của cuộc sống. Trong Đồ án I, em sẽ nghiên cứu và phát triển giải pháp cho bài toán phát hiện đối tượng (Object Detection). Đây là đề tài đầu tiên mà em nghiên cứu, nên không thể tránh khỏi những hạn chế và thiếu sót. Em rất mong được thầy bồi sung và đóng góp ý kiến để em có thể ngày càng hoàn thiện kiến thức.

LỜI CAM ĐOAN

Em là Nguyễn Trần Dũng, là sinh viên thực hiện Đồ án I thuộc lớp ET - LUH 01 K65. Em xin cam đoan rằng các dữ liệu được trình bày trong đồ án này là hoàn toàn trung thực và chính xác phản ánh kết quả đo đạc thực tế. Tất cả các thông tin trích dẫn tuân thủ các quy định về sở hữu trí tuệ và tất cả các tài liệu tham khảo đã được liệt kê rõ ràng. Em xin chịu hoàn toàn trách nhiệm về nội dung được trình bày trong báo cáo này.

Hà Nội, ngày 7 tháng 7 năm 2023

Người cam đoan

DŨNG

Nguyễn Trần Dũng

MỤC LỤC HÌNH ẢNH

Hình 1. Cấu tạo các lớp trong Deep Learning.....	10
Hình 2. Mối liên hệ giwac AI, Machine Learning & Deep Learning	11
Hình 3. R-CNN	13
Hình 4. Fast R-CNN	14
Hình 5. Faster R-CNN	15
Hình 6. So sánh về mặt thời gian	15
Hình 7. Đồ thị Precision Recall Curve.....	17
Hình 8. Non – max suppression.....	19
Hình 9. Mô hình kiến trúc mạng CNN.....	20
Hình 10. Convolutional Layer	21
Hình 11. Dimensions - filter	22
Hình 12. Stride	22
Hình 13. Parameter compatibility	24
Hình 14. Mô hình Training + Predicting.....	26
Hình 15. Kết nối Drive và cài đặt Tensorflow	27
Hình 16. Import thư viện.....	27
Hình 17. Quá trình train – 15 lần	39
Hình 18. Biểu đồ độ chính xác.....	40
Hình 19. Biểu đồ sai số	40
Hình 20. Hình ảnh giao diện	42
Hình 21. Các bước hoạt động của Yolo	44
Hình 22. Thông số của 1 Bounding box.....	44
Hình 23. Mã hóa nhiều Bounding Box	45
Hình 24. Các thao tác trên makesenes.ai.....	46
Hình 25. File dataset sau khi được chia thành các tập train/val	47
Hình 26. Clone Yolov5 & cài đặt môi trường, thư viện	48
Hình 27. Upload file Data	48
Hình 28. Giải nén file Data	48
Hình 29. Tệp yaml	49
Hình 30. File yaml cho Data đèn giao thông	49
Hình 31. Tải file coco128.yaml.....	50
Hình 32. Nội dung file coco128.yaml.....	50
Hình 33. File trafficLightData1.yaml	50
Hình 34. Upload file yaml vào thư mục Yolov5.....	51
Hình 35. File train.py	51
Hình 36. Kết quả training.....	52
Hình 37. Biểu đồ thể hiện độ chính xác của các nhãn	53
Hình 38. Biểu đồ kết quả quá trình Training.....	54
Hình 39. Biểu đồ dữ liệu, annotation box cho từng nhãn dữ	55
Hình 40. Quá trình Predicting	55
Hình 41. Kết quả của quá trình Predicting.....	56

MỤC LỤC BẢNG BIỂU

Bảng 1. Max Pooling & Average Pooling	21
Bảng 2. Zero Padding	23
Bảng 3. Bảng đánh giá độ phức tạp của một mô hình	24
Bảng 4. Các biến thể của ReLU	25
Bảng 5. So sánh thông số kĩ thuật của TF1 & TF2	27

MỤC LỤC

LỜI MỞ ĐẦU.....	2
LỜI CAM ĐOAN.....	3
DANH MỤC HÌNH ẢNH.....	4
DANH MỤC BẢNG BIỂU	5
MỤC LỤC.....	6
PHÂN CHIA CÔNG VIỆC.....	8
PHẦN I. TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO	9
1. Khái niệm.....	9
2. Phân loại	9
3. AI, Machine Learning & Deep Learning.....	9
PHẦN II. BÀI TOÁN OBJECT DETECTION.....	12
1. Object Detection.....	12
1.1. Khái niệm.....	12
1.2. Phân loại.....	12
1.3. Các thuật toán Object Detection – Deep Learning	12
2. Metrics đánh giá Object Detection model.....	16
PHẦN III. NHẬN DIỆN BIỂN BÁO GIAO THÔNG	20
1. Tổng quan	20
1.1. Tổng quan về mạng neural tích chập - CNN	20
1.2. Các kiểu tầng trong mạng CNN.....	20
1.3. Các siêu tham số của bộ lọc	22
1.4. Điều chỉnh siêu tham số	23
1.5. Các hàm kích hoạt thường gặp	25
2. Xây dựng kiến trúc CNN trên Google Colab.....	26
2.1. Import thư viện, sử dụng back-end Tensorflow	26
2.2. Import Data	28
2.3. Data Visualisation	29
2.4. Khởi tiền xử lý	30
2.5. Mô hình CNN	32
2.6. Module TEST	35
3. Pycharm (Module Train + GUI).....	36
3.1. Module Train.....	36
3.2. Kết quả module Train.....	39
3.3. Module GUI.....	41
PHẦN IV. NHẬN DIỆN ĐÈN GIAO THÔNG	43

1.	Tổng quát.....	43
1.1.	Tổng quan về Yolo	43
2.	Các bước thực hiện training Yolov5 trên Google Colab	45
2.1.	Chuẩn bị Dataset.....	45
2.2.	Quá trình training model trên Google Colab	47
	PHẦN V. KẾT LUẬN.....	57

PHẦN I. KHÁI QUÁT TRÍ TUỆ NHÂN TẠO

1. Khái niệm

Trí tuệ nhân tạo (Artificial Intelligence - AI) là một lĩnh vực trong khoa học máy tính, mô phỏng quá trình suy nghĩ và học tập của con người trên máy móc. Công nghệ AI cho phép máy móc học từ thông tin và quy tắc, suy luận và tự sửa lỗi. Nó đã thay đổi cách chúng ta sống và làm việc, từ ứng dụng trong y tế, giao thông đến cuộc sống hàng ngày. AI là công cụ quan trọng để gia tăng hiệu suất và tạo ra giá trị, và cũng mang lại triển vọng hứa hẹn như tự động hóa công việc và đổi mới với thách thức tương lai.

2. Phân loại

Trí Tuệ nhân tạo được phân loại dựa trên khả năng mô phỏng trí thông minh con người. Có hai dạng phổ biến của trí tuệ nhân tạo, tùy thuộc vào mục đích và cấu trúc sử dụng:

Trí tuệ nhân tạo hẹp (Artificial Narrow Intelligence - ANI): Được sử dụng cho các tác vụ cụ thể và đơn điệu như nhận dạng giọng nói, phát hiện đối tượng, nhận dạng khuôn mặt, và nhiều nhiệm vụ khác. Mặc dù ANI có nhiều khả năng, nhưng hoạt động của nó bị giới hạn và kiểm soát bởi con người, do đó nó được gọi là trí tuệ nhân tạo hẹp.

Trí tuệ nhân tạo tổng hợp (Artificial General Intelligence - AGI): Đạt được mức độ ý thức, suy nghĩ khách quan, tự nhận thức, tri giác và khôn ngoan. AGI, còn được gọi là trí tuệ nhân tạo mạnh, có khả năng tương đương với trí tuệ con người. AGI không bị ràng buộc bởi bất kỳ hạn chế nào. Nó có thể học hỏi, thực hiện nhiều tác vụ khác nhau và có khả năng thể hiện đa dạng như con người.

Điểm khác biệt giữa hai dạng trí tuệ nhân tạo này là ANI tập trung vào các tác vụ cụ thể, trong khi AGI đại diện cho một hệ thống tổng hợp thông minh có khả năng tổng quát và sáng tạo như con người.

3. AI, Machine Learning & Deep Learning

Machine Learning

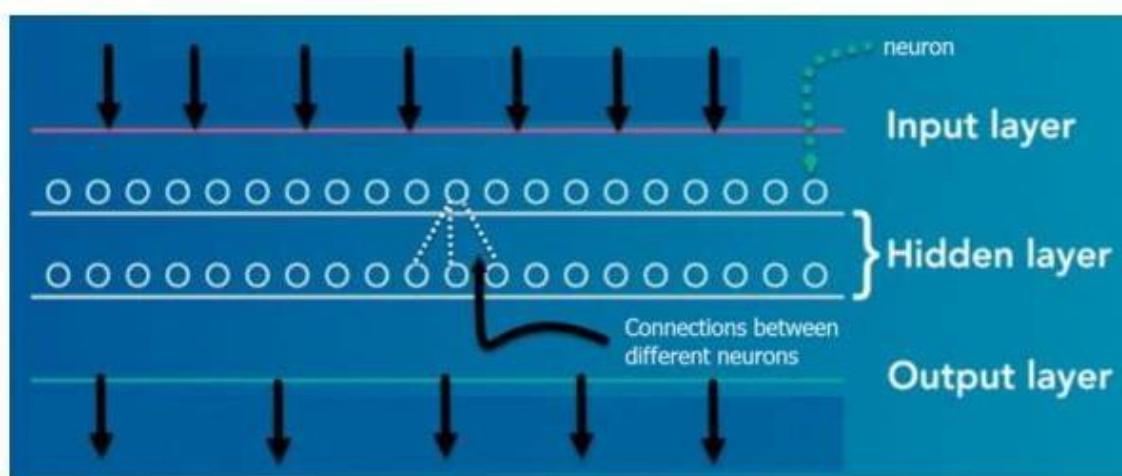
Machine Learning là quá trình ứng dụng các thuật toán để phân tích dữ liệu, học hỏi từ dữ liệu đó, và từ đó đưa ra quyết định hoặc dự đoán về các vấn đề liên quan. Thay vì viết mã phần mềm theo cách thủ công với các hướng dẫn cụ thể để hoàn thành một nhiệm vụ cụ thể, máy móc được "đào tạo" bằng cách sử dụng một lượng lớn dữ liệu và các thuật toán cho phép nó học cách thực hiện yêu cầu.

Các thuật toán Machine Learning xuất phát từ các khái niệm cơ bản của Trí tuệ Nhân tạo (AI) và tiếp cận thuật toán qua nhiều năm đã bao gồm: lập trình logic, phân cụm, học tăng cường và mạng Bayesian...

Deep Learning

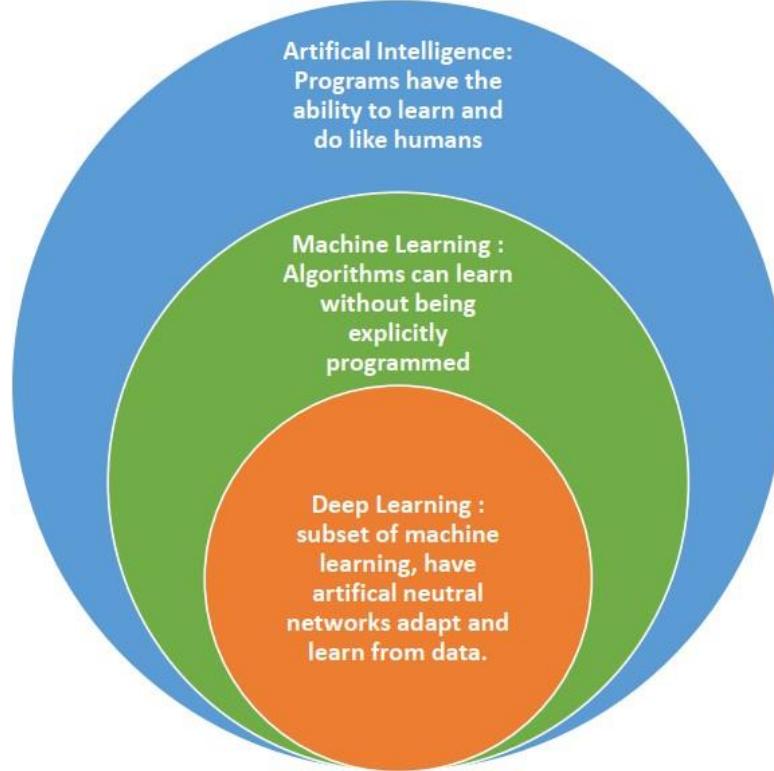
Deep Learning là một phần mềm máy tính mô phỏng mạng lưới các nơ-ron trong não con người. Nó là một phân nhánh của Machine Learning và sử dụng các mạng nơ-ron sâu (deep neural networks).

Các thuật toán Deep Learning được xây dựng với các lớp kết nối gồm lớp đầu vào (input layer), lớp đầu ra (output layer) và các lớp ẩn (hidden layers) nằm giữa chúng. Thuật ngữ "deep" được sử dụng để chỉ các mạng nơ-ron có nhiều hơn 2 lớp tham gia vào mạng.



Hình 1. Cấu tạo các lớp trong Deep Learning

Mối liên hệ giữa AI, Machine Learning & Deep Learning



Hình 2. Mối liên hệ giwac AI, Machine Learning & Deep Learning

PHẦN II. BÀI TOÁN OBJECT DETECTION

1. Object Detection

1.1. Khái niệm

Phát hiện đối tượng (Object Detection) là một kỹ thuật trong lĩnh vực thị giác máy tính, được sử dụng để nhận biết và xác định vị trí các đối tượng trong hình ảnh hoặc video. Đặc biệt, phát hiện đối tượng tạo ra các khung giới hạn xung quanh các đối tượng được phát hiện, cho phép xác định chính xác vị trí và chuyển động của chúng trong một cảnh cụ thể.

1.2. Phân loại

Phân định "Object Detection" hiện được chia thành hai phương pháp: tiếp cận dựa trên học máy (Machine Learning-based approaches) và tiếp cận dựa trên học sâu (Deep Learning-based approaches).

Tiếp cận dựa trên học máy: Sử dụng các kỹ thuật thị giác máy tính để phân tích các đặc điểm khác nhau của hình ảnh, như biểu đồ màu hoặc cạnh, để xác định nhóm pixel có thể thuộc về một đối tượng. Các đặc điểm này sau đó được đưa vào một mô hình hồi quy để dự đoán vị trí của đối tượng và nhãn tương ứng.

Tiếp cận dựa trên học sâu: Sử dụng mạng neural tích chập (Convolutional Neural Network) để thực hiện phát hiện đối tượng từ đầu đến cuối, không cần giám sát trước, trong đó các đặc điểm không cần được xác định và trích xuất riêng biệt.

Object Detection là một trong những bài toán phổ biến trong lĩnh vực Thị giác máy tính. Hiện nay, với sự phát triển của các thuật toán Học sâu, bài toán này đã được giải quyết tốt hơn. Các thuật toán nổi tiếng trong lĩnh vực này bao gồm: R-CNN, Fast R-CNN, Faster R-CNN, YOLOv1, YOLOv2, YOLOv3, YOLOv4, YOLOv5, SSD, EfficientDet.

1.3. Các thuật toán Object Detection – Deep Learning

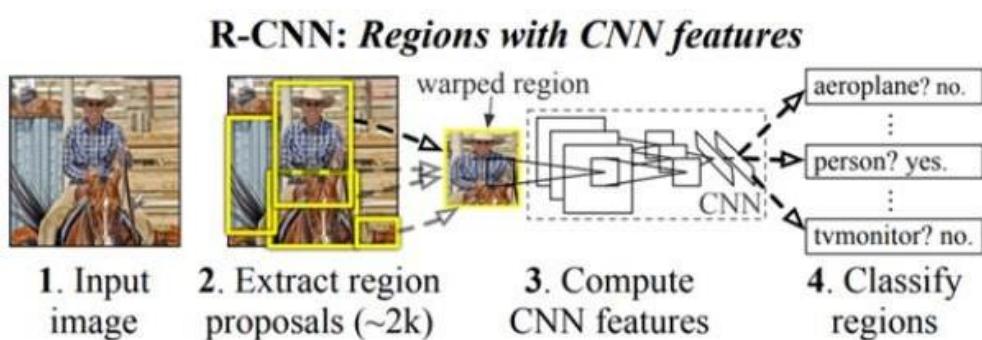
Thuật toán CNN - Convolutional Neural Network

Mạng nơ-ron phức hợp Convolutional Neural Network (CNN) là một mô hình tiên tiến trong lĩnh vực Học sâu. Với ưu điểm đặc biệt, CNN được áp dụng phổ biến trong bài toán nhận dạng đối tượng trên hình ảnh. Với khả năng này, nó giúp xây dựng những hệ thống thông minh với độ chính xác cao, mang lại kết quả đáng tin cậy.

1.3.1. R-CNN

Đây được xem là một trong những thuật toán Deep Learning đầu tiên giải quyết bài toán Computer Vision. Trước đó, đã có một số thuật toán khác (không phải Deep Learning) như Exhaustive Search. Tuy nhiên, nhược điểm chung của những thuật toán này là yêu cầu tài nguyên tính toán lớn và thời gian xử lý kéo dài.

Nhằm khắc phục những hạn chế đó, R-CNN đã đề xuất sử dụng phương pháp Selective Search để trích xuất thông tin từ khoảng 2000 khu vực trên bức ảnh (gọi là Region Proposal). Những thông tin này sau đó được đưa qua mạng CNN để xác định vị trí và phân loại đối tượng.

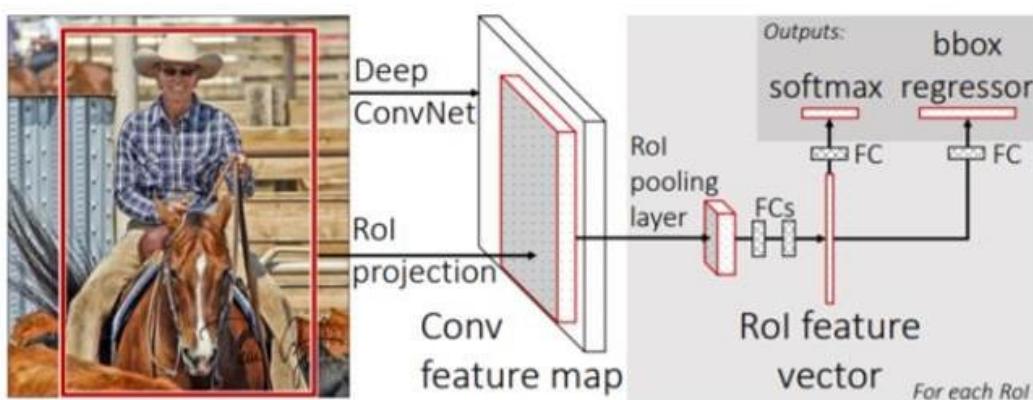


Hình 3. R-CNN

Tuy nhiên, R-CNN vẫn mất trung bình khoảng 50 giây để xử lý một bức ảnh. Đối với những bức ảnh có nhiều đối tượng, thời gian xử lý còn tăng lên đáng kể.

1.3.2. Fast R-CNN

Fast R-CNN được phát triển để khắc phục nhược điểm của R-CNN. Thay vì phân chia bức ảnh thành các đề xuất vùng (Region Proposals) và đưa chúng qua mạng CNN như R-CNN, Fast R-CNN thực hiện việc đưa toàn bộ bức ảnh qua mạng CNN một lần để tạo ra Bản đồ Đặc trưng (Feature Map). Từ Bản đồ Đặc trưng này, các đề xuất vùng mới được xác định và đưa qua mạng Fully Connected (FC). Cuối cùng, Softmax được sử dụng để dự đoán nhãn cho từng đối tượng và tính toán tọa độ của các Bounding Boxes tương ứng.

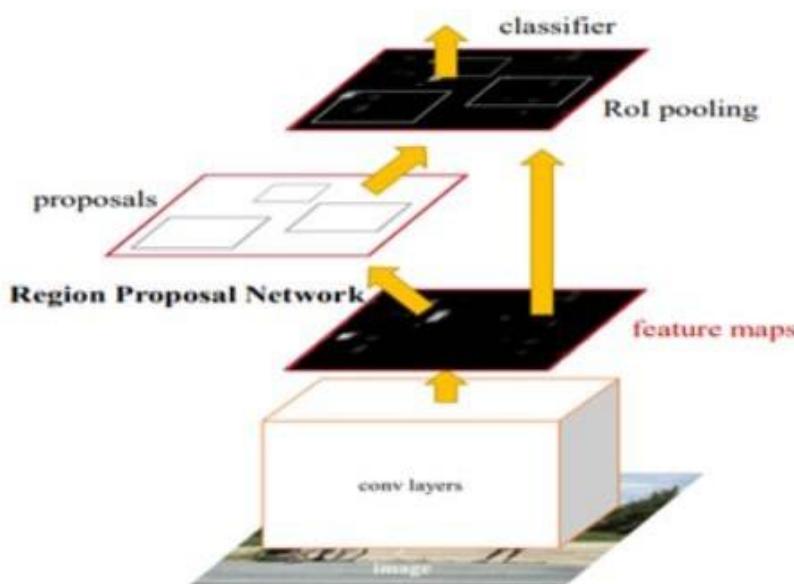


Hình 4. Fast R-CNN

Với việc chỉ thực hiện mạng CNN một lần trên mỗi bức ảnh, Fast R-CNN đạt được tốc độ nhanh hơn so với R-CNN.

1.3.3. Faster R-CNN

Tương tự như thuật toán Fast R-CNN, Faster R-CNN cũng sử dụng mạng CNN để tạo ra Feature Map từ toàn bộ bức ảnh. Tuy nhiên, điểm khác biệt đáng chú ý là Faster R-CNN sử dụng một Region Proposal Network (RPN) thay vì phương pháp Selective Search để tạo ra các Region Proposals. Sau đó, qua quá trình ROI Pooling, Faster R-CNN nhận các Region Proposals này làm đầu vào và tạo ra dự đoán nhãn và tọa độ Bounding Box cho mỗi đối tượng trong ảnh.



Hình 5. Faster R-CNN

Nếu so sánh về thời gian xử lý giữa ba thuật toán trong họ R-CNN, ta thấy sự giảm dần từ R-CNN sang Fast R-CNN và cuối cùng là Faster R-CNN.

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds

Hình 6. So sánh về mặt thời gian

1.3.4. YOLO – You Look Only Once

Các thuật toán trong họ R-CNN có thể được phân loại là nhóm Two Stages Detector vì chúng hoạt động theo 2 bước. Trước tiên, chúng xác định các vùng quan tâm (Region of Interest - ROI) trên bức ảnh, sau đó sử dụng mạng CNN để phân loại các ROI đó. Tuy nhiên, phương pháp này khiến cho tốc độ thực thi của các thuật toán này khá chậm.

Ngược lại, YOLO là một thuật toán Single Stage Detector (Single Shot Detector), tức là nó dự đoán nhãn và vị trí của đối tượng trực tiếp trên toàn bộ bức ảnh chỉ với một lần chạy thuật toán duy nhất. Điều này giúp YOLO có thời

gian xử lý nhanh, thích hợp cho các ứng dụng cần thời gian thực.

Như vậy, sự khác biệt giữa hai loại thuật toán này nằm ở cách tiếp cận và tốc độ xử lý. Các thuật toán trong họ R-CNN hoạt động theo hai giai đoạn, trong khi YOLO chỉ cần một giai đoạn và cho kết quả nhanh chóng.

2. Metrics đánh giá Object Detection model

Intersection over Union

Intersection over Union (IoU) là một phép đo để đánh giá mức độ chồng chéo giữa hai hộp giới hạn (bounding box). IoU là một phần quan trọng trong quá trình huấn luyện, nó tính toán sự chồng chéo giữa bounding box dự đoán và bounding box thực tế (bounding box đã được gán nhãn trước đó).

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Area of Overlap: diện tích phần giao nhau giữa predicted bounding box với growth-truth bounding box.

Area of Union: diện tích phần kết hợp predicted và growth-truth bounding box.

Trong tập training set và test set, chúng ta đã đánh nhãn các bounding box bằng cách thủ công.

Các chỉ tiêu được xét để đánh giá:

- + True Positive (TP): Đối tượng được nhận dạng với tỉ lệ $IoU > 0.5$.
- + False Positive (FP): Đối tượng được nhận dạng với tỉ lệ $IoU < 0.5$.
- + False Negative (FN): Có đối tượng trong hình nhưng model không nhận dạng được (không dự đoán được các bounding box).
- + True Negative (TN): Trong hình không có đối tượng và model cũng dự đoán là không có đối tượng.

Precision & Recall

+ Precision đo lường mức độ chính xác trong việc phát hiện các điểm Positive. Khi Precision càng cao, model sẽ nhận diện các điểm positive một cách chính xác hơn. Độ chính xác này được tính bằng công thức sau đây::

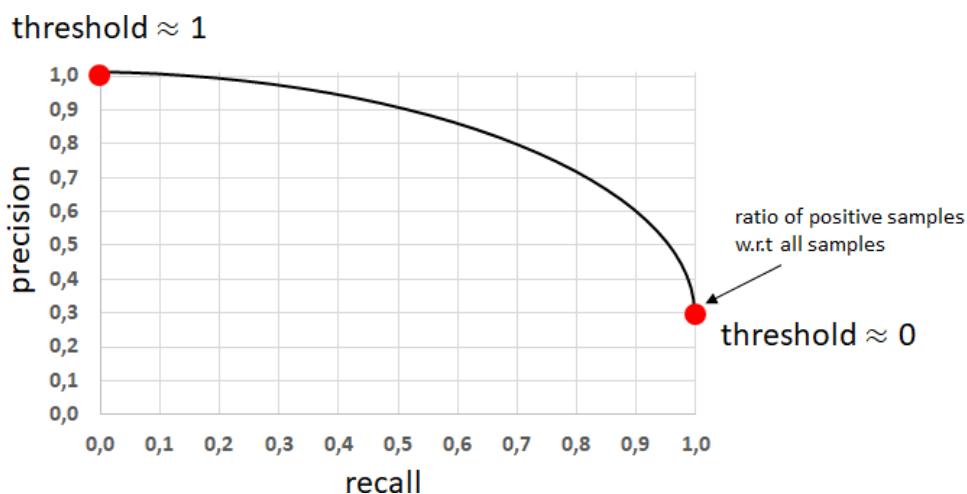
$$\text{Precision} = \frac{TP}{TP + FP}$$

+ Recall đo lường khả năng phát hiện các điểm Positive, tức là tỷ lệ bỏ sót các điểm positive. Khi Recall càng cao, khả năng bỏ sót các điểm positive sẽ càng thấp. Độ chính xác này được tính bằng công thức sau đây:

$$\text{Recall} = \frac{TP}{TP + FN}$$

AP (Average Precision) & mAP (Mean Average Precision)

+ Đồ thị Precision-Recall Curve được tạo ra bằng cách sử dụng giá trị IoU (Intersection over Union) và một ngưỡng (Threshold). Khi IoU lớn hơn ngưỡng, điểm được gọi là True Positive (được nhận diện chính xác); ngược lại, nó sẽ là False Positive (được nhận diện là Positive nhưng sai). Bằng cách thay đổi ngưỡng IoU, ta tính toán Precision và Recall tương ứng và vẽ đồ thị Precision-Recall Curve. Quá trình này thực hiện với Threshold chạy từ 0 đến 1 với bước nhảy là 0.001, như được mô tả trong Hình 7..



Hình 7. Đồ thị Precision Recall Curve

- + Precision-Recall Curve được vẽ ra để tính toán AP - Average Precision. AP đại diện cho phần diện tích nằm dưới đường cong Precision-Recall.
 - AP lớn khi có một vùng diện tích lớn nằm dưới đường cong, cho thấy Precision và Recall đồng đều cao ở các ngưỡng khác nhau. Điều này cho thấy mô hình đạt kết quả tốt.
 - AP nhỏ khi cả Precision và Recall đều thấp, cho thấy mô hình không hiệu quả.
- + Mỗi class sẽ có một giá trị AP riêng => mAP - Mean Average Precision là giá trị trung bình của các AP.

Anchor boxes

Anchor boxes là một kỹ thuật sử dụng để dự đoán các hộp giới hạn chồng lên nhau. Trong quá trình thực nghiệm, mạng neural có khả năng dự đoán nhiều hơn một hộp cùng lúc, trong đó mỗi dự đoán được ràng buộc bởi một tập hình học đã được xác định trước. Ví dụ, dự đoán đầu tiên có thể là một hộp hình chữ nhật theo một hình dạng nhất định, trong khi dự đoán thứ hai có thể là một hộp hình chữ nhật khác với hình dạng hình học khác.

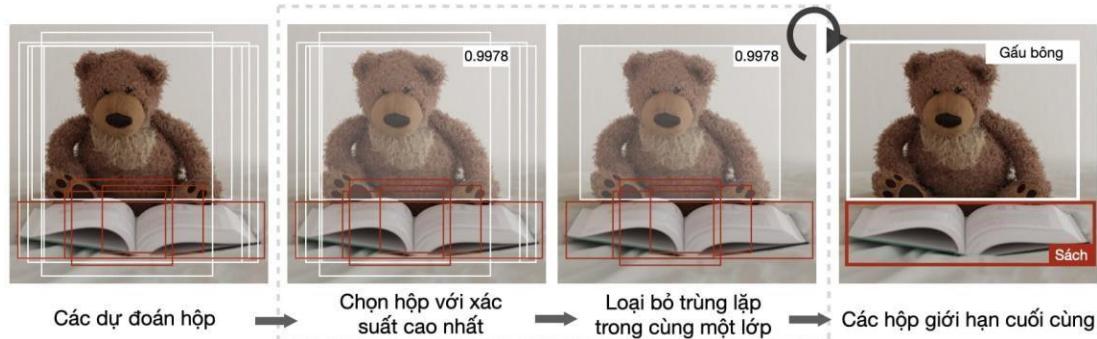
Non – max suppression

Kỹ thuật non-max suppression nhằm loại bỏ các hộp giới hạn trùng lặp của cùng một đối tượng bằng cách chỉ chọn hộp có tính đặc trưng cao nhất. Sau khi loại bỏ tất cả các hộp có xác suất dự đoán nhỏ hơn 0.6, quá trình này được lặp lại cho đến khi không còn tồn tại hộp nào khác..

Với một lớp cho trước

Bước 1. Chọn chiếc hộp có xác suất dự đoán lớn nhất.

Bước 2. Loại bỏ những hộp có $\text{IoU} \geq 0.5$ với hộp đã chọn.



Hình 8. Non – max suppression

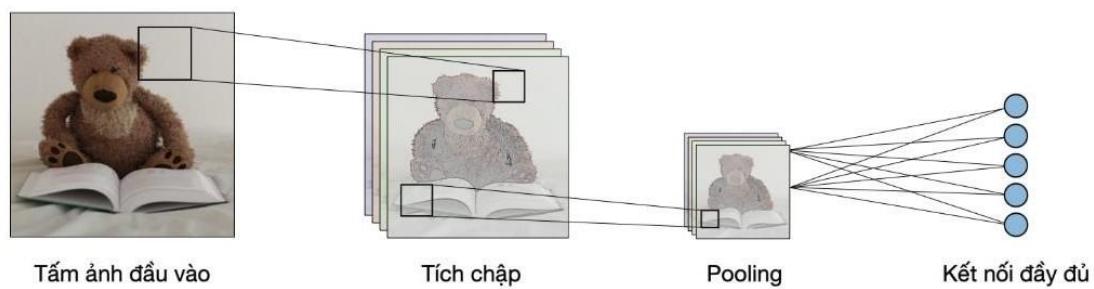
PHẦN III. NHẬN DIỆN BIỂN BÁO GIAO THÔNG

1. Tổng quan

Phần nội dung này em đã sử dụng kiến thức mạng CNN để thực hiện mục tiêu là nhận diện biển báo giao thông. Trong quá trình nhận diện biển báo giao thông, em đã tận dụng hai môi trường chính để xây dựng mô hình CNN: Google Colab để thực hiện quá trình huấn luyện và kiểm thử, và Pycharm để thực hiện việc huấn luyện và tạo giao diện kiểm thử - GUI..

1.1. Tổng quan về mạng neural tích chập - CNN

Kiến trúc truyền thông của một mạng CNN - Mạng neural tích chập (Convolutional Neural Networks), còn được biết đến với tên là CNNs, là một mạng neural được cấu thành bởi các tầng sau:



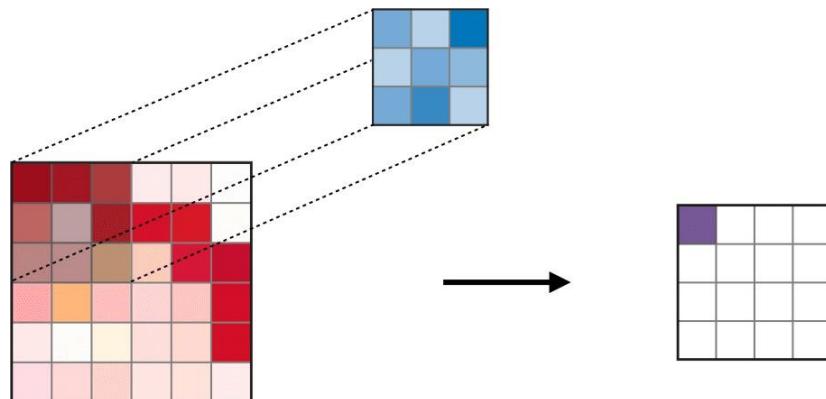
Hình 9. Mô hình kiến trúc mạng CNN

Tầng tích chập và tầng pooling có thể được hiệu chỉnh theo các siêu tham số (hyperparameters) được mô tả ở những phần tiếp theo.

1.2. Các kiểu tầng trong mạng CNN

1.2.1. Tầng tích chập (Convolutional layer – CONV)

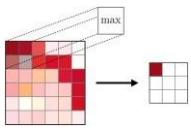
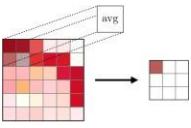
Tầng tích chập (CONV) sử dụng các bộ lọc để thực hiện phép tích chập khi đưa chúng đi qua đầu vào I theo các chiều của nó. Các siêu tham số của bộ lọc này bao gồm kích thước bộ lọc F và độ trượt S (Stride). Kết quả đầu ra O được gọi là feature map hay activation map.



Hình 10. Convolutional Layer

1.2.2. Tầng Pooling – POOL

Tầng pooling (POOL) là một phép downsampling, thường được sử dụng sau tầng tích chập, giúp tăng tính bất biến không gian. Cụ thể, max pooling và average pooling là những dạng pooling đặc biệt, mà tương ứng là trong đó giá trị lớn nhất và giá trị trung bình được lấy ra.

Kiểu	Max Pooling	Average Pooling
Chức năng	Từng phép pooling chọn giá trị lớn nhất trong khu vực mà nó đang được áp dụng	Từng phép pooling chọn giá trị trung bình trong khu vực mà nó đang được áp dụng
Minh họa		
Nhận xét	Bảo toàn các đặc trưng đã phát hiện. Được sử dụng thường xuyên.	Giảm kích thước feature map. Được sử dụng trong Lenet.

Bảng 1. Max Pooling & Average Pooling

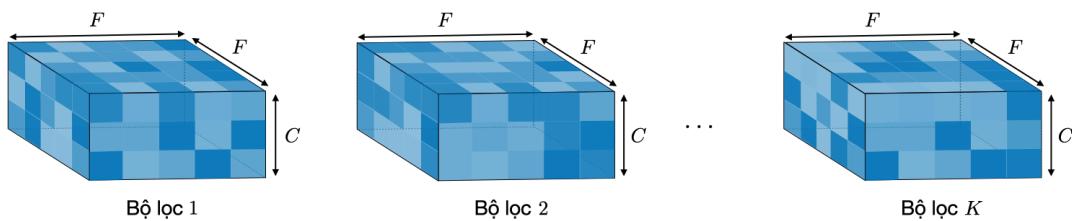
1.2.3. Fully Connected – FC

Tầng kết nối đầy đủ (FC) nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.

1.3. Các siêu tham số của bộ lọc

Các chiều của bộ lọc

Một bộ lọc kích thước $F \times F$ áp dụng lên đầu vào chứa C kênh (channels) thì có kích thước tổng kề là $F \times F \times C$ thực hiện phép tích chập trên đầu vào kích thước $I \times I \times C$ và cho ra một feature map (hay còn gọi là activation map) có kích thước $O \times O \times 1$.

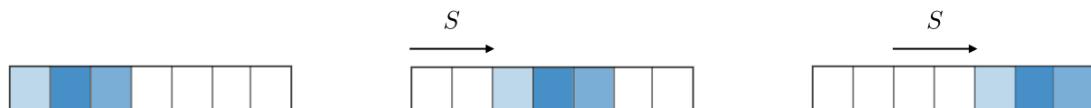


Hình 11. Dimensions - filter

Lưu ý: Việc áp dụng K bộ lọc có kích thước $F \times F$ cho ra một feature map có kích thước $O \times O \times K$.

Stride

Đối với phép tích chập hoặc phép pooling, độ trượt S ký hiệu số pixel mà cửa sổ sẽ di chuyển sau mỗi lần thực hiện phép tính.

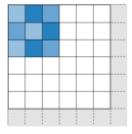
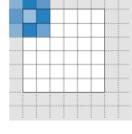


Hình 12. Stride

Zero-padding

Zero-padding là tên gọi của quá trình thêm P số không vào các biên của đầu vào. Giá trị này có thể được lựa chọn thủ công hoặc một cách tự động bằng một trong ba những phương pháp mô tả bên dưới:

Phương pháp	Valid	Same	Full
-------------	-------	------	------

Giá trị	$P = 0$	$P_{start} = \frac{S \left[\frac{I}{S} \right] - I + F - S}{2}$ $P_{end} = \frac{S \left[\frac{I}{S} \right] - I + F - S}{2}$	$P_{start} \in [0, F - 1]$ $P_{end} = F - 1$
Minh họa			
Mục đích	<ul style="list-style-type: none"> + Không sử dụng Padding. + Bỏ tích chập cuối nếu số chiều không khớp. 	<ul style="list-style-type: none"> + Sử dụng Padding làm cho feature map có kích thước $\left[\frac{I}{S} \right]$ + Kích thước đầu ra thuận lợi về mặt toán học. + Còn được gọi là ‘half’ padding. 	<ul style="list-style-type: none"> + Padding tối đa sao cho các phép tích chập có thể được sử dụng tại các rìa của đầu vào. + Bộ lọc thấy được đầu vào từ đầu đến cuối.

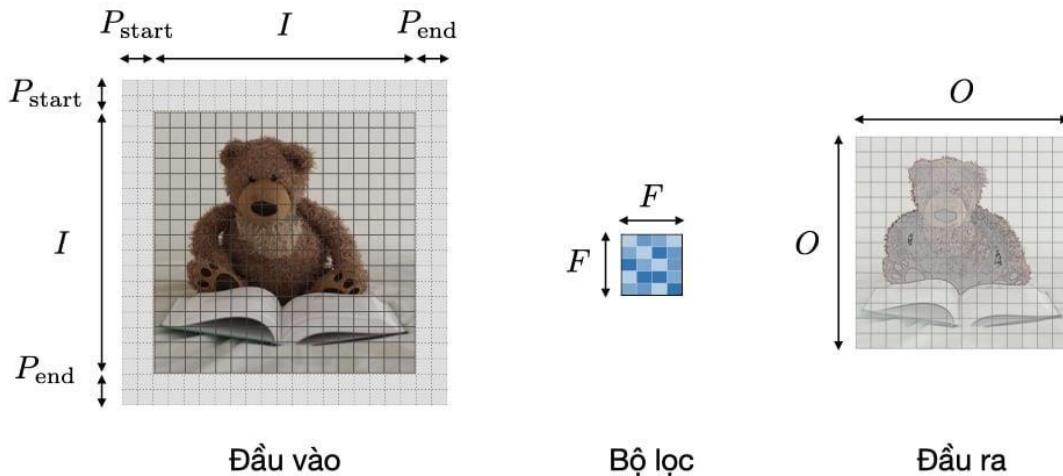
Bảng 2. Zero Padding

1.4. Điều chỉnh siêu tham số

Tính tương thích của tham số trong tầng tích chập

Bằng cách ký hiệu I là độ dài kích thước đầu vào, F là độ dài của bộ lọc, P là số lượng zero padding, S là độ trượt, ta có thể tính được độ dài O của feature map theo một chiều bằng công thức: Bằng cách ký hiệu I là độ dài kích thước đầu vào, F là độ dài của bộ lọc, P là số lượng zero padding, S là độ trượt, ta có thể tính được độ dài O của feature map theo một chiều bằng công thức:

$$O = \frac{I - F + P_{start} + P_{end}}{S} + 1$$



Hình 13. Parameter compatibility

Lưu ý: Trong một số trường hợp $P_{\text{start}}=P_{\text{end}} \triangleq P$, ta có thể thay thế $P_{\text{start}}+P_{\text{end}}$ bằng $2P$ trong công thức trên.

Understanding the complexity of the model

Để đánh giá độ phức tạp của một mô hình, cách hữu hiệu là xác định số tham số mà mô hình đó sẽ có. Trong một tầng của mạng neural tích chập, nó sẽ được tính toán như sau:

	CONV	POOL	FC
Mô hình	$\begin{matrix} F \\ \uparrow \downarrow \\ \otimes C \end{matrix} \times K$	$\begin{matrix} F \\ \uparrow \downarrow \\ \max \end{matrix}$	
Kích thước đầu vào	$I \times I \times C$	$I \times I \times C$	N_{in}
Kích thước đầu ra	$O \times O \times K$	$O \times O \times C$	N_{out}
Số lượng tham số	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Lưu ý	<ul style="list-style-type: none"> Một tham số bias với mỗi bộ lọc Trong đa số trường hợp, $S < F$ Một lựa chọn phổ biến cho K là $2C$ 	<ul style="list-style-type: none"> Phép pooling được áp dụng lên từng kênh (channel-wise) Trong đa số trường hợp, $S = F$ 	<ul style="list-style-type: none"> Đầu vào được làm phẳng Mỗi neuron có một tham số bias Số neuron trong một tầng FC phụ thuộc vào ràng buộc kết cấu

Bảng 3. Bảng đánh giá độ phức tạp của một mô hình

Receptive field - Trường thụ cảm

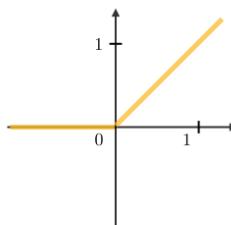
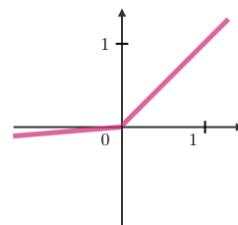
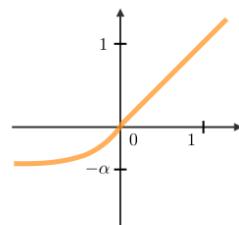
Trường thụ cảm (receptive field) tại tầng k là vùng được ký hiệu $Rk \times Rk$ của đầu vào mà những pixel của activation map thứ k có thể "nhìn thấy". Bằng cách gọi F_j là kích thước bộ lọc của tầng j và S_i là giá trị độ trượt của tầng i và để thuận tiện, ta mặc định $S0=1$, trường thụ cảm của tầng k được tính toán bằng công thức:

$$R_k = 1 + \sum_{j=1}^k (F_j - 1) GS_i$$

1.5. Các hàm kích hoạt thường gặp

Rectified Linear Unit – ReLU

Tầng rectified linear unit (ReLU) là một hàm kích hoạt g được sử dụng trên tất cả các thành phần. Mục đích của nó là tăng tính phi tuyến tính cho mạng. Những biến thể khác của ReLU được tổng hợp ở bảng dưới:

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ với $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ với $\alpha \ll 1$
		
• Độ phức tạp phi tuyến tính có thể thông dịch được về mặt sinh học	• Gán vấn đề ReLU chết cho những giá trị âm	• Khả vi tại mọi nơi

Bảng 4. Các biến thể của ReLU

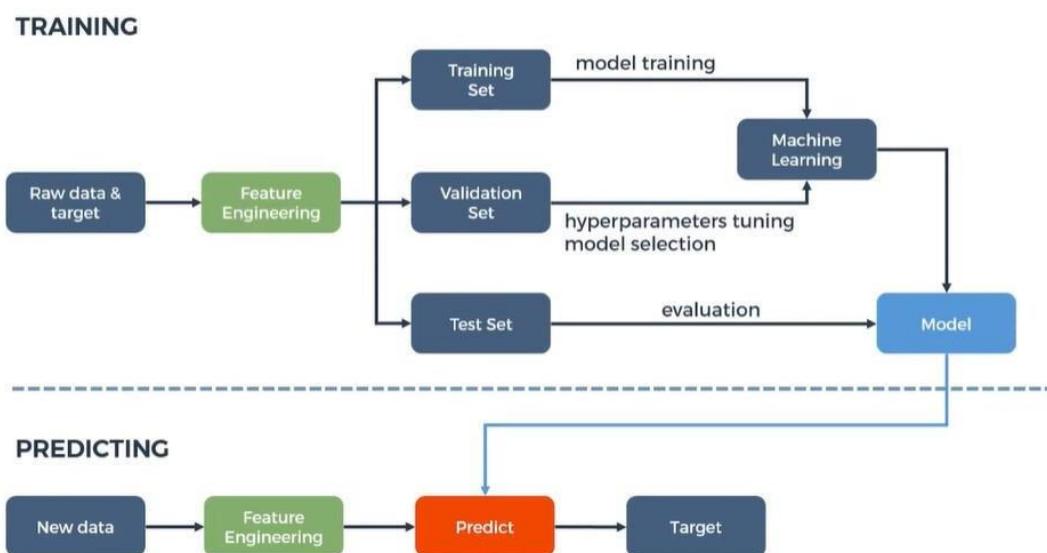
Softmax

Bước softmax có thể được coi là một hàm logistic tổng quát lấy đầu vào là một vector chứa các giá trị $x \in R^n$ và cho ra là một vector gồm các xác suất $p \in R^n$ thông qua một hàm softmax ở cuối kiến trúc. Nó được định nghĩa như sau:

$$p = \left(\frac{p_1}{p_n} \right) \text{ với } p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

2. Xây dựng kiến trúc CNN trên Google Colab

Ở phần này nhóm đã sử dụng mô hình Training và Predicting như trong **Hình 13** ở dưới đây.



Hình 14. Mô hình Training + Predicting

Có thể thấy trong **Hình 13** quá trình Training được thực hiện bằng cách đưa dữ liệu thô được đưa vào sau đó Feature Engineering xử lý dữ liệu thô và chia thành 3 bộ: Training Set, Validation Set và Test Set. Trong quá trình Training ta sử dụng 2 bộ chính là Training Set và Test Set, trong đó bộ Training Set dùng để training model còn bộ Validation Set dùng để kiểm tra mức độ chính xác sau mỗi bước train để kiểm tra độ hiệu quả của quá trình training. Sau khi kiểm tra mức độ chính xác thì model sẽ dựa vào data trong bộ Test Set để đưa ra được dự đoán thực. Quá trình Predicting: dữ liệu đầu vào không được gán nhãn, do đó phải dựa vào model đưa ra dự đoán thực để đưa ra được nhãn mới cho data.

2.1. Import thư viện, sử dụng back-end Tensorflow

Bước 1. Kết nối với Drive và cài đặt Back-end Tensorflow

```

5s [2] from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

0s [3] %cd /content/drive/MyDrive/Project_Traffic_Sign_Detection_final/github_ABC
      /content/drive/MyDrive/Project_Traffic_Sign_Detection_final/github_ABC

0s [4] %tensorflow_version 1.x
      TensorFlow 1.x selected.

```

Hình 15. Kết nối Drive và cài đặt Tensorflow

Trong phần này lựa chọn Tensorflow 1.x như trong **Hình 14**

Sau khi liên kết giữa colab với drive, việc tiếp theo cả nhóm cần làm là lựa chọn back-end tensorflow_version 1.x cho project. Việc lựa chọn version 1.x thay vì lựa chọn version 2.x như hiện tại vì lí do về hiệu suất làm việc của TF1, đạt tốc độ nhanh hơn từ 47% đến 276% so với TF2, Bảng 5 dưới đây là thông số kỹ thuật về kết quả điểm chuẩn giữa 2 version TF:

keras.layers + keras.optimizers.Adam

Model size	TF 2.0.0, Keras 2.3.0	TF 1.14.0, Keras 2.2.5	Ratio (TF2:TF1)
Small	0.0830 sec	0.0398 sec	2.09
Medium	16.6 sec	6.69 sec	2.48

tf.keras.layers + tf.keras.optimizers.Adam

Model size	TF 2.0.0, Keras 2.3.0	TF 1.14.0, Keras 2.2.5	Ratio (TF2:TF1)
Small	0.128 sec	0.0340 sec	3.76
Medium	11.3 sec	7.69 sec	1.47

Bảng 5. So sánh thông số kỹ thuật của TF1 & TF2

Bước 2. Import thư viện

```

0s [15] import numpy as np
       import matplotlib.pyplot as plt
       import keras
       import cv2
       from keras.models import Sequential
       from keras.optimizers import Adam
       from keras.layers import Dense
       from keras.utils.np_utils import to_categorical
       from keras.layers import Dropout, Flatten
       from keras.layers.convolutional import Conv2D, MaxPooling2D
       import pickle
       import random
       import pandas as pd

```

Hình 16. Import thư viện

Thư viện Numpy: là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

Thư viện Matplotlib: là một thư viện sử dụng để vẽ các đồ thị trong Python.

Thư viện Keras: Keras là một open source cho Neural Network, kết hợp tạo Convolutional Layers: Conv2D là convolution dùng để lấy feature từ ảnh với các tham số. Pooling Layers: sử dụng để làm giảm param khi train, nhưng vẫn giữ được đặc trưng của ảnh. Ngoài ra còn có: MaxPooling2D, AveragePooling1D, 2D với từng size.

Thư viện Pickle: được sử dụng để thực hiện chuyển đổi các cấu trúc đối tượng Python sang một dạng byte để có thể được lưu trữ trên ổ đĩa hoặc được gửi qua mạng. Sau đó, luồng ký tự này sau đó có thể được truy xuất và chuyển đổi trở lại sang dạng đối tượng ban đầu trong Python.

Pandas là một thư viện Python cung cấp các cấu trúc dữ liệu nhanh, mạnh mẽ, linh hoạt. Pandas được thiết kế để làm việc dễ dàng và trực quan với dữ liệu có cấu trúc (dạng bảng, đa chiều, không đồng nhất) và dữ liệu chuỗi thời gian.

2.2. Import Data

Bước 1. Thêm link chứa Data cần thực thi

```
[ ] !git clone https://bitbucket.org/jadslim/german-traffic-signs
```

Bước 2. Đọc Data

```
[ ] with open('german-traffic-signs/train.p', 'rb') as f:
    train_data = pickle.load(f)
with open('german-traffic-signs/valid.p', 'rb') as f:
    val_data = pickle.load(f)
with open('german-traffic-signs/test.p', 'rb') as f:
    test_data = pickle.load(f)
```

Bước 3. Tách tập dữ liệu

```

✓ [19] X_train, y_train = train_data['features'], train_data['labels']
      X_val, y_val = val_data['features'], val_data['labels']
      X_test, y_test = test_data['features'], test_data['labels']

✓ [20] print(X_train.shape)
      print(X_val.shape)
      print(X_test.shape)

(34799, 32, 32, 3)
(4410, 32, 32, 3)
(12630, 32, 32, 3)

```

Bước 4. Check dữ liệu – Giá trị tuyệt đối của đối số đầu ra

```

✓ [21] assert(X_train.shape[0] == y_train.shape[0]), "The number of images is not equal to the number of labels"
      assert(X_val.shape[0] == y_val.shape[0]), "The number of images is not equal to the number of labels"
      assert(X_test.shape[0] == y_test.shape[0]), "The number of images is not equal to the number of labels"
      assert(X_train.shape[1:] == (32, 32, 3)), "The dimensions of the image is not 32*32*3"
      assert(X_val.shape[1:] == (32, 32, 3)), "The dimensions of the image is not 32*32*3"
      assert(X_test.shape[1:] == (32, 32, 3)), "The dimensions of the image is not 32*32*3"

```

Để thực hiện được chương trình ta cần thêm các file data bao gồm ‘train.p’ chưa các dữ liệu phục vụ cho quá trình training, ‘file val.p’ chứa tất cả các giá trị liên quan đến mục ảnh bao gồm Name, ID. Và cuối cùng là ‘file test.p’ chứa các ảnh thực ngẫu nhiên phục vụ cho quá trình testing. Thư viện pickle đã thêm giúp chương trình có thể đọc được dữ liệu của các file đã thêm và sử dụng assert để kiểm tra dữ liệu và xét giá trị tuyệt đối của đối số đầu ra.

2.3. Data Visualisation

Bước 1. Giúp chương trình có thể trực quan được dữ liệu

```

✓ [22] data = pd.read_csv('german-traffic-signs/signnames.csv')

num_of_samples = []

cols = 5
num_classes = 43

fig, axs = plt.subplots(nrows = num_classes, ncols = cols, figsize = (5, 50))
fig.tight_layout()

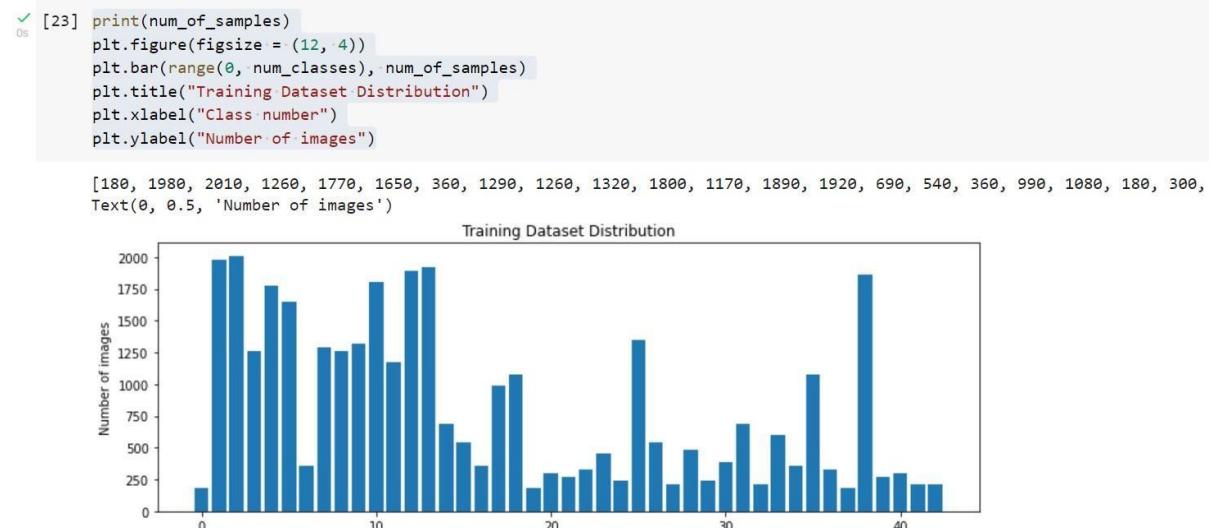
for i in range(cols):
    for j, row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, (len(x_selected)-1)), :, :], cmap = plt.get_cmap("gray"))
        axs[j][i].axis("off")
    if i == 2:
        axs[j][i].set_title(str(j) + "_" + row["SignName"])
        num_of_samples.append(len(x_selected))

```

Kết quả phân trực quan



Bước 2. Phân phối tập dữ liệu để training



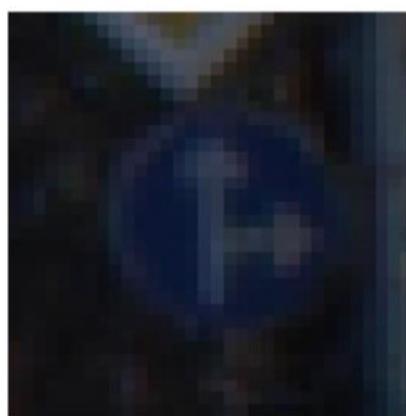
Ở mục này, thư viện matplotlib đã thêm được sử dụng để trực quan hóa dữ liệu, trong thư viện này chứa các module có sẵn để vẽ đồ thị khác nhau, giúp nhóm có được những thông kê liên quan đến các file ảnh được sử dụng trong quá trình training. Việc phân phối tập dữ liệu training giúp quá trình thực thi trở lên đơn giản hơn và có được hiệu quả cao nhất.

2.4. Khởi tiền xử lý

Bước 1. Hiển thị hình ảnh trên các trục

```
✓ 0s  plt.imshow(X_train[1000])
    plt.axis('off')
    print(X_train[1000].shape)
    print(y_train[1000])
```

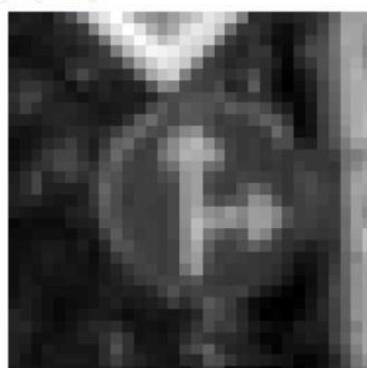
```
⌚ (32, 32, 3)
36
```

Bước 2. Sử dụng OpenCV để tách màu và xử lý ảnh (thư viện màu COLOR_BGR2GRAY)

```
[ ] def grayscale(img):
    image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plt.axis('off')
    return image
```

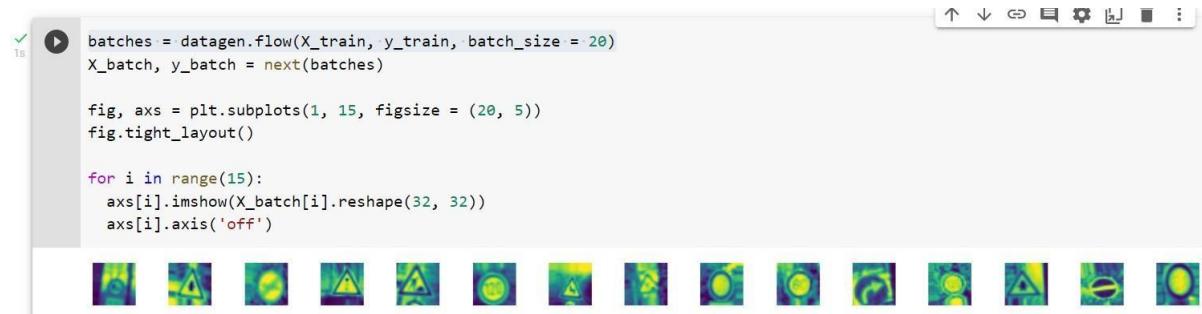
```
[ ] img = grayscale(X_train[1000])
plt.imshow(img, cmap = 'gray')
print(img.shape)
```

```
(32, 32)
```

Bước 3. Khởi tạo các mảng

```
X_train = np.array(list(map(preprocessing, X_train)))
X_val = np.array(list(map(preprocessing, X_val)))
X_test = np.array(list(map(preprocessing, X_test)))
```

Bước 4. Mô hình theo từng batch và tiến hành phát triển dữ liệu theo thời gian thực



```
1s    batches = datagen.flow(X_train, y_train, batch_size = 20)
      X_batch, y_batch = next(batches)

      fig, axs = plt.subplots(1, 15, figsize = (20, 5))
      fig.tight_layout()

      for i in range(15):
          axs[i].imshow(X_batch[i].reshape(32, 32))
          axs[i].axis('off')
```

The screenshot shows a Jupyter Notebook cell with Python code. The code uses a generator `datagen` to produce batches of images from `X_train` and their corresponding labels from `y_train`. A batch size of 20 is specified. The generated batches are then plotted in a 1x15 grid using `plt.subplots` and `imshow`. The images are 32x32 pixel grayscale plots of handwritten digits. The notebook interface includes a toolbar at the top and a status bar at the bottom.

Bước 5. Chuyển đổi các nhãn thành đoạn mã hóa

```
[34] 0s    y_train = to_categorical(y_train, 43)
      y_val = to_categorical(y_val, 43)
      y_test = to_categorical(y_test, 43)
```

Các file ảnh được biểu diễn trên trực của đồ thị xOy (2D), OpenCV được sử dụng giúp module xử lý ảnh gồm cả lọc hình ảnh tuyến tính và phi tuyến, các phép biến đổi hình học, chuyển đổi màu trong không gian và phát hiện các đặc tính nổi bật của bộ nhận diện và bộ truy xuất thông số. Bên cạnh là sự kết hợp của việc sử dụng numpy qua câu lệnh `np.array` để khởi tạo các mảng, tiếp đến là việc mô hình theo từng batch và phát triển dữ liệu với thời gian thực và chuyển đổi các nhãn thành đoạn mã hóa

2.5. Mô hình CNN

Bước 1. Xây dựng mô hình CNN

```

✓ [35] def neural_model():
    model = Sequential()
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(30, (3, 3), activation = 'relu'))
    model.add(Conv2D(30, (3, 3), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2, 2)))

    #model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(500, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = 'softmax'))
    model.compile(Adam(lr = 0.001), loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model

```

Bước 2. Kết quả của model.summary() và sắp xếp dữ liệu theo tuần tự

```

✓ [36] WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/ops
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_k

Model: "sequential_1"

Layer (type)          Output Shape         Param #
=====
conv2d_1 (Conv2D)      (None, 28, 28, 60)     1560
=====
conv2d_2 (Conv2D)      (None, 24, 24, 60)     90060
=====
max_pooling2d_1 (MaxPooling2D) (None, 12, 12, 60)   0
=====
conv2d_3 (Conv2D)      (None, 10, 10, 30)     16230
=====
conv2d_4 (Conv2D)      (None, 8, 8, 30)      8130
=====
max_pooling2d_2 (MaxPooling2D) (None, 4, 4, 30)   0
=====
flatten_1 (Flatten)    (None, 480)            0
=====
dense_1 (Dense)        (None, 500)            240500
=====
dropout_1 (Dropout)    (None, 500)            0

```

Bước 3. Tiến hành quá trình training

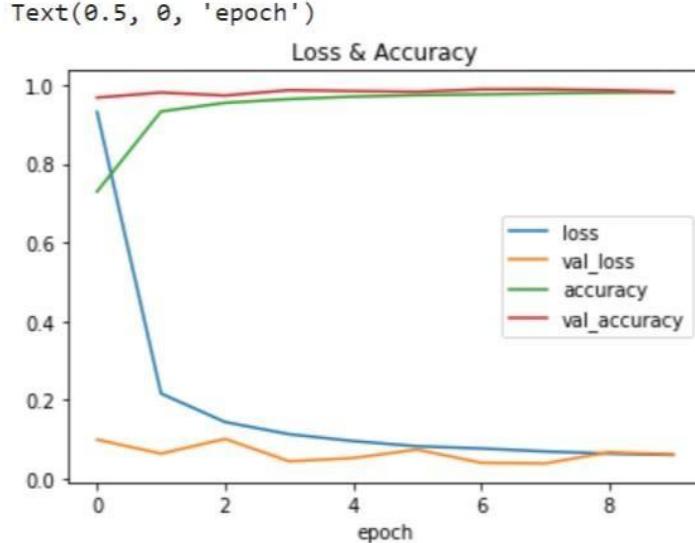
```
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size = 50), steps_per_epoch = 2000, epochs = 10, validation_data = (X_val, y_val))

WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.
```

Epoch 1/10
2000/2000 [=====] - 45s 22ms/step - loss: 0.9322 - accuracy: 0.7296 - val_loss: 0.0982 - val_accuracy: 0.0625
Epoch 2/10
2000/2000 [=====] - 43s 21ms/step - loss: 0.2154 - accuracy: 0.9334 - val_loss: 0.0625 - val_accuracy: 0.0625
Epoch 3/10
2000/2000 [=====] - 41s 20ms/step - loss: 0.1429 - accuracy: 0.9552 - val_loss: 0.1001 - val_accuracy: 0.0625
Epoch 4/10
2000/2000 [=====] - 35s 18ms/step - loss: 0.1120 - accuracy: 0.9647 - val_loss: 0.0431 - val_accuracy: 0.0625
Epoch 5/10
2000/2000 [=====] - 35s 18ms/step - loss: 0.0945 - accuracy: 0.9710 - val_loss: 0.0512 - val_accuracy: 0.0625
Epoch 6/10
2000/2000 [=====] - 35s 18ms/step - loss: 0.0813 - accuracy: 0.9749 - val_loss: 0.0728 - val_accuracy: 0.0625
Epoch 7/10
2000/2000 [=====] - 36s 18ms/step - loss: 0.0757 - accuracy: 0.9766 - val_loss: 0.0394 - val_accuracy: 0.0625
Epoch 8/10
2000/2000 [=====] - 34s 17ms/step - loss: 0.0681 - accuracy: 0.9792 - val_loss: 0.0376 - val_accuracy: 0.0625
Epoch 9/10
2000/2000 [=====] - 34s 17ms/step - loss: 0.0627 - accuracy: 0.9811 - val_loss: 0.0663 - val_accuracy: 0.0625
Epoch 10/10
2000/2000 [=====] - 35s 18ms/step - loss: 0.0591 - accuracy: 0.9823 - val_loss: 0.0606 - val_accuracy: 0.0625

Bước 4. Vẽ đồ thị về độ chính xác

```
[40] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.legend(['loss', 'val_loss', 'accuracy', 'val_accuracy'])
    plt.title('Loss & Accuracy')
    plt.xlabel('epoch')
```



Bước 5. Show kết quả train

```
[ ] score = model.evaluate(X_test, y_test, verbose = 1)
print('Test Score', score[0])
print('Test Accuracy', score[1])

12630/12630 [=====] - 1s 95us/step
Test Score 0.1556710412572101
Test Accuracy 0.9661124348640442
```

2.6. Module TEST

Bước 1. Thêm ảnh Test

```
[ ] import requests
from PIL import Image
url = 'https://c8.alamy.com/comp/A0RX23/cars-and-automobiles-must-turn-left-ahead-sign-A0RX23.jpg'
r = requests.get(url, stream=True)
image = Image.open(r.raw)
plt.axis('off')
plt.imshow(image, cmap=plt.get_cmap('gray'))
```

<matplotlib.image.AxesImage at 0x7f46480b4410>



Bước 2. Xử lý ảnh Test bằng OpenCV

```
[ ] img = np.asarray(image)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
img = img.reshape(1, 32, 32, 1)
```

(32, 32)



Bước 3. Dự đoán ảnh

```

▶ prediction = str(model.predict_classes(img))

prediction = prediction[1:-1]

print("predicted sign: "+ prediction)

predicted sign: 35

```

Bước 4. Show kết quả ảnh Test

```

✓ [45] for num, name in data.iteritems():
    name = name.values
    print("predicted sign: "+ str(name[pred]))

```

```

predicted sign: 35
predicted sign: Ahead only

```

**3. Pycharm (Module Train + GUI)**

Trong phần nhận diện biển báo thực hiện trên Pycharm nhóm đã sử dụng thư viện biển báo của Việt Nam.

3.1. Module TrainBước 1. Import thư viện

The screenshot shows a code editor with a file tree on the left and a code editor on the right. The file tree includes 'Lib', 'Scripts', 'share', 'TEST', 'TRAIN', '.gitignore', 'GUI.py', 'Module_Test.py', 'my_model.h5', 'myData.csv', 'Mô hình CNN.py', 'pyvenv.cfg', and 'TrafficSignTrain.py'. The code editor displays the following Python script:

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import cv2
5 import tensorflow as tf
6 from PIL import Image
7 import os
8 from sklearn.model_selection import train_test_split
9 from keras.utils import to_categorical
10 from keras.models import Sequential, load_model
11 from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
12
13 data = []
14 labels = []
15 classes = 28
16 cur_path = os.getcwd()
17

```

Bước 2. Truy xuất hình ảnh và nhãn của chúng

The code editor shows the implementation of the image loading loop. The script iterates through each class, finds the 'TRAIN' directory, lists all files, and then tries to open each file as an image, resize it to 30x30, and convert it to a numpy array. It then appends the image to the 'data' list and the class index to the 'labels' list. If an error occurs, it prints an error message.

```

#Truy xuất hình ảnh và nhãn của chúng
for i in range(classes):
    path = os.path.join(cur_path, 'TRAIN', str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            image = image.resize((30,30))
            image = np.array(image)

            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")

```

Bước 3. Chuyển đổi danh sách thành mảng trống

The code editor shows the conversion of the lists 'data' and 'labels' into numpy arrays. It prints the shapes of the resulting arrays.

```

#Chuyển đổi danh sách thành mảng trống
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)

```

Bước 4. Tách tập dữ liệu và đào tạo

The code editor shows the splitting of the dataset into training and testing sets using the 'train_test_split' function from 'sklearn.model_selection'. It prints the shapes of the resulting training and testing sets.

```

#Tách tập dữ liệu đào tạo và thử nghiệm
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=28)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

```

Bước 5. Chuyển đổi các nhãn thành một mã hóa

```
#Chuyển đổi các nhãn thành một mảng hóa
y_train = to_categorical(y_train, 29)
y_test = to_categorical(y_test, 29)
```

Bước 6. Xây dựng mô hình CNN

```
#Xây dựng mô hình CNN
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(29, activation='softmax'))
```

Bước 7. Tối ưu hóa mô hình

```
#Tối ưu hóa mô hình
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
```

Bước 8. Vẽ đồ thị chính xác

```
#Vẽ đồ thị về độ chính xác
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

Bước 9. Kiểm tra độ chính xác trên tập dữ liệu kiểm tra

```
#Kiểm tra độ chính xác trên tập dữ liệu kiểm tra
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('myData.csv')
labels = y_test["ClassId"].values
imgs = y_test["Name"].values
data=[]
for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))

X_test=np.array(data)

pred = model.predict_classes(X_test)
```

3.2. Kết quả module Train

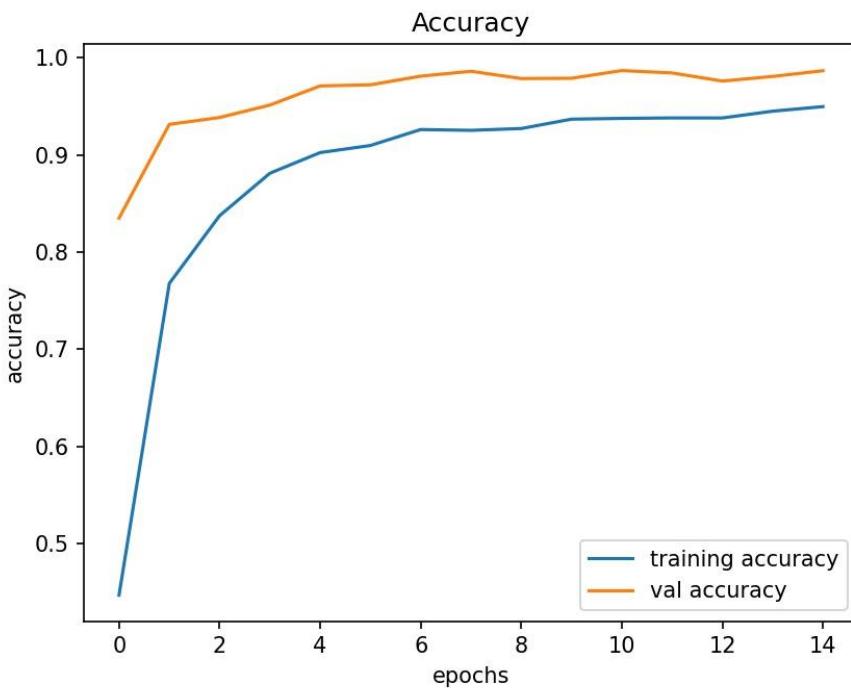
Quá trình train với Epoch là 15 lần

```
Epoch 1/15
870/870 [=====] - 96s 108ms/step - loss: 2.2331 - accuracy: 0.4472 - val_loss: 0.7487 - val_accuracy: 0.8349
Epoch 2/15
870/870 [=====] - 90s 104ms/step - loss: 0.8071 - accuracy: 0.7677 - val_loss: 0.2886 - val_accuracy: 0.9313
Epoch 3/15
870/870 [=====] - 91s 104ms/step - loss: 0.5609 - accuracy: 0.8374 - val_loss: 0.2193 - val_accuracy: 0.9384
Epoch 4/15
870/870 [=====] - 94s 108ms/step - loss: 0.4106 - accuracy: 0.8810 - val_loss: 0.1625 - val_accuracy: 0.9511
Epoch 5/15
870/870 [=====] - 92s 106ms/step - loss: 0.3382 - accuracy: 0.9023 - val_loss: 0.0975 - val_accuracy: 0.9708
Epoch 6/15
870/870 [=====] - 96s 111ms/step - loss: 0.3117 - accuracy: 0.9096 - val_loss: 0.1060 - val_accuracy: 0.9720
Epoch 7/15
870/870 [=====] - 98s 112ms/step - loss: 0.2651 - accuracy: 0.9260 - val_loss: 0.0675 - val_accuracy: 0.9809
Epoch 8/15
870/870 [=====] - 93s 107ms/step - loss: 0.2683 - accuracy: 0.9252 - val_loss: 0.0491 - val_accuracy: 0.9859
Epoch 9/15
870/870 [=====] - 90s 103ms/step - loss: 0.2552 - accuracy: 0.9270 - val_loss: 0.0775 - val_accuracy: 0.9784
Epoch 10/15
870/870 [=====] - 93s 107ms/step - loss: 0.2279 - accuracy: 0.9367 - val_loss: 0.0746 - val_accuracy: 0.9787
Epoch 11/15
870/870 [=====] - 91s 105ms/step - loss: 0.2275 - accuracy: 0.9375 - val_loss: 0.0483 - val_accuracy: 0.9866
```

Hình 17. Quá trình train – 15 lần

Biểu đồ về sự chính xác

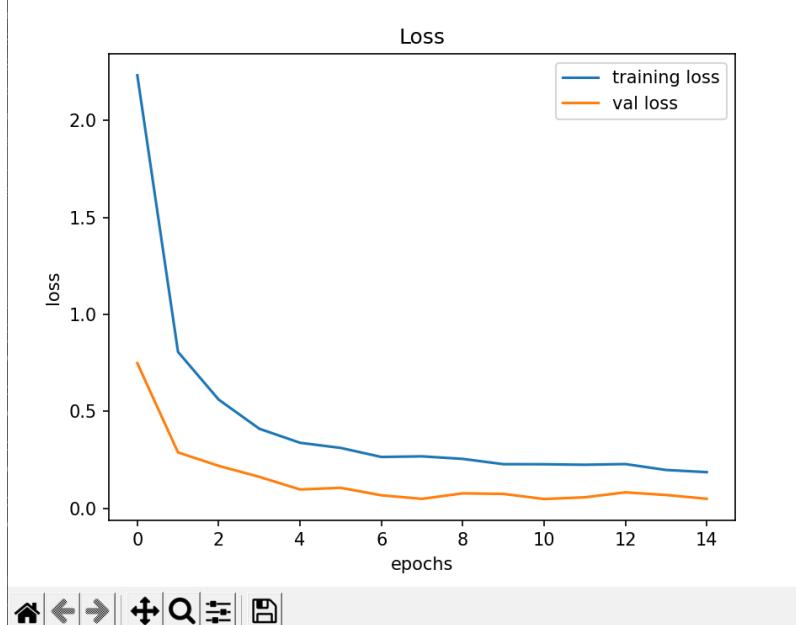
Figure 0



Hình 18. Biểu đồ độ chính xác

Biểu đồ về sai số

Figure 1



Hình 19. Biểu đồ sai số

3.3. Module GUI

Bước 1. Import thư viện và tải mô hình đã được train

```

6   import numpy
7   # load the trained model to classify sign
8   from keras.models import load_model
9   import tkinter as tk
10  from tkinter import filedialog
11  from tkinter import *
12  from PIL import ImageTk, Image
13
14  import numpy
15  # load the trained model to classify sign
16  from keras.models import load_model
17
18  model = load_model('my_model.h5')
19

```

Bước 2. Gán nhãn các loại biển báo

```

classes = { 0:'Cấm đi ngược chiều',
            1:'Cấm ô tô',
            2:'Cấm ô tô rẽ phải',
            3:'Cấm xe máy',
            4:'Cấm xe ô tô và xe gắn máy',
            5:'Cấm rẽ trái',
            6:'Cấm người đi bộ',
            7:'Cấm rẽ phải',
            8:'Cấm xe quay đầu',
            9:'Tốc độ tối đa cho phép (40 km/h)',
            10:'Tốc độ tối đa cho phép (50 km/h)',
            11:'Tốc độ tối đa cho phép (60 km/h)'}

```

Bước 3. Khởi tạo GUI

```

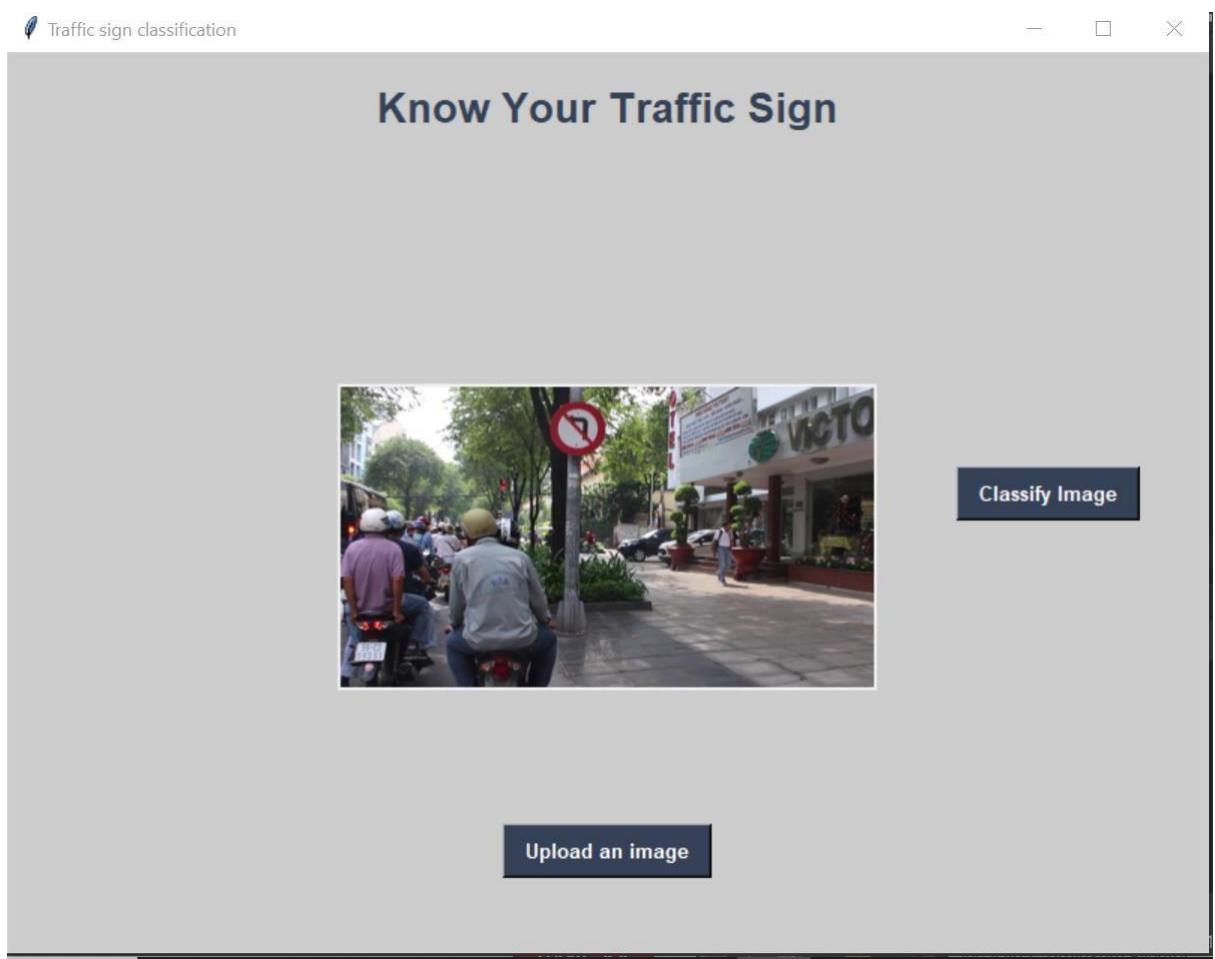
# initialise GUI
top = tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#CDCDCD')

label = Label(top, background='#CDCDCD', font=('arial', 15, 'bold'))
sign_image = Label(top)

def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30, 30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)

```

Kết quả giao diện đạt được



Hình 20. Hình ảnh giao diện

PHẦN IV. NHẬN DIỆN ĐÈN GIAO THÔNG

1. Tổng quát

Trong phần nhận diện đèn giao thông em đã sử dụng YoloV5 để thực hiện nhận diện và cho training cho Yolo trên Google Colab.

1.1. Tổng quan về Yolo

Khái niệm

Yolo – You Only Look One: là một mô hình nhận diện vật thể (object detection) trong thời gian thực (real-time). Vì không sử dụng region proposal nên kỹ thuật này có độ chính xác thấp hơn mô hình RCNN nhưng thời gian xử lý lại rất nhanh. YOLO hoạt động ở tốc độ 45 fps và tối đa 155 fps trong khi máy quay phim thông thường chỉ khoảng 24 fps.

Phương thức hoạt động

Bước 1: Phân chia tấm ảnh đầu vào thành một lưới $G \times G$.

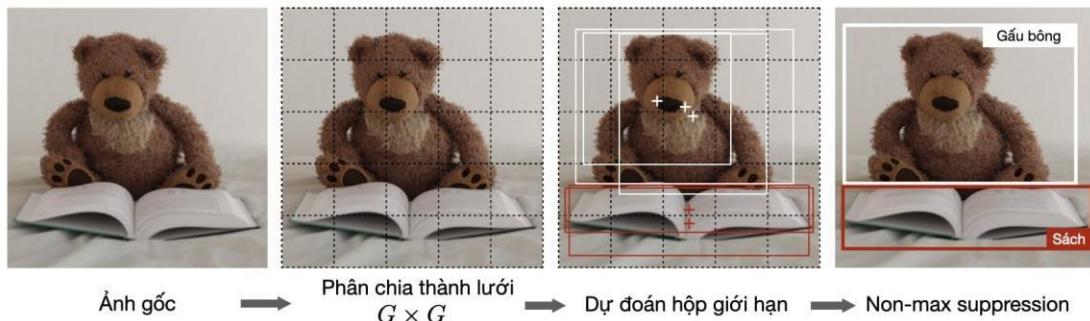
Bước 2: Với mỗi lưới, chạy một mạng CNN dự đoán y có dạng sau:

$$y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_p, \dots]^T \in R^{G \times G \times k \times (5+p)}$$

y lặp lại k lần

với p_c là xác suất dự đoán được một vật thể, b_x, b_y, b_h, b_w là những thuộc tính của hộp giới hạn được dự đoán, c_1, \dots, c_p là biểu diễn one-hot của việc lớp nào trong p các lớp được dự đoán, và k là số lượng các anchor box.

Bước 3: Chạy thuật toán non-max suppression để loại bỏ bất kỳ hộp giới hạn có khả năng bị trùng lặp.

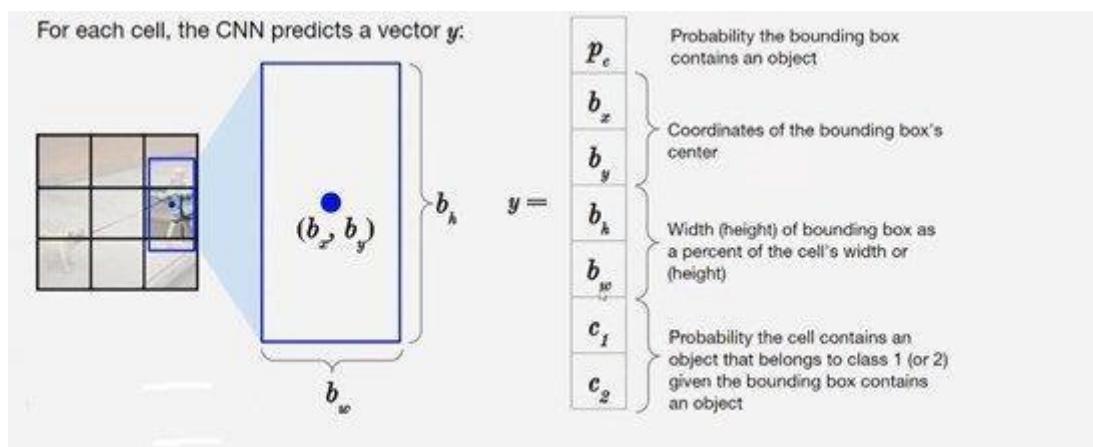


Hình 21. Các bước hoạt động của Yolo

Lưu ý: khi $pc=0$, thì mạng không phát hiện bất kỳ vật thể nào. Trong trường hợp đó, Các dự đoán liên quan $bx...cp$ sẽ bị lờ đi.

Cách mã hóa 1 Bounding box

Một Bounding box sẽ có các thông số được mô tả như trong Hình



Hình 22. Thông số của 1 Bounding box

p_c : xác suất bounding box chứa đối tượng

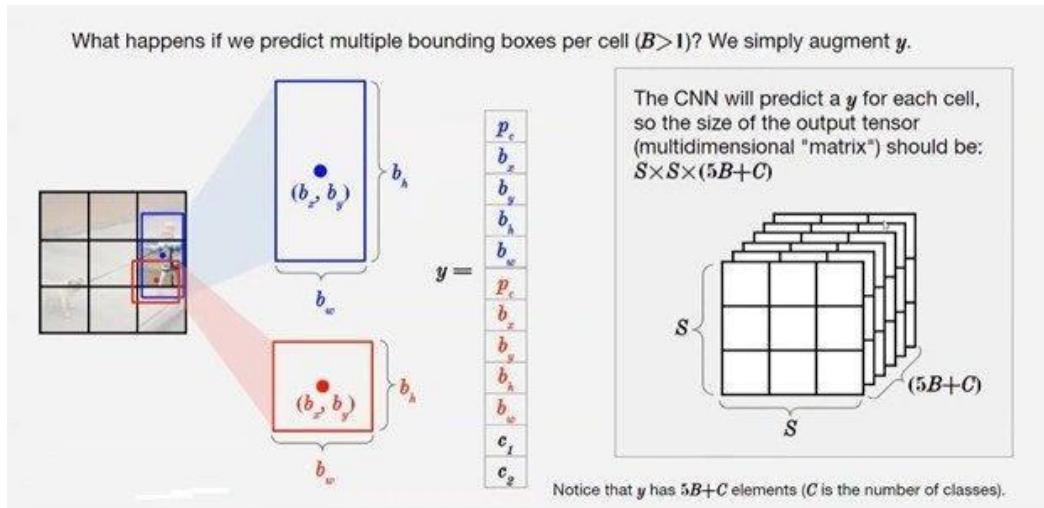
b_x, b_y : tọa độ của tâm bounding box

b_h, b_w : chiều rộng (chiều cao) của bounding box dưới dạng phần trăm chiều rộng (chiều cao) của ô vuông

c_1, c_2 : xác suất ô chứa đối tượng thuộc lớp 1 (hoặc 2) cho bounding box chứa một đối tượng

Cách mã hóa nhiều Bounding box

Để mã hóa nhiều Bounding box chúng ta sẽ tăng kích thước của y lên như trong Hình



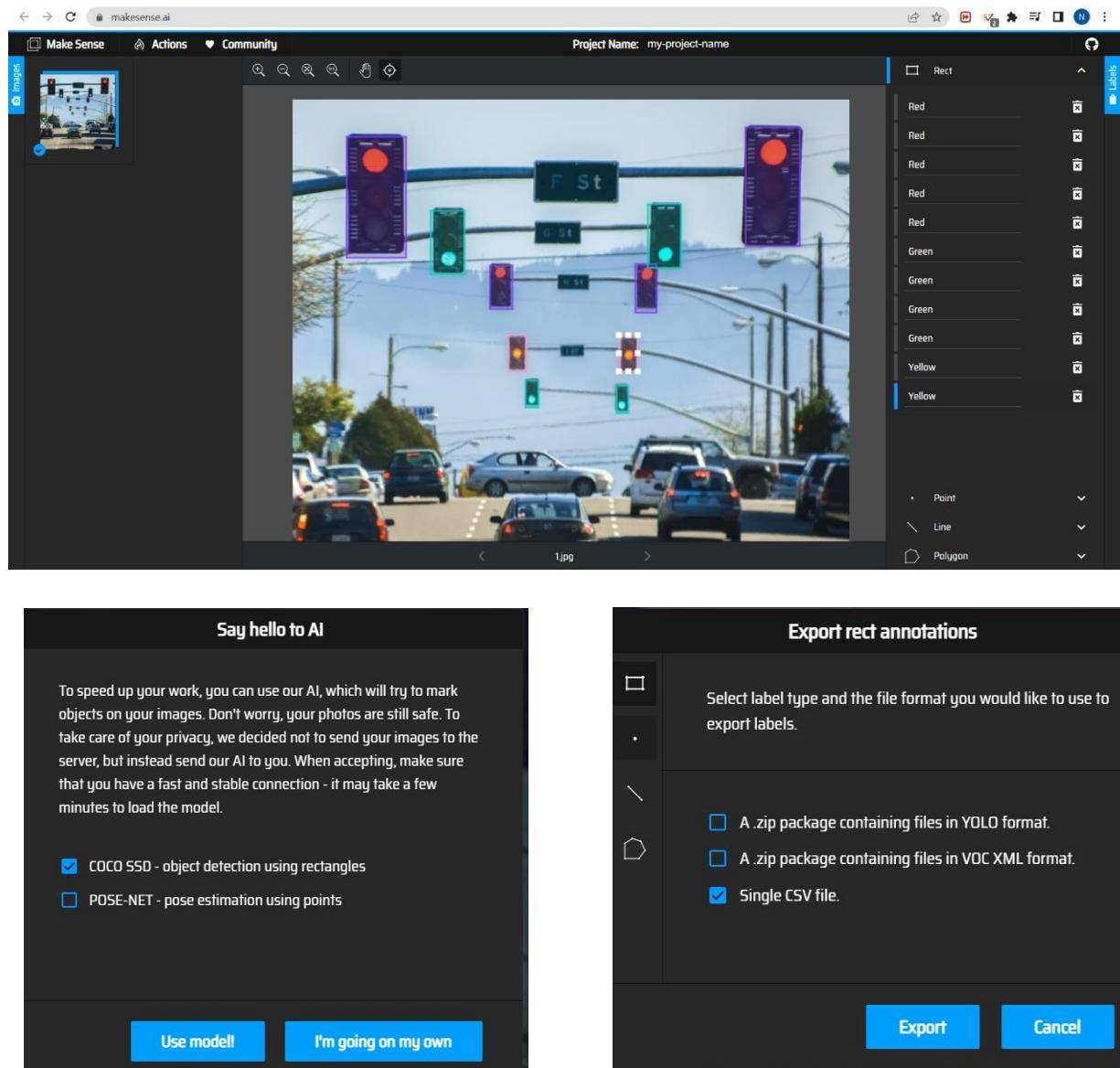
Hình 23. Mã hóa nhiều Bounding Box

Ở ví dụ trong **Hình** ta thấy được 1 cell ở đây dự đoán 2 bounding box (1 xanh và 1 đỏ). Ta đã tăng kích thước của y lên bằng cách thêm các thông số p_c, b_x, b_y, b_h, b_w tương tự như việc dự đoán 1 bounding box và giữ nguyên C1 và C2. Do C1 và C2 ở đây là xác suất ô chứa đối tượng không thuộc class nào nên nó sẽ không bị phụ thuộc vào 5 thông số trên nên ở đây ta vẫn giữ nguyên C1 và C2. Sau đó CNN sẽ làm nhiệm vụ dự đoán và đưa ra một đầu ra y là một ma trận đa chiều có kích thước “ $S \times S \times (5B + C)$ ” trong đó với B là số bounding box cần xác định và C ở đây là số class.

2. Các bước thực hiện training Yolov5 trên Google Colab

2.1. Chuẩn bị Dataset

Để phát hiện đối tượng nào đó, bước đầu tiên chúng ta cần thu thập thật nhiều ảnh của chúng, sau đó gán nhãn cho chúng rồi chia thành các tập train/val hoặc train/val/test. Ở đây chúng ta sử dụng web makesenes.ai để làm Tool gán nhãn (phổ biến nhất chắc là LabelImg)



Hình 24. Các thao tác trên makesense.ai

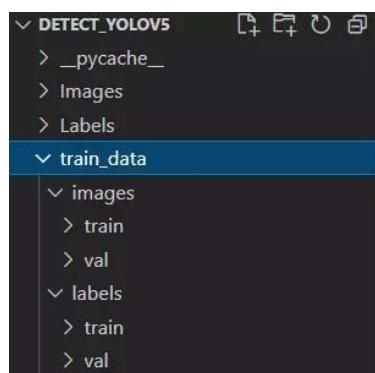
Sau khi đã tải tệp data đã thu thập được lên makesense.ai, ta có thể sử dụng model COCO SSD có sẵn để để tự nhận diện vật thể và gán nhãn ,tuy nhiên do là model có sẵn nên độ tương thích với tệp dữ liệu ảnh chúng ta tải lên là không cao ,đôi khi bỏ sót vật thể mới ,vì vậy sau khi để model nhận diện xong, chúng ta nên kiểm tra và sửa lại cho phù hợp.Sau đó chọn xuất file CSV và tải xuống .

Ở đây chúng ta sử dụng bộ dữ liệu Trafficlight gồm 8 đối tượng là các loại màu sắc, trạng thái đèn khác nhau :

- Class 0: Redleft
- Class 1: Redright
- Class 2: Red
- Class 3: Green
- Class 4: Greenleft
- Class 5: Greenright
- Class 6: Yellow
- Class 7: Off

Dataset khi tải về sẽ có 2 thư mục lớn là **Images** và **Labels**. Thư mục **Images** sẽ chứa tất cả các ảnh của 8 loại đèn kẽ trên. Còn thư mục **Labels** sẽ chứa thư mục con chứa nhãn của các ảnh đó dưới 4 dạng file: csv, tf record, txt và xml. Và model yolov5 nhận dữ liệu label để train dưới dạng file txt.

Chúng ta cần chia dataset ra thành 2 tập train/val phù hợp với model YOLOv5 như sau:



Hình 25. File dataset sau khi được chia thành các tập train/val

2.2. Quá trình training model trên Google Colab

Bước 1. Clone github Yolov5, check và cài đặt các môi trường thư viện cần thiết được mô tả như trong Hình 25

Setup

Clone repo, install dependencies and check PyTorch and GPU.

```
[ ] !git clone https://github.com/ultralytics/yolov5 # clone
%cd yolov5
%pip install -qr requirements.txt # install

import torch
import utils
display = utils.notebook_init() # checks

YOLOv5 🚀 v6.1-295-gf8722b4 Python-3.7.13 torch-1.11.0+cu102 CUDA:0 (Tesla T4, 15110MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 41.2/78.2 GB disk)
```

Hình 26. Clone Yolov5 & cài đặt môi trường, thư viện

Bước 2. Upload file Data và giải nén đã chuẩn bị trước lên Google Collab được mô tả trong Hình 26 và Hình 27

The screenshot shows a Google Colab notebook titled "YOLOv5 TrafficLightDetect". On the left, there's a sidebar with "Files" showing a folder named "yolov5" and a file named "train_data1.zip". The main area has a code editor with the following content:

```
+ Code + Text
[1]
import torch
import utils
display = utils.notebook_init() # checks

YOLOv5 🚀 v6.1-304-g51fb467 Python-3.7.13 torch-1.11.0+cu102 CUDA:0 (Tesla T4, 15110MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 41.2/78.2 GB disk)

!unzip -q ..train_data1.zip
unzip: cannot find or open ../runs.zip, ../runs.zip.zip or ../runs.zip.ZIP.
```

Below the code editor, there's a section titled "1. Inference" with some explanatory text and command examples:

```
detect.py runs YOLOv5 inference on a variety of sources, downloading models automatically from the latest YOLOv5 release, and saving results to runs/detect. Example inference sources are:
```

```
python detect.py --source 0 # webcam
img.jpg # image
vid.mp4 # video
path/ # directory
path/*.jpg # glob
'https://youtu.be/Zg19g1ksQHc' # YouTube
'rtsp://example.com/media.mp4' # RTSP, RTMP, HTTP stream
```

At the bottom right, it says "1m 53s completed at 8:05 PM".

Hình 27. Upload file Data

A terminal window showing the command `!unzip -q ..train_data1.zip` being run. The output message "unzip: cannot find or open ../runs.zip, ../runs.zip.zip or ../runs.zip.ZIP." is displayed.

Hình 28. Giải nén file Data

Bước 3. Thiết lập file Data yaml cho việc Training

Để đào tạo mô hình YOLO-V5, chúng ta cần có phải có tệp yaml. Tệp yaml có tác dụng: cho biết file dữ liệu của chúng ta nằm ở đâu, số lượng các lớp mà chúng ta muốn phát hiện và tên tương ứng với các lớp đó.

Tệp yaml thông thường sẽ có định dạng như trong Hình 28

```
train: ./train/images
val: ./valid/images

nc: 7
names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark',
'starfish', 'stingray']
```

Hình 29. Tệp yaml

Sau đó chúng ta sẽ tạo 1 file yaml riêng cho tập dữ liệu đèn giao thông như trong Hình 29. Khi đã có xong thư mục Yolov5 đã thực hiện ở Bước 1.



```
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files + Code + Text
RAM Disk Edit
(x) .. yolov5
Setup
Clone repo, install dependencies and check PyTorch and GPU.

!git clone https://github.com/ultralytics/yolov5 # clone
!cd yolov5
!pip install -qr requirements.txt # install

import torch
import utils
display = utils.notebook_init() # checks

YOLOv5 v6.1-204-g51fb467 Python-3.7.13 torch-1.11.0+cu102 CUDA:0 (Tesla T4, 15110MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 41.2/78.2 GB disk)
```

Hình 30. File yaml cho Data đèn giao thông

Mở thư mục Yolov5, truy cập đến thư mục con data và giải file coco128.yaml xuống như trong Hình 30

```

File Edit View Insert Runtime Tools Help All changes saved
Comment Share RAM Disk Editin
Files + Code + Text
Setup
Clone repo, install dependencies and check PyTorch and GPU.

git clone https://github.com/ultralytics/yolov5 # clone
cd yolov5
pip install -qr requirements.txt # install

import torch
import utils
display = utils.notebook_init() # checks

YOLOv5 v6.1-304-g51fb467 Python-3.7.13 torch-1.11.0+cu102 CUDA:0 (Tesla T4, 15110MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 41.2/78.2 GB disk)

```

Hình 31. Tải file coco128.yaml

Trong file coco128.yaml sẽ có nội dung như trong Hình 31

```

coco128.yaml - Notepad
File Edit Format View Help
# YOLOv5 by Ultralytics, GPL-3.0 license
# COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics
# Example usage: python train.py --data coco128.yaml
# parent
#   datasets
#     coco128 <-- downloads here (7 MB)

# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ...]
path: ../datasets/coco128 # dataset root dir
train: images/train2017 # train images (relative to 'path') 128 images
val: images/val2017 # val images (relative to 'path') 128 images
test: # test images (optional)

# Classes
nc: 80 # number of classes
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',
        'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
        'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
        'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',
        'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
        'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
        'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
        'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',
        'hair drier', 'toothbrush'] # class names

# Download script/URL (optional)
download: https://ultralytics.com/assets/coco128.zip

```

Hình 32. Nội dung file coco128.yaml

Chúng ta sẽ sửa lại nội dung trong file như trong Hình 32 và lưu file dưới tên mới trafficLightData1.yaml

```

trafficLightData1.yaml - Notepad
File Edit Format View Help
train: ../train_data1/images/train/ # train images (relative to 'path') 128 images
val: ../train_data1/images/val/ # val images (relative to 'path') 128 images

# Classes
nc: 8 # number of classes
names: ['RedLeft', 'Red', 'RedRight', 'GreenLeft', 'Green', 'GreenRight', 'Yellow', 'Off']

```

Hình 33. File trafficLightData1.yaml

Sau khi đã có file yaml mới ta sẽ upload file đó vào thư mục con Data nằm trong thư mục Yolov5 như trong Hình 33

```

File Edit View Insert Runtime Tools Help All changes saved
Files + Code + Text
{yolov5} [1]
import torch
import utils
display = utils.notebook_init() # checks
YOLOv5 v6.1-304-g51fb467 Python-3.7.13 torch-1.11.0+cu102 CUDA:0 (Tesla T4, 15110MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 41.2/78.2 GB disk)

!unzip ..train_data1.zip
unzip: cannot find or open .../runs.zip, .../runs.zip.zip or .../runs.zip.ZIP.

1. Inference
detect.py runs YOLOv5 inference on a variety of sources, downloading models automatically from the latest YOLOv5
results to runs/detect. Example inference sources are:
python detect.py --source 0 # webcam
img.jpg # image
vid.mp4 # video
path/ # directory
path/*.jpg # glob
'https://youtu.be/2gi9g1ksQHc' # YouTube

```

Hình 34. Upload file yaml vào thư mục Yolov5

Bước 4. Training

Chúng ta sẽ training thông qua file train.py nằm trong thư mục Yolov5 như trong Hình 34

from	n	params	module	arguments
0	-1	3520	models.common.Conv	[3, 32, 6, 2, 2]
1	-1	18560	models.common.Conv	[32, 64, 3, 2]
2	-1	18816	models.common.C3	[64, 64, 1]
3	-1	73984	models.common.Conv	[64, 128, 3, 2]
4	-1	115712	models.common.C3	[128, 128, 2]
5	-1	295424	models.common.Conv	[128, 256, 3, 2]
6	-1	625152	models.common.C3	[256, 256, 3]
7	-1	1180672	models.common.Conv	[256, 512, 3, 2]
8	-1	1182720	models.common.C3	[512, 512, 1]
9	-1	656896	models.common.SPPF	[512, 512, 5]
10	-1	131584	models.common.Conv	[512, 256, 1, 1]
11	-1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	models.common.Concat	[1]

Hình 35. File train.py

Trong đó:

- Img là kích thước của ảnh chúng ta truyền vào.

- Batch size: là số lượng mẫu trong trong 1 lần huấn luyện, ở đây batch size = 15 có nghĩa là 1 lần huấn luyện chúng ta sẽ lấy ngẫu nhiên 15 bức hình chạy lan truyền tiến trong mạng Yolov5. Tiếp theo lấy tiếp 15 hình ngẫu nhiên, không lặp với các hình trước đó, vào mạng, lấy đến khi nào không còn hình nào có thể lấy vào nữa -> hoàn thành 1 epoch.
 - Epoch: Một Epoch được tính là khi chúng ta đưa tất cả dữ liệu trong tập train vào mạng Yolov5 1 lần.

Kết quả training được thể hiện trong Hình 35

```
+ Code + Text Reconnect ▾ Editing ▾
178/499   8.27G 0.02326 0.01002 0.002436      13      1024: 100% 18/18 [00:10<00:00, 1.71it/s]
          Class Images Labels    P      R mAP@.5 mAP@.5:.95: 100% 2/2 [00:00<00:00, 2.65it/s]
          all    44     123    0.689    0.419    0.413    0.241

Epoch gpu_mem box obj cls labels img_size
179/499   8.27G 0.02346 0.01056 0.002791      18      1024: 100% 18/18 [00:10<00:00, 1.68it/s]
          Class Images Labels    P      R mAP@.5 mAP@.5:.95: 100% 2/2 [00:00<00:00, 2.77it/s]
          all    44     123    0.533    0.484    0.453    0.27

Epoch gpu_mem box obj cls labels img_size
180/499   8.27G 0.02324 0.01149 0.003059      9       1024: 100% 18/18 [00:10<00:00, 1.72it/s]
          Class Images Labels    P      R mAP@.5 mAP@.5:.95: 100% 2/2 [00:00<00:00, 2.68it/s]
          all    44     123    0.551    0.492    0.46     0.266

Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 80, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStop.

181 epochs completed in 0.591 hours.
Optimizer stripped from runs/train/exp3/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/exp3/weights/best.pt, 14.8MB

Validating runs/train/exp3/weights/best.pt...
Fusing layers...
Model summary: 213 layers, 7031701 parameters, 0 gradients, 15.8 GFLOPS
          Class Images Labels    P      R mAP@.5 mAP@.5:.95: 100% 2/2 [00:01<00:00, 1.35it/s]
          all    44     123    0.71    0.474    0.511    0.301
          RedLeft 44     12    0.698    0.833    0.882    0.556
          Red    44     30    0.669    0.943    0.872    0.453
          GreenLeft 44      4    0.636    0.454    0.414    0.256
          Green    44     60    0.691    0.856    0.861    0.476
          GreenRight 44      3    0.278    0.231    0.21     0.131
          Yellow   44     13    0.278    0.231    0.21     0.131
          Off     44      1    0.278    0.231    0.21     0.131

Results saved to runs/train/exp3
```

Hình 36. Kết quả training

Ta có thể thấy model không chạy hết 500 epoch vì do qua 100 epoch, độ chính xác của các epoch này là gần nhau, không có sự thay đổi rõ rệt lắm, nên model đã tự ngắt và dừng lại ở epoch thứ 180 với độ chính xác là 46%. Kết quả được lưu vào trong thư mục con train của thư mục run thuộc thư mục lớn Yolov5.

all : 51,1 %

RedLeft: 88.2%

Red: 87.2%

GreenLeft: 41.4%

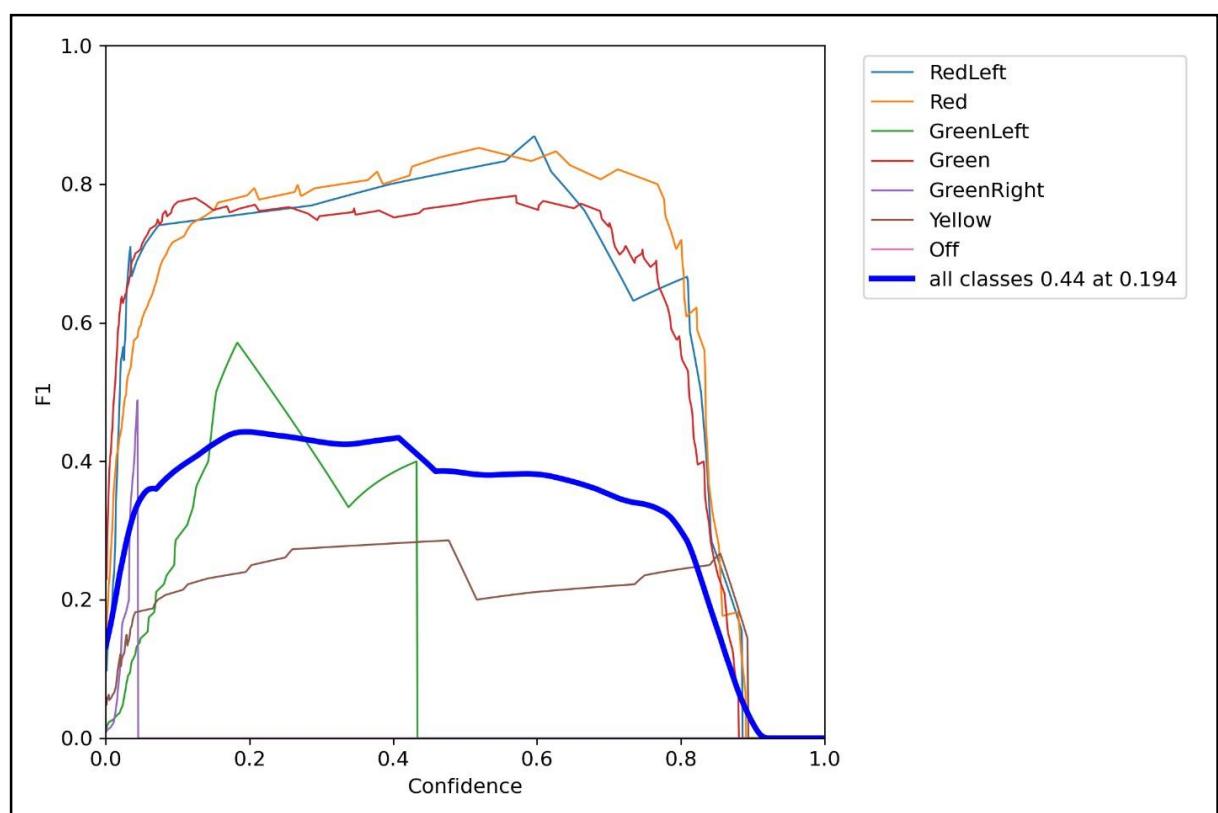
Green: 86.1%

GreenRight: 33.7%

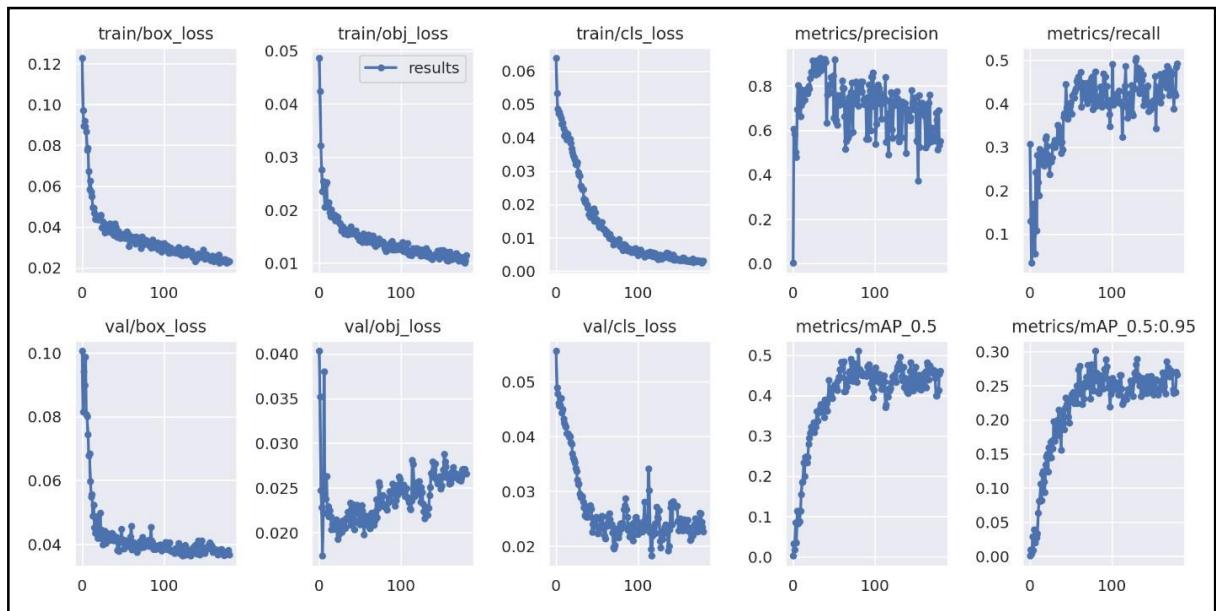
Yellow: 21%

Off: 0%

Dưới đây là biểu đồ chính xác

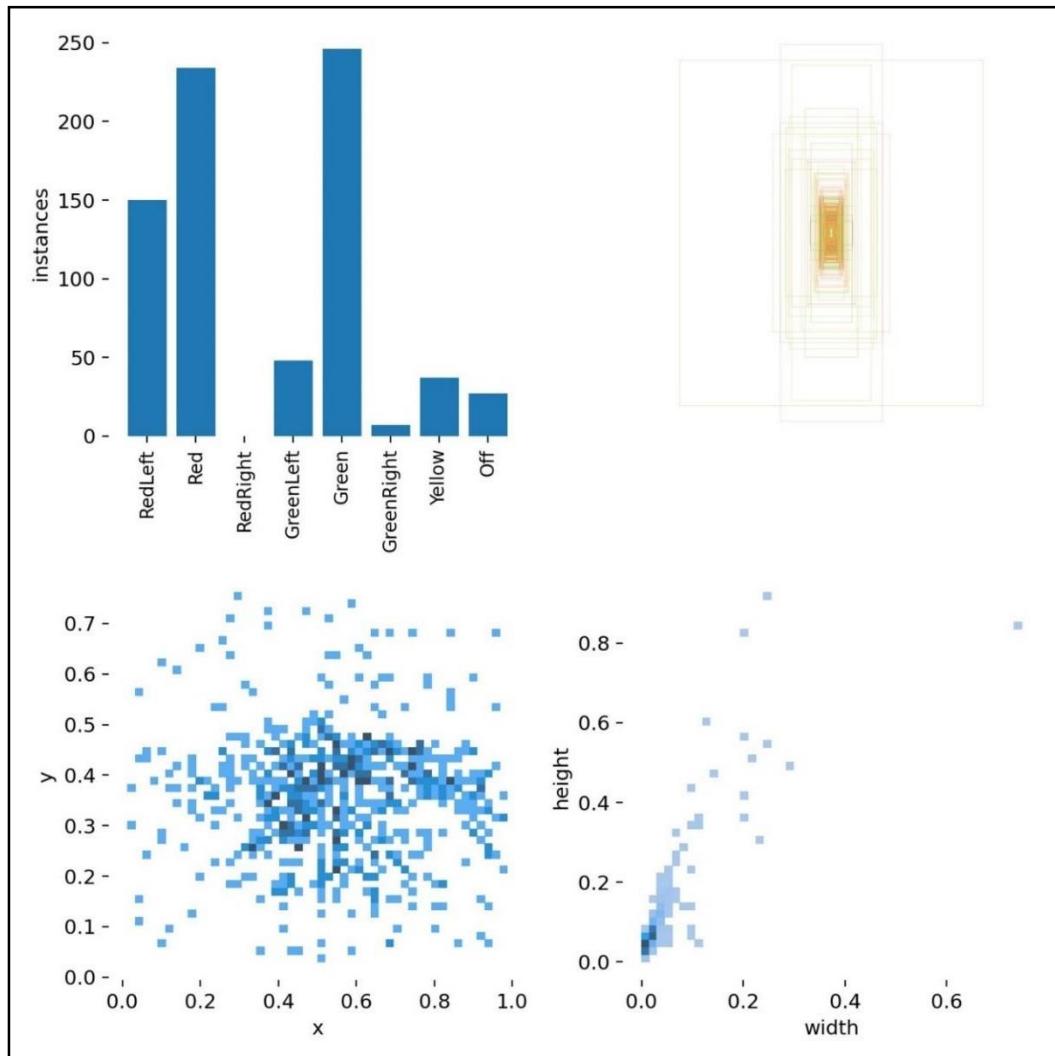


*Hình 37. Biểu đồ thể hiện độ chính xác của các nhãn
Biểu đồ kết quả của quá trình Training*



Hình 38. Biểu đồ kết quả quá trình Training

Biểu đồ dữ liệu, annotation box cho từng nhãn dữ liệu



Hình 39. Biểu đồ dữ liệu, annotation box cho từng nhãn dữ

Bước 5. Dự đoán

Chúng ta sẽ detect thông qua file detect.py thuộc thư mục Yolov5 như trong Hình 39

```
!python detect.py --weights runs/train/exp3/weights/best.pt --img 720 --conf 0.45 --source ../8.jpg
detect: weights=['runs/train/exp3/weights/best.pt'], source=../8.jpg, data=data/coco128.yaml, imgsz=[720, 720], conf_thres=0.45, iou_thres=0.45,
YOLOv5 v6.1-295-gf8722b4 Python-3.7.13 torch-1.11.0+cu102 CUDA:0 (Tesla T4, 15110MiB)
Fusing layers...
Model summary: 213 layers, 7031701 parameters, 0 gradients, 15.8 GFLOPS
WARNING: --img-size [720, 720] must be multiple of max stride 32, updating to [736, 736]
image 1/1 /content/8.jpg: 448x736 1 Red, Done. (0.013s)
Speed: 0.5ms pre-process, 12.6ms inference, 1.2ms NMS per image at shape (1, 3, 736, 736)
Results saved to runs/detect/exp6
```

Hình 40. Quá trình Predicting

Một số kết quả của quá trình dự đoán được thể hiện ở những hình dưới đây





Hình 41. Kết quả của quá trình Predicting

PHẦN V. KẾT LUẬN

Chương trình đã giải quyết được bài toán nhận dạng đèn giao thông và biển báo thông qua lý thuyết về xử lý ảnh, qua đó cũng giải quyết được yêu cầu đề tài đồ án đã đề ra. Về độ chính xác của chương trình bên cạnh việc xây dựng các Module đào tạo đúng đắn còn phụ thuộc vào các yếu tố bên ngoài như: ánh sáng môi trường, độ phân giải của camera sử dụng... Tuy nhiên các yếu tố này đều có giải pháp khắc phục. Do thời gian làm đề tài đồ án có giới hạn, ngoài ra việc tiếp cận với những kiến thức cơ bản về xử lý ảnh và kỹ năng về lập trình còn yêu cầu việc thực hiện còn nhiều thiếu sót và chưa đạt được kỳ vọng mà em đã đặt ra ngay từ ban đầu.

Để giải quyết các khó khăn của đề tài, em xin đề xuất 1 số ý kiến sau đây:

- Sử dụng camera chuyên dụng
- Thiết lập môi trường ổn định xung quanh vật thể để làm tăng độ chính xác của ảnh chụp
- Sử dụng máy chủ có hỗ trợ GPU để build chương trình thay vì Colaboratory, để tránh việc giới hạn thời gian sử dụng trong quá trình training.

Qua đề tài này một lần nữa em rất biết ơn thầy Nguyễn Tiến Hòa đã tận tình hướng dẫn và chỉ bảo trong suốt khoảng thời gian qua để có thể hoàn thành đề tài đồ án đã được giao. Em cũng đã có rất nhiều cố gắng để tiếp cận, tìm hiểu và thảo luận các nội dung nghiên cứu. Em cũng rất mong nhận được sự đóng góp ý kiến của thầy cô và các bạn, thời gian tới em vẫn tiếp tục học hỏi và phát triển tiếp sản phẩm để đi đến việc hoàn thiện một chương trình mang tính ứng dụng cao.

TÀI LIỆU THAM KHẢO

<https://github.com/ultralytics/yolov5>

<https://www.youtube.com/watch?v=hTCmL3S4Obw&t=1169s>

<https://github.com/initdebugs/Beginner-Traffic-Light-Detection-OpenCV-YOLOv3>

<https://github.com/AvivSham/German-Traffic-Signs-Classification>

- [1] K. He, X. Zhang, S. Ren, J. Sun. Identity mappings in deep residual networks, ECCV, 2016.
- [2] Nicolai Wojke, Alex Bewley. Deep Cosine Metric Learning for Person ReIdentification , 2016
- [3] Nicolai Wojke , Alex Bewley , Dietrich Paulus. Simple Online and Realtime Tracking with a deep association metric
- [4] Alex Bewley , Zongyuan Ge, Lionel Ott , Fabio Ramos , Ben Upcroft. Simple Online and Realtime Tracking ,2017
- [5] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems
- [6] Bui Tien Tung, SORT – DeepSORT: một góc nhìn về Object Tracking
- [7] S. Zagoruyko, N. Komodakis, “Wide residual networks,” BMVC, 2016
- [8] Jialian Wu , Jiale Cao, Liangchen Song , Yu Wang , Ming Yang , Junsong Yuan1, Track to Detect and Segment: An Online Multi-Object Tracker, CVPR 2021
- [9] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu Yichen Wei, Deformable Convolutional Networks, Microsoft Research Asia
- [10] Bui Tien Tung, TraDeS: từ object detection đến MOT.
- [11] Philipp Krahenbuhl, Dequan Wang, Xingyi Zhou. Objects as Points, 2019
- [12] (DORUK, 2020)
- [13] (How to Detect and Classify Traffic Lights, 2021)
- [14] (Shambhavi Lalsinge, 2022)
- [15] (Traffic Sign Classification, n.d.)
- [16] (Python Project on Traffic Signs Recognition with 95% Accuracy using CNN & Keras, n.d.)