

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA**



ĐỒ ÁN

**XÂY DỰNG MÔ HÌNH MÁY HỌC ỨNG DỤNG TRONG
NHẬN DIỆN VÀ ĐO TỐC ĐỘ NHÓM XE CƠ GIỚI Ô TÔ**

**BUILDING A MACHINE LEARNING MODEL FOR THE
RECOGNITION AND SPEED MEASUREMENT OF
AUTOMOBILE GROUPS**

Thành phố Hồ Chí Minh, tháng 8 năm 2023

TÓM TẮT ĐỒ ÁN

Hệ thống kiểm soát tốc độ được sử dụng ở hầu hết các quốc gia để thực thi giới hạn tốc độ và để ngăn ngừa tai nạn. Hệ thống được đề xuất phát hiện các phương tiện đang di chuyển bằng cách sử dụng bộ phát hiện chuyển động được tối ưu hóa. Tốc độ phương tiện được ước tính bằng cách so sánh quỹ đạo của các tính năng được theo dõi với các phép đo đã biết trong thế giới thực.

Đồ án này trình bày về đề tài: “Xây dựng mô hình máy học ứng dụng trong nhận diện và đo tốc độ nhóm xe cơ giới ô tô”. Để thực hiện được hệ thống này, đồ án nghiên cứu và sử dụng thuật toán nhận diện vật thể state-of-art YOLO (You only look once) version 8. Đầu vào là những video thu thập ngoài đời thực và thực hiện chỉnh sửa cho thuật toán huấn luyện mô hình. Rồi cho ra kết quả về nhận diện xe và tốc độ tương ứng.

Báo cáo đồ án bao gồm **04** chương với nội dung chính của từng chương như sau:

CHƯƠNG 1 – “GIỚI THIỆU”: Giới thiệu về tổng quan, tính cấp thiết của đề tài, đối tượng và phạm vi nghiên cứu cũng như mục tiêu của đề tài.

CHƯƠNG 2 – “CƠ SỞ LÝ THUYẾT”: Cơ sở lý thuyết liên quan về học máy, mạng máy tính, YOLOv8, ... Trình bày công thức xây dựng bài toán nhận diện và đo tốc độ xe.

CHƯƠNG 3 – “KẾT QUẢ VÀ PHÂN TÍCH”: Trình bày quá trình thực nghiệm bao gồm các thuật toán giải thuật, quá trình nghiên cứu xây dựng mô hình và thực hiện mô hình trên các tập dữ liệu. Nhận định kết quả đầu ra, so sánh kết quả qua các lần thực nghiệm, hình ảnh kiểm chứng.

CHƯƠNG 4 – “KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN”: Trình bày đánh giá và định hướng phát triển cho đề tài.

MỤC LỤC

TÓM TẮT ĐỒ ÁN.....	i
DANH SÁCH BẢNG	ivv
DANH SÁCH HÌNH VẼ	v
DANH SÁCH TỪ VIẾT TẮT	vi
CHƯƠNG 1. GIỚI THIỆU	1
1.1 Đặt vấn đề	1
1.2 Câu hỏi và mục tiêu nghiên cứu	1
1.3 Nội dung và phạm vi nghiên cứu	2
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....	4
2.1 Giới thiệu các cơ sở lý thuyết	4
2.1.1 Học máy (Machine Learning - ML).....	4
2.1.2 YOLOv8	9
2.2 Công thức đo tốc độ	24
2.3 Đo lưu lượng xe ra vào	26
2.4 Kết luận chương 2	27
CHƯƠNG 3. KẾT QUẢ VÀ PHÂN TÍCH.....	28
3.1 Nền tảng thực hiện	28
3.2 Phương pháp thiết lập	29
3.3 Giải thuật thực hiện	30
3.4 Kết quả và phân tích.....	32
3.4.1 Kết quả mô phỏng trên video thực nghiệm thực tế ghi lại từ camera đường bộ	32
3.4.2 Kết quả mô phỏng trên video thực nghiệm thực tế ghi lại từ camera điện thoại cá nhân	34
3.5 Đánh giá chất lượng model	38
3.6 Kết luận chương 3	39
CHƯƠNG 4. KẾT LUẬN.....	41
4.1 Tóm tắt và kết luận chung	41
4.2 Hướng phát triển.....	41
PHỤ LỤC 44	
A.1 Code chương trình giao tiếp colab	44
A.2 Code chương trình xử lý dữ liệu.....	45

TÀI LIỆU THAM KHẢO.....	54
-------------------------	----

DANH SÁCH BẢNG

Bảng 2-1: Các mẫu có sẵn trong yolov8	10
Bảng 2-2: Các thông số của các mẫu trong YOLOv8.....	11

DANH SÁCH HÌNH VẼ

Hình 2-1: Minh họa cấu trúc của một mạng nơron [1].	5
Hình 2-2: Cấu trúc của một mạng nơron tích chập [1].	6
Hình 2-3: Mô hình Many to Many của RNN [2].	8
Hình 2-4: Các dạng mô hình khác của RNN [2].	8
Hình 2-5: Kiến trúc GAN [3].	9
Hình 2-6: Đầu ra bằng cách sử dụng các mô hình phân đoạn phiên bản và phát hiện YOLOv8x [4].	13
Hình 2-7: Quá trình phát triển của YOLOv8 [4].	14
Hình 2-8: YOLOv8 so với các mô hình YOLO khác [4].	17
Hình 2-9: So sánh tổng thể Mô hình YOLOv8 so với mô hình YOLOv5 [4].	19
Hình 2-10: So sánh phát hiện đối tượng YOLOv8 & YOLOv5 [4].	20
Hình 2-11: So sánh phân đoạn phiên bản YOLOv8 & YOLOv5 [4].	21
Hình 2-12: So sánh phân loại hình ảnh YOLOv8 & YOLOv5 [4].	23
Hình 2-13: Minh họa công thức khoảng cách Euclidean giữa hai điểm.	24
Hình 3-1: Các tính năng tốt nhất của Google Colab [5].	29
Hình 3-2: Thiết lập thử nghiệm.	30
Hình 3-3: Một cảnh phát hiện chứa một số đường xâm nhập ($Lm + 1$, trong đó $m \in \{0, \dots, M - 1\}$) và một chiếc xe đi qua được quan sát bởi một chiếc máy ảnh đối diện với đường.	31
Hình 3-4: Phát hiện chuyển động: các khung hình video đầu vào được thu gọn thành các vùng quan tâm giới hạn bởi các phương tiện đang di chuyển.	32
Hình 3-5: Mô phỏng trên video thực nghiệm thực tế ghi lại từ camera đường bộ.	33
Hình 3-6: Xác định đếm xe mỗi khung hình trong video thực nghiệm từ camera đường bộ.	34
Hình 3-7: Xác định đếm xe mỗi khung hình ở video từ camera cá nhân.	35
Hình 3-8: Mô phỏng trên video thực nghiệm thực tế ghi lại từ camera cá nhân.	36
Hình 3-9: Mô phỏng trên video thực nghiệm thực tế ghi lại từ camera cá nhân.	37
Hình 3-10: Các thông số đánh giá của mô hình.	38

DANH SÁCH TỪ VIẾT TẮT

CNN	Convolution Neural Networks
GAN	Generative Adversarial Networks
IoT	Internet of Things
mAP	Mean Average Precision
ML	Machine learning
NN	Neural Networks
ppm	Pixels per meter
RNN	Recurrent Neural Networks
YOLO	You only look once
YOLOv8	You only look once version 8

CHƯƠNG 1. GIỚI THIỆU

1.1 Đặt vấn đề

Trong bối cảnh giao thông ngày càng phát triển và tốc độ giao thông không ngừng tăng, việc nhận diện và đo tốc độ các nhóm xe cơ giới ô tô trở nên cấp thiết. Đo tốc độ xe là một trong những thành phần chính của hệ thống giám sát giao thông. Dữ liệu cung cấp có thể được sử dụng cho quản lý giao thông và thực thi pháp luật. Việc theo dõi và kiểm soát tốc độ giao thông không chỉ giúp giảm thiểu tai nạn mà còn hỗ trợ việc giám sát và quản lý giao thông tốt hơn. Các hệ thống đo tốc độ có thể được chia thành hai danh mục chính dựa trên phương pháp của chúng, hoạt động theo cách chủ động hoặc bị động. Hệ thống chủ động đo cách các tín hiệu truyền được ảnh hưởng bởi một chiếc xe đang di chuyển để ước lượng tốc độ xe, ví dụ như hệ thống cảm biến dựa trên vòng dây cảm ứng, radar hoặc laser. Tuy nhiên, một số hệ thống này có giá đắt đỏ và đòi hỏi việc hiệu chỉnh hoặc bảo dưỡng rộng rãi dẫn đến hạn chế về chi phí, độ chính xác, và khả năng thích ứng với các điều kiện khác nhau.

Trong đề tài “Xây dựng mô hình máy học ứng dụng trong nhận diện và đo tốc độ nhóm xe cơ giới ô tô”, đồ án tập trung vào các hệ thống video quang học bị động, nghiên cứu vị trí, số lượng và tốc độ và tìm ra phương pháp phát hiện và nhận dạng đối tượng trong ảnh và video tối ưu hiện nay trong việc theo dõi và kiểm soát tốc độ giao thông. Các hệ thống dựa trên video trở nên phổ biến hơn do sự tiến bộ về công nghệ trong máy ảnh và thiết bị máy tính, đã trở nên hiệu quả hơn, sẵn có và đáng tin cậy. Ngoài ra, tính hiệu quả về chi phí hợp lý của các hệ thống này làm cho chúng trở nên cạnh tranh mạnh mẽ với các phương pháp truyền thống.

Các hệ thống đo tốc độ dựa trên video phân tích các khung video liên tiếp để theo dõi xe và do đó đo tốc độ của nó. Các phương pháp xử lý ảnh và nhận dạng mẫu khác nhau đã được sử dụng để thực hiện nhiệm vụ này. Các giả định chính là tốc độ của xe trong quá trình phát hiện là không đổi và máy ảnh đang hướng xuống cảnh. Một kỹ thuật thường được sử dụng là phân đoạn nền/đối tượng, phân tách xe đang di chuyển như một khối để ước lượng tốc độ của nó trong các khung liên tiếp.

1.2 Câu hỏi và mục tiêu nghiên cứu

Câu Hỏi Nghiên Cứu:

- Làm thế nào để xây dựng mô hình máy học có khả năng nhận diện và đo tốc độ chính xác?
- Sử dụng gì để đạt kết quả tối ưu?
- Mô hình này có thể được áp dụng trong các điều kiện giao thông khác nhau như thế nào?

Đồ án này tập trung vào việc khám phá và phát triển mô hình máy học, với mục tiêu áp dụng trong việc nhận diện và đo lường tốc độ của các nhóm xe cơ giới ô tô. Qua quá trình nghiên cứu, đề tài mong muốn đưa ra một phương pháp mới, chính xác, phù hợp với nhu cầu thực tế của hệ thống giao thông hiện đại, góp phần nâng cao hiệu quả trong việc kiểm soát và quản lý giao thông.

1.3 Nội dung và phạm vi nghiên cứu

Nội dung 1: Tìm hiểu lý thuyết về học máy và các nội dung trong đề tài:

- Nghiên cứu cơ sở lý thuyết về học máy, cụ thể là các phương pháp phân loại và học sâu.
- Phân tích và nắm vững yêu cầu và mục tiêu của đề tài, đồng thời xác định rõ các khía cạnh cần được giải quyết.

Nội dung 2: Tìm hiểu YOLOv8 và thử nghiệm để tìm ra cái tối ưu để áp dụng:

- Hiểu rõ về mô hình YOLOv8, cấu trúc, cách hoạt động, và ứng dụng của nó trong nhận diện đối tượng.
- Khảo sát và so sánh các thư viện và công nghệ có sẵn để xác định cái tốt nhất phù hợp với mô hình YOLOv8.
- Thực hiện thử nghiệm và tinh chỉnh các tham số để tối ưu hiệu suất của mô hình.
- Đánh giá và chọn lựa giải pháp tốt nhất để tiến xa hơn trong việc phát triển hệ thống.

Nội dung 3: Xây dựng sơ đồ hoạt động cho hệ thống:

- Thiết kế sơ đồ hoạt động tổng thể cho hệ thống, bao gồm các bước từ việc thu thập dữ liệu đến nhận diện và đo lường tốc độ.
- Xác định các module chính và mối liên hệ giữa chúng.
- Đảm bảo rằng sơ đồ hoạt động phản ánh đúng logic và yêu cầu của hệ thống.

Nội dung 4: Kiểm tra kết quả áp dụng trên video thực tế:

- Áp dụng mô hình đã được tối ưu hóa trên dữ liệu video thực tế để kiểm tra độ chính xác và hiệu suất trong việc nhận diện biển số và đo tốc độ.
- Phân tích kết quả để xác định các vấn đề còn tồn tại và cải tiến.
- Đảm bảo rằng mô hình hoạt động hiệu quả trong điều kiện thực tế và đáp ứng các yêu cầu và mục tiêu đã đề ra.

Nội dung 5: Nhận xét – đánh giá kết quả thực hiện và hướng phát triển

Các nhiệm vụ trên tạo nên quá trình hoàn chỉnh để phát triển và đánh giá mô hình máy học cho đề tài, từ việc tìm hiểu lý thuyết đến ứng dụng trong thực tế.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Giới thiệu các cơ sở lý thuyết

2.1.1 Học máy (*Machine Learning - ML*)

Cơ sở lý thuyết về học máy là một phần quan trọng của trí tuệ nhân tạo và khoa học dữ liệu. Dưới đây, sẽ trình bày một số khái niệm cơ bản về học máy, cụ thể là các phương pháp phân loại và học sâu.

✓ *Học Máy*

Machine learning (ML) hay máy học là một nhánh của trí tuệ nhân tạo (AI), nó là một lĩnh vực nghiên cứu cho phép máy tính có khả năng cải thiện chính bản thân chúng dựa trên dữ liệu mẫu (training data) hoặc dựa vào kinh nghiệm (những gì đã được học). Machine learning có thể tự dự đoán hoặc đưa ra quyết định mà không cần được lập trình cụ thể [1].

Bài toán machine learning thường được chia làm hai loại là dự đoán (prediction) và phân loại (classification). Các bài toán dự đoán như dự đoán giá nhà, giá xe... Các bài toán phân loại như nhận diện chữ viết tay, nhận diện đồ vật...

Có rất nhiều cách phân loại machine learning, thông thường thì machine learning sẽ được phân làm hai loại chính sau:

- Học Giám Sát (Supervised Learning)

Trong học giám sát, mô hình được huấn luyện trên dữ liệu đã được gán nhãn. Phương pháp phân loại và hồi quy là hai loại chính trong học giám sát.

- Học không giám sát (Unsupervised learning)

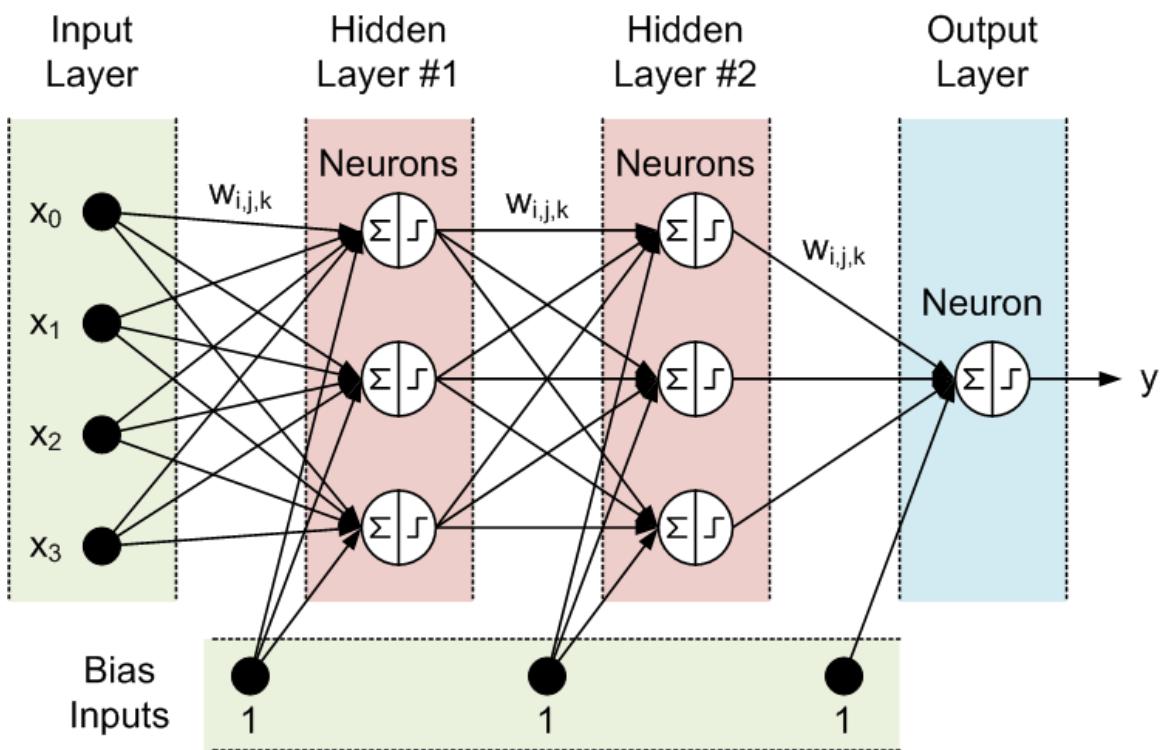
Unsupervised learning là cho máy tính học trên dữ liệu mà không được gán nhãn, các thuật toán machine learning sẽ tìm ra sự tương quan dữ liệu, mô hình hóa dữ liệu hay chính là làm cho máy tính có kiến thức, hiểu về dữ liệu, từ đó chúng có thể phân loại các dữ liệu về sau thành các nhóm, lớp (clustering) giống nhau mà chúng đã được học hoặc giảm số chiều dữ liệu (dimension reduction).

Ngoài ra, machine learning còn có thể phân làm các loại sau:

- Semi-supervised learning: học bán giám sát.
- Deep learning: học sâu (về một vấn đề nào đó).
- Reinforce learning: học củng cố/tăng cường.
- ✓ **Học Sâu (Deep Learning)**

Học sâu là một nhánh của học máy sử dụng các mạng nơ-ron nhân tạo có nhiều lớp (deep neural networks) để mô phỏng cách thức não người xử lý thông tin.

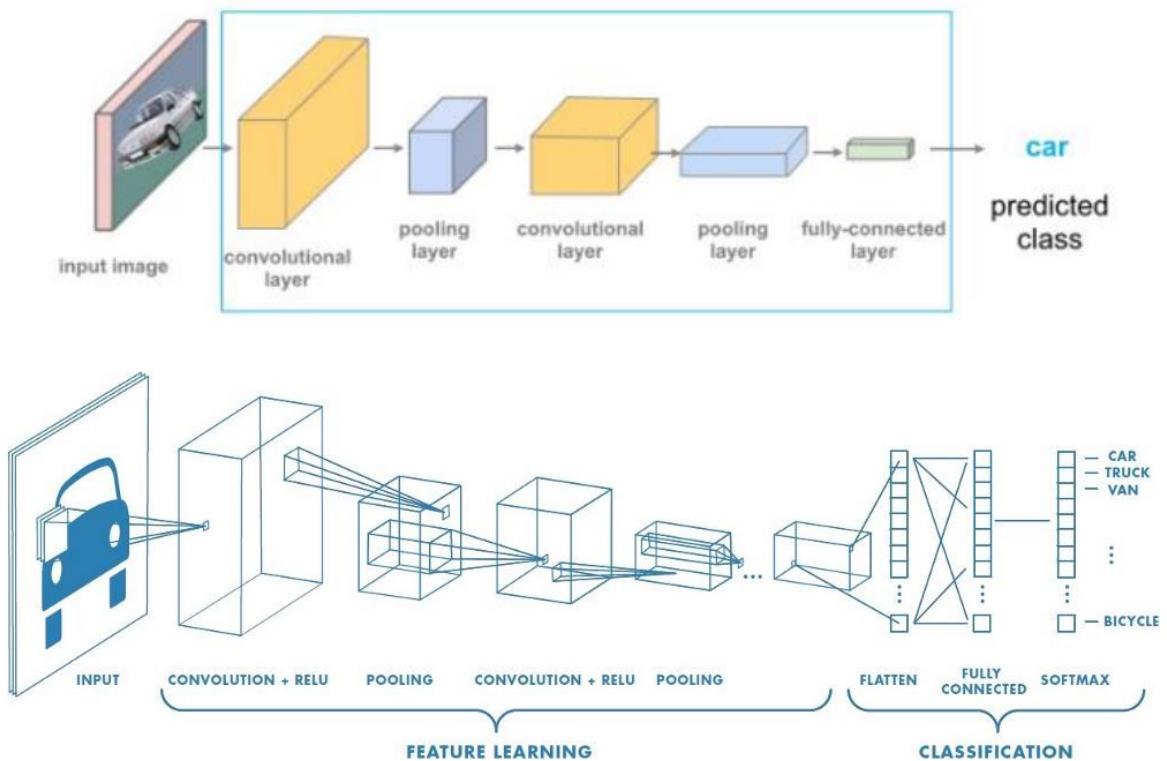
Mạng thần kinh (Neural Networks - NN) lấy ý tưởng xây dựng từ các neural sinh học trong bộ não. Mỗi phần tử trong mạng được gọi là một nút, hoặc một perceptron. Mạng thần kinh là sự kết hợp của nhiều layer chứa các perceptron và có 3 kiểu layer cơ bản: input layer, (đầu vào của mạng), hidden layer (thực hiện việc suy luận logic của mạng), output layer (đầu ra của mạng), chú thích trong hình 2-1.



Hình 2-1: Minh họa cấu trúc của một mạng nơron [1].

1. Mạng Nơ-ron Tích Chập (Convolutional Neural Networks - CNN)

Mạng nơron tích chập (Convolution Neural Networks - CNN) là một trong những mô hình nhận diện và phân loại vật thể. Cấu trúc mạng nơron tích chập bao gồm: input layer (đầu vào là các hình ảnh được chuyển hóa thành các mảng pixel), Convolution layer (lớp tích chập dùng để trích xuất các tính năng từ hình ảnh đầu vào), Pooling layer (lớp pooling được thực hiện sau mỗi lớp convolution nhằm giảm tải số lượng tham số khi hình quả quá lớn, tuy nhiên vẫn giữ được thông tin quan trọng), Fully Connected (lớp này nằm ở sau cùng và dùng để kết nối các đơn vị đại diện cho nhóm phân loại).



Hình 2-2: Cấu trúc của một mạng nơron tích chập [1].

Có 3 quá trình cơ bản trong một mạng nơron tích chập:

- Quá trình trích xuất đặc trưng: Ma trận đầu vào thực hiện tích chập với bộ lọc để tạo ra các đơn vị trong một lớp mới, quá trình này diễn ra liên tục ở phần đầu của mạng và thường sử dụng hàm Relu để kích hoạt.
- Quá trình tổng hợp: Các lớp ở sau quá trình trích xuất đặc trưng sẽ có kích thước lớn do số đơn vị ở các lớp sau sẽ tăng theo cấp số nhân. Dẫn đến việc làm tăng số lượng hệ số cũng như khối lượng tính toán cho mạng neural. Vì vậy chúng ta

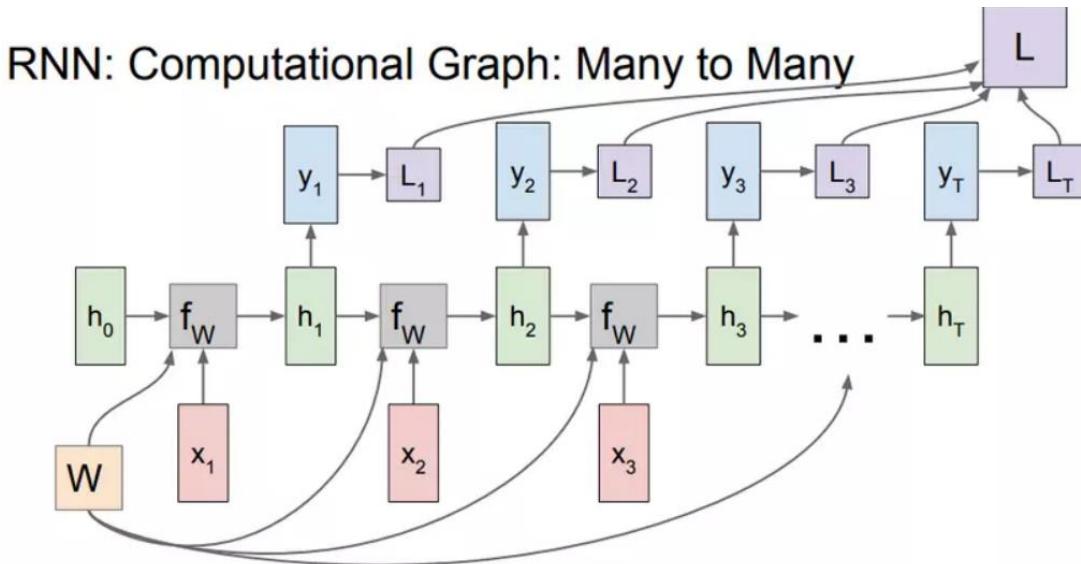
cần giảm tải số chiêu của ma trận (không thể giảm số đơn vị của lớp do mỗi đơn vị được tạo ra là kết quả của quá trình một bộ lọc áp dụng để tìm ra một đặc trưng cụ thể). Giảm kích thước ma trận thông qua việc tìm ra một giá trị đại diện cho vùng không gian mà bộ lọc đi qua, điều này không làm thay đổi các đường nét chính của bức ảnh. Quá trình này gọi là tổng hợp.

- Quá trình kết nối hoàn toàn: Sau khi giảm số lượng tham số đến một mức hợp lý, ma trận cần được làm dẹt (flatten thành một vector và sử dụng các kết nối hoàn toàn giữa các lớp, thường sử dụng hàm kích hoạt là Relu. Kết nối cuối cùng sẽ dẫn tới các đơn vị là đại diện cho mỗi lớp với hàm Softmax dùng để tính xác suất.

2. *Mạng Nơ-ron Hồi Quy (Recurrent Neural Networks - RNN)*

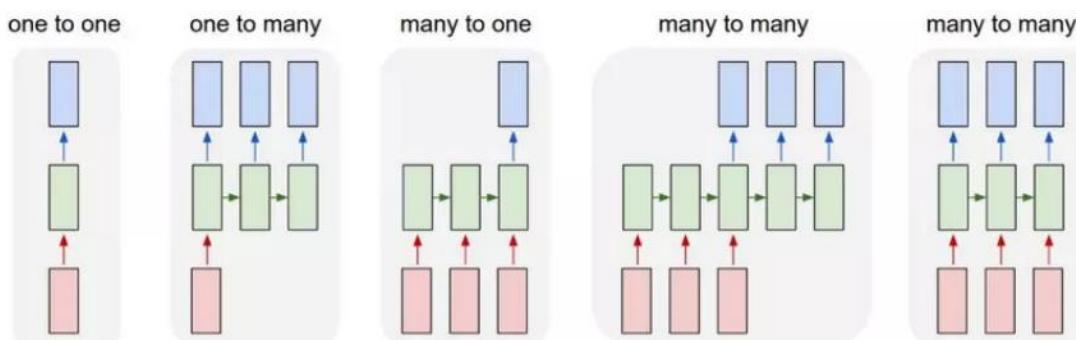
Như đã biết thì Neural Network bao gồm 3 phần chính là Input layer, Hidden layer và Output layer, ta có thể thấy là đầu vào và đầu ra của mạng neuron này là độc lập với nhau. Như vậy mô hình này không phù hợp với những bài toán dạng chuỗi như mô tả, hoàn thành câu, ... vì những dự đoán tiếp theo như từ tiếp theo phụ thuộc vào vị trí của nó trong câu và những từ đãng trước nó [2].

Và như vậy RNN ra đời với ý tưởng chính là sử dụng một bộ nhớ để lưu lại thông tin từ từ những bước tính toán xử lý trước để dựa vào nó có thể đưa ra dự đoán chính xác nhất cho bước dự đoán hiện tại. Nếu các bạn vẫn chưa hiểu gì thì hãy cùng xem mô hình mạng RNN sau và cùng phân tích để hiểu rõ hơn:



Hình 2-3: Mô hình Many to Many của RNN [2].

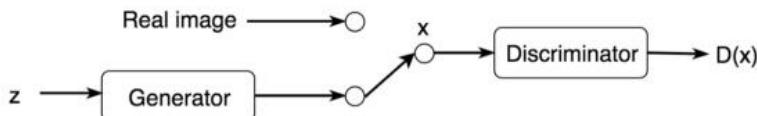
Ngoài mô hình Many to Many như ta thấy ở trên thì RNN còn rất nhiều dạng khác như sau:



Hình 2-4: Các dạng mô hình khác của RNN [2].

3. Mô Hình GAN (Generative Adversarial Networks)

GAN hướng tới việc sinh ra dữ liệu mới sau quá trình học. GAN có thể tự sinh ra một khuôn mặt mới, một con người, một đoạn văn, chữ viết, bản nhạc giao hưởng hay những thứ tương tự thế. Thế làm cách nào để GAN học và làm được điều đó, chúng ta cần phải điểm qua một vài khái niệm.

**Hình 2-5: Kiến trúc GAN [3].**

GAN được kết hợp từ 2 model: generator - G và discriminator - D. GAN giống như 1 trò chơi minimax, trò cảnh sát tội phạm: tội phạm G tạo ra tiền giả, cảnh sát D học cách phân biệt thật giả. Cảnh sát càng cố gắng phân biệt tiền thật-giả thì tội phạm lại dựa vào feedback của cảnh sát để cải thiện khả năng tạo tiền giả của mình, cố gắng khiến cảnh sát phân biệt nhầm [3].

2.1.2 YOLOv8

2.1.2.1 Tổng quan về YOLOv8

Loạt mô hình YOLO đã trở nên nổi tiếng trong lĩnh vực thị giác máy tính những năm gần đây. Sự nổi tiếng của YOLO là do độ chính xác đáng kể của nó trong khi vẫn duy trì kích thước mô hình tương đối nhỏ. Các mô hình YOLO có thể được đào tạo trên một GPU duy nhất, giúp nhiều nhà phát triển có thể tiếp cận mô hình này. Những người thực hành học máy có thể triển khai nó với chi phí thấp trên phần cứng hoặc trên đám mây [4].

YOLO đã được cộng đồng thị giác máy tính phát triển không ngừng kể từ lần ra mắt đầu tiên vào năm 2015 bởi Joseph Redmond. Trong những ngày đầu (phiên bản 1-4), YOLO được duy trì bằng ngôn ngữ C trong một khung học sâu tùy chỉnh do Redmond viết có tên là Darknet. Phiên bản YOLOv5 sau khi được Ultralytics ra mắt đã nhanh chóng được sử dụng rộng rãi nhờ cấu trúc linh hoạt của nó. Trong hai năm qua, nhiều mô hình đã phân nhánh từ YOLOv5, bao gồm Scaled-YOLOv4, YOLOR và YOLOv7. Các mô hình khác đã xuất hiện trên khắp thế giới từ những phiên bản ban đầu của riêng chúng, chẳng hạn như YOLOX và YOLOv6. Đồng thời, mỗi mô hình YOLO đã mang đến các kỹ thuật mới để tiếp tục nâng cao độ chính xác và hiệu quả của mô hình.

YOLOv8 là dòng mô hình Phát hiện đối tượng dựa trên YOLO mới nhất từ Ultralytics cung cấp hiệu suất hiện đại có thể được sử dụng cho các tác vụ phát hiện đối tượng và

phân loại hình ảnh, Ultralytics cũng chính là nhóm đã tạo ra mô hình YOLOv5 đã đạt được những thành công nhất định trước đây. YOLOv8 bao gồm nhiều thay đổi và cải tiến về kiến trúc và trải nghiệm người dùng so với YOLOv5. YOLOv8 đang tiếp tục được phát triển một cách tích cực bởi vì Ultralytics đang lắng nghe phản hồi từ cộng đồng để phát triển thêm những tính năng mới.

YOLOv8 đi kèm với các mô hình được đào tạo trước sau:

- Các điểm kiểm tra Phát hiện đối tượng được đào tạo trên bộ dữ liệu phát hiện COCO với độ phân giải hình ảnh là 640.
- Các điểm kiểm tra phân đoạn phiên bản được đào tạo trên bộ dữ liệu phân đoạn COCO với độ phân giải hình ảnh là 640.
- Các mô hình phân loại hình ảnh được đào tạo trước trên bộ dữ liệu ImageNet với độ phân giải hình ảnh là 224.

Có năm mô hình trong mỗi danh mục mô hình YOLOv8 để phát hiện, phân đoạn và phân loại. YOLOv8 Nano là nhanh nhất và nhỏ nhất, trong khi YOLOv8 Extra Large (YOLOv8x) là chính xác nhất nhưng chậm nhất trong số đó.

Bảng 2-1: Các mẫu có sẵn trong yolov8

YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	YOLOv8x

- YOLOV8n: Có thể đề cập đến phiên bản mới của mô hình YOLO, có tên là YOLO version 8, và chữ "n" có thể chỉ các thay đổi hoặc tùy chỉnh cụ thể trong kiến trúc mạng. Có thể cho thấy phiên bản này được tối ưu hóa cho xử lý hiệu quả hơn trên các thiết bị edge hoặc trong môi trường có công suất thấp.
- YOLOV8s: Tương tự, "s" có thể chỉ một phiên bản "small" hoặc "đơn giản," có thể được thiết kế để thực hiện nhanh chóng hoặc triển khai nhẹ nhàng.
- YOLOV8m: "m" có thể là viết tắt của "medium," cho thấy phiên bản này cân nhắc giữa độ phức tạp của mô hình và hiệu năng thời gian thực.
- YOLOV8l: Trong trường hợp này, "l" có thể đại diện cho "large," cho thấy phiên bản này là một mô hình phức tạp hơn có khả năng phát hiện đối tượng chính xác hơn nhưng đòi hỏi nhiều tài nguyên tính toán hơn.

- YOLOV8x: "x" có thể chỉ "extra-large" hoặc "mở rộng," cho thấy phiên bản này là mô hình YOLO mạnh mẽ nhất với hiệu suất tiên tiến nhất, nhưng có thể yêu cầu tài nguyên tính toán đáng kể.

Bảng 2-2: Các thông số của các mẫu trong YOLOv8

Model	size (pixels)	mAP _{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Bảng trên cho thấy các thông số và hiệu suất của các phiên bản YOLOv8 với kích thước đầu vào là 640 pixels.

YOLOv8n:

- Kích thước model: 640 pixels
- mAPval (Mean Average Precision) trên tập dữ liệu validation: 37.3%
- Tốc độ dự đoán trên CPU với định dạng ONNX: 80.4 ms
- Tốc độ dự đoán trên GPU A100 với TensorRT: 0.99 ms
- Số lượng tham số (params): 3.2 triệu
- Số lượng phép tính (FLOPs): 8.7 tỷ

YOLOv8s:

- Kích thước model: 640 pixels
- mAPval trên tập dữ liệu validation: 44.9%
- Tốc độ dự đoán trên CPU với định dạng ONNX: 128.4 ms
- Tốc độ dự đoán trên GPU A100 với TensorRT: 1.20 ms
- Số lượng tham số (params): 11.2 triệu
- Số lượng phép tính (FLOPs): 28.6 tỷ

YOLOv8m:

- Kích thước model: 640 pixels
- mAPval trên tập dữ liệu validation: 50.2%
- Tốc độ dự đoán trên CPU với định dạng ONNX: 234.7 ms
- Tốc độ dự đoán trên GPU A100 với TensorRT: 1.83 ms
- Số lượng tham số (params): 25.9 triệu
- Số lượng phép tính (FLOPs): 78.9 tỷ

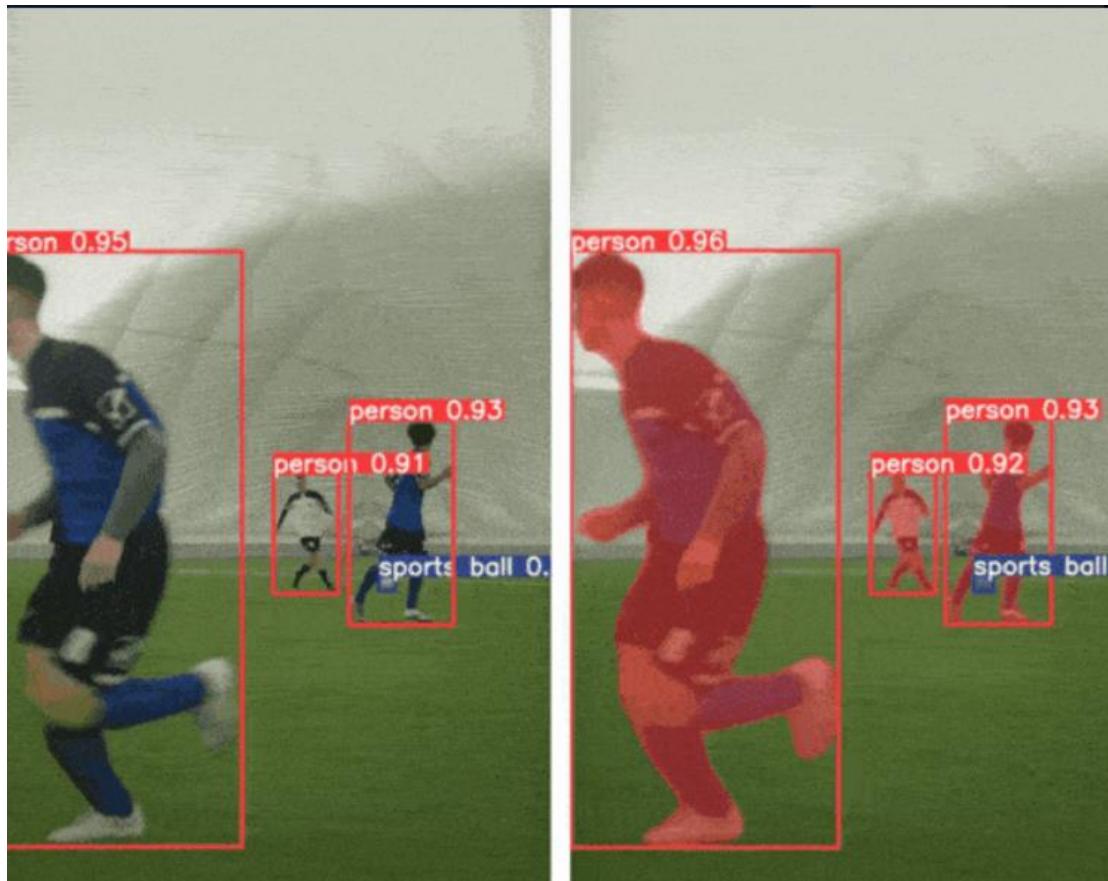
YOLOv8l:

- Kích thước model: 640 pixels
- mAPval trên tập dữ liệu validation: 52.9%
- Tốc độ dự đoán trên CPU với định dạng ONNX: 375.2 ms
- Tốc độ dự đoán trên GPU A100 với TensorRT: 2.39 ms
- Số lượng tham số (params): 43.7 triệu
- Số lượng phép tính (FLOPs): 165.2 tỷ

YOLOv8x:

- Kích thước model: 640 pixels
- mAPval trên tập dữ liệu validation: 53.9%
- Tốc độ dự đoán trên CPU với định dạng ONNX: 479.1 ms
- Tốc độ dự đoán trên GPU A100 với TensorRT: 3.53 ms
- Số lượng tham số (params): 68.2 triệu
- Số lượng phép tính (FLOPs): 257.8 tỷ

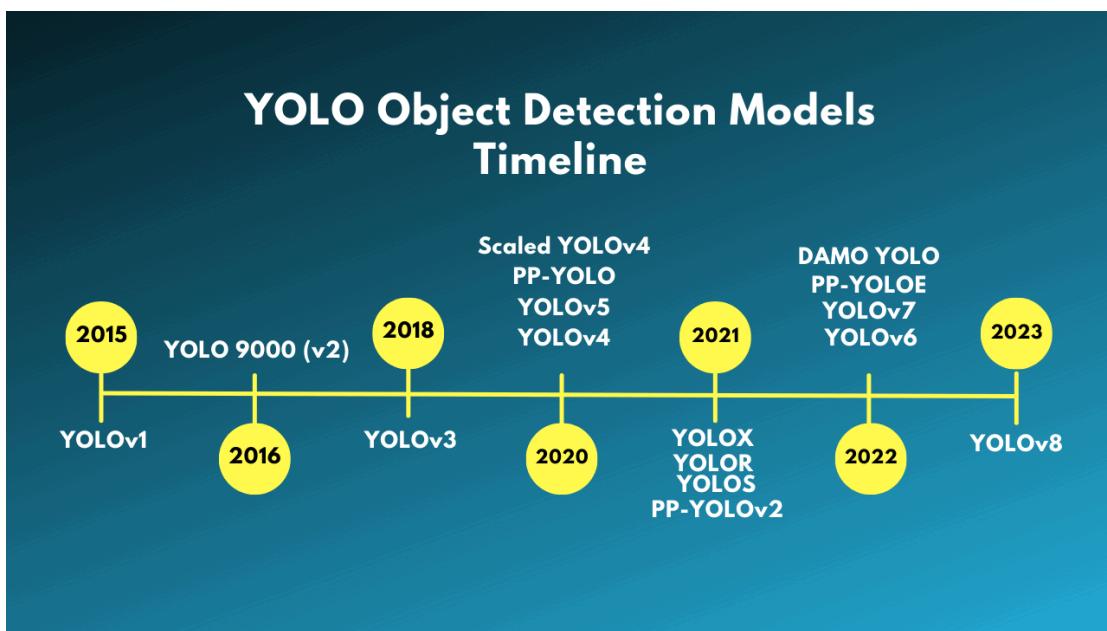
Các thông số như mAPval, tốc độ dự đoán, số lượng tham số và FLOPs đều cho thấy sự tăng dần của hiệu suất khi di chuyển từ YOLOv8n đến YOLOv8x, nhưng điều này cũng đi kèm với sự tăng về kích thước và phức tạp của mô hình. Cần cẩn nhắc lựa chọn phiên bản phù hợp dựa trên yêu cầu về hiệu suất và tài nguyên hệ thống.



Hình 2-6: Đầu ra bằng cách sử dụng các mô hình phân đoạn phiên bản và phát hiện YOLOv8x [4].

2.1.2.2 Sự phát triển của Mô hình phát hiện đối tượng YOLOv8

Đây là hình ảnh hiển thị dòng thời gian của các mô hình phát hiện đối tượng YOLO và quá trình phát triển của YOLOv8 đã diễn ra như thế nào.



Hình 2-7: Quá trình phát triển của YOLOv8 [4].

Phiên bản đầu tiên của phát hiện đối tượng YOLO, đó là YOLOv1 đã được xuất bản bởi Joseph Redmon et al. vào năm 2015. Đây là mô hình phát hiện đối tượng (SSD) một giai đoạn đầu tiên đã tạo ra SSD và tất cả các mô hình YOLO tiếp theo.

YOLOv2, còn được gọi là YOLO 9000 được xuất bản bởi tác giả gốc của YOLOv1, Joseph Redmon. Nó đã cải thiện YOLOv1 bằng cách giới thiệu khái niệm hộp neo và xương sống tốt hơn, đó là Darknet-19.

Năm 2018, Joseph Redmon và Ali Farhadi đã xuất bản YOLOv3. Nó không phải là một bước nhảy vọt về kiến trúc mà là một báo cáo kỹ thuật, nhưng dù sao cũng là một cải tiến lớn trong gia đình YOLO. YOLOv3 sử dụng xương sống Darknet-53, các kết nối còn lại, đào tạo trước tốt hơn và các kỹ thuật tăng cường hình ảnh để mang lại những cải tiến.

Tất cả các mô hình phát hiện đối tượng YOLO cho đến YOLOv3 đều được viết bằng ngôn ngữ lập trình C và sử dụng khung Darknet. Những người mới đến gặp khó khăn khi duyệt qua cơ sở mã và tinh chỉnh các mô hình.

Cùng khoảng thời gian với YOLOv3, Ultralytics đã phát hành YOLO (YOLOv3) đầu tiên được triển khai bằng khung PyTorch. Nó cũng dễ tiếp cận và dễ sử dụng hơn cho việc học chuyển tiếp.

Ngay sau khi xuất bản YOLOv3, Joseph Redmon đã rời khỏi cộng đồng nghiên cứu Thị giác Máy tính. YOLOv4 (của Alexey và cộng sự) là mô hình YOLO cuối cùng được viết trên Darknet. Sau đó, đã có nhiều vụ phát hiện đối tượng YOLO. YOLOv4, YOLOX, PP-YOLO, YOLOv6 và YOLOv7 được chia tỷ lệ là một số nổi bật trong số đó.

Sau YOLOv3, Ultralytics cũng phát hành YOLOv5 thậm chí còn tốt hơn, nhanh hơn và dễ sử dụng hơn tất cả các mô hình YOLO khác.

Tính đến thời điểm hiện tại (tháng 1 năm 2023), Ultralytics đã xuất bản YOLOv8 trong kho lưu trữ ultralytics, đây có lẽ là mô hình YOLO tốt nhất cho đến nay.

2.1.2.3 Tại sao nên sử dụng YOLOv8?

YOLOv8 là một phiên bản nâng cấp của mô hình nhận dạng vật thể YOLO (You Only Look Once). Đây là một mô hình nhận dạng vật thể nhanh chóng và hiệu quả, có nhiều ưu điểm giúp nó trở thành lựa chọn phổ biến trong lĩnh vực xử lý hình ảnh và thị giác máy tính. Dưới đây là một số lý do tại sao nên sử dụng YOLOv8:

1. Tốc độ xử lý cao: YOLOv8 được thiết kế để hoạt động với tốc độ cao. Với việc sử dụng kỹ thuật "chỉ nhìn một lần" (you only look once), nó có thể phân loại và định vị vật thể trong một hình ảnh chỉ trong một lần feedforward qua mạng neural. Điều này giúp YOLOv8 thực hiện nhận dạng vật thể nhanh chóng và thích hợp cho các ứng dụng thời gian thực.
2. Độ chính xác cao: YOLOv8 sử dụng kiến trúc mạng neural sâu để xây dựng mô hình nhận dạng vật thể. Kiến trúc này kết hợp giữa đặc điểm cục bộ và toàn cục của hình ảnh để đưa ra dự đoán chính xác về vị trí và loại vật thể. So với các phương pháp khác, YOLOv8 có khả năng nhận dạng chính xác các vật thể khác nhau trong cùng một hình ảnh.
3. Hỗ trợ đa lớp đối tượng: YOLOv8 có khả năng nhận dạng và phân loại các vật thể vào các lớp khác nhau trong một hình ảnh. Điều này rất hữu ích trong nhiều tình huống, đặc biệt là trong các ứng dụng thực tế. Ví dụ, khi áp dụng YOLOv8 trong một hệ thống giám sát an ninh, nó có thể nhận dạng và phân loại người, ô tô, xe máy, động vật và các đối tượng khác có liên quan. Thông

qua quá trình huấn luyện với dữ liệu đa lớp, YOLOv8 học được các đặc trưng độc lập về hình dạng, màu sắc và vị trí của các vật thể khác nhau. Điều này cho phép mô hình nhận biết và phân loại đa dạng các đối tượng một cách chính xác và đáng tin cậy.

4. Tích hợp dễ dàng: YOLOv8 có thể tích hợp vào các ứng dụng và hệ thống khác một cách dễ dàng. Mô hình này có sẵn trong các framework deep learning phổ biến như TensorFlow và PyTorch, cho phép người dùng triển khai nhanh chóng trên nền tảng của mình. Việc tích hợp YOLOv8 vào các ứng dụng như giám sát an ninh, xe tự hành, hay xử lý ảnh y tế trở nên đơn giản và thuận tiện.
5. Tính linh hoạt và dễ dàng tùy chỉnh: YOLOv8 cung cấp cho người dùng tính linh hoạt và khả năng tùy chỉnh mô hình theo yêu cầu cụ thể của họ. Người dùng có thể điều chỉnh các tham số như độ phân giải ảnh đầu vào, kích thước lưới, ngưỡng nhận dạng và các tham số khác để tối ưu hóa hiệu suất và độ chính xác của mô hình. Điều này cho phép YOLOv8 có thể được áp dụng trong các tình huống và ứng dụng đa dạng, từ việc nhận dạng đối tượng trong ảnh chụp từ camera an ninh đến việc phát hiện và phân loại vật thể trong các tác vụ xe tự hành.
6. Tiên bội và sự phát triển liên tục: YOLOv8 là một phiên bản nâng cấp của YOLO và đã được cải tiến với nhiều tính năng mới. Tuy nhiên, việc nghiên cứu và phát triển trong lĩnh vực nhận dạng vật thể vẫn đang diễn ra mạnh mẽ. Có sự xuất hiện của các phiên bản YOLO tiếp theo như YOLOv9 và các mô hình nhận dạng vật thể khác với hiệu suất và tính năng cải tiến.

Tuy YOLOv8 có nhiều ưu điểm, nhưng cũng có một số hạn chế:

1. Hiệu suất và tài nguyên: YOLOv8 đạt được tốc độ xử lý nhanh chóng, nhưng điều này đồng nghĩa với việc tiêu tốn nhiều tài nguyên tính toán và bộ nhớ. Nếu bạn có hạn chế về tài nguyên, hoặc cần thực hiện trên các thiết bị có khả năng tính toán hạn chế như điện thoại di động, bạn có thể cần xem xét các mô hình nhẹ hơn để đáp ứng yêu cầu của mình.
2. Kích thước đối tượng: YOLOv8 hoạt động tốt trên việc nhận dạng vật thể lớn và trung bình, nhưng có thể gặp khó khăn trong việc nhận dạng các vật thể nhỏ hoặc mờ. Nếu ứng dụng của bạn liên quan đến việc nhận dạng và phân

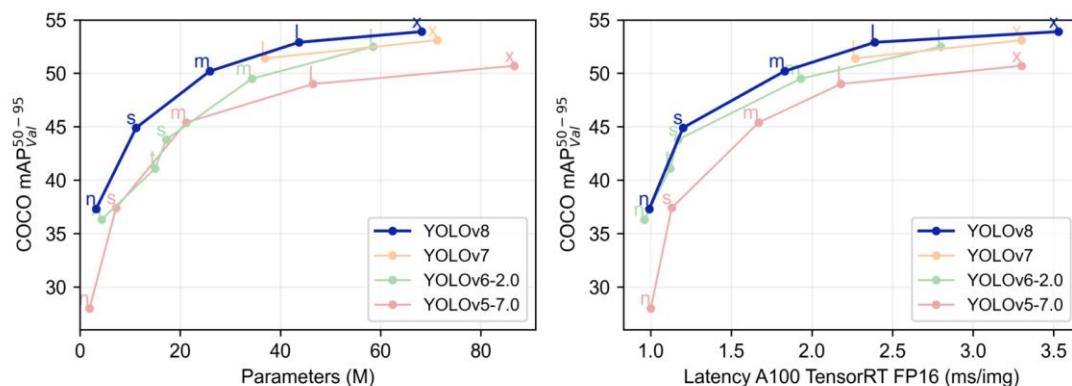
loại các vật thể nhỏ, có thể cần xem xét sử dụng các mô hình có khả năng đặc trưng tốt hơn đối với vật thể nhỏ hơn.

3. Dữ liệu huấn luyện: Hiệu suất của mô hình YOLOv8 phụ thuộc vào chất lượng và đa dạng của dữ liệu huấn luyện. Nếu bạn có một tập dữ liệu đa dạng và đại diện cho các vật thể mục tiêu trong ứng dụng của bạn, YOLOv8 có thể cho kết quả tốt. Tuy nhiên, nếu dữ liệu huấn luyện của bạn hạn chế hoặc không đại diện đúng mô hình trường thực tế, có thể xem xét việc sử dụng các mô hình có khả năng học từ dữ liệu ít hơn.
4. Thời gian đào tạo và triển khai: YOLOv8 yêu cầu quá trình huấn luyện trên một tập dữ liệu lớn và có thể tốn thời gian. Nếu bạn có hạn chế thời gian hoặc cần triển khai nhanh chóng, bạn có thể xem xét việc sử dụng các phiên bản YOLO khác đã được huấn luyện trước đó hoặc sử dụng các mô hình nhận dạng vật thể khác có khả năng triển khai nhanh.

Tuy YOLOv8 có nhiều ưu điểm, nhưng cũng có một số hạn chế, do đặc thù của phương pháp "chỉ nhìn một lần". Tuy nhiên, với những ứng dụng yêu cầu tốc độ xử lý nhanh và khả năng nhận dạng chính xác trên các vật thể lớn, YOLOv8 vẫn là một lựa chọn phổ biến và hữu ích.

2.1.2.4 So sánh mô hình YOLOv8 với mô hình YOLO khác

Các mô hình YOLOv8 thường như hoạt động tốt hơn nhiều so với các mô hình YOLO trước đó. Không chỉ các mẫu YOLOv5, YOLOv8 còn dẫn đầu so với các mẫu YOLOv7 và YOLOv6, được hiển thị chi tiết trong hình 2.8.



Hình 2-8: YOLOv8 so với các mô hình YOLO khác [4].

Biểu đồ với trục dọc là coco mAP (50-95): Đây là một chỉ số đo lường hiệu năng của mô hình phát hiện đối tượng trên tập dữ liệu COCO. Giá trị mAP cao hơn cho biết mô hình có hiệu năng tốt hơn.

Trục ngang của biểu đồ này có thể là:

- a. parameters (m): Số lượng tham số của mô hình. Mô hình với ít tham số hơn thường nhẹ hơn và nhanh hơn khi triển khai, nhưng có thể kém hiệu quả hơn.
- b. latency A100 TensorRT FP16 (ms/img): Độ trễ khi xử lý một hình ảnh trên GPU NVIDIA A100 sử dụng TensorRT ở chế độ FP16.

Biểu đồ so sánh giữa các phiên bản (n, s, m, l, x) của YOLOv8, YOLOv7, YOLOv6-2.0, và YOLOv5-7.0.

Trong YOLO (và một số mô hình khác), các phiên bản như "n", "s", "m", "l", "x" thường đại diện cho kích thước và độ phức tạp của mô hình, từ nhỏ nhất ("n" hoặc "s") đến lớn nhất ("x").

Về hiệu năng: Càng gần phía trên cùng của biểu đồ (giá trị mAP cao), mô hình càng hiệu quả. Có sự cải tiến dần từ YOLOv5-7.0, YOLOv6-2.0, YOLOv7, và YOLOv8 và cao dần từ n, s, m, l, x.

Khi so sánh với các mô hình YOLO khác được đào tạo ở độ phân giải hình ảnh 640, tất cả các mô hình YOLOv8 đều có thông lượng tốt hơn với cùng một số tham số.

Bây giờ, hãy xem chi tiết hiệu quả hoạt động của các mô hình YOLOv8 mới nhất khi kết hợp với các mô hình YOLOv5 từ Ultralytics. Các bảng sau đây cho thấy sự so sánh toàn diện giữa YOLOv8 và YOLOv5:

Performance Comparison of YOLOv8 vs YOLOv5

Model Size	Detection*	Segmentation*	Classification*
Nano	+33.21%	+32.97%	+3.10%
Small	+20.05%	+18.62%	+1.12%
Medium	+10.57%	+10.89%	+0.66%
Large	+7.96%	+6.73%	0.00%
Xtra Large	+6.31%	+5.33%	-0.76%

*Image Size = 640

*Image Size = 224

Hình 2-9: So sánh tổng thể Mô hình YOLOv8 so với mô hình YOLOv5 [4].

So sánh Hiệu suất giữa YOLOv8 và YOLOv5. Đối với từng kích thước mô hình, có một phần trăm tương ứng mô tả sự cải thiện (hoặc giảm) của YOLOv8 so với YOLOv5:

- Phát hiện (Detection)

Nano: +33.21%, Small: +20.05%, Medium: +10.57%, Large: +7.96%, Xtra Large: +6.31%

- Phân đoạn (Segmentation)

Đối với kích thước hình ảnh là 640.

Nano: +32.97%, Small: +18.62%, Medium: +10.89%, Large: +6.73%, Xtra Large: +5.33%

- Phân lớp (Classification):

Đối với kích thước hình ảnh là 224

Nano: +3.10%, Small: +1.12%, Medium: +0.66%, Large: 0.00%, Xtra Large: -0.76%

Dựa trên số liệu trên, ta có thể thấy:

YOLOv8 hiển thị mức độ cải tiến rõ rệt với kích thước mô hình "Nano" và "Small" so với YOLOv5.

Sự cải tiến giảm dần khi kích thước mô hình tăng lên, và thậm chí hiệu suất giảm ở mô hình "Xtra Large" khi thực hiện phân loại trên hình ảnh kích thước 640.

Tóm lại, dựa vào thông tin bạn cung cấp, YOLOv8 có hiệu suất tốt hơn so với YOLOv5, đặc biệt ở các mô hình nhỏ hơn. Tuy nhiên, sự cải thiện này giảm dần khi kích thước mô hình tăng lên.

Object Detection Performance Comparison (YOLOv8 vs YOLOv5)

Model Size	YOLOv5	YOLOv8	Difference
Nano	28	37.3	+33.21%
Small	37.4	44.9	+20.05%
Medium	45.4	50.2	+10.57%
Large	49	52.9	+7.96%
Xtra Large	50.7	53.9	+6.31%

*Image Size = 640

Hình 2-10: So sánh phát hiện đối tượng YOLOv8 & YOLOv5 [4].

- Hiệu suất YOLOv5:

Nano: 28mAP, Small: 37.4mAP, Medium: 45.4mAP, Large: 49mAP, Xtra Large: 50.7mAP

- Hiệu suất YOLOv8:

Nano: 37.3mAP, Small: 44.9mAP, Medium: 50.2mAP, Large: 52.9mAP, Xtra Large: 53.9mAP

- Sự khác biệt:

Nano: +33.21%, Small: +20.05%, Medium: +10.57%, Large: +7.96%, Xtra Large: +6.31%

- Nhận định:

YOLOv8 hiển thị mức độ cải tiến rõ rệt so với YOLOv5 ở tất cả các kích thước mô hình, nhưng sự cải thiện này giảm dần khi kích thước mô hình tăng lên.

Sự cải thiện lớn nhất (+33.21%) là ở kích thước "Nano", trong khi sự cải thiện thấp nhất (+6.31%) là ở kích thước "Xtra Large".

Dù có sự cải tiến ở tất cả các kích thước, nhưng cần chú ý rằng đối với những ứng dụng yêu cầu độ chính xác cao nhất, việc chọn mô hình "Xtra Large" của YOLOv8 chỉ mang lại sự cải thiện nhỏ so với YOLOv5.

Dựa trên số liệu trên, YOLOv8 có hiệu suất tốt hơn so với YOLOv5 ở tất cả các kích thước mô hình. Tuy nhiên, sự cải thiện này giảm dần khi kích thước mô hình tăng lên.

Instance Segmentation Performance Comparison (YOLOv8 vs YOLOv5)

Model Size	YOLOv5	YOLOv8	Difference
Nano	27.6	36.7	+32.97%
Small	37.6	44.6	+18.62%
Medium	45	49.9	+10.89%
Large	49	52.3	+6.73%
Xtra Large	50.7	53.4	+5.33%

*Image Size = 640

Hình 2-11: So sánh phân đoạn phiên bản YOLOv8 & YOLOv5 [4].

- Hiệu suất YOLOv5 (Instance Segmentation):

Nano: 27.6 mAP, Small: 37.6 mAP, Medium: 45 mAP, Large: 49 mAP, Xtra Large: 50.7 mAP

- Hiệu suất YOLOv8 (Instance Segmentation):

Nano: 36.7 mAP, Small: 44.6 mAP, Medium: 49.9 mAP, Large: 52.3 mAP, Xtra Large: 53.4 mAP

- Sự khác biệt giữa YOLOv8 và YOLOv5:

Nano: +32.97%, Small: +18.62%, Medium: +10.89%, Large: +6.73%, Xtra Large: +5.33%

- Nhận xét:

YOLOv8 lại một lần nữa cho thấy hiệu suất tốt hơn so với YOLOv5 trong việc Instance Segmentation ở mọi kích thước mô hình.

Sự cải tiến của YOLOv8 so với YOLOv5 giảm dần khi kích thước mô hình tăng lên, giống như quan sát về hiệu suất phát hiện đối tượng.

Cụ thể, sự cải tiến lớn nhất (+32.97%) được thấy ở kích thước "Nano", trong khi sự cải tiến thấp nhất (+5.33%) là ở kích thước "Xtra Large".

Tổng hợp, YOLOv8 cung cấp hiệu suất tốt hơn trong nhiệm vụ Instance Segmentation so với YOLOv5 ở mọi kích thước mô hình, nhưng mức độ cải tiến giảm khi kích thước mô hình tăng.

Image Classification Performance Comparison (YOLOv8 vs YOLOv5)

Model Size	YOLOv5	YOLOv8	Difference
Nano	64.6	66.6	+3.10%
Small	71.5	72.3	+1.12%
Medium	75.9	76.4	+0.66%
Large	78	78	0.00%
Xtra Large	79	78.4	-0.76%

*Image Size = 224

Hình 2-12: So sánh phân loại hình ảnh YOLOv8 & YOLOv5 [4].

- Hiệu suất YOLOv5 (Phân loại hình ảnh):

Nano: 64.6, Small: 71.5, Medium: 75.9, Large: 78, Xtra Large: 79.

- Hiệu suất YOLOv8 (Phân loại hình ảnh):

Nano: 66.6, Small: 72.3, Medium: 76.4, Large: 78, Xtra Large: 78.4.

- Sự khác biệt giữa YOLOv8 và YOLOv5:

Nano: +3.10%, Small: +1.12%, Medium: +0.66%, Large: 0.00%, Xtra Large: -0.76%.

- Nhận xét:

YOLOv8 có hiệu suất tốt hơn YOLOv5 ở mức độ nhỏ ở ba kích thước mô hình đầu tiên: Nano, Small và Medium.

Tuy nhiên, khi đến kích thước mô hình "Large", hiệu suất của cả hai mô hình là như nhau.

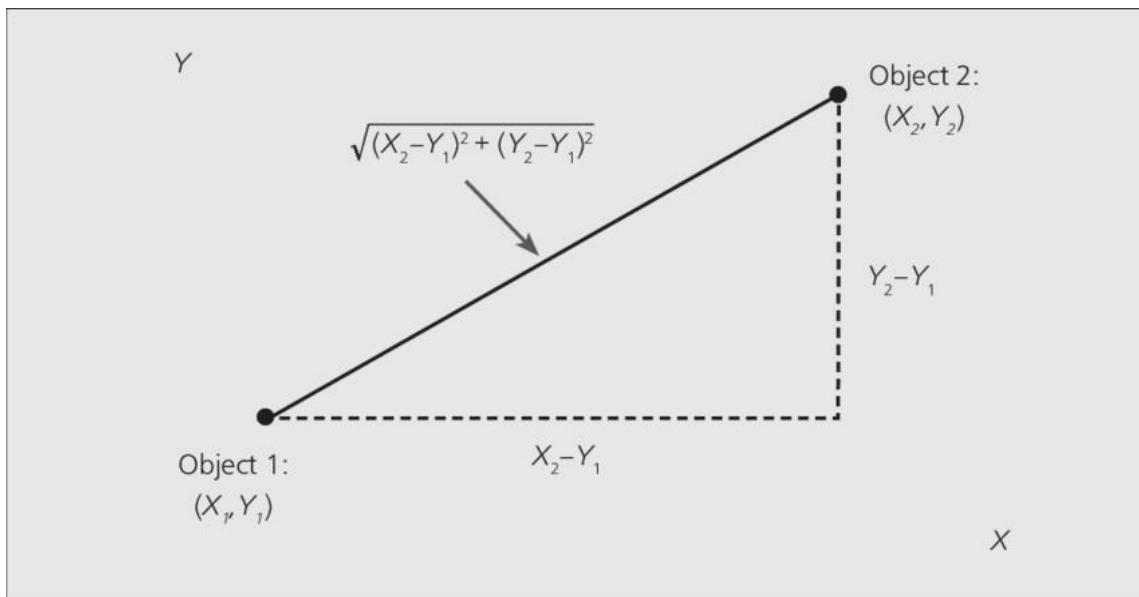
Đối với kích thước mô hình "Xtra Large", YOLOv8 lại có hiệu suất thấp hơn YOLOv5 một chút (-0.76%).

Tổng kết, YOLOv8 và YOLOv5 có hiệu suất phân loại hình ảnh khá gần nhau. Trong khi YOLOv8 có hiệu suất tốt hơn ở một số kích thước mô hình nhỏ hơn, nó lại không duy trì được lợi thế này ở kích thước lớn hơn.

2.2 Công thức đo tốc độ

Công thức ước tính tốc độ được xây dựng từ hàm hàm “*estimatespeed*” được sử dụng để ước tính tốc độ của một đối tượng dựa trên hai vị trí đã được ghi nhận của nó, có thể được viết như sau:

1. Công thức khoảng cách Euclidean:



Hình 2-13: Minh họa công thức khoảng cách Euclidean giữa hai điểm.

Công thức khoảng cách Euclidean giữa hai điểm (x_1, y_1) và (x_2, y_2) trong không gian hai chiều là:

$$d_{pixel} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

Trong đó:

d_{pixel} : Khoảng cách giữa hai vị trí của đối tượng được tính trong không gian pixel, đơn vị: *Pixel*

x_1, y_1 : tọa độ (x, y) của vị trí đầu tiên, đơn vị: *Pixel*

x_2, y_2 tọa độ (x, y) của vị trí thứ hai, đơn vị: *Pixel*

Đây là một hàm tính tốc độ dựa trên vị trí của một đối tượng trong hai khung hình khác nhau. Cụ thể, hàm này nhận vào hai tham số $Location_1$ và $Location_2$, đại diện cho tọa độ của đối tượng trong hai khung hình liên tiếp.

2. *Chuyển đổi pixel thành mét:*

ppm (*pixels per meter*) hằng số chuyển đổi giữa số pixel và thực tế khoảng cách trong không gian thực. Giá trị *ppm* rất hữu ích trong các ứng dụng liên quan đến xử lý ảnh và video, như đo lường khoảng cách giữa các đối tượng trong ảnh, ước lượng tốc độ di chuyển dựa trên sự di chuyển trong ảnh, hoặc bất kỳ ứng dụng nào cần chuyển đổi giữa không gian pixel và không gian thực tế.

Nó biểu thị số lượng pixel trên mỗi mét trong không gian thực, đơn vị: *Pixel/Meter*.

Để chuyển từ khoảng cách ảnh pixel sang khoảng cách thực tế (tính bằng mét), ta chia khoảng cách tính bằng pixel cho ppm:

$$d_{meters} = \frac{d_{pixel}}{ppm} \quad (2)$$

d_{meters} : Khoảng cách giữa hai vị trí của đối tượng được chuyển đổi sang không gian thực từ không gian pixel, đơn vị: *Meter (m)*

3. *Ước tính tốc độ:*

Hằng số thời gian (chuyển đổi từ khung hình sang giờ): $time_{constant} = 30 \times 3,6$

Trong đó:

$time_{constant}$: Biểu diễn thời gian mà đối tượng di chuyển từ vị trí điểm (x_1, y_1) và (x_2, y_2) .

Giá trị này được nhân với 3.6 ($1m/s = 3.6 km/h$) (chuyển đổi từ *meter (m)* sang *kilometer (km)* và từ *giây (s)* sang *giờ (h)*) để đưa ra ước tính tốc độ trong đơn vị *km/h*.

Giá trị 30 là đại diện cho khung hình mỗi giây vì chúng ta có tần số ở đây trong trường hợp này là $frequency = \frac{1}{time}$ (fps). Đây cũng là thường thấy trong các khung hình video.

Dựa trên công thức vật lý cơ bản, tốc độ (v) là quãng đường chia cho thời gian. Trong trường hợp này, quãng đường là d_{meters} và thời gian là $time_{constant}$. Vì vậy, tốc độ được tính như sau:

$$v = d_{meters} \times time_{constant} \quad (3)$$

v : Tốc độ ước tính của đối tượng dựa trên khoảng cách đã di chuyển và thời gian di chuyển. Đơn vị: Km/h.

2.3 Đo lưu lượng xe ra vào

Sử dụng một mã ID để theo dõi sự di chuyển của đối tượng qua thời gian trong dãy hình ảnh hoặc video. Mỗi đối tượng được xác định bởi một ID duy nhất.

Đầu tiên, vị trí trung tâm mới nhất của đối tượng được thêm vào đầu của `data_deque`, một danh sách hai chiều, dựa trên ID của đối tượng. Danh sách này giữ lại lịch sử vị trí trung tâm của đối tượng qua thời gian.

Khi danh sách này chứa ít nhất hai vị trí trung tâm, hướng di chuyển của đối tượng giữa hai vị trí này được xác định bằng hàm `get_direction`. Tốc độ của đối tượng cũng được ước tính dựa trên khoảng cách giữa hai điểm này.

Ngoài ra, hàm còn kiểm tra xem trong quá trình di chuyển từ vị trí này sang vị trí khác, đối tượng có cắt qua một đường ngang nhất định hay không. Nếu có, nó sẽ xác định hướng di chuyển của đối tượng là về phía Nam hay phía Bắc dựa vào hướng đã được xác định trước đó. Dựa vào điều này, mã sẽ đếm số lượng đối tượng di chuyển theo từng hướng.

Tóm lại, điều này giúp theo dõi và phân tích sự di chuyển của các đối tượng, xác định hướng và tốc độ của chúng, và đếm số lượng đối tượng di chuyển qua một đường ngang nhất định theo từng hướng.

2.4 Kết luận chương 2

Các phương pháp và thuật toán trong học máy và học sâu đóng vai trò quan trọng trong việc khai thác thông tin từ dữ liệu và tạo ra các mô hình mạnh mẽ cho nhiều ứng dụng thực tế. Sau khi so sánh các mô hình YOLO, có thể nhận thấy YOLOv8 là mô hình mới nhất có độ ưu việt hơn so với những cái còn lại. Công thức đo tốc độ là nền tảng cho đề tài này.

CHƯƠNG 3. KẾT QUẢ VÀ PHÂN TÍCH

3.1 Nền tảng thực hiện

Colaboratory hay còn gọi là Google Colab, là một sản phẩm được phát triển từ Google, cho phép thực thi các lệnh python thông qua trình duyệt kết nối mạng, hiện đang là một trong những công cụ hỗ trợ mạnh mẽ trong việc nghiên cứu máy học (Machine Learning), phân tích dữ liệu (Data Analysis) và giáo dục.

Google Colab là một sản phẩm giống Jupyter Notebook của Google Research. Nói cách khác, Google Colab là phiên bản được lưu trữ trên đám mây. Để sử dụng Google Colab, bạn không cần cài đặt và thời gian chạy hoặc nâng cấp phần cứng máy tính của mình. Bạn có thể viết mã Python bằng Colab trên trình duyệt web Google Chrome hoặc Mozilla Firefox của mình. Bạn cũng có thể thực thi các mã đó trên trình duyệt mà không cần bất kỳ môi trường thời gian chạy hoặc giao diện dòng lệnh nào.

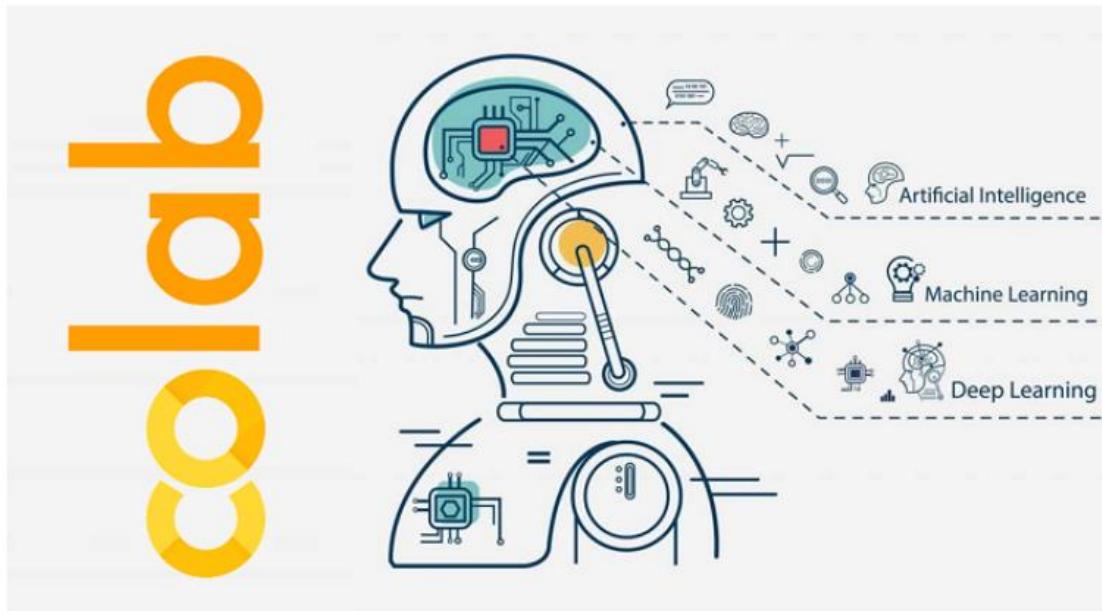
Hơn nữa, bạn có thể cung cấp cho sổ ghi chép dự án Python của mình một giao diện chuyên nghiệp bằng cách thêm các phương trình toán học, đồ thị, bảng, hình ảnh và đồ họa khác. Ngoài ra, bạn có thể mã hóa hình ảnh dữ liệu bằng Python và Colab sẽ hiển thị mã trong một tài sản trực quan. Colab cho phép bạn sử dụng lại các tệp Jupyter Notebook từ GitHub. Ngoài ra, bạn cũng có thể nhập các dự án khoa học dữ liệu và máy học tương thích từ các nguồn khác. Colab xử lý nội dung đã nhập một cách hiệu quả để hiển thị mã Python rõ ràng và không có lỗi.

Để đáp ứng các yêu cầu về khối lượng công việc nặng về CPU/GPU của Python. Hơn nữa, Colab cung cấp cho bạn quyền truy cập miễn phí vào cơ sở hạ tầng điện toán như bộ lưu trữ, bộ nhớ, khả năng xử lý, đơn vị xử lý đồ họa (GPU) và đơn vị xử lý tensor (TPU).

Google đã lập trình đặc biệt công cụ mã hóa Python dựa trên đám mây này để ghi nhớ nhu cầu của các lập trình viên máy học, nhà phân tích dữ liệu lớn, nhà khoa học dữ liệu, nhà nghiên cứu AI và người học Python.

Phần tốt nhất là một sổ ghi chép mã cho tất cả các thành phần cần thiết để trình bày một dự án khoa học dữ liệu hoặc máy học hoàn chỉnh cho người giám sát

hoặc nhà tài trợ chương trình. Ví dụ: sổ ghi chép Colab của bạn có thể chứa mã thực thi, mã Python trực tiếp, văn bản có định dạng, HTML, LaTeX, hình ảnh, hình ảnh hóa dữ liệu, biểu đồ, đồ thị, bảng, v.v.

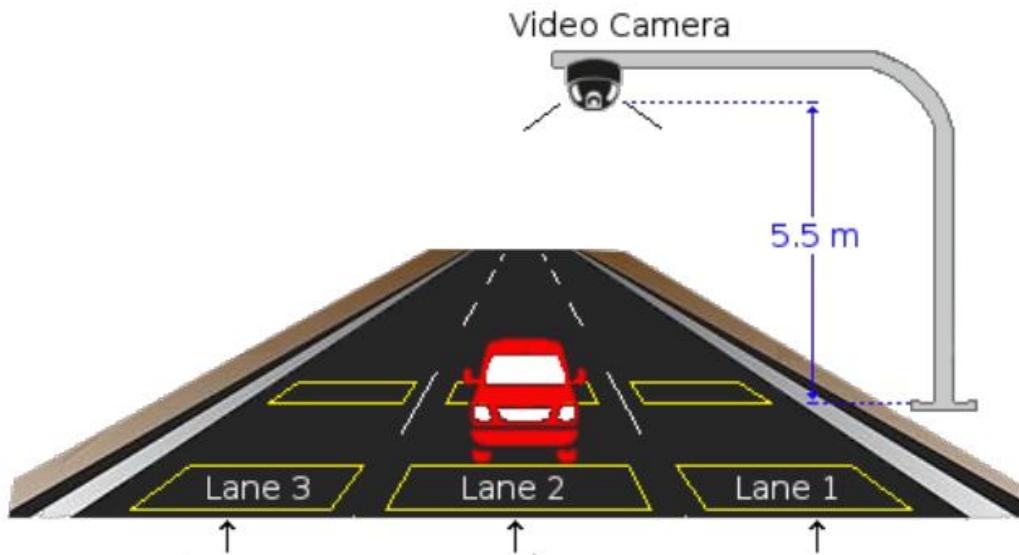


Hình 3-1: Các tính năng tốt nhất của Google Colab [5].

Google Colab không yêu cầu về cấu hình hay tài nguyên máy tính, mọi thứ được thực thi trên trình duyệt chính như một số tay, đặc biệt Colab cho phép sử dụng tài nguyên máy tính từ CPU tốc độ cao, sử dụng miễn phí GPUs và TPUs, phù hợp cho sinh viên trong việc học tập. Google Colab hạn chế về việc chọn lựa cũng như thời gian sử dụng tài nguyên, thời gian tối đa cho một lần sử dụng là 12 giờ đồng hồ liên tiếp nếu sử dụng miễn phí [5].

3.2 Phương pháp thiết lập

Trong nhiều hệ thống điều khiển tốc độ, một số cảm biến chuyên dụng (ví dụ như cảm biến vòng lặp, cảm biến siêu âm Doppler, v.v.) được sử dụng để kích hoạt một máy ảnh video để ghi lại các phương tiện vượt quá giới hạn tốc độ. Tuy nhiên, chúng ta có thể đơn giản hóa các hệ thống này bằng cách trích xuất thông tin về tốc độ từ các khung hình video đã có sẵn. Như vậy, chúng ta có thể thu được một hệ thống không xâm lấn với chi phí giảm đáng kể. Như một sản phẩm phụ của việc ghi lại video liên tục.



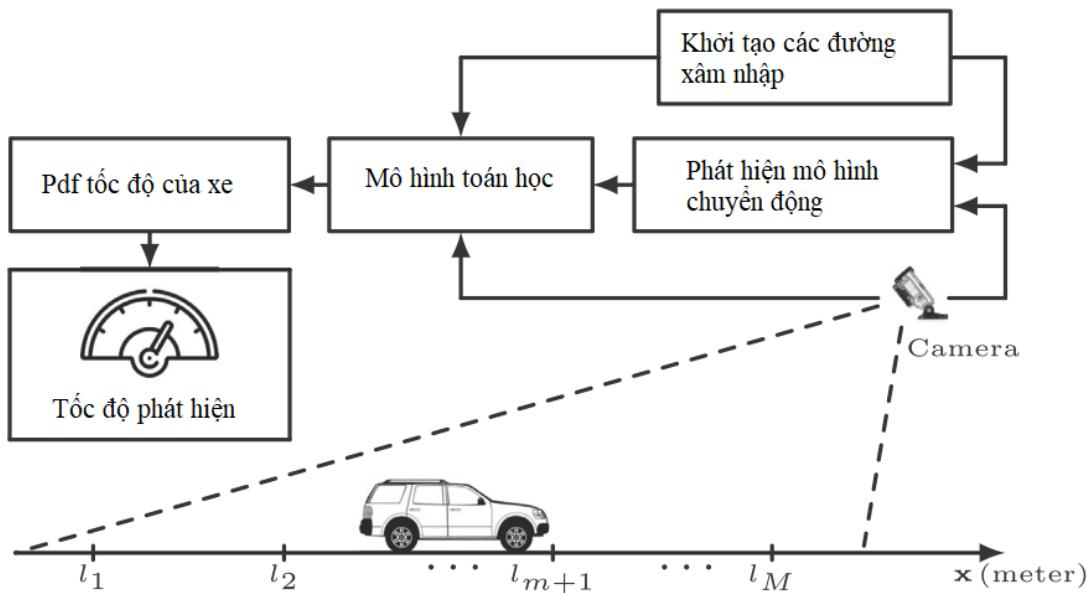
Hình 3-2: Thiết lập thử nghiệm.

Đầu vào dữ liệu cho bài toán nhận dạng và ước tính tốc độ là một video kỹ thuật số V , được định nghĩa là một chuỗi n hình ảnh (khung hình) $V(0), V(1), \dots, V(n - 1)$ với một miền chung, được cách đều nhau trong thời gian. Video này được ghi lại bởi một máy ảnh cố định đặt ở một vị trí phù hợp để ghi lại biển số xe ở ba làn đường kế nhau.

Các đầu ra bao gồm: các quỹ đạo được theo dõi của các khu vực này trong các khung hình liên tiếp, và ước tính tốc độ cho mỗi phương tiện.

Một số giả định được đưa ra về cảnh quan và miền vấn đề: mỗi làn đường nằm trên một mặt phẳng; các phương tiện, trong khu vực đo tốc độ, di chuyển với tốc độ không đổi và theo quỹ đạo thẳng từ phía dưới lên phía trên hay phía trên xuống dưới của hình ảnh.

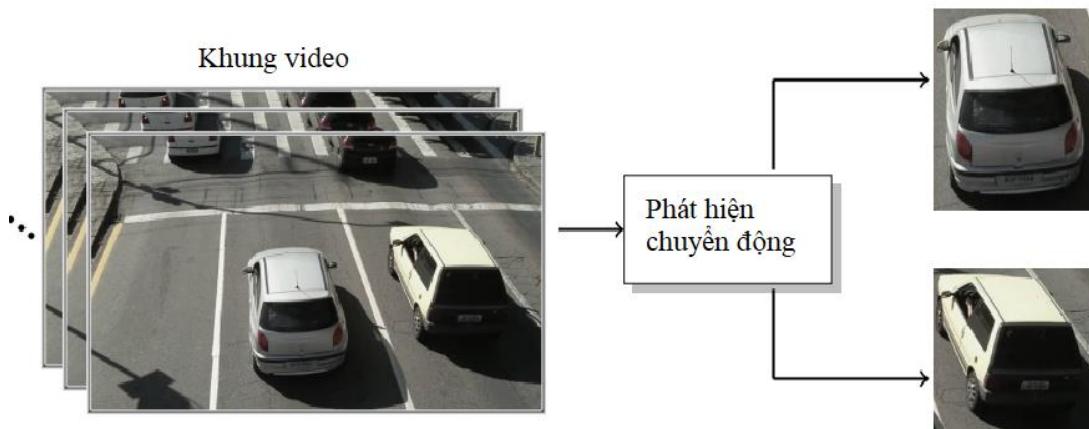
3.3 Giải thuật thực hiện



Hình 3-3: Một cảnh phát hiện chứa một số đường xâm nhập (L_{m+1} , trong đó $m \in \{0, \dots, M - 1\}$) và một chiếc xe đi qua được quan sát bởi một chiếc máy ảnh đối diện với đường.

Trong phần này, một mô hình được giới thiệu nhằm xem xét các thông số chính góp phần vào sự không chắc chắn của một hệ thống đo tốc độ. Hệ thống phát hiện dựa trên một kịch bản với nhiều đường xâm nhập, có kiến thức về vị trí tương đối của chúng trong thế giới thực và một phương tiện đi qua. Tốc độ của phương tiện được giả định là không đổi khi vượt qua các đường xâm nhập vì chiều dài của cảnh đo là rất ngắn (vài mét). Mô hình đề xuất sau đó tạo ra một phân phối xác suất của tốc độ phương tiện dựa trên một vector mẫu chuyển động được phát hiện. Triển khai hệ thống như vậy được miêu tả sau đó trong phần này. Việc sử dụng các đường xâm nhập cũng như lưu trữ các sự kiện trong một vector mẫu là vô cùng hiệu quả đối với cả bộ nhớ và độ phức tạp tính toán.

Mô tả một thuật toán nhanh cho việc phát hiện chuyển động. Mục tiêu ở đây là hiệu quả thu hẹp vị trí trong cảnh với chuyển động đang diễn ra.



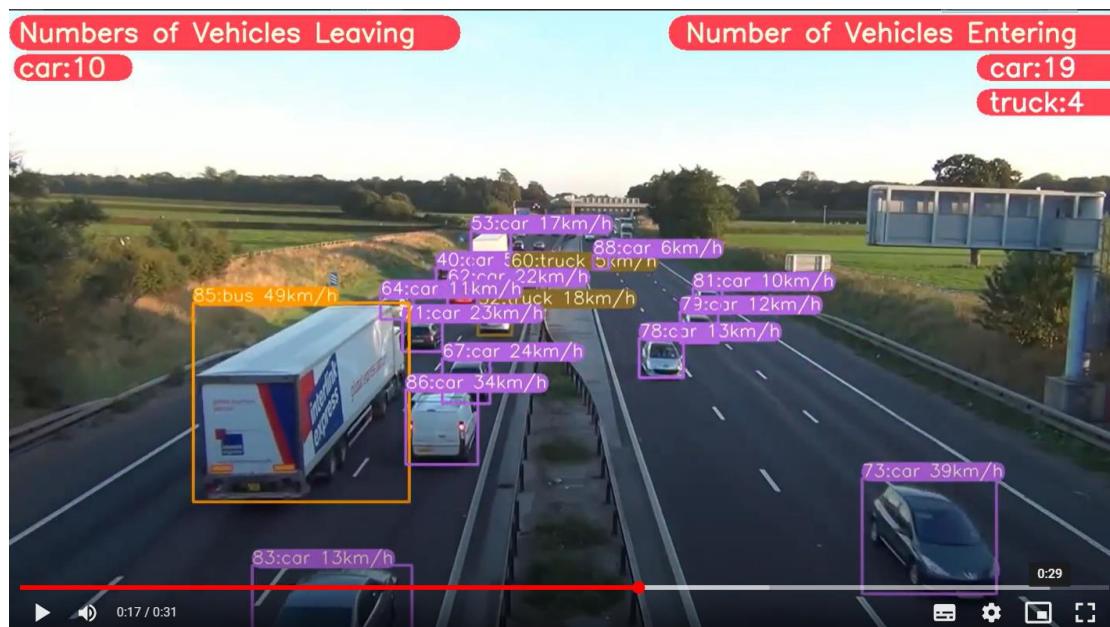
Hình 3-4: Phát hiện chuyển động: các khung hình video đầu vào được thu gọn thành các vùng quan tâm giới hạn bởi các phương tiện đang di chuyển.

Bộ phát hiện chuyển động được phát triển sử dụng một chuỗi các khung hình liên tiếp để tính toán một hình ảnh nhị phân, trong đó các vùng nền trắng biểu thị các phương tiện đang di chuyển, và vùng nền đen biểu thị cảnh tĩnh. Một khía cạnh quan trọng để tiết kiệm thời gian là việc sử dụng lưới thưa thớt thông thường để tính toán hình ảnh nhị phân. Đầu ra của bộ phát hiện chuyển động là một tập hợp các vùng quan tâm mà lý tưởng là chứa toàn bộ từng khung hình và loại xe có trong đó.

3.4 Kết quả và phân tích

3.4.1 Kết quả mô phỏng trên video thực nghiệm thực tế ghi lại từ camera đường bộ

Dưới đây là kết quả được cắt từ video thực nghiệm thực tế được ghi lại từ camera đường bộ:



Hình 3-5: Mô phỏng trên video thực nghiệm thực tế ghi lại từ camera đường bộ.

Vì đây là hình ảnh được lấy từ camera gắn cố định với góc rộng tốt, không có rung lắc dù nhẹ nên kết quả đầu ra về đo tốc độ khá khả quan và có độ chính xác tương đối, còn về đếm lưu lượng xe thì do đây là đường xe hai chiều, kết quả thể hiện rõ đầu vào và ra của mỗi chiều xe. Mỗi xe được gắn một mã ID khác nhau và mỗi loại xe được cho một màu khác nhau để thuận tiện cho việc đếm lưu lượng và phân biệt loại xe. Về phần phân loại và đếm lưu lượng xe thì mô hình chạy tốt và độ chính xác khá cao.

```
YOLOv8m summary: 218 layers, 25886080 parameters, 0 gradients, 78.9 GFLOPs
video 1/1 (1/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 9 cars, 1 bus, 3 trucks, 105.1ms
video 1/1 (2/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 9 cars, 1 bus, 3 trucks, 25.5ms
video 1/1 (3/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 9 cars, 1 bus, 3 trucks, 25.5ms
video 1/1 (4/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 10 cars, 1 bus, 3 trucks, 25.5ms
video 1/1 (5/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 3 trucks, 20.9ms
video 1/1 (6/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 26.0ms
video 1/1 (7/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 3 trucks, 26.1ms
video 1/1 (8/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 1 truck, 24.9ms
video 1/1 (9/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 1 truck, 27.5ms
video 1/1 (10/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 2 trucks, 20.8ms
video 1/1 (11/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 2 trucks, 24.2ms
video 1/1 (12/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 10 cars, 2 buss, 2 trucks, 24.9ms
video 1/1 (13/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 2 trucks, 24.0ms
video 1/1 (14/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 2 trucks, 22.4ms
video 1/1 (15/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 2 trucks, 22.3ms
video 1/1 (16/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 4 trucks, 26.0ms
video 1/1 (17/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 2 trucks, 24.1ms
video 1/1 (18/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 28.6ms
video 1/1 (19/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 31.5ms
video 1/1 (20/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 30.1ms
video 1/1 (21/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 26.5ms
video 1/1 (22/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 25.5ms
video 1/1 (23/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 12 cars, 1 bus, 3 trucks, 31.3ms
video 1/1 (24/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 2 trucks, 29.1ms
video 1/1 (25/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 2 trucks, 26.0ms
video 1/1 (26/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 2 trucks, 27.7ms
video 1/1 (27/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 11 cars, 1 bus, 2 trucks, 28.9ms
video 1/1 (28/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 10 cars, 1 bus, 2 trucks, 32.6ms
video 1/1 (29/942) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/test1.mp4: 384x640 10 cars, 1 bus, 2 trucks, 32.0ms
```

Hình 3-6: Xác định đếm xe mỗi khung hình trong video thực nghiệm từ camera đường bộ.

Kết quả cho thấy đoạn video này chứa 942 frames hình ảnh được trích xuất, mỗi frames chứa thông tin về các loại phương tiện và số lượng mỗi phương tiện trong đó và tốc độ xử lý từng frame.

3.4.2 Kết quả mô phỏng trên video thực nghiệm thực tế ghi lại từ camera điện thoại cá nhân

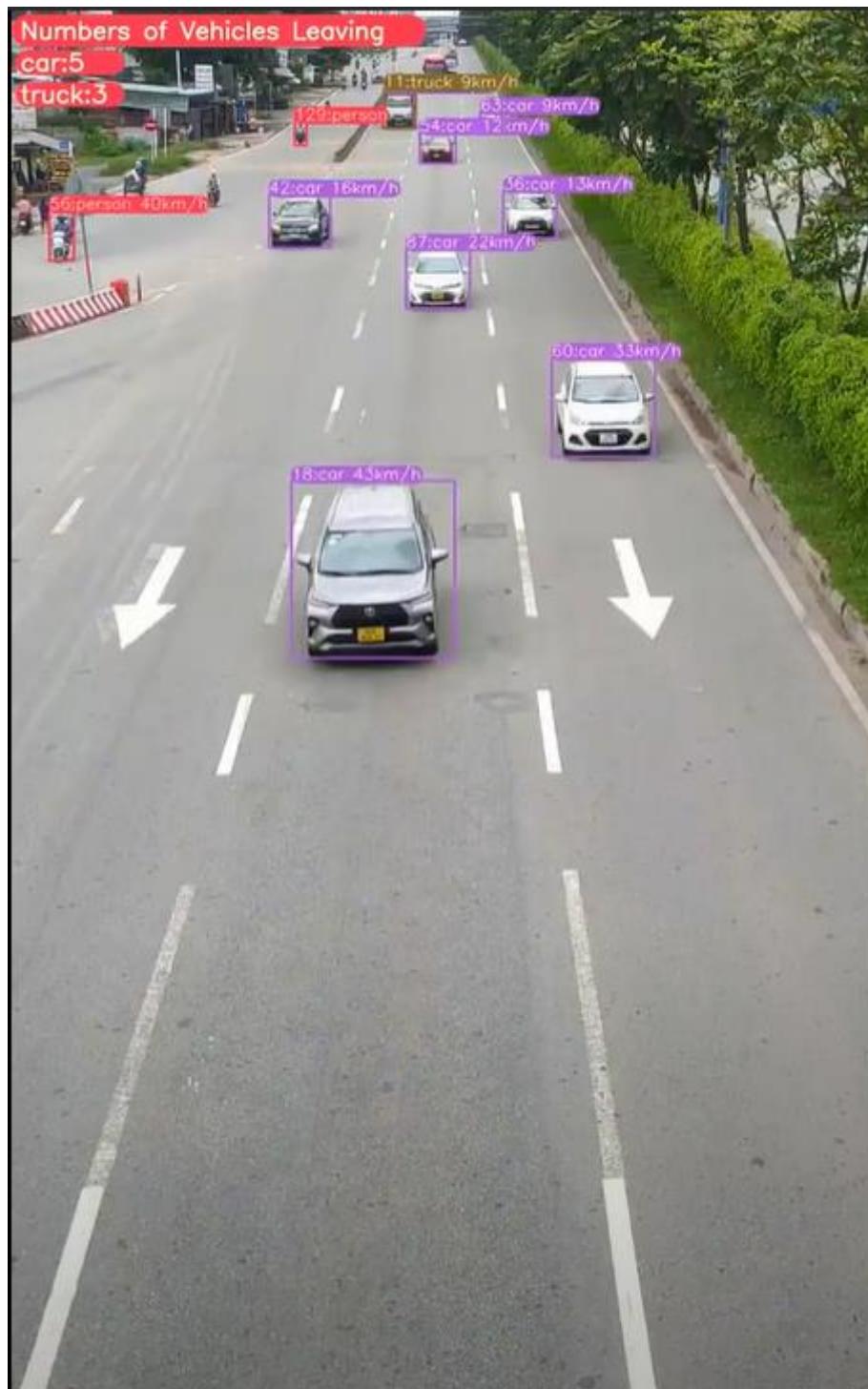
Đối với việc thử nghiệm, em đã tìm kiếm video thực tế từ nhiều nguồn khác nhau và trực tiếp đi quay video trong các làn đường và với thời gian khác nhau, được ghi lại bằng một camera duy nhất được đặt ở độ cao 4.75 mét trên cầu bộ hành. Các video được ghi lại ở ba làn đường khác nhau, với tốc độ thực trên mặt đất.

```
YOLOv8m summary: 218 layers, 25886080 parameters, 0 gradients, 78.9 GFLOPS
video 1/1 (1/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 2 motorcycles, 1 bus, 4 trucks, 68.7ms
video 1/1 (2/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 2 persons, 4 cars, 2 motorcycles, 4 trucks, 25.5ms
video 1/1 (3/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 1 motorcycle, 4 trucks, 25.4ms
video 1/1 (4/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 1 motorcycle, 4 trucks, 25.4ms
video 1/1 (5/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 1 person, 4 cars, 1 motorcycle, 3 trucks, 24.4ms
video 1/1 (6/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 1 motorcycle, 3 trucks, 29.6ms
video 1/1 (7/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 2 persons, 5 cars, 3 trucks, 24.4ms
video 1/1 (8/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 3 trucks, 28.4ms
video 1/1 (9/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 5 trucks, 24.3ms
video 1/1 (10/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 1 motorcycle, 5 trucks, 29.0ms
video 1/1 (11/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 5 cars, 3 trucks, 24.4ms
video 1/1 (12/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 5 persons, 6 cars, 4 trucks, 24.4ms
video 1/1 (13/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 5 persons, 6 cars, 2 motorcycles, 5 trucks, 24.4ms
video 1/1 (14/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 2 persons, 5 cars, 3 motorcycles, 5 trucks, 24.4ms
video 1/1 (15/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 5 cars, 3 motorcycles, 5 trucks, 26.7ms
video 1/1 (16/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 5 cars, 2 motorcycles, 3 trucks, 24.4ms
video 1/1 (17/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 2 motorcycles, 4 trucks, 24.4ms
video 1/1 (18/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 2 motorcycles, 3 trucks, 24.4ms
video 1/1 (19/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 2 motorcycles, 3 trucks, 24.4ms
video 1/1 (20/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 5 persons, 6 cars, 2 motorcycles, 5 trucks, 24.4ms
video 1/1 (21/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 2 persons, 5 cars, 1 motorcycle, 3 trucks, 24.4ms
video 1/1 (22/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 1 motorcycle, 4 trucks, 24.4ms
video 1/1 (23/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 6 cars, 2 motorcycles, 4 trucks, 24.4ms
video 1/1 (24/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 6 cars, 2 motorcycles, 4 trucks, 24.7ms
video 1/1 (25/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 1 motorcycle, 4 trucks, 23.9ms
video 1/1 (26/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 4 cars, 3 trucks, 23.9ms
video 1/1 (27/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 4 cars, 2 motorcycles, 3 trucks, 21.6ms
video 1/1 (28/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 4 cars, 2 motorcycles, 4 trucks, 21.6ms
video 1/1 (29/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 5 persons, 7 cars, 2 motorcycles, 3 trucks, 21.6ms
video 1/1 (30/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 6 cars, 1 motorcycle, 3 trucks, 29.4ms
video 1/1 (31/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 6 cars, 3 trucks, 21.7ms
video 1/1 (32/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 5 cars, 3 trucks, 21.7ms
video 1/1 (33/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 6 cars, 3 trucks, 21.6ms
video 1/1 (34/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 3 persons, 6 cars, 3 trucks, 21.7ms
video 1/1 (35/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 7 cars, 3 trucks, 21.7ms
video 1/1 (36/1083) /content/drive/MyDrive/testing_video_4k/VID_20230730134322.mp4: 640x384 4 persons, 6 cars, 1 motorcycle, 4 trucks, 30.7ms
```

Hình 3-7: Xác định đếm xe mỗi khung hình ở video từ camera cá nhân.

Kết quả cho thấy đoạn video này chứa 1083 frames hình ảnh được trích xuất, mỗi frames chứa thông tin về các loại phương tiện và số lượng mỗi phương tiện trong đó và tốc độ xử lý từng frame.

Dưới đây là kết quả được cắt từ video thực nghiệm thực tế được ghi lại từ camera điện thoại cá nhân:



Hình 3-8: Mô phỏng trên video thực nghiệm thực tế ghi lại từ camera cá nhân.

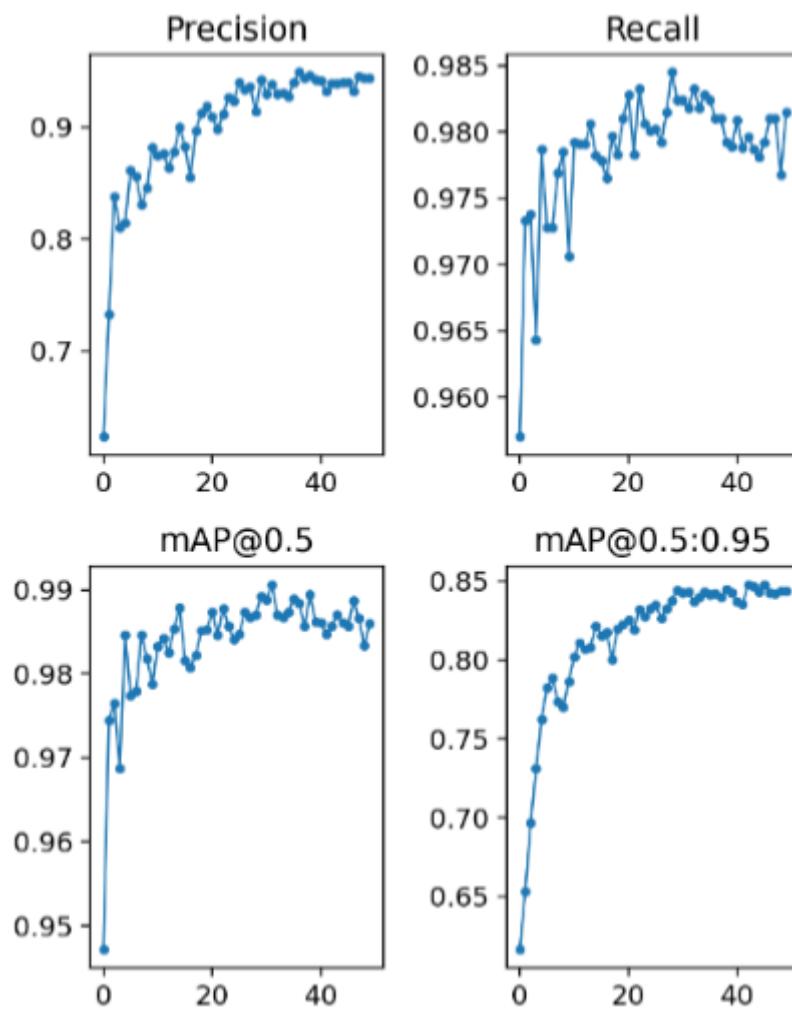


Hình 3-9: Mô phỏng trên video thực nghiệm thực tế ghi lại từ camera cá nhân.

Trong video từ điện thoại cá nhân thì chất lượng video không quá tốt dù đã quay ở chất lượng cao, về độ cõ định thì có thể có rung lắc nhẹ do yếu tố ngoại cảnh tác động như độ run tay hay gió mạnh và độ rộng không quá tốt do giới hạn độ cao và giới hạn tầm nhìn của camera, nên kết quả đầu ra về đo tốc độ không khả quan và

có độ chính xác không cao, còn về đếm lưu lượng xe thì do chỉ quay được đường xe một chiều, kết quả thể hiện rõ đầu ra của xe. Mỗi xe được gắn một mã ID khác nhau và mỗi loại xe được cho một màu khác nhau để thuận tiện cho việc đếm lưu lượng và phân biệt loại xe. Về phần phân loại và đếm lưu lượng xe thì mô hình chạy khá chính xác với ngoại cảnh trên.

3.5 Đánh giá chất lượng model



Hình 3-10: Các thông số đánh giá của mô hình.

Precision (Độ chính xác): Là tỷ lệ giữa số điểm được dự đoán là tích cực và thực sự là tích cực trên tổng số điểm dự đoán là tích cực. Trong ngữ cảnh phát hiện đối tượng, nếu mô hình dự đoán có 10 đối tượng và 9 trong số đó là chính xác, thì độ chính xác là 9/10 hay 0.9. Có thể nhận thấy mô hình có Precision ngày càng cao khi

số lần training tăng trong khoảng 0.8-0.95 cho nên về phát hiện đối tượng có độ chính xác khá cao.

Recall (Độ đầy đủ): Là tỷ lệ giữa số điểm được dự đoán là tích cực và thực sự là tích cực trên tổng số điểm thực sự là tích cực. Nếu có 15 đối tượng thực sự và mô hình dự đoán chính xác 9 trong số đó, thì độ đầy đủ là $9/15$ hay 0.6. Có thể nhận thấy mô hình có Recall ngày càng cao và rất cao khi số lần training tăng trong khoảng 0.975-0.9855 cho nên về dự đoán đối tượng có độ chính xác cao.

mAP (Mean Average Precision - Trung bình Độ chính xác trung bình): mAP cung cấp một số điểm duy nhất để cân bằng giữa Precision (Độ chính xác) và Recall (Độ đầy đủ) của mô hình qua tất cả các ngưỡng. Điều này được thực hiện bằng cách tính toán độ chính xác trung bình (AP) cho mỗi lớp và sau đó lấy trung bình cho các điểm số đó. map@0.5 (mean Average Precision at IOU 0.5) và map@0.5:0.95 liên quan đến những ngưỡng này:

Chỉ số mAP@0.5 có giá trị từ 0.97 đến 0.99. mAP là chỉ số đo lường hiệu suất tổng thể của mô hình phát hiện đối tượng dựa trên độ chính xác và recall ở nhiều ngưỡng IOU (Intersection Over Union). mAP@0.5 nghĩa là ngưỡng IOU được sử dụng để đo lường là 0.5. Giá trị mAP cao từ 0.97 đến 0.99 ở ngưỡng IOU 0.5 cho thấy mô hình có hiệu suất rất tốt và độ chính xác cao khi phát hiện đối tượng.

Giá trị ngưỡng mAP@0.5:0.95 trong khoảng 0.75 đến 0.85 là một chỉ số rất tốt, cho thấy mô hình phát hiện đối tượng hoạt động hiệu quả và chính xác trên nhiều ngưỡng IOU khác nhau. Điều này có nghĩa là mô hình không chỉ làm tốt việc phát hiện đối tượng với sự trùng khớp lỏng lẻo (IOU thấp như 0.5) mà còn chính xác ở các mức độ trùng khớp chặt chẽ hơn (lên đến IOU 0.95).

3.6 Kết luận chương 3

Thiết lập phương pháp thực ghi hình thực nghiệm và kết hợp mô hình YOLOv8 để tiến hành đo tốc độ xe. Quá trình ghi hình trong điều kiện khá thuận lợi với thời tiết ổn định và độ sáng cao giúp cho kết quả đạt được khá tốt.

Về phần đo đặc thực nghiệm, kết quả đo đặc có khả năng lặp lại được thí nghiệm trên nhiều video với độ dài ngắn khác nhau từ vài giây đến vài phút.

Trong kết quả có thể hiện ra từng loại xe trong từng khung hình khác nhau trong video. Hơn nữa, về nhận diện các phương tiện và đếm phương tiện vào ra trong khung hình tương đối chính xác. Tuy nhiên còn về phần đo tốc độ phụ thuộc rất nhiều vào góc máy và yếu tố ngoại cảnh và không có chứng thực cho nêu độ chính xác không quá cao.

CHƯƠNG 4. KẾT LUẬN

4.1 Tóm tắt và kết luận chung

Trong đồ án này, chủ yếu quan tâm đến bài toán ước tính tốc độ của phương tiện. Hệ thống đề xuất dựa trên việc chọn lọc và theo dõi các đặc điểm độc đáo bên trong mỗi phương tiện để ước tính tốc độ.

Mô hình đo tốc độ dựa trên video được trình bày cung cấp hàm mật độ xác suất của tốc độ của một phương tiện di chuyển dựa trên vector mô hình chuyển động của nó. Trong công việc này, phương pháp đề xuất đã được triển khai bằng cách sử dụng bốn đường chạm xâm nhập, và các khung hình đầu vào được chụp đồng thời bằng một máy ảnh cầm tay ngoài kệ hay một điện thoại thông minh với tốc độ khung hình là 30 fps.

Ngoài ra, tốc độ thực tế của các phương tiện nằm trong phạm vi hàm mật độ xác suất được ước tính, làm tăng đáng kể sự tin tưởng vào mô hình.

Kết quả thử nghiệm đã chứng minh rằng tốc độ phát hiện và giới hạn tốc độ có mối quan hệ cao. Những mối quan hệ này xác nhận mô hình đề xuất vì hầu hết các phương tiện nặng (như xe buýt, xe tải và rơ-moóc) được trang bị bộ điều khiển tốc độ buộc chúng phải tuân theo giới hạn tốc độ. Có thể kết luận rằng mô hình dựa trên video đề xuất hoạt động với độ chính xác tương đối và tính ổn định khi đo tốc độ của các phương tiện đi qua.

4.2 Hướng phát triển

Sau khi hoàn thành đề tài "Xây dựng mô hình học máy để nhận diện và đo lường tốc độ của các nhóm ô tô" và đạt được những kết quả nhất định, em đã suy nghĩ và đặt ra thêm các hướng phát triển cho hệ thống để có những hoàn thiện tốt hơn cho các ứng dụng vào thực tế.

Hướng phát triển của đề tài có thể bao gồm:

- Phát triển Mô Hình Phức Tạp Hơn:

Nghiên cứu và ứng dụng các mô hình học sâu hiện đại hơn như các biến thể mới của YOLO, Fast R-CNN hoặc EfficientDet để tăng độ chính xác và tốc độ nhận diện.

Tối ưu hóa mô hình với việc sử dụng transfer learning từ các mô hình được train trên dữ liệu lớn như ImageNet.

- Xử Lý Trong Điều Kiện Khác Nhau:

Phát triển mô hình có khả năng nhận diện và đo lường tốc độ xe trong mọi điều kiện thời tiết và ánh sáng.

Xử lý hiện tượng đôi tượng bị che khuất hoặc trường hợp nhiều xe di chuyển gần nhau.

- Tích Hợp Với Hệ Thống IoT:

Kết nối với các thiết bị cảm biến thông minh trên đường để thu thập dữ liệu thời gian thực và cung cấp thông tin về tốc độ và vị trí xe.

- Phân Loại Loại Xe:

Mở rộng mô hình để không chỉ nhận diện và đo tốc độ mà còn phân loại các loại xe như xe con, xe tải, xe bus, v.v.

- Tích hợp hệ thống cảnh báo:

Dựa trên dữ liệu đo lường, phát triển các tính năng cảnh báo khi xe vượt quá tốc độ cho phép hoặc khi có nhóm xe cơ giới di chuyển bất thường.

- Mở rộng dữ liệu đào tạo:

Thu thập và sử dụng nhiều dữ liệu đào tạo hơn từ các nguồn và môi trường khác nhau, cải thiện độ chính xác và khả năng tổng quát hóa của mô hình.

- Hợp tác với cơ quan quản lý giao thông:

Ứng dụng mô hình vào các hệ thống giám sát giao thông thực tế, giúp cải thiện an toàn và hiệu quả giao thông.

- Phát triển ứng dụng trực quan:

Xây dựng giao diện người dùng trực quan giúp hiển thị thông tin nhận diện và tốc độ xe một cách rõ ràng và trực quan.

Khi phát triển hướng đi mới, quan trọng là cần phải đánh giá lại hiệu suất của mô hình và đảm bảo rằng các tính năng mới không làm giảm độ chính xác hoặc ảnh hưởng đến hiệu suất của hệ thống.

PHỤ LỤC

A.1 Code chương trình giao tiếp colab

▼ Moving to the Required Directory

```
[ ] %cd /content/YOLOv8-DeepSORT-Object-Tracking/ultralytics/yolo/v8/detect
[Errno 2] No such file or directory: '/content/YOLOv8-DeepSORT-Object-Tracking/ultralytics/yolo/v8/detect'
/content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking
```

▼ Lets Run the Script

```
[ ] !gdown "https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8m.pt"
Downloading...
From: https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8m.pt
To: /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/yolov8m.pt
100% 52.1M/52.1M [00:00<00:00, 69.9MB/s]

✓ [ ] !python predict.py model=yolov8m.pt source="/content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/carsVideo.mp4"
[2023-08-06 05:34:17,514][root.tracker][INFO] - Loading weights from deep_sort_pytorch/deep_sort/deep/checkpoint/ckpt.t7... Done!
Config: {'task': 'detect', 'mode': 'train', 'model': 'yolov8m.pt', 'data': None, 'epochs': 100, 'patience': 50, 'batch': 16, 'imgsz': 640, 'save': True, 'cache': False, 'device': 'cuda:0'}
Image size: 640
2023-08-06 05:34:17.977382: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-08-06 05:34:18.948678: E tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Ultralytics YOLOv8 0.3 🚀 Python-3.10.12+torch-2.0.1+cu118+CUDA10 (Tesla T4, 15102MiB)
Fusing layers...
YOLOv8m summary: 218 layers, 25886080 parameters, 0 gradients, 78.0 GFLOPS
video 1/1 (1/2704) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/carsVideo.mp4: 384x640 9 cars, 1 umbrella, 110.0ms
video 1/1 (2/2704) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/carsVideo.mp4: 384x640 9 cars, 1 umbrella, 26.0ms
video 1/1 (3/2704) /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking/carsVideo.mp4: 384x640 9 cars, 1 umbrella, 26.0ms
```

▼ Before Running the Script Please Make Sure you Select the Run Time as GPU

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[6] cd /content/drive/MyDrive
/content/drive/MyDrive

[7] %cd /content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking
/content/drive/MyDrive/YOLOv8-DeepSORT-Object-Tracking
```

▼ Install all the Dependencies

```
[8] !pip install -e '.[dev]'
Requirement already satisfied: omegaconf>=2.4,>=2.4.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics[version=>1.2.0->ultralytics==0.0.3]) (2.4.0)
Requirement already satisfied: antlr4-python3-runtime==4.9.* in /usr/local/lib/python3.10/dist-packages (from hydra-core>=1.2.0->ultralytics==0.0.3) (4.9.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from hydra-core>=1.2.0->ultralytics==0.0.3) (23.1)
Requirement already satisfied: contourpy>1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics==0.0.3) (1.1.0)
Requirement already satisfied: cycler>0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics==0.0.3) (0.11.0)
Requirement already satisfied: fonttools>4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics==0.0.3) (4.41.1)
Requirement already satisfied: kiwisolver>1.0.* in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics==0.0.3) (1.4.4)
Requirement already satisfied: pyparsing>2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics==0.0.3) (3.1.0)
Requirement already satisfied: python-dateutil>2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->ultralytics==0.0.3) (2.8.2)
Requirement already satisfied: pytz>2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics==0.0.3) (2022.7.1)
Requirement already satisfied: urllib3>1.27,>=1.27.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics==0.0.3) (1.26.16)
```

A.2 Code chương trình xử lý dữ liệu

```
# Ultralytics YOLO 🚀, GPL-3.0 license

import hydra
import torch
import argparse
import time
from pathlib import Path
import math
import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random
from ultralytics.yolo.engine.predictor import BasePredictor
from ultralytics.yolo.utils import DEFAULT_CONFIG, ROOT, ops
from ultralytics.yolo.utils.checks import check_imgs
from ultralytics.yolo.utils.plotting import Annotator, colors,
save_one_box

import cv2
from deep_sort_pytorch.utils.parser import get_config
from deep_sort_pytorch.deep_sort import DeepSort
from collections import deque
import numpy as np
palette = (2 ** 11 - 1, 2 ** 15 - 1, 2 ** 20 - 1)
data_deque = {}

deepsort = None

object_counter = {}

object_counter1 = {}

line = [(50, 500), (1050, 500)]
speed_line_queue = {}
def estimatespeed(Location1, Location2):
    #Euclidean Distance Formula
    d_pixel = math.sqrt(math.pow(Location2[0] - Location1[0], 2) +
+ math.pow(Location2[1] - Location1[1], 2))
    # defining thr pixels per meter
    ppm = 8
    d_meters = d_pixel/ppm
    time_constant = 30*3.6
    #distance = speed/time
    speed = d_meters * time_constant

    return int(speed)
```

```

def init_tracker():
    global deepsort
    cfg_deep = get_config()

    cfg_deep.merge_from_file("deep_sort_pytorch/configs/deep_sort.yaml")

    deepsort= DeepSort(cfg_deep.DEEPSORT.REID_CKPT,
                        max_dist=cfg_deep.DEEPSORT.MAX_DIST,
                        min_confidence=cfg_deep.DEEPSORT.MIN_CONFIDENCE,
                        n_init=cfg_deep.DEEPSORT.N_INIT,
                        nn_budget=cfg_deep.DEEPSORT.NN_BUDGET,
                        use_cuda=True)
#####
#####

def xyxy_to_xywh(*xyxy):
    """ Calculates the relative bounding box from absolute pixel values. """
    bbox_left = min([xyxy[0].item(), xyxy[2].item()])
    bbox_top = min([xyxy[1].item(), xyxy[3].item()])
    bbox_w = abs(xyxy[0].item() - xyxy[2].item())
    bbox_h = abs(xyxy[1].item() - xyxy[3].item())
    x_c = (bbox_left + bbox_w / 2)
    y_c = (bbox_top + bbox_h / 2)
    w = bbox_w
    h = bbox_h
    return x_c, y_c, w, h

def xyxy_to_tlwh(bbox_xyxy):
    tlwh_bboxes = []
    for i, box in enumerate(bbox_xyxy):
        x1, y1, x2, y2 = [int(i) for i in box]
        top = x1
        left = y1
        w = int(x2 - x1)
        h = int(y2 - y1)
        tlwh_obj = [top, left, w, h]
        tlwh_bboxes.append(tlwh_obj)
    return tlwh_bboxes

def compute_color_for_labels(label):
    """
    Simple function that adds fixed color depending on the class
    """

```

```
if label == 0: #person
    color = (85,45,255)
elif label == 2: # Car
    color = (222,82,175)
elif label == 3: # Motobike
    color = (0, 204, 255)
elif label == 5: # Bus
    color = (0, 149, 255)
else:
    color = [int((p * (label ** 2 - label + 1)) % 255) for p
in palette]
    return tuple(color)

def draw_border(img, pt1, pt2, color, thickness, r, d):
    x1,y1 = pt1
    x2,y2 = pt2
    # Top left
    cv2.line(img, (x1 + r, y1), (x1 + r + d, y1), color,
thickness)
    cv2.line(img, (x1, y1 + r), (x1, y1 + r + d), color,
thickness)
    cv2.ellipse(img, (x1 + r, y1 + r), (r, r), 180, 0, 90, color,
thickness)
    # Top right
    cv2.line(img, (x2 - r, y1), (x2 - r - d, y1), color,
thickness)
    cv2.line(img, (x2, y1 + r), (x2, y1 + r + d), color,
thickness)
    cv2.ellipse(img, (x2 - r, y1 + r), (r, r), 270, 0, 90, color,
thickness)
    # Bottom left
    cv2.line(img, (x1 + r, y2), (x1 + r + d, y2), color,
thickness)
    cv2.line(img, (x1, y2 - r), (x1, y2 - r - d), color,
thickness)
    cv2.ellipse(img, (x1 + r, y2 - r), (r, r), 90, 0, 90, color,
thickness)
    # Bottom right
    cv2.line(img, (x2 - r, y2), (x2 - r - d, y2), color,
thickness)
    cv2.line(img, (x2, y2 - r), (x2, y2 - r - d), color,
thickness)
    cv2.ellipse(img, (x2 - r, y2 - r), (r, r), 0, 0, 90, color,
thickness)

    cv2.rectangle(img, (x1 + r, y1), (x2 - r, y2), color, -1,
cv2.LINE_AA)
```

```
        cv2.rectangle(img, (x1, y1 + r), (x2, y2 - r - d), color, -1,
cv2.LINE_AA)

        cv2.circle(img, (x1 +r, y1+r), 2, color, 12)
        cv2.circle(img, (x2 -r, y1+r), 2, color, 12)
        cv2.circle(img, (x1 +r, y2-r), 2, color, 12)
        cv2.circle(img, (x2 -r, y2-r), 2, color, 12)

    return img

def UI_box(x, img, color=None, label=None, line_thickness=None):
    # Plots one bounding box on image img
    tl = line_thickness or round(0.002 * (img.shape[0] +
img.shape[1]) / 2) + 1 # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    cv2.rectangle(img, c1, c2, color, thickness=tl,
lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1) # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3,
thickness=tf)[0]

        img = draw_border(img, (c1[0], c1[1] - t_size[1] - 3),
(c1[0] + t_size[0], c1[1]+3), color, 1, 8, 2)

        cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3,
[225, 255, 255], thickness=tf, lineType=cv2.LINE_AA)

def intersect(A,B,C,D):
    return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)

def ccw(A,B,C):
    return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])

def get_direction(point1, point2):
    direction_str = ""

    # calculate y axis direction
    if point1[1] > point2[1]:
        direction_str += "South"
    elif point1[1] < point2[1]:
        direction_str += "North"
    else:
        direction_str += ""
```

```

# calculate x axis direction
if point1[0] > point2[0]:
    direction_str += "East"
elif point1[0] < point2[0]:
    direction_str += "West"
else:
    direction_str += ""

return direction_str
def draw_boxes(img, bbox, names, object_id, identities=None,
offset=(0, 0)):
    # cv2.line(img, line[0], line[1], (46,162,112), 3)

    height, width, _ = img.shape
    # remove tracked point from buffer if object is lost
    for key in list(data_deque):
        if key not in identities:
            data_deque.pop(key)

    for i, box in enumerate(bbox):
        x1, y1, x2, y2 = [int(i) for i in box]
        x1 += offset[0]
        x2 += offset[0]
        y1 += offset[1]
        y2 += offset[1]

        # code to find center of bottom edge
        center = (int((x2+x1)/ 2), int((y2+y1)/ 2))

        # get ID of object
        id = int(identities[i]) if identities is not None else 0

        # create new buffer for new object
        if id not in data_deque:
            data_deque[id] = deque(maxlen= 64)
            speed_line_queue[id] = []
        color = compute_color_for_labels(object_id[i])
        obj_name = names[object_id[i]]
        label = '{}{:d}'.format("", id) + ":" + '%s' % (obj_name)

        # add center to buffer
        data_deque[id].appendleft(center)
        if len(data_deque[id]) >= 2:
            direction = get_direction(data_deque[id][0],
data_deque[id][1])
            object_speed = estimatespeed(data_deque[id][1],
data_deque[id][0])
            speed_line_queue[id].append(object_speed)

```

```

        if intersect(data_deque[id][0], data_deque[id][1],
line[0], line[1]):
            # cv2.line(img, line[0], line[1], (255, 255, 255),
3)
            if "South" in direction:
                if obj_name not in object_counter:
                    object_counter[obj_name] = 1
                else:
                    object_counter[obj_name] += 1
            if "North" in direction:
                if obj_name not in object_counter1:
                    object_counter1[obj_name] = 1
                else:
                    object_counter1[obj_name] += 1

        try:
            label = label + " " +
str(sum(speed_line_queue[id])//len(speed_line_queue[id])) +
"km/h"
        except:
            pass
        UI_box(box, img, label=label, color=color,
line_thickness=2)
        # draw trail
        for i in range(1, len(data_deque[id])):
            # check if on buffer value is none
            if data_deque[id][i - 1] is None or data_deque[id][i]
is None:
                continue
            # generate dynamic thickness of trails
            thickness = int(np.sqrt(64 / float(i + i)) * 1.5)
            # draw trails
            # cv2.line(img, data_deque[id][i - 1],
data_deque[id][i], color, thickness)

        #4. Display Count in top right corner
        for idx, (key, value) in
enumerate(object_counter1.items()):
            cnt_str = str(key) + ":" + str(value)
            cv2.line(img, (width - 500, 25), (width, 25),
[85, 45, 255], 40)
            cv2.putText(img, f'Number of Vehicles Entering',
(width - 500, 35), 0, 1, [225, 255, 255], thickness=2,
lineType=cv2.LINE_AA)
            cv2.line(img, (width - 150, 65 + (idx*40)), (width,
65 + (idx*40)), [85, 45, 255], 30)
    
```

```
        cv2.putText(img, cnt_str, (width - 150, 75 +
(idx*40)), 0, 1, [255, 255, 255], thickness = 2, lineType =
cv2.LINE_AA)

        for idx, (key, value) in
enumerate(object_counter.items()):
            cnt_str1 = str(key) + ":" +str(value)
            cv2.line(img, (20,25), (500,25), [85,45,255], 40)
            cv2.putText(img, f'Numbers of Vehicles Leaving', (11,
35), 0, 1, [225, 255, 255], thickness=2, lineType=cv2.LINE_AA)

            cv2.line(img, (20,65+ (idx*40)), (127,65+ (idx*40)),
[85,45,255], 30)
            cv2.putText(img, cnt_str1, (11, 75+ (idx*40)), 0, 1,
[225, 255, 255], thickness=2, lineType=cv2.LINE_AA)

    return img

class DetectionPredictor(BasePredictor):

    def get_annotator(self, img):
        return Annotator(img,
line_width=self.args.line_thickness,
example=str(self.model.names))

    def preprocess(self, img):
        img = torch.from_numpy(img).to(self.model.device)
        img = img.half() if self.model.fp16 else img.float() #
uint8 to fp16/32
        img /= 255 # 0 - 255 to 0.0 - 1.0
        return img

    def postprocess(self, preds, img, orig_img):
        preds = ops.non_max_suppression(preds,
                                         self.args.conf,
                                         self.args.iou,
agnostic=self.args.agnostic_nms,
max_det=self.args.max_det)

        for i, pred in enumerate(preds):
            shape = orig_img[i].shape if self.webcam else
orig_img.shape
            pred[:, :4] = ops.scale_boxes(img.shape[2:], pred[:, :
4], shape).round()
```

```

        return preds

    def write_results(self, idx, preds, batch):
        p, im, im0 = batch
        all_outputs = []
        log_string = ""
        if len(im.shape) == 3:
            im = im[None] # expand for batch dim
        self.seen += 1
        im0 = im0.copy()
        if self.webcam: # batch_size >= 1
            log_string += f'{idx}: '
            frame = self.dataset.count
        else:
            frame = getattr(self.dataset, 'frame', 0)

        self.data_path = p
        save_path = str(self.save_dir / p.name) # im.jpg
        self.txt_path = str(self.save_dir / 'labels' / p.stem) +
('' if self.dataset.mode == 'image' else f'_{frame}')
        log_string += '%gx%g ' % im.shape[2:] # print string
        self.annotator = self.get_annotator(im0)

        det = preds[idx]
        all_outputs.append(det)
        if len(det) == 0:
            return log_string
        for c in det[:, 5].unique():
            n = (det[:, 5] == c).sum() # detections per class
            log_string += f"{n} {self.model.names[int(c)]}{('s' * (n > 1))}, "
            # write
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain whwh
            xywh_bboxes = []
            confs = []
            oids = []
            outputs = []
            for *xyxy, conf, cls in reversed(det):
                x_c, y_c, bbox_w, bbox_h = xyxy_to_xywh(*xyxy)
                xywh_obj = [x_c, y_c, bbox_w, bbox_h]
                xywh_bboxes.append(xywh_obj)
                confs.append([conf.item()])
                oids.append(int(cls))
            xywhs = torch.Tensor(xywh_bboxes)
            confss = torch.Tensor(confs)

            outputs = deepsort.update(xywhs, confss, oids, im0)

```

```
if len(outputs) > 0:
    bbox_xyxy = outputs[:, :4]
    identities = outputs[:, -2]
    object_id = outputs[:, -1]

    draw_boxes(im0, bbox_xyxy, self.model.names,
object_id, identities)

    return log_string

@hydra.main(version_base=None,
config_path=str(DEFAULT_CONFIG.parent),
config_name=DEFAULT_CONFIG.name)
def predict(cfg):
    init_tracker()
    cfg.model = cfg.model or "yolov8n.pt"
    print("Config: ", cfg)
    print("Image size: ", cfg.imgsz)
    cfg.imgsz = check_imgsz(cfg.imgsz, min_dim=2)  # check image
size
    cfg.source = cfg.source if cfg.source is not None else ROOT /
"assets"
    predictor = DetectionPredictor(cfg)
    predictor()

if __name__ == "__main__":
    predict()
```

TÀI LIỆU THAM KHẢO

- [1] <https://topdev.vn/blog/machine-learning-la-gi/>.
- [2] <https://viblo.asia/p/recurrent-neural-networkphan-1-tong-quan-va-ung-dung-jvElaB4m5kw>.
- [3] <https://viblo.asia/p/gan-series-1-co-ban-ve-gan-trong-deep-learning-bWrZnE4YKxw>.
- [4] <https://learnopencv.com/ultralytics-yolov8/>.
- [5] <https://www.toponseek.com/blogs/google-colab/>.
- [6] D. C. LUVIZON, VEHICLE SPEED ESTIMATION BY LICENSE PLATE DETECTION AND TRACKING, Agosto 2015.
- [7] Saleh Javadi a,*Mattias Dahl b, Mats I. Petterssonb, Vehicle speed measurement model for video-based systems, Published by Elsevier Ltd, 2019.