

BỘ GIÁO DỤC VÀ ĐÀO TẠO  
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH  
KHOA CÔNG NGHỆ THÔNG TIN



NGUYỄN TRƯƠNG THÀNH LONG                    21133053  
NGUYỄN TÂN SƯƠNG                                21133078

Đề tài:  
**XÂY DỰNG HỆ THỐNG PHÁT HIỆN  
VƯỢT ĐÈN ĐỎ THEO THỜI GIAN THỰC**

**KHÓA LUẬN TỐT NGHIỆP  
NGÀNH KỸ THUẬT DỮ LIỆU**

GIÁO VIÊN HƯỚNG DẪN  
**ThS. Lê Thị Minh Châu**

TP. Hồ Chí Minh, tháng 06 năm 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO  
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH  
KHOA CÔNG NGHỆ THÔNG TIN



NGUYỄN TRƯƠNG THÀNH LONG                    21133053  
NGUYỄN TÂN SƯƠNG                                21133078

Đề tài:

**XÂY DỰNG HỆ THỐNG PHÁT HIỆN  
VƯỢT ĐÈN ĐỎ THEO THỜI GIAN THỰC**

**KHÓA LUẬN TỐT NGHIỆP  
NGÀNH KỸ THUẬT DỮ LIỆU**

GIÁO VIÊN HƯỚNG DẪN  
**ThS. Lê Thị Minh Châu**

TP. Hồ Chí Minh, tháng 06 năm 2025

## PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Họ và tên Sinh viên 1: Nguyễn Trương Thành Long      MSSV 1: 21133053

Họ và tên Sinh viên 2: Nguyễn Tân Sương      MSSV 2: 21133078

Ngành: Kỹ thuật dữ liệu

Tên đề tài: XÂY DỰNG HỆ THỐNG PHÁT HIỆN VƯỢT ĐÈN ĐỎ THEO THỜI GIAN THỰC

Họ và tên Giáo viên hướng dẫn: ThS. Lê Thị Minh Châu

### NHẬN XÉT

1. Về nội dung đề tài & khối lượng thực hiện:

.....  
.....  
.....

2. Ưu điểm:

.....  
.....  
.....

3. Khuyết điểm:

.....  
.....  
.....

4. Đề nghị cho bảo vệ hay không?

5. Đánh giá loại:

6. Điểm:

Tp. Hồ Chí Minh, ngày tháng năm 2025

Giáo viên hướng dẫn

(Ký & ghi rõ họ tên)

## PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

Họ và tên Sinh viên 1: Nguyễn Trương Thành Long      MSSV 1: 21133053

Họ và tên Sinh viên 2: Nguyễn Tân Sương      MSSV 2: 21133078

Ngành: Kỹ thuật dữ liệu

Tên đề tài: XÂY DỰNG HỆ THỐNG PHÁT HIỆN VƯỢT ĐÈN ĐỎ THEO THỜI GIAN THỰC

Họ và tên Giáo viên phản biện: TS. Nguyễn Thanh Tuấn

### NHẬN XÉT

1. Về nội dung đề tài & khối lượng thực hiện:

.....  
.....  
.....

2. Ưu điểm:

.....  
.....  
.....

3. Khuyết điểm:

.....  
.....  
.....

4. Đề nghị cho bảo vệ hay không?

5. Đánh giá loại:

6. Điểm:

Tp. Hồ Chí Minh, ngày tháng năm 2025

Giáo viên hướng dẫn

(Ký & ghi rõ họ tên)

## ĐỀ CƯƠNG KHÓA LUẬN TỐT NGHIỆP

Họ và tên Sinh viên 1: Nguyễn Trương Thành Long MSSV 1: 21133053

Họ và tên Sinh viên 2: Nguyễn Tân Sương MSSV 2: 21133078

Thời gian làm tiểu luận chuyên ngành từ: 10/02/2025 – 01/06/2025

Ngành: Kỹ thuật dữ liệu

Tên đề tài: XÂY DỰNG HỆ THỐNG PHÁT HIỆN VƯỢT ĐÈN ĐỎ THEO THỜI GIAN THỰC

Họ và tên Giáo viên hướng dẫn: ThS. Lê Thị Minh Châu

Nhiệm vụ của tiểu luận chuyên ngành:

### 1. Lý thuyết

- Tìm hiểu công nghệ sử dụng: Yolo11, DeepSORT, OpenCV, PostgreSQL, Python.
- Cơ chế và kiến trúc YOLO. So sánh các phiên bản YOLO (v3, v4, v5, v7, v8).  
Ứng dụng YOLO trong giám sát giao thông.
- Nguyên lý hoạt động và cách DeepSORT kết hợp với YOLO để theo dõi phương tiện vượt đèn đỏ.
- Xử lý ảnh với OpenCV:
- Khái niệm và cách PostgreSQL lưu trữ và truy vấn dữ liệu vi phạm.

### 2. Thực hành

- Xây dựng hệ thống phát hiện vượt đèn đỏ theo thời gian thực.
- Sử dụng Yolo11 để nhận diện biển số xe phương tiện giao thông (xe máy, ô tô, xe tải).
- Sử dụng OpenCV để phát hiện vạch dừng và xác định trạng thái đèn tín hiệu giao thông.
- Lưu thông tin vi phạm (ID phương tiện, thời gian, ảnh bằng chứng, biển số xe, loại vi phạm) và truy vấn dữ liệu.

## MỤC LỤC

LỜI CẢM ƠN.....	1
PHÂN CÔNG NHIỆM VỤ .....	2
Chương 1: GIỚI THIỆU.....	4
1.1. TÍNH CẤP THIẾT CỦA ĐỀ TÀI .....	4
1.3. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU .....	5
1.4. BỐ CỤC CỦA BÁO CÁO .....	5
Chương 2: CƠ SỞ LÝ THUYẾT .....	6
2.1. Nhận diện hình ảnh và thị giác máy tính .....	6
2.1.1. Thị giác máy tính .....	6
2.1.2. Tổng quan về nhận diện hình ảnh .....	6
2.1.3. Các phương pháp nhận diện .....	6
2.1.4. Phát hiện đối tượng (Object Detection) .....	8
2.2. YOLO .....	8
2.2.1. Giới thiệu YOLO .....	8
2.2.2. Cơ chế hoạt động và kiến trúc của YOLO.....	9
2.2.3. Các phiên bản YOLO .....	10
2.2.4. Giới thiệu Yolo11 .....	11
2.2.5. Kiến trúc Yolo11 .....	12
2.2.6. Ứng dụng Yolo trong hệ thống phát hiện vượt đèn đỏ .....	13
2.3. OpenCV – Thư viện xử lý ảnh mã nguồn mở.....	13
2.3.1. Giới thiệu về OpenCV .....	13
2.3.2. Các module trong OpenCV .....	13
2.3.3. Ứng dụng OpenCV trong hệ thống phát hiện vi phạm.....	13
2.4. DeepSORT – Thuật toán theo dõi đối tượng.....	14
2.4.1. Giới thiệu về DeepSORT .....	14
2.4.2. Cách hoạt động của DeepSORT .....	14
2.4.3. Ứng dụng trong hệ thống phát hiện vi phạm .....	14
2.5. PostgreSQL – Hệ quản trị cơ sở dữ liệu.....	15
2.5.1. Giới thiệu về PostgreSQL.....	15
2.5.2. Ứng dụng PostgreSQL trong hệ thống .....	15
Chương 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG .....	16
3.1. Kiến trúc hệ thống.....	16
3.2. Quy trình thu thập và xử lý dữ liệu .....	17

3.3. Quy trình hoạt động của hệ thống nhận dạng biển số xe .....	18
3.4. Thiết kế các module phần mềm .....	19
3.5. Thiết kế giao diện người dùng (UI) .....	20
<b>Chương 4: TRIỂN KHAI HỆ THỐNG .....</b>	<b>21</b>
4.1. Kiến trúc hệ thống.....	21
4.2. Công nghệ sử dụng.....	21
4.3. Xây dựng các thành phần chính.....	21
4.3.1. Nhận diện vùng chứa biển số xe của phương tiện vi phạm .....	21
4.3.2. Nhận diện các ký tự trong biển số của phương tiện vi phạm .....	26
4.3.3. Phát hiện đèn tín hiệu giao thông.....	33
4.4. Xây dựng hệ thống nhận diện đèn đỏ theo thời gian thực .....	38
4.4.1. Tổng quan về hệ thống.....	38
4.4.2. Khởi tạo và cấu hình.....	39
4.4.3. Kết nối cơ sở dữ liệu PostgreSQL .....	41
4.4.4. Phát hiện và xử lý vi phạm .....	43
4.4.5. Hiển thị lịch sử vi phạm.....	54
4.4.6. Tạo báo cáo PDF.....	57
4.4.7. Giao diện streamlit.....	58
<b>Chương 5: KẾT QUẢ VÀ ĐÁNH GIÁ .....</b>	<b>59</b>
5.1. Kiểm tra độ chính xác .....	59
5.2. Thực nghiệm thực tế .....	60
<b>Chương 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....</b>	<b>64</b>
6.1. Kết luận .....	64
6.2. Hướng phát triển .....	64
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>66</b>

## PHỤ LỤC HÌNH ẢNH

Hình 2.1. Kiến trúc mạng Yolo trong việc nhận diện các ký tự [6] .....	7
Hình 2.2. Phát hiện các đối tượng (phương tiện giao thông) [12] .....	8
Hình 2.3. Cơ chế hoạt động của Yolo [4] .....	9
Hình 2.4. Các thế hệ, phiên bản Yolo [4] .....	10
Hình 2.5. So sánh phiên bản Yolo11 với các phiên bản cũ [15].....	11
Hình 2.6. Kiến trúc chi tiết của Yolo11 [8] .....	12
Hình 2.7. Cách hoạt động của DeepSORT [6] .....	14
Hình 3.1. Quy trình hoạt động của hệ thống nhận dạng biển số xe .....	19
Hình 4.1. Giao diện Label Studio về đánh nhãn biển số xe .....	22
Hình 4.2. Tọa độ bounding box vị trí của biển số xe .....	22
Hình 4.3. Nội dung mydata_biensoxe.yaml .....	23
Hình 4.4. Nội dung mô hình huấn luyện .....	23
Hình 4.5. Kết quả mô hình nhận diện biển số sau huấn luyện .....	24
Hình 4.6. Chạy thực nghiệm nhận diện biển số ô tô .....	26
Hình 4.7. Chạy thực nghiệm nhận diện biển số xe máy.....	26
Hình 4.8. Biển số ô tô.....	27
Hình 4.9. Giao diện Label Studio về đánh nhãn biển số xe .....	28
Hình 4.10. Tọa độ bounding box vị trí của biển số xe .....	28
Hình 4.11. Nội dung mydata_bienso.yaml .....	29
Hình 4.12. Nội dung mô hình huấn luyện .....	29
Hình 4.13. Kết quả mô hình nhận diện biển số sau huấn luyện .....	31
Hình 4.14. Dự đoán trên kết quả thực tế .....	33
Hình 4.15. Giao diện hệ thống Label Studio về tín hiệu đèn giao thông .....	34
Hình 4.16. Tọa độ bounding box vị trí tín hiệu đèn giao thông .....	34
Hình 4.17. File mydata.yaml .....	34
Hình 4.18. Thông số huấn luyện phát hiện đèn giao thông.....	35
Hình 4.19. Kết quả mô hình nhận diện đèn giao thông sau khi huấn luyện.....	36
Hình 4.20. Kết quả thực nghiệm .....	37
Hình 4.21. Kết quả thực nghiệm .....	38
Hình 4.22. Tạo thư mục lưu ảnh vi phạm.....	39
Hình 4.23. Biến toàn cục .....	40
Hình 4.24. Lưu trữ trạng thái.....	40
Hình 4.25. Tải mô hình Yolo .....	41

Hình 4.26. Kết nối cơ sở dữ liệu PostgreSQL.....	42
Hình 4.27. Hàm saveViolation_to_db() .....	43
Hình 4.28. Hàm point_above_line() .....	43
Hình 4.29. Hàm saveViolation_image() .....	44
Hình 4.30. Khởi tạo và lấy đèn giao thông.....	45
Hình 4.31. Theo dõi vị trí phương tiện.....	46
Hình 4.32. Kiểm tra vượt vạch giữa .....	46
Hình 4.33. Kiểm tra trạng thái dừng.....	47
Hình 4.34. Xác định tuyến đường .....	47
Hình 4.35. Kiểm tra vi phạm .....	48
Hình 4.36. Xử lý lỗi và đóng kết nối .....	48
Hình 4.37. Khởi tạo và phát hiện đèn giao thông.....	49
Hình 4.38. Phát hiện phương tiện.....	50
Hình 4.39. Theo dõi với DeepSort .....	50
Hình 4.40. Cập nhật ngũ cảnh .....	51
Hình 4.41. Dọn dẹp ngũ cảnh cũ .....	51
Hình 4.42. Định nghĩa vạch kẻ đường.....	52
Hình 4.43. Tìm cửa sổ scrcpy .....	52
Hình 4.44. Chụp ảnh từ cửa sổ .....	53
Hình 4.45. Chuyển đổi và giải phóng tài nguyên.....	53
Hình 4.46. Xử lý lỗi.....	54
Hình 4.47. Khởi tạo giao diện lịch sử.....	54
Hình 4.48. Form tìm kiếm .....	55
Hình 4.49. Phân trang.....	55
Hình 4.50. Hiển thị danh sách vi phạm .....	56
Hình 4.51. Xử lý và đóng kết nối .....	56
Hình 4.52. Khởi tạo PDF .....	57
Hình 4.53. Thêm nội dung văn bản .....	57
Hình 4.54. Thêm ảnh bằng chứng .....	58
Hình 4.55. Sidebar điều hướng.....	58
Hình 5.1. Giao diện taskbar để chọn chức năng.....	60
Hình 5.2. Giao diện thông tin camera giao thông tại Thành phố Điện Biên Phủ.....	60
Hình 5.3. Hình ảnh camera được hiển thị lên giao diện .....	60
Hình 5.4. Giao diện lịch sử vi phạm.....	61

Hình 5.5. Giao diện hiển thị lịch sử vi phạm chi tiết .....	61
Hình 5.6. Báo cáo vi phạm .....	62

## **LỜI CẢM ƠN**

Nhóm chúng em xin gửi lời tri ân sâu sắc đến Khoa Công nghệ Thông tin - Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh, nơi đã tạo điều kiện thuận lợi để chúng em học tập, rèn luyện và phát triển kỹ năng nhằm thực hiện thành công đề tài này.

Đặc biệt, nhóm xin bày tỏ lòng biết ơn chân thành đến cô Lê Thị Minh Châu, người đã tận tình hướng dẫn và dành cho chúng em những lời khuyên quý báu trong suốt quá trình thực hiện đề tài. Sự nhiệt tình, tận tâm và hỗ trợ của cô là nguồn động lực to lớn giúp chúng em hoàn thành đề tài một cách tốt nhất.

Bên cạnh đó, những kiến thức chuyên môn mà chúng em đã được học tại trường, cùng với kinh nghiệm thực tế thu nhận được qua các hoạt động học tập và thực tập, đã giúp chúng em ứng dụng hiệu quả vào việc triển khai đề tài này.

Dẫu vậy, do thời gian nghiên cứu hạn chế và kiến thức còn nhiều thiếu sót, chúng em nhận thức rằng đề tài vẫn chưa thực sự hoàn thiện. Chúng em rất mong nhận được những ý kiến đóng góp từ quý thầy cô để cải thiện và phát triển hơn nữa trong tương lai. Chúng em xin chân thành cảm ơn!

**Nhóm thực hiện**

Nguyễn Trương Thành Long – 21133053

Nguyễn Tân Sương – 21133078

## PHÂN CÔNG NHIỆM VỤ

Thời gian	Công việc	Sinh viên thực hiện
<b>10/02/2025 - 16/02/2025</b>	- Nghiên cứu Yolo11, DeepSort, OpenCV, PostgreSQL, Streamlit. - Tìm hiểu tích hợp YOLO-DeepSort.	Nguyễn Tân Sương
	- Nghiên cứu PostgreSQL và chụp ảnh scrcpy.	Nguyễn Trương Thành Long
<b>17/02/2025 - 23/02/2025</b>	- Cài đặt môi trường: Python, OpenCV, Yolo11, DeepSort.	Nguyễn Trương Thành Long
	- Thu thập dữ liệu video, thiết lập phát hiện YOLO.	Nguyễn Tân Sương
<b>24/02/2025 - 09/03/2025</b>	- Xây dựng module phát hiện vạch và DeepSort theo dõi.	Nguyễn Trương Thành Long
	- Thiết lập bảng violations PostgreSQL, hàm detect_traffic_light_and_vehicles.	Nguyễn Tân Sương
<b>10/03/2025 - 23/03/2025</b>	- Triển khai detectViolation, tích hợp YOLO-DeepSort.	Nguyễn Tân Sương
	- Xây dựng captureWindow, saveViolationImage, kiểm thử module.	Nguyễn Trương Thành Long
<b>24/03/2025 - 06/04/2025</b>	- Xây dựng giao diện showCamera cho video trực tiếp.	Nguyễn Trương Thành Long
	- Xây dựng showHistory và createPdfReport cho báo cáo.	Nguyễn Tân Sương
<b>07/04/2025 - 20/04/2025</b>	- Sửa lỗi, thử nghiệm thực tế, tối ưu hiệu suất.	Nguyễn Tân Sương
	- Tích hợp nhận diện biển số, viết báo cáo sơ bộ.	Nguyễn Trương Thành Long
<b>21/04/2025 - 04/05/2025</b>	- Viết báo cáo chi tiết, chuẩn bị slide.	Nguyễn Trương Thành Long

	- Tối ưu giao diện Streamlit, tổng duyệt.	Nguyễn Tân Sương
<b>05/05/2025 - 18/05/2025</b>	- Tối ưu YOLO	Nguyễn Tân Sương
	- Hoàn thiện và đánh giá hệ thống.	Nguyễn Trương Thành Long
<b>19/05/2025 - 01/06/2025</b>	- Hoàn thiện báo cáo.	Nguyễn Trương Thành Long
	- Đánh giá tổng thể, lưu trữ mã nguồn, lập kế hoạch phát triển.	Nguyễn Tân Sương

## Chương 1: GIỚI THIỆU

### 1.1. TÍNH CẤP THIẾT CỦA ĐÈN ĐỎ

Trong những năm gần đây, cùng với sự phát triển mạnh mẽ của đô thị hóa, tình trạng ùn tắc và vi phạm giao thông ngày càng trở thành một vấn đề nghiêm trọng tại nhiều thành phố lớn. Một trong những hành vi vi phạm phổ biến nhất là vượt đèn đỏ, gây nguy hiểm không chỉ cho chính người vi phạm mà còn ảnh hưởng trực tiếp đến những người tham gia giao thông khác. Theo thống kê, một tỷ lệ lớn các vụ tai nạn giao thông nghiêm trọng có liên quan đến hành vi vượt đèn đỏ, đặc biệt là tại các giao lộ có mật độ phương tiện cao.

Hiện nay, việc giám sát và xử phạt vi phạm giao thông chủ yếu dựa vào hệ thống cảnh sát giao thông và một số camera giám sát truyền thống. Tuy nhiên, phương pháp này còn tồn tại nhiều hạn chế, bao gồm:

- Phụ thuộc vào nhân lực: Lực lượng cảnh sát giao thông không thể có mặt tại tất cả các giao lộ để giám sát liên tục.
- Khả năng phát hiện thủ công kém hiệu quả: Việc theo dõi và xử lý vi phạm bằng mắt thường dễ bị bỏ sót hoặc không đảm bảo độ chính xác cao.
- Thiếu bằng chứng trực quan: Khi xử lý vi phạm, việc thu thập và lưu trữ dữ liệu có thể chưa đầy đủ hoặc không kịp thời.

Với sự phát triển của trí tuệ nhân tạo (AI) và xử lý ảnh, việc áp dụng công nghệ này vào giám sát giao thông có thể mang lại giải pháp tối ưu hơn. Hệ thống nhận diện và phát hiện phương tiện vượt đèn đỏ sử dụng mô hình học sâu (Deep Learning) và thuật toán theo dõi đối tượng (Object Tracking) có khả năng giám sát giao thông liên tục, chính xác và tự động, giảm thiểu sự phụ thuộc vào con người.

Bằng cách ứng dụng công nghệ Yolo11, DeepSORT, OpenCV và các mô hình học sâu, hệ thống giám sát tự động này có thể hỗ trợ các cơ quan chức năng trong việc nâng cao ý thức chấp hành luật giao thông, đồng thời góp phần giảm thiểu tai nạn giao thông và bảo đảm an toàn cho người dân.

Vì những lý do trên, nhóm em lựa chọn đề tài "Xây dựng hệ thống phát hiện vượt đèn đỏ theo thời gian thực" nhằm nghiên cứu và triển khai một giải pháp ứng dụng AI giúp giám sát giao thông hiệu quả hơn.

### 1.2. MỤC TIÊU CỦA ĐỀ TÀI

Xây dựng hệ thống phát hiện phương tiện vượt đèn đỏ theo thời gian thực bằng camera giám sát.

Ứng dụng mô hình học sâu (Deep Learning) để nhận diện phương tiện và đèn tín hiệu giao thông.

Theo dõi chuyển động của phương tiện để xác định hành vi vi phạm.

Lưu trữ thông tin vi phạm vào cơ sở dữ liệu để hỗ trợ quản lý và xử lý vi phạm giao thông.

### **1.3. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU**

Dữ liệu đầu vào: Video từ camera giao thông tại các giao lộ.

Phát hiện đối tượng: Nhận diện phương tiện (ô tô, xe máy, xe tải) và đèn tín hiệu giao thông.

Theo dõi hành vi: Xác định quỹ đạo di chuyển của phương tiện và phát hiện vi phạm vượt đèn đỏ.

Lưu trữ và xử lý dữ liệu: Ghi nhận thông tin vi phạm vào cơ sở dữ liệu PostgreSQL.

Triển khai thử nghiệm: Chạy mô phỏng trên dữ liệu thực tế để đánh giá hiệu suất hệ thống.

### **1.4. BỘ CỤC CỦA BÁO CÁO**

#### **Chương 1: Giới thiệu**

Trình bày lý do chọn đề tài, mục tiêu, phạm vi nghiên cứu và bối cảnh báo cáo.

#### **Chương 2: Cơ sở lý thuyết**

Tìm hiểu về công nghệ sử dụng như YOLO, OpenCV, DeepSORT, PostgreSQL và các phương pháp nhận diện hình ảnh.

#### **Chương 3: Phân tích và thiết kế hệ thống**

Mô tả kiến trúc hệ thống, cách thu thập dữ liệu và phương pháp xử lý ảnh.

#### **Chương 4: Triển khai hệ thống**

Cài đặt mô hình nhận diện, theo dõi phương tiện và phát hiện vi phạm.

#### **Chương 5: Kết quả và đánh giá**

Kiểm tra độ chính xác, hiệu suất hệ thống và các thử nghiệm thực tế.

#### **Chương 6: Kết luận và hướng phát triển**

Tổng kết kết quả đạt được và đề xuất hướng cải tiến trong tương lai.

## **Chương 2: CƠ SỞ LÝ THUYẾT**

### **2.1. Nhận diện hình ảnh và thị giác máy tính**

#### **2.1.1. Thị giác máy tính**

Thị giác máy tính (Computer Vision) là một nhánh của trí tuệ nhân tạo (AI), tập trung vào việc giúp máy tính "nhìn" và "hiểu" thế giới xung quanh thông qua dữ liệu hình ảnh và video. Mục tiêu chính là phát triển các hệ thống có khả năng nhận biết, phân tích và xử lý thông tin thị giác, tương tự như khả năng thị giác của con người.

Lĩnh vực này đóng vai trò quan trọng trong nhiều ứng dụng thực tiễn, từ nhận diện khuôn mặt, phân loại hình ảnh đến các hệ thống lái xe tự động. Sự tiến bộ của các thuật toán học máy và mạng nơ-ron nhân tạo đã thúc đẩy mạnh mẽ việc áp dụng thị giác máy tính vào thực tế.

#### **2.1.2. Tổng quan về nhận diện hình ảnh**

Nhận diện hình ảnh (Image Recognition) là một mảng thuộc thị giác máy tính, cho phép hệ thống xác định và phân loại các đối tượng trong hình ảnh hoặc video. Công nghệ này tận dụng các mô hình học sâu (Deep Learning) để nhận diện đối tượng dựa trên dữ liệu được huấn luyện trước.

#### **2.1.3. Các phương pháp nhận diện**

Các kỹ thuật nhận diện ký tự hiện nay được chia thành hai nhóm chính: phương pháp truyền thống và phương pháp dựa trên học sâu.

Phương pháp truyền thống dựa vào các kỹ thuật xử lý hình ảnh và phân tích đặc điểm hình học của ký tự. Chúng thường sử dụng các bộ lọc để trích xuất đặc trưng như cạnh, góc hoặc hình dạng, sau đó áp dụng các thuật toán phân loại như SVM, KNN hoặc mô hình cây quyết định để nhận dạng. Tuy nhiên, các phương pháp này gặp nhiều hạn chế khi xử lý văn bản có độ biến dạng cao, chữ viết tay hoặc ảnh chất lượng thấp.

Phương pháp học sâu, đặc biệt là các mô hình như YOLO (You Only Look Once), đã chứng minh hiệu quả vượt trội trong các bài toán nhận diện ký tự hiện đại. Thay vì cần cắt riêng từng ký tự trước khi phân loại, mô hình YOLO có thể phát hiện đồng thời tất cả các ký tự trong ảnh bằng cách xác định các bounding box và phân loại trực tiếp từng ký tự trong đó.

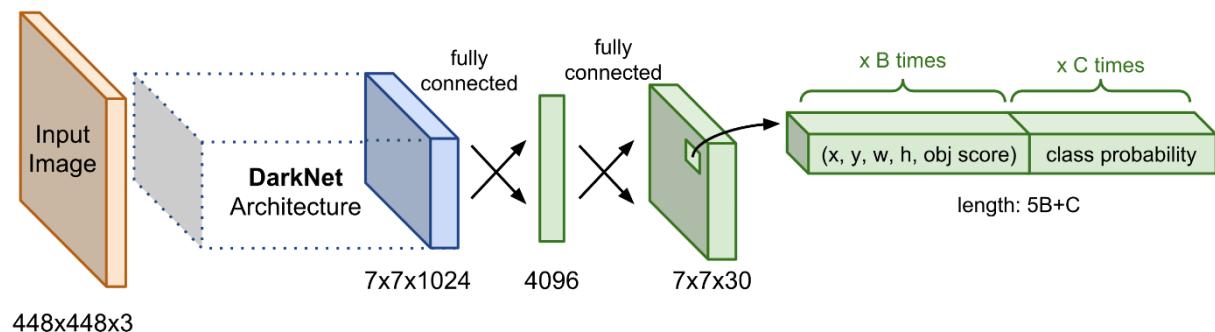
Trong đó, YOLOv1 – một phiên bản cải tiến của dòng mô hình YOLO – cung cấp độ chính xác cao hơn và tốc độ xử lý nhanh hơn. Mô hình được huấn luyện để nhận diện

các ký tự rời (ví dụ: A–Z, 0–9) trong ảnh biển số hoặc văn bản. Mỗi ký tự được phát hiện dưới dạng một đối tượng riêng biệt, với vị trí và nhãn lớp được dự đoán đồng thời.

Sau khi mô hình dự đoán, hệ thống sẽ sắp xếp các ký tự theo thứ tự từ trái sang phải dựa trên vị trí không gian để khôi phục lại chuỗi ký tự đầy đủ. Nhờ đó, quá trình nhận dạng không yêu cầu cắt ký tự thủ công, cũng như không cần sử dụng các cơ chế như RNN hay CTC để xử lý chuỗi.

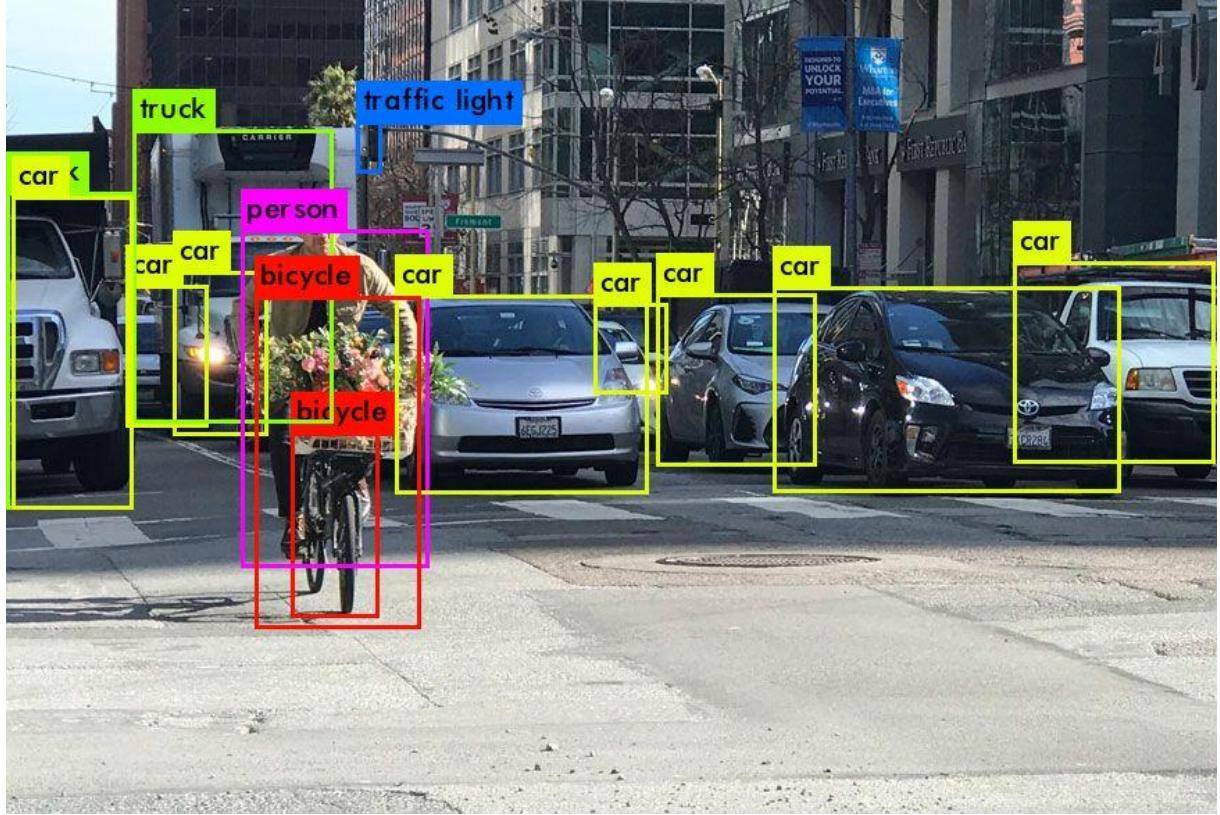
Cơ chế chú ý (Attention Mechanism) cũng có thể được tích hợp vào các mô hình dựa trên YOLO hoặc các pipeline xử lý sau để tăng cường khả năng tập trung vào vùng chứa ký tự, nhất là trong trường hợp ảnh bị mờ, méo hoặc có nhiều nhiễu.

Nhờ áp dụng các kỹ thuật học sâu hiện đại như Yolo11 và các cơ chế hỗ trợ như attention, quá trình nhận diện ký tự trở nên nhanh chóng, chính xác, và đặc biệt hiệu quả trong các bài toán thực tế như nhận dạng biển số xe, văn bản đường phố, chữ viết tay hoặc ảnh chụp trong điều kiện ánh sáng không ổn định



Hình 2.1. Kiến trúc mạng Yolo trong việc nhận diện các ký tự [6]

#### 2.1.4. Phát hiện đối tượng (Object Detection)



Hình 2.2. Phát hiện các đối tượng (phương tiện giao thông) [12]

Phát hiện đối tượng là một kỹ thuật quan trọng trong thị giác máy tính, cho phép xác định vị trí và phân loại các đối tượng trong hình ảnh hoặc video. Một số thuật toán phổ biến bao gồm:

- YOLO (You Only Look Once): Một mô hình thời gian thực với độ chính xác cao, phù hợp cho các ứng dụng yêu cầu tốc độ.
- Faster R-CNN: Một phương pháp mạnh mẽ nhưng có tốc độ chậm hơn so với YOLO.
- SSD (Single Shot MultiBox Detector): Hiệu suất nhanh, tối ưu cho các thiết bị di động.

Trong hệ thống này, Yolo11 được sử dụng để phát hiện phương tiện giao thông.

### 2.2. YOLO

#### 2.2.1. Giới thiệu YOLO

YOLO (You Only Look Once) là một thuật toán phát hiện đối tượng tiên tiến trong lĩnh vực thị giác máy tính. Điểm nổi bật của YOLO là khả năng dự đoán các hộp giới hạn (bounding box) và xác suất của đối tượng chỉ trong một lần xử lý hình ảnh.

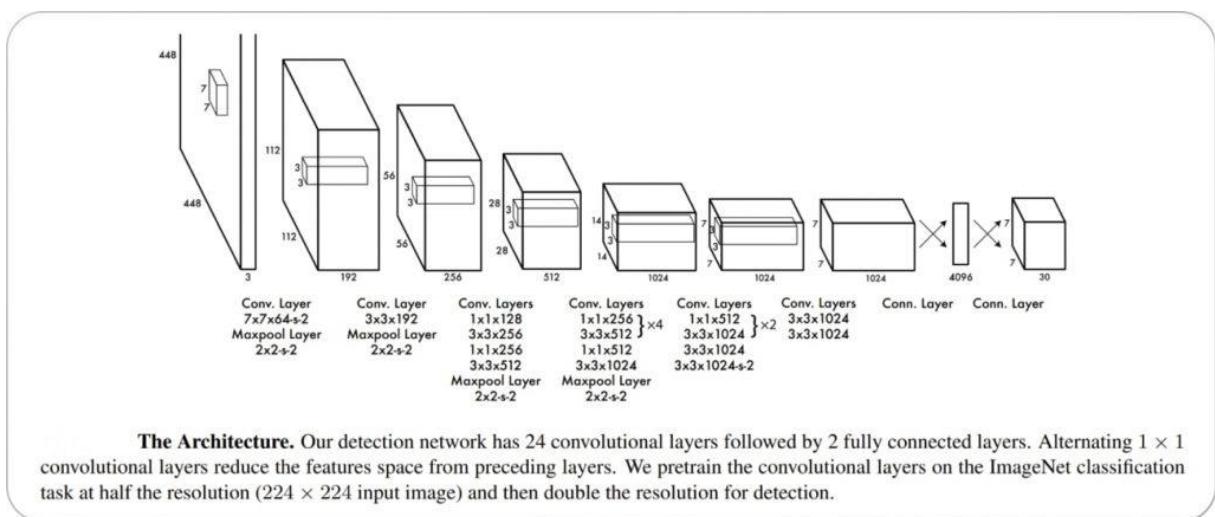
Không giống các phương pháp truyền thống, vốn sử dụng bộ phân loại đã được huấn luyện để nhận diện đối tượng sau khi xác định các vùng quan tâm, YOLO thực hiện toàn

bộ quy trình phát hiện và phân loại trong một lần duy nhất. Cách tiếp cận này giúp YOLO đạt được hiệu suất vượt trội, đặc biệt trong các ứng dụng yêu cầu xử lý theo thời gian thực.

So với các thuật toán trước đây như Faster R-CNN, vốn chia quy trình thành hai giai đoạn (xác định vùng quan tâm và phân loại), YOLO tích hợp tất cả vào một bước, mang lại hiệu quả tính toán cao hơn đáng kể.

### 2.2.2. Cơ chế hoạt động và kiến trúc của YOLO

YOLO nhận hình ảnh đầu vào và sử dụng mạng nơ-ron tích chập (CNN) sâu để phát hiện đối tượng. Kiến trúc CNN, đóng vai trò là xương sống của YOLO, được mô tả như sau



Hình 2.3. Cơ chế hoạt động của Yolo [4]

20 tầng tích chập đầu tiên của mô hình được huấn luyện trước trên tập dữ liệu ImageNet bằng cách sử dụng một tầng tổng hợp trung bình tạm thời và một tầng kết nối đầy đủ. Sau đó, mô hình này được tinh chỉnh để thực hiện nhiệm vụ phát hiện đối tượng. Tầng kết nối đầy đủ cuối cùng của YOLO dự đoán đồng thời xác suất của lớp và tọa độ của các hộp giới hạn.

YOLO chia hình ảnh đầu vào thành một lưới  $S \times S$ . Mỗi ô lưới chịu trách nhiệm phát hiện đối tượng nếu tâm của đối tượng nằm trong ô đó. Mỗi ô dự đoán B hộp giới hạn cùng với điểm tin cậy tương ứng, phản ánh mức độ chắc chắn rằng hộp chứa đối tượng và độ chính xác của hộp được dự đoán.

Trong quá trình huấn luyện, YOLO chỉ chọn một hộp giới hạn đại diện cho mỗi đối tượng, dựa trên chỉ số IOU (Intersection over Union) cao nhất so với thực tế. Điều này giúp các bộ dự đoán hộp giới hạn chuyên biệt hóa, trở nên hiệu quả hơn trong việc dự

đoán các kích thước, tỷ lệ khung hình hoặc loại đối tượng cụ thể, từ đó cải thiện tỷ lệ recall tổng thể.

Một kỹ thuật quan trọng trong YOLO là NMS (Non-Maximum Suppression), được sử dụng trong bước hậu xử lý để nâng cao độ chính xác.

### 2.2.3. Các phiên bản YOLO



Hình 2.4. Các thế hệ, phiên bản Yolo [4]

- YOLOv1 (2015)

YOLOv1 là phiên bản đầu tiên, đánh dấu bước ngoặt trong phát hiện đối tượng bằng cách sử dụng một mạng nơ-ron tích chập (CNN) duy nhất. Với phương pháp “nhìn một lần”, YOLOv1 cho phép phát hiện đối tượng nhanh chóng. Tuy nhiên, hạn chế của phiên bản này nằm ở độ chính xác chưa cao, đặc biệt khi nhận diện các đối tượng nhỏ.

- YOLOv4 (2020)

YOLOv4 mang lại bước tiến lớn với backbone CSPDarknet53, cải thiện cả độ chính xác và tốc độ xử lý. Mô hình này được tối ưu hóa để hoạt động hiệu quả trên các thiết bị di động và edge, trở thành lựa chọn lý tưởng cho các ứng dụng yêu cầu tốc độ cao và độ tin cậy ổn định.

- YOLOv7 (2022)

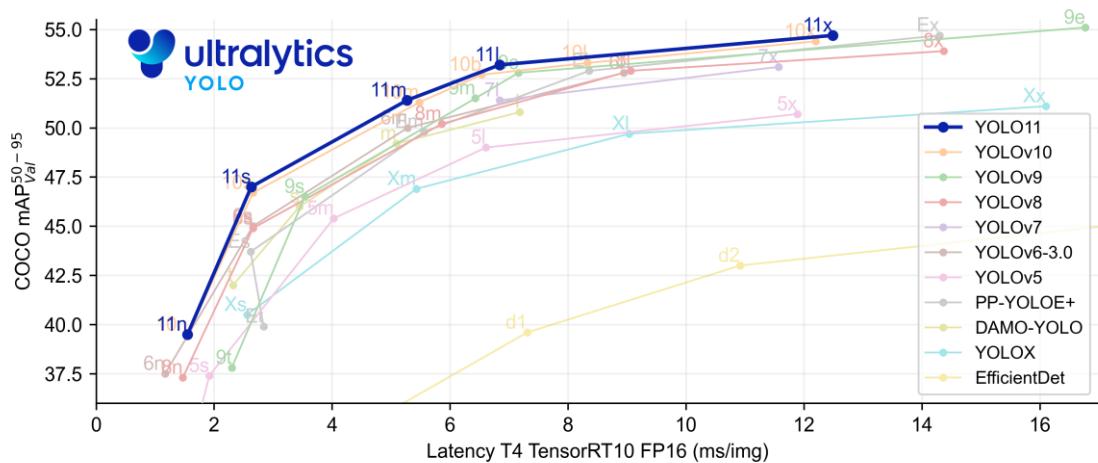
YOLOv7 tiếp tục nâng cao hiệu suất với các tối ưu hóa về mô hình và thuật toán học sâu. Phiên bản này vượt trội trong việc theo dõi đối tượng trong các tình huống thực tế phức tạp, nhờ vào sự cải thiện về độ chính xác và tốc độ.

- YOLOv8 (2023)

YOLOv8 là phiên bản tiên tiến, được tối ưu hóa cho cả nhận diện và phân đoạn đối tượng. Với sự cân bằng hoàn hảo giữa tốc độ và độ chính xác, YOLOv8 phù hợp cho các ứng dụng thời gian thực như giám sát giao thông và hệ thống an ninh, đồng thời hoạt động hiệu quả trên nhiều nền tảng khác nhau.

#### 2.2.4. Giới thiệu Yolo11

Yolo11, phiên bản mới nhất trong dòng YOLO của Ultralytics, tái định nghĩa chuẩn mực về độ chính xác, tốc độ và hiệu quả trong phát hiện đối tượng thời gian thực. Dựa trên nền tảng của các phiên bản trước, Yolo11 mang đến những cải tiến đáng kể về kiến trúc và quy trình huấn luyện, trở thành giải pháp linh hoạt cho nhiều tác vụ thị giác máy tính.



**Hình 2.5. So sánh phiên bản Yolo11 với các phiên bản cũ [15]**

Các tính năng nổi bật:

- Trích xuất đặc trưng nâng cao: Yolo11 sử dụng kiến trúc backbone và neck được cải tiến, tăng cường khả năng trích xuất đặc trưng để phát hiện đối tượng chính xác hơn, kể cả trong các tác vụ phức tạp.
- Tối ưu hóa tốc độ và hiệu quả: Với thiết kế kiến trúc tinh gọn và quy trình huấn luyện được cải thiện, Yolo11 đạt tốc độ xử lý nhanh hơn, đồng thời duy trì sự cân bằng giữa hiệu suất và độ chính xác.
- Độ chính xác cao với ít tham số hơn: Yolo11m đạt độ chính xác trung bình (mAP) cao hơn trên tập dữ liệu COCO, nhưng sử dụng ít hơn 22% tham số so với YOLOv8m, đảm bảo hiệu quả tính toán mà không hy sinh chất lượng.
- Tính linh hoạt đa nền tảng: Yolo11 có thể triển khai trên các thiết bị biến, nền tảng đám mây và hệ thống hỗ trợ GPU NVIDIA, mang lại sự linh hoạt tối đa.

- Hỗ trợ đa tác vụ: Yolo11 không chỉ giới hạn ở phát hiện đối tượng mà còn hỗ trợ phân đoạn thể hiện, phân loại hình ảnh, ước tính tư thế và phát hiện đối tượng theo hướng (OBB).

### 2.2.5. Kiến trúc Yolo11

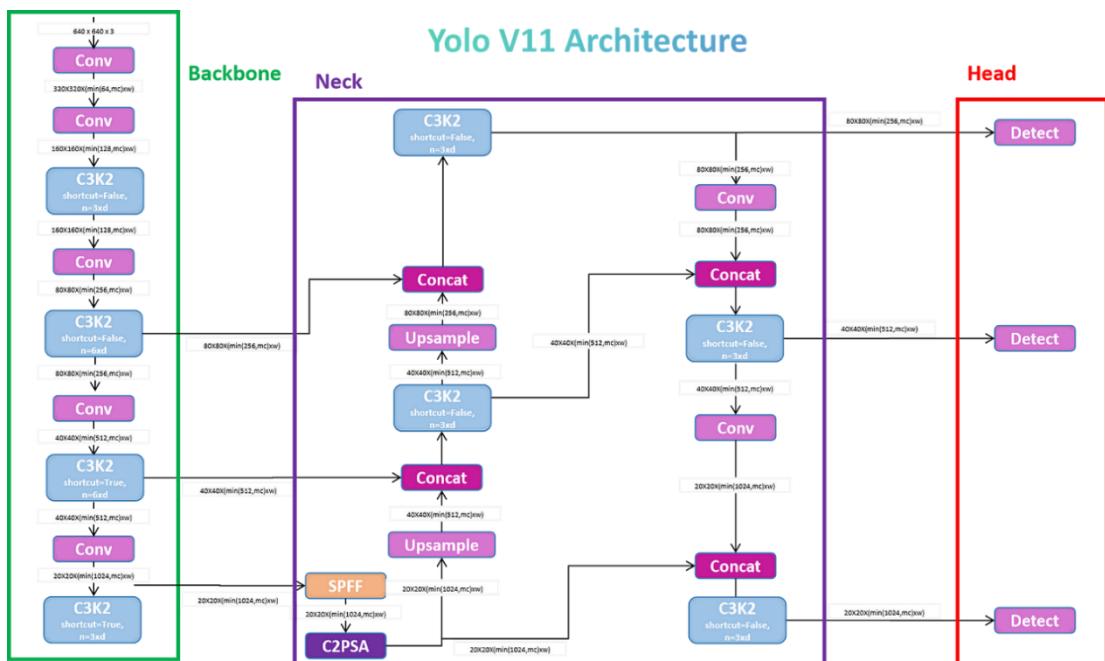
Yolo11 xây dựng và cải tiến từ nền tảng của YOLOv8, bao gồm ba thành phần chính: backbone (trích xuất đặc trưng), neck (tổng hợp đặc trưng) và head (dự đoán). Các cải tiến nổi bật bao gồm:

**Khối C3k2:** Thay thế khối C2f ở neck, được thiết kế để tăng tốc độ và hiệu quả.

**SPPF (Spatial Pyramid Pooling – Fast):** Cải thiện khả năng trích xuất và xử lý đặc trưng.

**C2PSA (Convolutional block with Parallel Spatial Attention):** Tăng cường khả năng tập trung vào các vùng quan trọng trong bản đồ đặc trưng.

Yolo11 cung cấp nhiều kích thước mô hình (từ nano đến cực lớn) để phù hợp với các ứng dụng khác nhau. So với YOLOv8m, Yolo11m đạt độ chính xác cao hơn với ít tham số hơn, đồng thời hỗ trợ nhiều tác vụ như phân đoạn thể hiện, ước tính tư thế và phát hiện đối tượng theo hướng (OBB). Quy trình huấn luyện được tối ưu hóa giúp tăng tốc độ xử lý và triển khai linh hoạt trên các thiết bị biên, đám mây và GPU NVIDIA. Những cải tiến này, cùng với cơ chế chú ý và các khối hiệu quả, giúp Yolo11 trở thành một bước tiến lớn trong lĩnh vực thị giác máy tính.



Hình 2.6. Kiến trúc chi tiết của Yolo11 [8]

## **2.2.6. Ứng dụng Yolo trong hệ thống phát hiện vượt đèn đỏ**

Nhận diện biển số các phương tiện giao thông như ô tô, xe máy,...

Xác định tọa độ của phương tiện để theo dõi hành trình di chuyển, nhận diện biển số xe, phối hợp với OpenCV để phát hiện vạch dừng, từ đó xác định vi phạm.

## **2.3. OpenCV – Thư viện xử lý ảnh mã nguồn mở**

### **2.3.1. Giới thiệu về OpenCV**

OpenCV (Open Source Computer Vision Library) là một thư viện mã nguồn mở chuyên biệt cho thị giác máy tính. Thư viện này cung cấp các công cụ để xử lý hình ảnh và phát triển các ứng dụng đồ họa thời gian thực.

OpenCV được thiết kế để tối ưu hóa hiệu suất CPU trong các tác vụ thời gian thực, đồng thời cung cấp một bộ sưu tập phong phú các hàm xử lý phục vụ cho thị giác máy tính và các ứng dụng học máy.

### **2.3.2. Các module trong OpenCV**

Core Functionality (core): Module này có thiết kế nhỏ gọn, tập trung vào việc định nghĩa các cấu trúc dữ liệu cơ bản như mảng đa chiều. Nó cũng cung cấp các hàm nền tảng hỗ trợ hoạt động của các module khác.

Image Processing (imgproc): Module này hỗ trợ các tác vụ xử lý hình ảnh, bao gồm lọc hình ảnh tuyến tính và phi tuyến, thực hiện các phép biến đổi hình học, chuyển đổi không gian màu, tạo biểu đồ, và nhiều thao tác khác liên quan đến xử lý ảnh.

Video Analysis (video): Đúng như tên gọi, module này tập trung vào phân tích video, cung cấp các chức năng như ước lượng chuyển động, tách nền, và theo dõi đối tượng.

Camera Calibration and 3D Reconstruction (calib3d): Module này cung cấp các thuật toán hình học đa chiều, hiệu chuẩn máy ảnh (đơn và stereo), dự đoán hình dạng đối tượng, sử dụng thuật toán thư tín âm thanh nổi và tái tạo 3D.

2D Features Framework (features2d): Module này hỗ trợ phát hiện và trích xuất các đặc trưng nổi bật, bao gồm bộ nhận diện, bộ mô tả đặc trưng, và các công cụ so khớp đặc trưng.

Ngoài ra, OpenCV còn bao gồm nhiều module khác với các tính năng đa dạng như FLANN (Fast Library for Approximate Nearest Neighbors), Google Test Wrapper, và nhiều công cụ hỗ trợ khác, đáp ứng.

### **2.3.3. Ứng dụng OpenCV trong hệ thống phát hiện vi phạm**

Tiền xử lý ảnh đầu vào để tăng độ chính xác của YOLO.

Đánh dấu vùng vi phạm và cắt ảnh bằng chứng.

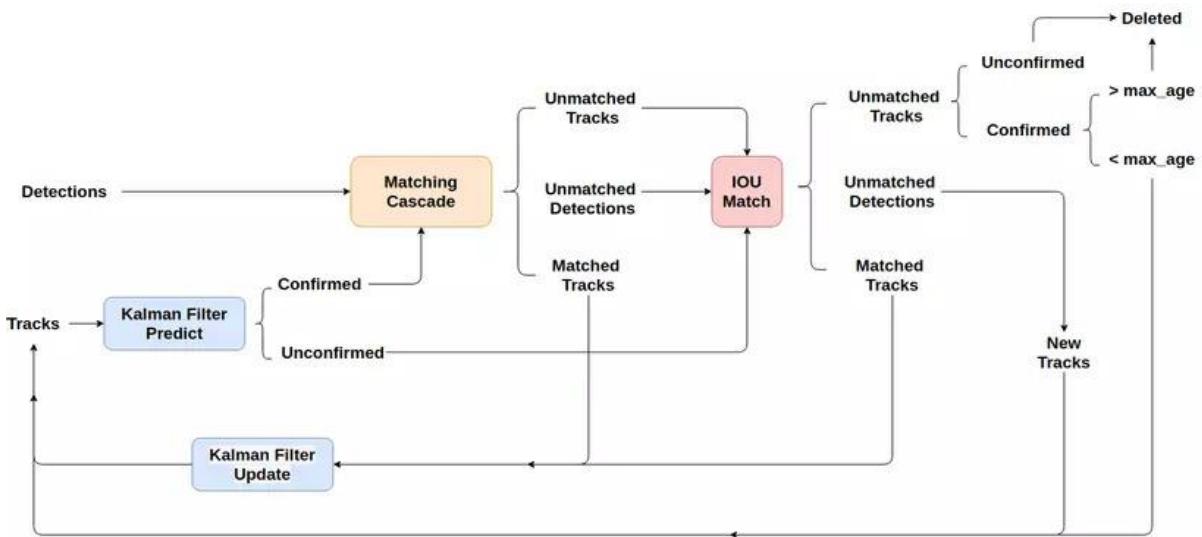
## 2.4. DeepSORT – Thuật toán theo dõi đối tượng

### 2.4.1. Giới thiệu về DeepSORT

DeepSORT (Simple Online and Realtime Tracking), được phát triển bởi Nicolai Wojke và Alex Bewley, là một phiên bản nâng cấp của thuật toán SORT nhằm khắc phục hạn chế về số lượng ID switches cao. DeepSORT sử dụng học sâu (Deep Learning) để trích xuất các đặc trưng của đối tượng, từ đó nâng cao độ chính xác trong việc liên kết dữ liệu. Ngoài ra, thuật toán này áp dụng chiến lược Matching Cascade, giúp cải thiện hiệu quả liên kết các đối tượng sau khi chúng tạm thời biến mất khỏi khung hình.

DeepSORT là một giải pháp theo dõi đối tượng trong video, cho phép xác định quỹ đạo di chuyển của các phương tiện sau khi được phát hiện bởi YOLO.

### 2.4.2. Cách hoạt động của DeepSORT



Hình 2.7. Cách hoạt động của DeepSORT [6]

DeepSORT kết hợp giữa cảm biến Kalman (Kalman Filter) và học sâu (Deep Learning) để:

Gán ID duy nhất cho mỗi phương tiện, tránh trùng lặp. Dự đoán vị trí tiếp theo của đối tượng, giúp theo dõi liên tục ngay cả khi phương tiện bị che khuất trong vài khung hình.

Kết hợp dữ liệu vị trí để ước tính hành vi của phương tiện.

### 2.4.3. Ứng dụng trong hệ thống phát hiện vi phạm

Theo dõi phương tiện qua các khung hình để xác định liệu xe có vượt vạch dừng khi đèn đỏ hay không.

Ghi nhận thông tin và vị trí của từng phương tiện trong video

## 2.5. PostgreSQL – Hệ quản trị cơ sở dữ liệu

### 2.5.1. Giới thiệu về PostgreSQL

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở (RDBMS) nổi bật với tính ổn định, khả năng mở rộng và các tính năng mạnh mẽ. Được phát triển từ năm 1986, PostgreSQL đã trở thành sự lựa chọn phổ biến cho các ứng dụng yêu cầu độ tin cậy cao và tuân thủ các tiêu chuẩn SQL.

Với sự hỗ trợ đầy đủ các tiêu chuẩn SQL và các tính năng nâng cao như lưu trữ dữ liệu dạng JSON, khả năng xử lý các truy vấn phức tạp và bảo đảm tính toàn vẹn của dữ liệu, PostgreSQL là sự lựa chọn tuyệt vời cho các hệ thống cần có khả năng mở rộng, bảo mật và xử lý giao dịch hiệu quả. Hệ quản trị này cũng dễ dàng tích hợp với nhiều ngôn ngữ lập trình và hệ thống quản lý dữ liệu khác, giúp các ứng dụng đa nền tảng hoạt động một cách liền mạch.

### 2.5.2. Ứng dụng PostgreSQL trong hệ thống

Lưu trữ thông tin vi phạm, bao gồm:

- ID phương tiện
- Thời gian vi phạm
- Ảnh bằng chứng
- Biển số xe
- Loại vi phạm

Truy vấn dữ liệu vi phạm theo thời gian thực, hỗ trợ thống kê và báo cáo.

## **Chương 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**

### **3.1. Kiến trúc hệ thống**

Hệ thống phát hiện phương tiện vượt đèn đỏ theo thời gian thực được thiết kế theo mô hình module hóa, nghĩa là toàn bộ hệ thống được chia thành nhiều module chức năng riêng biệt. Mỗi module đảm nhiệm một nhiệm vụ cụ thể, từ việc nhận diện phương tiện, theo dõi hành vi, phát hiện trạng thái đèn tín hiệu, đèn lưu trữ dữ liệu vi phạm. Việc phân chia này giúp tăng khả năng mở rộng, dễ dàng bảo trì và nâng cấp sau này.

#### **Mô-đun nhận diện đối tượng (Object Detection):**

- Sử dụng mô hình Yolo11 để phát hiện các đối tượng trong khung hình video, bao gồm: phương tiện giao thông (ô tô, xe máy, xe tải) và đèn tín hiệu giao thông với trạng thái (đỏ, vàng, xanh).
- Mô hình được huấn luyện tùy chỉnh để nhận diện cả các loại đèn (red\_light, yellow\_light, green\_light) bên cạnh các loại phương tiện.

#### **Mô-đun theo dõi và phân tích hành vi (Vehicle Tracking & Violation Analysis):**

- Áp dụng thuật toán DeepSORT để theo dõi các phương tiện đã được nhận diện xuyên suốt các khung hình.
- Kết hợp thông tin quỹ đạo di chuyển của từng phương tiện với trạng thái đèn giao thông tại thời điểm tương ứng để phát hiện hành vi vượt đèn đỏ.

#### **Mô-đun lưu trữ và quản lý dữ liệu (Data Storage):**

- Sử dụng hệ quản trị cơ sở dữ liệu **PostgreSQL** để lưu trữ toàn bộ thông tin vi phạm bao gồm:
  - ID phương tiện
  - Thời gian vi phạm
  - Ảnh bằng chứng
  - Biển số xe
  - Loại vi phạm

#### **Mô-đun giao diện người dùng (User Interface):**

- Cho phép giám sát thời gian thực và truy xuất báo cáo các vi phạm.
- Giao diện trực quan, dễ sử dụng cho cán bộ giao thông hoặc quản trị viên.

### **3.2. Quy trình thu thập và xử lý dữ liệu**

Quy trình xử lý dữ liệu được chia thành nhiều bước liên tục và logic nhằm đảm bảo tính chính xác, thời gian thực và hiệu quả hệ thống:

#### **Bước 1: Thu thập dữ liệu video**

- Dữ liệu đầu vào là video streaming trực tiếp từ app Điện Biên Smart có camera tại giao lộ GT\_Ngã 4 đèn đỏ Sân vận động\_4 – Thành phố Điện Biên Phủ.

#### **Bước 2: Tiền xử lý video**

- Áp dụng các kỹ thuật tiền xử lý hình ảnh nhằm cải thiện chất lượng video đầu vào:

- Giảm nhiễu (Noise Reduction)
- Điều chỉnh độ sáng – tương phản
- Cắt cúp khung hình để chỉ giữ lại vùng quan trọng (ví dụ: vùng chứa phương tiện và đèn tín hiệu)

#### **Bước 3: Nhận diện phương tiện**

- Yolo11 được sử dụng để phát hiện các phương tiện giao thông trong từng khung hình.
- Mỗi đối tượng được đánh dấu bằng hộp giới hạn (bounding box) cùng nhãn loại phương tiện, ví dụ: car, motorbike, truck.

#### **Bước 4: Theo dõi và phân tích hành vi**

- Mô hình DeepSORT theo dõi liên tục chuyển động của các phương tiện từ lúc xuất hiện đến khi ra khỏi khung hình.
- Dựa trên quỹ đạo di chuyển và trạng thái đèn, hệ thống xác định hành vi vi phạm:
  - Nếu một phương tiện vượt qua vạch dừng khi đèn đỏ, hệ thống đánh dấu là vi phạm.

#### **Bước 5: Phân tích tín hiệu đèn giao thông**

- Yolo11 được sử dụng để phát hiện các trạng thái đèn giao thông trong từng khung hình.
- Hệ thống phân biệt giữa các trạng thái đỏ – vàng – xanh theo thời gian thực, đồng bộ với hành vi của phương tiện.

#### **Bước 6: Lưu trữ và quản lý dữ liệu vi phạm**

- Mỗi sự kiện vi phạm được lưu trữ dưới dạng bản ghi gồm:
  - ID phương tiện

- Thời gian vi phạm
- Ảnh bằng chứng
- Biển số xe
- Loại vi phạm
- Dữ liệu này giúp dễ dàng kiểm tra lại sự kiện và trích xuất báo cáo theo ngày, theo loại phương tiện, hoặc theo thời gian.

### 3.3. Quy trình hoạt động của hệ thống nhận dạng biển số xe

Quy trình hoạt động của hệ thống nhận dạng biển số xe được thiết kế thành hai giai đoạn chính: giai đoạn phát hiện biển số và giai đoạn nhận dạng ký tự, lần lượt tương ứng với hai pipeline là Detection Pipeline và Recognition Pipeline. Mỗi giai đoạn bao gồm nhiều bước xử lý liên tiếp, phối hợp với nhau để đảm bảo kết quả đầu ra chính xác và hiệu quả.

Hệ thống bắt đầu với ảnh đầu vào, là hình ảnh được chụp từ camera giao thông hoặc tải lên từ người dùng. Ảnh này sẽ trải qua bước tiền xử lý, bao gồm các kỹ thuật chuẩn hóa kích thước, chuyển đổi màu hoặc tăng cường ảnh nếu cần thiết, nhằm chuẩn bị tốt nhất cho giai đoạn phát hiện.

Tiếp theo, ảnh đã qua xử lý được đưa vào pipeline phát hiện biển số, trong đó mô hình Yolo11 được sử dụng để phát hiện vị trí biển số trong ảnh đầu vào. Mô hình này không chỉ phát hiện các vùng chứa biển số mà còn xác định được hộp giới hạn có hướng (Oriented Bounding Boxes), cho phép xử lý các biển số bị nghiêng hoặc có góc chụp không thuận lợi.

Sau khi phát hiện, hệ thống sẽ thực hiện bước biến đổi phối cảnh (perspective transform) để hiệu chỉnh góc nhìn của biển số, đưa chúng về dạng vuông vức, giúp chuẩn bị cho giai đoạn cắt. Kết quả của bước này là các ảnh biển số đã được hiệu chỉnh, sau đó được cắt riêng biệt từ ảnh gốc để tạo thành tập ảnh biển số đầu vào cho quá trình nhận dạng.

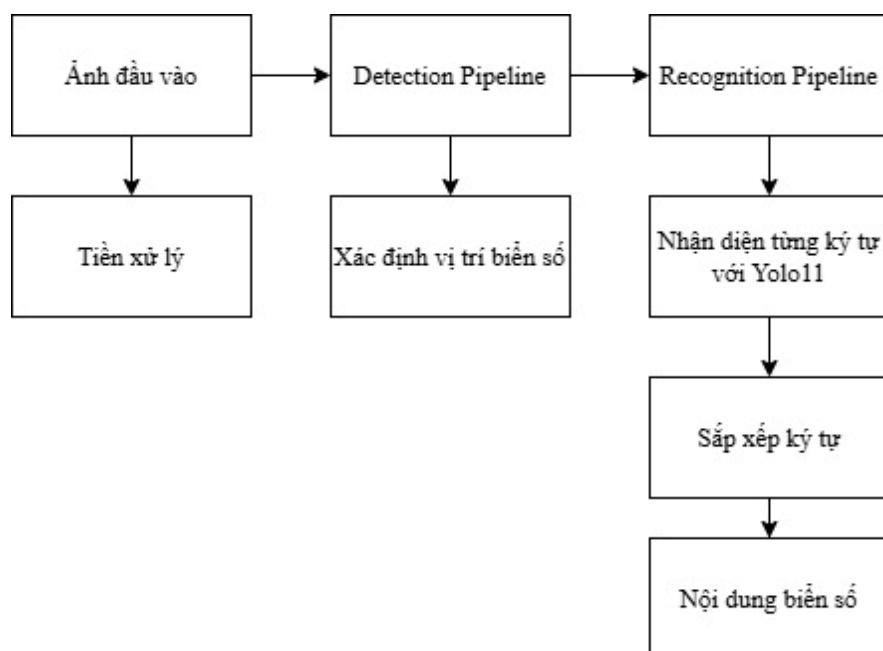
Trong pipeline nhận dạng, mỗi biển số đã được cắt riêng sẽ tiếp tục qua một bước chuẩn hóa (normalization), bao gồm điều chỉnh kích thước, định dạng và độ tương phản ảnh để phù hợp với mô hình nhận diện.

Sau đó, ảnh được đưa vào mô hình Yolo11, một mô hình phát hiện đối tượng mạnh mẽ, được huấn luyện chuyên biệt để nhận diện từng ký tự riêng lẻ trên biển số.

Mỗi ký tự (chữ hoặc số) trên biển số sẽ được mô hình phát hiện dưới dạng một bounding box với nhãn tương ứng là ký tự đó.

Các ký tự được phát hiện sẽ được sắp xếp theo vị trí từ trái sang phải, dựa trên tọa độ trung tâm của các bounding box, để tái dựng chuỗi ký tự chính xác theo đúng thứ tự xuất hiện trên biển số.

Đây là bước cuối cùng trong quy trình, nơi hệ thống trả về nội dung biển số đã nhận dạng được từ ảnh đầu vào ban đầu.



Hình 3.1. Quy trình hoạt động của hệ thống nhận dạng biển số xe

### 3.4. Thiết kế các module phần mềm

Hệ thống được tổ chức thành các mô-đun chức năng riêng biệt, dễ bảo trì và mở rộng:

- **Mô-đun nhận diện phương tiện (Vehicle Detection):**
  - Sử dụng Yolo11 để phát hiện phương tiện và đèn giao thông trong từng khung hình.
  - Đầu ra gồm các bounding box và nhãn loại đối tượng (car, motorbike, truck, red\_light, green\_light, yellow\_light).
- **Mô-đun theo dõi phương tiện (Vehicle Tracking):**
  - Dựa vào kết quả của Yolo11, DeepSORT sẽ theo dõi liên tục vị trí từng phương tiện với ID duy nhất.
  - Quỹ đạo và tốc độ di chuyển được cập nhật liên tục.
- **Mô-đun phát hiện hành vi vi phạm (Violation Detection):**

- Tích hợp thông tin từ mô-đun tracking và nhận diện đèn giao thông để xác định thời điểm vi phạm.
  - So sánh vị trí của phương tiện với vị trí vạch dừng và trạng thái đèn tại thời điểm đó.
- **Mô-đun lưu trữ dữ liệu (Data Storage):**
    - Lưu trữ các bản ghi vi phạm vào PostgreSQL.
    - Thông tin gồm: ID phương tiện, thời gian vi phạm, ảnh bằng chứng, biển số xe, loại vi phạm.

### 3.5. Thiết kế giao diện người dùng (UI)

Hệ thống cung cấp một giao diện trực quan, hỗ trợ người dùng quản lý và giám sát hiệu quả:

- **Giao diện giám sát thời gian thực:**
  - Hiển thị video từ camera giao thông cùng với các bounding box, ID phương tiện và trạng thái đèn.
  - Khi có vi phạm, thông báo trực tiếp trên màn hình (real-time alert).
- **Giao diện báo cáo và tra cứu vi phạm:**
  - Cho phép tìm kiếm và lọc vi phạm theo nhiều tiêu chí: thời gian, biển số, id phương tiện.
  - Hiển thị thông tin chi tiết và ảnh bằng chứng cho mỗi vi phạm đã ghi nhận.

## Chương 4: TRIỂN KHAI HỆ THỐNG

### 4.1. Kiến trúc hệ thống

Hệ thống giám sát và phát hiện phương tiện vi phạm tín hiệu giao thông được xây dựng theo kiến trúc gồm bốn thành phần chính:

- **Camera giám sát:** video streaming trực tiếp từ app Điện Biên Smart có camera tại giao lộ GT\_Ngã 4 đèn đỏ Sân vận động\_4 – Thành phố Điện Biên Phủ
- **Server xử lý trung tâm:** Chịu trách nhiệm chạy các mô hình trí tuệ nhân tạo (AI) để:
  - Phát hiện phương tiện giao thông
  - Nhận diện biển số xe
  - Nhận diện trạng thái đèn tín hiệu
- **Cơ sở dữ liệu (PostgreSQL):** Lưu trữ thông tin phương tiện vi phạm.
- **Giao diện người dùng:** Hiển thị trực quan các kết quả nhận diện, báo cáo thống kê và quản lý dữ liệu vi phạm.

### 4.2. Công nghệ sử dụng

**Ngôn ngữ lập trình:** Python

**Thư viện xử lý ảnh:** OpenCV, TensorFlow/Keras

**Mô hình AI chính:** Yolo11 (You Only Look Once phiên bản 11 - mô hình mạnh về phát hiện đối tượng thời gian thực)

**Cơ sở dữ liệu:** PostgreSQL

**Công cụ hỗ trợ huấn luyện:** GPU (sử dụng CUDA trên NVIDIA), PyTorch, Ultralytics YOLO

### 4.3. Xây dựng các thành phần chính

#### 4.3.1. Nhận diện vùng chứa biển số xe của phương tiện vi phạm

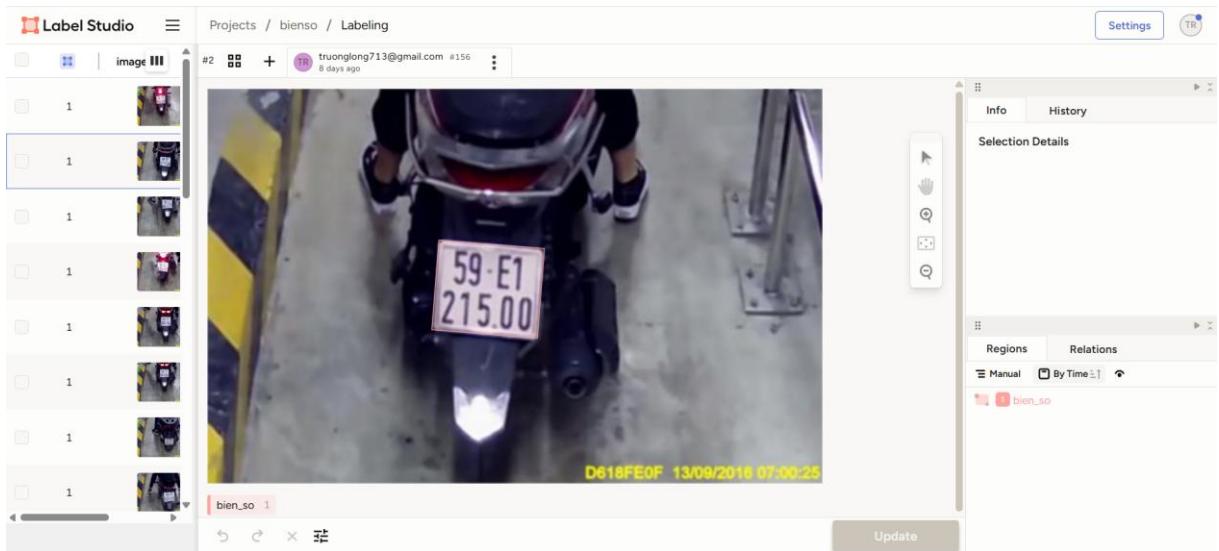
**Mục tiêu:** Tìm vị trí vùng chứa biển số xe trong khung hình (ảnh/video), nhằm cắt đúng khu vực biển số để đưa vào bước nhận dạng ký tự sau đó.

**Phương pháp:** Sử dụng mô hình **YOLO11x** - một mô hình deep learning tối ưu cho bài toán phát hiện đối tượng (object detection).

- Chuẩn bị dữ liệu

Dữ liệu gồm các ảnh phương tiện giao thông được lấy từ:

<https://thigiacmaytinh.com/tai-nguyen-xu-ly-anh/tong-hop-data-xu-ly-anh/>



**Hình 4.1. Giao diện Label Studio về đánh nhãn biển số xe**

Định dạng theo chuẩn YOLO:

- Mỗi ảnh có file .txt đi kèm chứa thông tin **tọa độ bounding box (center\_x, center\_y, width, height)** được chuẩn hóa theo kích thước ảnh.

```

0 0.12508316699933467 0.3113772455089821 0.19693945442448438
0.11576846307385236
0 0.7531603459747171 0.5119760479041916 0.21290751829673993
0.125748502994012

```

**Hình 4.2. Tọa độ bounding box vị trí của biển số xe**

- File mydata\_biensoxe.yaml mô tả các nhãn (labels) và đường dẫn tới dữ liệu huấn luyện, validation.



```
1 path: /kaggle/input/data-biensoxe/dataset
2 train: /kaggle/input/data-biensoxe/dataset/images/train
3 val: /kaggle/input/data-biensoxe/dataset/images/val
4 test:
5
6 # Classes
7 names:
8 0: bien_so
```

Hình 4.3. Nội dung mydata\_biensoxe.yaml

- Cấu hình và huấn luyện mô hình



```
1 from ultralytics import YOLO
2 import torch
3
4 if __name__ == '__main__':
5     torch.multiprocessing.set_start_method('spawn', force=True)
6
7 model = YOLO("/kaggle/input/yolo11-bienso/pytorch/default/1/yolo11x.pt")
8 device = 'cuda' if torch.cuda.is_available() else 'cpu'
9
10 results = model.train(
11     data="/kaggle/input/mydata-biensoxe/mydata_biensoxe.yaml",
12     epochs=150,
13     batch=16,
14     device='0,1',
15     workers=8,
16     cache=True,
17     amp=True,
18     imgsz=640,
19     project="/kaggle/working/",
20     name="license_plate",
21 )
```

Hình 4.4. Nội dung mô hình huấn luyện

Trong đó:

model: sử dụng mô hình Yolo11x đã được tiền huấn luyện (yolo11x.pt).

epochs: 150 – số vòng lặp huấn luyện.

batch: 16 – kích thước lô dữ liệu.

device: sử dụng 2 GPU (0,1) nếu có sẵn.  
 workers: 8 – số tiến trình tải dữ liệu song song.  
 cache=True: bật bộ nhớ đệm dữ liệu để tăng tốc độ huấn luyện.  
 amp=True: sử dụng tính toán số học chính xác hỗn hợp (mixed precision) giúp tăng hiệu suất.

imgsz=640: kích thước ảnh đầu vào được chuẩn hóa về 640x640 pixels.

- Nguyên lý học và tính toán trong mô hình Yolo11

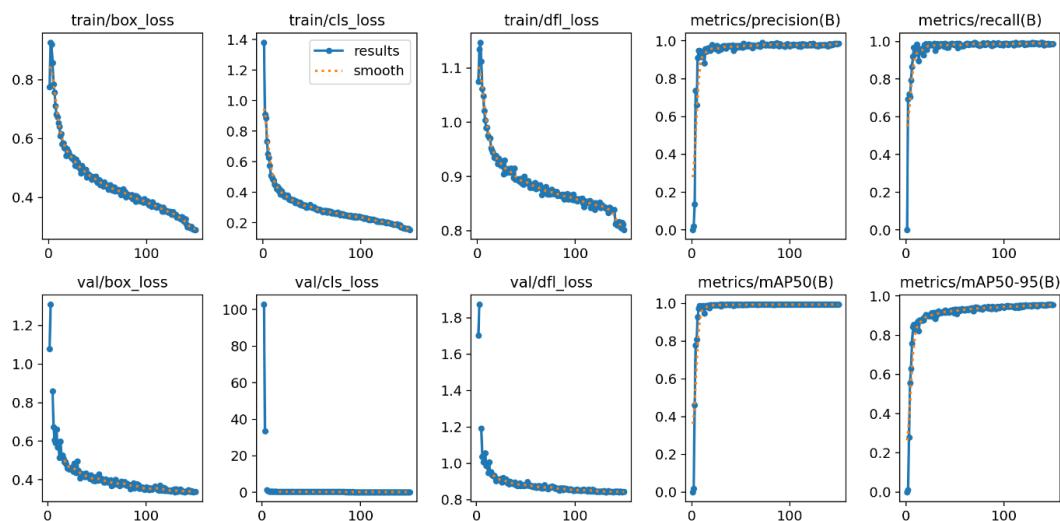
### Tiền xử lý ảnh

- Ảnh đầu vào sẽ được **chuẩn hóa kích thước** về 640x640.
- Được chia thành **S x S ô lưới** (ví dụ: 20x20).
- Mỗi ô lưới dự đoán **B bounding box**, mỗi box gồm:
  - x, y: tọa độ trung tâm (tương đối trong ô).
  - w, h: chiều rộng, chiều cao (tương đối toàn ảnh).
  - C: xác suất có đối tượng trong ô.
  - P(class\_i): xác suất thuộc từng lớp.

### Quá trình học

- Ảnh vào mô hình → qua nhiều lớp Convolution → xuất tensor đầu ra chứa thông tin: bounding boxes, confidence, class.
- So sánh với ground truth để tính hàm mất mát.
- Lan truyền ngược (Backpropagation) và tối ưu (SGD/Adam) để cập nhật trọng số.

### Kết quả của mô hình:



Hình 4.5. Kết quả mô hình nhận diện biển số sau huấn luyện

- **Các biểu đồ tổn thất (Loss)**

- Train Loss
  - + train/box\_loss: Tổn thất localization (vị trí bounding box) giảm đều từ ~0.85 về ~0.25 → cho thấy mô hình học dần cách xác định vị trí biến số tốt hơn.
  - + train/cls\_loss: Tổn thất phân loại giảm đều từ ~1.4 xuống dưới 0.2 → mô hình ngày càng phân biệt rõ đâu là biến số.
  - + train/dfl\_loss (Distribution Focal Loss): Giảm ổn định từ ~1.1 xuống ~0.8 → thể hiện mô hình học tốt cách dự đoán tọa độ chính xác hơn
- Validation Loss
  - + val/box\_loss: Giảm tương tự training, từ ~1.2 xuống dưới 0.4 → không có overfitting rõ ràng.
  - + val/cls\_loss: Ban đầu spike rất cao (100) nhưng nhanh chóng giảm xuống thấp (dưới 1) → có thể là do một vài batch đầu có vấn đề dữ liệu phân loại, nhưng mô hình đã học nhanh và ổn định.
  - + val/dfl\_loss: Tương tự train/dfl\_loss, giảm ổn định → xác nhận mô hình học tốt ở cả tập validation.

- **Các chỉ số đánh giá mô hình**

- Precision (Độ chính xác):
  - + metrics/precision(B): Tăng mạnh từ 0 đến ~0.97 và duy trì ổn định → mô hình ít dự đoán sai nhẫn biến số (false positives thấp).
- Recall (Độ bao phủ):
  - + metrics/recall(B): Tăng đều và đạt khoảng 0.99 → mô hình phát hiện gần như đầy đủ tất cả các biến số (false negatives thấp).
- mAP50 và mAP50-95:
  - + metrics/mAP50(B): Gần như đạt mức 1.0 (hoàn hảo) → tại ngưỡng IoU 0.5, mô hình xác định cực kỳ chính xác vị trí và nhẫn.
  - + metrics/mAP50-95(B): Đạt gần 0.95 → mô hình giữ được độ chính xác cao cả ở các ngưỡng IoU khó hơn (0.5 đến 0.95), cho thấy mức tổng quát hóa rất tốt.

- **Kết luận**

- Tổn thất giảm đều và ổn định trên cả tập train và validation.
- Không có dấu hiệu overfitting rõ rệt.
- Các chỉ số đánh giá (precision, recall, mAP) đều gần chạm ngưỡng hoàn hảo.

## Chạy thực nghiệm



Hình 4.6. Chạy thực nghiệm nhận diện biển số ô tô



Hình 4.7. Chạy thực nghiệm nhận diện biển số xe máy

### 4.3.2. Nhận diện các ký tự trong biển số của phương tiện vi phạm

Trong hệ thống đề xuất, quá trình nhận diện ký tự trên biển số được thực hiện bằng cách sử dụng mô hình Yolo11 – một mô hình phát hiện đối tượng hiện đại, có khả năng nhận diện chính xác từng ký tự trong ảnh đầu vào.

Phương pháp sử dụng Yolo11 sẽ xử lý ảnh biển số theo hướng phân tách từng ký tự trực tiếp thông qua phát hiện vật thể (object detection). Toàn bộ ảnh biển số sẽ được đưa vào mô hình Yolo11, nơi mỗi ký tự sẽ được dự đoán như một đối tượng riêng biệt với bounding box và nhãn lớp tương ứng (ví dụ: “3”, “0”, “F”, ...).

Sau khi mô hình trả về danh sách các ký tự được phát hiện, hệ thống sẽ sắp xếp các bounding box theo thứ tự từ trái sang phải dựa trên tọa độ x-center của từng box. Việc sắp xếp này cho phép tái dựng chuỗi biển số theo đúng trật tự xuất hiện các ký tự trên ảnh.

Phương pháp này đơn giản hóa pipeline nhận diện: không cần cắt thủ công từng ký tự trước đó, không cần giải mã chuỗi, và có thể xử lý cả các biển số có độ dài ký tự không cố định. Ngoài ra, việc huấn luyện mô hình Yolo11 cũng diễn ra theo cách trực quan, nhờ vào dữ liệu nhãn bounding box từng ký tự được gán sẵn trong ảnh.

- **Chuẩn bị dữ liệu**

Bộ dữ liệu được thu thập từ Kaggle – cụ thể là dataset chứa các ảnh biển số xe thật ở nhiều điều kiện môi trường khác nhau. Tác giả đã lựa chọn các ảnh rõ biển số, đủ chất lượng để phục vụ huấn luyện mô hình nhận dạng.

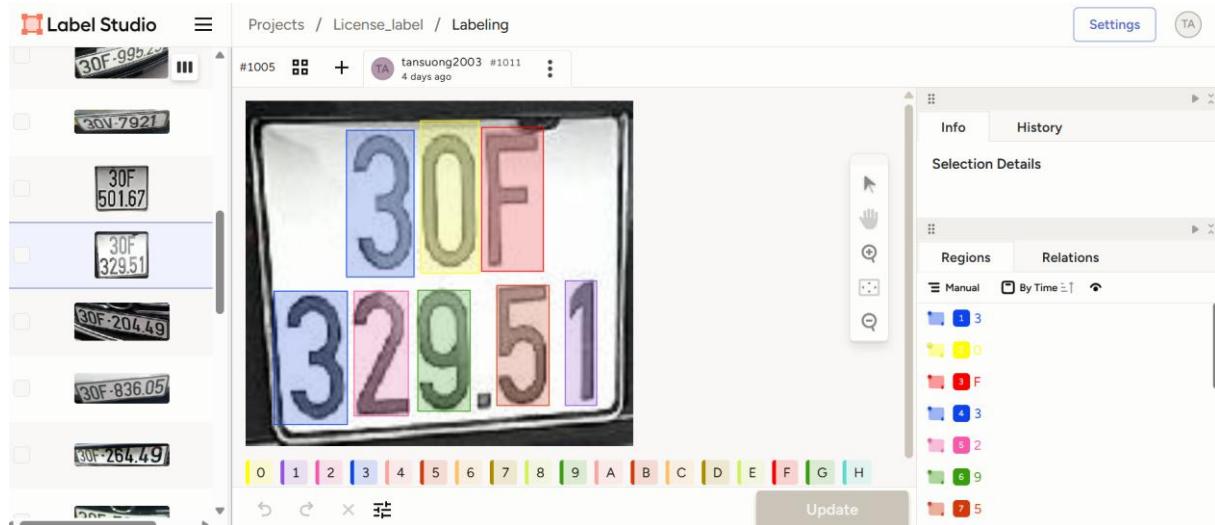
Tên dataset: Vietnamese License Plate OCR

Số lượng ảnh gốc: Khoảng 12.000 ảnh.

Link tham khảo: <https://www.kaggle.com/datasets/topkek69/vietnameselicense-plate-ocr>



**Hình 4.8. Biển số ô tô**



**Hình 4.9. Giao diện Label Studio về đánh nhãn biển số xe**

Định dạng theo chuẩn YOLO:

- Mỗi ảnh có file .txt đi kèm chứa thông tin **tọa độ bounding box (center\_x, center\_y, width, height)** được chuẩn hóa theo kích thước ảnh.

```

3 0.09649950819904872 0.3845789971617785 0.09366126805372658 0.47990838022207816
0 0.19725638599810785 0.4453268933924214 0.09271523178807946 0.45560922172982116
15 0.2904446546830653 0.4757008415077428 0.08798486281929989 0.4677588009759497
3 0.45411542100283825 0.5394861325499177 0.09176915799432357 0.46168401135288534
2 0.5610217596972563 0.5592291988248768 0.08798486281929996 0.4890205646566747
6 0.6631977294228949 0.5774535676940695 0.08420056764427628 0.4890205646566747
4 0.7965941093237396 0.5987144591055893 0.08987701040681188 0.45864661654135347
3 0.9020813123087855 0.6230149696151146 0.09649952696310311 0.4768700259144203

```

**Hình 4.10. Tọa độ bounding box vị trí của biển số xe**

- File mydata\_bienso.yaml mô tả các nhãn (labels) và đường dẫn tới dữ liệu huấn luyện, validation.



```

1 path: /kaggle/input/datakytu/datakytu
2 train: images/train
3 val: images/val
4 test: # Để tránh nếu không có test
5 names: ['0', '1', '2', '3', '4', '5', '6', '7',
6     '8', '9', 'A', 'B', 'C', 'D', 'E', 'F',
7     'G', 'H', 'K', 'L', 'M', 'N', 'P', 'R',
8     'S', 'T', 'U', 'V', 'X', 'Y', 'Z']
9

```

**Hình 4.11. Nội dung mydata\_bienso.yaml**

- Cấu hình và huấn luyện mô hình



```

1 import torch
2 from ultralytics import YOLO
3
4 def train_model(yaml_path):
5     torch.multiprocessing.set_start_method('spawn', force=True)
6     model = YOLO("/kaggle/input/yolo11-dengiaothong/pytorch/default/1/yolo11x.pt")
7     device = 'cuda' if torch.cuda.is_available() else 'cpu'
8
9     results = model.train(
10         data=yaml_path,
11         epochs=250,
12         batch=16,
13         device=device,
14         workers=8,
15         cache=False,
16         amp=True,
17         imgsz=416,
18         project="/kaggle/working/",
19         name="train_label_bienso",
20         verbose=True
21     )

```

**Hình 4.12. Nội dung mô hình huấn luyện**

Trong đó:

model: sử dụng mô hình Yolo11x đã được tiền huấn luyện (yolo11x.pt).  
 epochs: 250 – số vòng lặp huấn luyện.

batch: 16 – kích thước lô dữ liệu.

device: sử dụng 2 GPU (0,1) nếu có sẵn.

workers: 8 – số tiến trình tải dữ liệu song song.

cache=True: bật bộ nhớ đệm dữ liệu để tăng tốc độ huấn luyện.

amp=True: sử dụng tính toán số học chính xác hỗn hợp (mixed precision) giúp tăng hiệu suất.

imgsz=416: kích thước ảnh đầu vào được chuẩn hóa về 416x416 pixels.

- Nguyên lý học và tính toán trong mô hình Yolo11

### Tiền xử lý ảnh

- Ảnh đầu vào sẽ được **chuẩn hóa kích thước** về 416x416.
- Được chia thành **S x S ô lưới** (ví dụ: 20x20).
- Mỗi ô lưới dự đoán **B bounding box**, mỗi box gồm:
  - x, y: tọa độ trung tâm (tương đối trong ô).
  - w, h: chiều rộng, chiều cao (tương đối toàn ảnh).
  - C: xác suất có đối tượng trong ô.
  - P(class\_i): xác suất thuộc từng lớp (0-9, A-Z).

### Quá trình học

- Ảnh được đưa vào mô hình → đi qua nhiều tầng convolution để trích xuất đặc trưng → xuất ra một tensor đầu ra chứa thông tin:

Bounding boxes

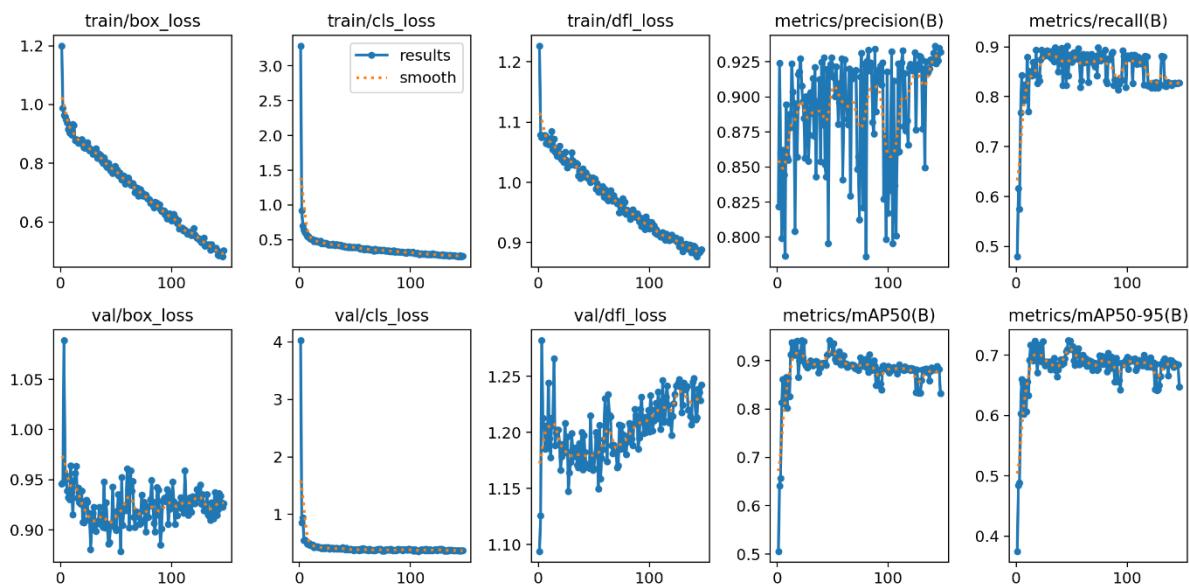
Confidence scores

Class probabilities

- Kết quả dự đoán được so sánh với ground truth để tính hàm mất mát (loss function).

- Cuối cùng, hệ thống thực hiện lan truyền ngược (backpropagation) và sử dụng thuật toán tối ưu như SGD hoặc Adam để cập nhật trọng số mạng lưới theo từng epoch.

### Kết quả của mô hình:



**Hình 4.13. Kết quả mô hình nhận diện biển số sau huấn luyện**

- **Các biểu đồ tổn thất (Loss)**

- **Train Loss**

- + train/box\_loss: Tổn thất định vị (localization loss) giảm đều từ khoảng 1.2 xuống dưới 0.6, thể hiện mô hình dần học tốt cách xác định vị trí các ký tự trên biển số.
- + train/cls\_loss: Tổn thất phân loại giảm rất nhanh từ ~3.2 xuống gần 0, cho thấy mô hình học tốt trong việc phân biệt giữa các lớp ký tự (từ A–Z và 0–9).
- + train/dfl\_loss (Distribution Focal Loss): Giảm từ ~1.2 xuống ~0.9, phản ánh khả năng mô hình học tọa độ chi tiết chính xác hơn theo thời gian.

- **Validation Loss**

- + val/box\_loss: Mặc dù dao động nhẹ, nhưng xu hướng giảm từ ~1.05 xuống ~0.9 cho thấy mô hình không bị overfitting rõ rệt và vẫn giữ được độ chính xác cao trên tập kiểm tra.
- + val/cls\_loss: Giảm rất nhanh từ ~4 xuống gần 0, điều này phản ánh mô hình có khả năng tổng quát hóa tốt khi áp dụng lên dữ liệu chưa thấy trước đó.
- + val/dfl\_loss: Ổn định và giảm nhẹ từ ~1.25 xuống ~1.15, phù hợp với xu hướng giảm của train/dfl\_loss, cho thấy quá trình huấn luyện diễn ra hiệu quả.

- **Các chỉ số đánh giá mô hình**

- **Precision (Độ chính xác)**

- + metrics/precision(B): Tăng từ ~0.85 lên gần 0.95, mặc dù có dao động nhưng xu hướng chung ổn định. Điều này cho thấy tỷ lệ nhãn dự đoán sai là thấp (false positives thấp).

- Recall (Độ bao phủ)
  - + metrics/recall(B): Tăng ổn định và duy trì ở mức ~0.90, cho thấy mô hình phát hiện được hầu hết các ký tự thực sự có mặt trong ảnh (false negatives thấp).
  - mAP50 và mAP50-95
  - + metrics/mAP50(B): Tăng nhanh từ ~0.5 và đạt gần 0.95, phản ánh độ chính xác cao của mô hình ở ngưỡng IoU 0.5.
  - + metrics/mAP50-95(B): Ổn định quanh mức 0.70, thể hiện mô hình vẫn duy trì khả năng nhận diện tốt khi đánh giá ở các ngưỡng IoU nghiêm ngặt hơn.
- **Kết luận**
  - Các hàm tổn thất (loss) trên cả tập huấn luyện và kiểm tra đều giảm ổn định, không có dấu hiệu overfitting.
  - Các chỉ số đánh giá (Precision, Recall, mAP50, mAP50-95) đều đạt mức tốt đến rất tốt.
  - Mô hình Yolo11 đã huấn luyện thể hiện khả năng nhận diện ký tự biển số hiệu quả, chính xác và sẵn sàng để triển khai thực tế hoặc tinh chỉnh (fine-tune) thêm nếu cần.

## Chạy thực nghiệm

Dự đoán trên ảnh thực tế:



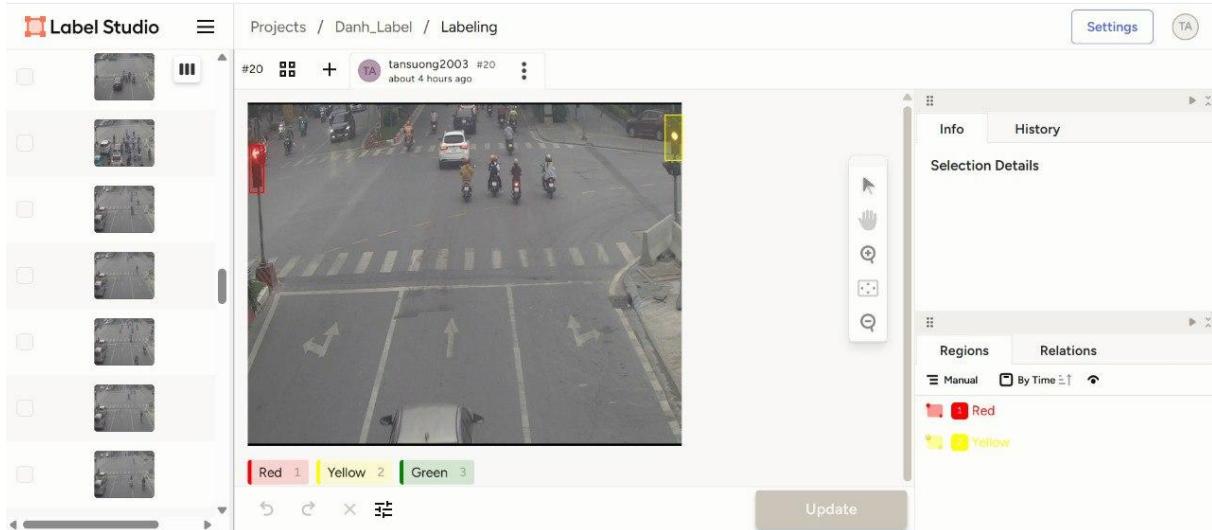
Hình 4.14. Dự đoán trên kết quả thực tế

#### 4.3.3. Phát hiện đèn tín hiệu giao thông

Trong hệ thống giám sát phương tiện vi phạm, việc nhận diện chính xác trạng thái của đèn tín hiệu giao thông (đỏ, vàng, xanh) tại thời điểm vi phạm là một yếu tố quan trọng. Nhóm sử dụng mô hình Yolo11 để phát hiện và phân loại đèn tín hiệu giao thông trong hình ảnh đầu vào, đảm bảo hiệu quả và độ chính xác trong môi trường thực tế.

- Chuẩn bị dữ liệu

Bộ dữ liệu được xây dựng từ các ảnh chụp thực tế tại các giao lộ, trong đó đèn tín hiệu giao thông xuất hiện ở ba trạng thái chính: đỏ, vàng, xanh. Dữ liệu được gán nhãn thủ công theo định dạng chuẩn YOLO.



**Hình 4.15. Giao diện hệ thống Label Studio về tín hiệu đèn giao thông**

- File .txt chứa nhãn lớp và tọa độ bounding box tương ứng từng ảnh.

```
1 0.9486278766598248 0.06593436171524614 0.04466812626704133
0.13186872343049227
```

**Hình 4.16. Tọa độ bounding box vị trí tín hiệu đèn giao thông**

- File mydata.yaml mô tả thông tin cấu trúc dữ liệu như: số lớp (nc=3) và đường dẫn đến tập huấn luyện và kiểm tra.

```
1 path: /kaggle/input/data-dengiaothong/dataset
2 train: /kaggle/input/data-dengiaothong/dataset/images/train
3 val: /kaggle/input/data-dengiaothong/dataset/images/val
4 test: # test images (optional)
5
6 # Classes
7 names:
8   0: green
9   1: red
10  2: yellow
```

**Hình 4.17. File mydata.yaml**

- Huấn luyện mô hình

Mô hình được huấn luyện với kiến trúc **YOLO11x**, sử dụng GPU và tính toán chính xác hỗn hợp (AMP) để tối ưu hiệu suất huấn luyện. Cấu hình huấn luyện như sau:



```
1  from ultralytics import YOLO
2  import torch
3
4  if __name__ == '__main__':
5      torch.multiprocessing.set_start_method('spawn', force=True)
6
7  model = YOLO("/kaggle/input/yolo11x/pytorch/default/1/yolo11x.pt")
8  device = 'cuda' if torch.cuda.is_available() else 'cpu'
9  results = model.train(
10      data="/kaggle/input/mydata-dengiaothong/mydata.yaml",
11      epochs=200,
12      batch=16,
13      device='0,1',
14      workers=8,
15      cache=True,
16      amp=True,
17      imgsz=640,
18      project="/kaggle/working/",
19      name="license_plate",
20      verbose=True
21  )
```

**Hình 4.18. Thông số huấn luyện phát hiện đèn giao thông**

Các tham số chính:

epochs = 200: số vòng huấn luyện.

batch = 16: kích thước lô dữ liệu.

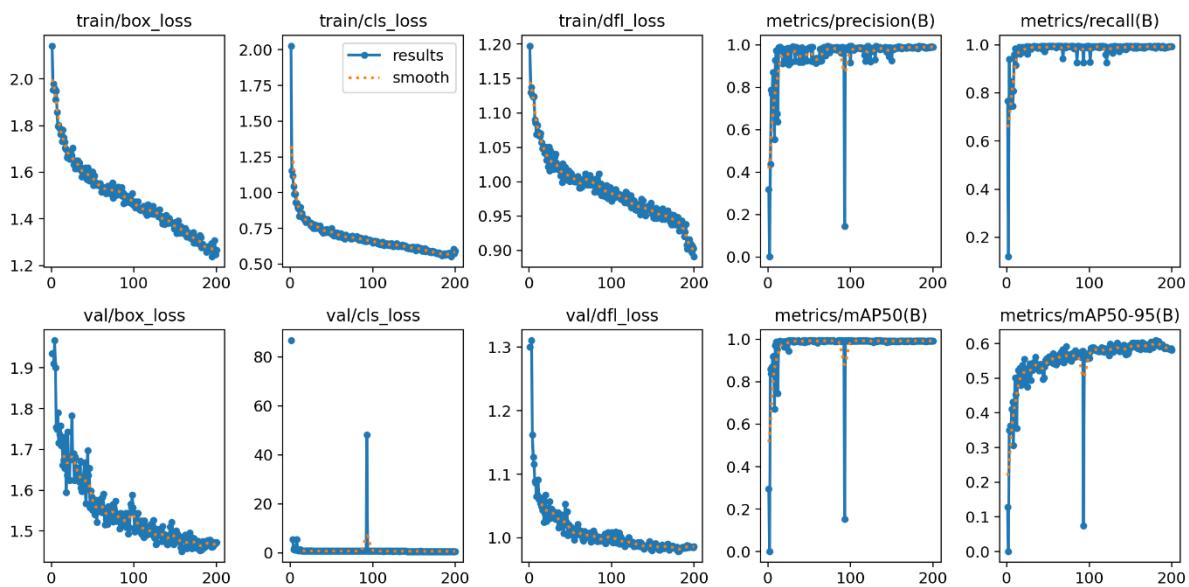
imgsz = 640: kích thước ảnh đầu vào.

device = '0,1': sử dụng hai GPU nếu có sẵn.

amp = True: bật chế độ huấn luyện hỗn hợp để tăng tốc và giảm bộ nhớ.

cache = True: lưu trữ dữ liệu trong RAM giúp tăng tốc độ tải.

- **Kết quả của mô hình**



**Hình 4.19. Kết quả mô hình nhận diện đèn giao thông sau khi huấn luyện**

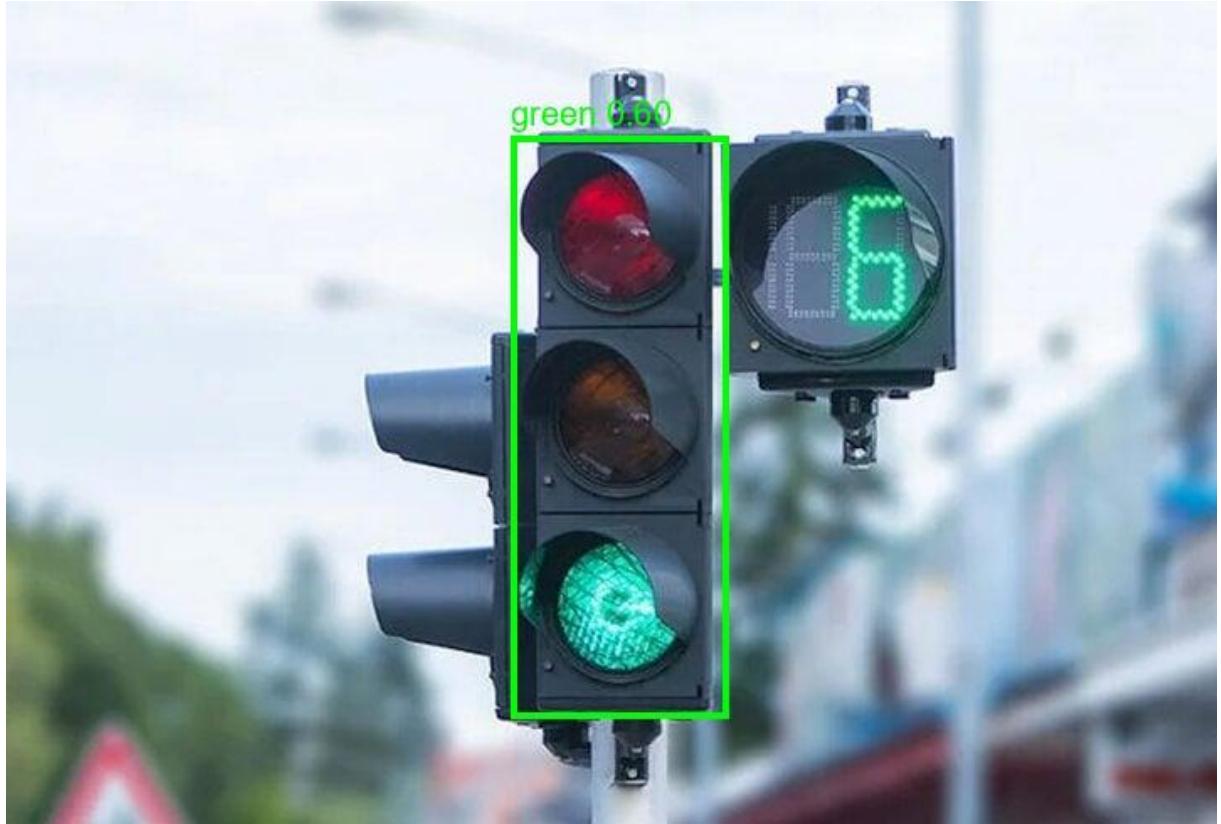
- Các biểu đồ tổn thất (Loss)
  - Train loss
    - + train/box\_loss: Giảm đều từ  $\sim 2.1$  về  $\sim 1.25$  → mô hình học cách định vị bounding box ngày càng chính xác.
    - + train/cls\_loss: Giảm từ  $\sim 2.0$  xuống  $\sim 0.55$  → mô hình cải thiện tốt trong việc phân biệt đối tượng.
    - + train/dfl\_loss: Giảm từ  $\sim 1.2$  về  $\sim 0.9$  → cho thấy mô hình học tốt hơn trong việc dự đoán tọa độ chính xác qua thời gian..
  - Validation loss
    - + val/box\_loss: Giảm từ  $\sim 1.95$  về  $\sim 1.45$  → xu hướng tương tự train/box\_loss, không có dấu hiệu overfitting rõ ràng.
    - + val/cls\_loss: Có spike rất lớn tại vài epoch ( $\sim 80$ ), vượt mức 80, nhưng nhanh chóng trở lại ổn định dưới 2 → có thể do batch lỗi hoặc vấn đề dữ liệu, nhưng mô hình vẫn hồi phục tốt.
    - + val/dfl\_loss: Giảm từ  $\sim 1.3$  về  $\sim 0.95$  → xu hướng ổn định như train/dfl\_loss, thể hiện mô hình tổng quát hóa tốt.
- Độ chính xác mô hình
  - metrics/precision(B): Tăng nhanh từ 0 lên  $\sim 0.98$  và duy trì ổn định → mô hình hiếm khi dự đoán nhầm nhẫn (false positives thấp), dù có 1 outlier rõ tại epoch  $\sim 90$ .

- metrics/recall(B): Tăng đều, đạt ~0.99 → mô hình phát hiện hầu hết các đối tượng cần nhận diện.
  - metrics/mAP50(B): Gần đạt 1.0, ngoại trừ một điểm giảm đột ngột duy nhất → hiệu suất gần như hoàn hảo ở ngưỡng IoU = 0.5.
  - metrics/mAP50-95(B): Đạt khoảng 0.6 và đang còn tăng dần → mô hình vẫn giữ độ chính xác tốt cả ở các ngưỡng IoU cao hơn.
- Kết luận
- Tổn thất trên cả tập train và val đều giảm ổn định, không có overfitting đáng kể.
  - Các chỉ số precision, recall và mAP đều cao, cho thấy mô hình học tốt, tổng quát hóa tốt và đạt độ chính xác cao trong nhiệm vụ nhận diện.

### Kết quả thực nghiệm



Hình 4.20. Kết quả thực nghiệm



Hình 4.21. Kết quả thực nghiệm

#### 4.4. Xây dựng hệ thống nhận diện đèn đỏ theo thời gian thực

##### 4.4.1. Tổng quan về hệ thống

Hệ thống này được thiết kế để:

- Chụp và phân tích video: Lấy hình ảnh từ một cửa sổ scrcpy (thường được dùng để hiển thị màn hình thiết bị như camera giao thông).
- Phát hiện đối tượng: Dùng mô hình YOLO để nhận diện đèn giao thông (xanh, đỏ, vàng) và các phương tiện (xe hơi, xe máy, xe buýt, xe tải, người đi bộ).
- Theo dõi đối tượng: Sử dụng DeepSort để theo dõi các phương tiện qua các khung hình, đảm bảo nhận diện liên tục.
- Phát hiện vi phạm: Xác định phương tiện vượt đèn đỏ dựa trên trạng thái đèn và vị trí của phương tiện so với các vạch kẻ đường.
- Lưu trữ và báo cáo: Lưu thông tin vi phạm vào cơ sở dữ liệu PostgreSQL và cung cấp giao diện để xem lịch sử, xuất báo cáo PDF.

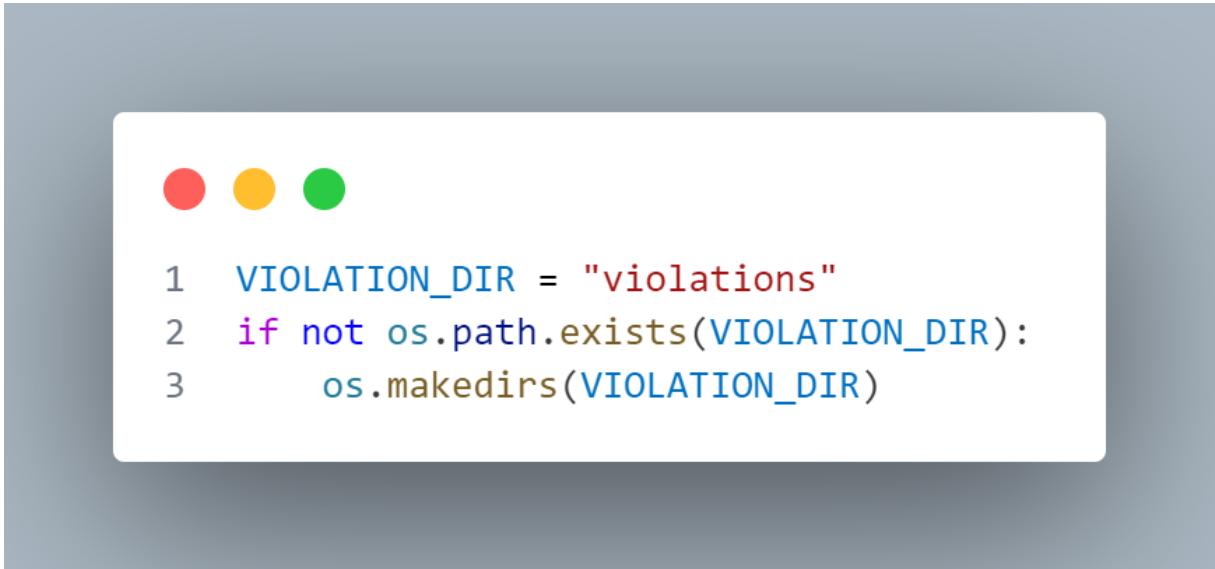
Các thư viện chính:

- Streamlit (st): Tạo giao diện web tương tác, hiển thị video, bảng dữ liệu, và modal.
- OpenCV (cv2): Xử lý hình ảnh, vẽ khung bao, và chuyển đổi định dạng màu.

- NumPy: Tính toán ma trận cho hình ảnh.
- Win32gui, Win32ui, Win32con: Chụp ảnh từ cửa sổ scrcpy trên Windows.
- YOLO (ultralytics): Mô hình AI để phát hiện đèn giao thông và phương tiện.
- DeepSort: Theo dõi các đối tượng qua các khung hình, gán ID duy nhất cho mỗi phương tiện.
- PostgreSQL (psycopg2): Kết nối và lưu trữ dữ liệu vi phạm.
- ReportLab, PIL: Tạo và xử lý báo cáo PDF, bao gồm chèn ảnh bằng chứng.
- Threading: Quản lý luồng để chạy video liên tục mà không làm treo giao diện.
- Streamlit Modal: Tạo cửa sổ bật lên để hiển thị chi tiết vi phạm.
- Base64, io: Xử lý mã hóa ảnh và lưu trữ PDF trong bộ nhớ.

#### 4.4.2. Khởi tạo và cấu hình

- Tạo thư mục lưu ảnh vi phạm



```

1 VIOLATION_DIR = "violations"
2 if not os.path.exists(VIOLATION_DIR):
3     os.makedirs(VIOLATION_DIR)

```

**Hình 4.22. Tạo thư mục lưu ảnh vi phạm**

- Tạo thư mục violations để lưu ảnh của các xe vi phạm nếu thư mục chưa tồn tại.
- Biến toàn cục



```
1 capture_running = False
2 capture_running_lock = threading.Lock()
3 latest_frame = None
4 frame_lock = threading.Lock()
```

Hình 4.23. Biến toàn cục

- capture\_running: Biến boolean để kiểm soát vòng lặp chụp ảnh từ cửa sổ scrcpy.
- capture\_running\_lock: Khóa đồng bộ để đảm bảo an toàn khi nhiều luồng truy cập vào capture\_running.
- latest\_frame: Lưu khung hình mới nhất từ video, được sử dụng để hiển thị và phân tích.
- frame\_lock: Khóa đồng bộ để tránh xung đột khi nhiều luồng truy cập vào latest\_frame.

- Lưu trữ trạng thái



```
1 if "vehicle_positions" not in st.session_state:
2     st.session_state.vehicle_positions = {}
3 if "captured_entities" not in st.session_state:
4     st.session_state.captured_entities = set()
5 if "context_mapping" not in st.session_state:
6     st.session_state.context_mapping = {}
7 if "processing_initialized" not in st.session_state:
8     st.session_state.processing_initialized = False
9 if "lastViolationCheck" not in st.session_state:
10    st.session_state.lastViolationCheck = 0
```

Hình 4.24. Lưu trữ trạng thái

- vehicle\_positions: Lưu trữ thông tin trạng thái của phương tiện (vị trí, thời gian, trạng thái dừng, tuyến đường, v.v.).
- captured\_entities: Tập hợp các ID phương tiện đã được chụp ảnh vi phạm, tránh xử lý trùng lặp.
- context\_mapping: Lưu thông tin ngữ cảnh (vị trí tâm, thời gian, diện tích) để hỗ trợ theo dõi và liên kết các đối tượng qua các khung hình.
- processing\_initialized giúp kiểm soát việc khởi động luồng và tiến trình, tránh xung đột hoặc khởi động lại không cần thiết.
- lastViolation\_check tối ưu hóa giao diện lịch sử bằng cách giới hạn tần suất kiểm tra vi phạm mới.

- Tải mô hình YOLO



```

● ● ●

1 traffic_light_model = YOLO("den_giao_thong.pt").to(device)
2 vehicle_model = YOLO("xe_co.pt").to(device)
3 yolo_LP_detect = YOLO("bien_so.pt").to(device)
4 yolo_license_plate = YOLO("ki_tu_bien_so.pt").to(device)

```

**Hình 4.25. Tải mô hình Yolo**

- traffic\_light\_model: Mô hình YOLO để phát hiện đèn giao thông (xanh, đỏ, vàng).
- vehicle\_model: Mô hình YOLO để phát hiện các loại phương tiện như xe hơi, xe máy, xe buýt, xe tải.
- yolo\_LP\_detect và yolo\_license\_plate: Mô hình YOLO để phát hiện và nhận diện ký tự trên biển số xe.
- Các mô hình được tải từ các file .pt (pre-trained models) và được chuyển sang thiết bị tính toán (device là CPU hoặc GPU).

#### 4.4.3. Kết nối cơ sở dữ liệu PostgreSQL

- Hàm init\_db\_connection()



```
1 def init_db_connection():
2     try:
3         connection = psycopg2.connect(
4             host="localhost",
5             port="5433",
6             database="traffic_violations",
7             user="postgres",
8             password="mysecretpassword"
9         )
10        cursor = connection.cursor()
11        create_table_query = """
12            CREATE TABLE IF NOT EXISTS violations (
13                id SERIAL PRIMARY KEY,
14                id_phuong_tien INTEGER NOT NULL,
15                thoi_gian_vi_pham TIMESTAMP NOT NULL,
16                anh_bang_chung VARCHAR(255) NOT NULL,
17                bien_so_xe VARCHAR(20),
18                loai_vi_pham VARCHAR(50)
19            );
20        """
21        cursor.execute(create_table_query)
22        connection.commit()
23        return connection, cursor
24    except (Exception, Error):
25        return None, None
```

Hình 4.26. Kết nối cơ sở dữ liệu PostgreSQL

- Kết nối đến cơ sở dữ liệu PostgreSQL với thông tin xác thực (host, port, database, user, password).
- Tạo bảng violations với các cột: id (khóa chính), id\_phuong\_tien (ID phương tiện), thoi\_gian\_vi\_pham (thời gian vi phạm), anh\_bang\_chung (đường dẫn ảnh), bien\_so\_xe (biển số), loai\_vi\_pham (loại vi phạm).
- Trả về đối tượng connection và cursor để thực hiện các truy vấn SQL.

- Hàm saveViolationToDb()

```

● ● ●

1 def saveViolation_to_db(cursor, connection, violation):
2     try:
3         insert_query = """
4             INSERT INTO violations (id_phuong_tien, thoi_gian_vi_pham, anh_bang_chung, bien_so_xe, loai_vi_pham)
5             VALUES (%s, %s, %s, %s, %s);
6             """
7         cursor.execute(insert_query, (
8             violation["id_phuong_tien"],
9             violation["thoi_gian_vi_pham"],
10            violation["anh_bang_chung"],
11            violation["bien_so_xe"],
12            violation["loai_vi_pham"]
13        ))
14     connection.commit()
15 except (Exception, Error):
16     connection.rollback()

```

**Hình 4.27. Hàm save\_violation\_to\_db()**

- Chèn dữ liệu vi phạm (ID phương tiện, thời gian, ảnh, biển số, loại vi phạm) vào bảng violations.
- Sử dụng connection.commit() để xác nhận giao dịch.
- Nếu xảy ra lỗi, gọi connection.rollback() để hủy giao dịch, đảm bảo tính toàn vẹn dữ liệu.

#### 4.4.4. Phát hiện và xử lý vi phạm

- Hàm point\_above\_line()

```

● ● ●

1 def point_above_line(x, y, line_start, line_end):
2     x1, y1 = line_start
3     x2, y2 = line_end
4     A = y2 - y1
5     B = x1 - x2
6     C = x2 * y1 - x1 * y2
7     return A * x + B * y + C > 0

```

**Hình 4.28. Hàm point\_above\_line()**

- Sử dụng phương trình đường thẳng  $Ax + By + C = 0$  để xác định vị trí của điểm  $(x, y)$  so với đường thẳng.
- Trả về True nếu điểm nằm phía trên đường, False nếu nằm phía dưới.

## - Hàm save\_violation\_image()

```
● ● ●  
1  def saveViolationImage(frame, box, violation,  
2      track_id, captured_entities, yolo_LP_detect, yolo_license_plate):  
3      if track_id in captured_entities:  
4          return None, "unknown"  
5  
6      x1, y1, x2, y2 = map(int, box)  
7      violation_frame = frame.copy()  
8      color = (0, 0, 255)  
9      cv2.rectangle(violation_frame, (x1, y1), (x2, y2), color, 1)  
10     cv2.putText(violation_frame, f"ID: {track_id}", (x1, y1 - 10),  
11                  cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)  
12  
13     crop_img = violation_frame[y1:y2, x1:x2]  
14     plates = yolo_LP_detect(crop_img, imgsz=640)  
15     license_plate = "unknown"  
16  
17     # Xử lý nhận diện biển số  
18  
19     timestamp = violation["thoi_gian_vi_pham"].replace(":", "-")  
20     filepath = os.path.join(VIOLATION_DIR, f"violation_{track_id}_{timestamp}.jpg")  
21     cv2.imwrite(filepath, violation_frame)  
22     captured_entities.add(track_id)  
23  
24     return filepath, license_plate
```

Hình 4.29. Hàm save\_violation\_image()

- Kiểm tra xem track\_id đã được chụp chưa để tránh trùng lặp.
- Vẽ khung đỏ quanh phương tiện và ghi ID: {track\_id} lên ảnh.
- Cắt vùng chứa phương tiện, sử dụng mô hình YOLO để nhận diện biển số (yolo\_LP\_detect và yolo\_license\_plate).
- Lưu ảnh vi phạm với tên file violation\_{track\_id}\_{timestamp}.jpg.
- Thêm track\_id vào captured\_entities để đánh dấu đã xử lý.

## - Hàm detect\_violation()



```
1 def detectViolation(frame, traffic_lights, tracked_objects, lines,
2                     context_mapping, vehicle_positions,
3                     captured_entities, yolo_LP_detect, yolo_license_plate):
4     violations = []
5     violating_track_ids = set()
6     frame_height, frame_width = frame.shape[:2]
7
8     connection, cursor = init_db_connection()
9     if connection is None or cursor is None:
10         return violations, violating_track_ids
11
12     line_middle, line_left, line_right, line_parallel = lines
13
14     try:
15         traffic_light = next((tl for tl in traffic_lights if tl['label'] == 'Red'), None)
```

**Hình 4.30. Khởi tạo và lấy đèn giao thông**

- Khởi tạo danh sách violations và tập violating\_track\_ids.
- Kết nối cơ sở dữ liệu bằng init\_db\_connection().
- Gán các đường tham chiếu (line\_middle, line\_left, line\_right, line\_parallel) từ lines.
- Tìm đèn giao thông màu đỏ từ danh sách traffic\_lights.

```

● ● ●

1  for obj in tracked_objects:
2      track_id = int(obj['id'][0])
3      x1, y1, x2, y2 = map(int, obj['xyxy'])
4      cls = int(obj['cls'][0])
5      center_x = (x1 + x2) // 2
6      center_y = (y1 + y2) // 2
7      current_time = time.time()
8
9      if track_id not in vehicle_positions:
10          vehicle_positions[track_id] = {
11              'positions': [],
12              'timestamp_middle': None,
13              'light_state': None,
14              'has_stopped': False,
15              'route': None,
16              'crossed_middle': False,
17              'cls': cls,
18              'area_history': []
19          }
20
21      entity_data = vehicle_positions[track_id]
22      entity_data['positions'].append((center_x, center_y, current_time))

```

**Hình 4.31. Theo dõi vị trí phương tiện**

- Lưu vị trí trung tâm (center\_x, center\_y) và thời gian hiện tại (current\_time) của phương tiện.
- Khởi tạo thông tin cho phương tiện mới trong vehicle\_positions (lịch sử vị trí, thời điểm vượt vạch, trạng thái dừng, v.v.).
- Giới hạn lịch sử vị trí để tránh tràn bộ nhớ.

```

● ● ●

1  if not entity_data['crossed_middle']:
2      above_middle = point_above_line(center_x, center_y, line_middle[0], line_middle[1])
3      if len(entity_data['positions']) >= 2:
4          prev_center_x, prev_center_y, _ = entity_data['positions'][-2]
5          prev_above_middle = point_above_line(prev_center_x, prev_center_y, line_middle[0], line_middle[1])
6          if prev_above_middle != above_middle:
7              entity_data['crossed_middle'] = True
8              entity_data['timestamp_middle'] = current_time
9              entity_data['light_state'] = 'Red' if traffic_light else 'Not Red'
10             entity_data['has_stopped'] = False
11             entity_data['route'] = None

```

**Hình 4.32. Kiểm tra vượt vạch giữa**

- Sử dụng point\_above\_line() để kiểm tra vị trí của phương tiện so với vạch giữa.

- Nếu phương tiện chuyển từ phía trên sang phía dưới vạch (hoặc ngược lại), đánh dấu crossed\_middle là True và lưu thời điểm cùng trạng thái đèn.



```

1  if entity_data['crossed_middle'] and not entity_data['has_stopped']:
2      if len(entity_data['positions']) >= 2:
3          first_pos = entity_data['positions'][0]
4          last_pos = entity_data['positions'][-1]
5          time_diff = last_pos[2] - first_pos[2]
6          x_diff = abs(last_pos[0] - first_pos[0])
7          y_diff = abs(last_pos[1] - first_pos[1])
8          if time_diff >= 2.0 and x_diff < 10 and y_diff < 10:
9              entity_data['has_stopped'] = True

```

**Hình 4.33. Kiểm tra trạng thái dừng**

- So sánh vị trí đầu tiên và cuối cùng trong lịch sử vị trí.
- Nếu thời gian giữa hai vị trí  $\geq 2$  giây và khoảng cách di chuyển nhỏ ( $< 10$  pixel), phương tiện được coi là đã dừng.



```

1  if entity_data['crossed_middle'] and entity_data['route'] is None:
2      above_parallel = point_above_line(center_x, center_y, line_parallel[0], line_parallel[1])
3      above_left = point_above_line(center_x, center_y, line_left[0], line_left[1])
4      above_right = point_above_line(center_x, center_y, line_right[0], line_right[1])
5      if len(entity_data['positions']) >= 2:
6          prev_center_x, prev_center_y, _ = entity_data['positions'][-2]
7          prev_above_parallel = point_above_line(prev_center_x, prev_center_y, line_parallel[0], line_parallel[1])
8          prev_above_left = point_above_line(prev_center_x, prev_center_y, line_left[0], line_left[1])
9          prev_above_right = point_above_line(prev_center_x, prev_center_y, line_right[0], line_right[1])
10         if prev_above_parallel != above_parallel:
11             entity_data['route'] = 'straight'
12         elif prev_above_left != above_left:
13             entity_data['route'] = 'left'
14         elif prev_above_right != above_right:
15             entity_data['route'] = 'right'

```

**Hình 4.34. Xác định tuyến đường**

- Kiểm tra vị trí của phương tiện so với các vạch line\_parallel, line\_left, line\_right.
- Nếu phương tiện vượt qua một vạch, gán tuyến đường tương ứng (straight, left, right).

```

● ● ●

1  if entity_data['crossed_middle'] and entity_data['route'] is not None \
2      and track_id not in captured_entities:
3      timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
4      violation = None
5      if cls in [2, 3, 5, 7]:
6          if (entity_data['route'] in ['straight', 'left']) and \
7              entity_data['light_state'] == 'Red' and not entity_data['has_stopped']:
8              violation_type = "Vượt đèn đỏ đi thẳng" \
9                  if entity_data['route'] == 'straight' else "Vượt đèn đỏ rẽ trái"
10             filepath, license_plate = saveViolationImage(
11                 frame, [x1, y1, x2, y2], {"thoi_gian_vipham": timestamp},
12                 track_id, captured_entities, yolo_LP_detect, yolo_license_plate
13             )
14             violation = {
15                 "id_phuong_tien": track_id,
16                 "thoi_gian_vipham": timestamp,
17                 "anh_bang_chung": filepath,
18                 "bien_so_xe": license_plate,
19                 "loai_vipham": violation_type
20             }
21         elif entity_data['route'] == 'right':
22             continue
23         if violation and violation["anh_bang_chung"]:
24             saveViolationToDb(cursor, connection, violation)
25             violations.append(violation)
26             violating_track_ids.add(track_id)
27             captured_entities.add(track_id)

```

**Hình 4.35. Kiểm tra vi phạm**

- Kiểm tra phương tiện có vượt đèn đỏ (đi thẳng/rẽ trái, đèn đỏ, không dừng).
- Lưu ảnh vi phạm và nhận diện biển số bằng save\_violation\_image().
- Lưu thông tin vi phạm vào cơ sở dữ liệu và thêm vào danh sách violations.

```

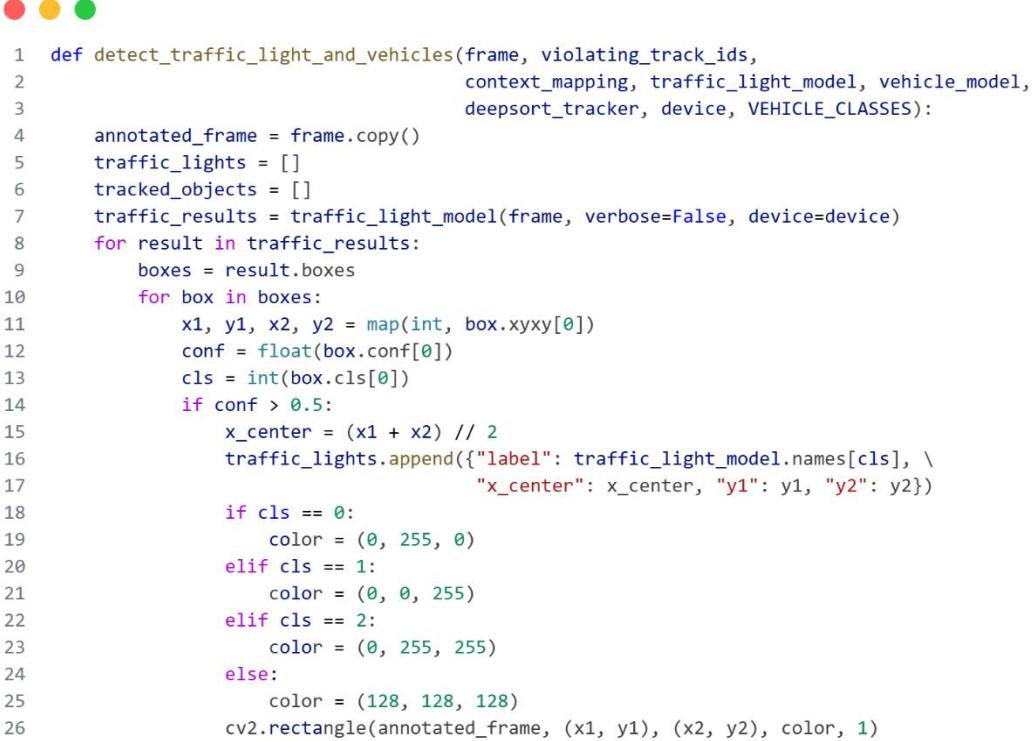
● ● ●

1  except Exception as e:
2      if connection:
3          connection.rollback()
4          print(f'Lỗi trong quá trình phát hiện vi phạm: {str(e)}')
5  finally:
6      if cursor:
7          cursor.close()
8      if connection:
9          connection.close()

```

**Hình 4.36. Xử lý lỗi và đóng kết nối**

- Bắt lỗi bằng try-except, rollback giao dịch nếu có lỗi.
  - Đóng cursor và connection trong khôi finally để giải phóng tài nguyên.
- Hàm detect\_traffic\_light\_and\_vehicles



```

1 def detect_traffic_light_and_vehicles(frame, violating_track_ids,
2                                     context_mapping, traffic_light_model, vehicle_model,
3                                     deepsort_tracker, device, VEHICLE_CLASSES):
4     annotated_frame = frame.copy()
5     traffic_lights = []
6     tracked_objects = []
7     traffic_results = traffic_light_model(frame, verbose=False, device=device)
8     for result in traffic_results:
9         boxes = result.bboxes
10        for box in boxes:
11            x1, y1, x2, y2 = map(int, box.xyxy[0])
12            conf = float(box.conf[0])
13            cls = int(box.cls[0])
14            if conf > 0.5:
15                x_center = (x1 + x2) // 2
16                traffic_lights.append({"label": traffic_light_model.names[cls], \
17                                       "x_center": x_center, "y1": y1, "y2": y2})
18                if cls == 0:
19                    color = (0, 255, 0)
20                elif cls == 1:
21                    color = (0, 0, 255)
22                elif cls == 2:
23                    color = (0, 255, 255)
24                else:
25                    color = (128, 128, 128)
26                cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), color, 1)

```

**Hình 4.37. Khởi tạo và phát hiện đèn giao thông**

- Sử dụng traffic\_light\_model để phát hiện đèn giao thông.
- Lưu thông tin đèn (nhãn, vị trí) vào traffic\_lights nếu độ tin cậy (conf) > 0.5.
- Vẽ khung màu tương ứng (xanh, đỏ, vàng) lên khung hình.

```

● ● ●

1 vehicle_results = vehicle_model(frame, classes=VEHICLE_CLASSES, verbose=False, device=device)
2 detections = []
3 boxes = []
4 scores = []
5 for result in vehicle_results:
6     boxes.extend([list(map(int, box.xyxy[0])) for box in result.boxes])
7     scores.extend([float(box.conf[0]) for box in result.boxes])
8 if boxes:
9     indices = nms_boxes([[x1, y1, x2, y2] for x1, y1, x2, y2 in boxes], scores)
10    for idx in indices:
11        x1, y1, x2, y2 = boxes[idx]
12        conf = scores[idx]
13        cls = int(vehicle_results[0].boxes.cls[idx])
14        if conf > 0.5:
15            detections.append(([x1, y1, x2 - x1, y2 - y1], conf, cls))

```

**Hình 4.38. Phát hiện phương tiện**

- Sử dụng vehicle\_model để phát hiện phương tiện với các lớp được chỉ định (VEHICLE\_CLASSES).
- Áp dụng Non-Maximum Suppression (NMS) để loại bỏ các khung trùng lặp.
- Lưu các phát hiện (vị trí, độ tin cậy, lớp) vào detections.

```

● ● ●

1 tracks = deepsort_tracker.update_tracks(detections, frame=frame)
2 for track in tracks:
3     if not track.is_confirmed() or track.time_since_update > 1:
4         continue
5     track_id = track.track_id
6     ltrb = track.to_ltrb()
7     x1, y1, x2, y2 = map(int, ltrb)
8     cls = track.det_class
9     center_x = (x1 + x2) // 2
10    center_y = (y1 + y2) // 2
11    area = (x2 - x1) * (y2 - y1)
12    color = (255, 0, 0) if track_id in violating_track_ids else (0, 255, 0)
13    label = f"ID: {track_id} (Violation)" if track_id in violating_track_ids else f"ID: {track_id}"
14    cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), color, 1)
15    cv2.putText(annotated_frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
16    tracked_objects.append({
17        'id': [track_id],
18        'xyxy': [x1, y1, x2, y2],
19        'conf': [track.det_conf],
20        'cls': [cls]
21    })

```

**Hình 4.39. Theo dõi với DeepSort**

- Cập nhật các track bằng deepsort\_tracker.update\_tracks().
- Vẽ khung và ghi ID lên khung hình cho các track được xác nhận.
- Lưu thông tin track (ID, vị trí, độ tin cậy, lớp) vào tracked\_objects.

```

● ○ ●

1 if track_id not in context_mapping:
2     context_mapping[track_id] = {
3         'center': (center_x, center_y),
4         'timestamp': current_time,
5         'area': area,
6         'cls': cls
7     }
8 else:
9     prev_context = context_mapping[track_id]
10    prev_center = prev_context['center']
11    prev_time = prev_context['timestamp']
12    prev_area = prev_context['area']
13    distance = calculate_distance((center_x, center_y), prev_center)
14    time_diff = current_time - prev_time
15    area_ratio = min(area, prev_area) / max(area, prev_area)
16    if distance < DISTANCE_THRESHOLD and time_diff < TIME_THRESHOLD and area_ratio > 0.7:
17        context_mapping[track_id] = {
18            'center': (center_x, center_y),
19            'timestamp': current_time,
20            'area': area,
21            'cls': cls
22        }

```

**Hình 4.40. Cập nhật ngữ cảnh**

- Lưu ngữ cảnh mới cho track nếu chưa có trong context\_mapping.
- Cập nhật ngữ cảnh nếu khoảng cách, thời gian và tỷ lệ diện tích thỏa mãn các ngưỡng (DISTANCE\_THRESHOLD, TIME\_THRESHOLD, 0.7).

```

● ○ ●

1 context_mapping = {k: v for k, v in context_mapping.items() \
                    if v['timestamp'] > current_time - TIME_THRESHOLD}

```

**Hình 4.41. Dọn dẹp ngữ cảnh cũ**

- Lọc context\_mapping để chỉ giữ các track có thời gian cập nhật trong vòng 5 giây (TIME\_THRESHOLD).

```

1 frame_height, frame_width = frame.shape[:2]
2 line_middle = [(150, 265), (650, 260)]
3 line_parallel = [(160, 80), (530, 80)]
4 line_left = [(130, 10), (130, 130)]
5 line_right = [(620, 40), (620, 120)]
6 cv2.line(annotated_frame, line_middle[0], line_middle[1], (0, 0, 255), 1)
7 cv2.line(annotated_frame, line_parallel[0], line_parallel[1], (255, 0, 0), 1)
8 cv2.line(annotated_frame, line_left[0], line_left[1], (0, 255, 0), 1)
9 cv2.line(annotated_frame, line_right[0], line_right[1], (255, 255, 0), 1)
10 lines = (line_middle, line_left, line_right, line_parallel)

```

**Hình 4.42. Định nghĩa vạch kẻ đường**

- Định nghĩa tọa độ các vạch (line\_middle, line\_parallel, line\_left, line\_right).
- Vẽ các vạch lên khung hình với màu sắc khác nhau để dễ phân biệt.

- Hàm capture\_window

```

1 def capture_window(window_title):
2     try:
3         hwnd = win32gui.FindWindow(None, window_title)
4         if not hwnd:
5             return None, "Không tìm thấy cửa sổ scrcpy với tiêu đề: " + window_title
6         left, top, right, bottom = win32gui.GetClientRect(hwnd)
7         width = right - left
8         height = bottom - top
9         region_left, region_top, region_right, region_bottom = 0, 128, 414, 870

```

**Hình 4.43. Tìm cửa sổ scrcpy**

- Sử dụng win32gui.FindWindow() để tìm cửa sổ scrcpy với tiêu đề window\_title.
- Lấy kích thước vùng chụp (tọa độ: 0, 128, 414, 870).



```
1 hwnd_dc = win32gui.GetWindowDC(hwnd)
2 mfc_dc = win32ui.CreateDCFromHandle(hwnd_dc)
3 compatible_dc = mfc_dc.CreateCompatibleDC()
4 bitmap = win32ui.CreateBitmap()
5 bitmap.CreateCompatibleBitmap(mfc_dc, capture_width, capture_height)
6 compatible_dc.SelectObject(bitmap)
7 compatible_dc.BitBlt((0, 0), (capture_width, capture_height), mfc_dc, \
                      (region_left, region_top), win32con.SRCCOPY)
8 bitmap_bits = bitmap.GetBitmapBits(True)
```

Hình 4.44. Chụp ảnh từ cửa sổ

- Tạo các đối tượng DC (hwnd\_dc, mfc\_dc, compatible\_dc) để xử lý đồ họa.
- Tạo bitmap và sao chép dữ liệu hình ảnh từ vùng chụp vào bitmap.



```
1 img = np.frombuffer(bitmap_bits, dtype=np.uint8) \
      .reshape((capture_height, capture_width, 4))
2 img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
3 img = cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)
4 win32gui.DeleteObject(bitmap.GetHandle())
5 compatible_dc.DeleteDC()
6 mfc_dc.DeleteDC()
7 win32gui.ReleaseDC(hwnd, hwnd_dc)
8 return img, None
```

Hình 4.45. Chuyển đổi và giải phóng tài nguyên

- Chuyển đổi dữ liệu bitmap thành mảng NumPy, đổi định dạng màu từ BGRA sang BGR và xoay ảnh 90 độ ngược chiều kim đồng hồ.
- Giải phóng tài nguyên (bitmap, DC) để tránh rò rỉ bộ nhớ.



```
1 except Exception as e:  
2     return None, f'Lỗi khi chụp: {str(e)}"
```

Hình 4.46. Xử lý lỗi

- Bắt lỗi trong quá trình chụp và trả về thông báo lỗi cụ thể.

#### 4.4.5. Hiển thị lịch sử vi phạm



```
1 def show_history():  
2     st.subheader("📋 Lịch sử vi phạm")  
3     modal = Modal("📋 Báo cáo vi phạm", key="bao_cao_popup")  
4     if "selected_record" not in st.session_state:  
5         st.session_state.selected_record = None  
6         st.session_state.full_img_path = None  
7     connection, cursor = init_db_connection()
```

Hình 4.47. Khởi tạo giao diện lịch sử

- Tạo tiêu đề "Lịch sử vi phạm" bằng st.subheader.
- Khởi tạo modal để hiển thị báo cáo chi tiết của vi phạm.
- Khởi tạo biến trạng thái để lưu bản ghi và đường dẫn ảnh được chọn.



```

1  st.subheader("🔍 Tìm kiếm vi phạm")
2  with st.form(key="search_form"):
3      col1, col2 = st.columns(2)
4      with col1:
5          search_id = st.text_input("ID phương tiện", "")
6          search_plate = st.text_input("Biển số xe", "")
7      with col2:
8          start_date = st.date_input("Từ ngày", value=None, format="YYYY-MM-DD")
9          end_date = st.date_input("Đến ngày", value=None, format="YYYY-MM-DD")
10         submit_button = st.form_submit_button("Tìm kiếm")

```

**Hình 4.48. Form tìm kiếm**

- Tạo form với các trường nhập: ID phương tiện, biển số xe, ngày bắt đầu và ngày kết thúc.
- Sử dụng st.columns để sắp xếp giao diện thành hai cột.



```

1  cursor.execute(count_query, query_params)
2  total_records = cursor.fetchone()[0]
3  st.subheader("📖 Phân trang")
4  records_per_page = st.selectbox("Số vi phạm mỗi trang", [10, 20, 50], index=0)
5  total_pages = (total_records + records_per_page - 1) // records_per_page
6  offset = (st.session_state.current_page - 1) * records_per_page
7  paginated_query = f"{select_query} LIMIT %s OFFSET %s"
8  cursor.execute(paginated_query, query_params + [records_per_page, offset])
9  records = cursor.fetchall()

```

**Hình 4.49. Phân trang**

- Đếm tổng số bản ghi vi phạm bằng count\_query.
- Cho phép người dùng chọn số bản ghi mỗi trang (10, 20, 50).
- Tính tổng số trang và truy vấn bản ghi theo trang hiện tại (current\_page).

```

● ● ●

1  for record in records:
2      col1, col2, col3, col4 = st.columns([1, 2, 1, 1])
3      with col1:
4          st.markdown(f"**ID:** {record[0]}")
5          st.markdown(f"**ID phương tiện:** {record[1]}")
6      with col2:
7          st.markdown(f"**Thời gian:** {record[2].strftime('%Y-%m-%d %H:%M:%S')} \
8                      if record[2] else 'N/A'")
9          st.markdown(f"**Biển số xe:** {record[4] if record[4] else 'Không có'}")
10         st.markdown(f"**Loại vi phạm:** {record[5] if record[5] else 'Không xác định'}")
11     with col3:
12         img_path = record[3]
13         img_path = img_path.replace('/', os.sep)
14         current_dir = os.path.dirname(os.path.abspath(__file__))
15         full_img_path = os.path.normpath(os.path.join(current_dir, img_path))
16         if os.path.exists(full_img_path):
17             try:
18                 img_data = cv2.imread(full_img_path)
19                 if img_data is not None:
20                     img_data = cv2.cvtColor(img_data, cv2.COLOR_BGR2RGB)
21                     img_data = cv2.resize(img_data, (200, 150))
22                     st.image(img_data, channels="RGB", use_container_width=True)

```

**Hình 4.50. Hiển thị danh sách vi phạm**

- Hiển thị thông tin vi phạm (ID, thời gian, biển số, loại vi phạm) trong các cột.
- Đọc và hiển thị ảnh bằng chứng, chuyển đổi từ BGR sang RGB để hiển thị đúng màu.

```

● ● ●

1  except Exception as error:
2      st.error(f'Lỗi khi truy vấn lịch sử vi phạm: {error}')
3  finally:
4      if cursor:
5          cursor.close()
6      if connection:
7          connection.close()

```

**Hình 4.51. Xử lý và đóng kết nối**

- Bắt lỗi trong quá trình truy vấn và hiển thị thông báo lỗi.
- Đóng cursor và connection để giải phóng tài nguyên.

#### 4.4.6. Tạo báo cáo PDF

```
● ● ●  
1 def create_pdf_report(record, full_img_path):  
2     buffer = io.BytesIO()  
3     doc = SimpleDocTemplate(buffer, pagesize=A4)  
4     styles = getSampleStyleSheet()  
5     try:  
6         pdfmetrics.registerFont(TTFont('DejaVuSansExtraLight', \  
7                                         'dejavu-sans.extralight.ttf'))  
8         styles['Title'].fontName = 'DejaVuSansExtraLight'  
9         styles['Normal'].fontName = 'DejaVuSansExtraLight'  
10    except Exception as e:  
11        st.warning(f"Không thể tải font DejaVuSansExtraLight: {str(e)} \  
12                  . Sử dụng font mặc định (Times-Roman).")  
13        styles['Title'].fontName = 'Times-Roman'  
14        styles['Normal'].fontName = 'Times-Roman'  
15    elements = []
```

Hình 4.52. Khởi tạo PDF

- Tạo tài liệu PDF bằng SimpleDocTemplate với kích thước A4.
- Cố gắng đăng ký font DejaVuSansExtraLight, nếu thất bại thì dùng font mặc định

```
● ● ●  
1 elements.append(Paragraph(f"Báo cáo vi phạm giao thông - ID: \  
2                               {record[0]}", styles['Title']))  
3 elements.append(Spacer(1, 12))  
4 elements.append(Paragraph(f"ID phương tiện: {record[1]}", \  
5                               styles['Normal']))  
6 elements.append(Paragraph(f"Thời gian vi phạm: {record[2].strftime('%Y-%m-%d %H:%M:%S')} \  
7                               if record[2] else 'N/A'", styles['Normal']))  
8 elements.append(Paragraph(f"Biển số xe: {record[4]} \  
9                               if record[4] else 'Không có'", styles['Normal']))  
10 elements.append(Paragraph(f"Loại vi phạm: {record[5]} \  
11                               if record[5] else 'Không xác định'", styles['Normal']))
```

Hình 4.53. Thêm nội dung văn bản

- Thêm tiêu đề và các thông tin chi tiết của vi phạm vào danh sách elements để hiển thị trong PDF.

```
● ● ●  
1 if os.path.exists(full_img_path):  
2     try:  
3         img = PILImage.open(full_img_path)  
4         img_width, img_height = img.size  
5         aspect = img_height / float(img_width)  
6         target_width = 150 * mm  
7         target_height = target_width * aspect  
8         if target_height > 100 * mm:  
9             target_height = 100 * mm  
10            target_width = target_height / aspect  
11            elements.append(Image(full_img_path, width=target_width, height=target_height))  
12        except Exception as e:  
13            elements.append(Paragraph(f'Lỗi khi chèn ảnh: {str(e)}", styles['Normal']))
```

Hình 4.54. Thêm ảnh bằng chứng

- Kiểm tra xem ảnh bằng chứng có tồn tại không.
- Tính toán kích thước ảnh để giữ tỷ lệ và thêm vào PDF bằng Image.

#### 4.4.7. Giao diện streamlit

- Sidebar điều hướng

```
● ● ●  
1 page = st.sidebar.radio("Chọn chức năng", ["Camera", "Lịch sử"])  
2 if page == "Camera":  
3     show_camera()  
4 elif page == "Lịch sử":  
5     show_history()
```

Hình 4.55. Sidebar điều hướng

## Chương 5: KẾT QUẢ VÀ ĐÁNH GIÁ

### 5.1. Kiểm tra độ chính xác

Hệ thống giám sát vượt đèn đỏ được thử nghiệm với dữ liệu video thực tế mô phỏng luồng giao thông tại một giao lộ, sử dụng luồng chiếu từ ứng dụng scrcpy. Hai mô hình YOLO (den\_giao\_thong.pt và xe\_co.pt) được sử dụng để nhận diện đèn giao thông và phương tiện. Kết quả kiểm tra được ghi nhận theo quan sát và đánh giá thủ công trên một tập khung hình đại diện:

- **Nhận diện đèn giao thông:**

- Hệ thống nhận diện tốt hai trạng thái đèn (đỏ, xanh) trong phần lớn khung hình thử nghiệm.
- Một số khung hình bị nhận sai do điều kiện ánh sáng thay đổi.
- Tình huống đèn vàng do gần với đèn đỏ nên hay bị nhận diện lầm.

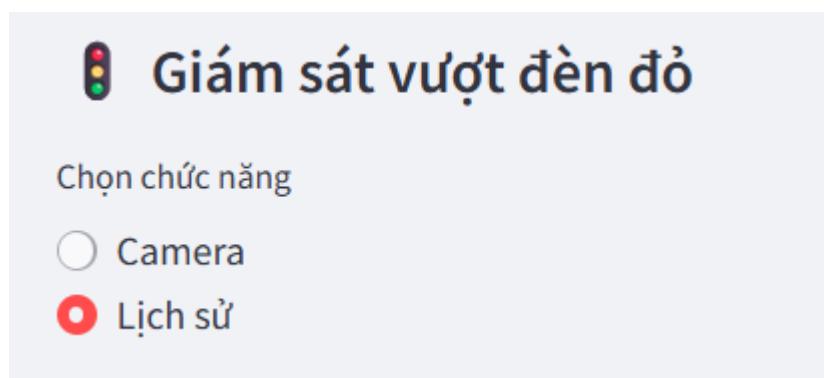
- **Nhận diện và theo dõi phương tiện:**

- Mô hình nhận diện ổn định các phương tiện thông dụng như xe máy, ô tô, xe buýt, xe tải trong đa số tình huống ban ngày.
- Tốc độ và độ chính xác giảm khi mật độ giao thông cao, đặc biệt khi các phương tiện chồng lấn nhau hoặc xảy ra chuyển động nhanh liên tục.
- Chức năng theo dõi (tracking) duy trì được ID cho các phương tiện trong thời gian ngắn, nhưng có thể mất ID hoặc gán sai trong tình huống bị che khuất hoặc thay đổi hướng đột ngột.

- **Phát hiện vi phạm:**

- Hệ thống vận hành dựa trên logic: xe vượt qua vạch giữa (line\_middle) và các line khác trong khi đèn đang đỏ.
- Trong các thử nghiệm thủ công (ghi hình và đổi chiếu bằng mắt), hệ thống phát hiện được hầu hết các tình huống vượt đèn.
- Một số trường hợp sai sót vẫn xảy ra, do:
  - Tọa độ xe không được xác định chính xác khi xe chuyển động nhanh.
  - Mật độ xe đông nên object tracking không đúng làm bỏ sót xe vi phạm

## 5.2. Thực nghiệm thực tế

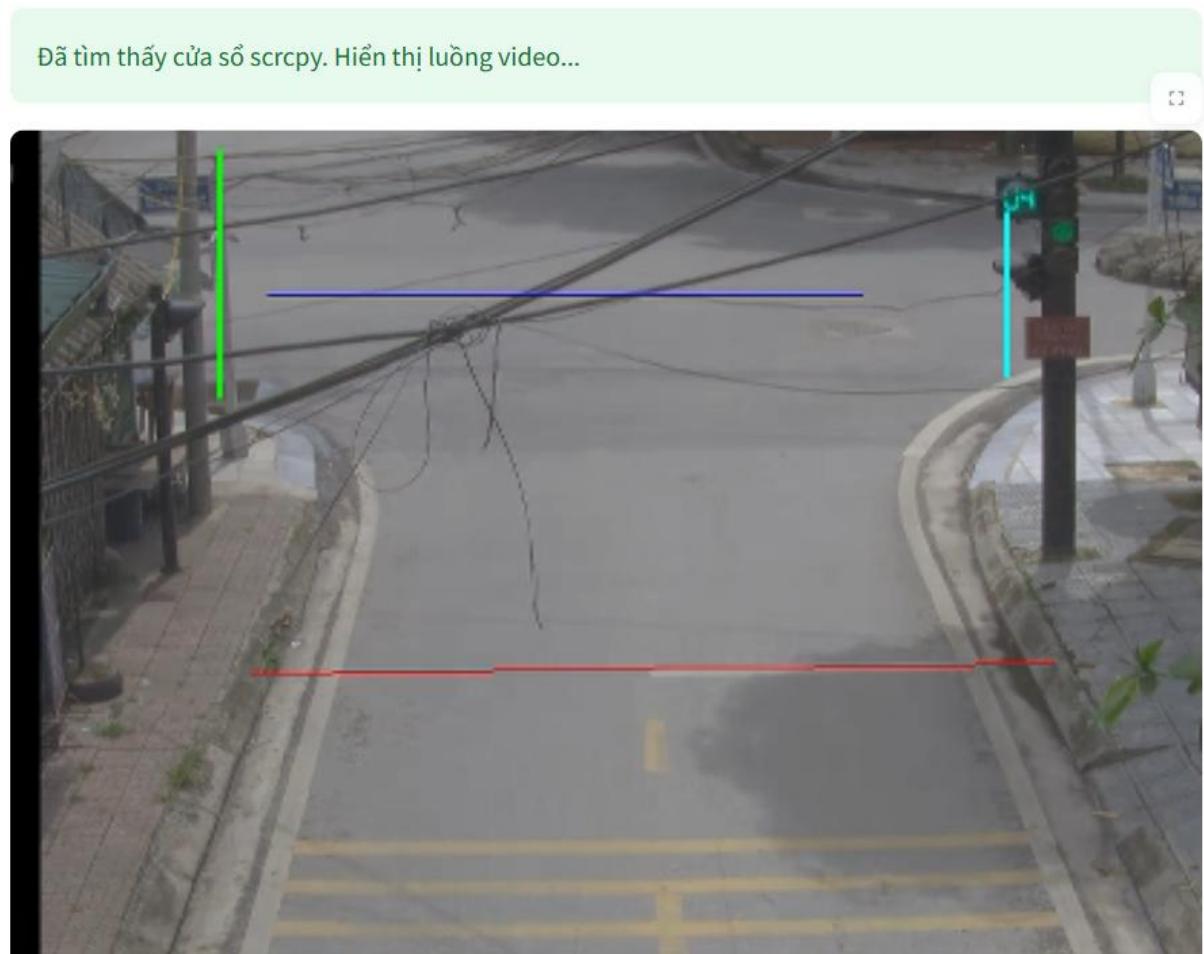


Hình 5.1. Giao diện taskbar để chọn chức năng

### 🎥 Camera giám sát ↗

- ⌚ Thời gian hiện tại: 01:51:02 PM, Monday, 16/06/2025
- 📍 Địa chỉ: GT\_Ngã 4 đèn đỏ Sân vận động\_4 - Thành phố Điện Biên Phủ

Hình 5.2. Giao diện thông tin camera giao thông tại Thành phố Điện Biên Phủ



Hình 5.3. Hình ảnh camera được hiển thị lên giao diện

## Lịch sử vi phạm

### Tìm kiếm vi phạm ↗

ID phương tiện	Từ ngày
<input type="text"/>	YYYY-MM-DD
Biển số xe	Đến ngày
<input type="text"/>	YYYY-MM-DD
<input type="button" value="Tìm kiếm"/>	

Hình 5.4. Giao diện lịch sử vi phạm

## Phân trang

Số vi phạm mỗi trang

10

Trang trước

Trang 1 / 3

Trang sau

ID: 29

Thời gian: 2025-06-08 15:48:17



Xuất báo cáo

ID phương tiện: 24

Biển số xe: XXXXX

Loại vi phạm: Vượt đèn đỏ rẽ trái

ID: 28

Thời gian: 2025-06-08 14:12:52



Xuất báo cáo

ID phương tiện:

283

Biển số xe: XXXXX

Loại vi phạm: Vượt đèn đỏ đi thẳng

ID: 27

Thời gian: 2025-06-08 14:12:51



Xuất báo cáo

ID phương tiện:

282

Biển số xe: XXXXX

Loại vi phạm: Vượt đèn đỏ đi thẳng

Hình 5.5. Giao diện hiển thị lịch sử vi phạm chi tiết



## Báo cáo vi phạm

X

### ⚑ Báo cáo vi phạm - ID: 410

⚠ Vi phạm: Vượt đèn đỏ

➡ Phương tiện: 167

➡ Biển số: 27A-06110

⌚ Thời gian: 2025-06-17 21:33:44



## Báo cáo vi phạm

X



⬇ Tải PDF

Hình 5.6. Báo cáo vi phạm

## Báo cáo vi phạm giao thông - ID: 410

ID phương tiện: 167

Thời gian vi phạm: 2025-06-17 21:33:44

Biển số xe: 27A-06110

Loại vi phạm: Vượt đèn đỏ



Hình 5.7. PDF chi tiết về báo cáo vi phạm

## Chương 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 6.1. Kết luận

Hệ thống giám sát vượt đèn đỏ bước đầu đã đáp ứng được các mục tiêu cơ bản trong việc phát hiện và ghi nhận hành vi vi phạm giao thông tại giao lộ:

- Nhận diện đèn giao thông và phương tiện: Hệ thống sử dụng hai mô hình YOLO để phát hiện đèn tín hiệu và các phương tiện phổ biến như xe máy, ô tô, xe tải. Qua các thử nghiệm thực tế trên video mô phỏng từ giao lộ, hệ thống có thể nhận diện được phần lớn các đối tượng trong điều kiện ánh sáng ban ngày và mật độ giao thông vừa phải.
- Lưu trữ và giao diện: Vi phạm được lưu trữ trực tiếp vào cơ sở dữ liệu PostgreSQL, đồng thời hiển thị lại qua giao diện Streamlit giúp dễ dàng kiểm tra và truy xuất thông tin.
- Tính ổn định: Trong quá trình chạy thử nghiệm, hệ thống hoạt động ổn định, không bị crash và có thể ghi nhận nhiều trường hợp vi phạm một cách liên tục.

### 6.2. Hướng phát triển

Trong quá trình triển khai và thử nghiệm, hệ thống vẫn còn một số hạn chế, từ đó gợi mở các hướng phát triển sau:

- **Cải thiện độ chính xác trong điều kiện khó khăn:** Nhận diện đèn giao thông, biến số có thể bị sai lệch trong điều kiện ánh sáng yếu, có ánh sáng chói từ đèn xe hoặc đèn quảng cáo. Ngoài ra, phương tiện bị khuất hoặc chồng lấn nhau khiến mô hình nhận diện và theo dõi ID không chính xác.
- **Tăng tính chính xác trong logic phát hiện vi phạm:** Thuật toán phát hiện xe vượt đèn đỏ đôi khi chưa nhận diện đúng ranh giới và thời điểm vi phạm, đặc biệt khi xe di chuyển nhanh hoặc có độ trễ giữa nhận diện đèn và nhận diện xe. Hướng cải tiến là đồng bộ hóa các tín hiệu và vị trí một cách chính xác hơn theo từng khung hình.
- **Tăng tính linh hoạt cho người dùng cuối:** Hiện tại vùng giám sát (line\_left, line\_middle, line\_right) được gán cố định trong code. Cần phát triển tính năng điều chỉnh vùng giám sát trực tiếp trên giao diện giúp người dùng dễ cấu hình với từng giao lộ khác nhau.
- **Cải thiện nhận dạng phương tiện:** Trong một số trường hợp, hệ thống không phân biệt được rõ các loại phương tiện giống nhau (ví dụ: xe tải nhỏ và ô tô

SUV). Việc huấn luyện lại mô hình với bộ nhãn chi tiết hơn sẽ giúp phân loại chính xác hơn.

## TÀI LIỆU THAM KHẢO

1. VNPTAI. (2025, January 18). Computer vision là gì? VNPTAI. <https://vnptai.io/vi/blog/detail/computer-vision-la-gi>
2. PBCQuoc. (2019). YOLO: Thuật toán phát hiện đối tượng nổi bật. PBCQuoc. <https://pbcquoc.github.io/yolo/>
3. MCTT. (2025). YOLO là gì?. MCTT. <https://mctt.vn/yolo-la-gi>
4. VinBigData. (2023, April 13). YOLOv7: Thuật toán phát hiện đối tượng có gì mới? VinBigData. [https://vinbigdata.com/kham-pha/yolo-v7-thuat-toan-phat-hien-doi-tuong-co-gi-moi.html#YOLO\\_hoat\\_dong\\_nhu\\_the nao\\_Kien\\_truc\\_YOLO](https://vinbigdata.com/kham-pha/yolo-v7-thuat-toan-phat-hien-doi-tuong-co-gi-moi.html#YOLO_hoat_dong_nhu_the nao_Kien_truc_YOLO)
5. Teky. (2023, October 25). OpenCV là gì?. Teky. <https://teky.edu.vn/blog/opencv-la-gi/>
6. Viblo. (2021, December 25). SORT (DeepSORT): Một góc nhìn về object tracking - Phần 1. Viblo. <https://viblo.asia/p/sort-deep-sort-mot-goc-nhin-ve-object-tracking-phan-1-Az45bPooZxY>
7. Thị Giác Máy Tính. (2024). Tổng hợp data xử lý ảnh. <https://thigiacmaytinh.com/tai-nguyen-xu-ly-anh/tong-hop-data-xu-ly-anh/>
8. Axle Networks. (2025). Intelligent transportation systems for public safety. Axle Networks Resources. <https://axlenetworks.com.au/resources/intelligent-transportationsystems-for-public-safety/>
9. Brownlee, J. (2021, January 27). Object recognition with deep learning. Machine Learning Mastery. <https://machinelearningmastery.com/object-recognition-with-deep-learning/>
10. Deep learning for license plate number recognition: A survey. (2024). IEEE Xplore. <https://ieeexplore.ieee.org/document/10799401>
11. Digi International. (2025). Intelligent transportation systems (ITS). Digi Resources. <https://www.digi.com/resources/definitions/its>
12. European Business Review. (2025). 4 types of businesses that can profit from using an ALPR camera. <https://www.europeanbusinessreview.com/4-types-of-businesses-that-can-profit-from-using-an-alpr-camera/>
13. International Association of Chiefs of Police. (2024, January 26). Automated license plate recognition. IACP Projects. <https://www.theiacp.org/projects/automated-license-plate-recognition>

14. LearnOpenCV. (2022, March 15). Automatic license plate recognition using deep learning. <https://learnopencv.com/automatic-license-plate-recognition-using-deep-learning/>
15. Viso.ai. (2024, December 6). YOLO explained: The state-of-the-art object detection algorithm. <https://viso.ai/computer-vision/yolo-explained/>
16. Phạm Đình Khánh. (2020, 9 tháng 3). Giải thuật Darknet trong YOLO. <https://phamdinhkhanh.github.io/2020/03/09/DarknetAlgorithm.html>