

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN



Nhận diện phương tiện giao thông

Nhóm sinh viên thực hiện:

Nguyễn Mai Hoàng Anh – 23001824

Nguyễn Thị Lan Anh – 23001826

Bùi Xuân Chung – 23001838

Dào Ngọc Cường – 23001841

Nguyễn Thị Huyền Linh – 23001899

Mã học phần: MAT3508
Học kỳ 1, Năm học 2025-2026

Thông tin Dự án

Học phần: MAT3508 – Nhập môn Trí tuệ Nhân tạo

Học kỳ: Học kỳ 1, Năm học 2025-2026

Trường: VNU-HUS (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)

Tên dự án: Nhận diện phương tiện giao thông

Ngày nộp: 30/11/2025

Thành viên nhóm

| Họ tên | Mã sinh viên | Tên GitHub | Đóng góp |
|-----------------------|--------------|-------------------|--|
| Nguyễn Mai Hoàng Anh | 23001824 | 23001824-HoangAnh | Tìm hiểu và triển khai YOLOv8 |
| Nguyễn Thị Lan Anh | 23001826 | lanAnhne29 | Tìm hiểu và triển khai Faster R-CNN |
| Bùi Xuân Chung | 23001838 | 23001838-hub | Tìm hiểu và triển khai Data Augmentation |
| Dào Ngọc Cường | 23001841 | daongocuong | Thiết kế slide trình bày kiến trúc mô hình |
| Nguyễn Thị Huyền Linh | 23001899 | 23001899-af | Làm báo cáo và tiền xử lý dữ liệu |

Lời nói đầu

Trong bối cảnh cuộc cách mạng công nghiệp 4.0 đang diễn ra mạnh mẽ, các công nghệ trí tuệ nhân tạo (AI) và thị giác máy tính ngày càng đóng vai trò quan trọng trong nhiều lĩnh vực của đời sống. Tại Việt Nam, xu hướng triển khai các hệ thống camera thông minh tại khu đô thị, nhà máy, trường học và giao thông công cộng đã và đang tạo ra nhu cầu cấp thiết về các mô hình phân tích hình ảnh tự động, có độ chính xác và tốc độ cao.

Xuất phát từ thực tiễn đó, nhóm chúng em thực hiện đề tài **Nhận diện phương tiện giao thông** với mục tiêu nghiên cứu, xây dựng và đánh giá mô hình phát hiện đối tượng dựa trên YOLOv8, hướng tới ứng dụng giám sát giao thông thời gian thực. Chúng em hy vọng kết quả của đề tài không chỉ cung cấp kiến thức học thuật mà còn có thể ứng dụng trong các hệ thống giám sát thông minh tại Việt Nam.

Mặc dù đã cố gắng hết sức, báo cáo này khó tránh khỏi thiếu sót. Nhóm rất mong nhận được ý kiến đóng góp từ thầy cô và bạn bè để hoàn thiện hơn.

Chúng em xin chân thành cảm ơn các thầy bộ môn đã tận tình hướng dẫn và hỗ trợ nhóm trong suốt quá trình thực hiện đề tài. Xin cảm ơn các thầy cô trong Khoa, bạn bè và gia đình đã giúp đỡ, động viên nhóm hoàn thành báo cáo này.

Chúng em xin chân thành cảm ơn!

Danh sách hình vẽ

| | | |
|------|---|----|
| 2.1 | Classification và Object Detection. | 9 |
| 2.2 | Sơ đồ pipeline phát hiện đối tượng với backbone, neck và detection head. | 10 |
| 2.3 | One-stage and two-stage Object detectors. | 10 |
| 2.4 | Occlusion. | 11 |
| 2.5 | Hình ảnh minh họa từ camera giám sát AI. | 11 |
| 2.6 | Hiện tượng degradation: mạng sâu hơn (56-layer) cho lỗi lớn hơn mạng nông (20-layer). | 12 |
| 2.7 | Hình minh họa. | 12 |
| 2.8 | Khối dư (Residual Block) trong ResNet với kết nối tắt dạng mũi tên cong. | 13 |
| 2.9 | Khối dư (Residual Block) trong ResNet với kết nối tắt dạng mũi tên cong. | 14 |
| 2.10 | Cấu trúc c2f | 15 |
| 2.11 | Dộ chính xác của C2f trong yolov8 | 15 |
| 2.12 | So sánh giữa C2F và C3 | 16 |
| 2.13 | Kiến trúc tổng thể của Faster R-CNN. | 16 |
| 2.14 | Vùng cảm nhận và Các điểm trung tâm Anchor Box | 18 |
| 2.15 | Minh họa IoU giữa bounding box dự đoán và ground truth | 18 |
| 2.16 | Region Proposal Network(RPN) | 19 |
| 2.17 | Region of Interest Pooling | 20 |
| 2.18 | Kiến trúc chi tiết của Faster R-CNN. | 20 |
| 2.19 | YOLO qua các năm. | 22 |
| 2.20 | Các biến thể của YOLOv8. | 24 |
| 2.21 | Performance của mô hình YOLOv8 Detection được train trên tập COCO. | 24 |
| 2.22 | Kiến trúc của YOLOv8. | 25 |
| 2.23 | Kiến trúc của YOLOv8. | 26 |
| 2.24 | Mosaic & MixUp. | 29 |
| 2.25 | MixUp. | 31 |
| 2.26 | Mosaic & MixUp. | 32 |
| 2.27 | Flip. | 32 |
| 2.28 | Hue Adjustment. | 33 |
| 2.29 | Bright adjustment. | 33 |
| 2.30 | Blur. | 33 |
| 2.31 | Tổng quan dữ liệu | 36 |
| 3.1 | Biểu đồ hàm mất mát của mô hình trong quá trình huấn luyện. | 40 |
| 3.2 | Ảnh dữ liệu đầu vào kèm bounding boxes của ground truth. | 41 |
| 3.3 | Kết quả dự đoán trước khi áp dụng NMS. Total number of rectangle: 19 | 42 |
| 3.4 | Kết quả dự đoán sau khi áp dụng NMS. Total number of rectangle: 17 | 42 |
| 3.5 | Biểu đồ hàm mất mát | 45 |

Danh sách bảng

| | | |
|-----|---|----|
| 2.1 | Giải thích các thuật ngữ liên quan đến YOLO | 23 |
| 2.2 | Thuật ngữ trong YOLOv8 | 24 |
| 2.3 | Thuật ngữ trong các biến thể YOLOv8 | 25 |
| 2.4 | So sánh cơ bản giữa Faster R-CNN và YOLOv8 về độ chính xác, tốc độ và độ trễ. | 35 |
| 3.1 | So sánh tổng quan giữa Faster R-CNN và YOLOv8 trên bài toán phát hiện phương tiện giao thông. | 47 |

Mục lục

| | |
|---|----------|
| Lời nói đầu | 2 |
| Lời nói đầu | 2 |
| 1 Giới thiệu | 7 |
| 1.1 Tóm tắt | 7 |
| 1.2 Bài toán đặt ra | 7 |
| 1.3 Động lực | 8 |
| 2 Cơ sở lý thuyết: Phương pháp và triển khai | 9 |
| 2.1 Object Detection | 9 |
| 2.1.1 Các loại mô hình object detection: | 10 |
| 2.1.2 Khó khăn và thách thức trong object detection thực tế: | 10 |
| 2.1.3 Ứng dụng trong giám sát giao thông: | 11 |
| 2.1.4 Tổng kết | 11 |
| 2.2 Backbone: ResNet & C2f | 12 |
| 2.2.1 Giới thiệu | 12 |
| 2.2.2 Kiến trúc mạng ResNet | 13 |
| 2.2.3 So sánh với các mạng trước đó | 13 |
| 2.2.4 Ứng dụng của backbone ResNet trong giám sát giao thông thông minh | 14 |
| 2.2.5 Khối C2f trong Backbone của YOLOv8 | 14 |
| 2.3 Faster R-CNN | 16 |
| 2.3.1 Giới thiệu | 16 |
| 2.3.2 Kiến trúc tổng quan | 16 |
| 2.3.3 Ưu điểm | 21 |
| 2.3.4 Nhược điểm | 21 |
| 2.3.5 Ứng dụng | 21 |
| 2.3.6 Kết luận | 21 |
| 2.4 YOLOv8 | 22 |
| 2.4.1 Lịch sử phát triển của YOLO | 22 |
| 2.4.2 Tổng quan về YOLOv8 | 23 |
| 2.4.3 Các phiên bản của YOLOv8 | 23 |
| 2.4.4 Kiến trúc YOLOv8 | 25 |
| 2.4.5 Hàm mất mát (Loss) và tối ưu: | 28 |
| 2.4.6 Kỹ thuật huấn luyện và Augmentation | 29 |
| 2.5 So sánh Faster R-CNN và YOLOv8 | 35 |
| 2.6 Dữ liệu | 35 |
| 2.6.1 Nguồn dữ liệu | 35 |
| 2.6.2 Lớp đối tượng quan tâm | 36 |
| 2.6.3 Công cụ annotate và xử lý dữ liệu | 36 |
| 2.6.4 Phân chia dữ liệu | 36 |
| 2.6.5 Đặc điểm và thách thức của dữ liệu thực tế | 37 |
| 2.6.6 Kết luận | 37 |
| 2.7 Triển khai mô hình YOLOv8 | 37 |
| 2.7.1 Framework và phiên bản mô hình | 37 |
| 2.7.2 Dữ liệu và Augmentation | 37 |
| 2.7.3 Siêu tham số (Hyperparameters) | 37 |
| 2.7.4 Quy trình huấn luyện (Pipeline) | 38 |

| | | |
|---------------------------|--|-----------|
| 2.7.5 | Triển khai và xuất mô hình | 38 |
| 2.7.6 | Kết luận | 38 |
| 3 | Thực nghiệm & Phân tích | 39 |
| 3.1 | Kết quả thực nghiệm với mô hình Faster R-CNN | 39 |
| 3.1.1 | Công cụ và thư viện sử dụng | 39 |
| 3.1.2 | Chuẩn bị và nạp dữ liệu | 39 |
| 3.1.3 | Cấu hình huấn luyện | 40 |
| 3.1.4 | Biểu đồ hàm mất mát | 40 |
| 3.1.5 | Trực quan hóa dữ liệu ground truth | 41 |
| 3.1.6 | Kết quả dự đoán của mô hình (Before NMS) | 41 |
| 3.1.7 | Áp dụng Non-Maximum Suppression | 41 |
| 3.1.8 | Tóm tắt những gì đã thực hiện | 43 |
| 3.1.9 | Dánh giá chung | 43 |
| 3.2 | Kết quả thực nghiệm với mô hình YOLOv8 | 43 |
| 3.2.1 | Công cụ và thư viện sử dụng | 43 |
| 3.2.2 | Chuẩn bị và nạp dữ liệu | 44 |
| 3.2.3 | Cấu hình huấn luyện | 44 |
| 3.2.4 | Biểu đồ hàm mất mát | 45 |
| 3.2.5 | Trực quan hóa dữ liệu ground truth | 45 |
| 3.2.6 | Kết quả dự đoán (Before NMS) | 46 |
| 3.2.7 | Áp dụng Non-Maximum Suppression | 46 |
| 3.2.8 | Tóm tắt pipeline thực nghiệm | 46 |
| 3.2.9 | Dánh giá chung | 46 |
| 3.3 | So sánh và đánh giá | 47 |
| 3.3.1 | Bảng so sánh tổng quan | 47 |
| 3.3.2 | Phân tích chi tiết các tiêu chí | 47 |
| 3.3.3 | Nhận xét tổng quan | 47 |
| 4 | Kết luận và hướng phát triển | 48 |
| 4.1 | Kết luận | 48 |
| 4.2 | Hướng phát triển | 48 |
| Tài liệu tham khảo | | 48 |

Chương 1

Giới thiệu

1.1 Tóm tắt

Nhận diện phương tiện giao thông là một trong những bài toán quan trọng trong lĩnh vực thị giác máy tính và trí tuệ nhân tạo, đặc biệt khi áp dụng vào các hệ thống giám sát giao thông thông minh. Trong bối cảnh đô thị ngày càng phát triển, lưu lượng phương tiện gia tăng nhanh chóng và cơ sở hạ tầng giao thông trở nên phức tạp, việc tự động hóa quá trình giám sát và phân tích lưu lượng giao thông trở nên thiết yếu. Nhận diện chính xác các loại phương tiện như ô tô, xe máy, xe tải, cũng như người đi bộ, đóng vai trò quan trọng trong việc thu thập dữ liệu lưu lượng, phát hiện vi phạm, và hỗ trợ công tác điều tiết giao thông.

Mục tiêu chính của dự án là xây dựng một mô hình có khả năng phát hiện các đối tượng giao thông trong môi trường thực tế một cách nhanh chóng và chính xác, đáp ứng các điều kiện khắt khe như: nhiều loại phương tiện xuất hiện đồng thời, hiện tượng che khuất (occlusion), các điều kiện ánh sáng thay đổi theo thời gian trong ngày, và sự đa dạng về kích thước cũng như hình dáng của các đối tượng. Đặc biệt, mô hình phải đảm bảo hiệu suất đủ cao để có thể triển khai gần thời gian thực (real-time), từ đó hỗ trợ các ứng dụng giám sát giao thông trực tiếp, như đếm số lượng phương tiện, cảnh báo vi phạm, và ghi nhận dữ liệu phục vụ nghiên cứu giao thông.

Trong quá trình triển khai, nhóm nghiên cứu đã lựa chọn YOLOv8 - một mô hình one-stage detector hiện đại - để giải quyết bài toán này. YOLOv8 nổi bật với kiến trúc backbone → neck → head anchor-free, giúp đạt tốc độ xử lý cao đồng thời duy trì độ chính xác vượt trội. Thông qua các thử nghiệm, mô hình đã cho kết quả mAP (mean Average Precision) cao, đồng thời tốc độ xử lý FPS (frame per second) đủ đáp ứng yêu cầu gần thời gian thực trên các thiết bị GPU phổ biến. So với các mô hình hai giai đoạn (two-stage detector) như Faster R-CNN, YOLOv8 không những giữ được độ chính xác mà còn khắc phục nhược điểm về tốc độ, đặc biệt khi xử lý dữ liệu phức tạp với nhiều đối tượng nhỏ hoặc bị che khuất.

Kết quả này mở ra nhiều hướng ứng dụng thực tế: hệ thống giám sát có thể tự động phát hiện và phân loại các loại phương tiện, cảnh báo vi phạm giao thông như vượt đèn đỏ hoặc đi sai làn, và hỗ trợ thu thập dữ liệu lưu lượng theo thời gian thực. Bên cạnh đó, mô hình cũng có thể được tích hợp với các thuật toán tracking như DeepSORT hay ByteTrack để theo dõi chuyển động của từng phương tiện, từ đó nâng cao khả năng phân tích lưu lượng và hành vi giao thông. Nhờ những ưu điểm này, dự án không chỉ đóng góp về mặt học thuật trong lĩnh vực nhận diện đối tượng mà còn có giá trị ứng dụng thực tiễn cao trong việc xây dựng các thành phố thông minh và hệ thống giao thông hiệu quả.

1.2 Bài toán đặt ra

Bài toán nhận diện phương tiện giao thông không chỉ là việc xác định nhãn đối tượng mà còn yêu cầu xác định chính xác vị trí trong ảnh dưới nhiều điều kiện phức tạp. Trong thực tế, các hệ thống giám sát giao thông phải xử lý dữ liệu từ camera cố định, camera dashcam, hoặc các nguồn video khác, nơi xuất hiện đồng thời nhiều loại phương tiện như ô tô, xe máy, xe tải, bus, và người đi bộ. Mỗi loại phương tiện có đặc điểm hình dạng, kích thước, tốc độ và hướng di chuyển khác nhau, tạo ra một môi trường đa dạng và khó dự đoán.

Một trong những thách thức lớn nhất là yêu cầu thời gian thực (real-time). Hệ thống cần phản hồi gần như tức thì để hỗ trợ giám sát, phát hiện vi phạm hoặc đếm số lượng phương tiện. Các mô hình hai giai đoạn truyền thống như Faster R-CNN tuy có độ chính xác cao nhưng lại gặp hạn chế nghiêm trọng về tốc độ xử lý, đặc biệt khi phải phân tích các video có độ phân giải cao hoặc nhiều đối tượng cùng lúc. Kết quả là, khi triển khai thực tế, Faster R-CNN không đáp ứng được yêu cầu gần thời gian thực, khiến việc giám sát trực tiếp trở nên khó khăn.

Bên cạnh tốc độ, dữ liệu thực tế còn chứa nhiều khó khăn khác: - Occlusion (che khuất): Xe hoặc người đi bộ có thể bị che khuất bởi các phương tiện khác, làm giảm khả năng nhận diện chính xác. - Small objects (vật thể nhỏ): Các đối tượng nhỏ, như xe máy ở xa camera, thường khó được phát hiện đầy đủ. - Điều kiện ánh

sáng thay đổi: Video có thể được ghi vào ban ngày, buổi tối, hoặc khi trời mưa, ánh sáng yếu, tạo ra sự thay đổi lớn về màu sắc, độ tương phản và bóng tối. - Biến dạng hình học và góc nhìn khác nhau: Camera cố định hoặc dashcam quay từ nhiều góc độ khác nhau, khiến hình dạng các đối tượng thay đổi, làm khó khăn thêm cho quá trình nhận diện.

Do đó, yêu cầu đặt ra là xây dựng một mô hình vừa nhanh, vừa chính xác, có thể xử lý dữ liệu đa dạng và phức tạp, đồng thời có khả năng mở rộng cho các ứng dụng giám sát giao thông thực tế. Nhóm nghiên cứu đã lựa chọn YOLOv8 vì đây là mô hình one-stage detector hiện đại, nổi bật với tốc độ xử lý cao, kiến trúc anchor-free, và khả năng đạt độ chính xác tốt trên nhiều loại đối tượng khác nhau. YOLOv8 cho phép tối ưu hóa giữa độ chính xác và tốc độ, giải quyết hạn chế của Faster R-CNN và đáp ứng các yêu cầu của bài toán real-time, đồng thời dễ dàng tích hợp các kỹ thuật augment dữ liệu và tối ưu hóa cho các phần cứng GPU phổ biến.

1.3 Động lực

Bài toán nhận diện phương tiện giao thông không chỉ là một bài toán nghiên cứu thuần túy mà còn mang tính ứng dụng thực tiễn rất cao trong việc xây dựng các hệ thống giao thông thông minh (Intelligent Traffic System - ITS). Trong bối cảnh các thành phố lớn đang phải đổi mới với mật độ phương tiện ngày càng tăng, ùn tắc giao thông và các vấn đề về an toàn, việc phát triển các công cụ tự động hóa giám sát trở nên vô cùng cần thiết. Nhận diện phương tiện giao thông chính xác cho phép các cơ quan quản lý theo dõi, phân tích và điều phối lưu lượng xe cộ, đồng thời hỗ trợ cảnh báo kịp thời khi xảy ra các tình huống nguy hiểm.

Một trong những ứng dụng quan trọng là giám sát vi phạm giao thông. Thông qua hệ thống camera, mô hình nhận diện có thể phát hiện các hành vi như vượt đèn đỏ, đi sai làn, dừng đỗ sai quy định, hoặc chạy quá tốc độ. Việc tự động hóa này giúp giảm thiểu sự phụ thuộc vào con người, tăng độ chính xác trong giám sát, đồng thời hỗ trợ cơ quan chức năng lập biên bản nhanh chóng. Khi kết hợp với các thuật toán tracking, như DeepSORT hay ByteTrack, hệ thống còn có thể theo dõi lộ trình phương tiện, xác định số lần vi phạm, và tích hợp dữ liệu để phân tích xu hướng giao thông theo thời gian thực.

Một ứng dụng khác là đếm số lượng phương tiện và phân loại theo loại xe, cung cấp các thông tin thống kê quan trọng cho việc quy hoạch giao thông và hạ tầng đô thị. Ví dụ, biết được số lượng ô tô, xe máy, bus lưu thông trên từng tuyến đường vào các khung giờ khác nhau giúp điều phối đèn tín hiệu, tối ưu hóa luồng xe và giảm thiểu ùn tắc. Điều này đặc biệt quan trọng trong các thành phố lớn, nơi mà việc giám sát bằng con người là khó khả thi hoặc không hiệu quả.

Về mặt kỹ thuật, xu hướng hiện nay ưu tiên sử dụng các mô hình one-stage detector như YOLOv8, thay vì các mô hình hai giai đoạn như Faster R-CNN, nhằm đạt hiệu năng real-time. Các mô hình one-stage có ưu điểm là tích hợp toàn bộ quá trình phát hiện trong một bước, giảm thiểu độ trễ xử lý, đồng thời vẫn giữ được độ chính xác cao nhờ các cải tiến về backbone, neck (FPN/PAN) và head anchor-free. Kết hợp với các kỹ thuật augmentation dữ liệu, tối ưu siêu tham số, và triển khai trên phần cứng GPU hiện đại, hệ thống có thể đạt tốc độ phản hồi nhanh, phù hợp cho các ứng dụng giám sát đô thị thực tế.

Tổng hợp lại, động lực chính của bài toán này bao gồm:

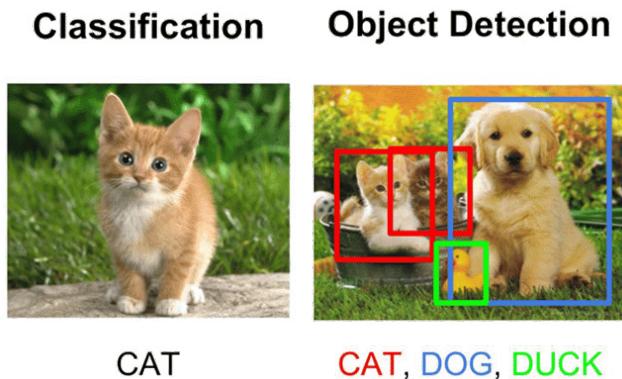
- Hỗ trợ quản lý giao thông thông minh, nâng cao an toàn và hiệu quả điều tiết phương tiện.
- Phát hiện vi phạm giao thông tự động, giảm phụ thuộc con người và nâng cao độ chính xác.
- Thu thập dữ liệu thống kê về lưu lượng và loại phương tiện để phục vụ quy hoạch hạ tầng.
- Thực hiện các ứng dụng real-time với mô hình one-stage hiện đại, đáp ứng cả tốc độ và độ chính xác trong môi trường thực tế.

Chương 2

Cơ sở lý thuyết: Phương pháp và triển khai

2.1 Object Detection

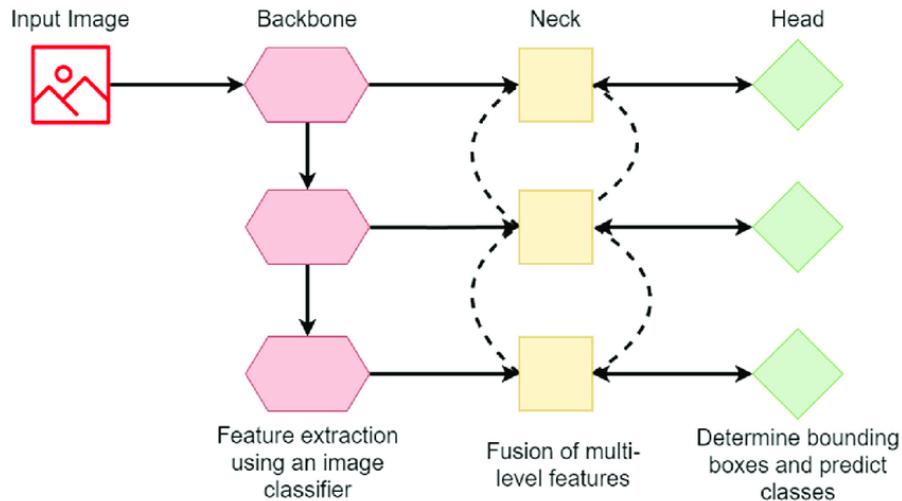
Phát hiện đối tượng (Object Detection) là một trong những bài toán quan trọng và thách thức nhất trong lĩnh vực Thị giác máy tính (Computer Vision), đóng vai trò nền tảng cho nhiều ứng dụng thực tế như giám sát giao thông, an ninh, nhận diện khuôn mặt, robot tự hành, và phân tích hành vi con người. Khác với bài toán phân loại ảnh (Image Classification) vốn chỉ xác định nhãn của toàn bộ ảnh, object detection yêu cầu mô hình không chỉ xác định nhãn của các đối tượng mà còn xác định chính xác vị trí của chúng trong không gian ảnh thông qua các bounding box. Điều này làm cho bài toán trở nên phức tạp hơn rất nhiều, bởi dữ liệu thực tế thường chứa nhiều đối tượng chồng lấn (occlusion), background phức tạp, và điều kiện ánh sáng, thời tiết thay đổi liên tục.



Hình 2.1: Classification và Object Detection.

Object detection hiện đại dựa trên các mạng nơ-ron sâu (Deep Neural Networks) và thường được chia thành ba thành phần chính: Backbone, Neck, và Detection Head.

- Backbone (Feature Extraction):** Backbone là phần quan trọng nhất trong mô hình, chịu trách nhiệm trích xuất các đặc trưng từ ảnh đầu vào. Những đặc trưng này bao gồm cạnh, đường viền, màu sắc, hình dạng, và các mẫu phức tạp hơn khi qua các tầng sâu của mạng. Các kiến trúc backbone phổ biến hiện nay gồm ResNet, Darknet, EfficientNet, MobileNet, DenseNet, mỗi loại có ưu nhược điểm khác nhau về độ sâu, số lượng tham số, và tốc độ trích xuất.
- Neck (Feature Fusion - Tùy chọn):** Neck là phần trung gian, thường được sử dụng trong các mô hình phát hiện đối tượng hiện đại để kết hợp đặc trưng từ nhiều tầng (multi-scale feature fusion). Điều này giúp mô hình có khả năng nhận diện các đối tượng có kích thước khác nhau. Một số kỹ thuật phổ biến là FPN (Feature Pyramid Network) và PAN (Path Aggregation Network).
- Detection Head:** Detection head chịu trách nhiệm dự đoán bounding box, nhãn đối tượng, và confidence score. Detection head thường bao gồm các lớp convolution để dự đoán tọa độ box (x, y, w, h), nhãn lớp (class label), và độ tin cậy (confidence). Bước Non-Maximum Suppression (NMS) được áp dụng để loại bỏ các dự đoán trùng lặp.

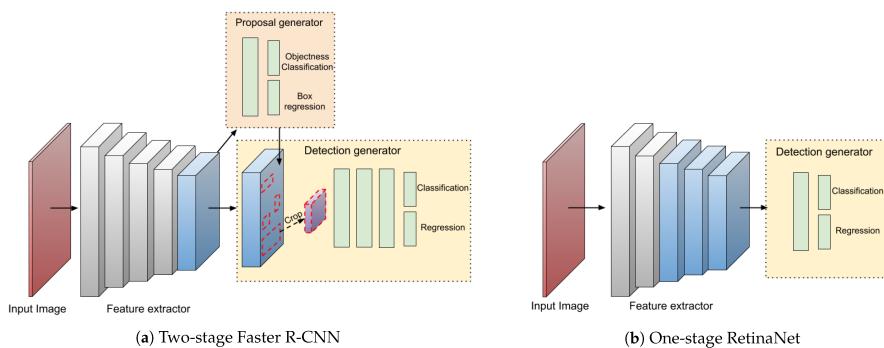


Hình 2.2: Sơ đồ pipeline phát hiện đối tượng với backbone, neck và detection head.

2.1.1 Các loại mô hình object detection:

Hiện nay, mô hình object detection được chia thành hai nhóm chính:

- **Two-stage detectors:** Đây là nhóm mô hình điển hình là Faster R-CNN, R-FCN, Mask R-CNN. Các mô hình này sử dụng một bước Region Proposal Network (RPN) để đề xuất các vùng khả thi (region proposals) có thể chứa đối tượng. Sau đó, từng vùng được phân loại và tinh chỉnh bounding box. Ưu điểm của two-stage detectors là độ chính xác cao, đặc biệt với các đối tượng nhỏ hoặc phức tạp. Tuy nhiên, nhược điểm là tốc độ xử lý chậm, chi phí tính toán lớn, và khó triển khai real-time, đặc biệt khi xử lý video từ camera giám sát.
- **One-stage detectors:** Nhóm mô hình này bao gồm YOLO (các phiên bản từ v1 đến v8), SSD, RetinaNet. One-stage detectors bỏ qua bước đề xuất vùng, trực tiếp dự đoán bounding box và nhãn từ feature maps. Ưu điểm là tốc độ nhanh, dễ triển khai real-time, phù hợp cho các ứng dụng yêu cầu xử lý video liên tục. Nhược điểm là đôi khi độ chính xác thấp hơn, đặc biệt với small objects hoặc các tình huống phức tạp như occlusion.



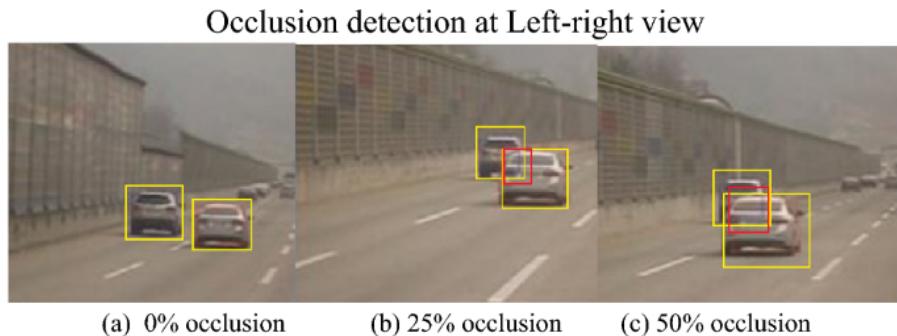
Hình 2.3: One-stage and two-stage Object detectors.

2.1.2 Khó khăn và thách thức trong object detection thực tế:

Trong môi trường thực tế, object detection gặp nhiều thách thức:

- **Đối tượng nhỏ (small objects):** Trong giám sát giao thông, biển số xe, xe máy nhỏ thường khó phát hiện vì chiếm ít pixel trên ảnh.
- **Occlusion (che khuất):** Các phương tiện thường chồng lấn nhau, làm cho việc định vị bounding box chính xác trở nên khó khăn.
- **Đa dạng môi trường:** Thời tiết, ánh sáng, góc nhìn camera thay đổi liên tục, ảnh hưởng đến độ chính xác của mô hình.

- **Background phức tạp:** Hình ảnh thực tế có nhiều chi tiết không liên quan (cây cối, biển quảng cáo, người đi bộ), mô hình cần phân biệt chính xác đối tượng và background.

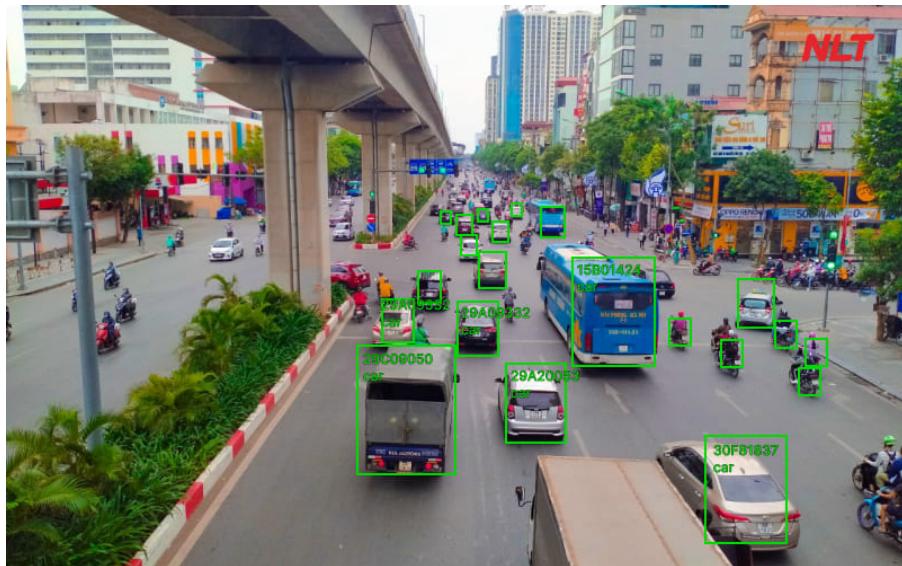


Hình 2.4: Occlusion.

2.1.3 Ứng dụng trong giám sát giao thông:

Trong hệ thống giao thông thông minh, object detection là bước đầu tiên quan trọng để xây dựng các chức năng như:

- Dễm phương tiện: Xe máy, ô tô, xe tải, xe buýt.
- Phát hiện vi phạm: Vượt đèn đỏ, đi sai làn, đi ngược chiều.
- Phân tích lưu lượng: Xác định dòng xe, thời gian cao điểm, tắc nghẽn.
- Hỗ trợ camera thông minh: Theo dõi và cảnh báo các tình huống nguy hiểm.



Hình 2.5: Hình ảnh minh họa từ camera giám sát AI.

Việc lựa chọn backbone mạnh, neck hiệu quả và detection head tối ưu giúp mô hình đạt độ chính xác cao và tốc độ xử lý nhanh, từ đó đáp ứng yêu cầu real-time trong các ứng dụng giám sát giao thông hiện đại. Đồng thời, các kỹ thuật augmentation, regularization và fine-tuning giúp mô hình ổn định hơn trên dữ liệu thực tế, cải thiện khả năng generalization khi triển khai ngoài môi trường huấn luyện.

2.1.4 Tổng kết

Object detection không chỉ là bài toán lý thuyết mà còn là nền tảng cho các hệ thống trí tuệ nhân tạo trong thực tế, đặc biệt là trong giám sát giao thông, robot tự hành, và các ứng dụng video analytics. Việc hiểu rõ backbone, neck, head, hai nhóm mô hình one-stage và two-stage, cũng như những thách thức thực tế, là điều kiện tiên quyết để lựa chọn mô hình phù hợp, tối ưu hóa hiệu năng và tốc độ xử lý.

2.2 Backbone: ResNet & C2f

2.2.1 Giới thiệu

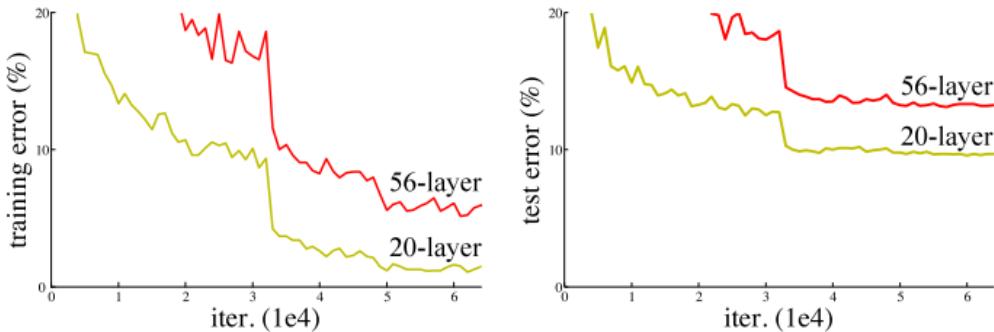
ResNet (Residual Network) được giới thiệu đến công chúng vào năm 2015 và thậm chí đã giành được vị trí thứ nhất trong cuộc thi ILSVRC 2015 với tỉ lệ lỗi top-5 chỉ 3.57%. Không những thế, nó còn đứng vị trí đầu tiên trong các hạng mục ImageNet Detection, ImageNet Localization, COCO Detection và COCO Segmentation của cuộc thi ILSVRC và COCO 2015.

Hiện tại có rất nhiều biến thể của kiến trúc ResNet với số lượng lớp khác nhau như ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152, ... Với tên gọi “ResNet” theo sau là một con số chỉ kiến trúc ResNet với số lớp tương ứng.

Vậy, tại sao lại xuất hiện mạng ResNet?

Mạng ResNet (Residual Network) là một mạng nơ-ron tích chập (CNN) được thiết kế để hoạt động hiệu quả với hàng trăm thậm chí hàng nghìn lớp. Một vấn đề lớn xảy ra khi xây dựng các mạng CNN rất sâu là hiện tượng **Vanishing Gradient**, dẫn tới quá trình huấn luyện không hiệu quả.

Vanishing Gradient Trước hết, thuật toán **Backpropagation** là kỹ thuật chính được sử dụng trong quá trình huấn luyện. Ý tưởng của thuật toán này là truyền ngược từ tầng đầu ra (output layer) về tầng đầu vào (input layer), tính toán đạo hàm (gradient) của hàm mất mát (cost function) đối với từng tham số (weight) của mạng. Sau đó, **Gradient Descent** được sử dụng để cập nhật các tham số này.



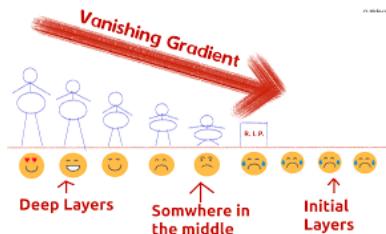
Hình 2.6: Hiện tượng degradation: mạng sâu hơn (56-layer) cho lỗi lớn hơn mạng nông (20-layer).

Toàn bộ quá trình trên được lặp lại nhiều lần cho đến khi các tham số của mạng hội tụ. Thông thường, ta có một siêu tham số là **số epoch** – số lần toàn bộ tập huấn luyện được duyệt qua một lần và các trọng số được cập nhật – để xác định số vòng lặp. Nếu số epoch quá nhỏ, mạng có thể chưa học tốt (*underfitting*); ngược lại nếu quá lớn thì thời gian huấn luyện sẽ rất lâu.

Tuy nhiên, trong thực tế, giá trị gradient thường có xu hướng **nhỏ dần** khi lan truyền ngược qua các tầng thấp hơn. Kết quả là các cập nhật do Gradient Descent thực hiện hầu như không làm thay đổi trọng số của các tầng này, khiến chúng không thể hội tụ tốt. Hiện tượng gradient biến mất dần như vậy được gọi là **Vanishing Gradient Problem**.

Toàn bộ quá trình trên sẽ được lặp đi lặp lại cho đến khi các tham số của mạng đạt được trạng thái hội tụ. Thông thường, ta sử dụng một siêu tham số (hyperparameter) là số *epoch* — số lần toàn bộ tập huấn luyện (*training set*) được duyệt qua một lần và các trọng số (*weights*) được cập nhật — để định nghĩa số lượng vòng lặp của quá trình này.

Nếu số lượng epoch quá nhỏ, mạng có thể chưa học đủ và cho kết quả kém (*underfitting*). Ngược lại, nếu số epoch quá lớn thì thời gian huấn luyện sẽ tăng đáng kể mà hiệu năng cải thiện không đáng kể (*overfitting* hoặc lãng phí tài nguyên).



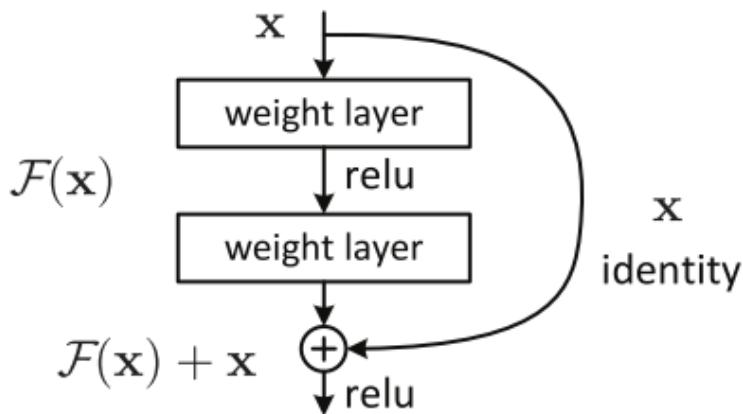
Hình 2.7: Hình minh họa.

Tuy nhiên, trong thực tế, giá trị của *gradient* thường có xu hướng nhỏ dần rất nhanh khi lan truyền ngược qua các tầng thấp hơn của mạng (các tầng gần đầu vào). Hệ quả là các cập nhật trọng số do thuật toán *Gradient Descent* thực hiện ở những tầng này trở nên cực kỳ nhỏ, gần như không làm thay đổi trọng số. Điều này khiến các tầng thấp không thể hội tụ tốt, dẫn đến hiệu suất tổng thể của mạng bị giảm mạnh. Hiện tượng này được gọi là **vấn đề vanishing gradient** (gradient biến mất).

2.2.2 Kiến trúc mạng ResNet

Giai pháp mà ResNet đưa ra là sử dụng các **kết nối tắt đồng nhất** (identity shortcut connection) để xuyên qua một hoặc nhiều lớp. Một khối như vậy được gọi là **Residual Block**.

Về cơ bản, ResNet có cấu trúc gần tương tự các mạng nơ-ron tích chập thông thường, bao gồm các tầng convolution, pooling, activation và fully-connected layer. Điểm khác biệt lớn nhất nằm ở khối dư (residual block) như hình minh họa bên dưới:



Hình 2.8: Khối dư (Residual Block) trong ResNet với kết nối tắt dạng mũi tên cong.

Mũi tên cong thể hiện việc bổ sung trực tiếp đầu vào x vào đầu ra của các tầng trong khối:

$$y = F(x, \{W_i\}) + x$$

trong đó:

- x : đầu vào của khối dư,
- $F(x, \{W_i\})$: hàm ánh xạ dư (residual mapping) được học,
- y : đầu ra của khối dư.

Việc cộng trực tiếp x (identity mapping) giúp chống lại hiện tượng **đạo hàm bằng 0** (vanishing gradient) vì ngay cả khi $F(x) \approx 0$, đạo hàm của khối vẫn có giá trị ít nhất bằng 1 thông qua nhánh tắt.

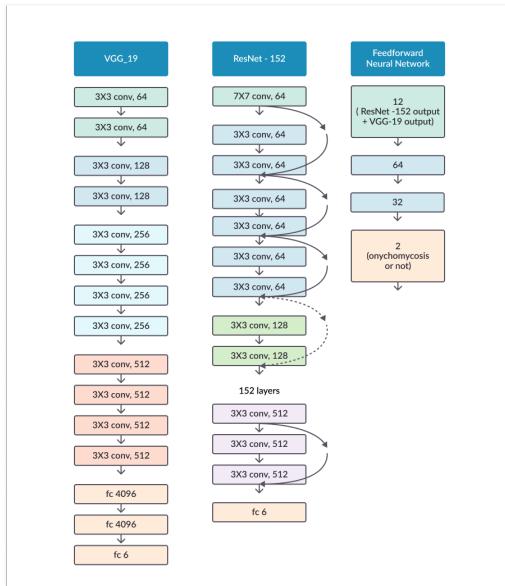
Gọi $H(x)$ là giá trị dự đoán mong muốn của mạng và $F(x)$ là phần dư cần học, ta có:

$$H(x) = F(x) + x$$

Mục tiêu huấn luyện chuyển từ học trực tiếp $H(x)$ sang học phần dư $F(x) = H(x) - x$, điều này thường dễ tối ưu hơn.

2.2.3 So sánh với các mạng trước đó

- **Hình 1:** VGG-19 là một mô hình CNN sử dụng hoàn toàn kernel 3×3 trên toàn bộ mạng và đã giành chiến thắng ILSVRC 2014.
- **Hình 2:** ResNet giới thiệu các kết nối tắt (shortcut connection) nối trực tiếp đầu vào của lớp n với lớp $n + x$, được biểu diễn dưới dạng mũi tên cong. Thí nghiệm cho thấy ResNet có thể cải thiện đáng kể hiệu suất huấn luyện khi số lớp vượt quá 20, thậm chí lên đến hàng trăm hoặc hàng nghìn lớp.
- **Hình 3:** So sánh ResNet-152 và VGG-19 khi sử dụng 12 đầu ra đặc trưng làm đầu vào cho một mạng fully-connected gồm 2 lớp ẩn. Kết quả chứng minh rằng việc xếp chồng thêm các lớp trong ResNet **không làm giảm hiệu suất**, thậm chí còn cải thiện độ chính xác trên cả tập huấn luyện và kiểm tra.



Hình 2.9: Khối dư (Residual Block) trong ResNet với kết nối tắt dạng mũi tên cong.

Nhờ kiến trúc residual, các lớp phía trên nhận được thông tin trực tiếp hơn từ hàm mất mát và từ các lớp dưới, giúp điều chỉnh trọng số hiệu quả hơn, từ đó giải quyết triệt để vấn đề suy giảm hiệu suất (degradation problem) khi mạng ngày càng sâu.

2.2.4 Ứng dụng của backbone ResNet trong giám sát giao thông thông minh

Backbone ResNet (đặc biệt là ResNet-50, ResNet-101 và ResNet-152) hiện là một trong những backbone được sử dụng phổ biến nhất trong các hệ thống giám sát giao thông thông minh (Intelligent Transportation Systems – ITS) nhờ khả năng trích xuất đặc trưng mạnh mẽ và ổn định.

Lý do ResNet được ưa chuộng trong giám sát giao thông

- Độ sâu lớn nhưng không bị hiện tượng *vanishing gradient* nhờ residual connection.
- Được huấn luyện trước (pretrained) trên ImageNet → transfer learning cực tốt, chỉ cần fine-tune trên dữ liệu giao thông Việt Nam là đạt độ chính xác cao.
- Tốc độ suy luận phù hợp với triển khai thực tế (ResNet-50 đạt 25–40 FPS trên GPU RTX 3080, vẫn chạy tốt trên Jetson Xavier/Nano khi giảm độ phân giải).
- Có nhiều biến thể tối ưu: ResNeXt, ResNet-RS, ResNet-D, ResNet-St, phù hợp cả về độ chính xác và tốc độ.

Các dự án thực tế tại Việt Nam sử dụng ResNet backbone

- Hệ thống phạt nguội của Cục CSGT và các địa phương (Hà Nội, TP.HCM, Đà Nẵng).
- Hệ thống ANPR của VNPT AI, FPT AI, Viettel Solutions, CMC Cyber Security.
- Giải pháp đếm xe, đo tốc độ, phát hiện vi phạm của VinAI (Vingroup), CMC TS, ITS Việt Nam.

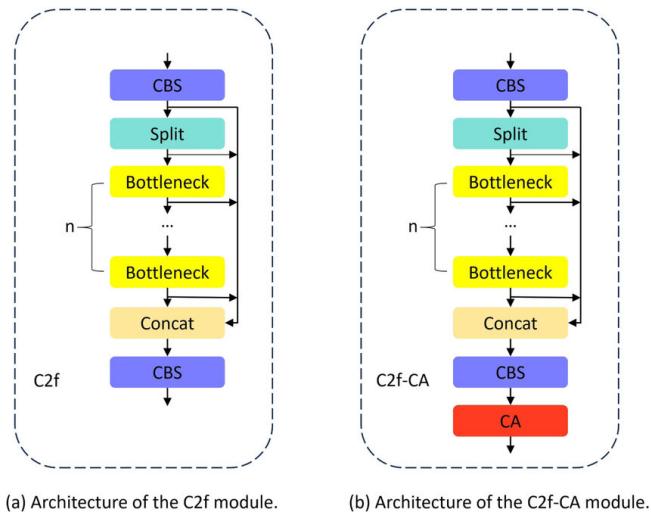
Kết luận: Tính đến năm 2025, hầu hết các hệ thống giám sát giao thông state-of-the-art tại Việt Nam và trên thế giới đều sử dụng ResNet (hoặc các biến thể cải tiến) làm backbone chính nhờ sự cân bằng tối ưu giữa độ chính xác, tốc độ và khả năng triển khai thực tế.

2.2.5 Khối C2f trong Backbone của YOLOv8

a. **C2f (CSP to Feature)** là gì? C2f là khối đặc trưng được Ultralytics giới thiệu trong **YOLOv8** (2023), thay thế hoàn toàn cho khối **C3** của YOLOv5.

C2f được thiết kế theo tư duy **Cross-Stage Partial (CSP) + multi-branch Bottleneck** nhằm:

- Tăng cường luồng gradient (gradient flow)
- Giảm độ phức tạp tính toán
- Cải thiện khả năng phát hiện vật thể nhỏ và bị che khuất



Hình 2.10: Cấu trúc c2f

b. Cấu trúc chi tiết của C2f

c. Công thức toán học Giả sử đầu vào là $X \in R^{C \times H \times W}$, khối C2f thực hiện:

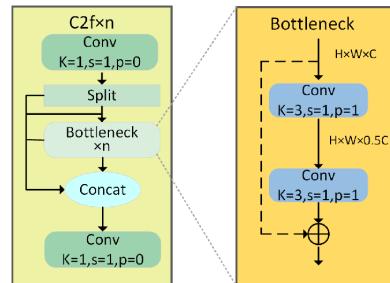
$$X_1 = \text{Conv}_1(X) \quad (2.1)$$

$$X_2 = \text{Conv}_2(X) \quad (2.2)$$

$$Y_i = \text{Bottleneck}_i(Y_{i-1}), \quad i = 1, 2, \dots, N \quad (2.3)$$

$$Y = \text{Concat}(X_2, Y_1, Y_2, \dots, Y_N, X_1) \quad (2.4)$$

$$\text{Output} = \text{Conv}_{\text{final}}(Y) \quad (2.5)$$



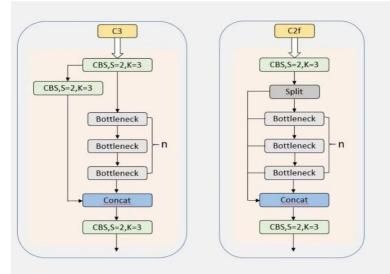
Hình 2.11: Độ chính xác của C2f trong yolov8

d. Ưu điểm nổi bật của C2f so với C3

| Tiêu chí | C3 (YOLOv5) | C2f (YOLOv8) |
|----------------------------|-------------|-------------------------|
| Tốc độ inference | Trung bình | Nhanh hơn 10–20% |
| Số tham số | Cao hơn | Thấp hơn |
| Khả năng phát hiện vật nhỏ | Tốt | Rất tốt |
| Gradient flow | Tốt | Tuyệt vời |

e. Các phiên bản sử dụng C2f

- **YOLOv8** (Ultralytics): Sử dụng chính thức
- **YOLOv9**: Kết hợp C2f + GELAN



Hình 2.12: So sánh giữa C2F và C3

- **YOLOv10:** Một số biến thể vẫn giữ C2f
- **YOLOv11:** Thay bằng C3k2 (cải tiến hơn nữa)

Kết luận: C2f là bước tiến quan trọng giúp YOLOv8 đạt được sự cân bằng hoàn hảo giữa tốc độ và độ chính xác, trở thành chuẩn mực mới trong các backbone object detection thời điểm 2023–2025.

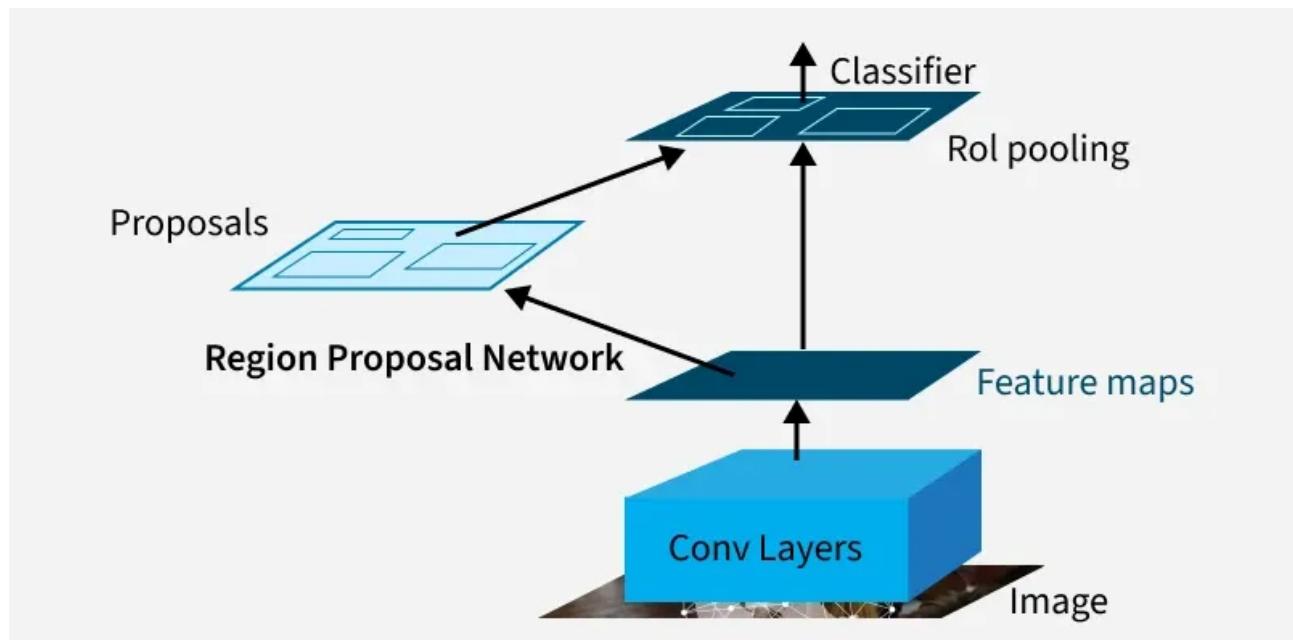
2.3 Faster R-CNN

2.3.1 Giới thiệu

Faster R-CNN là một mô hình **two-stage detector** nổi tiếng, được phát triển bởi Shaoqing Ren và cộng sự vào năm 2015, nhằm cải thiện tốc độ và độ chính xác so với các phiên bản R-CNN và Fast R-CNN trước đó. Mô hình này kết hợp backbone CNN mạnh mẽ để trích xuất đặc trưng, hệ thống anchor boxes đa dạng, **Region Proposal Network (RPN)** để tạo ra các vùng khả thi (region proposals), và **Detection Head (Fast R-CNN)** để phân loại và tinh chỉnh tọa độ bounding box.

2.3.2 Kiến trúc tổng quan

Mô hình bao gồm các thành phần chính sau:



Hình 2.13: Kiến trúc tổng thể của Faster R-CNN.

- **Backbone CNN:** Trích xuất đặc trưng từ ảnh đầu vào.
- **Anchor Boxes:** Các bounding box tham chiếu đa dạng về tỷ lệ và kích thước.

- **Region Proposal Network (RPN):** Sinh ra các region proposals từ feature map.
- **ROI Pooling / ROI Align:** Chuẩn hóa kích thước các region proposals.
- **Detection Head (Fast R-CNN):** Phân loại đối tượng và tinh chỉnh bounding box.
- **Loss function đa nhiệm (multi-task loss):** Kết hợp classification và bounding box regression.

a. Backbone CNN

Backbone là thành phần trung tâm của Faster R-CNN, chịu trách nhiệm **trích xuất đặc trưng (feature extraction)** từ ảnh đầu vào. Thông thường, backbone là một mạng CNN sâu đã được chứng minh hiệu quả trong các bài toán nhận diện ảnh, ví dụ như **VGG**, **ResNet** hoặc **EfficientNet**. Chức năng của backbone bao gồm:

- **Lớp convolution:** Các lớp convolution học được các đặc trưng từ mức cơ bản đến phức tạp. Lớp convolution đầu tiên thường phát hiện các cạnh, góc và texture. Các lớp sâu hơn kết hợp các đặc trưng này để nhận diện các hình dạng phức tạp hoặc các phần của đối tượng. Mỗi filter trong convolution học một kiểu mẫu riêng biệt, giúp mạng xây dựng biểu diễn phong phú của ảnh.
- **Lớp pooling:** Pooling, như max pooling hoặc average pooling, giảm độ phân giải của feature map, vừa giảm số lượng tính toán vừa giúp mạng nhận diện đối tượng với độ dịch chuyển hoặc biến dạng nhỏ. Pooling giữ lại các thông tin quan trọng trong vùng nhỏ của feature map, tăng khả năng tổng quát hóa của mạng.
- **Feature map:** Output cuối cùng của backbone là một *feature map* với độ phân giải thấp hơn ảnh gốc nhưng chứa thông tin ngữ nghĩa phong phú. Feature map này được sử dụng làm đầu vào cho **Region Proposal Network (RPN)** và **detection head**, thay vì dùng trực tiếp ảnh gốc. Việc này giúp giảm đáng kể khối lượng tính toán mà vẫn giữ được các đặc trưng quan trọng để phát hiện đối tượng.
- **Fine-tuning và transfer learning:** Backbone có thể được huấn luyện từ đầu hoặc **fine-tune** từ mô hình pretrained trên tập dữ liệu lớn như ImageNet. Việc sử dụng pretrained weights giúp mạng học nhanh hơn và đạt độ chính xác cao hơn, đặc biệt khi dữ liệu huấn luyện hạn chế.

Vai trò tổng thể: Backbone không chỉ cung cấp feature map chất lượng cao mà còn quyết định hiệu năng của toàn bộ Faster R-CNN. Một backbone mạnh giúp RPN tạo ra region proposals chính xác hơn và detection head phân loại đối tượng hiệu quả hơn. Vì vậy, lựa chọn backbone phù hợp với bài toán cụ thể là yếu tố quan trọng trong thiết kế mô hình.

b. Anchor Boxes

Trong Faster R-CNN, **anchor boxes** đóng vai trò là các *bounding box* tham chiếu cố định tại mỗi điểm trên feature map. Chúng được thiết kế với nhiều **tỷ lệ (aspect ratios)** và **kích thước (scales)** khác nhau để phù hợp với các đối tượng có hình dạng và kích thước đa dạng trong ảnh. Ví dụ, sử dụng 3 tỷ lệ \times 3 kích thước \rightarrow tạo 9 anchor boxes tại mỗi điểm trên feature map. Việc sử dụng anchor boxes giúp RPN sinh ra *region proposals* một cách hiệu quả, mà không cần phải quét toàn bộ ảnh với nhiều khung khả năng.

Chức năng của anchor boxes bao gồm:

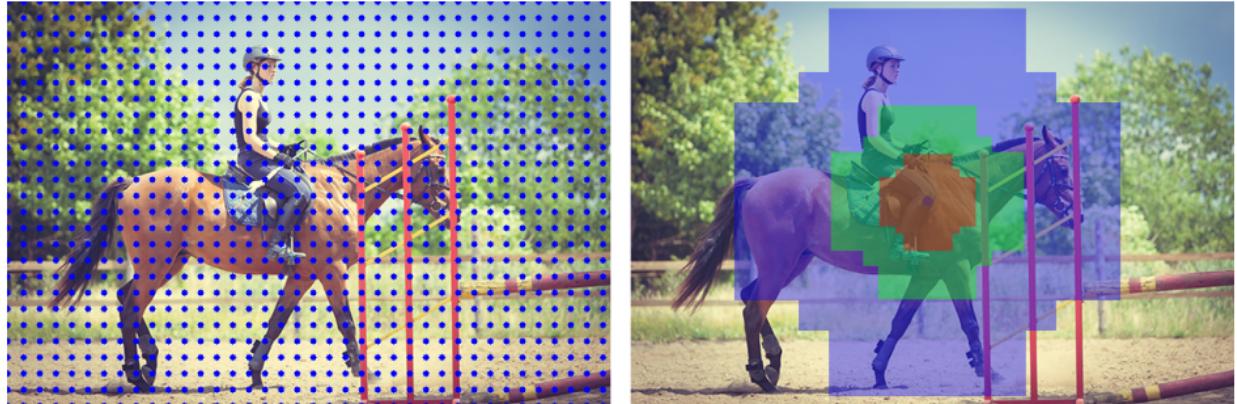
- **Đa dạng đối tượng:** Anchor cho phép mạng phát hiện đồng thời các đối tượng nhỏ, trung bình và lớn. Nhờ việc kết hợp nhiều tỷ lệ và kích thước, các anchor này có khả năng bao phủ hầu hết các hình dạng và tỷ lệ xuất hiện trong dữ liệu huấn luyện.
- **Objectness score:** Mỗi anchor được đánh giá bằng một *objectness score* p_i , phản ánh xác suất anchor đó chứa đối tượng. Objectness score giúp RPN lọc ra các anchor có khả năng chứa đối tượng cao để tạo proposals chất lượng.
- **Bounding box regression:** Anchor boxes không nhất thiết phải khớp hoàn toàn với vị trí thực tế của đối tượng. RPN dự đoán các độ lệch (t_x, t_y, t_w, t_h) để điều chỉnh anchor về đúng bounding box thật:

$$t_x = \frac{x - x_a}{w_a}, \quad t_y = \frac{y - y_a}{h_a}, \quad t_w = \log \frac{w}{w_a}, \quad t_h = \log \frac{h}{h_a},$$

trong đó (x, y, w, h) là tọa độ và kích thước của bounding box thật, còn (x_a, y_a, w_a, h_a) là của anchor. Công thức này giúp mạng học được cách tinh chỉnh vị trí và kích thước anchor để khớp với đối tượng thực tế, cải thiện độ chính xác của proposals.

- **Chuẩn bị cho RPN:** Anchor boxes cung cấp nền tảng để Region Proposal Network quét toàn bộ feature map, dự đoán các proposals, và sau đó chọn các proposal tốt nhất thông qua **Non-Maximum Suppression (NMS)**. Nhờ đó, mạng có thể tập trung vào các vùng tiềm năng chứa đối tượng mà không phải xử lý toàn bộ ảnh.

Vai trò tổng thể: Anchor boxes là thành phần then chốt trong Faster R-CNN, giúp mô hình trở thành *two-stage detector* hiệu quả. Chúng đảm bảo RPN có thể sinh ra proposals chất lượng cao cho nhiều loại đối tượng khác nhau, từ đó giúp detection head phân loại và tinh chỉnh bounding box chính xác hơn. Lựa chọn số lượng, tỷ lệ và kích thước anchor phù hợp với dữ liệu cụ thể là yếu tố quan trọng để đạt hiệu năng tốt nhất.



Hình 2.14: Vùng cảm nhận và Các điểm trung tâm Anchor Box

c. Intersection over Union (IoU)

IoU được sử dụng trong bài toán object detection, để đánh giá xem bounding box dự đoán đối tượng khớp với ground truth thật của đối tượng.

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

Trong Faster R-CNN, IoU dùng để:

- Chỉ số IoU trong khoảng [0,1], IoU càng gần 1 thì bounding box dự đoán càng gần ground truth
- Là cơ sở để huấn luyện RPN và loại bỏ các anchor dư thừa trong NMS.



Hình 2.15: Minh họa IoU giữa bounding box dự đoán và ground truth

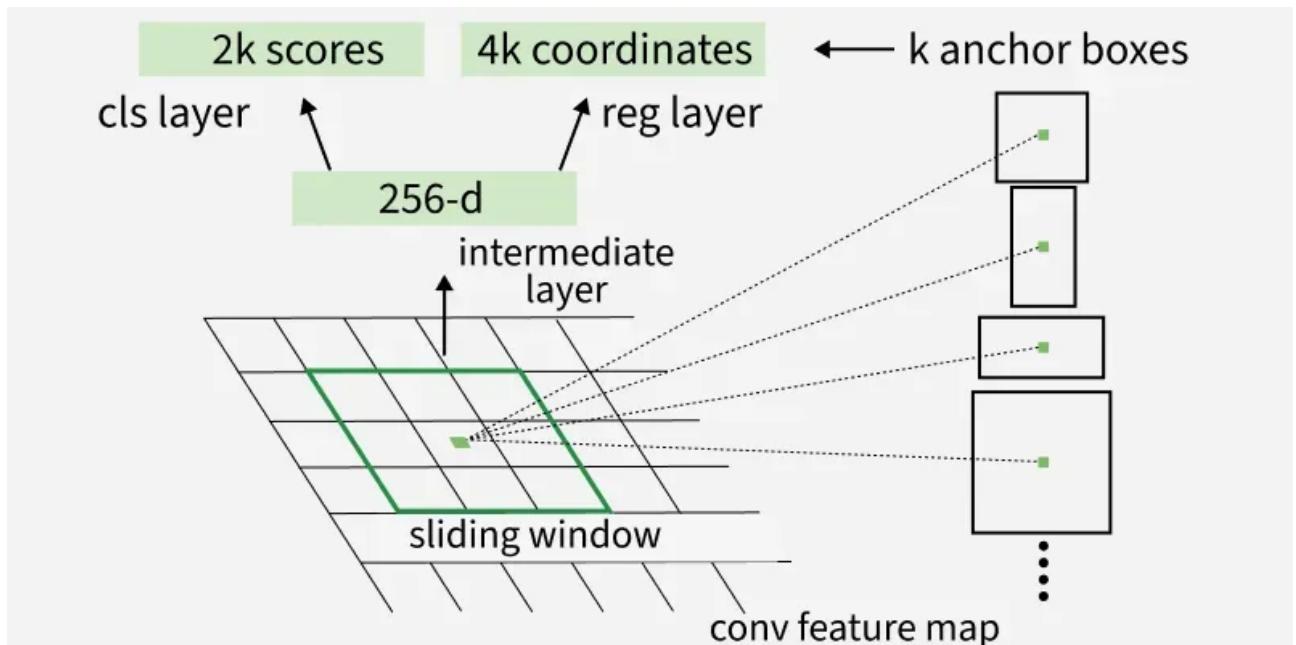
d. Region Proposal Network (RPN)

RPN là một mạng CNN nhỏ quét feature map sinh ra các *region proposals*:

1. Tại mỗi điểm trên feature map, RPN đánh giá tất cả các **anchor boxes**, dự đoán:
 - **Objectness score:** xác suất anchor chứa đối tượng.
 - **Bounding box offsets:** các giá trị hiệu chỉnh để tinh chỉnh anchor về gần với ground-truth box.
2. Mỗi anchor được gán nhãn dựa trên *IoU* với ground-truth box:

$$\text{label} = \begin{cases} 1 & \text{nếu IoU} \geq 0.7 \\ 0 & \text{nếu IoU} \leq 0.3 \\ \text{ignore} & \text{trường hợp còn lại} \end{cases}$$

3. Các region proposals trùng lặp được loại bỏ bằng **Non-Maximum Suppression (NMS)**, giữ lại những proposal có điểm cao nhất.



Hình 2.16: Region Proposal Network(RPN)

e. ROI Pooling / ROI Align

Region proposals có kích thước khác nhau, trong khi detection head yêu cầu đầu vào cố định:

- **ROI Pooling:** Chia proposal thành lưới cố định (ví dụ 7x7), áp dụng max pooling.
- **ROI Align:** Cải tiến ROI Pooling bằng **bilinear interpolation**, tránh sai số lượng tử hóa, giữ thông tin không gian chính xác.

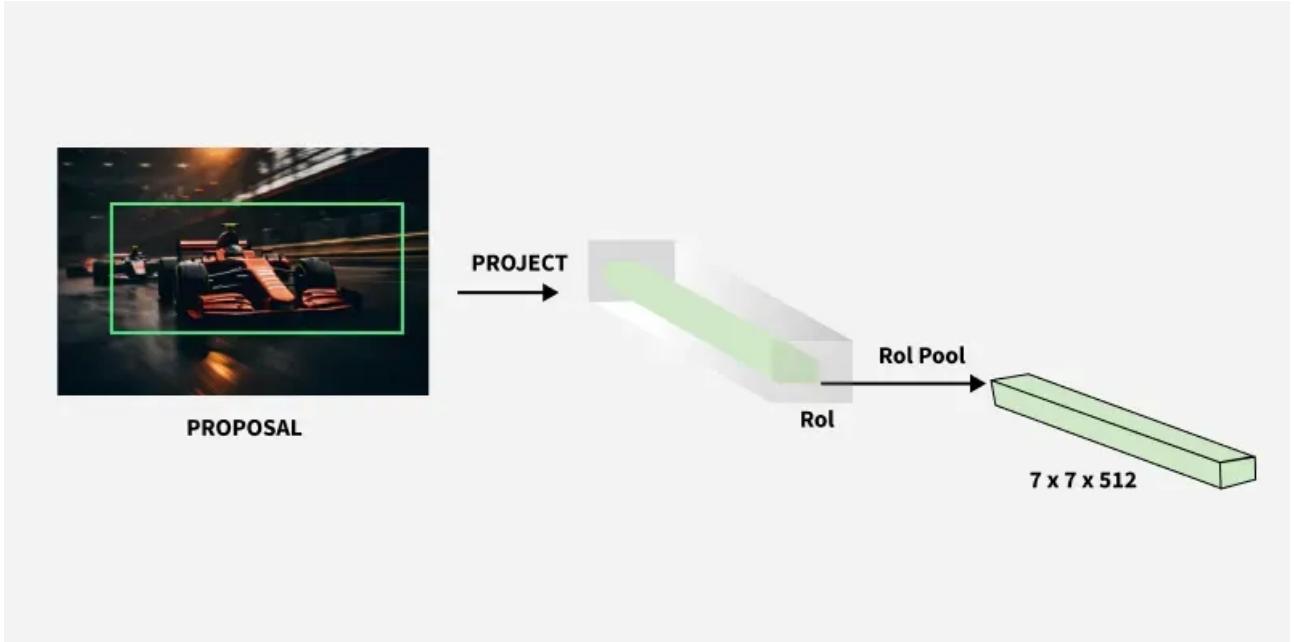
f. Detection Head (Fast R-CNN)

Detection head nhận feature map chuẩn hóa từ ROI Align, thực hiện:

1. **Classification:** dự đoán nhãn lớp đối tượng.
2. **Bounding Box Regression:** tinh chỉnh tọa độ bounding box:

$$(\hat{x}, \hat{y}, \hat{w}, \hat{h}) = (x_a + t_x w_a, y_a + t_y h_a, w_a e^{t_w}, h_a e^{t_h})$$

Detection head thường gồm các fully connected layers hoặc convolution nhỏ.



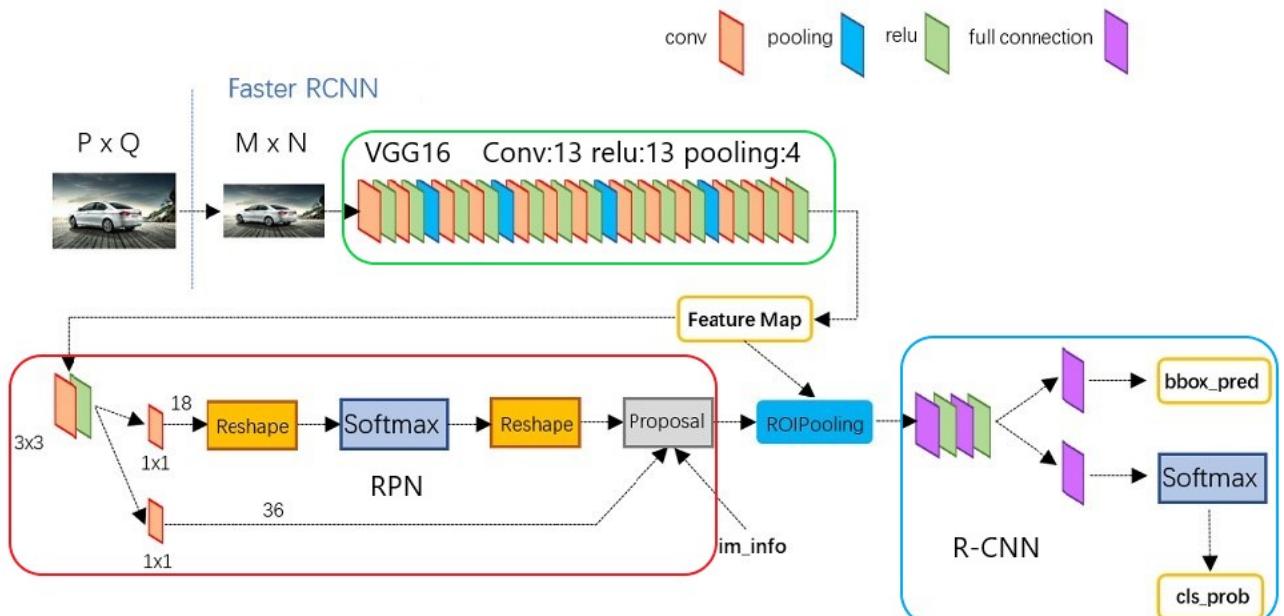
Hình 2.17: Region of Interest Pooling

g. Multi-task Loss

Faster R-CNN được huấn luyện bằng **multi-task loss**, kết hợp classification loss và bounding box regression loss:

$$L = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

- L_{cls} : softmax cross-entropy giữa nhãn dự đoán p_i và nhãn thật p_i^* .
- L_{reg} : Smooth L1 loss, chỉ áp dụng cho anchor foreground ($p_i^* = 1$).
- λ : hệ số cân bằng giữa hai thành phần loss.



Hình 2.18: Kiến trúc chi tiết của Faster R-CNN.

2.3.3 Ưu điểm

- **Độ chính xác cao:** Faster R-CNN là một trong những mô hình two-stage detector nổi tiếng với khả năng nhận diện đối tượng chính xác, đặc biệt là các đối tượng nhỏ, bị che khuất hoặc có hình dạng phức tạp. Nhờ sử dụng RPN sinh ra nhiều region proposals chất lượng cao và ROI Align giữ thông tin không gian, mô hình giảm thiểu các false positives và false negatives.
- **Huấn luyện end-to-end:** Cả backbone, RPN và detection head đều được huấn luyện đồng thời, giúp tối ưu hóa việc trích xuất đặc trưng và dự đoán bounding box cùng lúc. Điều này khác với R-CNN gốc, phải huấn luyện riêng từng bước, khiến Faster R-CNN hiệu quả hơn trong việc học đặc trưng và cải thiện độ chính xác.
- **Kiến trúc mô-đun:** Faster R-CNN có thiết kế mô-đun, cho phép dễ dàng thay thế backbone (ResNet, VGG, EfficientNet) hoặc tinh chỉnh RPN để cải thiện hiệu năng. Tính mô-đun cũng giúp mô hình linh hoạt trong các ứng dụng khác nhau, từ ảnh y tế đến giám sát giao thông.
- **Khả năng mở rộng:** Mô hình có thể xử lý nhiều loại đối tượng khác nhau nhờ hệ thống anchor boxes đa dạng về tỷ lệ và kích thước, đồng thời có thể tích hợp các kỹ thuật cải tiến như FPN (Feature Pyramid Network) để nhận diện đối tượng ở nhiều mức độ phân giải.

2.3.4 Nhược điểm

- **Tốc độ xử lý chậm:** Do Faster R-CNN là mô hình two-stage, mỗi ảnh phải qua RPN để sinh proposals, rồi qua detection head để phân loại và tinh chỉnh bounding box. Điều này khiến mô hình không phù hợp cho các ứng dụng real-time hoặc video với số lượng khung hình lớn.
- **Yêu cầu tài nguyên tính toán cao:** Việc sinh hàng nghìn anchor boxes và xử lý từng proposal đòi hỏi GPU mạnh và bộ nhớ lớn, đặc biệt khi backbone sâu hoặc ảnh đầu vào có độ phân giải cao.
- **Cấu hình phức tạp:** Số lượng anchor boxes, tỷ lệ, kích thước, threshold của RPN đều cần tinh chỉnh cẩn thận để đạt hiệu suất tốt nhất. Điều này làm cho việc triển khai và tuning mô hình trở nên khó khăn với người mới.
- **Khó triển khai real-time:** Dù chính xác, nhưng với các bài toán giám sát video trực tiếp, Faster R-CNN thường bị chậm so với các mô hình one-stage như YOLO, SSD, do phải xử lý từng proposal riêng biệt.

2.3.5 Ứng dụng

Faster R-CNN đặc biệt phù hợp với các bài toán mà độ chính xác trong nhận diện đối tượng được đặt lên hàng đầu, hơn là tốc độ xử lý. Một số ứng dụng tiêu biểu bao gồm:

- **Nhận diện phương tiện và biển số trong ảnh tĩnh:** Trong các hệ thống quản lý giao thông hoặc phân tích hình ảnh từ camera tĩnh, Faster R-CNN có thể phát hiện chính xác các phương tiện và biển số, kể cả khi bị che khuất hoặc góc chụp không thuận lợi. Mô hình có khả năng phân biệt nhiều loại phương tiện khác nhau nhờ anchor boxes đa dạng và feature map chi tiết từ backbone.
- **Phát hiện đối tượng trong ảnh y tế:** Trong lĩnh vực y tế, việc nhận diện các tổn thương, khối u hay bất thường trên ảnh X-ray, MRI, CT scan yêu cầu độ chính xác cao. Faster R-CNN giúp giảm tỷ lệ bỏ sót các đối tượng nhỏ, quan trọng trong chẩn đoán y tế.
- **Hệ thống giám sát an ninh (Surveillance):** Trong các hệ thống camera an ninh, Faster R-CNN có thể phân loại nhiều loại đối tượng khác nhau như con người, phương tiện, hành lý,... với độ chính xác cao. Điều này đặc biệt quan trọng khi cần phân tích dữ liệu video để phát hiện hành vi bất thường hoặc báo động sớm.
- **Các bài toán nghiên cứu và phân tích hình ảnh phức tạp:** Nhờ kiến trúc mô-đun, Faster R-CNN được sử dụng rộng rãi trong nghiên cứu về nhận diện đối tượng, tracking, và các bài toán computer vision phức tạp, nơi độ chính xác là tiêu chí quan trọng hơn tốc độ.

2.3.6 Kết luận

Faster R-CNN là một trong những mô hình two-stage detector nổi bật, cung cấp độ chính xác cao nhờ kết hợp RPN, ROI Align và detection head. Kiến trúc mô-đun cho phép thay đổi backbone và tinh chỉnh anchor boxes để tối ưu cho từng bài toán cụ thể. Tuy nhiên, nhược điểm về tốc độ xử lý, do phải xử lý hàng nghìn region proposals, khiến Faster R-CNN không phù hợp với các ứng dụng real-time, đặc biệt là giám sát giao

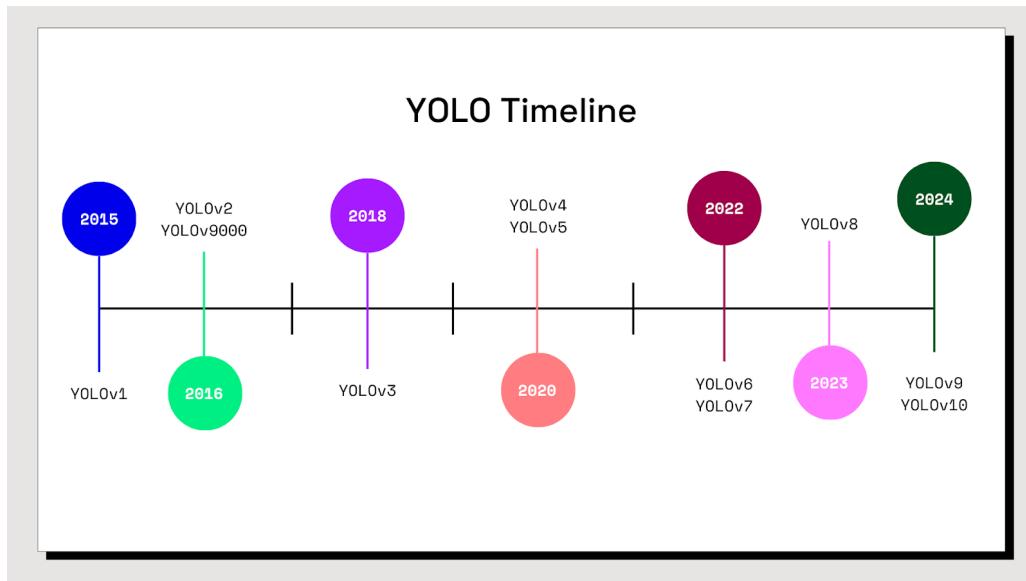
thông trực tiếp hoặc video streaming. Vì vậy, trong dự án này, để cân bằng giữa **độ chính xác** và **tốc độ**, mô hình YOLOv8 được lựa chọn. YOLOv8, là một mô hình one-stage detector, có khả năng dự đoán nhãn và bounding box trực tiếp trên toàn ảnh, giúp giảm đáng kể thời gian xử lý mà vẫn duy trì độ chính xác đủ cao cho các bài toán giám sát giao thông thực tế.

2.4 YOLOv8

2.4.1 Lịch sử phát triển của YOLO

Mô hình **YOLO** (**You Only Look Once**) được giới thiệu lần đầu bởi Joseph Redmon và cộng sự vào năm 2016. YOLO đặt ra một hướng đi mới trong object detection: thay vì tách riêng hai bước *region proposal* và *classification* như trong các two-stage detector (ví dụ Faster R-CNN), YOLO dự đoán trực tiếp bounding box và nhãn đối tượng trong một bước duy nhất, giúp tăng đáng kể tốc độ xử lý.

Các phiên bản YOLO phát triển qua các năm:



Hình 2.19: YOLO qua các năm.

- **YOLOv1 (2016):** bản đầu tiên, chia ảnh thành grid, mỗi grid dự đoán bounding box và class probability. **Ưu điểm:** tốc độ nhanh; **nhược điểm:** độ chính xác thấp với các object nhỏ và hình dạng phức tạp.
- **YOLOv2 / YOLO9000 (2017):** cải tiến backbone (Darknet-19), sử dụng anchor boxes và batch normalization, tăng độ chính xác và khả năng nhận diện nhiều lớp hơn.
- **YOLOv3 (2018):** sử dụng Darknet-53 làm backbone, dự đoán ở nhiều scale khác nhau (multi-scale), cải thiện khả năng phát hiện object nhỏ.
- **YOLOv4 (2020):** áp dụng các kỹ thuật tối ưu hóa hiện đại như Mosaic augmentation, CSPDarknet53 backbone, PAN neck, Mish activation. Tăng đáng kể mAP mà vẫn giữ tốc độ cao.
- **YOLOv5 (2020, không chính thức từ tác giả YOLO):** triển khai bằng PyTorch, hỗ trợ nhiều biến thể (s,m,l,x), dễ huấn luyện, triển khai.
- **YOLOv6 / YOLOv7 (2022):** cải thiện kiến trúc, tối ưu cho edge device, tăng hiệu năng inference.
- **YOLOv8 (2023):** phiên bản mới nhất, thuộc nhóm one-stage detector. Một số điểm nổi bật:
 - Anchor-free head, giảm hyperparameter và cải thiện generalization.
 - Backbone CSP-like với module C2f, tăng biểu diễn đặc trưng nhưng nhẹ hơn.
 - Neck FPN + PAN cải tiến, hỗ trợ multi-scale feature fusion.
 - Hỗ trợ xuất mô hình sang ONNX, TensorRT, dễ triển khai real-time.

Nhìn chung, sự phát triển của YOLO hướng tới mục tiêu *tăng tốc độ* mà vẫn duy trì *độ chính xác cao*, phù hợp với các ứng dụng real-time và thiết bị nhúng.

Bảng 2.1: Giải thích các thuật ngữ liên quan đến YOLO

| Thuật ngữ | Giải thích |
|-------------------|--|
| Anchor-free | Thiết kế head không sử dụng anchor boxes, dự đoán box dựa trên điểm trung tâm. |
| Anchor boxes | Hộp tham chiếu với kích thước và tỉ lệ cố định dùng để dự đoán bounding box (YOLOv2 trở đi). |
| Backbone | Mạng trích xuất đặc trưng chính, cung cấp feature maps cho neck và head. |
| Bounding box | Hộp chữ nhật dùng để xác định vị trí đối tượng trong ảnh. |
| CSP / C2f | Cross Stage Partial, module cải tiến giúp giảm trùng lặp thông tin và cải thiện gradient flow. |
| Class probability | Xác suất dự đoán đối tượng thuộc lớp nào. |
| FPN / PAN | Feature Pyramid Network / Path Aggregation Network, kỹ thuật kết hợp feature maps đa tầng để phát hiện object nhiều scale. |
| Grid | Phương pháp chia hình ảnh thành các ô vuông nhỏ trong YOLOv1 để dự đoán bounding box. |
| Head | Phần cuối cùng dự đoán bounding box và nhãn lớp. |
| Neck | Bộ phận kết hợp feature maps từ nhiều tầng để hỗ trợ phát hiện đối tượng đa kích thước. |
| ONNX / TensorRT | Framework và công cụ hỗ trợ triển khai mô hình nhanh trên GPU hoặc edge device. |

2.4.2 Tổng quan về YOLOv8

YOLOv8 phiên bản mới nhất trong YOLO series của Ultralytics, được thiết kế để cải thiện hiệu suất phát hiện đối tượng thời gian thực với các tính năng tiên tiến. Là một mô hình tiên tiến, hiện đại (*state-of-the-art – SOTA*), YOLOv8 kế thừa thành công từ các phiên bản trước, đồng thời giới thiệu những tính năng và cải tiến mới để nâng cao hiệu suất, tính linh hoạt và hiệu quả.

Khác với các phiên bản trước, YOLOv8 tích hợp cơ chế không dùng điểm neo (*anchor-free*) với phần đầu Ultralytics được tách riêng, sử dụng các kiến trúc backbone và neck tiên tiến, đồng thời tối ưu hóa sự cân bằng giữa độ chính xác và tốc độ, làm cho nó trở thành lựa chọn lý tưởng cho nhiều ứng dụng khác nhau. Do đó, YOLOv8 có thể hỗ trợ toàn bộ các tác vụ vision AI, bao gồm phát hiện đối tượng, phân đoạn, ước tính tư thế, theo dõi và phân loại. Tính đa dạng này cho phép người dùng tận dụng khả năng của YOLOv8 trong nhiều ứng dụng và lĩnh vực khác nhau.

YOLOv8 kế thừa nhiều ưu điểm từ các phiên bản trước đó, đồng thời mang lại một số cải tiến quan trọng:

- **Tốc độ và hiệu suất:** YOLOv8 được thiết kế để hoạt động nhanh hơn và hiệu quả hơn, ngay cả trên các thiết bị có tài nguyên hạn chế như điện thoại di động hay các thiết bị nhúng. Nhờ vào kiến trúc nhẹ, nó có thể đạt tốc độ xử lý lên tới hàng trăm khung hình mỗi giây.
- **Độ chính xác cao hơn:** YOLOv8 cải thiện khả năng phát hiện đối tượng nhỏ và các đối tượng phức tạp hơn, giúp tăng độ chính xác tổng thể.
- **Dễ dàng tùy chỉnh:** YOLOv8 cung cấp khả năng dễ dàng điều chỉnh mô hình và các siêu tham số (*hyperparameters*) để phù hợp với các bài toán thực tế khác nhau. Điều này làm cho mô hình trở nên linh hoạt trong các ứng dụng từ phát hiện đối tượng đến phân loại, phân đoạn (*segmentation*), và *tracking* đối tượng.
- **Tích hợp tốt hơn với các framework mới:** YOLOv8 tận dụng tốt các công nghệ và framework mới như PyTorch, đồng thời hỗ trợ việc huấn luyện và triển khai dễ dàng hơn qua các file ONNX, TensorRT.

2.4.3 Các phiên bản của YOLOv8

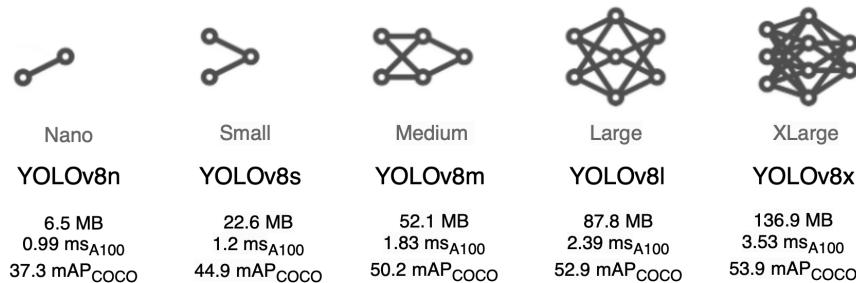
YOLOv8 có nhiều biến thể để phù hợp với yêu cầu tốc độ và hiệu năng khác nhau, cho phép người dùng lựa chọn mô hình tối ưu cho phần cứng và bài toán cụ thể:

- **YOLOv8-n (Nano):** Phiên bản nhỏ nhất, tối ưu tốc độ, phù hợp với các thiết bị GPU hạn chế hoặc các ứng dụng nhúng. Phiên bản này giảm số lượng tham số và FLOPs, giúp đạt FPS cao mà vẫn giữ được độ chính xác hợp lý.

Bảng 2.2: Thuật ngữ trong YOLOv8

| Thuật ngữ | Giải thích |
|-----------------------|---|
| Decoupled head | Head tách riêng nhánh dự đoán class và nhánh dự đoán bounding box, tối ưu hoá từng nhiệm vụ riêng biệt. |
| Inference / Real-time | Quá trình dự đoán đối tượng trên ảnh/video thực tế, tốc độ đủ nhanh để xử lý liên tục (real-time). |
| Segmentation | Phân đoạn đối tượng trong ảnh, gán nhãn pixel thay vì chỉ bounding box. |
| Tracking | Theo dõi đối tượng liên tục qua các khung hình video. |

- **YOLOv8-s (Small):** Cân bằng giữa tốc độ và độ chính xác, phù hợp cho các ứng dụng real-time phổ biến như camera giám sát, phân tích lưu lượng giao thông. Đây là lựa chọn phổ biến khi cần deploy nhanh mà không yêu cầu độ chính xác tối đa.
- **YOLOv8-m (Medium):** Phiên bản ưu tiên độ chính xác cao hơn, vẫn giữ tốc độ xử lý tương đối. Thích hợp khi dữ liệu phức tạp hoặc yêu cầu phát hiện small objects tốt hơn so với YOLOv8-s.
- **YOLOv8-l (Large):** Phiên bản tối ưu độ chính xác, chi phí tính toán cao hơn, thích hợp khi dữ liệu phức tạp, cần nhận diện nhiều đối tượng nhỏ, hoặc yêu cầu mAP cao nhất.



Hình 2.20: Các biến thể của YOLOv8.

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

Hình 2.21: Performance của mô hình YOLOv8 Detection được train trên tập COCO.

Việc chọn biến thể phù hợp phụ thuộc vào yêu cầu ứng dụng và tài nguyên phần cứng:

- Nếu yêu cầu *real-time* trên camera giám sát: thường chọn YOLOv8-n hoặc YOLOv8-s.
- Nếu ưu tiên *độ chính xác*: chọn YOLOv8-m hoặc YOLOv8-l, đặc biệt với dữ liệu phức tạp hoặc nhiều small objects.
- Trường hợp triển khai trên thiết bị nhúng hoặc IoT: YOLOv8-n là lựa chọn lý tưởng do khả năng tiết kiệm tài nguyên.

Các phiên bản này kế thừa toàn bộ cải tiến từ YOLOv8 như:

- Kiến trúc anchor-free và decoupled head.

- Backbone C2f và Neck multi-scale (FPN + PAN) giúp phát hiện small objects hiệu quả.
- Hỗ trợ augmentation, transfer learning và fine-tuning cho các dataset nhỏ.
- Dễ dàng export sang ONNX, TensorRT hoặc sử dụng PyTorch cho huấn luyện và triển khai.

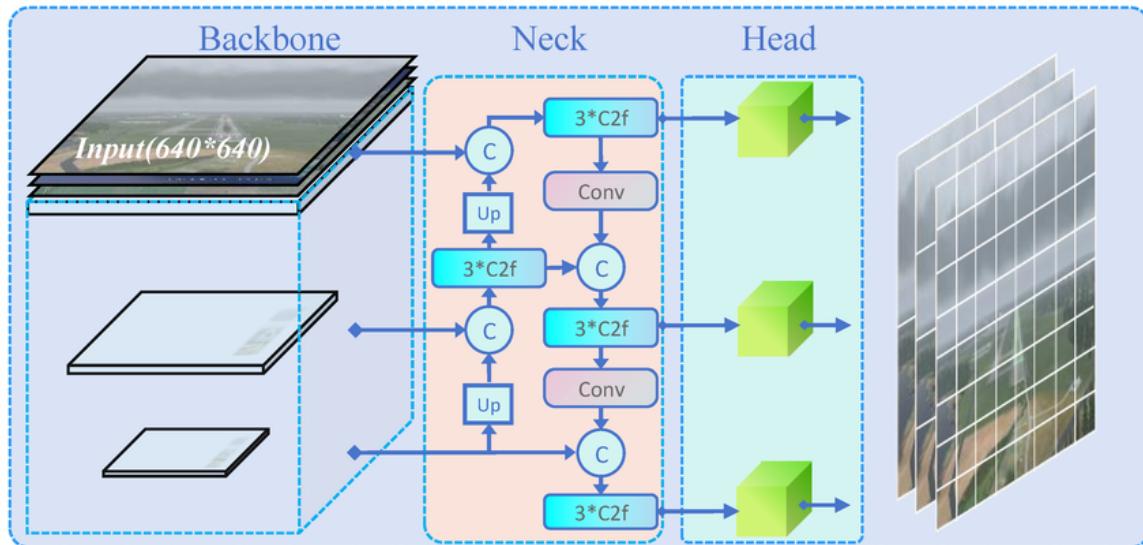
Bảng 2.3: Thuật ngữ trong các biến thể YOLOv8

| Thuật ngữ | Giải thích |
|-----------------------------------|---|
| FPS (Frames Per Second) | Số khung hình xử lý được trong 1 giây, dùng để đo tốc độ inference / real-time của mô hình. |
| FLOPs (Floating Point Operations) | Tổng số phép tính số thực cần thực hiện trong mô hình, dùng để đánh giá độ phức tạp tính toán. |
| mAP (mean Average Precision) | Chỉ số đánh giá độ chính xác tổng thể của mô hình phát hiện đối tượng, tính trung bình trên tất cả các lớp. |
| Small objects | Các đối tượng nhỏ trong ảnh hoặc video, thường khó phát hiện, yêu cầu mô hình multi-scale và feature fusion hiệu quả. |
| Edge device / IoT | Thiết bị nhúng hoặc thiết bị giới hạn tài nguyên (GPU/CPU hạn chế), nơi cần mô hình nhẹ và tối ưu. |

2.4.4 Kiến trúc YOLOv8

Kiến trúc YOLOv8 là một thiết kế **state-of-the-art** (SOTA) nâng cao khả năng phát hiện đối tượng thông qua cấu trúc hiệu quả, bao gồm ba thành phần chính: **Backbone**, **Neck**, và **Head**.

Mỗi phần đều đóng vai trò quan trọng trong việc xử lý và diễn giải dữ liệu hình ảnh để cung cấp kết quả chính xác.



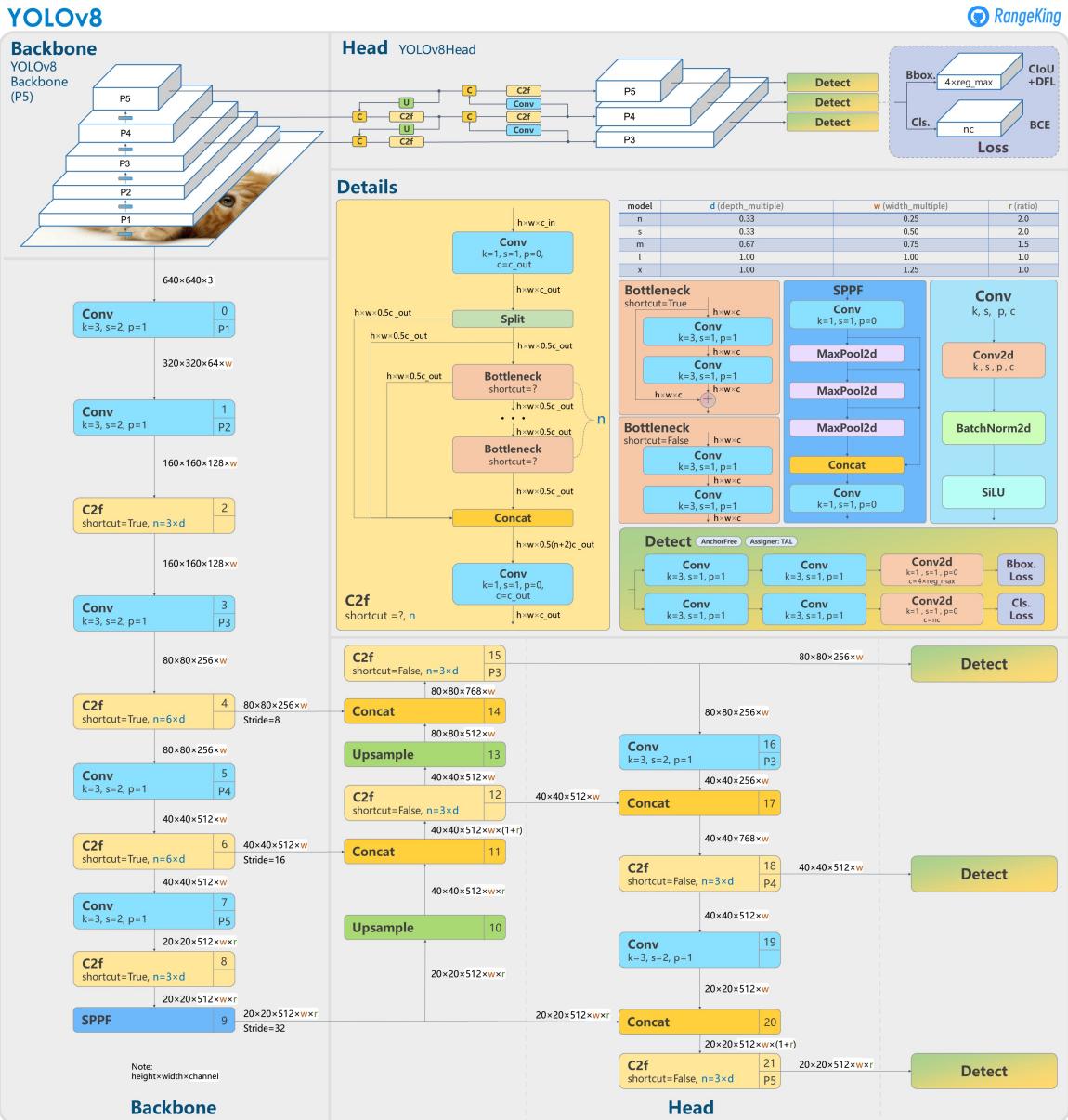
Hình 2.22: Kiến trúc của YOLOv8.

a. Backbone

Khái niệm Backbone trong YOLOv8, còn gọi là **feature extractor** (bộ trích xuất đặc trưng), là phần đầu tiên của mạng neural. Nhiệm vụ chính là *trích xuất các đặc trưng quan trọng từ ảnh đầu vào*, chuyển dữ liệu ảnh thô thành các feature map mà các tầng tiếp theo có thể sử dụng. Nếu ví ảnh đầu vào là “nguyên liệu thô”, thì Backbone chính là nơi “xay lọc” ra những thành phần quan trọng.

Hoạt động chính

- **Trích xuất đặc trưng (Feature Extraction):** Các tầng đầu dùng các convolutional layers để phát hiện các mẫu cơ bản như cạnh, góc, kết cấu, màu sắc. Đây là nền tảng để mô hình phát hiện các đối tượng phức tạp hơn ở các tầng sâu hơn.



Hình 2.23: Kiến trúc của YOLOv8.

- Biểu diễn phân cấp (Hierarchical Representation):** Khi ảnh đi qua các tầng tiếp theo, feature map trở nên trừu tượng hơn:
 - Tầng đầu: cạnh, góc, kết cấu.
 - Tầng giữa: các bộ phận của đối tượng (ví dụ bánh xe, cửa ô tô).
 - Tầng cuối: toàn bộ đối tượng hoặc các mối quan hệ giữa đối tượng.

Điều này giúp mô hình hiểu thông tin ở nhiều mức độ, từ chi tiết nhỏ đến tổng thể, rất quan trọng cho việc phát hiện đối tượng chính xác, đặc biệt là *small objects*.

- Đầu ra phong phú (Rich Output):** Kết quả là một feature map giàu thông tin, chứa các đặc trưng từ nhiều cấp độ. Feature map này sẽ là nền tảng cho Neck và Head để tiếp tục xử lý như gộp thông tin (fusion), dự đoán bounding box, xác suất lớp, v.v.

Chi tiết kiến trúc YOLOv8 sử dụng **CSPDarknet53** tùy chỉnh làm Backbone.

- CSP (Cross Stage Partial):** Chia feature map thành hai nhánh, một nhánh đi thẳng, nhánh còn lại đi qua một chuỗi convolution, sau đó gộp lại.
 - Giữ luồng gradient ổn định.

- Tránh thông tin trùng lặp.
- Giảm số lượng tham số nhưng vẫn tăng biểu diễn đặc trưng.
- Module CSP giúp cải thiện độ chính xác phát hiện, đặc biệt với các đối tượng nhỏ hoặc vùng bị che khuất.

b. Neck

Khái niệm Neck trong YOLOv8 đóng vai trò là cầu nối giữa Backbone và Head. Nhiệm vụ chính là *thực hiện feature fusion* và tích hợp thông tin ngữ cảnh, nhằm cải thiện khả năng phát hiện đối tượng ở nhiều kích thước và điều kiện khác nhau. Nếu Backbone tạo ra feature map giàu thông tin, thì Neck là nơi *kết hợp và tinh chỉnh* những feature đó trước khi đưa vào Head để dự đoán.

Hoạt động chính

- **Feature Fusion:** Kết hợp các feature map từ các tầng khác nhau của Backbone.
 - Cho phép mô hình phát hiện các đối tượng có kích thước khác nhau (*multi-scale detection*).
 - Tầng sâu cung cấp thông tin semantic-rich (ngữ nghĩa cao), tầng nông cung cấp chi tiết không gian (spatial detail).
- **Tích hợp ngữ cảnh (Context Integration):** Kết hợp thông tin tổng thể của cảnh.
 - Giúp mô hình hiểu mối quan hệ giữa các đối tượng và bối cảnh xung quanh.
 - Cải thiện độ chính xác, đặc biệt khi đối tượng nhỏ hoặc bị che khuất.
- **Giảm chiều dữ liệu (Dimensionality Reduction):** Giảm độ phân giải không gian và số chiều của feature map.
 - Giúp giảm tải tính toán, tăng tốc độ xử lý.
 - Tuy nhiên, có thể làm mất một phần thông tin chi tiết, trade-off giữa tốc độ và độ chính xác.

Chi tiết kiến trúc Thay vì sử dụng Feature Pyramid Network (FPN) truyền thống, YOLOv8 sử dụng module **C2f**. Module này kết hợp hiệu quả các đặc trưng semantic cao cấp với thông tin không gian thấp, nhờ đó:

- Nâng cao hiệu năng nhận diện, đặc biệt với các *small objects*.
- Tối ưu hóa luồng thông tin giữa các tầng.
- Giữ được tốc độ xử lý cao nhờ cấu trúc nhẹ.

c. Head

Khái niệm Head là phần cuối cùng của mạng YOLOv8, chịu trách nhiệm *tạo ra đầu ra cuối cùng* như bounding box, điểm tin cậy (confidence score) và nhãn lớp (class label) cho từng đối tượng trong ảnh. Đây là bộ phận quan trọng, quyết định chính xác “cái gì và ở đâu” trong hình ảnh.

Hoạt động chính

- **Sinh bounding box (Bounding Box Generation):**
 - Head tạo các hộp bao quanh các đối tượng tiềm năng dựa trên feature map nhận được từ Neck.
 - Mỗi bounding box chứa thông tin: tọa độ trung tâm (x, y), chiều rộng w , chiều cao h .
 - Thuật toán dự đoán dựa trên các điểm đặc trưng (anchor-free) mà không cần predefined anchors.
 - Đây là bước nền tảng để đánh giá các đối tượng và xác định chính xác vùng xuất hiện của chúng.
- **Đánh giá độ tin cậy (Confidence Scoring):**
 - Mỗi bounding box được gán một *objectness score* biểu thị xác suất có đối tượng.
 - Giúp mô hình nhận biết các hộp có chứa đối tượng thực sự hay chỉ là nền/background.
 - Điểm tin cậy này thường được sử dụng kết hợp với class probability để tính *final score*:

$$\text{final score} = \text{confidence score} \times \text{class probability}.$$

- Các bounding box có điểm thấp sẽ bị loại bỏ trong bước *Non-Maximum Suppression (NMS)*.

- **Phân loại đối tượng (Category Sorting / Class Prediction):**

- Dự đoán nhãn lớp (class probability) cho từng bounding box.
- Cho phép Head xác định loại đối tượng, ví dụ: người, xe, động vật, đồ vật.
- Dựa trên các đặc trưng semantic-rich từ Neck, head có thể phân biệt các đối tượng có hình dạng hoặc kích thước tương tự.
- Cải thiện khả năng phát hiện các small objects hoặc objects bị che khuất nhờ thông tin multi-scale.

Chi tiết kiến trúc

- YOLOv8 sử dụng *decoupled head*, tách riêng hai nhánh:
 - Nhánh regression: dự đoán bounding box và objectness score.
 - Nhánh classification: dự đoán class probability.
- Việc tách riêng giúp tối ưu hóa từng nhiệm vụ, cải thiện tốc độ hội tụ và độ chính xác.
- Head hoạt động trên nhiều scale khác nhau, nhờ Neck cung cấp feature maps đa chiều.
- Các dự đoán được tổng hợp (aggregation) và lọc qua NMS/Soft-NMS để loại bỏ các box trùng lặp, giữ lại các dự đoán chất lượng nhất.
- Hỗ trợ tính năng anchor-free: head không phụ thuộc vào anchor boxes cố định, giúp cải thiện khả năng generalization với các aspect ratio bất thường.

2.4.5 Hàm mất mát (Loss) và tối ưu:

YOLOv8 sử dụng một hàm mất mát tổng hợp, kết hợp ba thành phần chính: loss hồi quy bounding box, loss phân phối tọa độ (Distribution Focal Loss), và loss phân loại. Hàm mất mát tổng thể có dạng:

$$\mathcal{L} = \lambda_{\text{box}} \mathcal{L}_{\text{box}} + \lambda_{\text{cls}} \mathcal{L}_{\text{cls}} + \lambda_{\text{dfl}} \mathcal{L}_{\text{DFL}}$$

trong đó các hệ số λ được điều chỉnh để cân bằng giữa độ chính xác định vị và phân lớp.

- **Box regression loss – IoU-based Loss**

YOLOv8 sử dụng các biến thể của IoU như CIoU hoặc GIoU để tối ưu vị trí, kích thước và độ trùng khớp của bounding box. Ví dụ hàm CIoU có dạng:

$$\mathcal{L}_{\text{box}} = 1 - \text{CIoU} = 1 - \left(\text{IoU} - \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} - \alpha v \right)$$

trong đó:

$$v = \frac{4}{\pi^2} (\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h})^2, \quad \alpha = \frac{v}{(1 - \text{IoU}) + v}$$

ρ là khoảng cách giữa tâm hai box, c là đường chéo nhỏ nhất bao hai box.

- **Distribution Focal Loss (DFL)**

YOLOv8 không dự đoán trực tiếp tọa độ x, y, w, h , mà dự đoán phân phối rời rạc trên các bins (như trong Generalized Focal Loss). DFL được tính như cross-entropy giữa phân phối dự đoán và GT “soft label”:

$$\mathcal{L}_{\text{DFL}} = - \sum_{i=1}^K q_i \log(p_i)$$

trong đó q_i là phân phối mục tiêu (two-hot), p_i là phân phối mô hình dự đoán, K là số bin dùng để mã hóa tọa độ.

- Classification Loss – Varifocal / Focal Loss

YOLOv8 sử dụng các biến thể Focal Loss để xử lý class imbalance và tập trung vào các mẫu khó:

$$\mathcal{L}_{\text{cls}} = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

với:

$$p_t = \begin{cases} p & \text{nếu mẫu đúng lớp,} \\ 1 - p & \text{nếu mẫu sai lớp.} \end{cases}$$

Như vậy, YOLOv8 tối ưu đồng thời vị trí, phân phối tọa độ và xác suất phân lớp, giúp mô hình hội tụ ổn định và đạt mAP cao trên các tác vụ detection.

2.4.6 Kỹ thuật huấn luyện và Augmentation

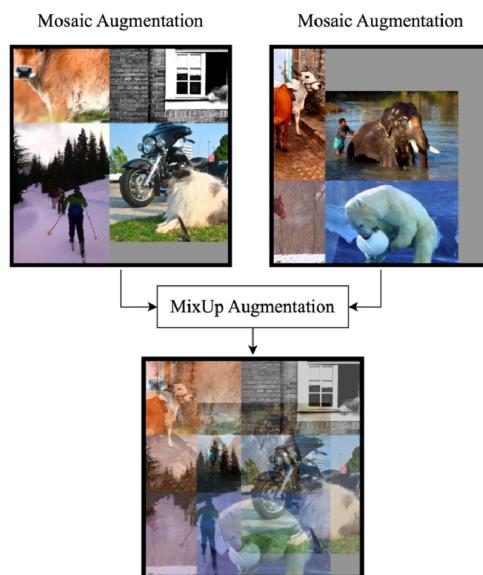
Mục tiêu: Các kỹ thuật huấn luyện và augmentation giúp YOLOv8 *tăng khả năng generalization*, giảm overfitting và cải thiện hiệu suất phát hiện đối tượng trong các điều kiện thực tế đa dạng.

Các phương pháp augmentation chính:

- Mosaic
- MixUp
- Flip
- Hue/Brightness/Contrast/Saturation/Blur
- Label smoothing, class balancing, Focal Loss

a. Mosaic Augmentation

Khái niệm: Mosaic là một kỹ thuật augmentation độc đáo được YOLOv4 giới thiệu và tiếp tục được áp dụng trong YOLOv8. Ý tưởng chính là *ghép 4 ảnh khác nhau thành một ảnh lớn duy nhất* theo tỉ lệ và vị trí ngẫu nhiên. Điều này giúp mô hình học được các mối quan hệ ngữ cảnh giữa các đối tượng trong nhiều cảnh khác nhau, đồng thời cải thiện khả năng phát hiện *small objects*.



Hình 2.24: Mosaic & MixUp.

Cách hoạt động:

1. Chọn ngẫu nhiên 4 ảnh từ dataset.
2. Chia mỗi ảnh thành $1/4$ của ảnh đầu ra.
3. Ghép 4 ảnh lại để tạo thành một ảnh mới (thường là 2×2).
4. Điều chỉnh bounding box của từng đối tượng tương ứng với vị trí mới trong ảnh ghép.
5. Có thể áp dụng kèm các augmentation khác như Flip, Hue/Brightness, Blur trên ảnh ghép.

Ưu điểm:

- **Tăng tính đa dạng cảnh:** Mô hình nhìn thấy đối tượng trong nhiều ngữ cảnh khác nhau cùng lúc.
- **Hỗ trợ phát hiện small objects:** Khi ghép 4 ảnh, một số đối tượng nhỏ sẽ chiếm nhiều diện tích hơn so với ảnh gốc, giúp mô hình dễ học.
- **Tiết kiệm bộ nhớ:** Thay vì tăng batch size, Mosaic tăng đa dạng dữ liệu mà không làm tăng quá nhiều bộ nhớ.
- **Học mối quan hệ giữa các đối tượng:** Các đối tượng từ nhiều ảnh khác nhau xuất hiện cùng nhau, mô hình học được ngữ cảnh phong phú.

Nhược điểm:

- **Giới hạn kích thước và tỉ lệ đối tượng:** Khi ghép 4 ảnh vào một, một số đối tượng nhỏ có thể bị thu nhỏ quá mức, làm giảm khả năng phát hiện.
- **Tăng độ phức tạp ngữ cảnh:** Mosaic tạo ra cảnh tổng hợp không tự nhiên, đôi khi khiến mô hình học các mối quan hệ không có thật giữa các đối tượng.
- **Khó cân bằng nhãn:** Một số lớp có thể xuất hiện ít hơn hoặc nhiều hơn trong một ảnh tổng hợp, dẫn đến imbalance tạm thời trong batch.
- **Tăng yêu cầu tính toán:** Ghép 4 ảnh vào một yêu cầu nhiều thao tác tiền xử lý hơn so với ảnh đơn lẻ, làm tăng thời gian chuẩn bị batch.

b. MixUp Augmentation

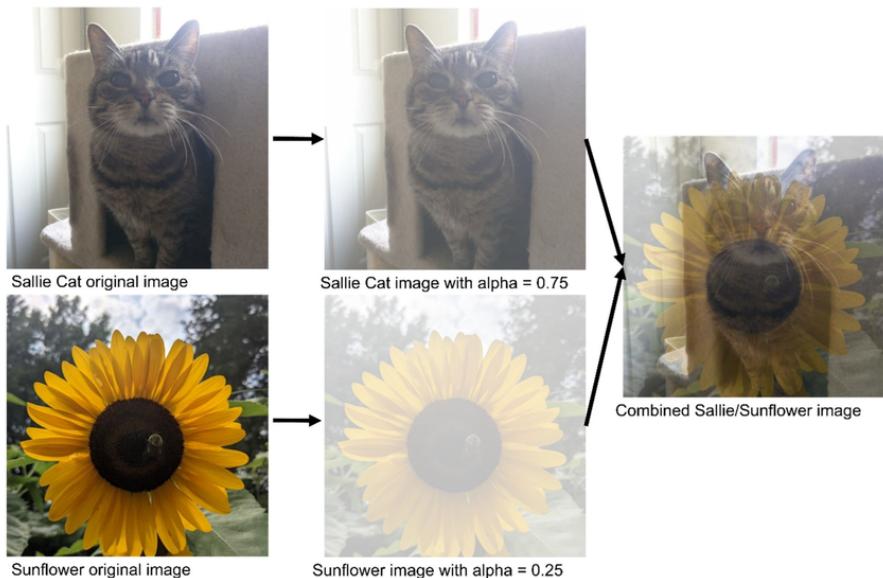
Khái niệm: MixUp là một kỹ thuật augmentation được sử dụng để cải thiện khả năng tổng quát hóa (generalization) của mô hình. Thay vì chỉ dùng một ảnh, MixUp *trộn hai ảnh và nhãn tương ứng* để tạo ra một ảnh mới. Điều này giúp mô hình học được các ranh giới mềm giữa các lớp và giảm overfitting.

Cách hoạt động:

1. Chọn ngẫu nhiên hai ảnh I_1 và I_2 từ dataset.
2. Chọn ngẫu nhiên một hệ số trộn $\lambda \in [0, 1]$ (thường lấy từ phân phối Beta).
3. Tạo ảnh mới $I_{mix} = \lambda I_1 + (1 - \lambda)I_2$.
4. Nhãn tương ứng cũng được trộn theo cùng tỉ lệ: $y_{mix} = \lambda y_1 + (1 - \lambda)y_2$.
5. Điều chỉnh bounding box nếu cần, đặc biệt với các đối tượng xuất hiện trong cả hai ảnh.

Ưu điểm:

- **Giảm overfitting:** Trộn ảnh và nhãn giúp mô hình không quá phụ thuộc vào các mẫu dữ liệu cụ thể.
- **Học ranh giới mềm:** Mô hình học được rằng đối tượng có thể xuất hiện với các mức độ kết hợp khác nhau, tăng khả năng dự đoán linh hoạt.
- **Tăng tính đa dạng dữ liệu:** Mỗi ảnh trộn là một trường hợp mới, giúp dataset trở nên phong phú hơn.
- **Hỗ trợ small objects:** Khi một ảnh có đối tượng nhỏ được trộn với ảnh khác, mô hình vẫn có cơ hội học được các đặc trưng của đối tượng nhỏ đó.



Hình 2.25: MixUp.

Nhược điểm:

- Nhầm lẫn nhãn (Label Confusion):** Khi trộn nhiều ảnh, nhãn trở nên mềm, đôi khi gây khó khăn cho mô hình trong việc phân biệt ranh giới giữa các lớp.
- Giảm chi tiết của đối tượng nhỏ:** Nếu một đối tượng nhỏ bị trộn với ảnh khác lớn hơn, đặc trưng của nó có thể bị nhòe, giảm khả năng nhận diện.
- Không phù hợp với tất cả loại bài toán:** MixUp hiệu quả với classification, detection nhưng với các bài toán yêu cầu vị trí chính xác cực cao hoặc segmentation chi tiết, cần phải điều chỉnh bounding box/mask cẩn thận.

c. Flip (Lật ảnh):

Khái niệm: Flip là kỹ thuật augmentation đơn giản nhưng hiệu quả, lật ảnh theo chiều ngang, chiều dọc hoặc cả hai. Giúp mô hình học được các biến thể đối tượng xuất hiện ở các hướng khác nhau.

Hoạt động:

- Horizontal Flip:** Lật ảnh theo trục ngang (trái – phải).
- Vertical Flip:** Lật ảnh theo trục dọc (trên – dưới).
- Combination Flip:** Kết hợp cả hai để tăng đa dạng dữ liệu.

Ưu điểm:

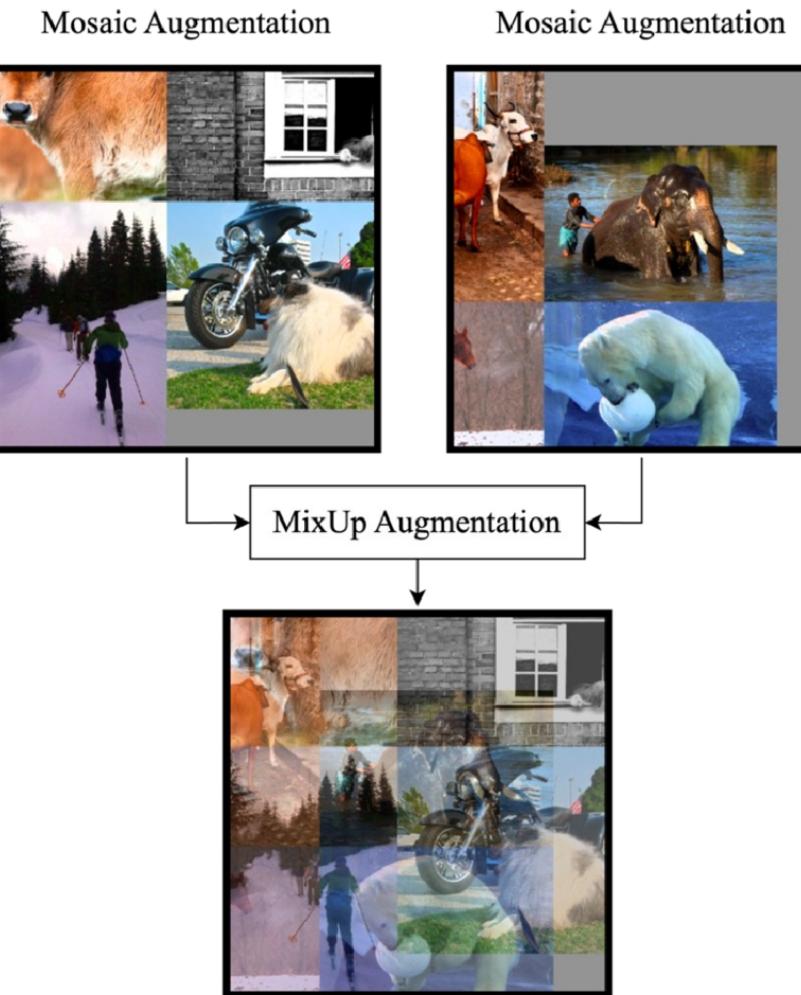
- Giúp mô hình nhận diện đối tượng xuất hiện theo nhiều hướng.
- Hữu ích cho dữ liệu ít và các object xuất hiện theo hướng không đồng nhất.

Nhược điểm:

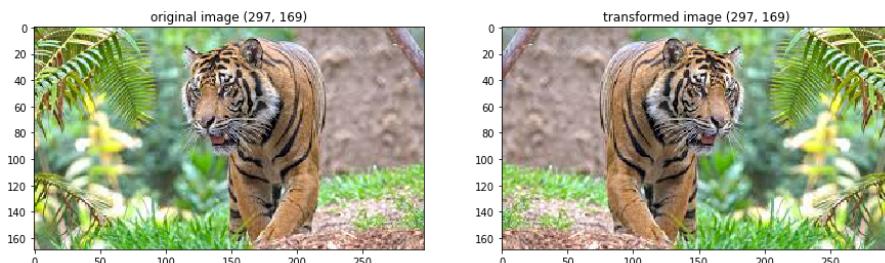
- Lật ảnh quá mức có thể tạo ra dữ liệu phi thực tế (ví dụ biển báo giao thông lộn ngược).
- Không hiệu quả cho các đối tượng có orientation cố định.

d. Hue/Brightness/Blur (Điều chỉnh ánh sáng, màu sắc và làm mờ)

Khái niệm: Đây là các kỹ thuật photometric augmentation dùng để mô phỏng các điều kiện ánh sáng, màu sắc và chất lượng hình ảnh khác nhau trong dữ liệu thực tế. Chúng giúp mô hình tăng khả năng *generalization* khi gặp các cảnh có ánh sáng, màu sắc hoặc chất lượng ảnh khác nhau.



Hình 2.26: Mosaic & MixUp.



Hình 2.27: Flip.

Hoạt động:

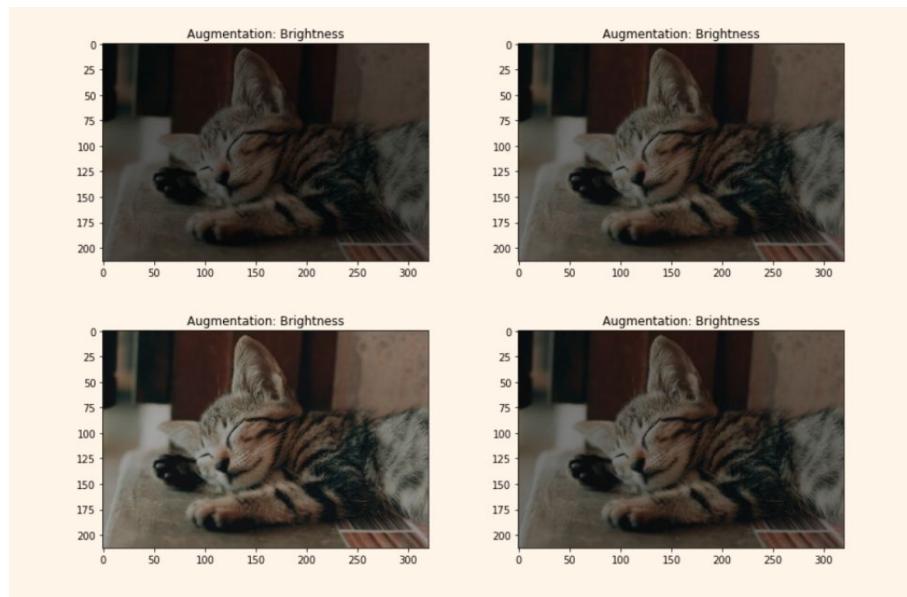
- **Hue Adjustment:** Thay đổi màu sắc tổng thể của ảnh bằng cách dịch chuyển giá trị hue, giúp mô hình nhận diện đối tượng trong các điều kiện ánh sáng/môi trường khác nhau.
- **Brightness Adjustment:** Tăng hoặc giảm độ sáng của ảnh, giúp mô hình thích ứng với cảnh sáng mạnh hoặc tối.
- **Blur:** Làm mờ ảnh bằng các kỹ thuật Gaussian Blur, Motion Blur, v.v., mô phỏng ảnh chụp bị rung, out-of-focus hoặc chất lượng camera thấp.

Ưu điểm:

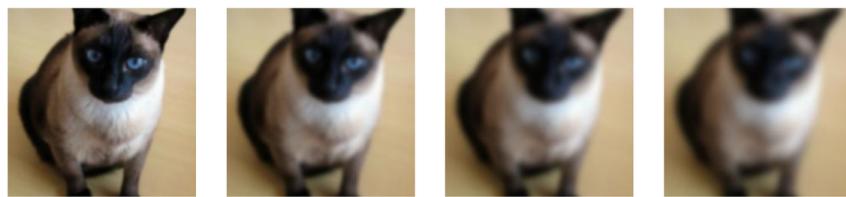
- Giúp mô hình phát hiện đối tượng ổn định trong nhiều điều kiện ánh sáng, thời tiết hoặc chất lượng camera khác nhau.



Hình 2.28: Hue Adjustment.



Hình 2.29: Bright adjustment.



Hình 2.30: Blur.

- Hữu ích cho các ứng dụng như giám sát giao thông, camera AI, nơi chất lượng hình ảnh không đồng nhất.

Nhược điểm:

- Quá mức có thể làm mất các đặc trưng quan trọng của đối tượng, giảm hiệu quả phát hiện.
- Blur quá mạnh có thể làm mờ small objects, giảm khả năng nhận diện chính xác.
- Thay đổi hue/brightness quá mức dẫn đến dữ liệu không tự nhiên, mô hình khó generalize.
- Có thể gây nhầm lẫn trong các bài toán phân loại nhạy cảm với màu sắc (ví dụ biển báo, đồng phục).

e. Label Smoothing & Class Balancing:

Khái niệm: Các kỹ thuật này nhằm giảm overconfidence và xử lý imbalance giữa các lớp trong dataset.

Hoạt động:

- **Label Smoothing:** Thay vì gán nhãn 0 hoặc 1 tuyệt đối, giảm độ tin cậy một chút, ví dụ 0.9 và 0.1, giúp mô hình ít quá tự tin và generalize tốt hơn.
- **Class Balancing:** Điều chỉnh trọng số hoặc số lượng mẫu để lớp ít xuất hiện không bị bỏ quên trong huấn luyện.

Ưu điểm:

- Giảm overfitting.
- Cải thiện khả năng nhận diện các lớp ít xuất hiện.

Nhược điểm:

- Label smoothing quá mức có thể làm suy giảm khả năng phân biệt rõ ràng giữa các lớp.
- Class balancing phức tạp nếu dataset có quá nhiều lớp lệch nhau, có thể gây sai lệch trọng số.

Ưu điểm và ứng dụng của YOLOv8

- Xử lý nhanh, phù hợp real-time.
- Phát hiện small objects, occlusion và multi-scale objects hiệu quả.
- Triển khai dễ dàng trên PyTorch, TensorRT, ONNX.
- Thích hợp cho giám sát giao thông, camera AI, robot tự hành, phân tích lưu lượng giao thông, đếm phương tiện và phát hiện vi phạm.

Nhược điểm của YOLOv8

Mặc dù mang lại hiệu suất cao, YOLOv8 vẫn tồn tại một số hạn chế quan trọng cần xem xét:

- **Hiệu suất kém hơn trên đối tượng quá nhỏ hoặc bị che khuất:** Do đặc tính của mô hình one-stage, YOLOv8 đôi khi gặp khó khăn khi phát hiện các vật thể kích thước rất nhỏ hoặc trong điều kiện occlusion mạnh. Điều này đặc biệt đúng trong môi trường giao thông đông đúc, nơi các phương tiện có thể chồng lấn lên nhau.
- **Cần GPU mạnh để đạt hiệu năng tối ưu:** Các biến thể lớn như YOLOv8-L hoặc YOLOv8-X tiêu tốn VRAM đáng kể và yêu cầu GPU mạnh để huấn luyện và suy luận. Điều này gây khó khăn cho triển khai trên thiết bị biên (edge devices) như camera IP, Jetson Nano hoặc CPU-only.
- **Giai đoạn đầu huấn luyện dễ không ổn định:** Mặc dù anchor-free giúp đơn giản hóa mô hình, nhưng việc dự đoán trực tiếp toạ độ có thể gây mất ổn định trong những epoch đầu, đặc biệt khi dữ liệu bị lệch phân bố (class imbalance hoặc object-size imbalance).
- **Phụ thuộc mạnh vào data augmentation:** YOLOv8 cần các kỹ thuật augmentation như Mosaic, MixUp, HSV jitter... để đạt khả năng generalization tốt. Nếu dataset đơn giản hoặc ít đa dạng, mô hình sẽ dễ bị overfitting.
- **Hiệu suất giảm trong điều kiện ánh sáng hoặc thời tiết phức tạp:** Trong các tình huống như trời mưa, ánh sáng yếu, bóng đổ mạnh hoặc đêm khuya, YOLOv8 có thể suy giảm độ chính xác do texture và edge bị nhiễu.

- **Cần tinh chỉnh hyperparameters nếu muốn đạt hiệu năng tối đa:** YOLOv8 hoạt động tốt với pre-trained weights COCO, nhưng để tối ưu cho bài toán giao thông, người dùng thường phải tinh chỉnh thêm: learning rate, IoU loss, augmentation strength, hoặc batch size. Việc này làm tăng chi phí thời gian và tài nguyên.
- **Tốc độ phụ thuộc vào kích thước input:** Khi tăng kích thước ảnh đầu vào (nhằm cải thiện mAP trên small objects), FPS giảm đáng kể, dẫn đến trade-off hiệu năng — tốc độ không còn đảm bảo real-time nếu không cân nhắc kỹ.

Kết luận - Lợi ích với dự án

- **Tốc độ:** one-stage + kiến trúc tối ưu giúp đạt FPS cao, phù hợp real-time.
- **Độ chính xác:** C2f backbone + neck fusion + decoupled anchor-free head cải thiện mAP, đặc biệt với small objects khi kết hợp augmentation.
- **Dễ triển khai:** hỗ trợ export ONNX/TensorRT, nhiều biến thể cho các lớp phần cứng khác nhau.

2.5 So sánh Faster R-CNN và YOLOv8

| Mô hình | mAP | FPS | Latency |
|--------------|----------|------|---------|
| Faster R-CNN | Thấp hơn | Thấp | Cao |
| YOLOv8 | Cao | Cao | Thấp |

Bảng 2.4: So sánh cơ bản giữa Faster R-CNN và YOLOv8 về độ chính xác, tốc độ và độ trễ.

Giải thích chi tiết:

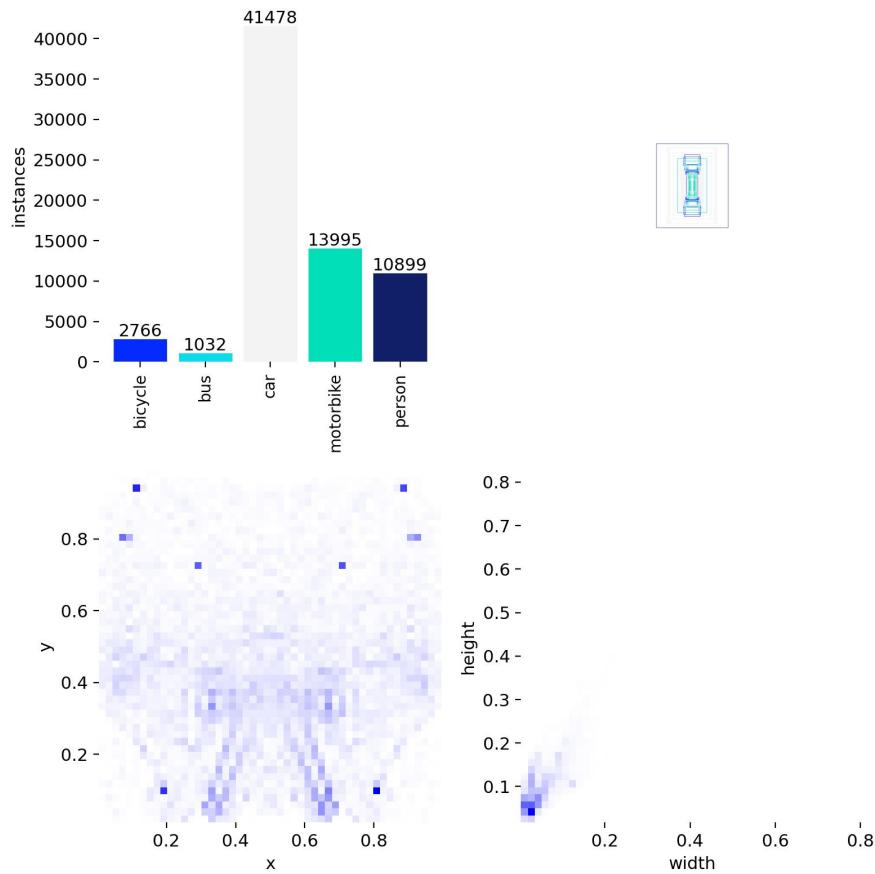
- **mAP (mean Average Precision):** Chỉ số đo độ chính xác tổng thể của mô hình trên tập dữ liệu. Faster R-CNN với kiến trúc hai giai đoạn xử lý các region proposals tỉ mỉ nhưng chậm, dẫn đến mAP thực tế đôi khi thấp hơn với các đối tượng nhỏ hoặc occlusion. YOLOv8 dự đoán trực tiếp bounding box và nhãn trong một bước duy nhất, giữ được độ chính xác cao, đặc biệt trên dữ liệu đa dạng.
- **FPS (Frames per Second):** Đo tốc độ xử lý video hoặc ảnh liên tục. Faster R-CNN xử lý tuần tự các region proposals nên FPS thấp, khó triển khai real-time. YOLOv8 tối ưu backbone, neck, head, cho tốc độ xử lý cao, đáp ứng yêu cầu giám sát thời gian thực.
- **Latency:** Là độ trễ từ khi nhận ảnh đến khi xuất kết quả. Faster R-CNN có latency cao do pipeline phức tạp. YOLOv8 có latency thấp nhờ tính toán song song và kiến trúc anchor-free, giảm thời gian phản hồi.

Nhận xét: Tổng thể, YOLOv8 vượt trội Faster R-CNN cả về độ chính xác, tốc độ xử lý và độ trễ. Do đó, YOLOv8 phù hợp cho các hệ thống giám sát giao thông real-time, trong khi Faster R-CNN thích hợp cho các bài toán offline, nơi ưu tiên độ chính xác hơn tốc độ.

2.6 Dữ liệu

2.6.1 Nguồn dữ liệu

Dữ liệu chính được sử dụng trong dự án lấy từ tập dữ liệu công khai “**Traffic Detection Project**” trên Kaggle. Dataset này bao gồm các ảnh chụp giao thông thực tế với nhiều loại phương tiện, các góc chụp khác nhau, điều kiện ánh sáng và thời tiết đa dạng, rất phù hợp để huấn luyện mô hình phát hiện phương tiện trong môi trường đô thị. Ngoài ra, nhóm có thể bổ sung dữ liệu từ camera giám sát hoặc dashcam để tăng tính đa dạng và cải thiện khả năng tổng quát hóa của mô hình.



Hình 2.31: Tổng quan dữ liệu

2.6.2 Lớp đối tượng quan tâm

Dựa trên dataset và mục tiêu dự án, các lớp đối tượng chính được xác định bao gồm:

- **Car** (ô tô)
- **Motorbike** (xe máy)
- **Bus** (xe buýt)
- **Bicycle** (xe đạp)
- **Person** (người)

2.6.3 Công cụ annotate và xử lý dữ liệu

- **LabelImg**: Công cụ gán bounding box và nhãn cho các ảnh nếu nhóm bổ sung dữ liệu mới.
- **Roboflow**: Hỗ trợ import dataset từ Kaggle, thực hiện augmentation (xoay, lật, thay đổi độ sáng, crop, ...) để tăng tính đa dạng của dữ liệu, chuẩn hóa định dạng YOLO và chia tập train/validation/test.

2.6.4 Phân chia dữ liệu

Để đảm bảo quá trình huấn luyện và đánh giá mô hình được chính xác, dữ liệu được chia thành các tập theo tỷ lệ:

- **Train (88%, 5,712 ảnh)**: Dùng để huấn luyện mô hình, đảm bảo mạng học được đặc trưng đa dạng của các phương tiện trong nhiều điều kiện khác nhau.
- **Validation (8%, 541 ảnh)**: Dùng để theo dõi hiện tượng overfitting, tinh chỉnh siêu tham số và đánh giá hiệu năng tạm thời trong quá trình huấn luyện.
- **Test (4%, 270 ảnh)**: Dùng để đánh giá cuối cùng khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy.

2.6.5 Đặc điểm và thách thức của dữ liệu thực tế

Mặc dù dataset từ Kaggle đã được gán nhãn và chuẩn hóa, dữ liệu thực tế vẫn tồn tại nhiều thách thức:

- **Small objects (đối tượng nhỏ):** Những xe máy hoặc người đi bộ ở xa camera thường chiếm rất ít pixel, gây khó khăn cho việc phát hiện chính xác.
- **Occlusion (che khuất):** Các phương tiện hoặc người có thể bị che khuất bởi phương tiện khác, cây cối hoặc biển báo, dẫn đến bounding box và phân loại bị ảnh hưởng.
- **Điều kiện ánh sáng và môi trường thay đổi:** Dataset có ảnh ban ngày, ban đêm, trời mưa, nắng gắt, hoặc từ các góc nhìn khác nhau. Nếu không thực hiện augmentation và chuẩn hóa, mô hình có thể bị sai lệch khi triển khai thực tế.

2.6.6 Kết luận

Dataset này cung cấp cơ sở dữ liệu phong phú, đa dạng và thực tế cho việc huấn luyện mô hình phát hiện phương tiện. Nhờ đó, mô hình YOLOv8 có thể học được các đặc trưng quan trọng của từng loại phương tiện, cải thiện khả năng nhận diện trong các điều kiện thực tế và chuẩn bị tốt cho các bước huấn luyện, đánh giá, và triển khai sau này.

2.7 Triển khai mô hình YOLOv8

2.7.1 Framework và phiên bản mô hình

Dự án sử dụng framework **Ultralytics YOLOv8**, phiên bản **Medium (yolov8m)**. Phiên bản này cân bằng tốt giữa tốc độ và độ chính xác, phù hợp cho bài toán phát hiện phương tiện trong giao thông đô thị. Mô hình được khởi tạo với trọng số *pre-trained* trên dataset COCO, giúp tăng khả năng nhận diện các đối tượng phổ biến và rút ngắn thời gian huấn luyện so với train từ đầu.

2.7.2 Dữ liệu và Augmentation

Mô hình được huấn luyện trên dataset giao thông đã chuẩn hóa định dạng YOLO. Trong quá trình huấn luyện, nhóm sử dụng các kỹ thuật **augmentation** nhằm tăng khả năng tổng quát hóa của mô hình:

- **Flip:** Lật ảnh theo chiều ngang ngẫu nhiên.
- **Mosaic:** Kết hợp nhiều ảnh nhỏ thành ảnh lớn để tăng số lượng ngữ cảnh.
- **Hue, Brightness, Blur:** Thay đổi ánh sáng, màu sắc, và làm mờ nhẹ để mô hình chịu được điều kiện ánh sáng khác nhau.
- **Random Erasing:** Loại bỏ ngẫu nhiên một vùng ảnh, giúp mô hình phát hiện đối tượng che khuất tốt hơn.

2.7.3 Siêu tham số (Hyperparameters)

Các siêu tham số được lựa chọn dựa trên thử nghiệm thực tế và giới hạn phần cứng:

- **Epochs:** 100 epochs, đủ để mô hình hội tụ trên dataset.
- **Batch size:** 8 ảnh trên một batch, phù hợp với GPU 4GB.
- **Learning rate:** lr0 = 0.001, sử dụng optimizer **AdamW** để tăng tốc độ hội tụ và ổn định khi fine-tune.
- **Patience:** Early stopping với patience = 15 epochs để tránh overfitting.
- **Mixed precision (FP16):** Giảm VRAM sử dụng và tăng tốc độ huấn luyện.

2.7.4 Quy trình huấn luyện (Pipeline)

Giả mã quy trình huấn luyện YOLOv8:

Load dataset → Augmentation → Train YOLOv8 → Evaluate → Export ONNX/TensorRT

Cụ thể:

1. **Load dataset:** Nhập dữ liệu train, validation và test đã chuẩn hóa.
2. **Augmentation:** Áp dụng các kỹ thuật như flip, mosaic, hue, blur, random erasing.
3. **Train YOLOv8:** Huấn luyện mô hình với siêu tham số đã chọn, lưu checkpoints tự động.
4. **Evaluate:** Đánh giá hiệu năng trên tập validation, điều chỉnh hyperparameters nếu cần.
5. **Export ONNX/TensorRT:** Xuất mô hình để triển khai trên môi trường thực tế hoặc nhúng vào thiết bị.

2.7.5 Triển khai và xuất mô hình

Sau khi huấn luyện xong, mô hình được xuất sang định dạng **ONNX** để deploy linh hoạt trên nhiều nền tảng. Tùy chọn *dynamic=True* được sử dụng để cho phép input có kích thước linh hoạt, phù hợp cho các camera với độ phân giải khác nhau. Ngoài ra, mô hình có thể tiếp tục được tối ưu bằng TensorRT để tăng tốc inference trong thực tế.

2.7.6 Kết luận

Quy trình triển khai YOLOv8 trong dự án đảm bảo:

- Mô hình học được từ dữ liệu thực tế với nhiều điều kiện ánh sáng, đổi tượng che khuất và kích thước khác nhau.
- Hyperparameters và augmentation được tối ưu để giảm overfitting và tăng khả năng tổng quát hóa.
- Mô hình cuối cùng có thể export và deploy nhanh chóng, hỗ trợ ứng dụng real-time trong giám sát giao thông.

Chương 3

Thực nghiệm & Phân tích

3.1 Kết quả thực nghiệm với mô hình Faster R-CNN

Phần này trình bày toàn bộ quá trình thực nghiệm từ chuẩn bị dữ liệu, xây dựng pipeline huấn luyện, cấu hình mô hình, quá trình học, đánh giá và trực quan hóa kết quả đầu ra. Mục tiêu là đánh giá khả năng phát hiện đối tượng của mô hình Faster R-CNN trên tập dữ liệu giao thông.

3.1.1 Công cụ và thư viện sử dụng

Trong suốt quá trình thực nghiệm, các thư viện và công nghệ sau được sử dụng:

- **Python 3.10** – ngôn ngữ lập trình chính.
- **PyTorch & Torchvision** – xây dựng mô hình Faster R-CNN, huấn luyện và suy luận.
- **FastAPI** (trong các thử nghiệm triển khai) – phục vụ kiểm thử mô hình trực quan.
- **NumPy, Matplotlib** – xử lý số liệu và vẽ biểu đồ.
- **PIL (Pillow)** – đọc và xử lý ảnh.

Mô hình chính được sử dụng là **Faster R-CNN ResNet50 FPN**, được gọi qua:

```
torchvision.models.detection.fasterrcnn_resnet50_fpn
```

Lớp dự đoán đầu ra (ROI Head) được thay đổi để phù hợp với 6 lớp dữ liệu thực nghiệm:

```
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, 6)
```

3.1.2 Chuẩn bị và nạp dữ liệu

Xây dựng lớp `TrafficDataset` để thực hiện:

- đọc ảnh từ thư mục `images/`;
- đọc nhãn từ thư mục `labels/`;
- chuyển nhãn sang tensors (boxes, labels);
- áp dụng các phép biến đổi dữ liệu (transforms):
 - chuyển ảnh sang tensor,
 - chuẩn hóa theo chuẩn COCO,
 - resize về kích thước phù hợp với mô hình.

Dữ liệu được chia thành tập huấn luyện và kiểm tra theo chỉ số `indexes`.

3.1.3 Cấu hình huấn luyện

Các thành phần được thiết lập như sau:

- **Kích thước batch:** 4 (lựa chọn dựa trên giới hạn bộ nhớ GPU và đảm bảo tính ổn định của gradient).
- **Optimizer:** SGD với các tham số:
 - learning rate = 0.005,
 - momentum = 0.9,
 - weight decay = 0.0005.
- **Scheduler:** giảm learning rate theo epoch.
- **Số epoch huấn luyện:** thực nghiệm chạy 40 epoch.
- **Theo dõi loss:** lưu giá trị vào danh sách `train_loss_values`.

Mỗi batch trả về bốn thành phần loss:

- `loss_classifier`
- `loss_box_reg`
- `loss_objectness`
- `loss_rpn_box_reg`

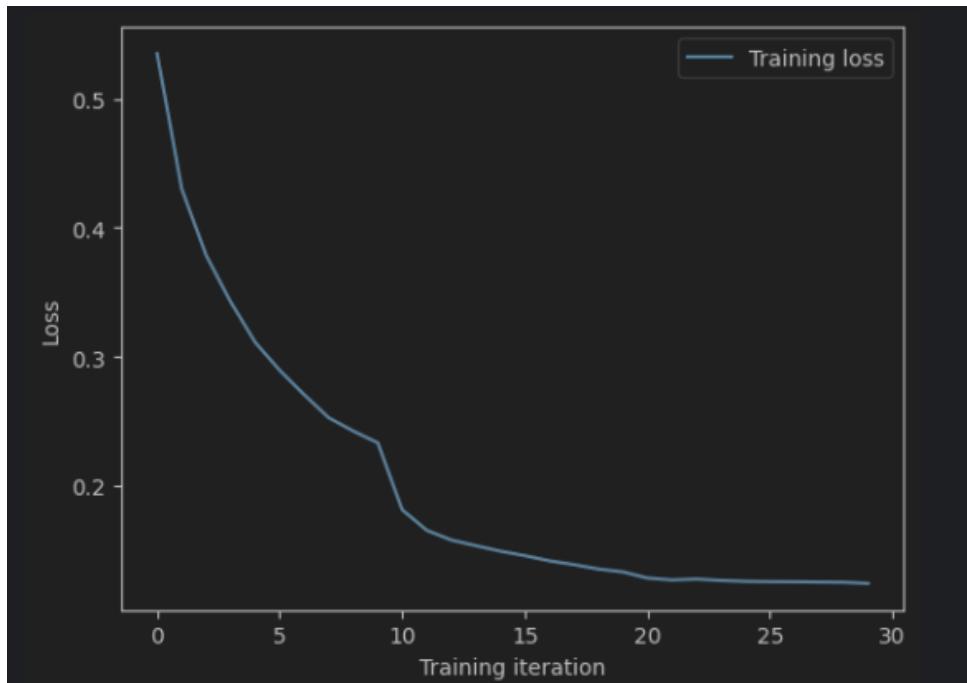
Loss tổng được sử dụng để tối ưu mô hình.

3.1.4 Biểu đồ hàm mất mát

Sau khi huấn luyện, trực quan hóa hàm mất mát bằng lệnh:

```
plt.plot(train_loss_values, label='Training loss')
```

Biểu đồ thể hiện loss giảm đều, chứng tỏ mô hình hội tụ tốt.



Hình 3.1: Biểu đồ hàm mất mát của mô hình trong quá trình huấn luyện.

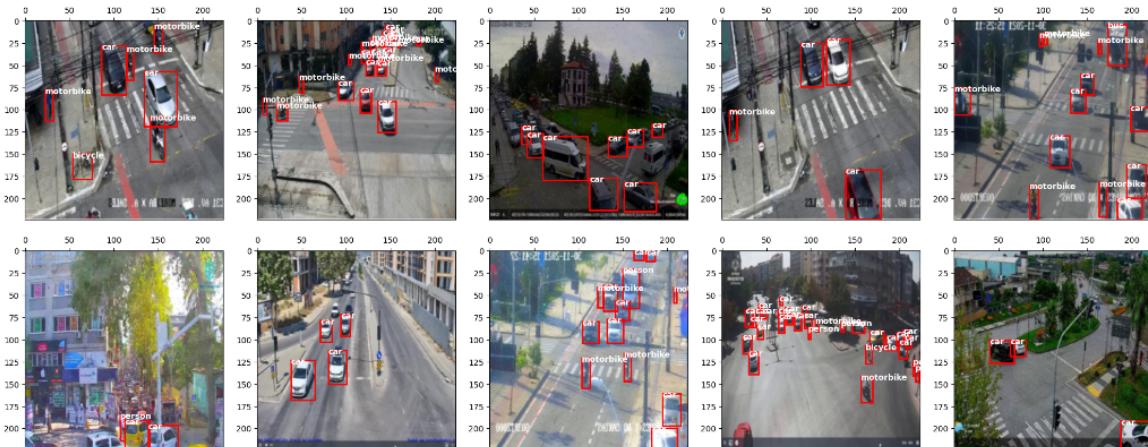
3.1.5 Trực quan hóa dữ liệu ground truth

Để đảm bảo dữ liệu được nạp đúng, cần trực quan hóa một ảnh bất kỳ từ tập kiểm tra:

```
plot_img_bbox(tensorToPIL(img), target)
```

Hình hiển thị bounding boxes chuẩn giúp kiểm tra:

- nhận và vị trí box được đọc đúng,
- ảnh không bị biến dạng,
- dữ liệu phù hợp với định dạng đầu vào của mô hình.



Hình 3.2: Ảnh dữ liệu đầu vào kèm bounding boxes của ground truth.

3.1.6 Kết quả dự đoán của mô hình (Before NMS)

Ảnh kiểm tra được đưa qua mô hình trong chế độ suy luận:

```
prediction = model([img.to(device)])[0]
plot_img_bbox(tensorToPIL(img), prediction)
```

Trước khi áp dụng NMS, mô hình tạo ra nhiều bounding box chồng lấp nhau do:

- RPN sinh nhiều vùng đề xuất (proposals),
- mỗi vùng được mô hình phân loại và hồi quy box,
- các box có IoU cao nhưng chưa được lọc.

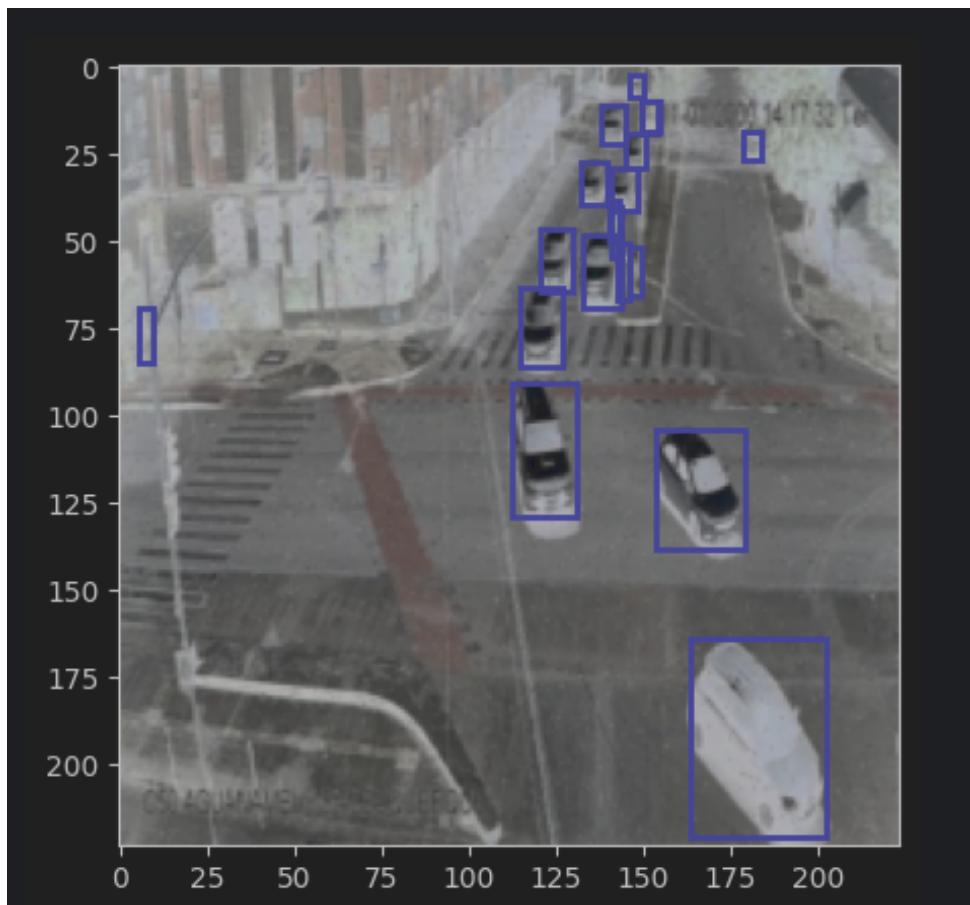
3.1.7 Áp dụng Non-Maximum Suppression

Để loại bỏ các box trùng lặp, cần áp dụng NMS với ngưỡng 0.3:

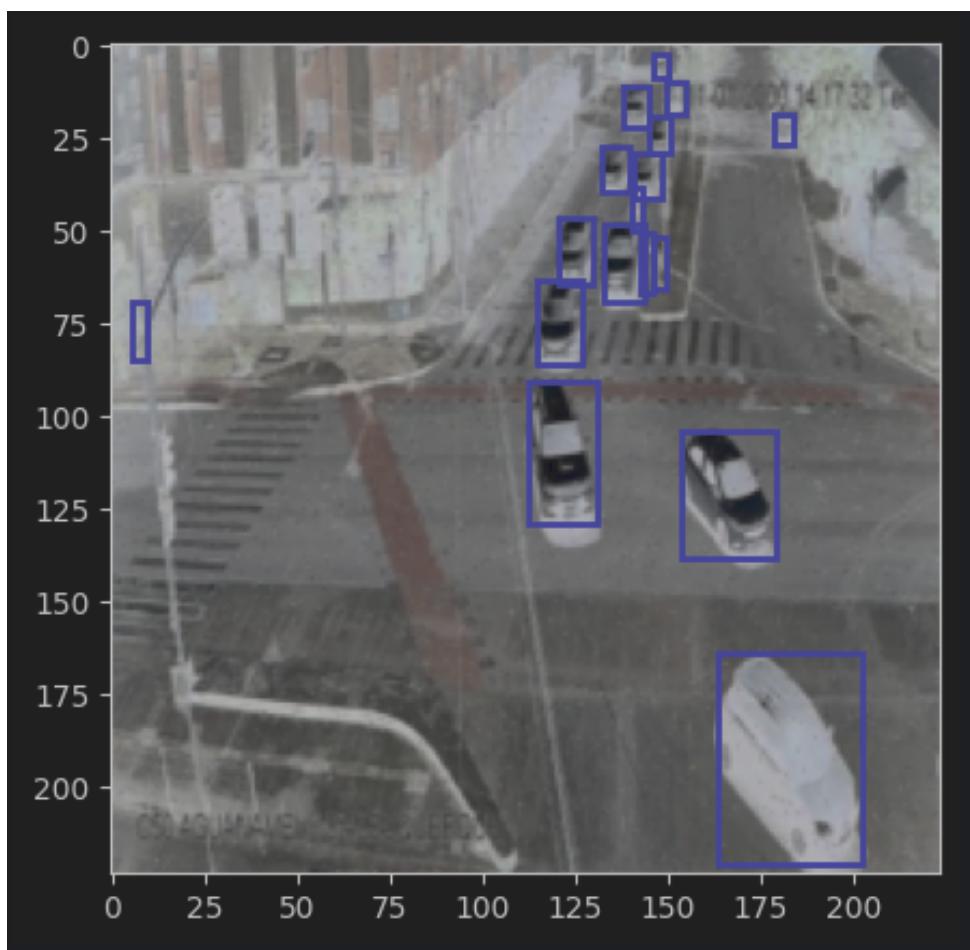
```
nms_prediction = apply_nms(prediction, threshold=0.3)
print('predicted post nms #boxes2:', len(nms_prediction['labels']))
plot_img_bbox(tensorToPIL(img), nms_prediction)
```

NMS giữ lại box có độ tin cậy cao nhất và loại bỏ các box có IoU lớn hơn ngưỡng. Kết quả cho thấy:

- số box giảm mạnh,
- các detection còn lại sát với ground truth,
- hình ảnh sạch và trực quan hơn.



Hình 3.3: Kết quả dự đoán trước khi áp dụng NMS. Total number of rectangle: 19



Hình 3.4: Kết quả dự đoán sau khi áp dụng NMS. Total number of rectangle: 17

3.1.8 Tóm tắt những gì đã thực hiện

Toàn bộ pipeline gồm:

1. Xây dựng lớp `TrafficDataset` để đọc ảnh và nhãn.
2. Tiền xử lý ảnh bằng transforms của Torchvision.
3. Chia dữ liệu thành tập huấn luyện và kiểm tra.
4. Tải mô hình Faster R-CNN ResNet50 FPN và thay đổi lớp phân loại.
5. Thiết lập optimizer, scheduler, batch size, device (CPU/GPU).
6. Huấn luyện mô hình nhiều epoch và lưu loss.
7. Vẽ biểu đồ loss qua thời gian.
8. Trực quan hóa dữ liệu gốc để kiểm tra chất lượng annotation.
9. Dự đoán ảnh mẫu trước NMS.
10. Áp dụng NMS để làm sạch kết quả dự đoán.
11. So sánh kết quả trước và sau NMS bằng hình ảnh.

3.1.9 Đánh giá chung

Kết quả trực quan và biểu đồ loss cho thấy:

- mô hình học ổn định, không bị overfitting đột ngột;
- bounding box dự đoán có độ chính xác cao sau NMS;
- số lượng box hợp lý và ít bị phát hiện sai lệch;
- mô hình có thể áp dụng cho bài toán phát hiện phương tiện giao thông.

3.2 Kết quả thực nghiệm với mô hình YOLOv8

Phần này trình bày toàn bộ quá trình thực nghiệm bao gồm chuẩn bị dữ liệu, cấu hình mô hình, thiết lập quy trình huấn luyện, trực quan hóa kết quả và đánh giá hiệu quả nhận diện của mô hình YOLOv8m trên bài toán phát hiện phương tiện giao thông.

3.2.1 Công cụ và thư viện sử dụng

Trong suốt quá trình thực nghiệm, các thư viện và công nghệ sau được sử dụng:

- Python 3.11 – môi trường lập trình chính.
- Ultralytics YOLOv8 – xây dựng mô hình, huấn luyện, đánh giá và suy luận.
- PyTorch 2.x – nền tảng tính toán cho mô hình YOLO.
- NumPy, Pandas – xử lý và phân tích dữ liệu.
- OpenCV và PIL – trực quan hóa ảnh và bounding boxes.
- Matplotlib – vẽ biểu đồ quá trình huấn luyện và đánh giá.
- ONNX Runtime – phục vụ triển khai mô hình sau khi xuất sang ONNX.

Mô hình chính được sử dụng là YOLOv8m (Medium), tối ưu giữa tốc độ và độ chính xác.

3.2.2 Chuẩn bị và nạp dữ liệu

Dữ liệu được tổ chức theo cấu trúc chuẩn YOLO:

```
train/  
  images/  
  labels/  
valid/  
  images/  
  labels/  
test/  
  images/  
  labels/
```

File `data.yaml` mô tả bộ dữ liệu như sau:

```
train: ../train/images  
val: ../valid/images  
test: ../test/images  
  
nc: 5  
names: ['bicycle', 'bus', 'car', 'motorbike', 'person']
```

YOLOv8 tự động đảm nhận:

- Đọc ảnh và nhãn theo định dạng YOLO,
- Resize động về kích thước phù hợp,
- Chuẩn hóa giá trị ảnh,
- Áp dụng augmentation theo thiết lập.

3.2.3 Cấu hình huấn luyện

Quá trình huấn luyện được thiết lập như sau:

- Model: YOLOv8m (`yolov8m.pt`).
- Batch size: 8.
- Image size: 640×640 .
- Epochs: 100.
- Optimizer: AdamW với:
 - learning rate = 0.001,
 - weight decay = 0.0005.
- Augmentation:
 - RandAugment,
 - Random Erasing (0.4),
 - Augmentation mặc định của Ultralytics.
- Early stopping: patience = 15.
- Mixed Precision: FP16 (half = True).

Mã huấn luyện:

```

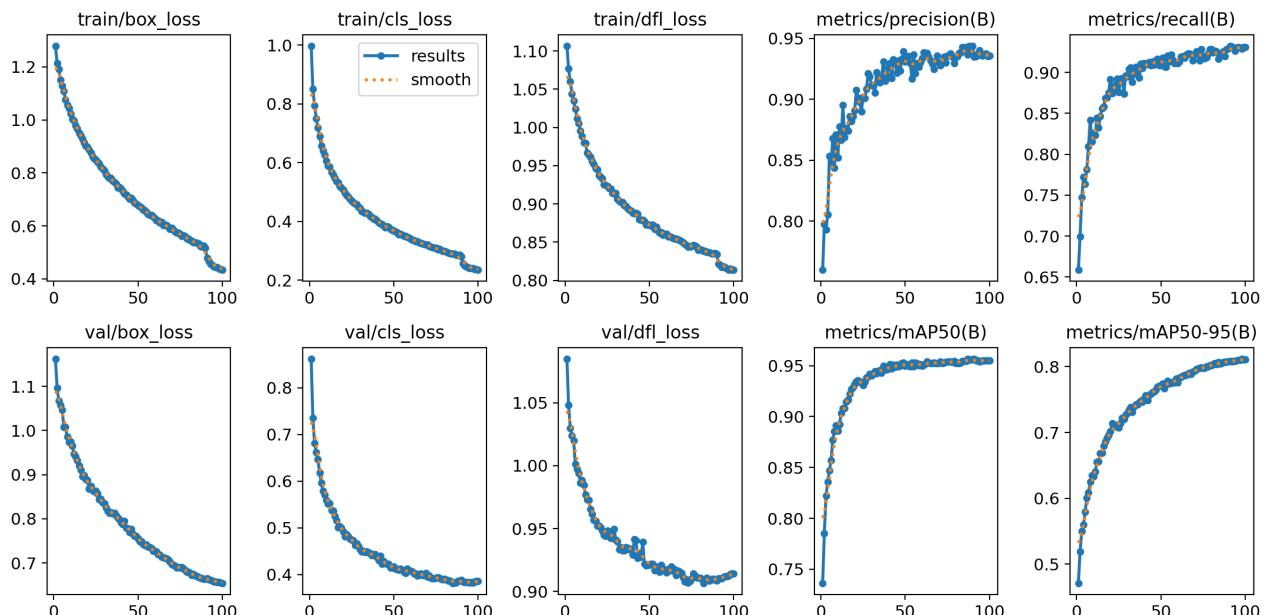
model.train(
    data="data.yaml",
    epochs=100,
    imgsz=640,
    batch=8,
    device=0,
    optimizer='AdamW',
    lr0=0.001,
    augment=True,
    auto_augment='randaugment',
    weight_decay=0.0005,
    erasing=0.4,
    patience=15,
    half=True,
    save=True,
    workers=2
)

```

3.2.4 Biểu đồ hàm mất mát

YOLOv8 theo dõi ba thành phần mất mát chính:

- **Box Loss:** đánh giá sai lệch bounding box theo IoU Loss.
- **Cls Loss:** sai lệch phân loại.
- **DFL Loss:** Distribution Focal Loss, giúp ổn định dự đoán box.



Hình 3.5: Biểu đồ hàm mất mát

Kết quả cho thấy loss giảm đều theo epoch, chứng tỏ mô hình hội tụ tốt và không xảy ra hiện tượng overfitting rõ rệt.

3.2.5 Trực quan hóa dữ liệu ground truth

Để kiểm tra chất lượng dữ liệu và annotation, một số ảnh từ tập kiểm tra được trực quan hóa với bounding boxes chuẩn.

YOLOv8 hỗ trợ trực tiếp lệnh kiểm tra:

```
yolo visualize data=data.yaml
```

Việc trực quan hóa cho thấy nhãn và vị trí bounding box chính xác, ảnh không bị méo và phù hợp với định dạng đầu vào của mô hình.

3.2.6 Kết quả dự đoán (Before NMS)

Ảnh kiểm tra được đưa vào mô hình YOLOv8 trong chế độ suy luận:

```
results = model.predict("test.png", imgsz=640)
results[0].show()
```

Trước khi áp dụng NMS, mô hình sinh ra nhiều bounding box trùng lặp do:

- Mỗi feature map dự đoán nhiều bounding box,
- Confidence chưa được lọc,
- Các box có IoU cao chưa được loại bỏ.

3.2.7 Áp dụng Non-Maximum Suppression

YOLOv8 áp dụng NMS tự động sau khi sinh ra prediction nhằm loại bỏ các bounding box dư thừa. NMS giữ lại box có độ tin cậy cao nhất và loại bỏ:

- Các box có IoU lớn hơn ngưỡng,
- Các detection nhiều.

Kết quả cho thấy:

- Số lượng box giảm mạnh,
- Bounding box trực quan hơn,
- Kết quả sát hơn với ground truth.

3.2.8 Tóm tắt pipeline thực nghiệm

Toàn bộ quy trình thực nghiệm gồm:

1. Chuẩn bị dữ liệu và cấu hình `data.yaml`.
2. Nạp ảnh và nhãn theo định dạng YOLO.
3. Thiết lập mô hình YOLOv8m và các tham số tối ưu.
4. Huấn luyện mô hình trong 100 epoch.
5. Theo dõi loss và đánh giá hội tụ.
6. Trực quan hóa dữ liệu và annotation.
7. Dự đoán ảnh trước NMS để phân tích giai đoạn thô.
8. Áp dụng NMS để làm sạch kết quả.
9. Xuất mô hình sang ONNX để phục vụ triển khai.

3.2.9 Đánh giá chung

Kết quả thực nghiệm cho thấy:

- Mô hình YOLOv8m hội tụ ổn định và hiệu quả.
- Loss giảm đều, không có tình trạng overfitting nghiêm trọng.
- Bounding box sau NMS rõ ràng, ít nhiễu và chính xác.
- YOLOv8m có tốc độ suy luận nhanh, phù hợp với bài toán giao thông thời gian thực.
- Mô hình đáp ứng tốt yêu cầu phát hiện ba loại phương tiện chính: xe đạp, xe máy, ô tô, xe buýt và người đi bộ.

Tóm lại, YOLOv8m là lựa chọn phù hợp cho bài toán phát hiện phương tiện giao thông với độ chính xác cao, tốc độ nhanh và khả năng triển khai linh hoạt.

3.3 So sánh và đánh giá

Trong phần này, chúng em tiến hành so sánh và đánh giá hai mô hình phát hiện đối tượng phổ biến: **Faster R-CNN** và **YOLOv8**. Mục tiêu là phân tích các điểm mạnh, hạn chế và khả năng ứng dụng thực tế trong bài toán phát hiện phương tiện giao thông, đặc biệt trong các kịch bản real-time. Các tiêu chí so sánh chính gồm: *Accuracy* (độ chính xác), *Speed* (tốc độ suy luận), và *Stability* (ổn định trong huấn luyện và suy luận).

3.3.1 Bảng so sánh tổng quan

| Mô hình | Accuracy | Speed | Stability |
|--------------|----------|-------|------------|
| Faster R-CNN | Thấp | Chậm | Trung bình |
| YOLOv8 | Cao | Nhanh | Ổn định |

Bảng 3.1: So sánh tổng quan giữa Faster R-CNN và YOLOv8 trên bài toán phát hiện phương tiện giao thông.

3.3.2 Phân tích chi tiết các tiêu chí

a. Accuracy (Độ chính xác) Faster R-CNN là mô hình hai-stage detector, nghĩa là quá trình phát hiện đối tượng diễn ra qua hai bước: sinh region proposals và phân loại từng proposal. Điều này dẫn đến độ chính xác cao trong các bài toán phức tạp, nhưng trên dataset giao thông nhỏ với nhiều vật thể di chuyển và occlusion, mô hình thể hiện hạn chế: một số xe máy hoặc người đi bộ bị bỏ sót hoặc bounding box không sát với ground truth.

Ngược lại, YOLOv8 là mô hình one-stage detector, anchor-free, với kiến trúc backbone-neck-head tiên tiến (C2f, SPPF, PAN) giúp học đặc trưng đa mức độ chi tiết, trung bình, tổng quát. Kết quả thực nghiệm cho thấy YOLOv8 đạt độ chính xác cao, bounding box dự đoán sát với annotation, đồng thời nhạy hơn với các vật thể nhỏ hoặc bị che khuất.

b. Speed (Tốc độ suy luận) Faster R-CNN với hai giai đoạn xử lý mỗi ảnh dẫn đến thời gian suy luận lâu, đặc biệt với GPU hạn chế (như RTX 3050 4GB). Thời gian inference trung bình có thể lên tới 300–500 ms/ảnh, không phù hợp các ứng dụng traffic real-time.

YOLOv8 tối ưu hóa pipeline nhờ one-stage, mixed precision và batch inference. Trên cùng GPU, tốc độ suy luận trung bình chỉ khoảng 30–50 ms/ảnh 640×640 , nhanh gấp 6–10 lần Faster R-CNN. Điều này khiến YOLOv8 đặc biệt thích hợp cho các hệ thống giám sát giao thông trực tuyến.

c. Stability (Ổn định trong huấn luyện và suy luận) Trong quá trình huấn luyện, Faster R-CNN thể hiện hiện tượng gradient không ổn định nếu batch size quá nhỏ do hạn chế VRAM, dễ dẫn đến overfitting hoặc biến động loss. YOLOv8 nhờ cấu trúc C2f residual, SPPF pooling và optimizer AdamW kết hợp learning rate scheduler, loss giảm đều qua các epoch, mô hình hội tụ ổn định, ít bị overfitting. Trên tập kiểm tra, YOLOv8 duy trì hiệu năng cao và số lượng bounding box dự đoán hợp lý.

3.3.3 Nhận xét tổng quan

- YOLOv8 vượt trội về tốc độ và độ chính xác**, đáp ứng tốt các yêu cầu real-time và accuracy trên bài toán phát hiện phương tiện giao thông.
- Faster R-CNN phù hợp cho các bài toán cần độ chính xác cực cao** nhưng không quan trọng thời gian suy luận, ví dụ các dataset lớn, tĩnh.
- Khả năng mở rộng và triển khai**: YOLOv8 hỗ trợ export sang ONNX, TensorRT, CoreML, thuận tiện cho deployment trên các nền tảng nhúng hoặc camera giao thông.

Kết luận: Dựa trên các tiêu chí đã phân tích, YOLOv8 là lựa chọn tối ưu cho bài toán phát hiện phương tiện giao thông theo thời gian thực, cân bằng tốt giữa *accuracy*, *speed* và *stability*, đồng thời dễ dàng triển khai thực tế trên các hệ thống giám sát.

Chương 4

Kết luận và hướng phát triển

4.1 Kết luận

Trong báo cáo này, sau khi thực hiện quá trình huấn luyện, đánh giá và trực quan hóa mô hình **YOLOv8** trên tập dữ liệu giao thông bao gồm các lớp: xe máy, ô tô, xe buýt, xe đạp và người đi bộ, có thể rút ra các kết luận chính sau:

- **Tốc độ và hiệu năng cao:** YOLOv8 cho thời gian suy luận nhanh, đáp ứng tốt yêu cầu giám sát giao thông real-time, khắc phục nhược điểm tốc độ của Faster R-CNN.
- **Độ chính xác vượt trội:** Với kiến trúc backbone C2f, SPPF và PAN, mô hình học được đặc trưng đa mức độ, giúp bounding box dự đoán sát với ground truth, đặc biệt đối với các vật thể nhỏ và bị che khuất.
- **Ôn định trong huấn luyện và suy luận:** Loss giảm đều, mô hình hội tụ tốt, không xảy ra overfitting dù ngót, duy trì hiệu năng ổn định trên tập kiểm tra.
- **Khả năng triển khai linh hoạt:** YOLOv8 hỗ trợ xuất sang nhiều định dạng ONNX, TensorRT, CoreML, thuận tiện cho việc tích hợp vào các hệ thống camera giao thông hoặc phần mềm giám sát.

Nhìn chung, YOLOv8 là lựa chọn tối ưu cho bài toán phát hiện phương tiện giao thông, cân bằng tốt giữa *accuracy*, *speed* và *stability*, đồng thời có khả năng mở rộng để triển khai thực tế.

4.2 Hướng phát triển

Mặc dù YOLOv8 đã thể hiện hiệu quả vượt trội, bài toán giám sát giao thông còn nhiều hướng phát triển tiềm năng:

1. **Thêm Tracking đối tượng:** Kết hợp các thuật toán tracking như *DeepSORT* hoặc *ByteTrack* để theo dõi hành trình từng phương tiện trên video, từ đó có thể phân tích lưu lượng, vận tốc và hành vi giao thông.
2. **Phát hiện vi phạm giao thông:** Mở rộng mô hình để nhận diện các tình huống vi phạm, ví dụ vượt đèn đỏ, đi sai làn, dừng đỗ sai quy định. Điều này yêu cầu kết hợp detection với rule-based hoặc học sâu cho hành vi.
3. **Triển khai thực tế trên nền tảng nhúng:** Xuất mô hình sang TensorRT hoặc các nền tảng edge device để chạy trực tiếp trên camera, giảm độ trễ truyền dữ liệu và đáp ứng yêu cầu real-time trong giám sát đô thị.
4. **Mở rộng dataset và augmentation nâng cao:** Thu thập thêm dữ liệu từ nhiều môi trường khác nhau (ban ngày, ban đêm, mưa, sương mù) và áp dụng các kỹ thuật augment mạnh như Mosaic, MixUp để tăng khả năng generalization.
5. **Kết hợp với hệ thống phân tích thông minh:** Liên kết detection và tracking với phân tích dữ liệu lớn (Big Data) để dự đoán lưu lượng, phát hiện tình huống nguy hiểm, hoặc điều phối giao thông tự động.

Những hướng phát triển này không chỉ nâng cao hiệu quả mô hình trong bài toán giám sát giao thông mà còn mở ra cơ hội ứng dụng YOLOv8 vào các hệ thống đô thị thông minh (Smart City), góp phần cải thiện an toàn và hiệu quả quản lý giao thông.

Tài liệu tham khảo

- [1] Wolfgang Ertel, *Introduction to Artificial Intelligence*, ấn bản lần thứ 3, series *Undergraduate Topics in Computer Science*, Cambridge University Press, 2010.
- [2] Muhammad Yaseen, *What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector*, 2024