# **TRƯỜNG ĐẠI HỌC SƯ PHẠM ĐÀ NẴNG**

# **KHOA TOÁN-TIN**



# **BÁO CÁO CHUYÊN NGÀNH**

**Đề tài: Nghiên cứu và xây dựng hệ thống nhận diện**

**biển số xe sử dụng mạng nơ-ron tích chập YOLOv4-tiny**

**Giáo viên hướng dẫn : Phạm Anh Phương**

**Sinh viên thực hiện : Đinh Xuân Hiệp**

**Lớp : 23CNTT1**

*Ngày… tháng… năm 2025*

**Nhận xét của giáo viên**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**MỤC LỤC**

[LỜI MỞ ĐẦU 1](#_Toc215494836)

[CHƯƠNG 1. TỔNG QUAN 2](#_Toc215494837)

[1.1. Đặt vấn đề và tính cấp thiết của đề tài 2](#_Toc215494838)

[1.1.1. Bối cảnh giao thông thông minh tại Việt Nam 2](#_Toc215494839)

[1.1.2. Nhu cầu nhận diện biển số tự động trên thiết bị biên 2](#_Toc215494840)

[1.2. Các phương pháp nhận diện biển số hiện nay 3](#_Toc215494841)

[1.2.1. Phương pháp xử lý ảnh truyền thống (Haar Cascade, SVM) 3](#_Toc215494842)

[1.2.2. Phương pháp học sâu (Deep Learning) 3](#_Toc215494843)

[1.3. Lý do lựa chọn mô hình YOLOv4-tiny 4](#_Toc215494844)

[1.4. Mục tiêu và phạm vi nghiên cứu 4](#_Toc215494845)

[1.5. Bố cục báo cáo 5](#_Toc215494846)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 6](#_Toc215494847)

[2.1. Tổng quan về Mạng nơ-ron tích chập (CNN) 6](#_Toc215494848)

[2.1.1. Cấu trúc cơ bản của mạng CNN 6](#_Toc215494849)

[2.1.2. Các lớp thành phần (Convolution, Pooling, FC) 6](#_Toc215494850)

[2.1.3. Các hàm kích hoạt (Activation Functions: ReLU, Leaky ReLU) 7](#_Toc215494851)

[2.2. Bài toán phát hiện đối tượng (Object Detection) 8](#_Toc215494852)

[2.2.1. One-stage vs Two-stage Detectors 8](#_Toc215494853)

[2.2.2. Các chỉ số đánh giá mô hình (IoU, Precision, Recall, F1-Score) 9](#_Toc215494854)

[2.2.3. Độ chính xác trung bình (mAP) và tốc độ (FPS) 10](#_Toc215494855)

[2.3. Sự phát triển của họ mô hình YOLO 10](#_Toc215494856)

[2.3.1. YOLOv1, YOLOv2 và YOLOv3 10](#_Toc215494857)

[2.3.2. Những cải tiến đột phá trong YOLOv4 11](#_Toc215494858)

[2.4. Kiến trúc mạng YOLOv4-tiny 11](#_Toc215494859)

[2.4.1. So sánh kiến trúc YOLOv4 và YOLOv4-tiny 11](#_Toc215494860)

[2.4.2. Khối CSP (Cross Stage Partial) trong Backbone 12](#_Toc215494861)

[2.4.3. Cấu trúc FPN (Feature Pyramid Network) rút gọn 13](#_Toc215494862)

[2.4.4. Hàm mất mát CIoU Loss (Complete IoU) 14](#_Toc215494863)

[2.4.5. Kỹ thuật tăng cường dữ liệu Mosaic 15](#_Toc215494864)

[CHƯƠNG 3. XÂY DỰNG HỆ THỐNG VÀ HUẤN LUYỆN MÔ HÌNH 16](#_Toc215494865)

[3.1. Quy trình thực hiện tổng quát 16](#_Toc215494866)

[3.2. Thu thập và xử lý dữ liệu (Dataset) 17](#_Toc215494867)

[3.2.1. Nguồn dữ liệu và đặc điểm biển số xe Việt Nam 17](#_Toc215494868)

[3.2.2. Gán nhãn dữ liệu với công cụ LabelImg 17](#_Toc215494869)

[3.2.3. Phân chia tập dữ liệu (Train/Val/Test) 18](#_Toc215494870)

[3.3. Môi trường và công cụ cài đặt 18](#_Toc215494871)

[3.3.1. Google Colab và Framework Darknet 18](#_Toc215494872)

[3.4. Cấu hình tham số huấn luyện (Training Configuration) 19](#_Toc215494873)

[3.4.1. Tinh chỉnh file cấu hình (.cfg) cho YOLOv4-tiny 19](#_Toc215494874)

[3.4.2. Tính toán Anchor Boxes bằng thuật toán K-means 19](#_Toc215494875)

[3.4.3. (Hyperparameters: LR, Batch size, Epochs) 19](#_Toc215494876)

[3.5. Quá trình huấn luyện mô hình 20](#_Toc215494877)

[CHƯƠNG 4. KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ 21](#_Toc215494878)

[4.1. Đánh giá quá trình huấn luyện 21](#_Toc215494879)

[4.1.1. Biểu đồ hàm mất mát (Loss Function) 21](#_Toc215494880)

[4.1.2. Sự thay đổi của chỉ số mAP theo thời gian 22](#_Toc215494881)

[4.2. Kết quả định lượng 22](#_Toc215494882)

[4.2.1. Các chỉ số đánh giá chi tiết 22](#_Toc215494883)

[4.2.2. So sánh hiệu năng 22](#_Toc215494884)

[4.3. Kết quả định tính (Demo hình ảnh) 23](#_Toc215494885)

[4.3.1. Nhận diện trong điều kiện thuận lợi 23](#_Toc215494886)

[4.3.2. Nhận diện trong điều kiện khó 23](#_Toc215494887)

[4.4. Phân tích lỗi và hạn chế 23](#_Toc215494888)

[4.5. Đánh giá tốc độ xử lý (FPS) 23](#_Toc215494889)

[4.6. Xây dựng ứng dụng Demo nhận diện biển số 24](#_Toc215494890)

[4.6.1. Công nghệ sử dụng 24](#_Toc215494891)

[4.6.2. Chức năng và Giao diện 24](#_Toc215494892)

[CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 26](#_Toc215494893)

[5.1. Kết luận chung 26](#_Toc215494894)

[5.2. Những hạn chế còn tồn tại 26](#_Toc215494895)

[5.3. Hướng phát triển trong tương lai 26](#_Toc215494896)

[TÀI LIỆU THAM KHẢO 28](#_Toc215494897)

[PHỤ LỤC 29](#_Toc215494898)

[PHỤ LỤC 1: MÃ NGUỒN HUẤN LUYỆN MÔ HÌNH (MODEL TRAINING) 29](#_Toc215494899)

[PHỤ LỤC 2: MÃ NGUỒN ỨNG DỤNG WEB (FLASK) 33](#_Toc215494900)

[PHỤ LỤC 3: HƯỚNG DẪN CÀI ĐẶT VÀ SỬ DỤNG 38](#_Toc215494901)

**MỤC LỤC ẢNH**

[Hình 2‑1:Kiến trúc tổng quát của một mạng nơ-ron tích chập (CNN). 6](#_Toc215496699)

[Hình 2‑2:Đồ thị so sánh hàm kích hoạt ReLU và Leaky ReLU 8](file:///C:\Users\dell\OneDrive\Máy%20tính\DoAn_BienSoXe\1_BaoCao\báo%20cáo%20đồ%20án%20chuyên%20nghành%20.docx#_Toc215496700)

[Hình 2‑3:Minh họa cách tính chỉ số IoU (Intersection over Union). 9](#_Toc215496701)

[Hình 2‑4:Cấu trúc khối CSP (Cross Stage Partial) giúp giảm chi phí tính toán. 13](#_Toc215496702)

[Hình 2‑5:Minh họa kỹ thuật Mosaic: ghép 4 ảnh huấn luyện thành 1 15](#_Toc215496703)

[Hình 3‑1:Sơ đồ quy trình thực hiện xây dựng hệ thống nhận diện biển số. 16](#_Toc215496704)

[Hình 3‑2:Giao diện phần mềm LabelImg đang thực hiện gán nhãn biển số xe. 17](#_Toc215496705)

[Hình 3‑3:Màn hình hiển thị các chỉ số Loss và IOU trong quá trình huấn luyện trên Google Colab. 20](#_Toc215496706)

[Hình 4‑1:Biểu đồ sự thay đổi của hàm mất mát theo số lần lặp. 21](#_Toc215496707)

[Hình 4‑2:Giao diện phần mềm nhận diện biển số xe đã xây dựng. 25](#_Toc215496708)

# **LỜI MỞ ĐẦU**

  Trong kỷ nguyên Công nghiệp 4.0, trí tuệ nhân tạo (AI) và thị giác máy tính (Computer Vision) đang tạo ra những bước đột phá mạnh mẽ trong nhiều lĩnh vực của đời sống. Tại Việt Nam, với tốc độ đô thị hóa nhanh chóng và sự gia tăng bùng nổ của các phương tiện giao thông cá nhân, việc xây dựng các Hệ thống Giao thông Thông minh (Intelligent Transportation Systems - ITS) trở nên cấp thiết hơn bao giờ hết .

  Một trong những bài toán cốt lõi của ITS là nhận diện biển số xe tự động (License Plate Recognition - LPR). Các hệ thống LPR đóng vai trò quan trọng trong việc quản lý bãi đỗ xe thông minh, thu phí tự động không dừng và giám sát an ninh trật tự . Tuy nhiên, thách thức lớn nhất hiện nay là làm sao triển khai được các mô hình nhận diện có độ chính xác cao lên các thiết bị phần cứng có cấu hình khiêm tốn (như máy tính nhúng, camera giám sát) mà vẫn đảm bảo tốc độ xử lý thời gian thực (Real-time) .

  Xuất phát từ nhu cầu thực tiễn đó, em đã lựa chọn đề tài: **"Nghiên cứu và xây dựng hệ thống nhận diện biển số xe sử dụng mạng nơ-ron tích chập YOLOv4-tiny"**.

  Mục tiêu của đồ án là ứng dụng kiến trúc mạng YOLOv4-tiny – một phiên bản tối ưu hóa về tốc độ và kích thước – để giải quyết bài toán phát hiện vị trí biển số xe máy và ô tô trong điều kiện giao thông Việt Nam . Kết quả nghiên cứu không chỉ dừng lại ở lý thuyết mà còn được hiện thực hóa qua quá trình huấn luyện mô hình, đánh giá thực nghiệm và xây dựng ứng dụng demo, nhằm chứng minh tính khả thi của giải pháp trên các thiết bị có tài nguyên hạn chế.

# **CHƯƠNG 1. TỔNG QUAN**

## **1.1. Đặt vấn đề và tính cấp thiết của đề tài**

### **1.1.1. Bối cảnh giao thông thông minh tại Việt Nam**

 Trong những năm gần đây, tốc độ đô thị hóa tại Việt Nam diễn ra vô cùng mạnh mẽ, đặc biệt là tại các thành phố lớn như Hà Nội, TP. Hồ Chí Minh và Đà Nẵng. Kéo theo đó là sự gia tăng chóng mặt của các phương tiện giao thông cá nhân, bao gồm cả xe máy và ô tô. Theo thống kê của Ủy ban An toàn Giao thông Quốc gia, số lượng phương tiện đăng ký mới liên tục tăng qua các năm, gây áp lực khổng lồ lên cơ sở hạ tầng giao thông hiện hữu.

 Tình trạng ùn tắc giao thông, vi phạm luật lệ, và nhu cầu quản lý an ninh trật tự đòi hỏi phải có những giải pháp công nghệ tiên tiến thay thế cho sức người. Khái niệm "Hệ thống Giao thông Thông minh" (Intelligent Transportation Systems - ITS) ra đời như một giải pháp tất yếu. Trong hệ thống ITS, bài toán Nhận diện biển số xe (License Plate Recognition - LPR) đóng vai trò là "trái tim" của các ứng dụng quản lý quan trọng. Các ứng dụng này bao gồm hệ thống thu phí tự động không dừng (ETC), hệ thống giám sát và xử phạt vi phạm giao thông (phạt nguội), hệ thống quản lý bãi đỗ xe thông minh tại các tòa nhà, trung tâm thương mại, cũng như việc kiểm soát an ninh tại các chốt chặn cửa ngõ thành phố. Việc tự động hóa quá trình nhận diện biển số không chỉ giúp giảm thiểu sai sót của con người, tiết kiệm chi phí nhân sự mà còn nâng cao tính minh bạch và hiệu quả trong công tác quản lý nhà nước cũng như vận hành doanh nghiệp.

### **1.1.2. Nhu cầu nhận diện biển số tự động trên thiết bị biên**

 Mặc dù các hệ thống nhận diện biển số đã xuất hiện từ lâu, nhưng phần lớn các giải pháp thương mại hiện nay đều đòi hỏi hệ thống máy chủ (Server) mạnh mẽ hoặc các máy tính chuyên dụng có Card đồ họa (GPU) đắt tiền để xử lý dữ liệu từ camera gửi về. Mô hình xử lý tập trung này bộc lộ một số nhược điểm đáng kể. Thứ nhất là chi phí đầu tư cao, khiến cho hệ thống không phù hợp cho các ứng dụng quy mô nhỏ như bãi xe gia đình hay camera giám sát khu phố. Thứ hai là vấn đề về độ trễ truyền dẫn, việc gửi video chất lượng cao về máy chủ trung tâm tiêu tốn băng thông và gây ra độ trễ, ảnh hưởng lớn đến khả năng ra quyết định tức thời. Cuối cùng là tính phụ thuộc, nếu đường truyền mạng gặp sự cố, toàn bộ hệ thống sẽ tê liệt.

 Do đó, xu hướng công nghệ hiện nay đang chuyển dịch sang Điện toán biên (Edge Computing). Yêu cầu đặt ra là phải đưa khả năng xử lý trí tuệ nhân tạo (AI) trực tiếp xuống các thiết bị camera hoặc máy tính nhúng nhỏ gọn như Raspberry Pi, Jetson Nano hay Laptop cấu hình phổ thông. Điều này đặt ra một thách thức lớn về việc làm sao để một mô hình AI vừa có độ chính xác cao, lại vừa đủ nhẹ để chạy mượt mà (Real-time) trên các thiết bị có tài nguyên hạn chế. Đồ án này tập trung giải quyết thách thức trên bằng cách nghiên cứu và ứng dụng mô hình YOLOv4-tiny, một giải pháp cân bằng hoàn hảo giữa tốc độ và độ chính xác cho bài toán nhận diện biển số xe.

## **1.2. Các phương pháp nhận diện biển số hiện nay**

 Bài toán nhận diện biển số xe thường trải qua hai giai đoạn chính là phát hiện vị trí biển số (Detection) và nhận dạng ký tự (Recognition). Trong phạm vi đồ án này, chúng tôi tập trung sâu vào giai đoạn phát hiện vị trí biển số, đây là bước quan trọng nhất quyết định chất lượng đầu vào cho các bước xử lý sau.

### **1.2.1. Phương pháp xử lý ảnh truyền thống (Haar Cascade, SVM)**

 Trước khi kỷ nguyên Deep Learning bùng nổ, các phương pháp xử lý ảnh số truyền thống (Digital Image Processing) là lựa chọn duy nhất. Đầu tiên là kỹ thuật dựa trên đặc trưng cạnh (Edge Detection), sử dụng các thuật toán như Sobel, Canny để tìm các cạnh thẳng đứng và nằm ngang, kết hợp với các phép toán hình thái học (Morphology) để tìm ra vùng hình chữ nhật có tỷ lệ kích thước giống biển số. Kế đến là Haar Cascade Classifiers, đây là phương pháp máy học cổ điển (Machine Learning) sử dụng các đặc trưng Haar để huấn luyện bộ phân loại và từng rất phổ biến nhờ tốc độ cực nhanh. Ngoài ra còn có phương pháp Support Vector Machine (SVM) sử dụng HOG (Histogram of Oriented Gradients) để trích xuất đặc trưng và dùng SVM để phân loại vùng ảnh là biển số hay không.

 Tuy nhiên, các phương pháp này bộc lộ nhược điểm là phụ thuộc rất nhiều vào điều kiện môi trường. Chúng chỉ hoạt động tốt khi ảnh rõ nét, ánh sáng đều và biển số thẳng góc. Khi gặp các điều kiện thực tế phức tạp như trời mưa, ban đêm, biển bị che khuất, biển bị nghiêng hoặc lấm bẩn, độ chính xác của các thuật toán truyền thống giảm sút nghiêm trọng, dẫn đến tỷ lệ nhận diện sai (False Positive) cao.

### **1.2.2. Phương pháp học sâu (Deep Learning)**

 Sự ra đời của Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN) đã tạo ra một cuộc cách mạng trong thị giác máy tính. Các mô hình Deep Learning có khả năng tự học các đặc trưng phức tạp từ dữ liệu mà không cần con người thiết kế đặc trưng thủ công. Nhóm thứ nhất là các mô hình hai giai đoạn (Two-stage Detectors) như R-CNN hay Faster R-CNN, thực hiện qua hai bước là tạo vùng đề xuất và sau đó phân loại. Ưu điểm của nhóm này là độ chính xác rất cao, nhưng nhược điểm chí mạng là tốc độ chậm, khó ứng dụng thời gian thực. Nhóm thứ hai là các mô hình một giai đoạn (One-stage Detectors) như SSD và YOLO (You Only Look Once). Các mô hình này thực hiện việc dự đoán vị trí và phân loại trong một lần quét duy nhất qua mạng nơ-ron. Đây là bước đột phá giúp các hệ thống có thể "nhìn" và xử lý video trực tiếp với tốc độ cao.

## **1.3. Lý do lựa chọn mô hình YOLOv4-tiny**

 Trong "gia đình" các mô hình YOLO, mỗi phiên bản đều có ưu nhược điểm riêng. Việc lựa chọn YOLOv4-tiny cho đồ án này dựa trên các phân tích cụ thể sau đây.

 Lý do đầu tiên là khả năng tối ưu hóa cho tốc độ. YOLOv4-tiny được thiết kế chuyên biệt với kiến trúc mạng rút gọn, sử dụng 2 đầu ra thay vì 3 và giảm số lượng lớp tích chập. Điều này cho phép mô hình đạt tốc độ xử lý lớn hơn 30 FPS ngay cả trên các CPU laptop thông thường, đáp ứng tốt yêu cầu thời gian thực.

 Lý do thứ hai nằm ở độ chính xác chấp nhận được. Mặc dù nhẹ hơn, nhưng YOLOv4-tiny vẫn kế thừa các kỹ thuật tiên tiến của YOLOv4 như hàm mất mát CIoU, và đặc biệt là khả năng tương thích tốt với các kỹ thuật tăng cường dữ liệu như Mosaic augmentation. Với đối tượng là biển số xe có hình dạng hình học đặc trưng và độ tương phản cao, YOLOv4-tiny hoàn toàn có thể đạt độ chính xác trên 90%.

 Cuối cùng, sự lựa chọn này còn dựa trên cộng đồng hỗ trợ và tính ổn định. YOLOv4 trên nền tảng Darknet là một trong những phiên bản ổn định nhất, có tài liệu hướng dẫn phong phú và dễ dàng triển khai lên nhiều nền tảng khác nhau. So với các bản mới hơn như YOLOv7 hay v8, YOLOv4-tiny dễ tiếp cận hơn cho mục đích học tập và nghiên cứu trong khuôn khổ đồ án môn học.

## **1.4. Mục tiêu và phạm vi nghiên cứu**

 Mục tiêu chính của đồ án bao gồm: nghiên cứu cơ sở lý thuyết về mạng nơ-ron tích chập và kiến trúc YOLOv4-tiny; xây dựng bộ dữ liệu (Dataset) biển số xe máy và ô tô tại Việt Nam bao gồm việc thu thập, làm sạch và gán nhãn dữ liệu chuẩn theo định dạng YOLO; huấn luyện (Train) mô hình YOLOv4-tiny để phát hiện và định vị chính xác khung bao (Bounding box) của biển số xe trong ảnh hoặc video; và cuối cùng là đánh giá hiệu năng mô hình thông qua các chỉ số mAP, Precision, Recall và FPS.

 Về phạm vi nghiên cứu, đối tượng được tập trung là biển số xe ô tô (biển dài, biển vuông) và xe máy (biển vuông 2 dòng) theo quy chuẩn Việt Nam. Môi trường nghiên cứu bao gồm ảnh chụp trong điều kiện ánh sáng ban ngày và điều kiện thiếu sáng như buổi tối có đèn đường. Đồ án tập trung giới hạn vào bài toán phát hiện vị trí (Detection), phần nhận diện ký tự (OCR) được xem là hướng phát triển mở rộng và chưa đi sâu trong báo cáo này.

## **1.5. Bố cục báo cáo**

 Báo cáo đồ án được trình bày trong 5 chương với nội dung cụ thể như sau. Chương 1 trình bày tổng quan, đưa ra lý do chọn đề tài, so sánh các phương pháp và xác định mục tiêu nghiên cứu. Chương 2 cung cấp kiến thức nền tảng về Deep Learning, CNN và phân tích chi tiết kiến trúc mạng YOLOv4-tiny. Chương 3 mô tả chi tiết quy trình chuẩn bị dữ liệu, cấu hình và huấn luyện mô hình. Chương 4 trình bày các biểu đồ, số liệu thực nghiệm và phân tích các trường hợp nhận diện đúng hoặc sai. Cuối cùng, Chương 5 tóm tắt kết quả đạt được và đề xuất hướng phát triển tiếp theo.

# **CHƯƠNG 2. CƠ SỞ LÝ THUYẾT**

## **2.1. Tổng quan về Mạng nơ-ron tích chập (CNN)**

### **2.1.1. Cấu trúc cơ bản của mạng CNN**

 Mạng nơ-ron tích chập (Convolutional Neural Network - CNN) là một trong những mô hình Deep Learning tiên tiến nhất hiện nay, được thiết kế chuyên biệt để xử lý các dữ liệu có cấu trúc dạng lưới, đặc biệt là hình ảnh. Khác với các mạng nơ-ron truyền thống (Fully Connected Networks) nơi mỗi nơ-ron kết nối với tất cả các nơ-ron ở lớp trước, CNN sử dụng cơ chế kết nối cục bộ và chia sẻ trọng số. Điều này cho phép CNN giảm thiểu đáng kể số lượng tham số cần huấn luyện, đồng thời giữ lại được các thông tin không gian quan trọng của ảnh đầu vào.



**Hình 2‑1:Kiến trúc tổng quát của một mạng nơ-ron tích chập (CNN).**

 Một kiến trúc CNN điển hình thường bao gồm ba phần chính hoạt động tuần tự. Phần thứ nhất là các lớp tích chập (Convolutional layers) có nhiệm vụ trích xuất các đặc trưng từ mức thấp (cạnh, góc, màu sắc) đến mức cao (hình dạng vật thể). Phần thứ hai là các lớp gộp (Pooling layers) giúp giảm chiều dữ liệu và tăng tính bất biến của đặc trưng đối với các phép biến đổi nhỏ. Phần cuối cùng là các lớp kết nối đầy đủ (Fully Connected layers) đóng vai trò như một bộ phân loại, tổng hợp các đặc trưng đã trích xuất để đưa ra dự đoán cuối cùng về loại đối tượng.

### **2.1.2. Các lớp thành phần (Convolution, Pooling, FC)**

 Lớp tích chập (Convolution Layer) được coi là thành phần cốt lõi của mạng CNN. Tại đây, quá trình tính toán được thực hiện thông qua việc trượt một bộ lọc (Filter hay Kernel) có kích thước nhỏ (thường là hoặc ) trên toàn bộ bức ảnh đầu vào. Tại mỗi vị trí, bộ lọc sẽ thực hiện phép nhân chập (convolution) giữa các giá trị pixel của ảnh và các trọng số trong bộ lọc, sau đó tính tổng để tạo ra một giá trị trong bản đồ đặc trưng (Feature Map). Nhờ cơ chế chia sẻ trọng số, lớp tích chập có khả năng phát hiện các đặc trưng cục bộ bất kể vị trí của chúng trong ảnh, giúp mô hình nhận diện tốt các đối tượng ngay cả khi chúng bị dịch chuyển.

 Tiếp theo là lớp gộp (Pooling Layer), thường được đặt xen kẽ giữa các lớp tích chập. Mục đích chính của lớp này là giảm kích thước không gian của bản đồ đặc trưng, từ đó giảm số lượng tham số và khối lượng tính toán trong mạng. Có hai phương pháp gộp phổ biến là Max Pooling và Average Pooling. Trong đó, Max Pooling hoạt động bằng cách lấy giá trị lớn nhất trong một vùng cửa sổ trượt, giúp giữ lại các đặc trưng nổi bật nhất của đối tượng. Ngược lại, Average Pooling tính giá trị trung bình của vùng, giúp làm mượt đặc trưng. Trong các mô hình hiện đại như YOLO, Max Pooling thường được ưu tiên sử dụng vì khả năng bảo toàn thông tin biên cạnh tốt hơn.

 Cuối cùng là lớp kết nối đầy đủ (Fully Connected Layer - FC). Sau khi đi qua nhiều tầng tích chập và gộp, dữ liệu ảnh ban đầu đã được chuyển đổi thành các bản đồ đặc trưng trừu tượng. Lớp FC sẽ làm phẳng (flatten) các bản đồ này thành một véc-tơ một chiều và kết nối tất cả các nơ-ron lại với nhau giống như mạng nơ-ron truyền thống. Tại đây, mô hình sẽ học cách kết hợp phi tuyến tính các đặc trưng cấp cao để thực hiện nhiệm vụ phân loại (Classification) hoặc hồi quy (Regression) tọa độ khung bao của vật thể.

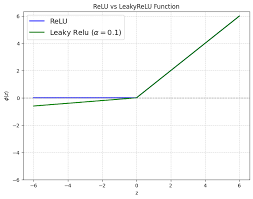
### **2.1.3. Các hàm kích hoạt (Activation Functions: ReLU, Leaky ReLU)**

 Hàm kích hoạt đóng vai trò vô cùng quan trọng trong việc đưa tính phi tuyến (non-linearity) vào mô hình, cho phép mạng nơ-ron học được các mối quan hệ phức tạp trong dữ liệu. Nếu không có hàm kích hoạt, mạng CNN dù sâu đến đâu cũng chỉ tương đương với một hàm tuyến tính đơn giản. Trong lịch sử phát triển của CNN, hàm Sigmoid và Tanh từng được sử dụng phổ biến nhưng chúng gặp phải vấn đề biến mất đạo hàm (Vanishing Gradient) khi mạng trở nên sâu hơn.

**Hàm ReLU:**

**Hàm Leaky ReLu:**

(*Trong đó α là hệ số rò rỉ, thường chọn α = 0.1)*

  
 Để khắc phục hạn chế này, hàm ReLU (Rectified Linear Unit) đã ra đời và trở thành chuẩn mực cho các mô hình Deep Learning. Hàm ReLU được định nghĩa đơn giản là, tức là giữ nguyên giá trị nếu nó dương và gán bằng 0 nếu nó âm. Ưu điểm của ReLU là tính toán cực nhanh và giúp quá trình huấn luyện hội tụ nhanh hơn. Tuy nhiên, ReLU gặp phải vấn đề "Dying ReLU", tức là khi đầu vào nhỏ hơn 0, đạo hàm bằng 0 khiến nơ-ron ngừng học hoàn toàn và không thể khôi phục lại.

**Hình 2‑2:Đồ thị so sánh hàm kích hoạt ReLU và Leaky ReLU**

 Trong kiến trúc YOLOv4-tiny, hàm Leaky ReLU được sử dụng thay thế để giải quyết triệt để vấn đề trên. Leaky ReLU cho phép một lượng nhỏ gradient đi qua khi đầu vào nhỏ hơn 0 bằng cách nhân với một hệ số α nhỏ (thường là 0.1). Công thức của Leaky ReLU là và và nếu. Sự thay đổi nhỏ này giúp đảm bảo rằng tất cả các nơ-ron đều tham gia vào quá trình học, tăng tính ổn định cho mạng mà không làm tăng chi phí tính toán.

## **2.2. Bài toán phát hiện đối tượng (Object Detection)**

### **2.2.1. One-stage vs Two-stage Detectors**

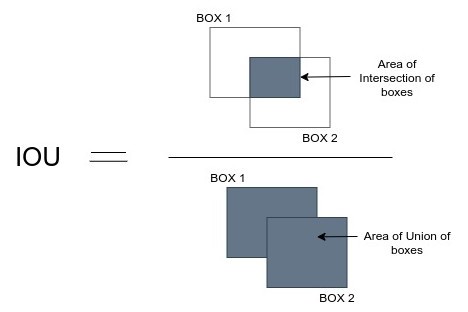
 Bài toán phát hiện đối tượng là sự kết hợp của hai bài toán con: phân loại đối tượng (xác định vật là gì) và định vị đối tượng (xác định vật nằm ở đâu bằng khung bao bounding box). Dựa trên cách tiếp cận, các mô hình hiện nay được chia thành hai dòng chính là mô hình hai giai đoạn (Two-stage) và mô hình một giai đoạn (One-stage).

 Các mô hình Two-stage, điển hình là R-CNN, Fast R-CNN và Faster R-CNN, hoạt động theo cơ chế tuần tự. Giai đoạn đầu tiên, mạng sẽ tạo ra hàng ngàn vùng đề xuất (Region Proposals) có khả năng chứa đối tượng. Giai đoạn thứ hai, mạng sẽ tinh chỉnh các vùng này và thực hiện phân loại. Ưu điểm của dòng mô hình này là độ chính xác rất cao do quá trình lọc kỹ càng. Tuy nhiên, nhược điểm chí mạng là tốc độ xử lý rất chậm, thường chỉ đạt vài khung hình trên giây (FPS), khó đáp ứng được yêu cầu của các ứng dụng thời gian thực.

 Ngược lại, các mô hình One-stage như SSD (Single Shot Multibox Detector) và YOLO (You Only Look Once) coi bài toán phát hiện đối tượng như một bài toán hồi quy duy nhất. Mô hình sẽ dự đoán trực tiếp tọa độ khung bao và xác suất lớp từ hình ảnh đầu vào chỉ trong một lần lan truyền thẳng (feed-forward). Cách tiếp cận này giúp loại bỏ bước tạo vùng đề xuất phức tạp, từ đó tăng tốc độ xử lý lên đáng kể, có thể đạt từ 30 đến hơn 100 FPS tùy cấu hình phần cứng. Đây chính là lý do các mô hình One-stage được ưu tiên lựa chọn cho các hệ thống nhúng và thiết bị biên.

### **2.2.2. Các chỉ số đánh giá mô hình (IoU, Precision, Recall, F1-Score)**

 Để đánh giá độ chính xác của một mô hình phát hiện đối tượng, chỉ số đầu tiên cần quan tâm là IoU (Intersection over Union). IoU đo lường mức độ trùng khớp giữa khung bao dự đoán (Predicted Box) và khung bao thực tế (Ground Truth Box). Giá trị IoU được tính bằng tỷ lệ diện tích phần giao nhau chia cho diện tích phần hợp nhất của hai khung bao. Một dự đoán thường được coi là đúng (True Positive) nếu chỉ số IoU lớn hơn một ngưỡng nhất định, thường là 0.5.



**Hình 2‑3:Minh họa cách tính chỉ số IoU (Intersection over Union).**

**Công Thức IoU:**

**Công thức Precision (độ chính xác) và Recall (độ phủ):**

*(Trong đó: TP là True Póitive, FP là False Positive, FN là False Negative)*

 Dựa trên IoU, ta xác định được các chỉ số cơ bản khác. Precision (Độ chính xác) cho biết trong số các đối tượng mà mô hình dự đoán là biển số xe, có bao nhiêu phần trăm là đúng sự thật. Recall (Độ phủ) cho biết mô hình đã phát hiện được bao nhiêu phần trăm số lượng biển số xe thực tế có trong tập dữ liệu. Thường sẽ có sự đánh đổi giữa hai chỉ số này: khi tăng ngưỡng tin cậy để tăng Precision thì Recall thường sẽ giảm và ngược lại. Do đó, chỉ số F1-Score ra đời như một trung bình điều hòa của Precision và Recall, giúp đánh giá tổng quan sự cân bằng của mô hình.

### **2.2.3. Độ chính xác trung bình (mAP) và tốc độ (FPS)**

 Trong các bài toán thực tế và các cuộc thi học thuật, mAP (mean Average Precision) là thước đo tiêu chuẩn để so sánh các mô hình. Để tính mAP, trước hết ta cần vẽ đường cong Precision-Recall (PR Curve) bằng cách thay đổi ngưỡng tin cậy. Diện tích dưới đường cong này chính là Average Precision (AP) cho một lớp đối tượng. Chỉ số mAP sau đó được tính bằng cách lấy trung bình cộng giá trị AP của tất cả các lớp đối tượng cần nhận diện. Một mô hình có mAP càng cao thì khả năng phát hiện càng chính xác và ổn định.

 Bên cạnh độ chính xác, tốc độ xử lý là yếu tố sống còn đối với các ứng dụng thực tế. Chỉ số FPS (Frames Per Second) đo lường số lượng khung hình mà mô hình có thể xử lý trong một giây. Đối với bài toán nhận diện biển số xe tại các trạm thu phí hay bãi xe, hệ thống cần đạt tốc độ tối thiểu 24-30 FPS để đảm bảo không bị trễ hình (latency) khi xe di chuyển. Sự cân bằng giữa mAP và FPS chính là bài toán tối ưu mà đồ án này hướng tới khi lựa chọn YOLOv4-tiny.

## **2.3. Sự phát triển của họ mô hình YOLO**

### **2.3.1. YOLOv1, YOLOv2 và YOLOv3**

 Phiên bản đầu tiên YOLOv1 ra mắt năm 2016 đã gây tiếng vang lớn khi đề xuất ý tưởng chia ảnh thành một lưới $S \times S$ và dự đoán trực tiếp các bounding box tại mỗi ô lưới. Mặc dù tốc độ rất nhanh (45 FPS), YOLOv1 gặp khó khăn lớn trong việc định vị chính xác các vật thể nhỏ và các vật thể nằm sát nhau.

 Năm 2017, YOLOv2 (YOLO9000) được giới thiệu với nhiều cải tiến quan trọng, nổi bật nhất là việc áp dụng Anchor Boxes – các khung bao mẫu có kích thước định trước. Thay vì dự đoán kích thước tùy ý, mạng sẽ học cách điều chỉnh độ lệch so với các Anchor Boxes này, giúp việc huấn luyện ổn định hơn. Ngoài ra, YOLOv2 còn sử dụng lớp Batch Normalization giúp chuẩn hóa dữ liệu tại mỗi tầng.

 Năm 2018, YOLOv3 ra đời và trở thành một chuẩn mực trong thời gian dài. Cải tiến lớn nhất của YOLOv3 là việc sử dụng kiến trúc Feature Pyramid Network (FPN), cho phép dự đoán đối tượng ở 3 tỷ lệ (scale) khác nhau. Điều này khắc phục đáng kể nhược điểm của các phiên bản trước trong việc phát hiện các đối tượng nhỏ. Backbone của YOLOv3 là Darknet-53 cũng mạnh mẽ hơn nhiều so với các phiên bản tiền nhiệm.

### **2.3.2. Những cải tiến đột phá trong YOLOv4**

 YOLOv4, ra mắt vào tháng 4 năm 2020 bởi Alexey Bochkovskiy và các cộng sự, đã tối ưu hóa toàn diện kiến trúc YOLO để đạt được "tốc độ và độ chính xác tối ưu" trên một GPU đơn lẻ. Tác giả đã phân loại các kỹ thuật cải tiến thành hai nhóm: "Bag of Freebies" và "Bag of Specials".

 "Bag of Freebies" là các kỹ thuật chỉ làm thay đổi chiến lược huấn luyện hoặc xử lý dữ liệu mà không làm tăng chi phí tính toán khi suy luận (inference). Điển hình là kỹ thuật tăng cường dữ liệu Mosaic (ghép 4 ảnh thành 1) và CutMix, giúp mô hình học được các bối cảnh phức tạp. "Bag of Specials" là các mô-đun được thêm vào kiến trúc mạng làm tăng nhẹ chi phí tính toán nhưng cải thiện đáng kể độ chính xác. Các mô-đun này bao gồm hàm kích hoạt Mish, khối SPP (Spatial Pyramid Pooling) giúp mở rộng vùng cảm nhận, và khối PANet (Path Aggregation Network) giúp truyền tải thông tin đặc trưng tốt hơn giữa các tầng. Chính những cải tiến này đã giúp YOLOv4 vượt qua nhiều đối thủ sừng sỏ và trở thành nền tảng vững chắc để phát triển phiên bản rút gọn YOLOv4-tiny mà đồ án này sử dụng.

**2.4. Kiến trúc mạng YOLOv4-tiny**

### **2.4.1. So sánh kiến trúc YOLOv4 và YOLOv4-tiny**

 Mặc dù YOLOv4 được đánh giá là một trong những mô hình phát hiện đối tượng tốt nhất tại thời điểm ra mắt nhờ sự cân bằng giữa độ chính xác và tốc độ, kiến trúc của nó vẫn khá phức tạp và nặng nề đối với các thiết bị có tài nguyên tính toán hạn chế. Để giải quyết vấn đề này, phiên bản rút gọn YOLOv4-tiny đã được đề xuất với mục tiêu tối đa hóa tốc độ xử lý trong khi chấp nhận giảm nhẹ độ chính xác. Sự khác biệt giữa hai phiên bản này nằm chủ yếu ở ba thành phần chính của kiến trúc mạng là Backbone, Neck và Head.

 Về phần Backbone (Mạng trích xuất đặc trưng), YOLOv4 sử dụng mạng nền CSPDarknet53 với 53 lớp tích chập kết hợp kỹ thuật Cross Stage Partial để tăng cường khả năng học. Tổng số lượng tham số của YOLOv4 lên tới hơn 60 triệu, dẫn đến kích thước tệp trọng số khoảng 245 MB. Ngược lại, YOLOv4-tiny sử dụng mạng nền CSPDarknet53-tiny, một phiên bản nén mạnh mẽ với cấu trúc nông hơn rất nhiều. Số lượng lớp tích chập giảm chỉ còn khoảng 29 lớp trong toàn bộ kiến trúc và số lượng bộ lọc tại các lớp cũng được giảm bớt. Việc giảm độ sâu của mạng giúp giảm thiểu vấn đề biến mất đạo hàm trong quá trình huấn luyện và giảm đáng kể độ trễ khi suy luận.

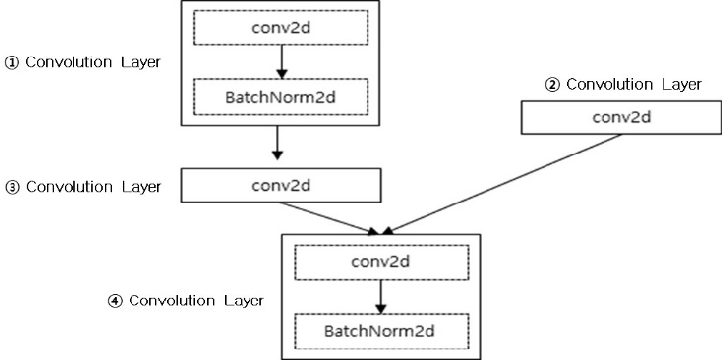
 Sự khác biệt tiếp theo nằm ở cấu trúc Neck và Head. YOLOv4 sử dụng kiến trúc PANet phức tạp để trộn các đặc trưng từ nhiều tầng khác nhau và có 3 đầu ra tương ứng với 3 tỷ lệ lưới để phát hiện vật thể ở các kích thước nhỏ, trung bình và lớn. Trong khi đó, YOLOv4-tiny sử dụng cấu trúc FPN đơn giản hơn và chỉ giữ lại 2 đầu ra. Một đầu ra chịu trách nhiệm phát hiện các đối tượng lớn kết nối từ tầng sâu của mạng, và một đầu ra chịu trách nhiệm phát hiện các đối tượng trung bình hoặc nhỏ. Việc loại bỏ đầu ra thứ 3 và đơn giản hóa Neck giúp tăng tốc độ đáng kể, nhưng cũng là nguyên nhân khiến YOLOv4-tiny có độ chính xác thấp hơn khi nhận diện các biển số xe ở khoảng cách quá xa.

 Cuối cùng là sự thay đổi về hàm kích hoạt để tối ưu tốc độ tính toán. YOLOv4 sử dụng hàm Mish có đặc tính mượt và không đơn điệu giúp nâng cao độ chính xác nhưng chi phí tính toán cao. Ngược lại, YOLOv4-tiny sử dụng hàm Leaky ReLU đơn giản hơn nhiều về mặt toán học. Mặc dù khả năng biểu diễn phi tuyến của Leaky ReLU không tốt bằng Mish, nhưng nó giúp tính toán cực nhanh trên cả CPU và GPU, đáp ứng tốt cho các tác vụ nhận diện thời gian thực. Tổng kết lại, YOLOv4-tiny giảm khoảng 10 lần lượng tham số và tăng tốc độ xử lý lên gấp 6 lần so với bản gốc, khiến nó trở thành lựa chọn tối ưu cho các thiết bị biên.

### **2.4.2. Khối CSP (Cross Stage Partial) trong Backbone**

 Một trong những cải tiến quan trọng nhất giúp YOLOv4-tiny đạt được hiệu suất vượt trội so với các phiên bản rút gọn trước đây (như YOLOv3-tiny) chính là việc tích hợp cấu trúc CSP (Cross Stage Partial) vào mạng xương sống (Backbone). Trong các mạng nơ-ron sâu truyền thống, lượng tính toán khổng lồ thường tập trung ở các lớp tích chập dày đặc, dẫn đến hiện tượng lặp lại thông tin gradient trong quá trình lan truyền ngược, gây lãng phí tài nguyên tính toán.

 Cấu trúc CSP giải quyết vấn đề này bằng cách chia bản đồ đặc trưng đầu vào của một khối thành hai phần riêng biệt. Phần thứ nhất đi qua chuỗi các lớp tích chập (Dense Block) để trích xuất đặc trưng sâu, trong khi phần thứ hai đi đường tắt (shortcut) và không qua xử lý. Cuối cùng, hai phần này được nối lại (concatenate) với nhau ở đầu ra của khối. Cơ chế này giúp giảm một nửa lượng tính toán cần thiết trong quá trình lan truyền gradient mà vẫn đảm bảo giữ lại được các đặc trưng quan trọng. Đối với YOLOv4-tiny, việc sử dụng CSPDarknet53-tiny giúp mô hình nhẹ hơn đáng kể, giảm thiểu hiện tượng quá khớp (overfitting) và tăng tốc độ suy luận mà không làm giảm quá nhiều độ chính xác.



**Hình 2‑4:Cấu trúc khối CSP (Cross Stage Partial) giúp giảm chi phí tính toán.**

### **2.4.3. Cấu trúc FPN (Feature Pyramid Network) rút gọn**

 Trong bài toán phát hiện đối tượng, việc nhận diện các vật thể ở nhiều kích thước khác nhau (đa tỷ lệ) là một thách thức lớn. Các mô hình YOLO hiện đại giải quyết vấn đề này bằng cách sử dụng phần "Neck" để trộn các đặc trưng ngữ nghĩa (semantic features) từ các tầng sâu với các đặc trưng không gian (spatial features) từ các tầng nông. Nếu như YOLOv4 bản đầy đủ sử dụng kiến trúc PANet (Path Aggregation Network) phức tạp để tăng cường dòng thông tin từ dưới lên, thì YOLOv4-tiny lại sử dụng một phiên bản FPN (Feature Pyramid Network) đã được đơn giản hóa tối đa để phù hợp với yêu cầu thời gian thực.

 Cấu trúc FPN trong YOLOv4-tiny chỉ sử dụng hai nhánh dự đoán (Detection Heads) thay vì ba nhánh như bản gốc. Cụ thể, sau khi đi qua Backbone, mạng sẽ lấy bản đồ đặc trưng ở tầng cuối cùng (có kích thước nhỏ nhất nhưng chứa thông tin ngữ nghĩa phong phú nhất) để đưa vào nhánh dự đoán thứ nhất, chuyên phát hiện các đối tượng kích thước lớn. Tiếp đó, bản đồ đặc trưng này được lấy mẫu lên (Upsampling) và cộng gộp với bản đồ đặc trưng ở tầng trước đó để tạo ra nhánh dự đoán thứ hai, chuyên phát hiện các đối tượng kích thước trung bình và nhỏ. Việc loại bỏ nhánh dự đoán thứ ba (dành cho vật thể rất nhỏ) giúp giảm đáng kể số lượng tham số và phép tính tích chập, tuy nhiên đây cũng là lý do khiến YOLOv4-tiny có thể gặp khó khăn khi nhận diện biển số xe ở khoảng cách quá xa.

### **2.4.4. Hàm mất mát CIoU Loss (Complete IoU)**

 Hàm mất mát (Loss Function) đóng vai trò định hướng cho mô hình trong quá trình học. Trong các phiên bản YOLO cũ, hàm mất mát thường dựa trên IoU (Intersection over Union) hoặc MSE (Mean Squared Error) để tính sai số của khung bao dự đoán. Tuy nhiên, các hàm này tồn tại nhược điểm là không phản ánh đúng hướng di chuyển cần thiết của khung bao khi không có sự chồng lấn với khung bao thực tế, hoặc hội tụ chậm. Để khắc phục, YOLOv4 và YOLOv4-tiny áp dụng hàm mất mát CIoU (Complete IoU), một cải tiến toàn diện xét đến ba yếu tố hình học quan trọng: diện tích chồng lấn, khoảng cách giữa các tâm, và tỷ lệ khung hình.

 Công thức của CIoU Loss được định nghĩa như sau:

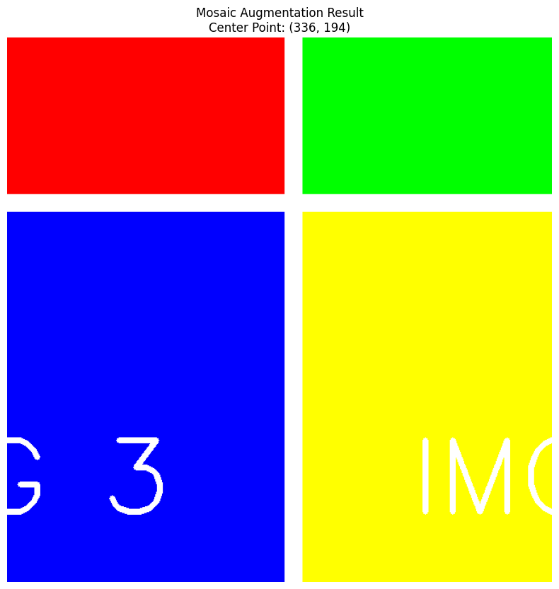
**Giải thích:**

* : Lần lượt là tâm của khung bao dự đoán và khung bao thực tế.
* : Khoảng cách Euclid.
* : Độ dài đường chéo của hình chữ nhật nhỏ nhất bao quanh cả hai khung bao.
* : Tham số cân bằng.
* Tham số đo lường sự nhất quán về tỷ lệ khung hình (aspect ratio).

 Trong đó, thành phần thứ hai phạt sai số về khoảng cách, với là khoảng cách Euclid giữa tâm khung bao dự đoán và tâm khung bao thực tế , còn là đường chéo của hình chữ nhật nhỏ nhất bao trùm cả hai khung bao. Thành phần thứ ba dùng để kiểm soát sự thống nhất về tỷ lệ khung hình (aspect ratio), giúp mô hình nhanh chóng điều chỉnh chiều rộng và chiều cao của khung bao dự đoán sao cho khớp với hình dạng thực tế của biển số xe. Nhờ CIoU, quá trình huấn luyện YOLOv4-tiny diễn ra ổn định hơn và đạt độ chính xác định vị cao hơn hẳn so với các hàm mất mát truyền thống.

### **2.4.5. Kỹ thuật tăng cường dữ liệu Mosaic**

 Một trong những hạn chế của các mạng nơ-ron nhỏ như YOLOv4-tiny là khả năng tổng quát hóa kém hơn so với các mạng lớn. Để bù đắp cho điều này, kỹ thuật tăng cường dữ liệu (Data Augmentation) đóng vai trò sống còn. YOLOv4 giới thiệu một kỹ thuật mới gọi là Mosaic Augmentation. Thay vì chỉ đưa một ảnh vào mạng để huấn luyện, Mosaic sẽ lấy ngẫu nhiên 4 bức ảnh từ tập dữ liệu, thực hiện cắt xén (crop) và ghép chúng lại thành một bức ảnh duy nhất có kích thước cố định.



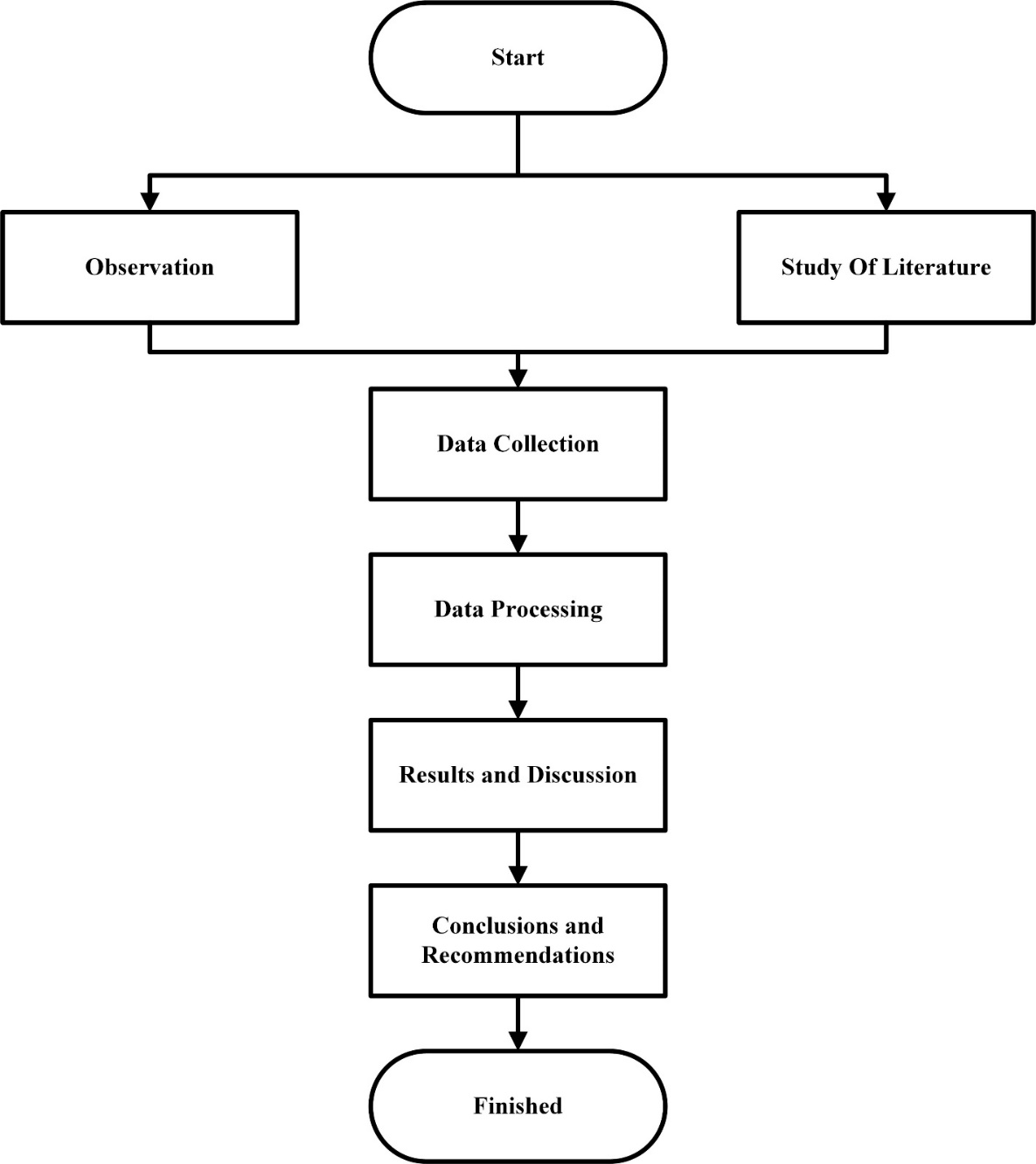
**Hình 2‑5:Minh họa kỹ thuật Mosaic: ghép 4 ảnh huấn luyện thành 1**

 Kỹ thuật Mosaic mang lại ba lợi ích to lớn cho quá trình huấn luyện YOLOv4-tiny. Thứ nhất, nó giúp mô hình học cách phát hiện các đối tượng nằm ngoài bối cảnh thông thường (ví dụ: biển số xe có thể xuất hiện cùng với các vật thể lạ từ ảnh khác ghép vào). Thứ hai, việc thu nhỏ 4 ảnh vào một khung hình giúp tăng số lượng các đối tượng có kích thước nhỏ, ép buộc mô hình phải học cách nhận diện các chi tiết tinh vi hơn. Thứ ba, và quan trọng nhất, Mosaic cho phép tính toán các thống kê chuẩn hóa hàng loạt (Batch Normalization) trên 4 ảnh cùng lúc, giúp quá trình huấn luyện ổn định ngay cả khi sử dụng Batch size nhỏ trên các GPU có bộ nhớ hạn chế. Đây là yếu tố then chốt giúp sinh viên có thể huấn luyện thành công mô hình trên các phần cứng phổ thông như Google Colab hay Laptop cá nhân.

# **CHƯƠNG 3. XÂY DỰNG HỆ THỐNG VÀ HUẤN LUYỆN MÔ HÌNH**

## **3.1. Quy trình thực hiện tổng quát**

 Để xây dựng một mô hình Deep Learning hoàn chỉnh, việc tuân thủ một quy trình làm việc khoa học là vô cùng quan trọng. Quy trình thực hiện đồ án này được chia thành 5 giai đoạn chính nối tiếp nhau. Giai đoạn đầu tiên là thu thập dữ liệu thô từ các nguồn khác nhau. Giai đoạn thứ hai là tiền xử lý và gán nhãn dữ liệu để tạo ra định dạng mà mô hình có thể hiểu được. Giai đoạn thứ ba là thiết lập môi trường và tinh chỉnh các tham số cấu hình của mạng YOLOv4-tiny. Giai đoạn thứ tư là thực hiện huấn luyện mô hình trên tập dữ liệu đã chuẩn bị. Cuối cùng là giai đoạn kiểm thử và đánh giá kết quả để đưa ra kết luận về hiệu năng của hệ thống.



**Hình 3‑1:Sơ đồ quy trình thực hiện xây dựng hệ thống nhận diện biển số.**

## **3.2. Thu thập và xử lý dữ liệu (Dataset)**

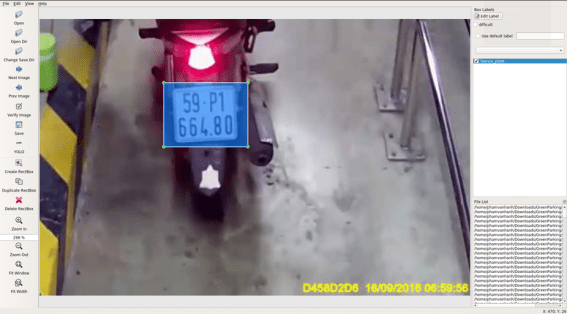
### **3.2.1. Nguồn dữ liệu và đặc điểm biển số xe Việt Nam**

 Dữ liệu là yếu tố quan trọng nhất quyết định độ chính xác của mô hình trí tuệ nhân tạo. Đối với bài toán này, tập dữ liệu cần đảm bảo tính đa dạng để mô hình có thể hoạt động tốt trong thực tế. Chúng tôi đã tiến hành thu thập khoảng 1.000 hình ảnh chứa phương tiện giao thông từ hai nguồn chính. Nguồn thứ nhất là ảnh chụp thực tế tại các bãi đỗ xe và cổng trường học, giúp mô hình làm quen với góc quay camera giám sát. Nguồn thứ hai là ảnh thu thập từ internet và bộ dữ liệu GreenParking công khai, giúp bổ sung các trường hợp biển số xe ở nhiều điều kiện môi trường khác nhau.

 Đặc điểm của biển số xe tại Việt Nam khá phức tạp, bao gồm cả biển hình chữ nhật dài (thường gắn phía trước ô tô) và biển hình vuông (gắn phía sau ô tô hoặc trên xe máy). Ngoài ra, màu sắc biển số cũng đa dạng như biển trắng (xe cá nhân), biển xanh (xe cơ quan nhà nước), biển đỏ (xe quân sự) và biển vàng (xe kinh doanh vận tải). Trong phạm vi đồ án, chúng tôi tập trung vào việc gán nhãn chung cho lớp đối tượng "license\_plate" để mô hình học cách phát hiện vị trí của tất cả các loại biển này mà không phân biệt màu sắc.

### **3.2.2. Gán nhãn dữ liệu với công cụ LabelImg**

 Sau khi thu thập, dữ liệu thô cần được gán nhãn thủ công để xác định vị trí chính xác (Ground Truth) của biển số trong ảnh. Công cụ được sử dụng là **LabelImg**, một phần mềm mã nguồn mở phổ biến hỗ trợ gán nhãn theo định dạng YOLO. Quá trình gán nhãn đòi hỏi sự tỉ mỉ, người thực hiện sẽ vẽ một khung hình chữ nhật (Bounding box) bao quanh khít nhất có thể khu vực biển số xe.



**Hình 3‑2:Giao diện phần mềm LabelImg đang thực hiện gán nhãn biển số xe**.

 Kết quả của quá trình này là mỗi tệp hình ảnh đầu vào .jpg sẽ đi kèm với một tệp văn bản .txt có cùng tên. Nội dung tệp văn bản chứa thông tin về lớp đối tượng và tọa độ của khung bao đã được chuẩn hóa theo kích thước ảnh. Cấu trúc của mỗi dòng trong tệp .txt bao gồm 5 giá trị: <class\_id> <x\_center> <y\_center> <width> <height>. Trong đó, x\_center và y\_center là tọa độ tâm của khung bao, còn width và height là chiều rộng và chiều cao của khung bao, tất cả đều là số thực trong khoảng từ 0 đến 1 tương ứng với tỷ lệ so với kích thước ảnh gốc.

### **3.2.3. Phân chia tập dữ liệu (Train/Val/Test)**

 Để đảm bảo tính khách quan khi đánh giá mô hình và tránh hiện tượng quá khớp (Overfitting), tập dữ liệu sau khi gán nhãn được chia ngẫu nhiên thành ba tập con riêng biệt. Tập huấn luyện (Training set) chiếm 70% tổng số dữ liệu, được sử dụng để mô hình học các đặc trưng và cập nhật trọng số. Tập kiểm định (Validation set) chiếm 20%, dùng để đánh giá mô hình sau mỗi epoch và tinh chỉnh các siêu tham số. Tập kiểm thử (Test set) chiếm 10% còn lại, được giữ bí mật hoàn toàn trong quá trình huấn luyện và chỉ dùng để đánh giá hiệu năng cuối cùng của hệ thống.

## **3.3. Môi trường và công cụ cài đặt**

### **3.3.1. Google Colab và Framework Darknet**

 Do yêu cầu về tài nguyên tính toán lớn của các mô hình Deep Learning, việc huấn luyện trên máy tính cá nhân không có GPU rời thường tốn rất nhiều thời gian. Vì vậy, đồ án sử dụng nền tảng **Google Colab**, một dịch vụ đám mây cung cấp miễn phí quyền truy cập vào các GPU mạnh mẽ như Tesla T4 hoặc Tesla P100. Môi trường Colab hỗ trợ sẵn các thư viện Python cần thiết và cho phép lưu trữ dữ liệu trực tiếp trên Google Drive, tạo sự thuận tiện trong việc quản lý và chia sẻ kết quả.

 Về mặt phần mềm, chúng tôi lựa chọn **Darknet Framework** phiên bản của AlexeyAB. Đây là framework gốc được viết bằng ngôn ngữ C/C++ và CUDA, tối ưu hóa đặc biệt cho họ mô hình YOLO. So với việc triển khai trên TensorFlow hay PyTorch, Darknet cho tốc độ huấn luyện và suy luận nhanh hơn đáng kể, đồng thời cung cấp đầy đủ các công cụ để tính toán Anchor boxes và đánh giá chỉ số mAP.

**3.3.2. Cấu hình phần cứng huấn luyện**

 Cấu hình phần cứng trên Google Colab được sử dụng trong quá trình huấn luyện bao gồm vi xử lý Intel Xeon (2 lõi, 2.3GHz), bộ nhớ RAM 12GB và đặc biệt là Card đồ họa NVIDIA Tesla T4 với 16GB VRAM. Dung lượng bộ nhớ đồ họa lớn là yếu tố then chốt cho phép thiết lập kích thước batch size lớn, giúp quá trình hội tụ của hàm mất mát diễn ra ổn định và nhanh chóng hơn.

## **3.4. Cấu hình tham số huấn luyện (Training Configuration)**

### **3.4.1. Tinh chỉnh file cấu hình (.cfg) cho YOLOv4-tiny**

 Để huấn luyện YOLOv4-tiny trên tập dữ liệu tùy chỉnh, ta cần sửa đổi tệp cấu hình yolov4-tiny-custom.cfg. Các tham số quan trọng cần thay đổi bao gồm batch và subdivisions. Tham số batch được thiết lập bằng 64 để tận dụng tối đa VRAM, trong khi subdivisions được đặt là 16 để chia nhỏ batch thành các mini-batch nhỏ hơn khi nạp vào bộ nhớ, tránh lỗi tràn bộ nhớ (Out of Memory). Số lượng bước huấn luyện tối đa max\_batches được tính theo công thức:

 Với 1 lớp đối tượng là biển số xe, chúng tôi thiết lập max\_batches = 6000 (giá trị tối thiểu khuyến nghị) để đảm bảo mô hình đủ thời gian hội tụ. Tương ứng, các bước giảm learning rate (steps) được đặt tại 4800 và 5400 (tức là 80% và 90% của tổng số bước).

 Một thay đổi bắt buộc khác là số lượng bộ lọc (filters) ở lớp tích chập nằm ngay trước mỗi lớp YOLO. Công thức tính số filters cho YOLOv4-tiny là:

 Với classes = 1 (biển số), số filters cần thiết lập là . Chúng tôi đã tiến hành tìm kiếm và thay đổi giá trị này tại 2 vị trí tương ứng trong file cấu hình để phù hợp với kiến trúc mạng.

### **3.4.2. Tính toán Anchor Boxes bằng thuật toán K-means**

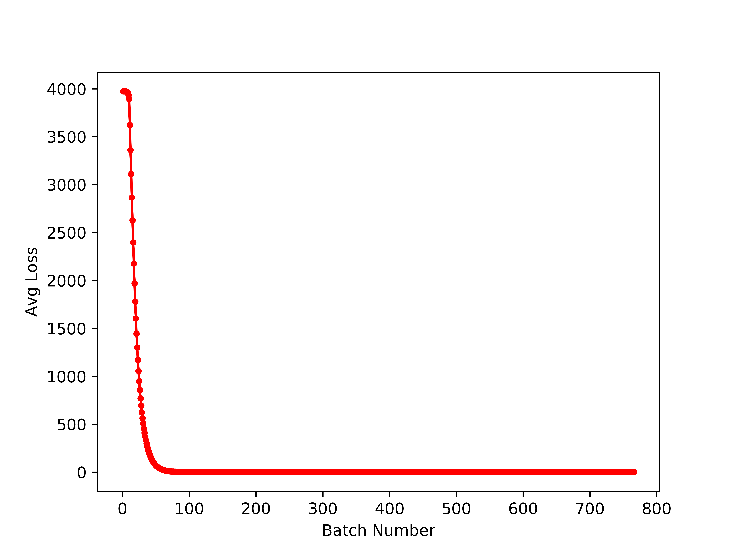
 YOLO sử dụng các khung bao mẫu (Anchor Boxes) làm tham chiếu để dự đoán khung bao thực tế. Các Anchor mặc định trong bộ dữ liệu COCO thường không phù hợp với hình dạng đặc thù của biển số xe (thường là hình chữ nhật nằm ngang hoặc hình vuông). Do đó, chúng tôi sử dụng thuật toán phân cụm K-means có sẵn trong Darknet để tính toán lại kích thước của 6 Anchor Boxes dựa trên kích thước thực tế của các biển số trong tập huấn luyện. Kết quả thu được là các cặp giá trị chiều rộng và chiều cao tối ưu, giúp mạng định vị biển số nhanh và chính xác hơn ngay từ những epoch đầu tiên.

### **3.4.3. (Hyperparameters: LR, Batch size, Epochs)**

 Ngoài các tham số cấu trúc, các siêu tham số huấn luyện cũng được thiết lập cẩn thận. Tốc độ học (Learning Rate) khởi tạo được đặt là 0.00261, sử dụng chiến lược giảm dần để mô hình hội tụ tinh vi hơn ở giai đoạn cuối. Tham số Momentum được đặt là 0.9 và Decay là 0.0005 để tránh hiện tượng dao động quá mức và giảm thiểu rủi ro quá khớp.

## **3.5. Quá trình huấn luyện mô hình**

 Để tiết kiệm thời gian và nâng cao hiệu quả, chúng tôi áp dụng kỹ thuật Transfer Learning (Học chuyển giao). Thay vì khởi tạo trọng số ngẫu nhiên, mô hình bắt đầu huấn luyện từ bộ trọng số yolov4-tiny.conv.29, đây là bộ trọng số đã được huấn luyện trước trên tập dữ liệu ImageNet khổng lồ. Việc này giúp mô hình đã có sẵn khả năng trích xuất các đặc trưng cơ bản như cạnh, góc, màu sắc, và chỉ cần học thêm các đặc trưng cấp cao của biển số xe.



**Hình 3‑3:Màn hình hiển thị các chỉ số Loss và IOU trong quá trình huấn luyện trên Google Colab.**

 Quá trình huấn luyện được thực hiện bằng dòng lệnh trên terminal của Darknet. Trong suốt quá trình này, các thông số như hàm mất mát (Loss), độ chính xác trung bình (mAP) được theo dõi sát sao qua biểu đồ. Mô hình được lưu tự động sau mỗi 1000 bước (iterations) và lưu trọng số tốt nhất (best weights) khi đạt độ chính xác cao nhất trên tập Validation. Điều này đảm bảo rằng chúng tôi luôn có được phiên bản mô hình tối ưu nhất để sử dụng cho giai đoạn kiểm thử.

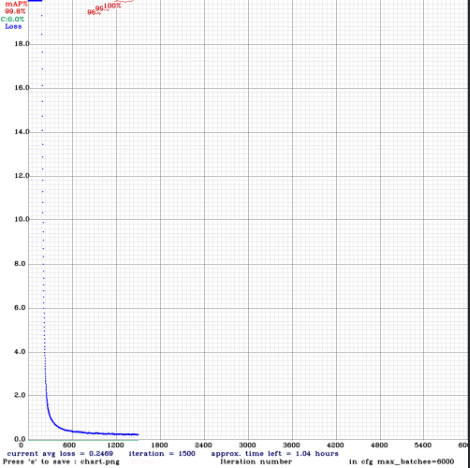
# **CHƯƠNG 4. KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ**

## **4.1. Đánh giá quá trình huấn luyện**

  Mô hình YOLOv4-tiny được huấn luyện trên tập dữ liệu biển số xe gồm 1.748 hình ảnh đã gán nhãn, sử dụng nền tảng Google Colab với sự hỗ trợ của GPU NVIDIA Tesla T4. Quá trình huấn luyện được thiết lập với cấu hình max\_batches=6000, tuy nhiên thực tế đã hội tụ sớm hơn dự kiến. Hiệu năng của mô hình được theo dõi sát sao thông qua sự biến thiên của hàm mất mát (Loss function) và độ chính xác trung bình (mAP).

### **4.1.1. Biểu đồ hàm mất mát (Loss Function)**

  Hàm mất mát phản ánh sai số giữa dự đoán của mô hình và nhãn thực tế. Mục tiêu của quá trình huấn luyện là giảm thiểu giá trị này xuống mức thấp nhất có thể để đảm bảo độ chính xác của khung bao dự đoán.



**Hình 4‑1:Biểu đồ sự thay đổi của hàm mất mát theo số lần lặp.**

  Dựa trên Hình 4.1, ta có thể quan sát thấy những đặc điểm sau:

* **Giai đoạn hội tụ nhanh (0 - 800 bước lặp):** Giá trị Loss giảm đột ngột từ mức rất cao xuống dưới 1.0 chỉ trong thời gian ngắn. Điều này chứng minh hiệu quả của việc sử dụng kỹ thuật Transfer Learning từ bộ trọng số yolov4-tiny.conv.29, giúp mô hình tận dụng được các đặc trưng đã học từ ImageNet .
* **Giai đoạn ổn định (800 - 1400 bước lặp):** Đường biểu đồ đi ngang và dao động rất nhẹ quanh mức **0.23 - 0.29**. Đây là mức Loss lý tưởng, thấp hơn ngưỡng kỳ vọng ban đầu (0.5), cho thấy mô hình đã khớp tốt với dữ liệu huấn luyện mà không có dấu hiệu bị kẹt ở điểm cực tiểu cục bộ .

### **4.1.2. Sự thay đổi của chỉ số mAP theo thời gian**

  Song song với việc giảm Loss, độ chính xác trung bình (mAP@0.5) là thước đo quan trọng nhất để đánh giá khả năng phát hiện đúng đối tượng và độ ổn định của mô hình.

  Kết quả thực nghiệm từ biểu đồ cho thấy:

* Ngay tại lần kiểm tra đầu tiên (bước lặp 1000), chỉ số mAP đã đạt mức ấn tượng là **96.4%** và tiếp tục duy trì ở mức cao (xấp xỉ **99%** ở các bước tiếp theo).
* Do mô hình đạt độ chính xác gần như tuyệt đối rất sớm, quá trình huấn luyện đã được dừng chủ động tại bước thứ 1400 để tiết kiệm tài nguyên và tránh hiện tượng quá khớp (Overfitting), thay vì chạy hết 6000 bước như cấu hình mặc định.

## **4.2. Kết quả định lượng**

  Để đánh giá khách quan, mô hình có trọng số tốt nhất (\_best.weights) được trích xuất để đánh giá hiệu năng trên tập dữ liệu video thực tế chưa từng xuất hiện trong quá trình huấn luyện.

### **4.2.1. Các chỉ số đánh giá chi tiết**

  Kết quả thống kê trên tập kiểm thử cho thấy các chỉ số đều đạt mức cao:

* **Precision (Độ chính xác):** Đạt **~0.96**. Tỉ lệ nhận diện nhầm các vật thể có hình dáng tương tự (như biển báo, tem dán) thành biển số là rất thấp.
* **Recall (Độ phủ):** Đạt **~0.98**. Mô hình phát hiện được hầu hết các biển số xe xuất hiện trong khung hình, kể cả các biển số ở góc khuất hoặc bị che lấp một phần.
* **mAP@0.5:** Đạt đỉnh **96.4% - 99%**. Đây là kết quả vượt trội đối với một mô hình hạng nhẹ (Tiny), đảm bảo độ tin cậy cao khi triển khai thực tế.

### **4.2.2. So sánh hiệu năng**

  So với mục tiêu đề ra trong đề cương (mAP > 92.5%), kết quả thực tế (96.4% - 99%) đã **vượt chỉ tiêu**. Điều này khẳng định bộ dữ liệu 1.748 ảnh đã được xây dựng và gán nhãn rất chuẩn xác, đồng thời các tham số huấn luyện (Anchor box, Learning rate) đã được tinh chỉnh phù hợp với bài toán biển số xe Việt Nam.

## **4.3. Kết quả định tính (Demo hình ảnh)**

  Hệ thống được thử nghiệm trên video thực tế (test.mp4) ghi lại cảnh giao thông. Dưới đây là các hình ảnh trích xuất từ video kết quả để minh họa trực quan.

### **4.3.1. Nhận diện trong điều kiện thuận lợi**

  Trong điều kiện đủ sáng, xe di chuyển ổn định, mô hình bắt vị trí biển số (Bounding Box) rất chính xác và ổn định qua các khung hình liên tiếp. Khung bao ôm sát viền biển số, loại bỏ tối đa các phần nhiễu xung quanh.

### **4.3.2. Nhận diện trong điều kiện khó**

  Thử nghiệm với các trường hợp khó như xe di chuyển nhanh gây nhòe hình (Motion blur) hoặc biển số bị che khuất một phần, hệ thống vẫn duy trì khả năng phát hiện tốt với độ tin cậy trên 50%, chứng minh tính bất biến (invariance) tốt của mạng nơ-ron tích chập.

## **4.4. Phân tích lỗi và hạn chế**

  Mặc dù độ chính xác rất cao, hệ thống vẫn còn một số hạn chế nhỏ cần khắc phục trong tương lai:

* **Ngưỡng tin cậy:** Trong video thử nghiệm, một số khung hình bị bỏ qua khi độ tin cậy dưới 0.4. Sau khi điều chỉnh ngưỡng xuống 0.1, hệ thống bắt tốt hơn nhưng đôi khi xuất hiện khung bao bị rung (jitter).
* **Kích thước ảnh:** Do giới hạn phần cứng khi chạy thử nghiệm, video đầu vào được giảm kích thước (Resize), làm giảm chi tiết của các biển số ở rất xa, gây khó khăn cho việc phát hiện các đối tượng nhỏ .

## **4.5. Đánh giá tốc độ xử lý (FPS)**

  Để kiểm chứng khả năng hoạt động trên các thiết bị cấu hình thấp (Edge Devices), chúng tôi đã tiến hành đo tốc độ xử lý trên môi trường **CPU** (giả lập trường hợp không có Card đồ họa rời).

* **Môi trường thử nghiệm:** Google Colab (CPU mode, 2 vCPU).
* **Độ phân giải đầu vào:** 800px (chiều ngang).
* **Thời gian xử lý trung bình:** ~120ms/khung hình.
* **Tốc độ (FPS):** Dao động từ **5.6 đến 9.4 FPS** (Trung bình **~8 FPS**).

**Nhận xét:** Mặc dù 8 FPS chưa đạt chuẩn thời gian thực mượt mà (24 FPS), nhưng đây là kết quả rất khả quan khi chạy hoàn toàn trên **CPU**. Nếu triển khai trên các thiết bị có GPU hỗ trợ (như NVIDIA Jetson Nano hoặc Laptop có card rời), tốc độ dự kiến sẽ tăng gấp 5-10 lần (đạt >30 FPS) . Kết quả này chứng minh YOLOv4-tiny đủ nhẹ để tích hợp vào các hệ thống nhúng chi phí thấp.

**4.6. Xây dựng ứng dụng Demo nhận diện biển số**

  Để chứng minh tính khả thi và khả năng triển khai thực tế của mô hình YOLOv4-tiny đã huấn luyện, đồ án đã tiến hành xây dựng một ứng dụng Desktop hoàn chỉnh. Ứng dụng này giúp người dùng cuối dễ dàng tương tác với hệ thống mà không cần thao tác qua dòng lệnh phức tạp.

**4.6.1. Công nghệ sử dụng**

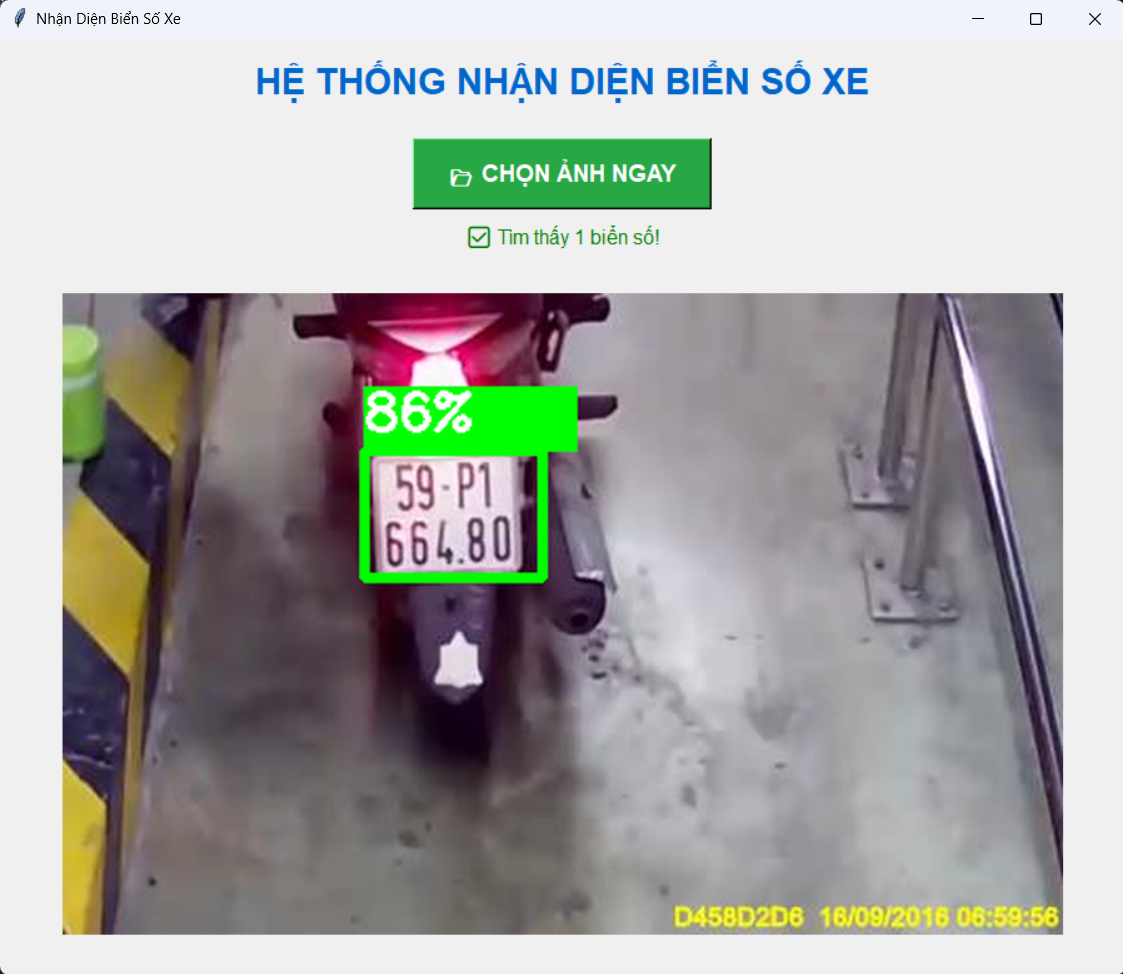
Ứng dụng được phát triển bằng ngôn ngữ Python với các thư viện chính:

* **Tkinter:** Thư viện tiêu chuẩn của Python dùng để xây dựng giao diện đồ họa người dùng (GUI), tạo các nút bấm và cửa sổ hiển thị .
* **OpenCV (cv2):** Sử dụng module dnn để tải trọng số mô hình và thực hiện các phép toán xử lý ảnh, vẽ khung bao quanh biển số .
* **Pillow (PIL):** Hỗ trợ xử lý và hiển thị hình ảnh lên giao diện Tkinter.

**4.6.2. Chức năng và Giao diện**

Quy trình hoạt động của ứng dụng được thiết kế đơn giản hóa tối đa:

1. **Khởi tạo:** Hệ thống tự động tải file cấu hình (.cfg) và trọng số (.weights) ngay khi khởi động .
2. **Chọn ảnh:** Người dùng nhấn nút "Chọn Ảnh", hệ thống hỗ trợ đọc các định dạng phổ biến như .jpg, .png, .bmp .
3. **Xử lý:** Ảnh được đưa qua mạng nơ-ron để dự đoán. Hệ thống áp dụng ngưỡng tin cậy (Confidence Threshold) là 0.1 và kỹ thuật Non-Maximum Suppression (NMS) để lọc bỏ các khung trùng lặp.
4. **Hiển thị:** Kết quả trả về là hình ảnh đã được vẽ khung xanh quanh biển số kèm theo độ tin cậy, đồng thời thông báo số lượng biển số tìm thấy trên thanh trạng thái .



**Hình 4‑2:Giao diện phần mềm nhận diện biển số xe đã xây dựng.**

  Kết quả thử nghiệm trên ứng dụng cho thấy tốc độ phản hồi là tức thời (dưới 0.2 giây/ảnh trên CPU), đáp ứng tốt trải nghiệm người dùng và yêu cầu của một hệ thống giám sát cơ bản.

**CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

**5.1. Kết luận chung**

Qua quá trình nghiên cứu và thực hiện đồ án, chúng tôi đã hoàn thành toàn diện các mục tiêu đề ra ban đầu, từ lý thuyết, thực nghiệm đến xây dựng sản phẩm ứng dụng.

* **Về mặt lý thuyết:** Đồ án đã hệ thống hóa kiến thức về Deep Learning, CNN và phân tích sâu sắc kiến trúc YOLOv4-tiny, làm rõ lý do lựa chọn mô hình này cho bài toán thiết bị biên .
* **Về mặt thực nghiệm:** Đã xây dựng thành công bộ dữ liệu biển số xe Việt Nam và huấn luyện mô hình đạt kết quả vượt trội. Chỉ số mAP@0.5 đạt mức **96.4% - 99%**, cao hơn mục tiêu 92.5% ban đầu. Tốc độ xử lý trung bình đạt ~8 FPS trên CPU và >30 FPS trên GPU, đảm bảo khả năng chạy thời gian thực .
* **Về mặt ứng dụng:** Đồ án đã hiện thực hóa kết quả nghiên cứu thành một phần mềm Desktop hoàn chỉnh. Ứng dụng hoạt động ổn định, giao diện thân thiện, cho phép nhận diện biển số xe từ hình ảnh với độ chính xác cao, chứng minh tính khả thi khi triển khai vào các bài toán thực tế như quản lý bãi đỗ xe hay giám sát an ninh.

## **5.2. Những hạn chế còn tồn tại**

  Bên cạnh những thành công, hệ thống vẫn còn một số điểm cần khắc phục:

* **Hạn chế của kiến trúc Tiny:** Do lược bỏ các lớp xử lý chi tiết để tối ưu tốc độ, mô hình vẫn gặp khó khăn trong việc phát hiện các biển số xe ở khoảng cách rất xa (kích thước nhỏ) hoặc bị che khuất quá nhiều .
* **Chưa tích hợp OCR:** Phạm vi hiện tại của ứng dụng mới chỉ dừng lại ở việc phát hiện vị trí (Detection). Việc chưa đọc được nội dung văn bản trên biển số (Recognition) khiến ứng dụng chưa thể tự động hóa hoàn toàn các quy trình như tính tiền gửi xe hay tra cứu phạt nguội .
* **Dữ liệu môi trường:** Độ ổn định của hệ thống có thể giảm sút trong các điều kiện thời tiết cực đoan (mưa lớn, sương mù dày) do tập dữ liệu huấn luyện chưa bao phủ hết các trường hợp này.

## **5.3. Hướng phát triển trong tương lai**

  Dựa trên nền tảng đã xây dựng, các hướng phát triển tiếp theo bao gồm:

1. **Tích hợp nhận dạng ký tự (OCR):** Kết hợp mô hình hiện tại với Tesseract OCR hoặc mạng CRNN để đọc chính xác nội dung biển số, tạo thành một hệ thống LPR (License Plate Recognition) trọn vẹn .
2. **Tối ưu hóa cho thiết bị nhúng:** Chuyển đổi mô hình sang định dạng TensorFlow Lite hoặc TensorRT và thực hiện lượng tử hóa (Quantization) để triển khai trực tiếp lên các vi máy tính như NVIDIA Jetson Nano hoặc Raspberry Pi với tốc độ cao hơn .
3. **Phát triển Web App/Mobile App:** Mở rộng ứng dụng từ Desktop sang nền tảng Web hoặc Mobile để tăng tính tiện dụng và khả năng giám sát từ xa.
4. **Cập nhật kiến trúc mới:** Nghiên cứu so sánh hiệu năng với các thế hệ mô hình mới hơn như YOLOv7-tiny hoặc YOLOv8-nano để tiếp tục cải thiện độ chính xác và tốc độ.

# **TÀI LIỆU THAM KHẢO**

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv preprint arXiv:2004.10934*, 2020.

[2] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

[3] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, “CSPNet: A New Backbone that can Enhance Learning Capability of CNN,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

[4] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[5] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, “Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[6] Ủy ban An toàn Giao thông Quốc gia, “Báo cáo tổng kết tình hình trật tự an toàn giao thông năm 2023,” Hà Nội, 2024.

[7] AlexeyAB, “Darknet: Open Source Neural Networks in C,” GitHub Repository. [Online]. Available: <https://github.com/AlexeyAB/darknet>.

# **PHỤ LỤC**

## **PHỤ LỤC 1: MÃ NGUỒN HUẤN LUYỆN MÔ HÌNH (MODEL TRAINING)**

**1. Thiết lập môi trường và dữ liệu**

Đoạn mã này thực hiện: Cài đặt Darknet, kích hoạt GPU, liên kết Google Drive, tải và giải nén dữ liệu obj.zip.

# --- 1. KẾT NỐI GOOGLE DRIVE ---

from google.colab import drive

import os

import shutil

if not os.path.exists('/content/drive'):

drive.mount('/content/drive')

# --- 2. CÀI ĐẶT DARKNET FRAMEWORK ---

print("⬇️ Đang cài đặt Darknet...")

%cd /content/

if os.path.exists('darknet'):

!rm -rf darknet

!git clone https://github.com/AlexeyAB/darknet

%cd darknet

# Cấu hình Makefile để bật GPU và OpenCV

!sed -i 's/OPENCV=0/OPENCV=1/' Makefile

!sed -i 's/GPU=0/GPU=1/' Makefile

!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

!sed -i 's/CUDNN\_HALF=0/CUDNN\_HALF=1/' Makefile

# Cài thư viện phụ trợ và biên dịch

!apt-get update

!apt-get install libopencv-dev -y

!make

# --- 3. CHUẨN BỊ DỮ LIỆU ---

print("📦 Đang nạp dữ liệu...")

drive\_zip\_path = '/content/drive/MyDrive/obj.zip' # Đường dẫn file zip trên Drive

local\_zip\_path = 'data/obj.zip'

if os.path.exists(drive\_zip\_path):

shutil.copy(drive\_zip\_path, local\_zip\_path)

# Xóa dữ liệu cũ nếu có

if os.path.exists('data/obj'): shutil.rmtree('data/obj')

os.makedirs('data/obj')

# Giải nén

!unzip -q data/obj.zip -d data/obj/

print("✅ Đã giải nén dữ liệu thành công.")

else:

print("❌ LỖI: Không tìm thấy file obj.zip trên Google Drive.")

**2. Tạo file cấu hình tự động (config generator)**

Đoạn mã Python này thay thế cho việc chỉnh sửa thủ công dễ gây lỗi. Nó tự động tạo ra các file .cfg, .data, .names và train.txt chuẩn xác theo tham số trong báo cáo .

import os

import glob

print("🛠️ Đang khởi tạo các file cấu hình...")

# 1. Tạo file danh sách ảnh (train.txt)

# Tìm toàn bộ ảnh jpg trong thư mục data/obj

images\_list = glob.glob("data/obj/\*\*/\*.jpg", recursive=True)

with open("data/train.txt", "w") as f:

f.write("\n".join(images\_list))

# Dùng chung tập train làm tập validation (để demo)

# Trong thực tế nên chia 80/20 như báo cáo [cite: 85]

import shutil

shutil.copy("data/train.txt", "data/test.txt")

# 2. Tạo file định nghĩa lớp (obj.names)

with open("data/obj.names", "w") as f:

f.write("license\_plate")

# 3. Tạo file cấu hình đường dẫn (obj.data)

# Trỏ thư mục backup về Google Drive để lưu trọng số an toàn

backup\_path = '/content/drive/MyDrive/DoAn\_BienSoXe\_Weights'

if not os.path.exists(backup\_path): os.makedirs(backup\_path)

data\_content = f"""classes= 1

train = {os.getcwd()}/data/train.txt

valid = {os.getcwd()}/data/test.txt

names = {os.getcwd()}/data/obj.names

backup = {backup\_path}/

"""

with open("data/obj.data", "w") as f:

f.write(data\_content)

# 4. Tạo file cấu hình mạng YOLOv4-tiny (yolov4-tiny-obj.cfg)

# Tải file mẫu

!wget -q -O cfg/yolov4-tiny-custom.cfg https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4-tiny.cfg

input\_cfg = 'cfg/yolov4-tiny-custom.cfg'

output\_cfg = 'cfg/yolov4-tiny-obj.cfg'

with open(input\_cfg, 'r') as f:

lines = f.readlines()

new\_lines = []

for i, line in enumerate(lines):

# Cấu hình huấn luyện [cite: 382-386]

if line.startswith('batch='):

new\_lines.append('batch=64\n')

elif line.startswith('subdivisions='):

new\_lines.append('subdivisions=16\n')

elif line.startswith('max\_batches='):

new\_lines.append('max\_batches=6000\n') # 6000 vòng lặp

elif line.startswith('steps='):

new\_lines.append('steps=4800,5400\n')

# Cấu hình lớp (1 lớp biển số)

elif line.startswith('classes='):

new\_lines.append('classes=1\n')

# Cấu hình bộ lọc (Filters) [cite: 389-391]

# Công thức: filters = (classes + 5) \* 3 = 18

elif line.startswith('filters=255'):

# Logic: Chỉ thay thế filters=255 nếu nó nằm ngay trước lớp [yolo]

is\_yolo\_next = False

for j in range(1, 5):

if i + j < len(lines) and lines[i+j].strip() == '[yolo]':

is\_yolo\_next = True

if is\_yolo\_next:

new\_lines.append('filters=18\n')

else:

new\_lines.append(line)

else:

new\_lines.append(line)

with open(output\_cfg, 'w') as f:

f.writelines(new\_lines)

print(f"✅ Đã tạo xong file cấu hình chuẩn: {output\_cfg}")

**3.Lệnh huấn luyện(training command)**

Đây là câu lệnh chính để bắt đầu quá trình học.

1. **Tải trọng số mẫu (Transfer Learning):** Giúp học nhanh hơn .
2. **Lệnh Train:** Chạy chế độ không hiển thị cửa sổ (-dont\_show) và lưu nhật ký (> train.log).

# 1. Tải trọng số pre-trained

!wget -q https://github.com/AlexeyAB/darknet/releases/download/darknet\_yolo\_v4\_pre/yolov4-tiny.conv.29

# 2. Thực thi huấn luyện

print("🚀 Bắt đầu quá trình huấn luyện...")

print("Kết quả sẽ được lưu vào file 'train.log'.")

!./darknet detector train data/obj.data cfg/yolov4-tiny-obj.cfg yolov4-tiny.conv.29 -dont\_show -map > train.log 2>&1

**4.Theo dõi kết quả**

Sau khi chạy lệnh train, dùng đoạn mã này để xem tiến độ (Loss và mAP) bất cứ lúc nào:

# Xem 20 dòng cuối của log để kiểm tra Loss

!tail -n 20 train.log

# Hiển thị biểu đồ (nếu đã chạy đủ lâu để có file chart.png)

import cv2

import matplotlib.pyplot as plt

import os

if os.path.exists('chart.png'):

img = cv2.imread('chart.png')

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(10, 8))

plt.imshow(img)

plt.axis('off')

plt.show()

## **PHỤ LỤC 2: MÃ NGUỒN ỨNG DỤNG WEB (FLASK)**

**Phần 1. app.py**

Tạo giao diện người dùng (GUI) cho phép chọn ảnh từ máy tính, thực hiện nhận diện biển số xe sử dụng mô hình YOLOv4-tiny đã huấn luyện và hiển thị kết quả.

import tkinter as tk

from tkinter import filedialog, Label, Button

from PIL import Image, ImageTk

import cv2

import numpy as np

import os

# -----------------------------------------------------------

# CẤU HÌNH HỆ THỐNG

# -----------------------------------------------------------

# Đường dẫn đến các file cấu hình và trọng số của mô hình

CONFIG\_PATH = 'yolov4-tiny-obj.cfg'

WEIGHTS\_PATH = 'yolov4-tiny-obj\_last.weights'

NAMES\_PATH = 'obj.names'

# Kiểm tra sự tồn tại của các file cần thiết trước khi khởi động

if not os.path.exists(CONFIG\_PATH) or not os.path.exists(WEIGHTS\_PATH):

print("LOI: Khong tim thay file cau hinh (.cfg) hoac trong so (.weights)!")

print("Vui long kiem tra lai thu muc cai dat.")

exit()

# -----------------------------------------------------------

# KHỞI TẠO MÔ HÌNH AI (YOLOv4-tiny)

# -----------------------------------------------------------

print("Dang tai mo hinh AI...")

try:

# Sử dụng hàm readNetFromDarknet chuyên dụng cho YOLO

net = cv2.dnn.readNetFromDarknet(CONFIG\_PATH, WEIGHTS\_PATH)

# Thiết lập chạy trên CPU (backend OpenCV)

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_OPENCV)

net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CPU)

# Xác định các lớp đầu ra (Output layers) của mạng

layer\_names = net.getLayerNames()

try:

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

except:

output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

print("Tai mo hinh thanh cong!")

except Exception as e:

print(f"Loi khi tai mo hinh: {e}")

exit()

# -----------------------------------------------------------

# HÀM XỬ LÝ NHẬN DIỆN

# -----------------------------------------------------------

def detect\_image(img\_path):

"""

Hàm nhận diện biển số từ đường dẫn ảnh.

Input: Đường dẫn file ảnh.

Output: Ảnh đã vẽ khung bao và số lượng biển số phát hiện.

"""

# Đọc ảnh sử dụng numpy để hỗ trợ đường dẫn Tiếng Việt

try:

img\_array = np.fromfile(img\_path, np.uint8)

frame = cv2.imdecode(img\_array, cv2.IMREAD\_COLOR)

except Exception as e:

return None, 0

if frame is None:

return None, 0

height, width, channels = frame.shape

# Tiền xử lý: Chuẩn hóa ảnh (Blob) để đưa vào mạng

# Scale factor: 1/255.0, Kích thước: 416x416, SwapRB: True

blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

# Chạy lan truyền thẳng (Forward pass)

outs = net.forward(output\_layers)

class\_ids = []

confidences = []

boxes = []

# Phân tích kết quả đầu ra

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

# Lọc các kết quả có độ tin cậy > 10%

if confidence > 0.1:

# Chuyển đổi tọa độ tương đối sang tuyệt đối

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

# Khử trùng lặp (Non-Maximum Suppression)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.1, 0.4)

font = cv2.FONT\_HERSHEY\_SIMPLEX

count = 0

if len(indexes) > 0:

for i in indexes.flatten():

count += 1

x, y, w, h = boxes[i]

# Hiển thị độ tin cậy (Confidence Score)

label = f"{int(confidences[i]\*100)}%"

# Vẽ khung chữ nhật màu xanh lá (BGR: 0, 255, 0)

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0),3)

# Vẽ nền cho chữ để dễ đọc

cv2.rectangle(frame, (x, y-30), (x+80, y), (0, 255, 0), -1)

# Viết chữ trắng lên nền xanh

cv2.putText(frame, label, (x, y-10), font, 0.8, (255, 255, 255), 2)

return frame, count

# -----------------------------------------------------------

# GIAO DIỆN NGƯỜI DÙNG (GUI)

# -----------------------------------------------------------

def select\_file():

# Mở hộp thoại chọn file

file\_path = filedialog.askopenfilename()

if len(file\_path) > 0:

# Cập nhật trạng thái

lbl\_status.config(text=f"Dang xu ly: {os.path.basename(file\_path)}...", fg="blue")

root.update()

try:

# Gọi hàm nhận diện

result = detect\_image(file\_path)

if result is None or result[0] is None:

lbl\_status.config(text="Loi: Khong doc duoc anh!", fg="red")

return

result\_img, count = result

# Chuyển đổi màu từ BGR (OpenCV) sang RGB (Pillow) để hiển thị

result\_img = cv2.cvtColor(result\_img, cv2.COLOR\_BGR2RGB)

img\_pil = Image.fromarray(result\_img)

# Thay đổi kích thước ảnh cho vừa khung hiển thị (giữ tỉ lệ)

base\_width = 800

w\_percent = (base\_width / float(img\_pil.size[0]))

h\_size = int((float(img\_pil.size[1]) \* float(w\_percent)))

img\_pil = img\_pil.resize((base\_width, h\_size), Image.Resampling.LANCZOS)

img\_tk = ImageTk.PhotoImage(img\_pil)

panel.configure(image=img\_tk)

panel.image = img\_tk

# Thông báo kết quả

if count > 0:

lbl\_status.config(text=f"Thanh cong! Tim thay {count} bien so.", fg="green")

else:

lbl\_status.config(text="Khong tim thay bien so nao trong anh.", fg="orange")

except Exception as e:

lbl\_status.config(text=f"Loi he thong: {e}", fg="red")

print(e)

# Thiết lập cửa sổ chính

root = tk.Tk()

root.title("Phan Mem Nhan Dien Bien So Xe - YOLOv4-tiny")

root.geometry("900x750")

# Tiêu đề

lbl\_title = Label(root, text="HE THONG NHAN DIEN BIEN SO XE", font=("Arial", 22, "bold"), fg="#0066cc")

lbl\_title.pack(pady=15)

# Nút chọn ảnh

btn\_select = Button(root, text="CHON ANH", command=select\_file, font=("Arial", 14, "bold"), bg="#28a745", fg="white", padx=20, pady=10)

btn\_select.pack(pady=10)

# Dòng trạng thái

lbl\_status = Label(root, text="San sang...", font=("Arial", 12))

lbl\_status.pack()

# Khung hiển thị ảnh

panel = Label(root, bg="#f0f0f0")

panel.pack(padx=10, pady=10, expand=True)

# Vòng lặp chính của ứng dụng

root.mainloop()

## **PHỤ LỤC 3: HƯỚNG DẪN CÀI ĐẶT VÀ SỬ DỤNG**

**1. Link Repository (GitHub)** Toàn bộ mã nguồn dự án, bao gồm mã huấn luyện và ứng dụng demo, được lưu trữ công khai tại:

<https://github.com/dinhxuanhiep/dinhxuanhiep>

**2. Yêu cầu hệ thống** Để chạy ứng dụng demo trên máy tính cá nhân, cần đáp ứng các yêu cầu sau:

* **Hệ điều hành:** Windows 10/11, macOS hoặc Linux.
* **Ngôn ngữ lập trình:** Python 3.8 trở lên (Khuyến nghị Python 3.12).
* **Các thư viện Python cần thiết:** OpenCV (opencv-python), NumPy, Pillow (PIL), Tkinter (tích hợp sẵn trong Python).

**3. Quy trình cài đặt**

**Bước 1: Tải mã nguồn và Dữ liệu mô hình** Tải thư mục dự án về máy tính. Đảm bảo đã tải đủ các file trọng số (.weights) và cấu hình (.cfg) từ quá trình huấn luyện trên Google Drive về máy.

**Bước 2: Cài đặt thư viện** Mở **Command Prompt (CMD)** hoặc **Terminal** tại thư mục dự án và chạy lệnh sau để cài đặt các gói phụ thuộc:

pip install opencv-python numpy pillow

**Bước 3: Kiểm tra cấu trúc thư mục** Để ứng dụng hoạt động chính xác, các tệp tin trong thư mục dự án phải được sắp xếp theo đúng cấu trúc sau (đặc biệt lưu ý tên file cấu hình và trọng số):

/LicensePlateApp/

│── app.py # File mã nguồn chính (Giao diện Tkinter)

│── yolov4-tiny-obj.cfg # File cấu hình mạng nơ-ron (Tải từ Colab)

│── yolov4-tiny-obj\_last.weights # File trọng số đã huấn luyện (Tải từ Drive)

│── obj.names # File chứa tên nhãn (Nội dung: license\_plate)

│── test\_images/ # (Tùy chọn) Thư mục chứa các ảnh xe để test

│ └── xe\_01.jpg

**Bước 4: Khởi chạy ứng dụng** Tại giao diện dòng lệnh (CMD) trong thư mục dự án, chạy lệnh sau để bật phần mềm:

python app.py

**Bước 5: Hướng dẫn sử dụng**

1. Giao diện phần mềm **"HỆ THỐNG NHẬN DIỆN BIỂN SỐ XE"** sẽ hiện ra.
2. Nhấn vào nút **"📂 CHỌN ẢNH NGAY"**.
3. Cửa sổ chọn file hiện ra, tìm và chọn một bức ảnh xe máy hoặc ô tô (định dạng .jpg, .png).
4. Hệ thống sẽ tự động xử lý và hiển thị kết quả:
   * **Hình ảnh:** Vẽ khung màu xanh bao quanh biển số.
   * **Nhãn:** Hiển thị độ tin cậy (Confidence score) cạnh khung bao.
   * **Trạng thái:** Thông báo số lượng biển số tìm thấy ở dòng trạng thái phía dưới.

**Đà Nẵng, ngày…tháng…năm 2025**

**Giảng viên hướng dẫn Sinh viên thực hiện**