TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: HỌC MÁY

ĐỀ TÀI: NGHIÊN CỨU VÀ XÂY DỰNG HỆ THỐNG NHẬN DIỆN BIỂN SỐ XE VIỆT NAM

Giảng viên hướng dẫn: ThS. Phạm Việt Anh

Sinh viên thực hiện: Nguyễn Thảo Linh – 2022601125

Nguyễn Mạnh Hoàn – 2022600679

Nguyễn Trung Hiếu – 2022600419

Nguyễn Đình Huỳnh – 2022603411

Trần Huy Quang – 2022601715

Vũ Xuân Oanh – 2022606209

Nhóm: 6

Lớp – Khóa: 20242IT6047001 – K17

Hà Nội, năm 2025

LỜI CẢM ƠN

Trong quá trình thực hiện đề tài "Nghiên cứu và xây dựng hệ thống nhận diện biển số xe Việt Nam", nhóm sinh viên đã nhận được sự quan tâm, hỗ trợ và hướng dẫn tận tình từ các thầy cô trong khoa, đặc biệt là Thạc sĩ Phạm Việt Anh – giảng viên phụ trách môn Học máy.

Nhóm xin chân thành cảm ơn thầy vì đã tạo điều kiện, định hướng phương pháp và góp ý chuyên môn để nhóm có thể hoàn thiện đề tài đúng yêu cầu và đúng tiến độ. Đồng thời, nhóm cũng xin gửi lời cảm ơn đến nhà trường và Trường Công nghệ Thông tin và Truyền thông – Trường Đại học Công nghiệp Hà Nội đã cung cấp môi trường học tập và nghiên cứu thuận lợi.

Mặc dù đã nỗ lực thực hiện với tinh thần trách nhiệm cao, nhưng do hạn chế về mặt thời gian và kinh nghiệm nên báo cáo không tránh khỏi thiếu sót. Nhóm rất mong nhận được sự góp ý của thầy cô để hoàn thiện hơn trong các nghiên cứu sau này.

Nhóm sinh viên thực hiện

Nhóm 6

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC KÝ HIỆU, THUẬT NGỮ, TỪ VIẾT TẮT

STT	Ký hiệu, từ tắt	Tên tiếng anh	Tên tiếng việt
1	AI	Artificial Intelligence	Trí tuệ nhân tạo
2	ML	Machine Learning	Học máy
3	Deep learning	Deep Learning	Học sâu
4	IoT	Internet of Things	Internet vạn vật
5	OCR	Optical Character Recognition	Nhận diện ký tự quang học
6	YOLO	You Only Look Once	Bạn chỉ nhìn một lần
7	CNN	Convolutional Neural Network	Mạng nơ-ron tích chập
8	CRNN	Convolutional Recurrent Neural Network	Mạng nơ-ron tích chập hồi quy
9	supervised learning	Supervised Learning	Học có giám sát
10	unsupervised learning	Unsupervised Learning	Học không giám sát
11	reinforcement learning	Reinforcement Learning	Học tăng cường
12	GPU	Graphics Processing Unit	Bộ xử lý đồ họa
13	perceptron	Perceptron	Mạch nhận thức
14	backpropagation	Backpropagation	Lan truyền ngược
15	Pattern Recognition	Pattern Recognition	Nhận diện mẫu
16	SVM	Support Vector Machine	Máy vectơ hỗ trợ
17	ALPR	Automatic License Plate Recognition	Nhận diện biển số tự động

18	camera	Camera	Máy ảnh
19	Input	Input	Đầu vào
20	Output	Output	Đầu ra
21	TP.HCM	Ho Chi Minh City	Thành phố Hồ Chí Minh
22	Faster R-CNN	Faster Region-based CNN	Mạng CNN tăng tốc vùng
23	RPN	Region Proposal Network	Mạng đề xuất vùng
24	RoI	Regions of Interest	Vùng quan tâm
25	bounding boxes	Bounding Boxes	Hộp giới hạn
26	FPS	Frames Per Second	Khung hình trên giây
27	IoU	Intersection over Union	Giao trên hợp
28	SSD	Single Shot MultiBox Detector	Bộ phát hiện đa hộp một lần
29	default boxes	Default Boxes	Hộp mặc định
30	LSTM	Long Short-Term Memory	Bộ nhớ dài hạn ngắn hạn
31	CTC	Connectionist Temporal Classification	Phân loại theo thời gian
32	classification	Classification	Phân loại
33	regression	Regression	Hồi quy
34	hyperplane	Hyperplane	Siêu phẳng
35	margin	Margin	Biên
36	Maximum Margin Hyperplane	Maximum Margin Hyperplane	Siêu phẳng biên tối đa
37	2D, 3D	Two-Dimensional, Three-Dimensional	Hai chiều, ba chiều
38	Kernel Trick	Kernel Trick	Mẹo nhân hạt

39	Linear, Polynomial, RBF, Sigmoid	Linear, Polynomial, Radial Basis Function, Sigmoid	Tuyến tính, đa thức, hàm cơ sở hướng tâm, sigmoid
40	outliers	Outliers	Quan sát ngoại lai
41	feature	Feature	Đặc trưng
42	Convolution Layer	Convolution Layer	Lớp tích chập
43	Activation Function	Activation Function	Hàm kích hoạt
44	FC	Fully Connected	Kết nối đầy đủ
45	pixel	Pixel	Điểm ảnh
46	stride	Stride	Bước nhảy
47	convolution	Convolution	Tích chập
48	ReLU	Rectified Linear Unit	Đơn vị tuyến tính đã chuẩn hóa
49	pooling	Pooling	Gộp
50	Softmax	Softmax	Hàm Softmax
51	confidence score	Confidence Score	Điểm tin cậy
52	NMS	Non-Maximum Suppression	Loại bỏ không cực đại
53	HOG	Histogram of Oriented Gradients	Biểu đồ gradient có hướng
54	Random Forest	Random Forest	Rừng ngẫu nhiên
55	k NN	k-Nearest Neighbors	k láng giềng gần nhất
56	cross validation	Cross Validation	Kiểm định chéo
57	confusion matrix	Confusion Matrix	Ma trận nhầm lẫn
58	dataset	Dataset	Tập dữ liệu

59	BSD	Biển số dài	Biển số dài
60	BSV	Biển số vuông	Biển số vuông
61	annotation format	Annotation Format	Định dạng chú thích
62	.jpg và .png	.jpg và .png	Định dạng ảnh jpg và png
63	grayscale	Grayscale	Ảnh xám
64	labels	Labels	Nhãn
65	train set	Training Set	Tập huấn luyện
66	test set	Test Set	Tập kiểm thử
67	models	Models	Mô hình
68	load pretrained weights	Load Pretrained Weights	Tải trọng số đã huấn luyện sẵn
69	hyperparameters	Hyperparameters	Siêu tham số
70	ms	Milliseconds	Mi-li giây
71	visualization	Visualization	Hiển thị dữ liệu
72	metrics	Metrics	Các chỉ số đánh giá
73	overfitting	Overfitting	Quá khớp
74	underfitting	Underfitting	Quá thưa
75	mAP@x	mean Average Precision at x	Điểm trung bình độ chính xác tại x
76	Precision-Recall (PR)	Precision-Recall	Độ chính xác - Hồi gọi
77	Precision Confidence	Precision Confidence	Độ chính xác (tin cậy)
78	Recall Confidence	Recall Confidence	Hồi gọi (tin cậy)
79	F1-Confidence	F1-Confidence	Độ tin cậy F1
80	Accuracy	Accuracy	Độ chính xác

81	validation	Validation	Xác thực
82	epoch	Epoch	Vòng lặp
83	morphology	Morphology	Phép hình thái

MỞ ĐẦU

1. Lý do chọn đề tài

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, việc áp dụng trí tuệ nhân tạo (AI) và học máy (Machine Learning) vào các lĩnh vực thực tiễn đang trở thành xu hướng tất yếu, đặc biệt trong lĩnh vực giao thông và an ninh. Hệ thống nhận diện biển số xe là một ví dụ điển hình, đóng vai trò quan trọng trong việc giám sát, quản lý phương tiện, giảm thiểu nhân lực và nâng cao hiệu quả vận hành ở các khu vực như bãi đỗ xe, trạm thu phí không dừng, cổng ra vào tòa nhà, khu công nghiệp, khu dân cư, ...

Tuy nhiên, các hệ thống hiện có tại Việt Nam vẫn còn nhiều hạn chế như chi phí đầu tư cao, tính tương thích với biển số xe trong nước chưa tối ưu, và độ chính xác chưa ổn định trong các điều kiện thực tế như thời tiết xấu, ánh sáng yếu, biển số bị mờ, lệch góc hoặc bị che khuất. Chính vì vậy, nhóm em quyết định chọn đề tài "Nghiên cứu và xây dựng hệ thống nhận diện biển số xe Việt Nam" với mong muốn tạo ra một giải pháp vừa hiệu quả, vừa khả thi về mặt kỹ thuật và chi phí, đồng thời cũng là cơ hội để vận dụng các kiến thức đã học vào một bài toán mang tính ứng dụng thực tế cao.

Dưới sự hướng dẫn tận tình của Thạc sĩ Phạm Việt Anh, nhóm em đã có thêm định hướng rõ ràng, tiếp cận đúng đắn với các kiến thức chuyên ngành, từ đó triển khai đề tài một cách bài bản, khoa học và có chiều sâu.

2. Mục tiêu nghiên cứu

Đề tài hướng đến việc xây dựng một hệ thống hoàn chỉnh có khả năng tự động phát hiện, cắt và nhận diện biển số xe từ ảnh hoặc video đầu vào, với độ chính xác cao và thời gian xử lý ngắn. Cụ thể, các mục tiêu chính bao gồm:

- Thiết kế và triển khai một hệ thống có khả năng nhận diện biển số xe trong điều kiện môi trường đa dạng (độ sáng khác nhau, biển số nghiêng, bị mờ...).

- So sánh, đánh giá hiệu quả giữa các phương pháp khác nhau về độ chính xác, tốc độ xử lý và khả năng triển khai thực tế.
- Đề xuất cải tiến hoặc hướng phát triển mở rộng trong tương lai như tích hợp với hệ thống IoT, nhận diện biển số xe máy và xe ô tô riêng biệt, phân loại phương tiện, ...

3. Phương pháp nghiên cứu

Để thực hiện đề tài một cách khoa học và hiệu quả, nhóm áp dụng kết hợp giữa phương pháp nghiên cứu lý thuyết và thực nghiệm như sau:

- Khảo sát tài liệu: Nghiên cứu các tài liệu học thuật, giáo trình, bài báo quốc tế về xử lý ảnh, học sâu (deep learning), nhận diện biển số xe và các mô hình OCR.
- Xử lý dữ liệu: Thu thập và làm sạch dữ liệu ảnh biển số xe, tiền xử lý bằng các kỹ thuật như chuyển ảnh xám, lọc nhiễu, cân bằng sáng, điều chỉnh kích thước, ...
- Xây dựng mô hình: Sử dụng các thư viện Python phổ biến như OpenCV, TensorFlow, Keras để xây dựng mô hình phát hiện biển số (bằng YOLO, Haar Cascade...) và nhận diện ký tự (sử dụng CNN hoặc CRNN).
- Huấn luyện và kiểm thử: Đưa dữ liệu đã xử lý vào huấn luyện mô hình, đánh giá bằng các chỉ số như độ chính xác (accuracy), độ nhạy (recall), độ đặc hiệu (precision).
- Thực nghiệm và đánh giá: Kiểm thử mô hình với dữ liệu thực tế, từ đó điều chỉnh tham số, cải thiện độ chính xác và tốc độ nhận diện.

CHƯƠNG 1: TỔNG QUAN

1.1. Giới thiệu chung về học máy và xử lý ảnh.

Học máy (Machine Learning) là một lĩnh vực cốt lõi của trí tuệ nhân tạo, tập trung vào việc phát triển các thuật toán và mô hình cho phép máy tính học hỏi từ dữ liệu và cải thiện hiệu suất mà không cần được lập trình cụ thể cho từng nhiệm vụ. Theo Tom Mitchell (1997), học máy được định nghĩa là "nghiên cứu về các thuật toán máy tính có thể tự động cải thiện thông qua kinh nghiệm". Khái niệm này đã trở thành nền tảng cho nhiều ứng dụng hiện đại, từ các hệ thống đề xuất trong thương mại điện tử, chẩn đoán y tế, đến các giải pháp tự động hóa trong công nghiệp và giao thông. Học máy cho phép các hệ thống tự động phân tích dữ liệu, trích xuất các mẫu và đưa ra quyết định dựa trên dữ liệu, từ đó giảm thiểu sự phụ thuộc vào các quy tắc được lập trình thủ công [1].

Học máy được chia thành ba loại chính: học có giám sát (supervised learning), học không giám sát (unsupervised learning), và học tăng cường (reinforcement learning). Trong học có giám sát, mô hình được huấn luyện trên dữ liệu có nhãn để dự đoán đầu ra từ đầu vào. Học không giám sát tập trung vào việc tìm kiếm các mẫu hoặc cấu trúc ẩn trong dữ liệu không có nhãn. Học tăng cường liên quan đến việc học cách đưa ra quyết định tối ưu thông qua thử và sai. Những phương pháp này đã được áp dụng rộng rãi trong nhiều lĩnh vực, từ tài chính, y tế, đến giao thông và an ninh, nhờ vào khả năng xử lý dữ liệu phức tạp và đưa ra các dự đoán chính xác.

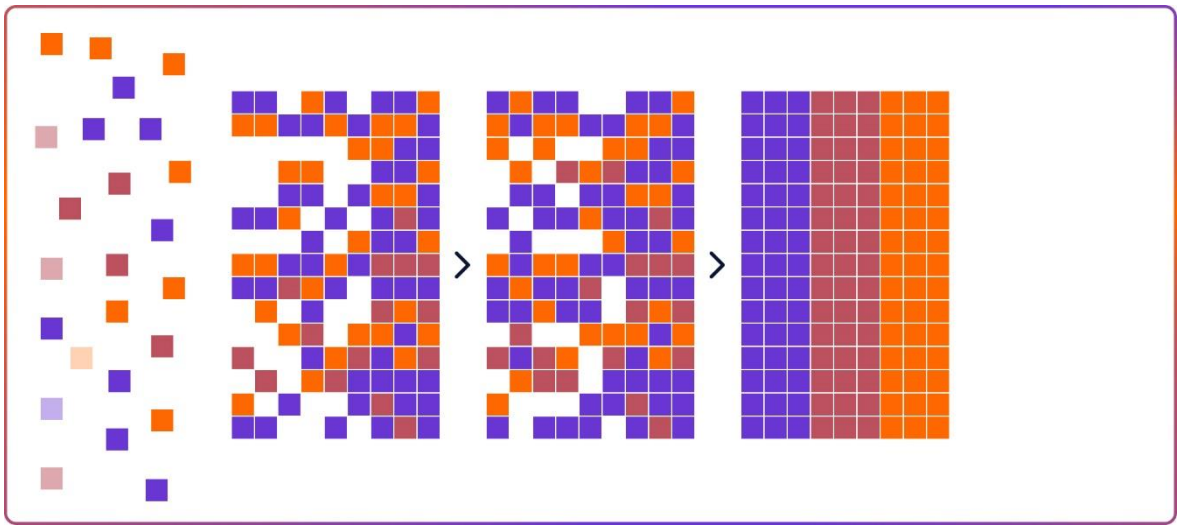
Sự phát triển của học máy đã có những bước tiến vượt bậc trong những thập kỷ gần đây, đặc biệt nhờ vào ba yếu tố chính: sự gia tăng của dữ liệu lớn, sức mạnh tính toán được cải thiện thông qua các đơn vị xử lý đồ họa (GPU), và những đột phá trong các thuật toán, đặc biệt là học sâu (Deep Learning). Những năm 1950 chứng kiến sự ra đời của các mô hình đơn giản như perceptron, nhưng phải đến những năm 1980, với sự phát triển của thuật toán lan truyền ngược (backpropagation), học máy mới bắt đầu cho thấy tiềm năng thực sự.

Tuy nhiên, bước ngoặt lớn nhất đến vào năm 2012, khi mô hình AlexNet của Krizhevsky et al. (2012) đạt được độ chính xác vượt trội trong bài toán phân loại hình ảnh trên tập dữ liệu ImageNet [2]. Sự thành công này đã đánh dấu sự khởi đầu của kỷ nguyên học sâu, với các mạng nơ-ron tích chập (CNN) và các kiến trúc phức tạp hơn như Transformer trở thành công cụ mạnh mẽ trong nhiều ứng dụng.

Sự sẵn có của các tập dữ liệu lớn như ImageNet, cùng với sự gia tăng sức mạnh tính toán từ GPU và các nền tảng đám mây như Google Cloud hay AWS, đã thúc đẩy học máy trở thành một công cụ không thể thiếu. Các công ty công nghệ lớn như Google, Meta, và Amazon đã đầu tư mạnh mẽ vào học máy, dẫn đến sự phát triển của các thư viện mã nguồn mở như TensorFlow và PyTorch, giúp đơn giản hóa việc triển khai các mô hình học máy. Những tiến bộ này đã mở rộng phạm vi ứng dụng của học máy, từ các hệ thống đề xuất, nhận diện giọng nói, đến các ứng dụng trong giao thông thông minh và an ninh.

1.2. Tổng quan về bài toán nhận diện.

Nhận diện mẫu (Pattern Recognition) là một lĩnh vực con quan trọng của học máy, tập trung vào việc nhận diện các mẫu và quy luật trong dữ liệu. Theo Duda (2001), nhận diện mẫu là "hành động nhận dữ liệu thô và thực hiện một hành động dựa trên loại của mẫu" [3]. Trong bối cảnh xử lý hình ảnh, nhận diện mẫu liên quan đến việc trích xuất các đặc trưng từ hình ảnh, chẳng hạn như cạnh, màu sắc, hoặc hình dạng, và phân loại chúng vào các danh mục cụ thể. Các kỹ thuật nhận diện mẫu thường sử dụng các mô hình học máy như mạng nơ-ron tích chập (CNN), máy vector hỗ trợ (SVM), hoặc các phương pháp học sâu để đạt được độ chính xác cao.



Hình 1.1. Nhận diện mẫu [4]

Các ứng dụng này cho thấy tiềm năng to lớn của nhận diện mẫu trong việc tự động hóa các nhiệm vụ phức tạp, cải thiện hiệu quả và độ chính xác trong các hệ thống thực tế. Đặc biệt, trong bối cảnh giao thông thông minh, nhận diện mẫu đóng vai trò quan trọng trong việc xử lý các bài toán liên quan đến hình ảnh, chẳng hạn như nhận diện các đối tượng hoặc ký tự trong môi trường thực tế với nhiều thách thức như ánh sáng thay đổi, góc nhìn đa dạng, hoặc nhiễu môi trường.

1.3. Học máy và bài toán nhận diện biển số xe.

1.3.1. Giới thiệu bài toán.

Nhận diện biển số xe tự động (Automatic License Plate Recognition - ALPR) là một công nghệ quan trọng trong lĩnh vực thị giác máy tính, sử dụng các kỹ thuật xử lý ảnh và nhận diện mẫu để tự động đọc biển số xe từ hình ảnh hoặc video mà không cần sự can thiệp của con người [7]. Công nghệ này đóng vai trò thiết yếu trong các hệ thống giao thông thông minh, hỗ trợ các ứng dụng như quản lý giao thông, thực thi pháp luật, thu phí cầu đường, và quản lý bãi đỗ xe. Với sự gia tăng số lượng phương tiện giao thông tại Việt Nam, ALPR mang lại tiềm năng lớn trong việc cải thiện hiệu quả giám sát giao thông, tăng cường an ninh, và tối ưu hóa các dịch vụ liên quan đến phương tiện [9].

Bài toán nhận diện biển số xe từ ảnh hoặc video là một bài toán thị giác máy tính phổ biến, với mục tiêu xác định và trích xuất chính xác chuỗi ký tự trên biển số xe. Theo Lubna, ALPR là một quá trình phức hợp, yêu cầu xử lý các hình ảnh đầu vào trong các điều kiện thực tế đa dạng, từ đó tạo ra đầu ra là chuỗi ký tự biểu thị biển số. Bài toán này không chỉ đòi hỏi độ chính xác cao mà còn cần tốc độ xử lý nhanh để đáp ứng các ứng dụng thời gian thực, chẳng hạn như giám sát giao thông hoặc thu phí tự động [7].

1.3.2. Đặc điểm bài toán nhận diện biển số xe.

Trong các hệ thống truyền thống, nhận diện biển số xe thường dựa vào các phương pháp xử lý ảnh thuần túy như lọc màu, phát hiện cạnh, phân ngưỡng và tách vùng ký tự bằng các quy tắc thủ công. Tuy nhiên, những phương pháp này gặp nhiều hạn chế khi xử lý ảnh trong môi trường thực tế – nơi mà ánh sáng, góc chụp, chất lượng ảnh, sự che khuất hoặc biến dạng của biển số có thể ảnh hưởng nghiêm trọng đến độ chính xác.

Học máy (Machine Learning), đặc biệt là học sâu (Deep Learning), mở ra một hướng tiếp cận mới mang tính đột phá cho bài toán nhận diện biển số xe. Thay vì lập trình từng quy tắc cụ thể, mô hình học máy có khả năng tự học từ dữ liệu, phát hiện các đặc trưng quan trọng và đưa ra quyết định nhận diện với độ chính xác cao hơn nhiều so với phương pháp truyền thống.

Một số lý do cụ thể để sử dụng học máy trong bài toán này bao gồm:

- Khả năng tổng quát hóa tốt hơn: Mô hình học máy có thể xử lý hiệu quả các trường hợp phức tạp như biển số bị nghiêng, ánh sáng yếu, bị bẩn hoặc mờ vốn là thách thức lớn với các thuật toán xử lý ảnh truyền thống.
- Tự động hóa hoàn toàn quá trình nhận diện: Nhờ học từ tập dữ liệu lớn, mô hình có thể phát hiện và phân loại biển số mà không cần lập trình cụ thể cho từng tình huống.

- Tính thích nghi cao: Hệ thống có thể dễ dàng cập nhật, mở rộng hoặc tinh chỉnh khi có thêm dữ liệu mới, giúp nâng cao hiệu suất theo thời gian mà không cần viết lại toàn bộ mã nguồn.
- Khả năng kết hợp với các công nghệ hiện đại: Mô hình học máy có thể tích hợp dễ dàng với các hệ thống camera giám sát, IoT hoặc hệ thống giao thông thông minh để tạo thành một chuỗi ứng dụng liên kết.

Việc ứng dụng học máy vào nhận diện biển số xe không chỉ mang lại độ chính xác và hiệu quả cao mà còn giúp rút ngắn thời gian xử lý, giảm thiểu sai sót và nâng cao tính khả thi trong triển khai thực tế. Chính vì vậy, trong đề tài này, nhóm quyết định lựa chọn học máy làm phương pháp cốt lõi để xây dựng hệ thống nhận diện biển số xe thông minh và hiệu quả hơn.

1.3.3. Mô tả chi tiết đầu vào và đầu ra của bài toán

Đầu vào (Input)

Đầu vào của bài toán là các hình ảnh tĩnh hoặc khung hình trích xuất từ video có chứa phương tiện giao thông. Dữ liệu hình ảnh này thường được thu thập từ nhiều nguồn khác nhau như camera giao thông cố định, camera giám sát an ninh, camera hành trình trên xe, thiết bị di động, hoặc hệ thống camera tại các trạm kiểm soát. Mục tiêu là phát hiện chính xác vị trí và loại biển số xe xuất hiện trong ảnh.

Đầu ra (Output):

Đầu ra mà hệ thống cần tạo ra là thông tin về biển số xe trong ảnh, bao gồm vị trí của biển số và nội dung của biển số. Tức là hệ thống phải xác định được vùng nào trong ảnh là biển số, khoanh lại bằng một khung chữ nhật. Nếu trong ảnh có nhiều xe, thì có thể có nhiều biển số và hệ thống cần phát hiện hết. Sau khi tìm được vị trí, hệ thống cần đọc được chính xác các ký tự trên biển số, Đầu ra cần chính xác và tuân theo định dạng biển số của quốc gia tương ứng, ví dụ :“30A4-12345” cho biển số tại Việt Nam [9].

1.3.4. Những khó khăn và thách thức.

Nhận diện biển số xe đối mặt với nhiều thách thức kỹ thuật, đặc biệt trong các điều kiện thực tế [8]. Một số thách thức chính bao gồm:

- Điều kiện ánh sáng: Ánh sáng thay đổi (ban ngày, ban đêm, thời tiết mưa hoặc sương mù) ảnh hưởng đến chất lượng ảnh, làm giảm khả năng phát hiện và nhận diện biển số [7].
- Góc nhìn đa dạng: Phương tiện có thể được chụp từ các góc độ khác nhau (chính diện, nghiêng, hoặc từ xa), gây khó khăn trong việc định vị chính xác vùng biển số [8].
- Che khuất: Biển số có thể bị che bởi bụi bẩn, vật cản, hoặc hư hỏng, làm giảm độ chính xác của quá trình nhận diện [8].
- Chất lượng ảnh thấp: Ảnh mờ, nhiễu, hoặc có độ phân giải thấp làm giảm khả năng nhận diện ký tự [7].
- Định dạng biển số đa dạng: Mỗi quốc gia có định dạng biển số khác nhau, với các ký tự, màu sắc, và bố cục riêng. Tại Việt Nam, định dạng biển số như DDLL-DDD.DD (D là chữ số, L là chữ cái, bao gồm ký tự đặc biệt 'Đ') và bao gồm 2 loại là biển số dài (dành cho ô tô) và biển số vuông (dành cho xe máy) [9].

1.3.5. Miền ứng dụng của bài toán.

Nhận diện biển số xe có nhiều ứng dụng thực tế, đặc biệt trong các lĩnh vực sau:

- **Quản lý giao thông**: ALPR hỗ trợ giám sát các vi phạm giao thông, như chạy quá tốc độ, vượt đèn đỏ, hoặc không đội mũ bảo hiểm, góp phần nâng cao an toàn giao thông.
- **An ninh**: Hệ thống ALPR giúp xác định các phương tiện liên quan đến tội phạm, tìm kiếm xe bị mất cắp, hoặc giám sát tại các khu vực nhạy cảm như sân bay, cảng biển.

- **Thu phí tự động**: Tại các trạm thu phí, ALPR cho phép tính phí mà không cần dừng xe, cải thiện hiệu quả và giảm ùn tắc.
- **Quản lý bãi đỗ xe**: ALPR tự động ghi nhận thời gian ra vào của phương tiện, hỗ trợ quản lý bãi đỗ xe thông minh, đặc biệt tại các thành phố lớn như Hà Nội và TP.HCM.

CHƯƠNG 2: PHƯƠNG PHÁP VÀ QUY TRÌNH THỰC HIỆN

2.1. Các phương pháp giải quyết bài toán nhận diện.

2.1.1. Mô hình Faster R-CNN.

Faster R-CNN, được giới thiệu bởi Shaoqing Ren (2015), là một mô hình phát hiện đối tượng dựa trên vùng, cải tiến từ các phiên bản trước như R-CNN và Fast R-CNN. Mô hình này sử dụng mạng đề xuất vùng (Region Proposal Network - RPN) để tạo ra các vùng quan tâm (regions of interest - RoI) từ hình ảnh đầu vào, sau đó phân loại và tinh chỉnh các hộp giới hạn (bounding boxes) để xác định vị trí và lớp của đối tượng. Trong ALPR, Faster R-CNN có thể được sử dụng để phát hiện biển số với độ chính xác cao, đặc biệt trong các điều kiện phức tạp như ánh sáng yếu, góc nhìn nghiêng, hoặc nhiễu nền.

Mô hình này tận dụng kiến trúc mạng nơ-ron tích chập (CNN) như VGG16 hoặc ResNet để trích xuất đặc trưng, sau đó áp dụng RPN để đề xuất các vùng tiềm năng chứa biển số. Tuy nhiên, do yêu cầu xử lý hai giai đoạn (đề xuất vùng và phân loại), Faster R-CNN có tốc độ chậm hơn so với các mô hình một lần quét như YOLO hoặc SSD, thường chỉ đạt 5-7 FPS trên GPU mạnh, khiến nó ít phù hợp cho các ứng dụng thời gian thực như giám sát giao thông hoặc thu phí tự động Lubna et al., 2021. Ngoài ra, Faster R-CNN yêu cầu tài nguyên tính toán lớn và quá trình huấn luyện phức tạp hơn, đòi hỏi tinh chỉnh nhiều siêu tham số như số lượng vùng đề xuất và ngưỡng IoU (Intersection over Union).

2.1.2. Mô hình SSD.

SSD (Single Shot MultiBox Detector), được đề xuất bởi Wei Liu (2016), là một mô hình phát hiện đối tượng một lần quét, tương tự YOLO, nhưng sử dụng các hộp mặc định (default boxes) ở nhiều tỷ lệ và kích thước để dự đoán vị trí và lớp của đối tượng trong một lần xử lý. SSD tận dụng các lớp tích chập ở các độ phân giải khác nhau để phát hiện đối tượng ở nhiều tỷ lệ, từ các đối tượng lớn đến các đối tượng nhỏ. Trong ALPR, SSD có thể được sử dụng để phát hiện biển số với tốc độ nhanh, thường đạt 20-30 FPS trên GPU, phù hợp

cho các ứng dụng thời gian thực như giám sát giao thông. Tuy nhiên, SSD thường kém chính xác hơn YOLO trong các trường hợp có nhiều đối tượng nhỏ hoặc nhiễu nền phức tạp, như trong các cảnh giao thông đông đúc tại Việt Nam, do không xử lý tốt các đối tượng nhỏ hoặc bị che khuất Lubna (2021). SSD cũng yêu cầu tinh chỉnh các hộp mặc định và siêu tham số để đạt hiệu suất tối ưu, nhưng việc triển khai đơn giản hơn so với Faster R-CNN nhờ kiến trúc một lần quét. Mặc dù vậy, trong các điều kiện thực tế phức tạp, SSD có thể không đạt được độ chính xác cao như YOLO, đặc biệt khi xử lý các biển số nhỏ hoặc bị biến dạng.

2.1.3. Mô hình CRNN.

CRNN (Convolutional Recurrent Neural Network), được giới thiệu bởi Shiyu., 2017, là một mô hình kết hợp mạng nơ-ron tích chập (CNN) và mạng nơ-ron hồi quy (RNN) để nhận diện chuỗi ký tự end-to-end. CRNN sử dụng CNN để trích xuất đặc trưng từ hình ảnh, sau đó truyền các đặc trưng này qua RNN (thường là LSTM - Long Short-Term Memory) để mô hình hóa chuỗi ký tự, và cuối cùng sử dụng hàm mất mát CTC (Connectionist Temporal Classification) để dự đoán chuỗi ký tự mà không cần phân đoạn trước. Trong ALPR, CRNN đặc biệt hữu ích ở giai đoạn nhận diện ký tự, khi cần đọc toàn bộ chuỗi ký tự trên biển số (ví dụ: “51AB-123.45”) mà không cần tách từng ký tự riêng lẻ. CRNN có khả năng xử lý các chuỗi ký tự có độ dài thay đổi, như định dạng biển số Việt Nam, và đặc biệt hiệu quả khi biển số có bố cục phức tạp, như biển số hai dòng của xe máy.

Tuy nhiên, CRNN không phù hợp cho bước phát hiện biển số, vì nó được thiết kế để nhận diện chuỗi ký tự chứ không phải xác định vị trí đối tượng trong hình ảnh. Do đó, CRNN thường được sử dụng kết hợp với một mô hình phát hiện riêng biệt, như YOLO hoặc SSD, để tạo thành một hệ thống ALPR hoàn chỉnh. Mặc dù hiệu quả trong nhận diện ký tự, CRNN yêu cầu dữ liệu huấn

luyện lớn và có thể phức tạp trong việc tinh chỉnh để xử lý các ký tự đặc biệt như ‘Đ’ trong biển số Việt Nam.

2.1.4. Máy vector hỗ trợ (Support Vector Machine).

Máy Vectơ Hỗ trợ (Support Vector Machine - SVM) là một thuật toán học máy có giám sát mạnh mẽ, được sử dụng rộng rãi cho cả bài toán phân loại (classification) và hồi quy (regression). Tuy nhiên, SVM được biết đến và áp dụng chủ yếu cho các bài toán phân loại, đặc biệt hiệu quả khi xử lý dữ liệu phức tạp và có nhiều chiều.

Ý tưởng cốt lõi của SVM

Mục tiêu chính của SVM là tìm một siêu phẳng (hyperplane) tốt nhất để phân tách các lớp dữ liệu trong một không gian đa chiều. "Tốt nhất" ở đây có nghĩa là siêu phẳng đó phải có biên độ (margin) lớn nhất giữa các lớp.

Hãy hình dung dữ liệu của bạn là các điểm trong không gian. Nếu bạn có hai loại trái cây (ví dụ: táo và cam) được biểu diễn bằng các đặc trưng (màu sắc, độ tròn), SVM sẽ tìm một "đường phân chia" (trong không gian 2D) hoặc "mặt phẳng phân chia" (trong không gian 3D, gọi là siêu phẳng trong không gian nhiều chiều hơn) sao cho hai loại trái cây này được tách biệt rõ ràng nhất có thể.

Siêu phẳng và Biên độ tối đa (Maximum Margin Hyperplane)

Siêu phẳng (Hyperplane): Là một đường thẳng (trong không gian 2D), một mặt phẳng (trong không gian 3D), hoặc một không gian con phẳng (trong không gian nhiều chiều hơn) có tác dụng phân chia các điểm dữ liệu thuộc các lớp khác nhau.

Biên độ (Margin): Là khoảng cách giữa siêu phẳng phân chia và các điểm dữ liệu gần siêu phẳng nhất của mỗi lớp. Các điểm dữ liệu này được gọi là vectơ hỗ trợ (Support Vectors).

Mục tiêu của SVM: Tìm siêu phẳng sao cho biên độ này là lớn nhất. Việc tối đa hóa biên độ giúp mô hình có khả năng khái quát hóa tốt hơn, tức là nó sẽ

phân loại chính xác các dữ liệu mới mà nó chưa từng thấy trong quá trình huấn luyện.

Vectơ hỗ trợ (Support Vectors)

Các vectơ hỗ trợ là các điểm dữ liệu nằm gần siêu phẳng phân chia nhất và đóng vai trò quan trọng trong việc định hình siêu phẳng đó. Nếu loại bỏ một vectơ hỗ trợ, vị trí của siêu phẳng có thể thay đổi. Điều này cho thấy SVM chỉ phụ thuộc vào một phần nhỏ dữ liệu (các vectơ hỗ trợ) chứ không phải toàn bộ dữ liệu, giúp nó hiệu quả và mạnh mẽ.

Xử lý dữ liệu không phân tách tuyến tính và Kernel Trick

Trong nhiều trường hợp thực tế, dữ liệu không thể được phân tách một cách tuyến tính bằng một đường thẳng hoặc mặt phẳng đơn giản. Để giải quyết vấn đề này, SVM sử dụng một kỹ thuật mạnh mẽ gọi là Kernel Trick (phép biến đổi hạt nhân).

Ý tưởng: Thay vì tìm một siêu phẳng trong không gian ban đầu, Kernel Trick ánh xạ dữ liệu từ không gian ban đầu có chiều thấp lên một không gian có chiều cao hơn. Trong không gian chiều cao này, các điểm dữ liệu có thể trở nên dễ phân tách tuyến tính hơn.

- Hàm Kernel: Là các hàm toán học (ví dụ: Linear, Polynomial, Radial Basis Function (RBF) hay Sigmoid) dùng để tính toán "tích vô hướng" giữa các điểm dữ liệu trong không gian chiều cao mà không cần phải thực sự ánh xạ rõ ràng từng điểm dữ liệu lên không gian đó. Điều này giúp giảm đáng kể chi phí tính toán.
- Kernel tuyến tính (Linear Kernel): Dành cho dữ liệu có thể tách biệt tuyến tính.
- Kernel đa thức (Polynomial Kernel): Ánh xạ dữ liệu lên các không gian chiều cao hơn bằng các hàm đa thức.
- Kernel hàm cơ sở xuyên tâm (Radial Basis Function - RBF Kernel): Một lựa chọn rất phổ biến cho các mối quan hệ phi tuyến tính phức tạp.

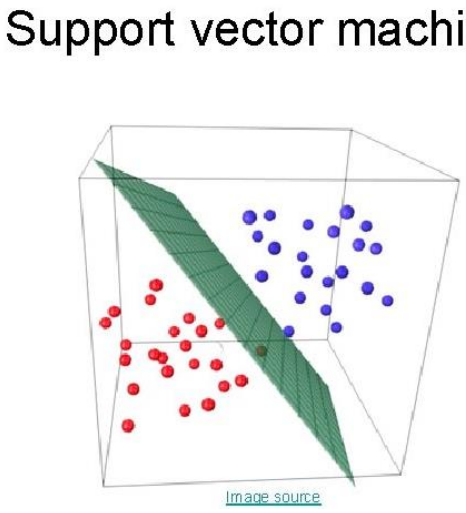
- Kernel Sigmoid: Tương tự như hàm kích hoạt trong mạng nơ-ron.

Ưu điểm của SVM

- Hiệu quả với không gian chiều cao: Hoạt động tốt trong các không gian có số chiều lớn, làm cho nó phù hợp với các bài toán có nhiều đặc trưng.
- Hiệu quả với tập dữ liệu nhỏ: Khá hiệu quả ngay cả khi số lượng mẫu huấn luyện không quá lớn.
- Kiểm soát quá khớp: Việc tối đa hóa biên độ giúp SVM ít bị quá khớp (overfitting) hơn so với một số thuật toán khác.
- Linh hoạt với Kernel: Khả năng sử dụng các hàm Kernel khác nhau cho phép SVM xử lý cả dữ liệu phân tách tuyến tính và phi tuyến tính.

Nhược điểm của SVM

- Tính toán phức tạp: Việc huấn luyện SVM có thể tốn kém về mặt tính toán và thời gian, đặc biệt với các tập dữ liệu rất lớn.
- Nhạy cảm với nhiễu: SVM có thể nhạy cảm với dữ liệu nhiễu (outliers) vì chúng có thể ảnh hưởng đến vị trí của các vectơ hỗ trợ và do đó là siêu phẳng phân chia.
- Giải thích mô hình: Mô hình SVM đôi khi khó giải thích (ít "trong suốt" hơn) so với các mô hình đơn giản hơn như cây quyết định.



Hình 2.1. Mô tả mô hình SVM

Ứng dụng của SVM trong Nhận diện biển số xe

Trong hệ thống nhận diện biển số xe, SVM có thể được sử dụng ở nhiều giai đoạn khác nhau, đặc biệt là trong bước nhận diện ký tự quang học (OCR):

Phân loại ký tự: Sau khi biển số được phát hiện và các ký tự riêng lẻ được phân đoạn, SVM có thể được huấn luyện để phân loại từng hình ảnh ký tự thành các chữ số (0-9) hoặc chữ cái (A-Z).

Phân loại các thành phần hình ảnh: SVM cũng có thể được sử dụng trong các bước tiền xử lý để phân loại các vùng ảnh có phải là ký tự hay không.

Mặc dù trong những năm gần đây, các mô hình học sâu (Deep Learning) như CNN đã trở nên vượt trội trong nhiều bài toán nhận diện hình ảnh, SVM vẫn là một công cụ phân loại mạnh mẽ và được sử dụng rộng rãi, đặc biệt là khi dữ liệu huấn luyện hạn chế hoặc khi cần một mô hình với khả năng giải thích tốt hơn cho một số tác vụ phụ.

2.1.5. Mạng nơ-ron tích chập (Convolutional neural network – CNN).

Mạng nơ-ron tích chập (CNN - Convolutional Neural Network) là một loại mô hình học sâu (Deep Learning) được thiết kế đặc biệt để xử lý dữ liệu có dạng lưới, điển hình là hình ảnh. CNN mô phỏng hoạt động của thị giác sinh học – nơi các tế bào thần kinh trong não phản ứng với các vùng nhỏ trên trường nhìn, từ đó tạo ra khả năng nhận biết vật thể.

Khác với các mạng nơ-ron truyền thống (Fully Connected Neural Network), CNN khai thác tính chất cục bộ và không gian của hình ảnh để tự động trích xuất đặc trưng (feature) mà không cần phải thiết kế thủ công. CNN đã chứng minh hiệu quả vượt trội trong nhiều bài toán xử lý ảnh như phân loại ảnh, phát hiện đối tượng, nhận diện khuôn mặt, đặc biệt là nhận diện biển số xe.

Cấu trúc tổng quát của CNN

Một mô hình CNN điển hình bao gồm các thành phần chính sau:

1) Lớp tích chập (Convolution Layer):

Đây là lớp cốt lõi trong CNN, sử dụng các bộ lọc (kernel/filter) trượt trên ảnh đầu vào để trích xuất các đặc trưng quan trọng như cạnh, góc, họa tiết, ...

Các bộ lọc này được học trong quá trình huấn luyện, không cần thiết kế thủ công.

2) Lớp phi tuyến (Activation Function – thường là ReLU):

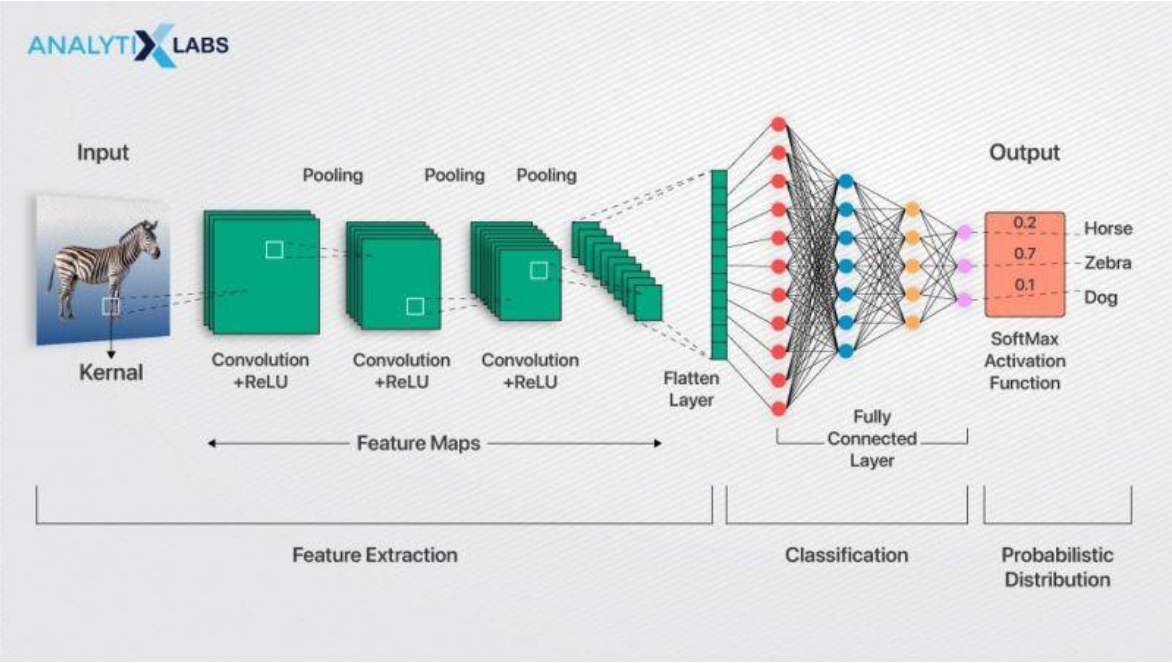
Sau mỗi bước tích chập, hàm kích hoạt như ReLU (Rectified Linear Unit) được sử dụng để đưa mô hình về dạng phi tuyến, giúp mạng học được các mối quan hệ phức tạp hơn.

3) Lớp pooling (giảm kích thước – thường là MaxPooling):

Pooling giúp giảm kích thước dữ liệu và số tham số trong mô hình, đồng thời giữ lại các đặc trưng quan trọng. MaxPooling là kỹ thuật phổ biến, lấy giá trị lớn nhất trong vùng lân cận.

4) Lớp fully connected (FC - kết nối đầy đủ):

Sau nhiều tầng tích chập và pooling, dữ liệu được chuyển thành vector và đưa vào một hoặc nhiều lớp FC để ra quyết định cuối cùng.



Hình 2.2. Cấu trúc mô hình CNN[14]

Nguyên lý hoạt động của CNN

Hoạt động của một CNN có thể được mô tả qua các bước sau:

1) Nhận ảnh đầu vào:

– Ảnh được đưa vào hệ thống dưới dạng ma trận điểm ảnh (pixel). Ví dụ, ảnh kích thước 100x100 sẽ có 10.000 điểm ảnh.

2) Trích xuất đặc trưng qua lớp tích chập:

– Bộ lọc (ví dụ 3x3) trượt qua ảnh theo từng bước (stride), tính toán tích chập (convolution) giữa bộ lọc và vùng ảnh tương ứng.
– Mỗi bộ lọc học ra một loại đặc trưng, tạo nên nhiều bản đồ đặc trưng khác nhau.

3) Phi tuyến hóa bằng ReLU:

– Kết quả từ lớp tích chập sẽ đi qua ReLU, loại bỏ các giá trị âm để tăng khả năng biểu diễn phi tuyến.

4) Giảm chiều qua lớp pooling:

– Ở mỗi bản đồ đặc trưng, thuật toán pooling sẽ chọn giá trị lớn nhất (MaxPooling) trong từng vùng nhỏ (thường 2x2), giảm kích thước và tăng tính khái quát.

5) Lặp lại các bước trên nhiều lần:

– Mô hình có thể gồm nhiều khối convolution + pooling để học từ đặc trưng cơ bản đến đặc trưng trừu tượng hơn.

6) Làm phẳng và phân loại:

– Sau khi qua các tầng trích xuất đặc trưng, dữ liệu được làm phẳng (flatten) thành vector 1 chiều và đưa vào các lớp fully connected.
– Lớp cuối dùng Softmax (hoặc sigmoid) để dự đoán đầu ra. Trong bài toán nhận diện ký tự, mỗi đầu ra tương ứng với một ký tự (0-9, A-Z...).

Ưu điểm của CNN trong bài toán nhận diện biển số xe:

- **Tự động trích xuất đặc trưng**: CNN không yêu cầu thiết kế đặc trưng thủ công, giúp tiết kiệm thời gian và tăng tính linh hoạt với nhiều dạng biển số khác nhau.
- **Khả năng xử lý ảnh mạnh mẽ:** Mô hình có thể học được cả đặc trưng hình dạng, đường viền và độ tương phản – phù hợp với biển số bị nghiêng, mờ hoặc biến dạng.
- **Khả năng kết hợp phát hiện và phân loại:** CNN có thể được tích hợp vào các hệ thống nhận diện toàn diện, từ phát hiện vùng biển số đến nhận diện ký tự trong biển số.
- **Hiệu quả và khả năng mở rộng:** Với sự hỗ trợ của GPU và thư viện như TensorFlow, PyTorch, CNN có thể huấn luyện và triển khai trên quy mô lớn với tốc độ xử lý nhanh.

Ứng dụng CNN trong đề tài

Trong khuôn khổ đề tài, mạng CNN được sử dụng chủ yếu trong hai giai đoạn:

1) Nhận diện vùng biển số xe:
- CNN có thể được kết hợp với các kiến trúc như YOLO (You Only Look Once) hoặc SSD (Single Shot Detector) để phát hiện chính xác vùng chứa biển số trong ảnh.
2) Nhận diện ký tự trong biển số:
- Sau khi cắt riêng từng ký tự, CNN được huấn luyện để phân loại từng ký tự là chữ cái hoặc số tương ứng.

Ví dụ một mô hình CNN đơn giản gồm: 2 lớp tích chập và 1 lớp fully connected cũng có thể đạt được độ chính xác cao nếu có đủ dữ liệu huấn luyện và được tối ưu tốt. Do đó, CNN là một thành phần then chốt trong các hệ thống nhận diện biển số xe hiện đại. Với khả năng học sâu các đặc trưng thị giác,

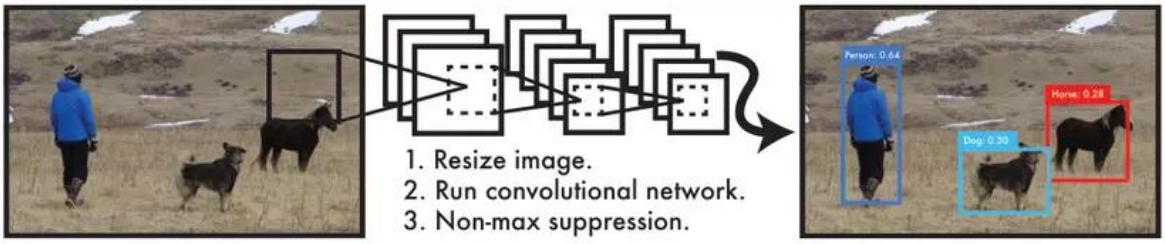
mạng CNN giúp cải thiện đáng kể độ chính xác, tốc độ và khả năng tổng quát của hệ thống, đặc biệt là trong môi trường thực tế nhiều nhiễu và biến động. Việc tích hợp CNN vào hệ thống nhận diện biển số không chỉ là lựa chọn hợp lý mà còn là yếu tố bắt buộc nếu muốn đạt hiệu quả cao trong các ứng dụng thực tiễn như giao thông thông minh, bãi đỗ xe tự động hay giám sát an ninh.

2.1.6. Mô hình YOLO.

Định nghĩa và nguyên lý hoạt động

YOLO (You Only Look Once) là một mô hình học sâu tiên tiến trong lĩnh vực nhận diện đối tượng, được giới thiệu lần đầu tiên bởi Redmond (2016). Không giống như các phương pháp truyền thống như RCNN, vốn chia bài toán nhận diện đối tượng thành nhiều giai đoạn (đề xuất vùng, phân loại, và tinh chỉnh), YOLO thực hiện phát hiện đối tượng trong một lần quét duy nhất, biến bài toán thành một bài toán hồi quy. Mô hình chia hình ảnh đầu vào thành một lưới các ô (grid cells), và mỗi ô dự đoán các hộp giới hạn (bounding boxes), xác suất đối tượng, và lớp đối tượng. Cách tiếp cận này giúp YOLO đạt được tốc độ xử lý vượt trội, phù hợp cho các ứng dụng thời gian thực như giám sát giao thông, an ninh, và nhận diện biển số xe [5].

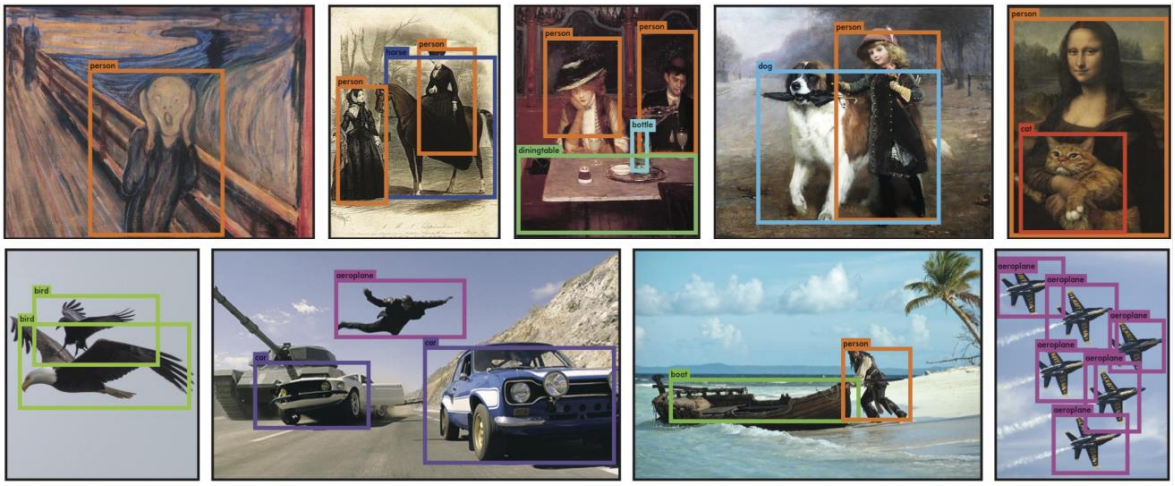
Cụ thể, YOLO sử dụng một mạng nơ-ron tích chập (CNN) để trích xuất đặc trưng từ hình ảnh, sau đó dự đoán đồng thời vị trí và lớp của các đối tượng. Mỗi ô trong lưới được giao nhiệm vụ dự đoán một số lượng cố định các hộp giới hạn, cùng với điểm tin cậy (confidence score) và xác suất lớp. Điểm tin cậy phản ánh khả năng một hộp giới hạn chứa đối tượng, trong khi xác suất lớp xác định loại đối tượng (ví dụ: biển số xe, người đi bộ). Sau đó, các kỹ thuật như non-maximum suppression (NMS) được sử dụng để loại bỏ các hộp giới hạn trùng lặp, giữ lại các dự đoán có độ tin cậy cao nhất [5].



Hình 2.3. Hệ thống nhận diện của YOLOv1 [6]

Hình trên là một ví dụ minh hoạ về hệ thống detection của YOLOv1. Việc xử lý ảnh với YOLOv1 rất đơn giản và dễ dàng. Mô hình trên bao gồm 3 bước:

1) Resize ảnh đầu vào về kích thước 448×448
2) Chạy một mạng CNN lên bức ảnh.
3) Tinh chỉnh kết quả của mô hình.



Hình 2.4. Dự đoán kết của của YOLOv1 trên các bức vẽ [6]

Hình trên là kết quả dự đoán của YOLOv1 trên các bức ảnh vẽ (artwork) và các bức ảnh thực từ internet. Nó cho kết quả dự đoán hầu hết đều chính xác, mặc dù nó nghĩ một người là một cái máy bay. (Hình thứ 7 từ trái qua phải, từ trên xuống dưới.) [6]

Ưu điểm và hạn chế

YOLO có nhiều ưu điểm nổi bật, đặc biệt trong các ứng dụng thời gian thực:

- Tốc độ cao: YOLO có thể đạt tốc độ xử lý lên đến 45 khung hình mỗi giây (FPS) trên phần cứng mạnh như GPU NVIDIA, và thậm chí 15-20 FPS trên các thiết bị nhúng như Jetson Nano. Điều này làm cho YOLO lý tưởng cho các ứng dụng như giám sát giao thông hoặc an ninh.
- Độ chính xác: Các phiên bản mới như YOLO v5 và YOLLO v8 đạt được độ chính xác cao (mAP@50 lên đến 90% trong các điều kiện lý tưởng) khi được huấn luyện trên các tập dữ liệu lớn như COCO hoặc CCPD.
- Tính linh hoạt: YOLO hỗ trợ nhiều biến thể mô hình, từ nano (nhẹ, phù hợp cho thiết bị nhúng) đến xlarge (mạnh mẽ, phù hợp cho các ứng dụng yêu cầu độ chính xác cao), cho phép triển khai trên nhiều nền tảng khác nhau.
- Dễ triển khai: Với các thư viện như Ultralytics YOLO v5 và YOLLO v8, việc huấn luyện và triển khai mô hình trở nên đơn giản, đặc biệt với các công cụ như PyTorch.

Tuy nhiên, YOLO cũng có một số hạn chế:

- Khó khăn với đối tượng nhỏ: Do cách chia lưới, YOLO có thể gặp khó khăn trong việc phát hiện các đối tượng nhỏ hoặc các đối tượng nằm gần nhau trong hình ảnh [5].
- Phụ thuộc vào dữ liệu huấn luyện: Hiệu suất của YOLO phụ thuộc mạnh vào chất lượng và sự đa dạng của tập dữ liệu. Trong các điều kiện không được huấn luyện trước (ví dụ: ánh sáng yếu, góc nhìn bất thường), hiệu suất có thể giảm.
- Độ phức tạp trong tinh chỉnh: Để đạt được hiệu suất tối ưu, YOLO yêu cầu tinh chỉnh các siêu tham số như learning rate, batch size, và số lượng anchor boxes, điều này có thể phức tạp đối với người mới bắt đầu.

Các phiên bản mới như YOLO v5 và YOLLO v8 đã cải thiện một số hạn chế này bằng cách sử dụng các kỹ thuật như chia lưới tinh hơn, tăng cường dữ

liệu, và tích hợp các lớp tích chập bổ sung để cải thiện khả năng phát hiện đối tượng nhỏ.

2.2. Quy trình tổng quát.

2.2.1. Phát hiện vị trí biển số xe (License Plate Detection).

Đây là bước đầu tiên và quan trọng nhất. Học máy được sử dụng để huấn luyện các mô hình có khả năng xác định chính xác vị trí của biển số xe trong một hình ảnh hoặc luồng video. Các kỹ thuật phổ biến bao gồm:

Mạng nơ-ron tích chập (Convolutional Neural Networks - CNNs): Đặc biệt là các kiến trúc phát hiện đối tượng như YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), và Faster R-CNN. Các mô hình này được huấn luyện trên tập dữ liệu lớn chứa hình ảnh xe và các hộp giới hạn (bounding boxes) bao quanh biển số. Chúng có khả năng phát hiện biển số ngay cả khi bị che khuất một phần, ở các góc độ khác nhau, hoặc trong điều kiện ánh sáng phức tạp.

Thuật toán phát hiện đặc trưng (Feature Detection Algorithms): Một số phương pháp truyền thống hơn như Haar Cascades hoặc HOG (Histogram of Oriented Gradients) kết hợp với SVM (Support Vector Machine) cũng có thể được sử dụng để tìm kiếm các đặc trưng hình ảnh điển hình của biển số xe. Tuy nhiên, các phương pháp dựa trên CNN thường cho độ chính xác và khả năng tổng quát hóa vượt trội.

2.2.2. Chuẩn hóa hình ảnh biển số (License Plate Normalization).

Sau khi phát hiện, hình ảnh biển số thường cần được chuẩn hóa để dễ dàng cho quá trình nhận diện ký tự sau này. Học máy giúp trong việc:

- Chỉnh sửa góc nghiêng và méo mó (Skew and Distortion Correction): Các mô hình học sâu có thể được huấn luyện để nhận diện và sửa chữa các biến dạng phối cảnh do góc chụp hoặc độ cong của biển số.
- Cải thiện chất lượng hình ảnh: Các kỹ thuật học máy như loại bỏ nhiễu (denoising) hoặc tăng cường độ tương phản (contrast enhancement) giúp

làm rõ các ký tự trên biển số, đặc biệt trong điều kiện ánh sáng yếu hoặc hình ảnh mờ.

2.2.3. Nhận diện ký tự quang học (Optical Character Recognition - OCR).

Đây là trái tim của hệ thống LPR, nơi các ký tự trên biển số được nhận diện. Học máy, đặc biệt là học sâu, đã cách mạng hóa lĩnh vực này:

- Mạng nơ-ron tích chập (CNNs): Được sử dụng để phân loại từng ký tự riêng lẻ (ví dụ: chữ số, chữ cái) sau khi chúng được tách ra.
- Mạng nơ-ron hồi quy (Recurrent Neural Networks - RNNs) và Long Short-Term Memory (LSTM): Thường được kết hợp với CNNs (trong kiến trúc CRNN – CNN-RNN) để nhận diện chuỗi ký tự mà không cần phải tách rời từng ký tự một. Điều này rất hiệu quả khi các ký tự dính liền hoặc khoảng cách không đều.
- Học không giám sát và bán giám sát: Trong một số trường hợp, các kỹ thuật này có thể được dùng để học các đặc trưng của ký tự từ dữ liệu không có nhãn đầy đủ, giúp giảm công sức gán nhãn.

2.2.4. Xử lý hậu kỳ và xác thực (Post-processing and Validation).

Học máy cũng hỗ trợ trong việc cải thiện độ tin cậy của kết quả:

Kiểm tra tính hợp lệ của biển số: Các mô hình học máy có thể học các quy tắc định dạng của biển số (ví dụ: số lượng ký tự, cấu trúc chữ/số, vị trí dấu gạch ngang) để loại bỏ các kết quả nhận diện sai hoặc không hợp lệ.

So khớp với cơ sở dữ liệu: Học máy có thể giúp so sánh kết quả nhận diện với cơ sở dữ liệu biển số đã biết để xác thực hoặc tìm kiếm thông tin liên quan.

2.3. Lựa chọn mô hình.

Dựa trên yêu cầu cùng quy trình thực hiện của bài toán, YOLOv8 sẽ được sử dụng để phát hiện vùng biển số và CNN để nhận diện ký tự, YOLOv8 được chọn làm mô hình chính cho bước phát hiện nhờ tốc độ xử lý nhanh và độ chính xác cao, trong khi CNN được ưu tiên cho nhận diện ký tự do khả năng trích xuất đặc trưng tự động.

Mô hình YOLO là một hệ thống phát hiện đối tượng thời gian thực dựa trên mạng CNN. YOLO chỉ yêu cầu một lần truyền tiến duy nhất qua mạng để dự đoán vị trí và nhãn của đối tượng trong ảnh, do đó cho tốc độ xử lý rất nhanh. Khả năng phát hiện đa tỷ lệ (multi-scale) giúp YOLO nhận diện hiệu quả biển số ở nhiều cự ly và góc chụp khác nhau. Nhờ khả năng này, YOLO phù hợp với yêu cầu phát hiện vùng biển số trên ảnh thật, đảm bảo thu được vùng biển số chính xác với độ trễ thấp.

Mạng CNN được sử dụng cho khâu nhận diện ký tự vì nó tự động trích xuất đặc trưng từ ảnh và đạt hiệu suất hàng đầu trong các bài toán nhận diện hình ảnh. CNN có khả năng học các đặc trưng cục bộ (như nét chữ, hình dạng ký tự) thông qua các lớp tích chập và gộp mà không cần dựa vào đặc trưng thủ công. Nhiều nghiên cứu chỉ ra rằng CNN đạt độ chính xác rất cao (khoảng 98%) trong các tác vụ nhận diện ký tự và biển số xe, vượt trội so với các phương pháp cổ điển. Nhờ vậy, sử dụng CNN cho phép hệ thống nhận diện chính xác các ký tự Việt Nam trên biển số, ngay cả khi ảnh chụp trong điều kiện ánh sáng phức tạp hoặc ký tự bị méo mó.

Như vậy chúng ta có YOLOv8 chuyên giải quyết tốt yêu cầu phát hiện vùng biển số với tốc độ và độ chính xác cao; CNN khai thác sức mạnh học sâu để nhận diện ký tự phức tạp. Kết hợp hai mô hình này đảm bảo toàn bộ hệ thống hoạt động hiệu quả: YOLOv8 trích vùng biển số nhanh chóng, CNN nhận diện ký tự chính xác, giúp đầu ra cuối cùng là biển số hợp lệ và đáng tin cậy.

2.4. Quy trình thực hiện

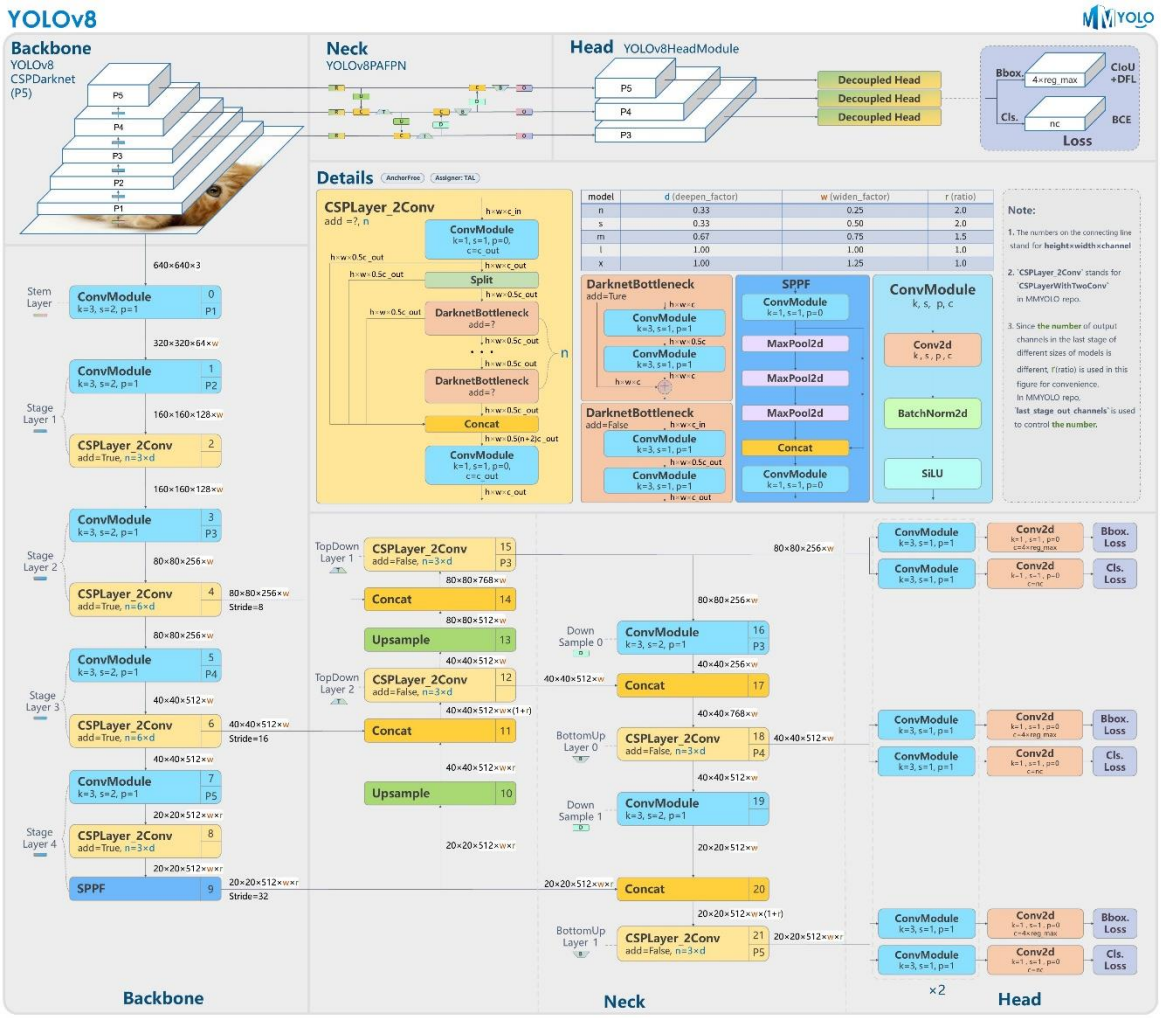
2.4.1. Phát hiện biển số với YOLOv8

Cấu trúc tổng quan:

YOLOv8 chia thành ba thành phần chính – *backbone* (trích xuất đặc trưng), *neck* (kết hợp đặc trưng đa quy mô) và *head* (dự đoán). Backbone của YOLOv8 là CSPDarknet53 với Cross-Stage Partial connections để tăng luồng thông tin giữa các lớp[11]. Neck sử dụng các kiến trúc tương tự FPN (Feature

Pyramid) và PAN (Path Aggregation): module C2f mới thay thế CSPLayer truyền thống, giúp kết hợp đặc trưng ngữ nghĩa sâu và đặc trưng không gian nông[11]. Cuối cùng, head của YOLOv8 là *anchor-free* và tách riêng nhánh phân loại/hồi quy, nghĩa là nó trực tiếp dự đoán tâm và kích thước hộp giới hạn mà không cần các hộp neo định. Nhờ vậy, YOLOv8 cho phép phát hiện đối tượng đa tỉ lệ với độ chính xác cao và tốc độ nhanh.

Các phiên bản: YOLOv8 có nhiều kích cỡ (nano, small, medium, large, x-large). Ví dụ, YOLOv8n (nano) là phiên bản nhỏ nhất, nhanh nhất nhưng độ chính xác (mAP) thấp hơn so với bản lớn (như YOLOv8x). Phiên bản YOLOv8n phù hợp với ứng dụng trên thiết bị nhúng hoặc cần tốc độ cao như nhận diện biển số trong thời gian thực.



Hình 2.5. Kiến trúc YOLOv8 [12]

Backbone Network

Backbone của YOLOv8 xuất phát từ Darknet và được cải tiến theo phong cách CSP. Đầu vào ảnh (ví dụ 640×640×3 pixel) đi qua một lớp Conv 3×3 khởi tạo (thay cho Conv 6×6 của YOLOv5)[12], sau đó lần lượt qua các block CSP và *bottleneck*. Cụ thể, YOLOv8 thay thế các khối C3 (được dùng từ YOLOv5) bằng khối C2f mới. Khối C2f nối (concatenate) kết quả của nhiều convolution nhỏ, giúp tăng số đường skip connection và làm phong phú đặc trưng trích xuất. Ví dụ, trong C2f mỗi bottleneck chia nhánh, tính toán song song và gộp lại, trong khi C3 chỉ dùng đầu ra của bottleneck cuối[12]. Cuối backbone là lớp SPPF (Spatial Pyramid Pooling Fast), cho phép thu thập thông tin đa quy mô từ các kích thước bộ lọc khác nhau và đưa ra ba bản đồ đặc trưng ở các tỉ lệ khác nhau (thường gọi là P3, P4, P5 tương ứng với các kích thước 80×80, 40×40, 20×20 trên ảnh 640×640). Lớp SPPF giúp mạng nắm được cả chi tiết cục bộ và ngữ cảnh tổng thể của ảnh[11].

Neck (FPN + PAN)

Trong phần *neck*, YOLOv8 kết hợp các đặc trưng ở nhiều tầng (multi-scale feature fusion). Mạng sử dụng cấu trúc tương tự FPN-PAN: các feature map ở tầng sâu (P5) được upsample lên và ghép (concatenate) với feature map tại tầng giữa (P4), rồi tiếp tục lên P3; đồng thời có đường downsample (PANet) đưa thông tin từ P3 về P5 để hoàn thiện tích hợp[13]. Khác với FPN truyền thống dùng phép cộng (add) và ép kênh, YOLOv8 ghép trực tiếp (concatenate) các bản đồ mà không bắt buộc kênh phải bằng nhau[13], làm giảm tham số và độ lớn tensor. Tóm lại, neck của YOLOv8 nâng cao khả năng phát hiện đối tượng ở đa kích thước bằng cách tập trung thông tin cả từ đặc trưng "ngữ nghĩa" (lớp sâu) và đặc trưng "hình thái" (lớp nông)[11].

Cách hoạt động:

– Đường xuống (top-down): upsample từ P5→P4→P3, sau mỗi bước nối với feature map tầng trước.

- Đường lên (bottom-up): downsample từ P3→P4→P5, nối thêm vào các bản đồ đã có.
- Các phép concatenate đặc trưng thay cho add, nhờ đó giữ được cả đặc trưng mịn và thô.
- Kết quả neck là các bản đồ feature đã được làm giàu thông tin, sẵn sàng cho head dự đoán.

Detection Head

Anchor-free và decoupled head: Phần head của YOLOv8 hoàn toàn không dựa vào các hộp neo cố định. Thay vào đó, tại mỗi vị trí lưới (grid cell) mạng trực tiếp dự đoán toạ độ tâm và kích thước hộp giới hạn của đối tượng.

YOLOv8 là mô hình *anchor-free*, nghĩa là nó không dùng hộp neo định sẵn. Đối với YOLO truyền thống, toạ độ hộp (b_x, b_y) được tính theo công thức tiêu chuẩn $b_x = \sigma(t_x) + c_x$ dựa trên mã hoá và tâm anchor[13]. Trong khi đó, YOLOv8 chỉ định trực tiếp trọng số sigmoid để dự đoán tâm đối tượng ngay trên lưới mà không cần tham chiếu anchor[13]. Nhờ vậy, số lượng dự đoán giảm đi và bước hậu xử lý NMS cũng nhanh hơn.

Đầu ra của head được tách nhánh rõ ràng: một nhánh chỉ học phân loại (xác suất lớp của đối tượng), một nhánh chỉ học hồi quy toạ độ hộp (center, width, height)[12]. Thiết kế này gọi là *decoupled head*. Cụ thể, YOLOv8 loại bỏ hoàn toàn nhánh "objectness" riêng biệt của các YOLO trước đây, chỉ giữ nhánh phân loại và nhánh hồi quy[12]. Thông tin xác suất xuất hiện đối tượng được gộp chung vào đầu ra phân loại.

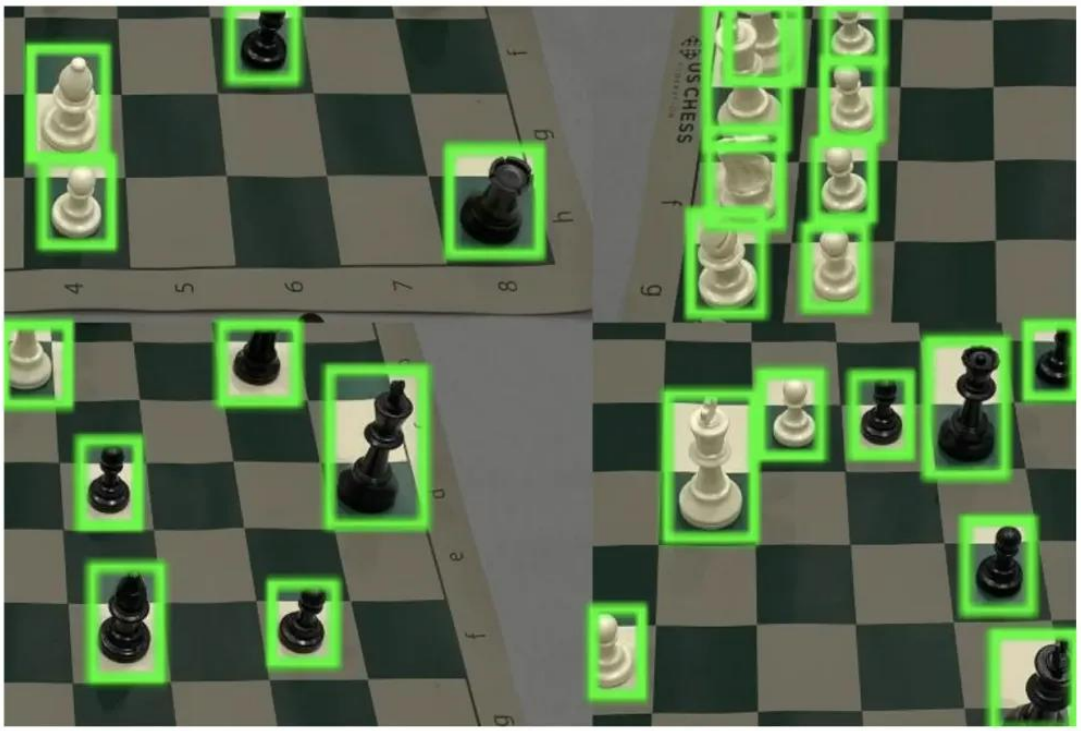
Cấu trúc detection head của YOLOv8. Từ một feature map đầu ra, mạng tách tín hiệu thành hai nhánh song song. Nhánh bên trên (hình 2.5) dự đoán tham số tọa độ hộp giới hạn (kênh hồi quy), nhánh bên dưới dự đoán xác suất thuộc lớp đối tượng (kênh phân loại). Hai nhánh đều áp dụng sigmoid trên output cuối để đưa giá trị về [0,1]. Thiết kế tách nhánh này tăng khả năng tối ưu hóa riêng biệt cho hai nhiệm vụ (class/regression)[12].

- **Nhánh phân loại (Classification)**: đưa ra xác suất mỗi lớp.
- **Nhánh hồi quy (Regression)**: dự đoán tọa độ (x_{center}, y_{center}) và kích thước (w, h) của hộp giới hạn.
- **Không có nhánh objectness riêng**: YOLOv8 tổng hợp độ tin cậy đối tượng vào kết quả phân loại, khác với YOLOv5 trước đây.

Quá trình huấn luyện

Trong huấn luyện YOLOv8, dữ liệu đầu vào là ảnh và nhãn YOLO (tọa độ tâm và kích thước hộp chuẩn hóa). Mỗi ảnh qua nhiều bước xử lý và tính toán gradient để cập nhật trọng số. Một số điểm đáng chú ý trong quá trình huấn luyện:

- **Augmentation (tăng cường dữ liệu)**: YOLOv8 áp dụng nhiều kỹ thuật augmentation để làm phong phú tập huấn luyện. Tiêu biểu là mosaic augmentation: 4 ảnh ngẫu nhiên được ghép lại thành 1 ảnh lớn, giúp mạng học được đối tượng ở nhiều vị trí và tỷ lệ khác nhau. Ví dụ minh họa bên dưới là 4 ảnh bàn cờ được ghép thành một ảnh lớn:



Hình 2.6. Tăng cường dữ liệu từ ảnh các bàn cờ [12]

Ví dụ mosaic augmentation: bốn ảnh (trong đó có các ô bàn cờ màu xanh) được nối thành một ảnh kích thước lớn. Kỹ thuật này buộc mô hình phát hiện đối tượng (ô xanh) trong nhiều vị trí, góc quay và nền khác nhau. Thông thường, mosaic được sử dụng trong phần lớn epoch đầu, nhưng **tắt ở 10 epoch cuối** để ổn định huấn luyện và tránh giảm hiệu năng[12].

- **Multi-scale training**: Ảnh được thay đổi ngẫu nhiên về kích thước trong mỗi batch huấn luyện để mô hình làm quen với các quy mô khác nhau[11]. Điều này đặc biệt hữu ích khi đối tượng (biển số) có kích thước đa dạng.
- **Các augmentation khác**: Ngoài mosaic, còn có các phép biến đổi như xoay, lật ảnh, thay đổi màu sắc, tỷ lệ, giúp tăng tính đa dạng của dữ liệu.
- **Mixed precision**: Huấn luyện sử dụng tính năng *mixed precision* (kết hợp float16 và float32) để tăng tốc và giảm bộ nhớ GPU[11].
- **Hàm mất mát (Loss)**: YOLOv8 tối ưu đồng thời hai thành phần chính: phân loại và hồi quy hộp. Mạng sử dụng chiến lược *TaskAlignedAssigner* (học theo TOOD) để gán các dự đoán tích cực dựa trên kết hợp điểm phân loại và IoU[12]. Hộp giới hạn được huấn luyện bằng Distribution Focal Loss, một biến thể cải tiến của IoU loss giúp hội tụ tốt hơn khi khung hẹp lớn nhỏ khác nhau[12]. Tổng quan, hàm loss bao gồm:
    1. Loss phân loại: thường là Binary Cross-Entropy giữa xác suất dự đoán và nhãn lớp.
    2. Loss hộp: dựa trên IoU mở rộng (ví dụ CIoU/GIoU hoặc distribution focal) giữa bbox dự đoán và gt.
- **Quá trình huấn luyện**: Mỗi epoch, dữ liệu (sau augmentation) được forward qua mạng, tính loss tổng, và cập nhật bằng *backpropagation*. Ví dụ, khi dùng YOLOv8-PyTorch, các tham số như learning rate, batch size, số epoch (thường ~300-500) được đặt trong file cấu hình hoặc dòng

lệnh. Với đầu ra 2 lớp, cuối cùng mạng học được khả năng phát hiện và phân loại biển số theo đúng các nhãn cung cấp.

Ví dụ luồng dữ liệu với nhận diện biển số xe: Khi huấn luyện phát hiện biển số Việt Nam, mỗi ảnh biển số (ở chế độ dọc hoặc vuông) được chuyền qua các bước trên. Backbone trích xuất 3 độ phân giải đặc trưng, neck kết hợp chúng, head dự đoán các vị trí hộp biển số và xác suất hai loại biển số. Nhờ anchor-free, mạng học được vị trí biển số trực tiếp, không phụ thuộc phân bố anchor. Các kỹ thuật augmentation như mosaic giúp mạng nhận biết biển số dù xuất hiện lác đác hoặc che khuất một phần. Sau huấn luyện, YOLOv8n sẽ đưa ra trọng số tối ưu để trên ảnh mới, nó có thể dự đoán hộp biển số và phân loại chính xác.

2.4.2. Nhận diện chữ số với CNN

Tiền xử lý và tách ký tự là bước quan trọng để đảm bảo đầu vào cho CNN có chất lượng cao. Ảnh biển số từ YOLOv8 không được đưa trực tiếp vào CNN nhận diện ký tự. Thay vào đó, nó phải trải qua các bước sau:

1. Chuyển đổi sang ảnh xám (Grayscale): Màu sắc không mang nhiều thông tin cần thiết để nhận diện ký tự, việc chuyển sang ảnh xám giúp giảm độ phức tạp của dữ liệu (từ 3 kênh RGB xuống 1 kênh) và tăng tốc độ xử lý [14].
2. Nhị phân hóa (Binarization): Áp dụng một ngưỡng để chuyển ảnh xám thành ảnh đen trắng. Kỹ thuật ngưỡng động (adaptive thresholding) hoặc ngưỡng Otsu thường được sử dụng để xử lý hiệu quả các điều kiện ánh sáng không đồng đều trên biển số, làm nổi bật ký tự trên nền [14].
3. Tách ký tự (Character Segmentation): Sử dụng các thuật toán xử lý ảnh như tìm đường bao (contour detection) để xác định vị trí của từng ký tự riêng lẻ. Mỗi đường bao khép kín, thỏa mãn các tiêu chí về tỷ lệ khung hình và diện tích, sẽ được coi là một ký tự. Các ảnh ký tự này sau đó được cắt ra.

4. Chuẩn hóa kích thước: Tất cả các ảnh ký tự sau khi cắt sẽ được thay đổi kích thước về một kích thước cố định (ví dụ: 32×32 hoặc 28×28 pixel) để làm đầu vào đồng nhất cho mạng CNN [15].

Mô hình CNN cho nhận diện ký tự thường có kiến trúc tương đối đơn giản nhưng hiệu quả, bao gồm các lớp chính sau:

1. Lớp Tích chập (Convolutional Layer): Đây là thành phần cốt lõi của CNN. Các bộ lọc (kernel) trượt qua ảnh đầu vào để trích xuất các đặc trưng cấp thấp như cạnh, góc, đường cong.
2. Hàm kích hoạt (Activation Function): Thông thường là ReLU (Rectified Linear Unit), được áp dụng sau mỗi lớp tích chập để đưa yếu tố phi tuyến vào mô hình, giúp mạng học được các mối quan hệ phức tạp hơn.
3. Lớp Gộp (Pooling Layer): Thường là Max Pooling, được dùng để giảm kích thước không gian của các bản đồ đặc trưng (feature map), giúp giảm số lượng tham số, chống overfitting và làm cho mô hình bất biến hơn với các thay đổi nhỏ về vị trí của đặc trưng [14].
4. Lớp Duỗi phẳng (Flatten Layer): Chuyển đổi ma trận đặc trưng 2D từ lớp gộp cuối cùng thành một vector 1D để chuẩn bị cho các lớp kết nối đầy đủ.
5. Lớp Kết nối đầy đủ (Fully Connected/Dense Layer): Các nơ-ron trong lớp này được kết nối với tất cả các đầu ra từ lớp trước. Chúng có nhiệm vụ tổng hợp các đặc trưng đã được trích xuất để thực hiện phân loại. Thường có một hoặc hai lớp kết nối đầy đủ trước lớp đầu ra.
6. Lớp Dropout: Một kỹ thuật điều chuẩn (regularization) được sử dụng giữa các lớp kết nối đầy đủ để chống overfitting bằng cách ngẫu nhiên vô hiệu hóa một tỷ lệ nơ-ron trong quá trình huấn luyện [15].
7. Lớp Đầu ra (Output Layer): Là một lớp kết nối đầy đủ với số nơ-ron bằng số lớp ký tự cần phân loại (ví dụ: 36 lớp cho A-Z và 0-9). Hàm kích hoạt Softmax được sử dụng ở lớp này để chuyển đổi đầu ra thành một phân phối xác suất, trong đó mỗi giá trị biểu thị xác suất ký tự đầu vào

thuộc về một lớp cụ thể. Ký tự có xác suất cao nhất sẽ được chọn làm kết quả dự đoán.

Mô hình CNN cần được huấn luyện trên một tập dữ liệu lớn chứa các ảnh ký tự đã được gán nhãn.

Tập dữ liệu: Bao gồm hàng ngàn ảnh của mỗi ký tự (0-9, A-Z) được trích xuất từ các biển số thực tế và đã qua các bước tiền xử lý (ảnh xám, nhị phân hóa, chuẩn hóa kích thước). Việc tăng cường dữ liệu (data augmentation) như xoay nhẹ, dịch chuyển, thêm nhiễu cũng được áp dụng để tăng tính đa dạng và độ chính xác cho mô hình [14].

Hàm mất mát (Loss Function): Vì đây là bài toán phân loại đa lớp, hàm mất mát Categorical Cross-Entropy là lựa chọn phổ biến nhất. Hàm này đo lường sự khác biệt giữa phân phối xác suất dự đoán bởi mô hình và phân phối xác suất thực tế (nhãn one-hot), và cố gắng tối thiểu hóa sự khác biệt này [15].

Bộ tối ưu hóa (Optimizer): Các thuật toán như Adam hoặc SGD (Stochastic Gradient Descent) được sử dụng để cập nhật trọng số của mạng dựa trên giá trị loss tính được, nhằm đưa mô hình hội tụ đến điểm tối ưu.

Đánh giá: Hiệu năng của mô hình được đánh giá dựa trên độ chính xác (accuracy) trên tập dữ liệu kiểm thử (validation set) trong suốt quá trình huấn luyện và trên tập dữ liệu thử nghiệm (test set) sau khi huấn luyện xong.

Tích hợp và tạo kết quả cuối cùng sau khi mô hình CNN được huấn luyện, quy trình nhận diện hoàn chỉnh diễn ra:

1. Ảnh đầu vào được xử lý bởi YOLOv8 để lấy ra ảnh cắt của biển số.
2. Ảnh biển số được tiền xử lý và tách thành các ảnh ký tự riêng lẻ.
3. Từng ảnh ký tự được đưa vào mô hình CNN.
4. CNN trả về ký tự dự đoán cho mỗi ảnh (ví dụ: '5', '1', 'F', '8', ...).
5. Các ký tự được sắp xếp lại theo thứ tự vị trí của chúng trên biển số (thường từ trái sang phải, từ trên xuống dưới) để tạo thành chuỗi biển số cuối cùng, ví dụ: "51F-89912".

2.5. Các công cụ hỗ trợ.

Trong quá trình phát triển và thực nghiệm hệ thống nhận diện biển số xe Việt Nam, nhóm đã sử dụng ngôn ngữ lập trình Python, môi trường Google Colab và một số thư viện chính, cụ thể như sau:

- **Python**: Ngôn ngữ lập trình bậc cao, dễ đọc, nhiều thư viện hỗ trợ khoa học dữ liệu và học máy. Python cho phép triển khai nhanh các thuật toán, kết hợp linh hoạt giữa xử lý ảnh (OpenCV), học sâu (TensorFlow/Keras) và phân tích số liệu (NumPy, Pandas).
- **Google Colab**: Môi trường Jupyter Notebook chạy trên đám mây, cung cấp miễn phí GPU/TPU, dễ dàng chia sẻ và tương tác. Colab giúp nhóm thực thi mã, huấn luyện mô hình quy mô lớn mà không cần cấu hình cục bộ, đồng thời lưu trữ kết quả trên Google Drive để quản lý thuận tiện.
- **TensorFlow / Keras**: TensorFlow là framework học sâu do Google phát triển, hỗ trợ xây dựng và huấn luyện mạng nơ-ron phức tạp. Keras là API cấp cao tích hợp trong TensorFlow, cho phép định nghĩa mô hình CNN, YOLO một cách ngắn gọn và trực quan. Bộ đôi này giúp triển khai nhanh các kiến trúc mạng, tối ưu hoá bằng GPU và đơn giản hoá quy trình debug.
- **OpenCV**: Thư viện mã nguồn mở hàng đầu về xử lý ảnh và thị giác máy tính. OpenCV cung cấp hàng trăm hàm xử lý ảnh cơ bản (lọc, làm sạch nhiễu, phân ngưỡng) và nâng cao (phát hiện cạnh, biến đổi phối cảnh) để chuẩn bị dữ liệu đầu vào cho mô hình học sâu.
- **NumPy**: Thư viện cốt lõi cho tính toán mảng đa chiều. NumPy cung cấp cấu trúc dữ liệu ndarray hiệu quả và các phép toán tuyến tính, giúp xử lý ma trận ảnh, chuyển đổi kích thước, chuẩn hoá giá trị pixel trước khi đưa vào mô hình.
- **Pandas**: Thư viện chuyên dụng cho thao tác và phân tích dữ liệu dạng bảng. Pandas hỗ trợ đọc, ghi file CSV/Excel, xử lý nhãn (labels) của ảnh,

thống kê kết quả thử nghiệm và xuất báo cáo dưới dạng DataFrame trực quan.

- **Matplotlib**: Thư viện vẽ đồ thị 2D cơ bản. Matplotlib được sử dụng để trực quan hoá quá trình huấn luyện (đồ thị hàm mất mát – loss, độ chính xác – accuracy), biểu diễn phân phối kích thước vùng biển số, hoặc so sánh kết quả giữa các mô hình.

CHƯƠNG 3: THỰC NGHIỆM VÀ KẾT QUẢ

3.1. Bộ dữ liệu sử dụng.

3.1.1. Dữ liệu biển số xe Việt Nam.

Tên bộ dữ liệu: Vietnam License Plate Segment Datasets

Nguồn:
https://www.kaggle.com/datasets/duydieunguyen/licenseplates?resource=download

Tổng số: ~5000 ảnh

Số lớp: 2 (BSD – Biển số dài, BSV – Biển số vuông)

Định dạng ảnh: .jpg và .png

Định dạng nhãn: YOLO annotation format

Kích thước ảnh: Đa dạng từ 640x480 đến 1920x1080

3.1.2. Dữ liệu số và chữ.

Tên bộ dữ liệu: standard OCR dataset

Nguồn:
https://www.kaggle.com/datasets/preatcher/standard-ocr-dataset/data

Số lớp: 36 lớp (10 chữ số + 26 chữ cái)

Format: Ảnh JPEG, RGB

Đặc điểm: Mỗi ảnh chứa một ký tự đơn lẻ đã được cắt từ biển số xe

Mô tả bộ dữ liệu

Bộ dữ liệu Standard OCR được tạo ra bởi Abhishek Jaiswal và được công bố trên nền tảng Kaggle. Bộ dữ liệu này là một tập dữ liệu phổ biến và hữu ích trong lĩnh vực nhận diện ký tự quang học (OCR – Optical Character Recognition), phục vụ mục đích huấn luyện và kiểm thử các mô hình học máy và học sâu. Bộ dữ liệu được sử dụng trong bài báo cáo này tạo ra bằng cách tổng hợp ảnh nhân tạo (synthetic) bằng cách sử dụng các font chữ máy tính kết hợp với các kỹ thuật thêm nhiễu và biến dạng ảnh nhằm giả lập điều kiện thực tế.

Bộ dữ liệu bao gồm ảnh đơn kênh (grayscale) với kích thước chuẩn là 28x28 pixel với định dạng .png. Mỗi ảnh biểu diễn một ký tự riêng lẻ – có thể là chữ số, chữ cái. Các nhãn (labels) sẽ được gán vào tên tệp của các ký tự chạy từ 0 - 9 và chữ in hoa từ A - Z. Tập dữ liệu được chia sẵn thành hai phần: tập huấn luyện (train set) và tập kiểm thử (test set).

Mỗi tập huấn luyện hay tập kiểm thử đều chứa tổng cộng 36 thư mục. Trong tập train mỗi thư mục tương ứng với một chữ cái in hoa từ A đến Z và từ 0 đến 9. Mỗi thư mục trong tập train có hơn 573 hình ảnh của ký tự tương ứng, dẫn đến tổng số hình ảnh của tập train là 20,648. Đối với tập test, mỗi thư mục có hơn 27 hình ảnh của ký tự tương ứng nên tổng số hình ảnh của tập test là 1

Bộ dữ liệu Standard OCR Dataset không chỉ cung cấp một nguồn dữ liệu phong phú và đa dạng cho các dự án về nhận diện ký tự văn bản, mà còn góp phần quan trọng vào sự phát triển của lĩnh vực OCR. Với cấu trúc dữ liệu rõ ràng, cân bằng và dễ dàng xử lý, bộ dữ liệu này là một công cụ hữu ích cho cả những người mới bắt đầu và các chuyên gia trong lĩnh vực học máy và trí tuệ nhân tạo.

3.2. Quy trình thực nghiệm.

3.2.1. Phát hiện vùng biển số với YOLOv8.

Thiết lập môi trường

Thiết lập môi trường phát triển cho YOLO v8, bao gồm việc cài đặt thư viện cần thiết và kiểm tra khả năng sử dụng GPU để tăng tốc quá trình training.

*Lưu ý: Ở đây, nhóm sử dụng PyTorch với CUDA GPU trong quá trình huấn luyện. GPU là yếu tố quan trọng trong việc huấn luyện models. Với GPU, thời gian training có thể giảm từ vài ngày xuống còn vài giờ. YOLO v8 yêu cầu ít nhất 4GB VRAM để training hiệu quả.

```
# Kiểm tra PyTorch đã nhận GPU chưa
import torch
print('CUDA available:', torch.cuda.is_available())
print('CUDA device count:', torch.cuda.device_count())

# Kiểm tra current device chỉ khi CUDA available
if torch.cuda.is_available():
    print('Current device:', torch.cuda.current_device())
    print('Device name:',
torch.cuda.get_device_name(torch.cuda.current_device()))
else:
    print('CUDA không khả dụng, không thể lấy thông tin device')

# Kiểm tra Ultralytics YOLO
import ultralytics
print('Ultralytics version:', ultralytics.__version__)

```

Kết quả trả về sẽ được hiển thị cho biết máy có GPU chưa, nếu không có, mô hình sẽ tự động lấy CPU để huấn luyện. Ảnh dưới là kết quả trả về nếu GPU được nhận diện.

```
CUDA available: True
CUDA device count: 1
Current device: 0
Device name: NVIDIA GeForce GTX 1650
Ultralytics version: 8.3.160
```

Hình 3.1. Nhận diện GPU cho quá trình huấn luyện YOLOv8

Chuẩn bị dữ liệu

Xác minh cấu trúc dataset "Vietnam License Plate Segment Datasets" đã nêu và file cấu hình YOLOv8 để đảm bảo dữ liệu được tổ chức đúng định dạng. Bước này rất quan trọng vì YOLOv8 yêu cầu cấu trúc thư mục và annotation format cụ thể.

```
from ultralytics import YOLO

# Đường dẫn file cấu hình dữ liệu
DATASET_YAML = '..\\vietnamese-license-plates\\dataset.yaml'
# Khởi tạo model YOLOv8n (có thể thay bằng yolov8s, yolov8m...)
model = YOLO('yolov8n.pt')
print("Dataset configuration:")
with open(DATASET_YAML, 'r') as f:
    print(f.read())
```

```
Dataset configuration:
names:
- BSD
- BSV
nc: 2
path: c:\Users\Seotow\OneDrive - Thang Long University\Hieu\Learn\ThirdYear\MachineLearning\BTL\vietnamese-
test: /images/test/
train: images/train
val: images/val
```

Hình 3.2. Kiểm tra bộ dữ liệu cho mô hình YOLOv8

Huấn luyện dữ liệu

Quá trình này bao gồm việc load pretrained weights, cấu hình hyperparameters, và thực hiện training với nhiều kỹ thuật tối ưu hóa.

```python
# Train model
results = model.train(
    data=DATASET_YAML,
    epochs=50,            # Số epoch
    imgsz=640,            # Kích thước ảnh
    batch=16,             # Batch size, điều chỉnh theo VRAM
    device=0,             # 0: GPU đầu tiên
    project='runs/detect', # Thư mục lưu kết quả
    name='yolo-plates-custom', # Tên folder/model
    exist_ok=True         # Ghi đè nếu đã tồn tại
)

# Lưu model đã train
model.save('yolo-plates-custom.pt')
print('Đã train xong và lưu model tại runs/detect/yolo-plates-custom/')
```

```
50 epochs completed in 1.734 hours.
Optimizer stripped from runs\detect\yolo-plates-custom\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\yolo-plates-custom\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\yolo-plates-custom\weights\best.pt, 6.2MB

Validating runs\detect\yolo-plates-custom\weights\best.pt...
Ultralytics 8.3.160  Python-3.11.0 torch-2.5.1+cu121 CUDA:0 (NVIDIA GeForce GTX 1650, 4096MiB)
Optimizer stripped from runs\detect\yolo-plates-custom\weights\best.pt, 6.2MB

Validating runs\detect\yolo-plates-custom\weights\best.pt...
Ultralytics 8.3.160  Python-3.11.0 torch-2.5.1+cu121 CUDA:0 (NVIDIA GeForce GTX 1650, 4096MiB)
Model summary (fused): 72 layers, 3,006,038 parameters, 0 gradients, 8.1 GFLOPs
Model summary (fused): 72 layers, 3,006,038 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|

                   all       1145       1313      0.983      0.992      0.994      0.913
                   BSD        409        410      0.992      0.995      0.995      0.908
                   BSV        753        903      0.975      0.989      0.994      0.918
Speed: 0.3ms preprocess, 5.8ms inference, 0.0ms loss, 2.7ms postprocess per image
                   BSD        409        410      0.992      0.995      0.995      0.908
                   BSV        753        903      0.975      0.989      0.994      0.918
Speed: 0.3ms preprocess, 5.8ms inference, 0.0ms loss, 2.7ms postprocess per image
Results saved to runs\detect\yolo-plates-custom
Results saved to runs\detect\yolo-plates-custom
Đã train xong và lưu model tại runs/detect/yolo-plates-custom/
Đã train xong và lưu model tại runs/detect/yolo-plates-custom/
```

Hình 3.3. Hoàn thành huấn luyện mô hình YOLOv8

Mô hình đã được huấn luyện trong 50 epochs trên tập dữ liệu gồm hai loại biển số: biển số dài (BSD) và biển số vuông (BSV). Quá trình huấn luyện được thực hiện trên GPU NVIDIA GeForce GTX 1650 (4GB VRAM), với thời gian hoàn thành khoảng 1.73 giờ. Mô hình gồm 72 lớp và hơn 3 triệu tham số, đạt tốc độ suy luận trung bình khoảng 5.8 ms mỗi ảnh, phù hợp cho ứng dụng thời gian thực.

Thực hiện dự đoán

Vậy là mô hình đã được huấn luyện xong, tiếp đến ta đem mô hình đi thực hiện dự đoán trên ảnh thực tế. Trước hết ta cần chuẩn bị tập ảnh

```
import cv2
import matplotlib.pyplot as plt
import os
import glob
from PIL import Image
import numpy as np

# Danh sách các tệp ảnh thử nghiệm
test_images = ['1.jpg', '11.jpg', '3.jpg', '745.jpg', '9145.jpg']

# Kiểm tra và tạo danh sách các ảnh có sẵn
available_images = []
for img_file in test_images:
    full_path = os.path.join('..', img_file)
    if os.path.exists(full_path):
        available_images.append(full_path)
        print(f"Tìm thấy ảnh test: {full_path}")
    else:
        print(f"Không tìm thấy ảnh: {full_path}")

# Nếu không tìm thấy ảnh nào, tìm kiếm trong thư mục val
if not available_images:
    print("Không tìm thấy ảnh test trong thư mục gốc, tìm kiếm trong thư mục val...")
    val_images = glob.glob('../vietnamese-license-plates/images/val/*.jpg')[:5]
    if val_images:
        available_images = val_images
        for img in available_images:
            print(f"Sử dụng ảnh val: {img}")
    else:
        print("Không tìm thấy ảnh nào trong thư mục val")

# Hiển thị ảnh đầu tiên nếu có
```

```
if available_images:
    img = cv2.imread(available_images[1])
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(10, 8))
    plt.imshow(img_rgb)
    plt.title("Ảnh test đầu tiên")
    plt.axis('off')
    plt.show()
else:
    print("Không tìm thấy ảnh test nào để hiển thị")

```

```
Tìm thấy ảnh test: ..\1.jpg
Tìm thấy ảnh test: ..\11.jpg
Tìm thấy ảnh test: ..\3.jpg
Tìm thấy ảnh test: ..\745.jpg
Tìm thấy ảnh test: ..\9145.jpg
```

Hình 3.4. Ảnh được lấy ngẫu nhiên cho quá trình dự đoán

Sau đó ta đưa ảnh vào mô hình đã huấn luyện và xem kết quả.

```
import torch

if not available_images:
    print("Không có ảnh test nào để dự đoán")
else:
    # Chọn thiết bị để dự đoán (GPU hoặc CPU)
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    print(f"Sử dụng thiết bị: {device}")
```

```
 9.
10.     # Thực hiện dự đoán trên ảnh đầu tiên
11.     first_image = available_images[1]
12.     print(f"Đang dự đoán trên ảnh: {first_image}")
13.
14.     # Thực hiện dự đoán
15.     results = model.predict(first_image, conf=0.25,
device=device)  # conf là ngưỡng tin cậy
16.
17.     print(f"Đã dự đoán xong, tìm thấy {len(results[0].boxes)} đối
tượng")
18.
19.     # In thông tin chi tiết về các đối tượng phát hiện được
20.     for i, box in enumerate(results[0].boxes):
21.         confidence = box.conf.item()
22.         cls_id = int(box.cls.item())
23.         cls_name = results[0].names[cls_id]
24.         print(f"Đối tượng {i+1}: {cls_name}, độ tin cậy:
{confidence:.2f}")
25. import matplotlib.pyplot as plt
26. import matplotlib.patches as patches
27. import numpy as np
28. from PIL import Image, ImageDraw, ImageFont
29.
30. if not available_images or len(results[0].boxes) == 0:
31.     print("Không có kết quả để hiển thị")
32. else:
33.     plt.figure(figsize=(12, 10))
34.     img_result = results[0].plot()
35.     plt.imshow(cv2.cvtColor(img_result, cv2.COLOR_BGR2RGB))
36.     plt.title("Kết quả nhận diện")
37.     plt.axis('off')
38.     plt.show()
```

Bảng 3.1. Phát hiện vùng biển số trên ảnh

Như vậy, đầu ra là ảnh đã được khoanh vùng biển số và phân chính xác vào lớp BSV với độ tin cậy 0.94.

Đánh giá mô hình

Cuối cùng, ta cần theo dõi quá trình training thông qua visualization của các metrics quan trọng. Việc đánh giá giúp phát hiện overfitting, underfitting và điều chỉnh hyperparameters kịp thời. Để đánh giá mô hình đã huấn luyện, ta coi việc nhận diện có biển số xe là Positive và không có biển số xe là Negative và True/False thể hiện dự đoán đúng sai.

- True Positive (TP): dự đoán có biển số - Positive, thực tế thì có biển (True)
- False Positive (FP): dự đoán có biển số - Positive, thực tế thì không có biển số (False).
- True Negative (TN): dự đoán không có biển số - Negative, thực tế không có biển số (True)
- False Negative (FN): dự đoán không có biển số - Negative, thực tế thì có biển số (False)

Từ đó ta có các chỉ số đánh giá:

- Precision (Độ chính xác): Tỷ lệ dự đoán positive đúng trong tất cả các dự đoán positive.

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

- Recall (Độ bao phủ): Tỷ lệ dự đoán positive đúng trong tổng số trường hợp nhãn positive

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

- F1-score: Trung bình điều hòa (harmonic mean) giữa Precision và Recall.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

– mAP@50 (mean Average Precision at IoU=0.5): chỉ số trung bình của Average Precision (AP) cho tất cả các lớp tại ngưỡng IoU=0.5

$$mAP@50 = \frac{1}{N}\sum_{i=1}^{N} AP_i^{@50}$$

– mAP@50:95 (mAP trung bình từ IoU=0.5 đến 0.95): giá trị trung bình của mAP tại các ngưỡng IoU từ 0.5 đến 0.95, bước nhảy 0.05

$$mAP@50{:}\,95 = \frac{1}{10}\sum_{k=0}^{9} mAP@(0.5 + 0.05k)$$

Với: N là số lớp (tổng BSD + BSV)

$AP_i^{@x}$ là AP của lớp I tại ngưỡng IoU = x

IoU (Intersection over Union): $\frac{Diện\ tích\ vùng\ giao\ nhau}{Diện\ tích\ vùng\ hợp\ nhất}$

Cách tính AP@x:

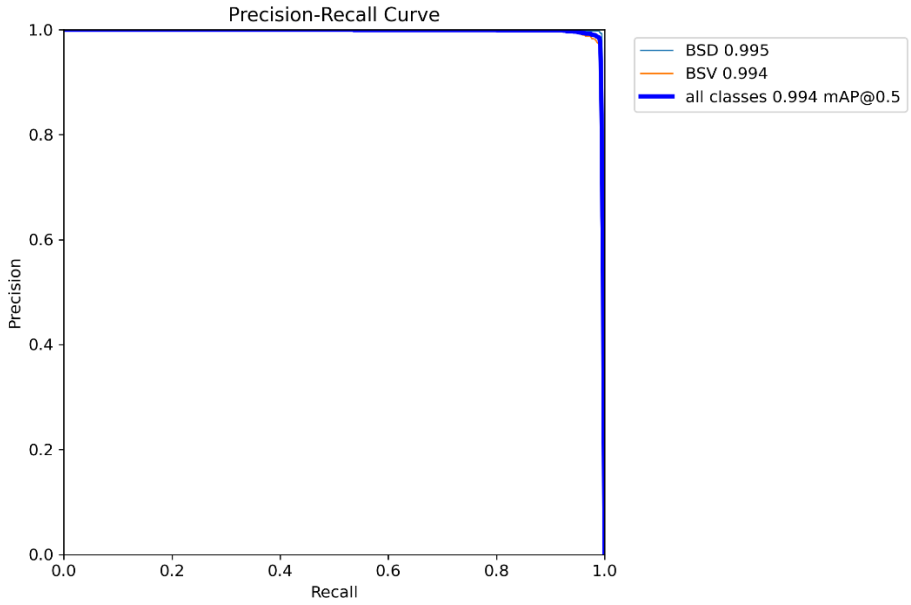
– Vẽ đường Precision-Recall curve khi thay đổi ngưỡng confidence

– Tính diện tích đường cong (AUC)

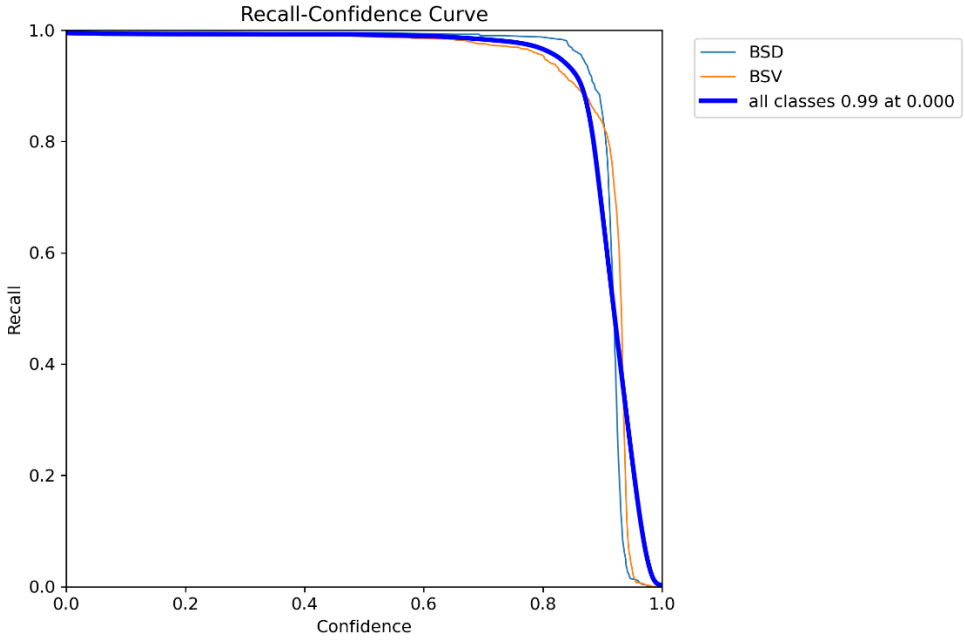
```
# Đường dẫn đến tập validation
val_dir = '../vietnamese-license-plates/images/val'

# Kiểm tra nếu tập validation tồn tại
if os.path.exists(val_dir):

    # Thực hiện đánh giá
    results = model.val(data=DATASET_YAML, split='val')

    # In kết quả đánh giá
    metrics = results.box
    print("\nKết quả đánh giá mô hình:")
    print(f"mAP@50: {metrics.map50}")
    print(f"mAP@50-95: {metrics.map}")
    print(f"Precision: {metrics.p}")
    print(f"Recall: {metrics.r}")
```

```
    print(f"F1-Score: {metrics.f1}")
else:
    print(f"Không tìm thấy thư mục validation: {val_dir}")

```

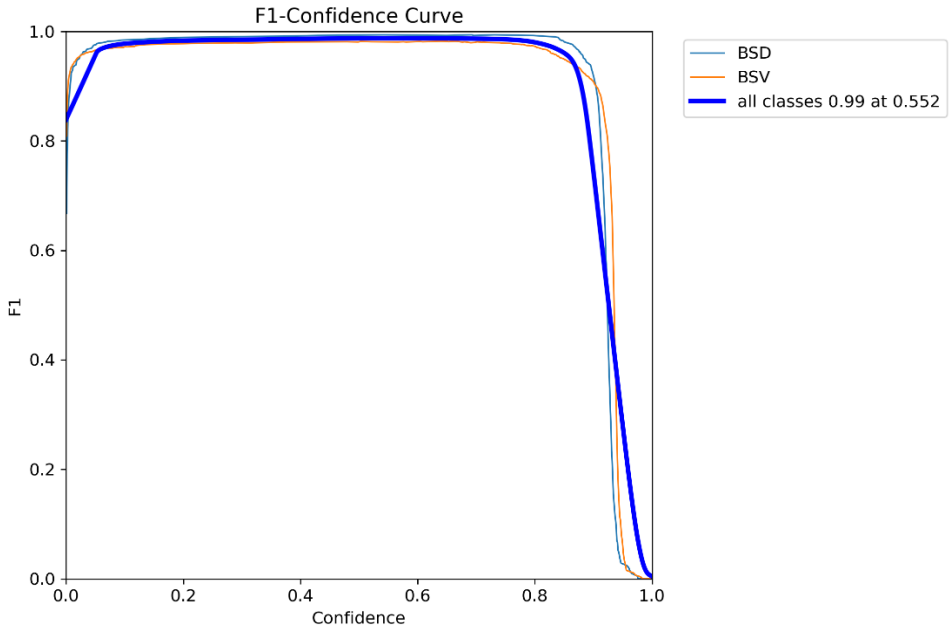
Bảng 3.2. Các chỉ số tổng quan mô hình YOLO



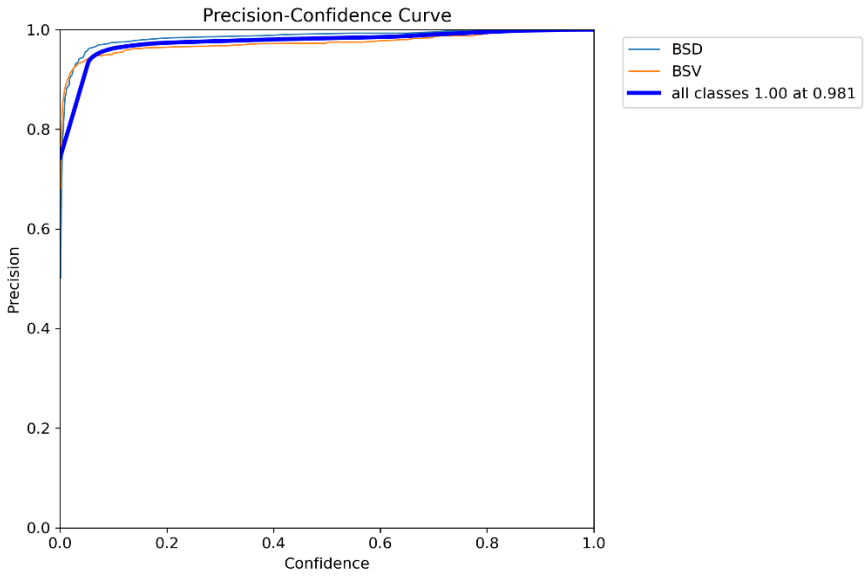
Hình 3.5. Đường Precision-Recall của mô hình YOLO



Hình 3.6. Đường Recall-Confidence của mô hình YOLO



Hình 3.7. Đường F1-Confidence của mô hình YOLO



Hình 3.8. Đường Precision-Confidence của mô hình YOLO

Giải thích các chỉ số:

– mAP@0.5 = 0.994182: Mô hình nhận diện đúng với độ chính xác rất cao ở ngưỡng IoU ≥ 0.5 (gần như hoàn hảo).

- mAP@0.5:0.95 = 0.912403: Độ chính xác trung bình ở nhiều mức IoU từ 0.5 đến 0.95, cho thấy mô hình ổn định và chính xác trong nhiều tình huống chồng lắp.
- Precision-Recall (PR): Precision trung bình 0.994 → Ít bị ảnh hưởng bởi false positive ở mọi mức Recall.
- Precision-Confidence (P): Precision = 1.0 tại confidence > 0.981 → Dự đoán chính xác tuyệt đối khi model tự tin cao.
- Recall-Confidence (R): Recall > 0.99 tại confidence thấp (~0.000) → Khả năng phát hiện đối tượng vượt trội.
- F1-Confidence (F1): F1 tối ưu 0.99 tại confidence = 0.552 → Ngưỡng lý tưởng cho cân bằng Precision-Recall.

Như vậy, Mô hình YOLO được huấn luyện đã đạt kết quả khá xuất sắc trên tập validation với độ chính xác gần như tuyệt đối, tốc độ xử lý nhanh và ổn định. F1-score cao thể hiện sự cân bằng giữa độ nhạy và độ chính xác. Từ đó, phù hợp để triển khai cho bài toán nhận diện biển số xe trong môi trường thực tế với độ tin cậy cao.

3.2.2. Nhận diện chữ số với CNN.

Sau khi huấn luyện được mô hình phát hiện vùng biển số với YOLOv8, ta tiếp tục tiến hành nhận diện chữ với CNN. Đầu tiên, ta cần nạp các thư viện cần thiết cho quá trình huấn luyện.

```
1. import numpy as np
2. import pandas as pd
3. import cv2
4. import os
5. import string
6. import tensorflow as tf
7. import matplotlib.pyplot as plt
8. from sklearn.model_selection import train_test_split
9. from tensorflow.keras.utils import to_categorical,
image_dataset_from_directory
10. from tensorflow.keras import optimizers
11. from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D,
Dropout, Flatten
```

```
12. from tensorflow.keras.callbacks import ReduceLROnPlateau,
ModelCheckpoint
13. from tensorflow.keras.models import Sequential
14. import tensorflow.keras.backend as K
15. from tensorflow.keras.preprocessing.image import
ImageDataGenerator
```

Đối với mô hình CNN nhóm sử dụng ImageDataGenerator để tăng cường dữ liệu cho việc huấn luyện mô hình học máy. Với mục đích chuẩn hóa hết giá trị ảnh đầu vào về chạy từ 0-1. Đồng thời cài dịch chuyển ảnh theo chiều dọc và chiều ngang để tạo ra nhiều biến thể cho mô hình

```
 1. train_datagen = ImageDataGenerator(rescale=1./255,
 2.                                     width_shift_range=0.1,
 3.                                     height_shift_range=0.1)
 4.
 5. # Set paths to local data
 6. data_path = os.path.join(BASE_PATH, "data", "data")
 7. train_path = os.path.join(data_path, "train")
 8. val_path = os.path.join(data_path, "val")
 9.
10. print(f"Data path: {data_path}")
11. print(f"Train path: {train_path}")
12. print(f"Validation path: {val_path}")
13.
14. # Verify paths exist
15. print(f"Train path exists: {os.path.exists(train_path)}")
16. print(f"Validation path exists: {os.path.exists(val_path)}")
17.
18. train_generator = train_datagen.flow_from_directory(
19.         train_path,  # this is the target directory
20.         target_size=(28,28),  # tất cả các hình ảnh sẽ được thay
đổi kích thước thành 28x28
21.         batch_size=32,  # Increase batch size for better training
22.         class_mode='sparse')
23.
24. validation_generator = train_datagen.flow_from_directory(
25.     val_path,  # validation directory
26.     target_size=(28,28),    # tất cả các hình ảnh sẽ được thay
đổi kích thước thành 28x28
27.     batch_size=32,
28.     class_mode='sparse')
29.
30. print(f"Training classes: {train_generator.class_indices}")
31. print(f"Number of training samples: {train_generator.samples}")
32. print(f"Number of validation samples:
{validation_generator.samples}")
```

```
Train path exists: True
Validation path exists: True
Found 864 images belonging to 36 classes.
Found 216 images belonging to 36 classes.
Training classes: {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, 'A': 10,
Number of training samples: 864
Number of validation samples: 216
```

Hình 3.9. Dữ liệu sau tiền xử lý

Sau đó đọc ảnh từ thư mục, áp dụng các phép biến đổi (augmentation) đã định nghĩa ở train_datagen, và cung cấp dữ liệu cho quá trình huấn luyện mô hình. Trong quá trình đọc, ảnh sẽ được resize về kích thước 28x28 pixel.

Huấn luyện mô hình

Xây dựng mô hình

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Input
from keras import optimizers

model = Sequential([

    Conv2D(16, (22, 22), input_shape=(28, 28, 3), activation='relu', padding='same'),
    Conv2D(32, (16, 16), activation='relu', padding='same'),  # Không cần input_shape
    Conv2D(64, (8, 8), activation='relu', padding='same'),
    Conv2D(64, (4, 4), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(4, 4)),
    Dropout(0.4),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(36, activation='softmax')  # Đầu ra từ 0-9 và các ký tự từ A-Z
])

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=optimizers.Adam(learning_rate=0.0001),
    metrics=[F1Score()]
)
model.summary()
```

Ta được mạng nơ-ron với các lớp:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	23,248
conv2d_1 (Conv2D)	(None, 28, 28, 32)	131,104
conv2d_2 (Conv2D)	(None, 28, 28, 64)	131,136
conv2d_3 (Conv2D)	(None, 28, 28, 64)	65,600
max_pooling2d (MaxPooling2D)	(None, 7, 7, 64)	0
dropout (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense_2 (Dense)	(None, 128)	401,536
dense_3 (Dense)	(None, 36)	4,644

Total params: 757,268 (2.89 MB)

Trainable params: 757,268 (2.89 MB)

Non-trainable params: 0 (0.00 B)

Hình 3.10. Mạng nơ-ron sử dụng trong quá trình huấn luyện

Tiếp đến ta đặt điểm dừng cho quá trình huấn luyện nếu val_custom_f1score > 0.99 thì ta sẽ dừng quá trình huấn luyện hoặc sẽ dừng nếu đạt đủ 20 epochs.

```
1. class stop_training_callback(tf.keras.callbacks.Callback):
2.     def on_epoch_end(self, epoch, logs=None):
3.         # Kiểm tra nếu 'val_custom_f1score' có trong logs
4.         if logs.get('val_custom_f1score') and
logs['val_custom_f1score'] > 0.99:
5.             print(f"Stopping training as val_custom_f1score >
0.99 at epoch {epoch}")
```

```
            self.model.stop_training = True
# Huấn luyện mô hình
batch_size = 32
callbacks = [stop_training_callback()]
history = model.fit(
      train_generator,
      steps_per_epoch = train_generator.samples // batch_size,
      validation_data = validation_generator,
      validation_steps = validation_generator.samples //
batch_size,
      epochs = 20,
      verbose = 1,
      callbacks=[stop_training_callback()])

```

```
Epoch 1/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 261ms/step - f1_score: 0.8929 - loss: 0.0273 - val_f1_score: 1.0000
Epoch 2/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 245ms/step - f1_score: 1.0000 - loss: 0.0330 - val_f1_score: 1.0000
Epoch 3/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 259ms/step - f1_score: 0.9286 - loss: 0.0528 - val_f1_score: 1.0000
Epoch 4/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 313ms/step - f1_score: 0.9643 - loss: 0.0267 - val_f1_score: 1.0000
Epoch 5/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 305ms/step - f1_score: 1.0000 - loss: 0.0424 - val_f1_score: 1.0000
Epoch 6/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 300ms/step - f1_score: 0.9286 - loss: 0.0427 - val_f1_score: 1.0000
Epoch 7/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 269ms/step - f1_score: 0.9643 - loss: 0.0361 - val_f1_score: 1.0000
Epoch 8/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 280ms/step - f1_score: 0.9643 - loss: 0.0265 - val_f1_score: 1.0000
Epoch 9/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 264ms/step - f1_score: 1.0000 - loss: 0.0298 - val_f1_score: 1.0000
Epoch 10/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 266ms/step - f1_score: 1.0000 - loss: 0.0361 - val_f1_score: 1.0000
Epoch 11/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 287ms/step - f1_score: 1.0000 - loss: 0.0251 - val_f1_score: 1.0000
Epoch 12/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 7s 255ms/step - f1_score: 0.9286 - loss: 0.0361 - val_f1_score: 1.0000
Epoch 13/20
...
Epoch 19/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 279ms/step - f1_score: 0.9643 - loss: 0.0512 - val_f1_score: 1.0000
Epoch 20/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 279ms/step - f1_score: 0.9643 - loss: 0.0284 - val_f1_score: 1.0000
```

Hình 3.11. Quá trình huấn luyện

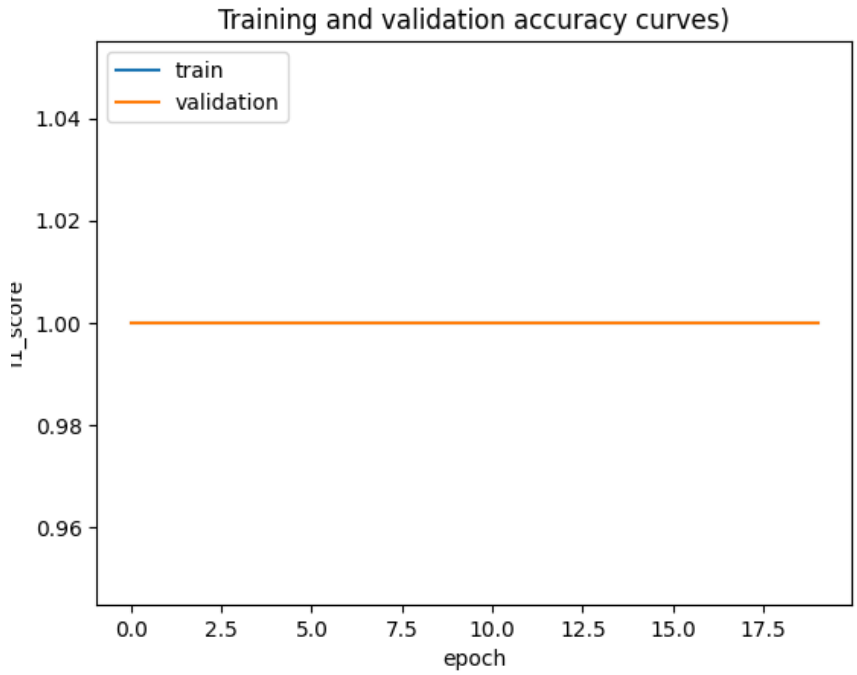
Đánh giá mô hình

Thiết kế của mô hình sử dụng bao gồm: các lớp tích chập (Conv2d) và max_pooling2d để trích xuất đặc trưng từ ảnh đầu vào, sau đó là các lớp fully connected (dense) để học các mẫu phức tạp tạp hơn. Cấu trúc cụ thể như sau:

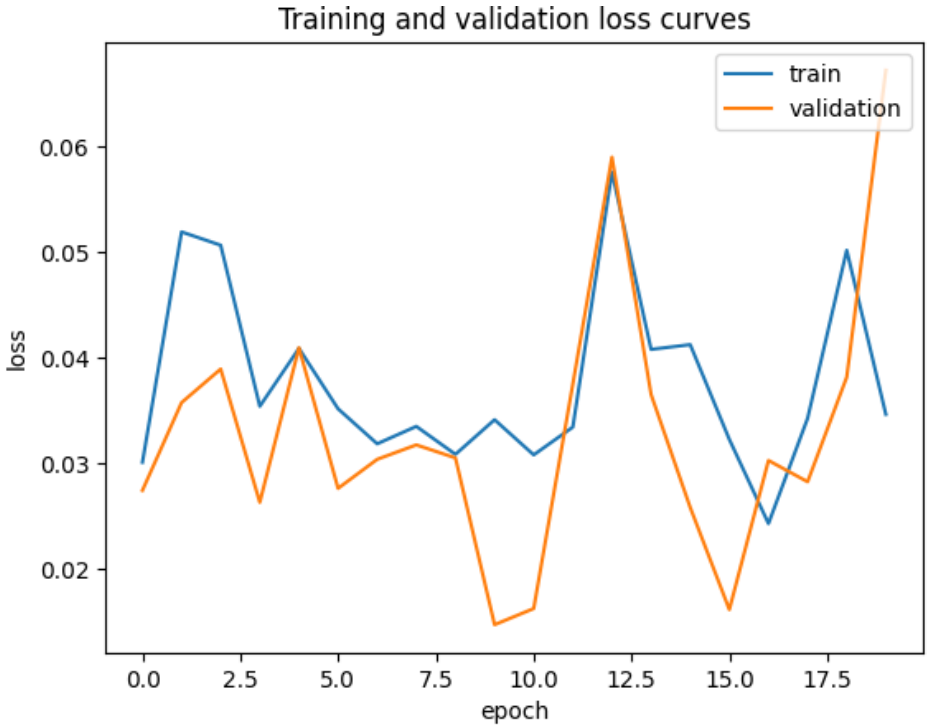
– Lớp tích chập đầu tiên (Conv2D): Sử dụng 16 bộ lọc với kích thước kernel 22×22, đầu vào có kích thước (28, 28, 3), đầu ra có kích thước

(28, 28, 16) do sử dụng padding “same”. Tổng số tham số là (22×22×3 + 1) × 16 = 23,056.

- Lớp tích chập thứ hai (Conv2D): Sử dụng 32 bộ lọc với kernel 16×16, đầu vào (28, 28, 16), đầu ra (28, 28, 32). Tổng số tham số là (16×16×16 + 1) × 32 = 131,104.
- Lớp tích chập thứ ba (Conv2D): Sử dụng 64 bộ lọc với kernel 8×8, đầu vào (28, 28, 32), đầu ra (28, 28, 64). Tổng số tham số là (8×8×32 + 1) × 64 = 131,136.
- Lớp tích chập thứ tư (Conv2D): Sử dụng 64 bộ lọc với kernel 4×4, đầu vào (28, 28, 64), đầu ra (28, 28, 64). Tổng số tham số là (4×4×64 + 1) × 64 = 65,600.
- Lớp gộp (Max Pooling 2D): Sử dụng kích thước pool 4×4, đầu vào (28, 28, 64), đầu ra (7, 7, 64). Không có tham số học.
- Lớp Dropout: Với tỷ lệ 0.4, giữ nguyên kích thước đầu vào. Không có tham số học.
- Lớp làm phẳng (Flatten): Chuyển đầu vào từ (7, 7, 64) thành vector 1 chiều có 3136 phần tử.
- Lớp Dense đầu tiên: Gồm 128 nút, đầu vào 3136, tổng số tham số là 3136 × 128 + 128 = 401,856.
- Lớp Dense cuối cùng: Gồm 36 nút (ứng với 36 lớp đầu ra), tổng số tham số là 128 × 36 + 36 = 4,644.



Hình 3.12. Biểu đồ Accuracy giữa train và validation của mô hình CNN



Hình 3.13. Biểu đồ hàm mất mát giữa train và validation của mô hình CNN

Tổng quan có thể thấy, mô hình đạt được độ chính xác rất cao trên cả tập huấn luyện:

- Độ chính xác huấn luyện (*train*) và kiểm định (*validation*) hội tụ ổn định quanh mức ~0.99 sau epoch 10.
- Không có dấu hiệu overfitting (khoảng cách giữa hai đường ≤ 0.01).
- Xu hướng giảm loss:
  - Loss huấn luyện giảm mạnh từ 0.06 → 0.02 (epoch 0 → 5).
  - Loss kiểm định hội tụ song song ở mức ~0.02 sau epoch 7.5 → Cân bằng tối ưu.
- Sai số kiểm định dao động nhẹ (±0.005) từ epoch 10 → 17.5 → Mô hình đạt ngưỡng tối ưu.

3.2.3. Tích hợp hệ thống.

Với 2 mô hình được huấn luyện có độ chính xác khá cao ở trên, bây giờ điều cần làm là đưa 2 model vào cùng một pipeline hoàn chỉnh tự động từ ảnh đầu vào đến chuỗi ký tự biển số.

Chuẩn bị tham số và cài đặt chung

```
1. import cv2
2. import numpy as np
3. import matplotlib.pyplot as plt
4. from ultralytics import YOLO
5. yolo_detect = YOLO('..//CODE_YOLO/runs/detect/yolo-plates-
custom/weights/best.pt')
6. dsize = (30, 80)
7. threshold_val = 110 # 62.5%
8. blur_size = (3, 3)
9. predict_dsize = (500, 390)
```

Xử lý ảnh (Pre-processing)

```
1. def maximizeContrast(img):
2.     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
3.     height, width = gray.shape
4.     imgTopHat = np.zeros((height, width, 1), np.uint8)
5.     imgBlackHat = np.zeros((height, width, 1), np.uint8)
6.     structuringElement =
cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))  # tạo bộ lọc
kernel
7.     imgTopHat = cv2.morphologyEx(gray, cv2.MORPH_TOPHAT,
structuringElement,
8.                                  iterations=10)  # nổi bật chi
tiết sáng trong nền tối
```

```
    imgBlackHat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT,
structuringElement,
                                   iterations=10)  # Nổi bật chi
tiết tối trong nền sáng
    imgGrayscalePlusTopHat = cv2.add(gray, imgTopHat)
    imgGrayscalePlusTopHatMinusBlackHat =
cv2.subtract(imgGrayscalePlusTopHat, imgBlackHat)
    return imgGrayscalePlusTopHatMinusBlackHat
```

Trích xuất đặc trưng HOG (Histogram of Oriented Gradients) được thiết lập 12 hướng gradient, cell 14×14, block 1×1.

```
from skimage.feature import hog
#Trich xuat đặt trưng
def make_feature(img):
    return hog(
        img, orientations=12,
        pixels_per_cell=(14, 14),
        cells_per_block=(1, 1),
        block_norm="L2"
    )

def imshow(img):
    plt.figure(figsize=(5, 3))
    plt.imshow(img)
    plt.show()
```

Phát hiện biển số (License-Plate Detection)

```
def detect_lp(path):
    list_plate = []
    plate_types = []
    img = cv2.imread(path)
    results = yolo_detect.predict(source=img, conf=0.25)

    print(f"Đã dự đoán xong, tìm thấy {len(results[0].boxes)} đối
tượng")

    for result in results:
        boxes = result.boxes.xyxy.tolist()
        classes = result.boxes.cls.tolist()
        names = result.names
        confidences = result.boxes.conf.tolist()

        for box, cls, conf in zip(boxes, classes, confidences):
            if conf > 0.25:
                x1, y1, x2, y2 = map(int, box)
                plate = img[y1:y2, x1:x2]
                cls_name = names[int(cls)]

                list_plate.append(plate)
```

```
                plate_types.append(cls_name)

                # In thông tin chi tiết về đối tượng phát hiện được
                print(f"Phát hiện biển số: {cls_name}, độ tin cậy: {conf:.2f}")

    return list_plate, plate_types
plate_img, plate_types = detect_lp('../test.jpg')
print("Detected plate types:", plate_types)

for i, plate in enumerate(plate_img):
    plt.figure(figsize=(8, 4))
    plt.imshow(plate)
    plt.title(f"License Plate {i+1} - Type: {plate_types[i]}")
    plt.show()
plt.show()  # Hiển thị biểu đồ
```

```
0: 576×640 1 BSV, 72.5ms
Speed: 5.8ms preprocess, 72.5ms inference, 2.2ms postprocess per image at shape (1, 3, 576, 640)
Đã dự đoán xong, tìm thấy 1 đối tượng
Phát hiện biển số: BSV, độ tin cậy: 0.96
Detected plate types: ['BSV']
```

Hình 3.14. Phát hiện biển số

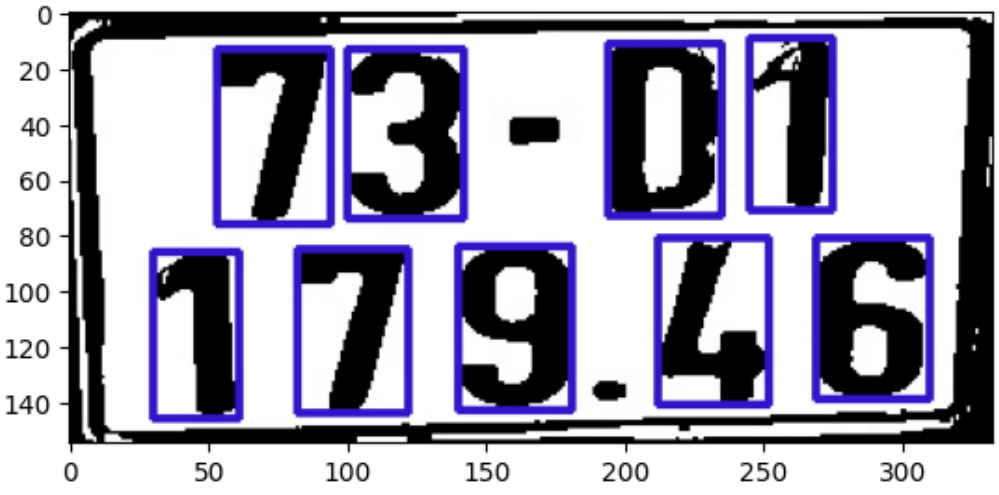
Phân đoạn ký tự theo dạng nhị phân + morphology (erode/dilate) để giảm nhiễu, tính giới hạn kích thước contour dựa trên tỉ lệ biển. Sau đó lọc contour theo diện tích → chuẩn hóa 24×44 + padding và sắp xếp ký tự theo toạ độ X.

```
# Tìm ký tự số và cắt ra - cập nhật để trả về thông tin vị trí
def find_contours(dimensions, img) :

    # Tìm tất cả đường viền
    cntrs, _ = cv2.findContours(img.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
 6.
 7.      lower_width = dimensions[0]
 8.      upper_width = dimensions[1]
 9.      lower_height = dimensions[2]
10.      upper_height = dimensions[3]
11.
12.      # Check largest 5 or  15 contours for license plate or
character respectively
13.      cntrs = sorted(cntrs, key=cv2.contourArea,
reverse=True)[:15]
14.
15.      ii = cv2.imread('contour.jpg')
16.
17.      x_cntr_list = []
18.      y_cntr_list = []  # Thêm danh sách lưu tọa độ y
19.      target_contours = []
20.      img_res = []
21.      char_positions = []  # Lưu thông tin vị trí của từng ký tự
22.
23.      for cntr in cntrs :
24.          # detects contour in binary image and returns the
coordinates of rectangle enclosing it
25.          intX, intY, intWidth, intHeight = cv2.boundingRect(cntr)
26.
27.          # checking the dimensions of the contour to filter out
the characters by contour's size
28.          if intWidth > lower_width and intWidth < upper_width and
intHeight > lower_height and intHeight < upper_height :
29.              x_cntr_list.append(intX) #stores the x coordinate of
the character's contour
30.              y_cntr_list.append(intY) #stores the y coordinate of
the character's contour
31.
32.              char_copy = np.zeros((44,24))
33.              # extracting each character using the enclosing
rectangle's coordinates.
34.              char = img[intY:intY+intHeight, intX:intX+intWidth]
35.              char = cv2.resize(char, (20, 40))
36.
37.              cv2.rectangle(ii, (intX,intY), (intWidth+intX,
intY+intHeight), (50,21,200), 2)
38.              plt.imshow(ii, cmap='gray')
39.
40.              # Make result formatted for classification: invert
colors
41.              char = cv2.subtract(255, char)
42.
43.              # Resize the image to 24x44 with black border
44.              char_copy[2:42, 2:22] = char
45.              char_copy[0:2, :] = 0
46.              char_copy[:, 0:2] = 0
```

```
 47.             char_copy[42:44, :] = 0
 48.             char_copy[:, 22:24] = 0
 49.
 50.             img_res.append(char_copy) # List that stores the
character's binary image (unsorted)
 51.             char_positions.append((intX, intY, intWidth,
intHeight))  # Lưu vị trí
 52.
 53.     # Return characters on ascending order with respect to the
x-coordinate (most-left character first)
 54.     # In ra số lượng contour tìm được
 55.     print("Số contour tìm được:", len(cntrs))
 56.
 57.     # In ra kích thước các contour
 58.     for i, cntr in enumerate(cntrs):
 59.         x, y, w, h = cv2.boundingRect(cntr)
 60.         print(f"Contour {i}: width={w}, height={h}")
 61.
 62.     # In ra ngưỡng dimensions
 63.     print("Ngưỡng dimensions:", dimensions)
 64.     plt.show()
 65.
 66.     # Sắp xếp theo x coordinate và giữ thông tin vị trí
 67.     indices = sorted(range(len(x_cntr_list)), key=lambda k:
x_cntr_list[k])
 68.     img_res_copy = []
 69.     positions_copy = []
 70.     for idx in indices:
 71.         img_res_copy.append(img_res[idx])
 72.         positions_copy.append(char_positions[idx])
 73.
 74.     img_res = np.array(img_res_copy)
 75.     print(f"Đã phân đoạn {len(img_res)} ký tự")
 76.
 77.     return img_res, positions_copy
 78. def segment_characters(image) :
 79.
 80.     # Preprocess cropped license plate image
 81.     img_lp = cv2.resize(image, (333, 155))
 82.     img_gray_lp = cv2.cvtColor(img_lp, cv2.COLOR_BGR2GRAY)
 83.     _, img_binary_lp = cv2.threshold(img_gray_lp, 200, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)
 84.     img_binary_lp = cv2.erode(img_binary_lp, (3,3))
 85.     img_binary_lp = cv2.dilate(img_binary_lp, (3,3))
 86.
 87.     LP_WIDTH = img_binary_lp.shape[0]
 88.     LP_HEIGHT = img_binary_lp.shape[1]
 89.
 90.     # Estimations of character contours sizes of cropped license
plates
 91.     dimensions = [LP_WIDTH/7,
```

```
                          LP_WIDTH/2,
                          LP_HEIGHT/10,
                          LP_HEIGHT/3]
    plt.imshow(img_binary_lp, cmap='gray')
    plt.show()
    cv2.imwrite('contour.jpg',img_binary_lp)

    # Get contours within cropped license plate with position info
    char_list, char_positions = find_contours(dimensions, img_binary_lp)

    return char_list, char_positions
# Lấy ảnh biển số và loại biển số
plate_img, plate_types = detect_lp('../20.jpg')

# Xử lý từng biển số riêng biệt
all_chars_per_plate = []
all_positions_per_plate = []

for i, plate in enumerate(plate_img):
    chars, positions = segment_characters(plate)
    all_chars_per_plate.append(chars)
    all_positions_per_plate.append(positions)
    print(f"Biển số {i+1} ({plate_types[i]}): phân đoạn được {len(chars)} ký tự")

print("Tổng số biển số phát hiện:", len(plate_img))
```




```
Đã phân đoạn 9 ký tự
Biển số 1 (BSV): phân đoạn được 9 ký tự
Tổng số biển số phát hiện: 1
```

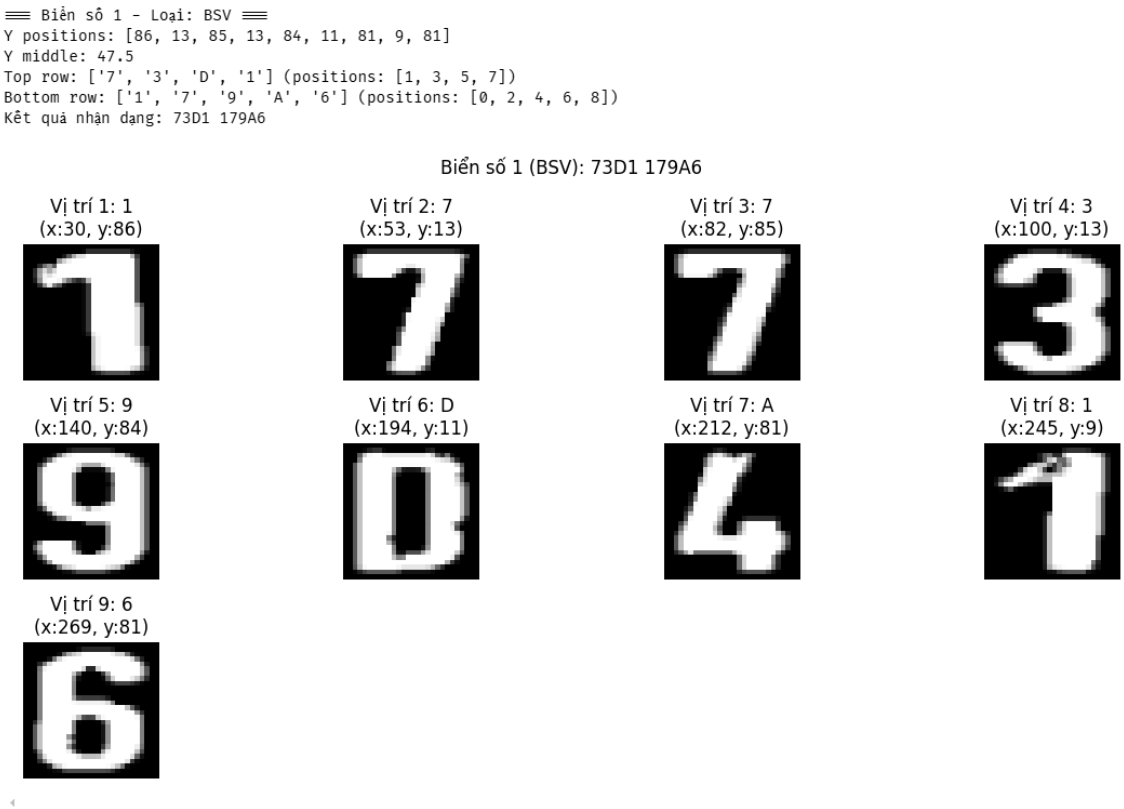
Hình 3.15. Kết quả sau khi phân đoạn

Nhận diện ký tự (Character Recognition): Chuẩn hóa ảnh đầu vào 28×28×3, dùng softmax → argmax → ánh xạ ký tự. Bên cạnh đó cần xử lý các trường hợp biển 2 hàng và 1 hàng, biển 4 số và biển 5 số. Phân chia dựa trên vị trí tọa độ y (nửa trên/nửa dưới)

```
# Gọi lại mô hình
from keras.models import load_model
model = load_model('../CODE_CNN/my_model.h5')
import numpy as np

# Hàm để chuẩn hóa ảnh về đúng kích thước
def fix_dimension(img):
    new_img = np.zeros((28,28,3))
    for i in range(3):
        new_img[:,:,i] = img
    return new_img

# Hàm để dự đoán kết quả với xử lý sắp xếp theo loại biển số
def show_results(characters, char_positions=None, plate_type="BSD"):
    dic = {}
    characters_list = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    for i, c in enumerate(characters_list):
        dic[i] = c

    output = []
    for ch in characters:
        img_ = cv2.resize(ch, (28,28), interpolation=cv2.INTER_AREA)
        img = fix_dimension(img_)
        img = img.reshape(1, 28, 28, 3)

        y_ = model.predict(img, verbose=0)
        y_ = np.argmax(y_, axis=1)
        character = dic[y_[0]]
        output.append(character)  # Lưu kết quả vào danh sách

    # Xử lý sắp xếp theo loại biển số
    if plate_type == "BSV" and char_positions is not None and len(output) >= 6:  # Biển số 2 hàng
        # Tính toán điểm giữa theo trục y để phân chia nửa trên/nửa dưới
        y_positions = [pos[1] for pos in char_positions]
        y_min, y_max = min(y_positions), max(y_positions)
        y_middle = (y_min + y_max) / 2

        print(f"Y positions: {y_positions}")
        print(f"Y middle: {y_middle}")
```

```
 40.
 41.         # Phân loại ký tự theo hàng dựa trên vị trí y
 42.         top_chars = []  # Hàng trên
 43.         bottom_chars = []  # Hàng dưới
 44.
 45.         for i, (char, pos) in enumerate(zip(output,
char_positions)):
 46.             y_pos = pos[1]
 47.             x_pos = pos[0]
 48.             if y_pos < y_middle:
 49.                 top_chars.append((char, x_pos, i))
 50.             else:
 51.                 bottom_chars.append((char, x_pos, i))
 52.
 53.         # Sắp xếp từng hàng theo tọa độ x (từ trái sang phải)
 54.         top_chars.sort(key=lambda x: x[1])
 55.         bottom_chars.sort(key=lambda x: x[1])
 56.
 57.         print(f"Top row: {[c[0] for c in top_chars]} (positions:
{[c[2] for c in top_chars]})")
 58.         print(f"Bottom row: {[c[0] for c in bottom_chars]}
(positions: {[c[2] for c in bottom_chars]})")
 59.
 60.         # Tạo chuỗi kết quả
 61.         top_line = ''.join([c[0] for c in top_chars])
 62.         bottom_line = ''.join([c[0] for c in bottom_chars])
 63.
 64.         plate_number = top_line + ' ' + bottom_line
 65.     else:
 66.         # Biển số 1 hàng hoặc không có thông tin vị trí
 67.         plate_number = ''.join(output)
 68.
 69.     return plate_number
 70. import math
 71.
 72. # Xử lý và hiển thị kết quả cho từng biển số
 73. for plate_idx, (chars, positions, plate_type) in
enumerate(zip(all_chars_per_plate, all_positions_per_plate,
plate_types)):
 74.     if len(chars) == 0:
 75.         print(f"Biển số {plate_idx + 1}: Không phân đoạn được ký
tự nào")
 76.         continue
 77.
 78.     print(f"\n=== Biển số {plate_idx + 1} - Loại: {plate_type}
===")
 79.
 80.     # Dự đoán kết quả cho biển số này
 81.     predicted_result = show_results(chars, positions,
plate_type)
 82.     print(f"Kết quả nhận dạng: {predicted_result}")
```

```
 83.
 84.     # Hiển thị các ký tự đã phân đoạn
 85.     plt.figure(figsize=(12, 6))
 86.
 87.     # Tính toán số hàng và số cột của subplot
 88.     rows = math.ceil(len(chars) / 4)
 89.     cols = min(4, len(chars))
 90.
 91.     for i, ch in enumerate(chars):
 92.         img = cv2.resize(ch, (28,28),
interpolation=cv2.INTER_AREA)
 93.         plt.subplot(rows, cols, i + 1)
 94.         plt.imshow(img, cmap='gray')
 95.
 96.         # Dự đoán từng ký tự để hiển thị
 97.         img_fixed = fix_dimension(img)
 98.         img_reshaped = img_fixed.reshape(1, 28, 28, 3)
 99.         y_pred = model.predict(img_reshaped, verbose=0)
100.         y_pred = np.argmax(y_pred, axis=1)
101.
102.         dic = {}
103.         characters_list = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
104.         for j, c in enumerate(characters_list):
105.             dic[j] = c
106.         character = dic[y_pred[0]]
107.
108.         # Hiển thị thông tin vị trí
109.         pos_info = f"(x:{positions[i][0]}, y:{positions[i][1]})"
if i < len(positions) else ""
110.         plt.title(f'Vị trí {i+1}: {character}\n{pos_info}')
111.         plt.axis('off')
112.
113.     plt.suptitle(f'Biển số {plate_idx + 1} ({plate_type}):
{predicted_result}')
114.     plt.tight_layout()
115.     plt.show()
```



Hình 3.16. Nhận diện ký tự

3.3. Đánh giá kết quả.

Dựa trên quy trình thực nghiệm và các kết quả được trình bày, hệ thống nhận diện biển số xe đã xây dựng thể hiện hiệu năng rất cao trong môi trường được kiểm soát. Việc đánh giá sẽ được chia thành ba phần.

3.3.1. Đánh giá Mô hình Phát hiện Biển số (YOLOv8).

Mô hình YOLOv8 được huấn luyện để phát hiện vùng chứa biển số đã đạt được kết quả tốt với độ chính xác cao: Các chỉ số đánh giá trên tập validation là minh chứng rõ ràng: mAP@0.5 = 0.994: Cho thấy ở ngưỡng IoU (Intersection over Union) là 0.5, mô hình gần như xác định hoàn hảo tất cả các biển số trong tập dữ liệu kiểm thử. Điều này có nghĩa là các hộp giới hạn (bounding box) dự đoán có độ chồng lấp rất tốt so với nhãn thực tế, mAP@0.5:0.95 = 0.912: Chỉ số này quan trọng hơn vì nó đánh giá mô hình ở nhiều ngưỡng IoU khắt khe hơn (từ 0.5 đến 0.95). Việc đạt được mAP trên 0.9

cho thấy mô hình không chỉ phát hiện được biển số mà còn khoanh vùng với độ chính xác rất cao về mặt vị trí. Precision = 0.994 và F1-Score tối ưu ở mức 0.99: Điều này khẳng định rằng mô hình rất hiếm khi đưa ra dự đoán sai (False Positive) và đạt được sự cân bằng gần như tuyệt đối giữa độ chính xác và độ bao phủ (Recall).

Tốc độ xử lý nhanh: Với thời gian suy luận trung bình 5.8 ms trên GPU GTX 1650, mô hình hoàn toàn đáp ứng yêu cầu của các ứng dụng thời gian thực như giám sát giao thông hay kiểm soát bãi đỗ xe.

Sự ổn định: Các biểu đồ Precision-Recall, F1-Confidence cho thấy mô hình hoạt động ổn định trên các ngưỡng tin cậy khác nhau. Đặc biệt, việc Precision đạt 1.0 khi confidence > 0.981 có nghĩa là khi mô hình "tự tin" về một dự đoán, dự đoán đó gần như chắc chắn đúng.

Về YOLOv8: Mô hình phát hiện biển số là một thành công lớn, tạo ra một nền tảng vững chắc cho các bước xử lý sau. Độ chính xác cao và tốc độ nhanh đảm bảo rằng đầu vào cho giai đoạn nhận diện ký tự có chất lượng tốt và đáng tin cậy.

3.3.2. Đánh giá Mô hình Nhận diện Ký tự (CNN).

Mô hình CNN được huấn luyện để phân loại từng ký tự cũng cho thấy kết quả rất khả quan trên tập dữ liệu đã sử dụng với hiệu suất học tập xuất sắc: Các biểu đồ Accuracy và Loss cho thấy:

- Độ chính xác trên cả tập huấn luyện (train) và kiểm định (validation) đều hội tụ ổn định ở mức rất cao (~0.99).
- Khoảng cách giữa đường train và validation rất nhỏ, cho thấy mô hình không bị overfitting. Nó đã học được các đặc trưng tổng quát của dữ liệu thay vì chỉ "ghi nhớ" tập huấn luyện.

Kiến trúc hợp lý: Mô hình sử dụng một chuỗi các lớp Conv2D để trích xuất đặc trưng theo cấp bậc, kết hợp với MaxPooling để giảm chiều dữ liệu và

Dropout để chống overfitting. Lớp cuối cùng với 36 nơ-ron và hàm softmax là thiết kế tiêu chuẩn và hiệu quả cho bài toán phân loại đa lớp này.

Tuy nhiên bên cạnh đó còn những hạn chế vì sự phụ thuộc vào dữ liệu tổng hợp (Synthetic Data) là hạn chế lớn nhất của mô hình này. Bộ dữ liệu "Standard OCR Dataset" được tạo ra từ font chữ máy tính và thêm nhiễu. Mặc dù rất tốt cho việc chứng minh khái niệm (proof-of-concept), nó không thể phản ánh hết sự đa dạng và phức tạp của ký tự trên biển số xe thực tế, vốn bị ảnh hưởng bởi:

- Méo mó, biến dạng do góc chụp.
- Mòn, bẩn, trầy xước.
- Điều kiện ánh sáng không đồng đều (lóa, bóng đổ).
- Các loại font chữ khác nhau trên biển số.
- Do đó, độ chính xác ~99% trên tập dữ liệu này có thể sẽ giảm đáng kể khi áp dụng trên ảnh thực tế.

Kiến trúc lớp Conv2D: Việc sử dụng các kernel có kích thước rất lớn (22x22, 16x16) trên ảnh đầu vào 28x28 là một lựa chọn thiết kế không phổ biến. Thông thường, các kernel nhỏ (3x3, 5x5) được xếp chồng lên nhau sẽ hiệu quả hơn trong việc học các đặc trưng phức tạp và có ít tham số hơn. Mặc dù mô hình vẫn đạt kết quả tốt trên dữ liệu này, đây có thể không phải là kiến trúc tối ưu nhất.

Về CNN: Trong phạm vi bộ dữ liệu được sử dụng, mô hình CNN đã học rất tốt và đạt độ chính xác gần như tuyệt đối. Tuy nhiên, hiệu suất thực tế của nó sẽ phụ thuộc rất nhiều vào chất lượng của giai đoạn tiền xử lý và phân đoạn ký tự, cũng như khả năng tổng quát hóa trên dữ liệu thực.

3.3.3. Đánh giá Hệ thống Tích hợp và Toàn cục

Khi kết hợp YOLOv8 và CNN, hệ thống đã tạo thành một pipeline hoàn chỉnh phân đoạn ký tự hệ thống đã tự động hóa toàn bộ quy trình từ ảnh đầu vào đến chuỗi ký tự đầu ra. Xử lý được các loại biển số khác nhau: Logic phân

chia ký tự thành hàng trên và hàng dưới cho biển số vuông (BSV) dựa trên tọa độ Y là một bước xử lý thông minh và cần thiết, cho thấy hệ thống đã tính đến các trường hợp thực tế.

Những hạn chế và rủi ro còn gặp phải điển hình tính mong manh của khâu phân đoạn ký tự (Character Segmentation): Đây là gót chân Achilles của toàn bộ hệ thống. Quy trình này dựa trên các kỹ thuật xử lý ảnh cổ điển (nhị phân hóa, morphology, tìm contour) và các ngưỡng được định sẵn. Nó cực kỳ nhạy cảm với:

- Chất lượng ảnh đầu vào: Nếu biển số bị lóa, mờ, hoặc quá tối, quá trình nhị phân hóa có thể thất bại, dẫn đến việc không tìm thấy contour hoặc contour bị sai.
- Ký tự dính liền hoặc bị gãy: Các phép erode/dilate có thể vô tình làm gãy một ký tự hoặc nối hai ký tự lại với nhau. Khi đó, find_contours sẽ nhận diện sai.
- Các ngưỡng cứng (Hard-coded Thresholds): Việc lọc contour dựa trên các dimensions được tính toán theo tỷ lệ của ảnh biển số là một giả định cứng nhắc. Nếu YOLOv8 trả về một bounding box không hoàn toàn chính xác (ví dụ, bị lệch hoặc bao gồm cả một phần viền xe), các tỷ lệ này sẽ sai, dẫn đến việc bỏ sót ký tự. Trong ví dụ, mã nguồn đã in ra "Số contour tìm được: 10" nhưng chỉ phân đoạn được 9 ký tự, cho thấy một contour đã bị loại bỏ do không thỏa mãn điều kiện kích thước.

Sự lan truyền lỗi (Error Propagation): Bất kỳ một lỗi nhỏ ở giai đoạn đầu cũng sẽ khuếch đại ở các giai đoạn sau:

- YOLO sai -> Kích thước biển số sai -> Ngưỡng dimensions sai -> Phân đoạn sai.
- Phân đoạn sai (ví dụ, cắt thiếu một phần của ký tự "8" làm nó trông giống "3") -> CNN nhận diện sai.

- Phân đoạn sai (ví dụ, gộp “1” và “1” thành một contour) -> CNN không thể nhận diện -> Kết quả cuối cùng thiếu ký tự.

Hệ thống được xây dựng là một minh chứng xuất sắc về khả năng ứng dụng của học sâu, với mô hình YOLOv8 đạt hiệu năng vượt trội, và mô hình CNN cho thấy khả năng học tập tốt trên dữ liệu lý tưởng. Tuy nhiên, độ tin cậy của toàn bộ hệ thống trong thực tế bị ảnh hưởng nghiêm trọng bởi sự mong manh của khâu phân đoạn ký tự dựa trên thuật toán cổ điển.

Định hướng cải thiện mô hình trong tương lai với việc nâng cấp bộ dữ liệu OCR xây dựng một bộ dữ liệu nhận diện ký tự từ các biển số xe thực tế của Việt Nam. Điều này sẽ giúp mô hình CNN học được cách đối phó với các biến thể trong đời thực và cải thiện đáng kể khả năng tổng quát hóa. Cải thiện hoặc thay thế khâu phân đoạn ký tự bằng việc loại bỏ hoàn toàn bước phân đoạn thủ công. Thay vào đó, sử dụng một mô hình học sâu khác cho nhiệm vụ nhận diện chuỗi ký tự trực tiếp, chẳng hạn như kiến trúc CRNN (Convolutional Recurrent Neural Network) kết hợp với CTC Loss. Mô hình này có thể nhận diện toàn bộ chuỗi ký tự từ ảnh biển số mà không cần phải cắt ra từng ký tự, do đó loại bỏ được mắt xích yếu nhất trong pipeline hiện tại.

KẾT LUẬN

Qua quá trình thực hiện đề tài "Nghiên cứu và xây dựng hệ thống nhận diện biển số xe Việt Nam", nhóm em không chỉ có cơ hội áp dụng các kiến thức lý thuyết đã học vào một bài toán thực tế mà còn tích lũy thêm nhiều kỹ năng quan trọng như: tư duy thuật toán, xử lý dữ liệu, lập trình hướng dự án, làm việc nhóm và giải quyết vấn đề. Đề tài cũng giúp nhóm tiếp cận gần hơn với những công nghệ hiện đại như thị giác máy tính, mạng nơ-ron tích chập (CNN), học sâu – những lĩnh vực đang rất được quan tâm hiện nay. Mặc dù còn nhiều khó khăn và hạn chế trong quá trình thực hiện – đặc biệt là ở giai đoạn tiền xử lý và tối ưu nhận diện trong môi trường phức tạp – nhưng nhóm đã cố gắng hoàn thiện đề tài với tinh thần nghiêm túc, học hỏi và sáng tạo. Kết quả thu được bước đầu đã đáp ứng được mục tiêu đặt ra, mở ra nhiều tiềm năng để mở rộng trong tương lai.

Nhóm xin gửi lời cảm ơn sâu sắc tới Thạc sĩ Phạm Việt Anh – giảng viên hướng dẫn – đã tận tình định hướng, hỗ trợ nhóm cả về mặt kiến thức lẫn tinh thần trong suốt quá trình thực hiện đề tài. Những góp ý và chỉ dẫn của Thầy là động lực giúp nhóm em vượt qua khó khăn và hoàn thiện đề tài này. Cuối cùng, nhóm hy vọng sản phẩm của đề tài có thể được phát triển và ứng dụng thực tiễn, đồng thời là tiền đề cho những nghiên cứu sâu hơn trong lĩnh vực trí tuệ nhân tạo và thị giác máy trong tương lai.

TÀI LIỆU THAM KHẢO

[1]. Mitchell, T. M.: *Machine Learning*. McGraw-Hill, 1997.

[2]. Krizhevsky, A., Sutskever, I., Hinton, G. E.: *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems, 2012.

[3]. Duda, R. O., Hart, P. E., Stork, D. G.: *Pattern Classification*, 2nd Edition. Wiley, 2001.

[4]. Noda.live: https://noda.live/glossary/pattern-recognition, truy cập ngày 21/6/2025.

[5]. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: *You Only Look Once: Unified, Real-Time Object Detection*, 2016.

[6]. Viblo: *YOLOv1 – Nhận diện vật thể chỉ ngó 1 lần: Khởi nguồn*, https://viblo.asia/p/yolov1-nhan-dien-vat-the-chi-ngo-1-lan-khoi-nguon-aNj4v0b0L6r, truy cập ngày 21/6/2025.

[7]. Lubna, Mufti, N., Shah, S. A. A.: *Automatic Number Plate Recognition: A Detailed Survey of Relevant Algorithms*, 2021.

[8]. Saha, S.: *A Review on Automatic License Plate Recognition System*, 2019.

[9]. Thư viện Pháp Luật: https://thuvienphapluat.vn/phap-luat/ho-tro-phap-luat/mau-bien-so-xe-may-chinh-thuc-tu-2025-mau-sac-seri-bien-so-xe-trong-nuoc-tu-2025-nhu-the-nao-869717-190701.html, truy cập ngày 21/6/2025.

[10]. Huytranvan2010: *mAP trong Object Detection*, https://huytranvan2010.github.io/mAP-object-detection/, truy cập ngày 21/6/2025.

[11]. YOLOv8.org: https://yolov8.org/, truy cập ngày 21/6/2025.

[12]. MMYOLO Documentation: https://mmyolo.readthedocs.io/en/latest/recommended_topics/algorithm_descriptions/yolov8_description.html, truy cập ngày 21/6/2025.

[13]. Roboflow Blog: *What is YOLOv8*, https://blog.roboflow.com/what-is-yolov8, truy cập ngày 21/6/2025.

[14]. Decoding CNNs: A Beginner's Guide to Convolutional Neural Networks and Their Applications," Medium, 2020. https://ravjot03.medium.com/decoding-cnns-a-beginners-guide-to-convolutional-neural-networks-and-their-applications-1a8806cbf536