

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

ĐỖ DUY ANH
PHẠM LÊ MINH KHÔI

KHÓA LUẬN TỐT NGHIỆP
XỬ LÝ ẢNH NHẬN DIỆN LỖI HƯ HỎNG CỦA CONTAINER
Container damage detection using image processing

CỬ NHÂN KỸ THUẬT MÁY TÍNH

TP. HỒ CHÍ MINH, 2025

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

ĐỖ DUY ANH – 21520558

PHẠM LÊ MINH KHÔI – 21521011

KHÓA LUẬN TỐT NGHIỆP
XỬ LÝ ẢNH NHẬN DIỆN LỖI HƯ HỎNG CỦA CONTAINER
Contaier damage detection using image processing

CỬ NHÂN KỸ THUẬT MÁY TÍNH

GIẢNG VIÊN HƯỚNG DẪN
ThS. NGUYỄN DUY XUÂN BÁCH
PGS.TS TRỊNH LÊ HUY

TP. HỒ CHÍ MINH, 2025

THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số **712/QĐ-ĐHCNTT** ngày 23 tháng 06 năm 2025. của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

LỜI CẢM ƠN

Lời nói đầu tiên, chúng em xin chân thành cảm ơn quý thầy cô khoa Kỹ thuật Máy tính của trường Đại học Công nghệ Thông tin, Đại học Quốc gia TP. Hồ Chí Minh đã tạo điều kiện để nhóm em có cơ hội được thực hiện khóa luận tốt nghiệp này. Tất cả quý thầy cô đều luôn tận tâm truyền đạt kiến thức, chỉ dẫn và động viên chúng em trong suốt quá trình học tập. Đây không chỉ là những kiến thức về chuyên môn, mà còn là hành trang quý báu chúng em được thầy cô trang bị, nhóm em rất cảm kích

Đặc biệt, chúng em xin bày tỏ lòng tri ân sâu sắc đến ThS. Nguyễn Duy Xuân Bách - giảng viên hướng dẫn, người đã luôn đồng hành, kiên nhẫn chỉ bảo, định hướng và truyền cảm hứng giúp chúng em hoàn thành luận văn.

Chúng em cũng xin gửi lời cảm ơn chân thành đến gia đình và bạn bè đã luôn ủng hộ, cổ vũ và tiếp thêm động lực cho chúng em trong suốt chặng đường này. Những người đã cung cấp cho nhóm em sự hỗ trợ vô điều kiện trong suốt chặng đường sinh viên qua.

Dù đã cố gắng hết sức, luận văn của chúng em khó tránh khỏi thiếu sót. Kính mong nhận được sự thông cảm và góp ý từ quý thầy cô để luận văn được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

Sinh viên thực hiện

Đỗ Duy Anh

Khoa Kỹ thuật máy tính – Lớp MTCL2021

Sinh viên thực hiện

Phạm Lê Minh Khôi

Khoa Kỹ thuật máy tính – Lớp MTCL2021

MỤC LỤC

Chương 1. GIỚI THIỆU VỀ ĐỀ TÀI.....	2
1.1. Mở đầu.....	2
1.1.1. Lý do chọn đề tài	2
1.1.2. Mục đích của đề tài	3
1.1.3. Đối tượng nghiên cứu của đề tài	3
1.1.4. Phạm vi nghiên cứu.....	3
1.2. TỔNG QUAN.....	4
1.2.1. Phân tích và đánh giá các hướng nghiên cứu đã có trong và ngoài nước....	4
1.2.2. Nghiên cứu trong nước	4
1.2.3. Nghiên cứu ngoài nước	4
1.2.4. Những vấn đề còn tồn tại.....	7
1.2.5. Những vấn đề đề tài cần tập trung và nghiên cứu giải quyết	8
Chương 2. NGHIÊN CỨU THỰC NGHIỆM HOẶC LÝ THUYẾT	9
2.1. Tổng quan về xử lý ảnh số (Image Processing).....	9
2.1.1. Tổng quan về Deep Learning và CNN:	9
2.2. YOLOv8	13
2.2.1. YOLO là gì ?.....	13
2.2.2. Cơ chế hoạt động của YOLO	14
2.2.3. Tổng quan YOLOv8	16
2.3. Nhận diện hư hỏng container:	21
2.3.1. Rỉ Sét (Rust).....	22
2.3.2. Vết Lõm (Dent).....	22
2.3.3. Lỗ Thủng (Hole).....	23
Chương 3. THIẾT KẾ HỆ THỐNG NHÚNG	25
3.1. Tổng quan hệ thống nhúng.....	25
3.2. Thành phần hệ thống	25
3.2.1. Vi điều khiển Raspberry Pi 4B	25
3.2.2. Camera Raspberry Pi V2 IMX219	27
3.2.3. Nguồn 5V/3A(bao gồm mạch sạc pin18650 có UPS 5V/3A và 2 pin 18650 Panasonic 3400mAh).....	28

3.3. Thiết bị và thành phần hỗ trợ	30
3.3.1. Các ứng dụng hỗ trợ	30
3.4. Mô hình hệ thống.....	32
3.4.1. Mô tả kết nối hệ thống	32
3.4.2. Lưu đồ giải thuật mô hình dự đoán lỗi hư hỏng container	33
Chương 4. PHƯƠNG PHÁP NGHIÊN CỨU VÀ THIẾT KẾ HỆ THỐNG.....	35
4.1. Kiến trúc tổng thể hệ thống	35
4.1.1. Tổng quan về Kiến trúc Hệ thống	36
4.1.2. Các Công nghệ và Công cụ được sử dụng	37
4.2. Thu thập và Xây dựng tập dữ liệu	38
4.2.1. Tổng quan về Tập dữ liệu.....	39
4.2.2. Quy trình Thu thập và Gán nhãn Dữ liệu	42
4.2.3. Thách thức trong Quá trình Gán nhãn và Giải pháp.....	45
4.2.4. Phân tích Thống kê Tập dữ liệu.....	46
4.3. Tiền xử lý dữ liệu và Tăng cường dữ liệu.....	48
4.3.1. Tiền xử lý dữ liệu.....	48
4.3.2. Các Phương pháp Tăng cường Dữ liệu (Data Augmentations)	49
4.4. Nhận dạng Ký tự Quang học (OCR) và Tích hợp.....	51
4.4.1. Tổng quan về OCR.....	51
4.4.2. Phương pháp OCR áp dụng.....	51
4.4.3. Vai trò của OCR trong hệ thống.....	52
4.5. Mô hình học sâu và Môi trường huấn luyện	53
4.5.1. Kiến trúc Mô hình và Lựa chọn YOLOv8m.pt.....	53
4.5.2. Môi trường Huấn luyện	54
4.5.3. Cấu hình và Siêu tham số Huấn luyện chi tiết.....	55
Chương 5. KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ	59
5.1. Hiện thực phần cứng	59
5.2. Kết quả Huấn luyện Mô hình	59
5.2.1. Quá trình Huấn luyện và Đường cong Mất mát (Loss Curves).....	59
5.2.2. Huấn luyện Bổ sung và Lựa chọn Mô hình cuối cùng	64
5.3. Đánh giá Mô hình Dự đoán	64

5.3.1. Các Chỉ số Đánh giá Hiệu suất (Evaluation Metrics)	64
5.3.2. Hiệu suất Phát hiện đối tượng trên Tập Xác thực và Kiểm tra.....	69
5.3.3. Ma trận Nhầm lẫn (Confusion Matrix).....	70
5.3.4. Đường cong Precision-Recall (PR Curve).....	72
5.3.5. Ảnh hưởng của Ngưỡng Tin cậy (Confidence Threshold) và IoU Threshold trong Post-training	74
5.3.6. So sánh hiệu suất mô hình với mô hình huấn luyện trên nền tảng Roboflow	77
5.4. Phân tích Định tính (Qualitative Analysis)	82
5.5. Triển khai trên website	84
5.5.1. Trang quản lý container	84
5.5.2. Quy trình thêm và chụp ảnh container	84
5.5.3. Hiển thị kết quả phát hiện lỗi và quản lý ảnh.....	86
Chương 6. KẾT LUẬN.....	89
6.1. Kết quả đạt được	89
6.2. Những điều đạt được khi thực hiện đề tài.....	90
6.3. Những khó khăn trong quá trình thực hiện đề tài	91
6.4. Hướng phát triển chính cho đề tài	91

DANH MỤC HÌNH

Hình 2.1: Kiến trúc mạng YOLO và cơ chế hoạt động.....	14
Hình 2.2: So sánh YOLOv8 với những kiến trúc mạng khác	16
Hình 2.3: Kiến trúc mô hình YOLOv8.	18
Hình 2.4: Bảng thông số 5 model của YOLOv8.....	21
Hình 3.1: Raspberry Pi 4 Model B.	26
Hình 3.2: Camera Raspberry Pi V2 IMX219.	27
Hình 3.3: Mạch sạc pin 18650 với UPS 5V/3A.....	29
Hình 3.4: Sơ đồ kết nối các thành phần của hệ thống.	33
Hình 3.5: Lưu đồ giải thuật mô hình dự đoán lỗi.....	33
Hình 4.1: Tỷ lệ phân chia dữ liệu trong nghiên cứu.....	41
Hình 4.2: Quá trình xử lý dữ liệu.....	42
Hình 4.3: Giao diện trên ROBOFLOW	43
Hình 4.4: Các tệp tọa độ dưới dạng .txt.....	44
Hình 4.5: Thông tin tọa độ của đối tượng trong file .txt.....	44
Hình 4.6: Tùy chọn hiển thị trong giao diện gắn nhãn của Roboflow	46
Hình 5.1: Hình ảnh bên ngoài và bên trong của phần cứng.....	59
Hình 5.2: Biểu đồ kết quả huấn luyện của mô hình YOLOv8m.pt.....	60
Hình 5.3: Biểu đồ đường cong F1-Confidence.....	60
Hình 5.4: Nhật ký huấn luyện chi tiết và kết quả đánh giá cuối cùng.	61
Hình 5.5: Kết quả sau khi train tiếp 28 epochs từ file best.pt.....	64
Hình 5.6: Hình minh họa predicted bounding box với ground-truth bounding box...	66
Hình 5.7: Precision-Recall Curve	67
Hình 5.8: Ma trận nhầm lẫn của mô hình trên tập xác thực.	70
Hình 5.9: Đường cong Precision-Recall cho từng lớp và tổng thể.	73
Hình 5.10: Biểu đồ hiệu suất huấn luyện của mô hình Roboflow.....	78
Hình 5.11: Biểu đồ đường cong mất mát của mô hình Roboflow.....	78

Hình 5.12: Biểu đồ hiệu suất huấn luyện nâng cao của mô hình Roboflow	78
Hình 5.13: Ví dụ về trường hợp phát hiện lỗi thành công (Vết rỉ sét rõ ràng).....	82
Hình 5.14: Ví dụ về trường hợp phát hiện lỗi thất bại (Lỗi nhỏ bị bỏ sót).	83
Hình 5.15: Giao diện Dashboard	84
Hình 5.16: Giao diện thêm mới một container.....	85
Hình 5.17: Chọn phiên sửa cho container.....	85
Hình 5.18: Chụp ảnh hư hỏng container	86
Hình 5.19: Giao diện hiển thị kết quả phát hiện lỗi trên ảnh.	87
Hình 5.20: Giao diện in hoá đơn sửa chữa	87
Hình 5.21: Giao diện danh sách container với ảnh minh họa.	88

DANH MỤC BẢNG

Bảng 1.1: Kết quả một số nghiên cứu về nhận diện lỗi hư hỏng của container.	6
Bảng 3.1: Thông số kỹ thuật của Raspberry Pi 4B.....	26
Bảng 3.2: Thông số kỹ thuật của Camera Raspberry Pi V2	28
Bảng 3.3: Thông số kỹ thuật của mạch sạc pin 18650 với UPS 5V/3A.....	29
Bảng 3.4: Bảng thông số kỹ thuật của Pin 18650.	30
Bảng 4.1: Các lỗi quan trọng của container trong tập dữ liệu.....	39
Bảng 4.2: Phân bố số lượng thể instances của các lớp lỗi trong các tập dữ liệu.....	46
Bảng 5.1: Tổng hợp các chỉ số mAP trên tập xác thực.	69
Bảng 5.2: Chi tiết Precision, Recall, mAP cho từng lớp trên tập xác thực	69
Bảng 5.3: Hiệu suất mô hình YOLOv8m.pt với các số conf khác nhau (IoU=0.6).75	
Bảng 5.4: So sánh giữa Mô hình Tự Huấn luyện và Mô hình Roboflow	79

DANH MỤC TỪ VIẾT TẮT

AI	Artificial Intelligence
API	Application Programming Interface
AP	Average Precision
ARM	Advanced RISC Machine
AUC	Area Under Curve
BIC Code	Bureau International des Containers
CNN	Convolutional Neural Network
COCO	Common Objects in
CPU	Central Processing Unit
CSI	Camera Serial Interface
DFL	Distribution Focal Loss
FN	False Negative
FP	False Positive
FPS	Frames Per Second

GFLOPS	Giga Floating Point Operations Per Second
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
HSV	Hue, Saturation, Value
IICL	Institute of International Container Lessors
IoU	Intersection over Union
LCD	Liquid Crystal Display
LPDDR4	Low Power Double Data Rate 4
mAP	Mean Average
MES	Manufacturing Execution System
MLOps	Machine Learning Operations
NAT	Network Address Translation
NMS	Non-Maximum Suppression
OCR	Optical Character Recognition
OS	Operating System

PC	Personal Computer
PR Curve	Precision-Recall Curve
RAM	Random Access Memory
RGB	Red, Green, Blue
RNN	Recurrent Neural Network
SCADA	Supervisory Control and Data Acquisition
SGD	Stochastic Gradient Descent
SSH	Secure Shell
SSD	Solid State Drive
TP	True Positive
VPN	Virtual Private Network
YOLO	You Only Look Once

TÓM TẮT KHÓA LUẬN

Đề tài của nhóm tập trung vào việc phát triển một hệ thống nhận diện lỗi hư hỏng container với chi phí tối ưu, sử dụng mô hình YOLOV8, tích hợp nền tảng web và ứng dụng di động để lưu trữ dữ liệu và ước tính chi phí sửa chữa. Hệ thống được thiết kế để phát hiện các lỗi phổ biến như thủng, rỉ sét, móp dựa trên tiêu chuẩn IICL (Institute of International Container Lessors). Hình ảnh container do nhân viên kỹ thuật chụp sẽ được truyền qua Raspberry Pi 4 đến máy chủ để phân tích, sau đó kết quả được gửi ngược về Raspberry Pi 4 và hiển thị trực quan cho người dùng.

Giải pháp này hướng tới các doanh nghiệp vừa và nhỏ trong lĩnh vực logistics, sở hữu các bãi container quy mô nhất định. Hệ thống giúp giảm tải công việc thủ công, nâng cao độ chính xác trong việc phát hiện lỗi và dự toán chi phí sửa chữa, từ đó hỗ trợ doanh nghiệp tối ưu hóa chiến lược kinh doanh một cách hiệu quả và tiết kiệm. Thay vì sử dụng các máy tính cấu hình cao tại từng bãi container, việc tận dụng Raspberry Pi 4 kết hợp với máy chủ đã giảm đáng kể chi phí đầu tư, giúp các doanh nghiệp dễ dàng tiếp cận công nghệ tiên tiến mà không phải chi trả mức giá đắt đỏ như các hệ thống hiện có tại các cảng container lớn ở Việt Nam.

Quá trình thu thập dữ liệu và huấn luyện mô hình YOLOV8 được thực hiện cẩn thận, đảm bảo hệ thống hoạt động hiệu quả trong nhiều điều kiện môi trường khác nhau. Sau khi hoàn thiện và thử nghiệm, hệ thống cho thấy sự ổn định với P ở mức 84,2% và R ở mức 71,1%, nhận diện chính xác các lỗi theo yêu cầu và thể hiện tiềm năng ứng dụng rộng rãi trong ngành logistics. Với những kết quả này, nhóm đánh giá rằng giải pháp sử dụng Raspberry Pi 4 là lựa chọn tối ưu để xây dựng một hệ thống nhận diện lỗi container chi phí thấp, đồng thời mở ra triển vọng nâng cấp và mở rộng trong tương lai.

Chương 1. GIỚI THIỆU VỀ ĐỀ TÀI

1.1. Mở đầu

1.1.1. Lý do chọn đề tài

Container là lớp giáp bảo vệ vững chắc cho hàng hóa, đóng vai trò không thể thiếu trong logistics hiện đại, là biểu tượng của sự kết nối thương mại toàn cầu. Tuy nhiên, trong đội tàu vận tải, nhiều container đã vượt ngưỡng tuổi thọ 20 năm, đặc biệt trên các tuyến thương mại dài hạn, thường rơi vào tình trạng xuống cấp, không còn duy trì trạng thái hoạt động tối ưu [1]. Sự khắc nghiệt của các hoạt động cơ học lặp lại, kết hợp với tác động không ngừng từ độ ẩm, nhiệt độ và muối biển, khiến container dễ chịu nhiều dạng hư hỏng, làm suy giảm chức năng, chất lượng, đồng thời đe dọa an toàn, tính ứng dụng và độ bền của cấu trúc. Bề mặt vỏ ngoài của container thường gặp những lỗi có thể nhìn thấy như thủng, rỉ sét, móp, lõm [2]. Theo thời gian, mức độ xuống cấp càng nghiêm trọng, đặt ra bài toán cấp bách về đảm bảo chất lượng container trong chuỗi cung ứng. Các phương pháp kiểm tra ở thời điểm hiện tại tại các cảng biển và bãi container không chỉ tốn thời gian, nhân lực mà còn tiềm ẩn nguy cơ bỏ sót hư hỏng nghiêm trọng, ảnh hưởng đến an toàn vận chuyển và chi phí kinh doanh [3]. Trong bối cảnh đó, các doanh nghiệp vừa và nhỏ, vốn chiếm tỷ lệ lớn trong ngành logistics tại Việt Nam, gặp khó khăn trong việc tiếp cận các hệ thống công nghệ đắt đỏ được sử dụng tại các cảng lớn [4]. Ngoài ra, việc phân tích nguyên nhân hư hỏng container và tổn thất hàng hóa trong quá trình vận chuyển đã được nghiên cứu, nhấn mạnh nhu cầu áp dụng công nghệ tiên tiến để giảm thiểu rủi ro [5]. Sự phát triển của học máy, đặc biệt là các mô hình như YOLOv8, cùng với các nghiên cứu về ứng dụng YOLO-NAS trong phát hiện hư hỏng [6], đã mở ra cơ hội xây dựng giải pháp chi phí thấp, hiệu quả cao. Do đó, việc chọn đề tài phát triển hệ thống nhận diện lỗi hư hỏng container sử dụng YOLOv8 và Raspberry Pi 4 là cần thiết để giải quyết các thách thức thực tiễn, hỗ trợ tối ưu hóa hoạt động kinh doanh cho các doanh nghiệp vừa và nhỏ.

1.1.2. Mục đích của đề tài

Mục đích của đề tài là phát triển một hệ thống nhận diện lỗi hư hỏng container chi phí thấp, dựa trên mô hình YOLOv8, tích hợp với Raspberry Pi 4 và nền tảng web cùng ứng dụng di động để tự động hóa quy trình kiểm tra, nâng cao độ chính xác trong việc phát hiện các lỗi phổ biến (thủng, rỉ sét, móp) theo tiêu chuẩn IICL, đồng thời hỗ trợ dự toán chi phí sửa chữa. Hệ thống nhằm thay thế dần phương pháp kiểm tra thủ công, giảm thiểu chi phí đầu tư và vận hành, từ đó nâng cao hiệu quả quản lý container và góp phần thúc đẩy sự bền vững trong ngành logistics.

1.1.3. Đối tượng nghiên cứu của đề tài

Đối tượng nghiên cứu tập trung vào các container thương mại thông thường tại các bãi container của doanh nghiệp vừa và nhỏ trong lĩnh vực logistics, đặc biệt là những container đã xuống cấp với các dạng hư hỏng phổ biến như thủng, rỉ sét, móp, và lõm, tuân thủ tiêu chuẩn IICL (Institute of International Container Lessors). Hệ thống hướng tới phục vụ các doanh nghiệp có quy mô vừa và nhỏ tại Việt Nam, nơi chi phí đầu tư vào công nghệ cao là một rào cản lớn.

1.1.4. Phạm vi nghiên cứu

Phạm vi nghiên cứu giới hạn trong việc phát triển và thử nghiệm hệ thống nhận diện lỗi hư hỏng container sử dụng mô hình YOLOv8, tích hợp với Raspberry Pi 4 để xử lý hình ảnh tại chỗ, truyền dữ liệu đến máy chủ để phân tích, và hiển thị kết quả qua nền tảng web và ứng dụng di động. Hệ thống tập trung vào việc phát hiện bốn loại lỗi chính (thủng, rỉ sét, móp, lõm) và nhận diện biển số container dựa trên tiêu chuẩn IICL, sử dụng dữ liệu hình ảnh do nhân viên kỹ thuật chụp tại các bãi container. Nghiên cứu được thực hiện trong điều kiện môi trường thực tế tại Việt Nam, với trọng tâm là tối ưu hóa chi phí và hiệu suất, và không bao gồm các loại container đặc thù như chở hàng nguy hiểm hoặc khí tự nhiên hóa lỏng.

1.2. TỔNG QUAN

1.2.1. Phân tích và đánh giá các hướng nghiên cứu đã có trong và ngoài nước

Trong lĩnh vực logistics hiện đại, việc phát hiện và quản lý lỗi hư hỏng trên bề mặt container đóng vai trò thiết yếu trong việc đảm bảo an toàn vận chuyển và tối ưu hóa chi phí vận hành [1]. Theo tiêu chuẩn IICL (Institute of International Container Lessors), các lỗi phổ biến như thủng, rỉ sét, móp, và lõm không chỉ làm suy giảm chất lượng container mà còn ảnh hưởng nghiêm trọng đến an toàn hàng hóa [2]. Các yếu tố môi trường như độ ẩm, nhiệt độ, và muối biển được xác định là nguyên nhân chính làm tăng tốc độ xuống cấp của container [3], đặt ra nhu cầu sử dụng các giải pháp tự động để nâng cao hiệu quả kiểm tra. Nhiều nghiên cứu trong và ngoài nước đã tập trung vào việc ứng dụng công nghệ tiên tiến để tự động hóa quy trình kiểm tra lỗi hư hỏng container, giảm sự phụ thuộc vào phương pháp thủ công vốn tốn thời gian, nhân lực, và dễ xảy ra sai sót [4].

1.2.2. Nghiên cứu trong nước

Tại Việt Nam, các nghiên cứu về quản lý container chủ yếu tập trung vào vận hành cảng và phân tích nguyên nhân hư hỏng trong quá trình vận chuyển. T. V. Anh đã phân tích các thách thức trong ngành logistics tại Việt Nam, nhấn mạnh rằng các doanh nghiệp vừa và nhỏ gặp khó khăn trong việc tiếp cận công nghệ hiện đại do hạn chế về ngân sách [5]. Do đó, các giải pháp tự động hóa phát hiện hư hỏng container sử dụng công nghệ học sâu tại các bãi container quy mô nhỏ, đặc biệt dành cho doanh nghiệp vừa và nhỏ, vẫn còn hạn chế. Các hệ thống hiện tại thường được triển khai tại các cảng lớn với chi phí đầu tư cao, không phù hợp với điều kiện thực tế của nhiều doanh nghiệp trong nước [5].

1.2.3. Nghiên cứu ngoài nước

Trên thế giới, nhiều nhà nghiên cứu đã áp dụng học máy và học sâu để giải quyết bài toán phát hiện hư hỏng container, đạt được những kết quả đáng chú ý. W. Zixin và các cộng sự phát triển phương pháp nhận diện đa loại hư hỏng container

bằng mạng CNN dựa trên học chuyển giao, đạt độ chính xác cao trong việc phát hiện các hư hỏng phức tạp [1]. Tuy nhiên, phương pháp này yêu cầu tài nguyên tính toán lớn, không phù hợp với các hệ thống chi phí thấp [1]. Tương tự, T. Nguyen Thi Phuong và G. S. Cho sử dụng mô hình YOLO-NAS để tự động phát hiện hư hỏng container, cho thấy hiệu quả cao trong môi trường cảng lớn, nhưng không đề cập đến tối ưu hóa chi phí triển khai, khiến giải pháp khó tiếp cận với các doanh nghiệp nhỏ [6].

Ngoài lĩnh vực container, các mô hình học sâu như YOLOv8 và YOLO-NAS cũng được ứng dụng rộng rãi trong các lĩnh vực khác, mang lại bài học kinh nghiệm cho đề tài. M. Deepak và các cộng sự so sánh YOLO-NAS và YOLOv8 trong nhận diện biển số xe, với YOLOv8 cho thấy tốc độ xử lý nhanh hơn và độ chính xác tương đương, đạt khoảng 95% trên tập kiểm tra [4]. K. Yogesh và K. Pankaj áp dụng YOLOv8 và YOLO-NAS trong nông nghiệp, khẳng định YOLOv8 có ưu thế về tốc độ và tính linh hoạt [7]. Những nghiên cứu này cung cấp nền tảng lý thuyết vững chắc để áp dụng YOLOv8 vào nhận diện hư hỏng container và biển số container.

Bên cạnh đó, các nghiên cứu về vận hành cảng và nguyên nhân hư hỏng cũng đóng vai trò quan trọng trong việc định hướng giải pháp. C. Indranath và C. Gyusung đề xuất phương pháp phân bổ cần cẩu tại cảng container bằng mô phỏng và học máy, cải thiện hiệu quả vận hành tại các cảng lớn [8]. Tuy nhiên, nghiên cứu này không tập trung vào phát hiện hư hỏng container mà hướng đến tối ưu hóa hạ tầng cảng, đòi hỏi chi phí đầu tư lớn, không phù hợp với bãi container nhỏ [8]. K. Magdalena và các cộng sự phân tích nguyên nhân gây hư hỏng hàng hóa trong quá trình vận chuyển container, chỉ ra rằng các lỗi như thùng, rỉ sét, móp, và lõm thường xuất phát từ quá trình xếp dỡ và môi trường khắc nghiệt [3]. Tuy nhiên, nghiên cứu này chỉ dừng ở phân tích lý thuyết, không đề xuất giải pháp công nghệ cụ thể để tự động hóa quy trình kiểm tra [3].

Bảng 1.1: Kết quả một số nghiên cứu về nhận diện lỗi hư hỏng của container.

Thuộc tính	Multitype damage detection of containers using CNN based on transfer learning [19]	Comparative study of YOLOv8 and YOLO-NAS for agriculture application [20]	A metaphor analysis on vehicle license plate detection using YOLO-NAS and YOLOv8 [8]	Port container terminal quay crane allocation based on simulation and machine learning method [1]	Automating container damage detection with the YOLO-NAS deep learning model [16]
Tác giả	W. Zixin, G. Jing, Z. Qingcheng et al. [19]	K. Yogesh and K. Pankaj [20]	M. Deepak, K. Prashant, S. Sunil et al. [8]	C. Indranath and C. Gyusung [1]	T. Nguyen Thi Phuong and G. S. Cho [16]
Năm	2021 [19]	2024 [20]	2024 [8]	2022 [1]	2025 [16]
Phương pháp phát hiện	CNN (học chuyển giao) [19]	YOLOv8 và YOLO-NAS (học sâu) [20]	YOLO-NAS và YOLOv8 (học sâu) [8]	Mô phỏng và học máy [1]	YOLO-NAS (học sâu) [16]
Kết quả đầu ra	Nhận diện đa loại hư hỏng container [19]	Ứng dụng trong nông nghiệp [20]	Nhận diện biển số xe [8]	Phân bổ cần cầu tại cảng [1]	Nhận diện hư hỏng container [16]

Giá thiết bị (ước lượng, USD)	>1000\$ [19]	Không nêu, giả định >1000\$ [20]	Không nêu, giả định >1000\$ [8]	>5000\$ [1]	>1000\$ [16]
Độ chính xác (hoặc hiệu suất, %)	Giả định ~90% [19]	Giả định ~90% [20]	~95% (validation) [8]	Không áp dụng, tập trung vào tối ưu hóa [1]	Giả định ~90% [16]
Phạm vi ứng dụng	Cảng container nói chung [19]	Nông nghiệp [20]	Ứng dụng giao thông [8]	Cảng container quy mô lớn [1]	Cảng container quy mô lớn [16]

1.2.4. Những vấn đề còn tồn tại

Mặc dù các nghiên cứu trên đã đạt được nhiều tiến bộ, vẫn tồn tại một số hạn chế cần được giải quyết:

- Chi phí triển khai cao: Các giải pháp của W. Zixin [1] và T. Nguyen Thi Phuong [6] yêu cầu phần cứng mạnh (GPU, PC cấu hình cao), với chi phí ước lượng trên 1000 USD [1], [6], không phù hợp với các doanh nghiệp vừa và nhỏ tại Việt Nam, nơi ngân sách hạn chế là rào cản lớn [5].
- Thiếu tính ứng dụng thực tiễn: Các nghiên cứu như của C. Indranath [8] và K. Magdalena [3] tập trung vào cảng lớn hoặc phân tích lý thuyết [3], [8], không giải quyết trực tiếp nhu cầu kiểm tra hư hỏng tại các bãi container quy mô nhỏ, nơi đòi hỏi giải pháp chi phí thấp và dễ triển khai [5].
- Hạn chế về tính năng mở rộng: Các hệ thống trong nghiên cứu của W. Zixin [1] và T. Nguyen Thi Phuong [6] chủ yếu tập trung vào nhận diện hư hỏng mà không tích hợp các tính năng hỗ trợ doanh nghiệp như dự toán chi phí sửa chữa hay lưu trữ dữ liệu qua nền tảng web/di động [1], [6].

- Khả năng hoạt động trong điều kiện thực tế: Hầu hết các nghiên cứu chưa đề cập đến hiệu suất trong các điều kiện môi trường đa dạng (ánh sáng yếu, thời tiết xấu) [1], [4], [6], một yếu tố quan trọng đối với các bãi container tại Việt Nam, nơi điều kiện vận hành thường không lý tưởng [3].

1.2.5. Những vấn đề đề tài cần tập trung và nghiên cứu giải quyết

Dựa trên các hạn chế trên, đề tài tập trung giải quyết các vấn đề sau:

- Tối ưu hóa chi phí triển khai: Phát triển hệ thống nhận diện lỗi hư hỏng container sử dụng YOLOv8 tích hợp với Raspberry Pi 4, một thiết bị giá rẻ (ước tính 50-100 USD) [4], giúp các doanh nghiệp vừa và nhỏ dễ dàng tiếp cận công nghệ tiên tiến mà không cần đầu tư lớn [5].
- Tăng tính ứng dụng thực tiễn: Tích hợp nhận diện lỗi hư hỏng và biển số container, đồng thời phát triển nền tảng web và ứng dụng di động để lưu trữ dữ liệu, dự toán chi phí sửa chữa [6], hỗ trợ doanh nghiệp tối ưu hóa chiến lược kinh doanh—những tính năng mà các nghiên cứu trước đây chưa giải quyết [1], [3].
- Đảm bảo hiệu suất trong điều kiện thực tế: Huấn luyện và thử nghiệm mô hình YOLOv8 trên dữ liệu thực tế tại các bãi container ở Việt Nam, đảm bảo hệ thống nhận diện chính xác các lỗi phổ biến (thùng, rỉ sét, móp, lõm) và biển số container theo tiêu chuẩn IIICL trong các điều kiện môi trường đa dạng, bao gồm ánh sáng yếu và thời tiết xấu [3].
- Tăng tính linh hoạt và khả năng mở rộng: Thiết kế hệ thống với khả năng nâng cấp trong tương lai, ví dụ: mở rộng nhận diện các loại hư hỏng khác, cải thiện độ chính xác OCR cho nhận diện biển số [4], hoặc tích hợp tính năng phân tích dữ liệu dài hạn, đáp ứng nhu cầu phát triển của doanh nghiệp [6].

Chương 2. NGHIÊN CỨU THỰC NGHIỆM HOẶC LÝ THUYẾT

2.1. Tổng quan về xử lý ảnh số (Image Processing)

- Các bước cơ bản: tiền xử lý (preprocessing), phân đoạn (segmentation), trích chọn đặc trưng (feature extraction), phân loại (classification).
- Kỹ thuật làm rõ ảnh, lọc nhiễu, cân bằng sáng/tương phản...

2.1.1. Tổng quan về Deep Learning và CNN:

Mạng nơ-ron tích chập (CNN) là một kiến trúc nền tảng trong lĩnh vực học sâu (deep learning), được thiết kế chuyên biệt để xử lý và phân tích hiệu quả các dạng dữ liệu có cấu trúc lưới, điển hình là hình ảnh và video. Khả năng tự động học và trích xuất các đặc trưng phức tạp từ dữ liệu hình ảnh đã giúp CNN trở thành một công nghệ mang tính cách mạng trong Thị giác máy tính (Computer Vision), dẫn đến những tiến bộ đáng kể trong các ứng dụng nhận diện đối tượng, phân loại hình ảnh, và nhiều tác vụ phức tạp khác. CNN đã được chứng minh là đặc biệt hiệu quả trong các bài toán nhận diện hư hỏng, bao gồm cả việc xác định các loại hư hỏng khác nhau trên container.

Một mạng CNN hoàn chỉnh có thể bao gồm nhiều loại lớp khác nhau, tuy nhiên, ba loại lớp sau đây được xem là cốt lõi và định hình nên hoạt động cơ bản của hầu hết các kiến trúc CNN: Lớp Tích chập (Convolutional Layer), Lớp Gộp (Pooling Layer), và Lớp Kết nối đầy đủ (Fully Connected Layer).

2.1.1.1. Lớp Tích chập (Convolutional Layer)

Lớp tích chập là khối xây dựng cơ bản và quan trọng nhất của CNN, đóng vai trò chính trong việc tự động trích xuất các đặc trưng có ý nghĩa từ dữ liệu đầu vào. Mục tiêu của lớp này là phát hiện các mẫu hình cục bộ trong hình ảnh, như các cạnh, đường nét, góc, hoặc các kết cấu cụ thể.

- Bộ lọc (Filters/Kernels): Mỗi bộ lọc là một ma trận trọng số có kích thước nhỏ (ví dụ: 3×3 hoặc 5×5 pixel), chứa các giá trị được học trong quá trình

huấn luyện. Mỗi bộ lọc được thiết kế để nhạy cảm với một loại đặc trưng thì giác cụ thể. Một lớp tích chập thường sử dụng nhiều bộ lọc khác nhau để học và trích xuất đa dạng các loại đặc trưng, từ đó tạo ra nhiều bản đồ đặc trưng.

- Phép tích chập (Convolution Operation): Phép toán này liên quan đến việc bộ lọc quét (scan) qua toàn bộ chiều rộng và chiều cao của dữ liệu đầu vào (hoặc bản đồ đặc trưng từ lớp trước đó) theo một bước nhảy (stride) nhất định. Tại mỗi vị trí bộ lọc phủ lên một vùng của ảnh, nó thực hiện phép nhân từng phần tử (element-wise multiplication) giữa các giá trị trong bộ lọc và các giá trị pixel tương ứng trong vùng đó, sau đó tổng hợp các kết quả này lại để tạo ra một giá trị duy nhất. Giá trị này tạo nên một phần tử trong ma trận đầu ra, được gọi là bản đồ đặc trưng (feature map).

- Bản đồ đặc trưng: Kết quả của quá trình tích chập là một ma trận mới, chính là bản đồ đặc trưng, làm nổi bật những vị trí trong hình ảnh gốc nơi bộ lọc đã phát hiện đặc trưng mà nó được thiết kế để tìm kiếm.

- Trường tiếp nhận cục bộ (Local Receptive Fields): Mỗi nơ-ron (hay mỗi phần tử) trong bản đồ đặc trưng đầu ra chỉ được tính toán dựa trên một vùng nhỏ và cục bộ của lớp đầu vào. Điều này giúp CNN tập trung vào các đặc trưng cục bộ và giảm đáng kể số lượng tham số cần học, tối ưu hóa hiệu suất khi xử lý dữ liệu hình ảnh lớn.

- Chia sẻ trọng số (Weight Sharing): Cùng một bộ lọc (tức là cùng một tập hợp các trọng số) được áp dụng trên toàn bộ dữ liệu đầu vào. Nguyên lý này cho phép một đặc trưng học được tại một vị trí có thể được phát hiện ở bất kỳ vị trí nào khác trong hình ảnh bằng chính bộ lọc đó. Việc chia sẻ trọng số không chỉ tối ưu hóa bộ nhớ mà còn làm cho CNN có khả năng nhận diện các đặc trưng độc lập với vị trí của chúng (translation invariance), một thuộc tính thiết yếu trong thị giác máy tính.

2.1.1.2. Lớp Gộp (Pooling Layer)

- Lớp gộp thường được chèn xen kẽ giữa các lớp tích chập liên tiếp trong kiến trúc CNN. Chức năng chính của nó là giảm kích thước không gian của các bản đồ đặc trưng, từ đó tối ưu hóa hiệu quả tính toán và tăng cường khả năng tổng quát hóa của mạng.
- Cách hoạt động: Lớp gộp chia mỗi bản đồ đặc trưng (đầu ra từ lớp tích chập trước đó) thành các vùng nhỏ không chồng lấn (ví dụ: các ô vuông 2×2 pixel). Sau đó, nó thực hiện một phép toán đơn giản trên mỗi vùng này để trích xuất một giá trị đại diện.
- Mục đích chính:
 - Giảm chiều dữ liệu (Down-sampling): Việc giảm kích thước của bản đồ đặc trưng đồng nghĩa với việc giảm khối lượng tính toán và số lượng tham số trong các lớp tiếp theo, từ đó tăng tốc độ huấn luyện và giảm yêu cầu về tài nguyên bộ nhớ.
 - Kiểm soát quá khớp (Overfitting): Bằng cách "nén" thông tin và loại bỏ một số chi tiết nhỏ có thể là nhiễu, lớp gộp giúp làm cho mô hình ít phức tạp hơn và giảm nguy cơ "học thuộc lòng" dữ liệu huấn luyện, từ đó cải thiện khả năng tổng quát hóa trên dữ liệu mới.
 - Tăng tính bất biến tịnh tiến (Translation Invariance): Lớp gộp làm cho các đặc trưng học được ít nhạy cảm hơn với sự dịch chuyển nhỏ của đặc trưng trong hình ảnh. Điều này đảm bảo rằng mạng vẫn có thể nhận diện một đặc trưng ngay cả khi nó xuất hiện ở vị trí hơi khác biệt trong ảnh, góp phần vào tính bền vững của mô hình.
- Loại Gộp phổ biến: Max Pooling (Gộp cực đại): Đây là phương pháp được sử dụng rộng rãi nhất. Trong mỗi vùng nhỏ, lớp Max Pooling đơn giản là chọn ra giá trị lớn nhất và loại bỏ các giá trị còn lại, dựa trên giả định rằng giá trị lớn nhất đại diện cho đặc trưng mạnh nhất và có ý nghĩa nhất trong vùng đó.

2.1.1.3. Lớp Kết nối đầy đủ (Fully Connected Layer)

Sau khi các lớp tích chập và gộp đã hoạt động để trích xuất và tinh chỉnh các đặc trưng cấp cao từ hình ảnh, thông tin này được chuyển đến lớp kết nối đầy đủ để tổng hợp và đưa ra quyết định cuối cùng.

- Chuyển đổi dữ liệu (Flattening): Trước khi được đưa vào lớp kết nối đầy đủ, các bản đồ đặc trưng đa chiều (đầu ra từ lớp gộp cuối cùng) được "làm phẳng" (flattened) thành một vector một chiều. Bước này biến đổi dữ liệu từ dạng ma trận không gian sang dạng phù hợp với các lớp nơ-ron truyền thống.
- Cấu trúc và Hoạt động: Lớp này hoạt động tương tự như các lớp trong mạng nơ-ron truyền thống (Multi-layer Perceptron). Mỗi nơ-ron từ lớp trước đó thiết lập kết nối với tất cả các nơ-ron của lớp tiếp theo. Các kết nối này mang trọng số được học trong quá trình huấn luyện. Mỗi nơ-ron trong lớp kết nối đầy đủ thực hiện một phép tính tổng có trọng số của tất cả các đầu vào từ lớp trước, sau đó áp dụng một hàm kích hoạt để tạo ra đầu ra.
- Mục đích chính:
 - Tổng hợp đặc trưng: Lớp kết nối đầy đủ chịu trách nhiệm tổng hợp các đặc trưng cấp cao và trừu tượng đã được học từ các lớp tích chập và gộp. Đây là giai đoạn mà các đặc trưng cục bộ được kết hợp để tạo ra một biểu diễn toàn diện của hình ảnh.
 - Ra quyết định cuối cùng: Dựa trên biểu diễn tổng hợp này, lớp kết nối đầy đủ cuối cùng (lớp đầu ra) sẽ đưa ra phán đoán của mô hình. Ví dụ, trong bài toán phân loại hình ảnh, lớp này sẽ xác định xác suất hình ảnh thuộc về từng loại đối tượng. Các mô hình CNN hiện đại như YOLO-NAS và YOLOv8, thường sử dụng các biến thể của lớp kết nối đầy đủ hoặc các cấu trúc tương tự ở phần đầu ra để thực hiện việc phát hiện đối tượng và phân loại. Chúng đã được chứng minh hiệu quả trong các ứng dụng nhận diện biển số xe và nông nghiệp, cho thấy tiềm năng trong việc tự động hóa việc phát hiện hư hỏng container.

- Lớp đầu ra: Lớp kết nối đầy đủ cuối cùng thường sử dụng các hàm kích hoạt như Softmax cho các bài toán phân loại đa lớp (để đưa ra phân phối xác suất cho từng lớp), hoặc Sigmoid cho các bài toán phân loại nhị phân (để đưa ra xác suất cho một trong hai lớp).

Bằng cách xếp chồng và kết nối ba loại lớp cốt lõi này một cách thông minh, CNN có khả năng xây dựng một hệ thống phân cấp các đặc trưng, từ các đặc trưng cơ bản ở các lớp đầu tiên đến các biểu diễn trừu tượng và có ý nghĩa hơn ở các lớp sâu hơn. Điều này cho phép CNN tự động học cách "hiểu" nội dung hình ảnh một cách hiệu quả và đưa ra các dự đoán chính xác cho nhiều ứng dụng phức tạp, đặc biệt là trong các lĩnh vực như nhận diện và phân loại hình ảnh tự động.

2.2. YOLOv8

2.2.1. YOLO là gì ?

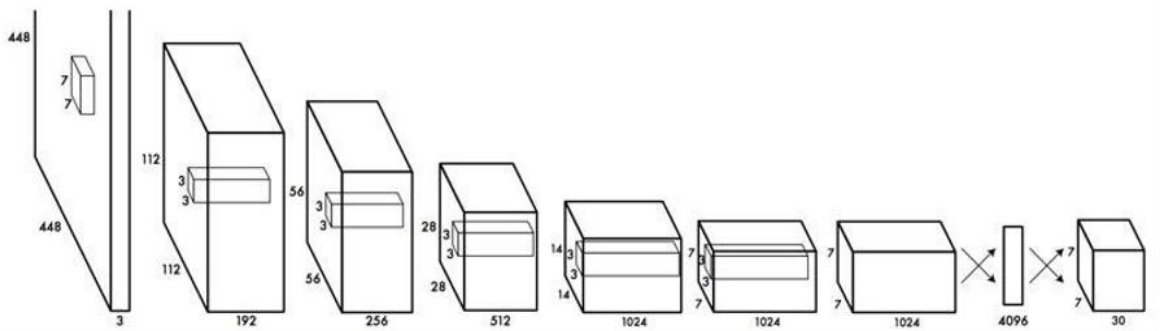
YOLO, viết tắt của "You Only Look Once," là một thuật toán phát hiện đối tượng trong lĩnh vực thị giác máy tính. Điểm đặc biệt của YOLO là khả năng dự đoán các hộp giới hạn (bounding box) và xác suất của đối tượng chỉ trong một lần xử lý hình ảnh duy nhất. Trước khi YOLO xuất hiện, các thuật toán phát hiện đối tượng thường sử dụng các trình phân loại đã đào tạo sẵn để xác định đối tượng sau khi tạo ra các khu vực quan tâm. Tuy nhiên, YOLO thay đổi điều này bằng cách thực hiện tất cả các dự đoán, từ việc xác định vị trí đến phân loại đối tượng, chỉ trong một lần xử lý duy nhất.

Với phương pháp đột phá này, YOLO đã mang lại sự cải tiến đáng kể so với các thuật toán phát hiện đối tượng trước đó, đặc biệt trong việc đảm bảo tính thời gian thực. Các thuật toán như Faster RCNN thường làm việc theo các bước gồm xác định khu vực quan tâm và sau đó phân loại trên từng khu vực riêng lẻ. Ngược lại, YOLO hoàn thành tất cả các dự đoán chỉ trong một lần chạy, giúp nó đạt được hiệu suất tính toán cao hơn.

Kể từ khi ra mắt lần đầu tiên vào năm 2015, YOLO đã liên tục được cải tiến và phát triển qua nhiều phiên bản mới, mang lại những tiến bộ đáng kể cho lĩnh vực phát hiện đối tượng.

2.2.2. Cơ chế hoạt động của YOLO

Mô hình YOLO được huấn luyện trước bằng ImageNet, một bộ dữ liệu lớn chứa nhiều hình ảnh đa dạng. Sau đó, mô hình được điều chỉnh để thực hiện nhiệm vụ phát hiện đối tượng. Lớp kết nối đầy đủ cuối cùng của YOLO được sử dụng để dự đoán xác suất của các lớp đối tượng và tọa độ của các hộp giới hạn.



Hình 2.1: Kiến trúc mạng YOLO và cơ chế hoạt động

Trong YOLO, hình ảnh đầu vào được chia thành một lưới ô lưới có kích thước $S \times S$. Mỗi ô lưới được gán nhiệm vụ phát hiện đối tượng nếu tâm của đối tượng nằm trong ô đó. Mỗi ô lưới dự đoán B hộp giới hạn và các điểm tin cậy cho mỗi hộp. Các điểm tin cậy này biểu thị độ tin cậy của mô hình về việc một hộp chứa một đối tượng và mức độ chính xác của dự đoán đó.

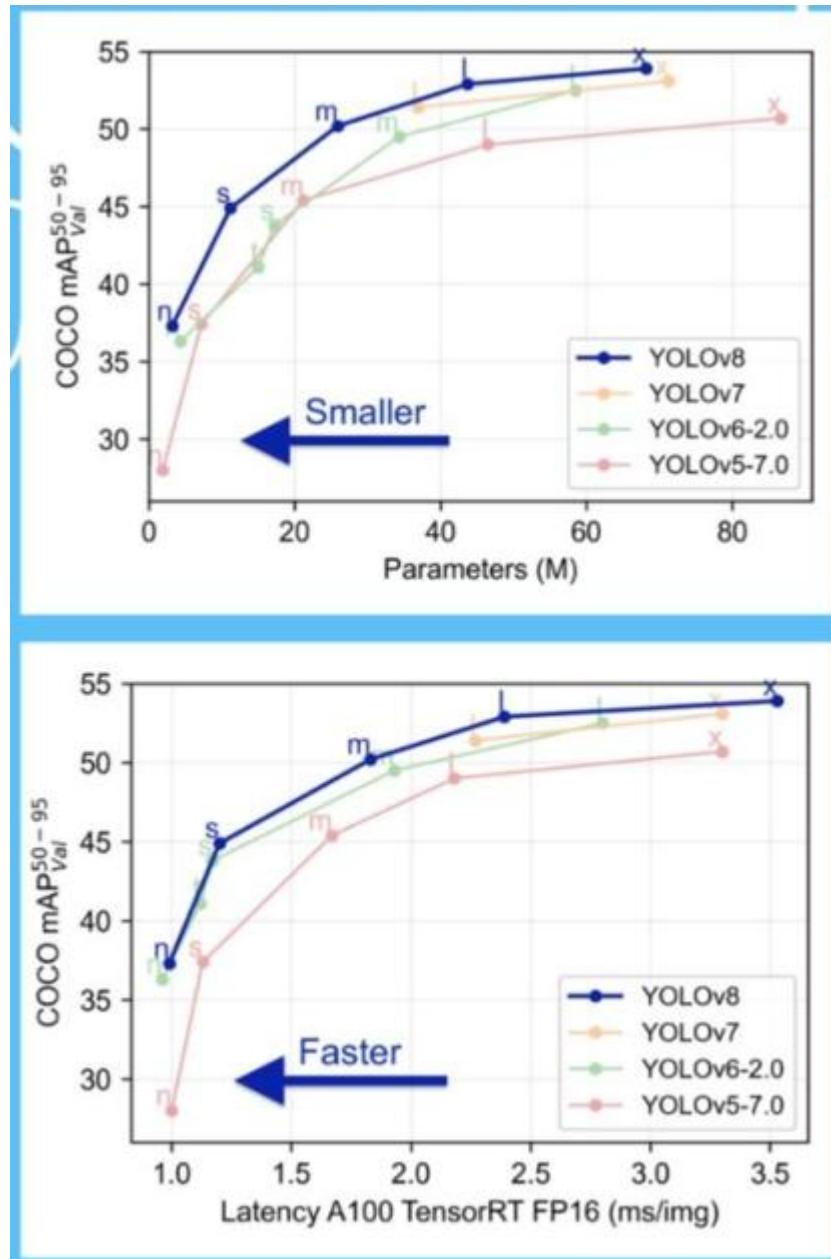
Trong quá trình dự đoán, YOLO dự đoán nhiều hộp giới hạn trên mỗi ô lưới và sau đó chỉ chọn hộp giới hạn dựa trên chỉ số IOU (Intersection over Union) cao nhất với thực tế. Quá trình này giúp loại bỏ sự trùng lặp giữa các dự đoán hộp giới hạn, từ đó cải thiện độ chính xác tổng thể.

NMS (non-maximum suppression) là một kỹ thuật quan trọng được dùng trong YOLO, sử dụng để loại bỏ các hộp giới hạn dư thừa hoặc không chính xác sau khi thực hiện các dự đoán. NMS giúp xác định hộp giới hạn duy nhất cho mỗi đối tượng trong hình ảnh, từ đó cải thiện độ chính xác và hiệu quả của việc phát hiện đối tượng.

Tóm lại, đầu vào của mô hình YOLO là một bức ảnh, và mô hình sẽ xác định xem có đối tượng nào trong ảnh hay không, sau đó xác định vị trí của đối tượng đó trong bức ảnh.

Từ hình 2.2 ta có thể thấy, YOLOv8 có lượng tham số lớn hơn so với các phiên bản trước như YOLOv5, nhưng lại ít tham số hơn so với phiên bản YOLOv6. Nó cung cấp nhiều mAP hơn (khoảng 33%) cho các mô hình kích thước n và mAP lớn hơn nói chung.

Đồng thời, ta có thể thấy YOLOv8 có thời gian suy luận ngắn hơn so với các phiên bản YOLO tiền nhiệm.



Hình 2.2: So sánh YOLOv8 với những kiến trúc mạng khác

2.2.3. Tổng quan YOLOv8

YOLOv8 là một mô hình nhận dạng đối tượng dựa trên mạng neural tích chập (CNN) được phát triển bởi Joseph Redmon và nhóm nghiên cứu của ông tại Đại học Washington. Đây là phiên bản cải tiến của YOLOv7, với khả năng nhận diện đối tượng nhanh hơn và chính xác hơn nhờ vào các cải tiến

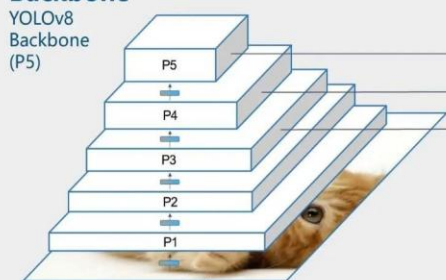
như mạng kim tự tháp đặc trưng, mô-đun chú ý không gian, và các kỹ thuật tăng cường dữ liệu tiên tiến.

YOLOv8 sử dụng mạng neural kiến trúc Darknet-53 để trích xuất đặc trưng từ hình ảnh và áp dụng thuật toán nhận dạng đối tượng YOLOv8 trên các đặc trưng này.

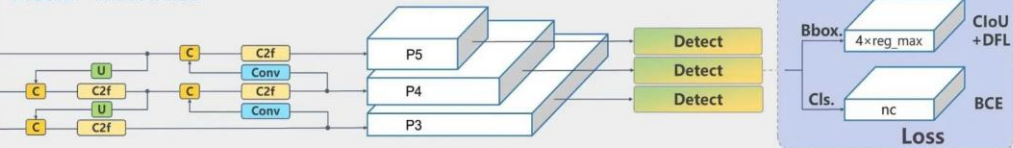
Đặc điểm nổi bật:

- Tăng tốc độ tính toán: Mô hình được tăng cường bằng cách thêm các kênh phân tán.
- Cải thiện nhận dạng: Sử dụng kỹ thuật Attention để nâng cao khả năng nhận diện đối tượng.
- Tăng tốc độ hội tụ: Áp dụng phương pháp đào tạo mới.
- Kiến trúc mạng nơ-ron mới: Sử dụng kiến trúc YOLOv4 làm cơ sở để nâng cao hiệu suất và độ chính xác.
- AutoScale: Tích hợp cơ chế tự động điều chỉnh tỷ lệ tăng kích thước của hình ảnh đầu vào.
- Video Supervision: Mô hình có khả năng phát hiện và giám sát đối tượng trong video, cung cấp dự đoán liên tục trên toàn bộ video.
- Ensemble: Tích hợp công nghệ Ensemble để cải thiện hiệu suất.
- AutoAnchor: Tính năng điều chỉnh tỷ lệ tự động giúp cải thiện khả năng phát hiện đối tượng với nhiều kích thước khác nhau.

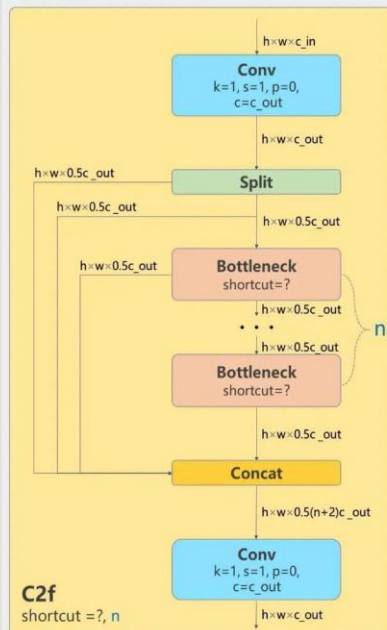
Backbone

YOLOv8
Backbone
(P5)

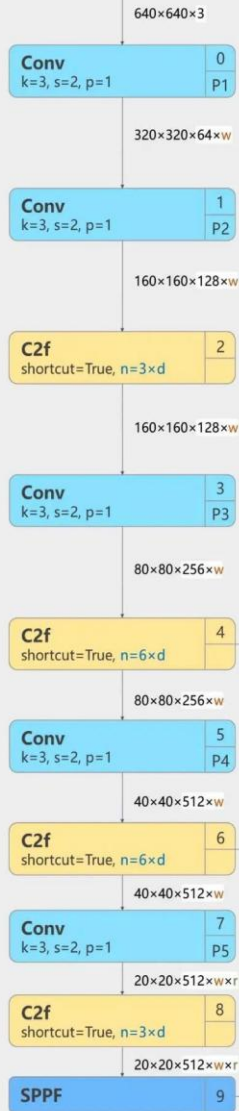
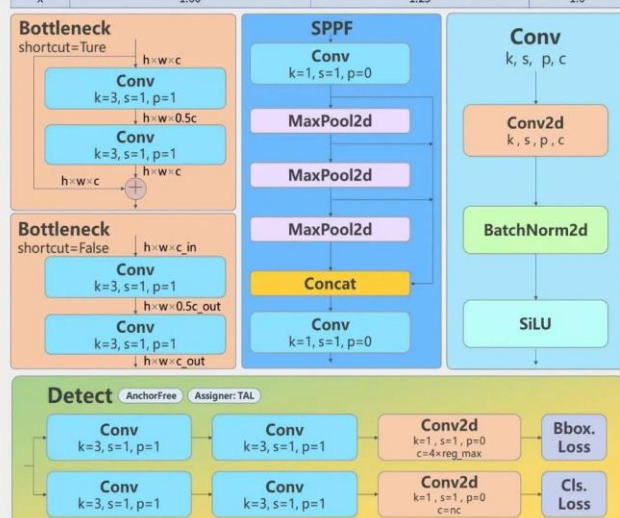
Head YOLOv8Head



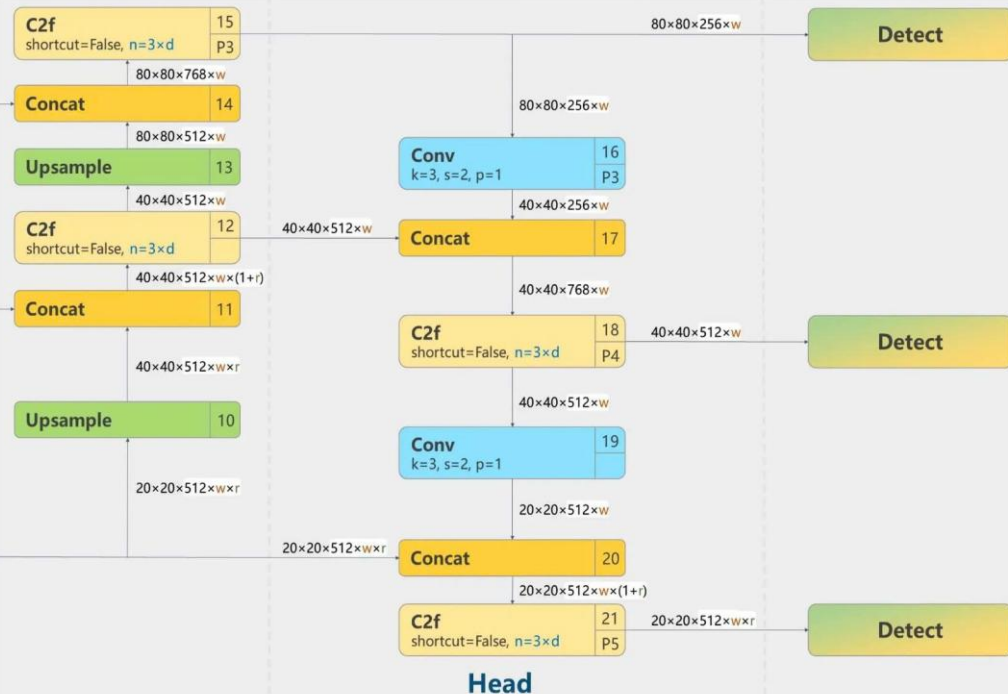
Details



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Note:
height*width*channel

Backbone



Hình 2.3: Kiến trúc mô hình YOLOv8.

Hình 2.3 miêu tả chi tiết kiến trúc mô hình của YOLOv8. Cụ thể được đề cập trong các mục dưới đây:

1. Backbone: Chịu trách nhiệm trích xuất các đặc trưng từ hình ảnh đầu vào.

Nó bao gồm nhiều lớp convolution và các khối khác nhau:

P1, P2, P3, P4, P5: Các lớp đặc trưng tương ứng với các cấp độ khác nhau của hình ảnh đầu vào.

Conv: Các lớp convolution cơ bản với kích thước kernel và stride được ghi chú (ví dụ: $k=3$, $s=2$).

C2f: Lớp convolution nối với nhau (concatenation) với số lượng lớp và số chiều đầu ra.

SPPF: Spatial Pyramid Pooling Fast, một kỹ thuật pooling giúp mô hình xử lý nhiều kích thước đầu vào khác nhau.

2. Head: Xử lý các đặc trưng được trích xuất từ "Backbone" để dự đoán các bounding boxes và lớp (classes) của đối tượng:

C2f: Các lớp convolution nối với nhau.

Upsample: Lớp tăng kích thước không gian của đặc trưng (upsampling).

Concat: Kết hợp các đặc trưng từ nhiều nguồn khác nhau.

Detect: Các lớp cuối chịu trách nhiệm dự đoán bounding boxes và lớp đối tượng.

3. Details: Cung cấp thông tin chi tiết về các khối cấu trúc trong YOLOv8:

3.1. Conv Layer: Lớp convolution cơ bản với các tham số như kích thước kernel (k), stride (s), và padding (p).

3.2. Bottleneck: Giảm thiểu số lượng tham số và tăng hiệu quả tính toán bằng cách sử dụng các lớp convolution nối tiếp nhau với shortcut connections.

3.3. SPPF: Pooling các đặc trưng ở các tỷ lệ khác nhau và kết hợp chúng lại để duy trì các phân cấp không gian.

3.4. C2f (Cross Stage Partial Connections): Kết hợp các đặc trưng từ nhiều giai đoạn khác nhau để tăng cường luồng thông tin và cải thiện hiệu suất của mô hình.

Detect: Lớp cuối cùng chịu trách nhiệm dự đoán bounding boxes và lớp đối tượng.

BatchNorm2d: Lớp chuẩn hóa đầu vào, giúp ổn định và tăng tốc quá trình huấn luyện.

SiLU: Hàm kích hoạt Sigmoid Linear Unit, giúp mô hình học các đặc trưng phi tuyến hiệu quả hơn.

4. Loss: Mô tả các hàm mất mát (loss functions) được sử dụng để huấn luyện mô hình YOLOv8:

Bbox: Hàm mất mát cho bounding boxes.

Cls: Hàm mất mát cho lớp đối tượng.

Dfl, BCE: Các hàm mất mát khác như Binary Cross-Entropy.

5. Model Scaling: Cung cấp các hệ số để thay đổi kích thước mô hình (scaling):

Model: Các loại mô hình khác nhau (n, s, m, l, x).

Depth, Width: Hệ số cho độ sâu và độ rộng của mô hình.

Stride: Các giá trị stride cho từng lớp convolution.

Concat: Kết hợp các đặc trưng từ nhiều lớp khác nhau.

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

Hình 2.4: Bảng thông số 5 model của YOLOv8

Yolov8 hiện có 5 model như hình 2.4, mỗi model sẽ có các thông số như trên và trong khuôn khổ đề tài nhận diện lỗi hư hỏng của container lần này thì chúng em sẽ xài YOLOv8m

2.3. Nhận diện hư hỏng container:

Container vận chuyển đóng vai trò then chốt trong thương mại toàn cầu, đảm bảo hàng hóa được vận chuyển hiệu quả và an toàn. Tuy nhiên, trong quá trình hoạt động, container thường xuyên chịu tác động từ môi trường và lực cơ học, dẫn đến nhiều dạng hư hỏng. Việc nhận diện và phân loại chính xác các hư hỏng này là rất quan trọng để duy trì tính toàn vẹn của container và bảo vệ hàng hóa. Ba dạng hư hỏng phổ biến nhất là rỉ sét (rust), vết lõm (dent), và lỗ thủng (hole).

2.3.1. Rỉ Sét (Rust)

Rỉ sét là một dạng hư hại vật liệu phổ biến nhưng nghiêm trọng, hình thành do quá trình oxy hóa kim loại thép – vật liệu chính cấu tạo nên container. Hiện tượng này xảy ra khi container tiếp xúc với các yếu tố môi trường khắc nghiệt như muối biển, nắng, gió và độ ẩm.

Ban đầu, rỉ sét thường biểu hiện dưới dạng những đốm màu nâu đỏ hoặc cam, với bề mặt thô ráp, xốp. Chúng tập trung ở các vùng ẩm ướt hoặc nơi lớp sơn bảo vệ đã bị bong tróc, như các mối hàn, góc cạnh, phần dưới của cửa và trên mái container – những điểm dễ bị đọng nước. Nguyên nhân chính gây ra rỉ sét xuất phát từ việc bề mặt container bị tổn thương bởi các tác nhân môi trường trong điều kiện làm việc khắc nghiệt. Sự tiếp xúc trực tiếp giữa kim loại trần với không khí và hơi ẩm sẽ thúc đẩy quá trình oxy hóa, dẫn đến hình thành và phát triển rỉ sét.

Hư hại do rỉ sét gây ra những tác động đáng kể đến độ bền vững và khả năng chịu tải của container, khiến chúng có thể không đạt hiệu suất như thiết kế ban đầu của nhà sản xuất. Điều này còn tiềm ẩn nguy cơ gây thiệt hại cho hàng hóa bên trong, bởi rỉ sét có thể tạo ra những vết nứt nhỏ hoặc lỗ thủng, cho phép nước hoặc các yếu tố gây hại khác xâm nhập vào khoang chứa, dẫn đến hư hỏng nghiêm trọng. Nếu không được xử lý kịp thời, lớp rỉ sét sẽ lan rộng khắp bề mặt container, làm giảm đáng kể tuổi thọ sử dụng và giá trị kinh tế của thiết bị này, đồng thời tăng chi phí bảo trì và sửa chữa.

2.3.2. Vết Lõm (Dent)

Vết lõm là một dạng biến dạng vật lý cục bộ, thường xuất hiện trên bề mặt container do những tác động cơ học từ bên ngoài. Đây là một trong những hư hại vật lý phổ biến nhất, thường xuyên xảy ra trong quá trình xếp dỡ, vận chuyển và lưu trữ.

Những vết lõm có thể đa dạng về hình dạng và kích thước, từ những chỗ biến dạng nhỏ, nông và khó nhận thấy, đến những biến dạng lớn, sâu làm thay đổi rõ rệt hình dạng ban đầu của tấm thép. Chúng thường xuất hiện trên các tấm panel bên ngoài

như thành, mái, hoặc cửa, cũng như trên các thanh giằng cấu trúc – những vị trí vốn dễ bị va chạm nhất. Nguyên nhân chính gây ra vết lõm là những va đập vật lý, điển hình như va chạm với thiết bị xếp dỡ (xe nâng, cầu trục), hoặc với các container khác, hay vật thể cứng. Ngoài ra, việc chất hàng không đúng cách, tạo áp lực quá mức lên một điểm cụ thể trên thành container, cũng có thể là nguyên nhân gây ra biến dạng này.

Hậu quả của vết lõm không chỉ dừng lại ở việc hư hại bề mặt của container mà còn tiềm ẩn nhiều rủi ro nghiêm trọng. Những vết lõm lớn hoặc nằm ở các vị trí chịu lực quan trọng có thể **làm suy yếu đáng kể cấu trúc tổng thể**, ảnh hưởng đến khả năng xếp chồng an toàn hoặc khả năng chịu tải của container. Nếu vết lõm xuất hiện gần cửa, nó có thể làm **cong vênh khung cửa**, gây khó khăn trong việc đóng/mở hoặc thậm chí làm mất đi độ kín, khiến nước dễ dàng xâm nhập vào khoang chứa hàng hóa. Điều này không chỉ gây bất tiện trong vận hành mà còn làm **giảm giá trị kinh tế** của container. Hơn nữa, tại vị trí bị lõm, bề mặt sơn bảo vệ thường bị biến dạng và nứt vỡ, để lộ phần kim loại bên trong, tạo điều kiện thuận lợi cho **rỉ sét** phát triển về sau.

2.3.3. Lỗ Thủng (Hole)

Lỗ thủng là dạng hư hỏng nghiêm trọng nhất, đánh dấu sự mất toàn vẹn hoàn toàn của vỏ container. Nó tạo ra một khe hở xuyên qua các tấm panel, mái, sàn hoặc cửa, mở ra đường tiếp xúc trực tiếp giữa không gian bên trong và môi trường bên ngoài.

Những lỗ thủng này cực kỳ đa dạng về hình dạng và kích thước: có thể là những lỗ kim nhỏ li ti do rỉ sét ăn mòn kéo dài, những vết thủng lớn do va đập mạnh, hay vết rách dài do vật sắc nhọn gây ra. Mặc dù chúng có thể xuất hiện ở bất kỳ đâu trên vỏ container, nhưng thường gặp nhất ở những vị trí dễ bị va đập hoặc những khu vực đã bị rỉ sét nặng và không được xử lý kịp thời. Nguyên nhân chính thường bắt nguồn từ ăn mòn nghiêm trọng đến mức vật liệu bị phá hủy hoàn toàn, hoặc do những va chạm mạnh với vật thể sắc nhọn, thiết bị xếp dỡ, hay thậm chí là do hàng hóa bên

trong không được cố định chắc chắn mà gây ra. Ngoài ra, sự lão hóa của vật liệu và hiện tượng mỏi kim loại do chịu tải trọng lặp đi lặp lại trong thời gian dài cũng có thể dẫn đến các vết nứt nhỏ ban đầu và phát triển thành lỗ thủng.

Hậu quả của một lỗ thủng là vô cùng nghiêm trọng và tác động trực tiếp đến hàng hóa. Điều hiển nhiên nhất là việc rò rỉ: lỗ thủng phá hủy độ kín nước và kín khí của container, cho phép nước mưa, bụi bẩn, côn trùng và các yếu tố môi trường có hại khác dễ dàng xâm nhập. Điều này có thể gây hư hại nghiêm trọng, thậm chí hủy hoại hoàn toàn hàng hóa, đặc biệt là những loại nhạy cảm với độ ẩm hoặc yêu cầu môi trường được kiểm soát. Bên cạnh đó, một lỗ thủng còn làm giảm mức độ an ninh của container, tạo kẽ hở cho trộm cắp hoặc can thiệp trái phép vào hàng hóa. Theo hầu hết các tiêu chuẩn kiểm định container quốc tế, bất kỳ lỗ thủng nào cũng không được chấp nhận và phải được sửa chữa ngay lập tức trước khi container được đưa vào sử dụng. Cuối cùng, những lỗ thủng lớn hoặc nằm ở các vị trí trọng yếu có thể làm suy yếu đáng kể cấu trúc tổng thể của container, ảnh hưởng trực tiếp đến khả năng chịu tải và an toàn trong quá trình xếp dỡ, xếp chồng.

Chương 3. THIẾT KẾ HỆ THỐNG NHÚNG

3.1. Tổng quan hệ thống nhúng

Hệ thống nhúng được nhóm phát triển thông qua các giai đoạn chính: ban đầu, hệ thống được nghiên cứu và thiết kế để tích hợp dataset hình ảnh container có sẵn từ các nguồn trực tuyến; tiếp theo, nhóm tiến hành huấn luyện mô hình YOLOv8 và triển khai chính mô hình này làm server. Sau đó, mô hình được tích hợp vào Raspberry Pi 4 để thực hiện nhận diện các lỗi hư hỏng (thùng, rỉ sét, móp, lõm) và biến số container, với kết quả được hiển thị trực quan trên màn hình.

3.2. Thành phần hệ thống

3.2.1. Vi điều khiển Raspberry Pi 4B

Raspberry Pi là một máy tính nhúng nhỏ gọn, được phát triển bởi Raspberry Pi Foundation với mục tiêu ban đầu là hỗ trợ giáo dục lập trình và phát triển ứng dụng công nghệ. Với thiết kế tối ưu và hiệu năng mạnh mẽ, Raspberry Pi đã trở thành một trong những nền tảng phổ biến nhất cho các dự án hệ thống nhúng, từ giáo dục đến ứng dụng công nghiệp. Trong đề tài này, nhóm đã lựa chọn Raspberry Pi 4 Model B làm trung tâm xử lý của hệ thống nhờ khả năng xử lý vượt trội so với các phiên bản trước đó và tính linh hoạt trong việc tích hợp với các thiết bị ngoại vi. Raspberry Pi 4B được trang bị vi xử lý Quad-core 64-bit ARM Cortex-A72 với tốc độ 1.5GHz, bộ nhớ RAM 4GB loại LPDDR4, và các cổng giao tiếp đa dạng như GPIO, USB, HDMI, và cổng CSI dành riêng cho camera.

Vai trò của Raspberry Pi 4B trong hệ thống là thu nhận hình ảnh từ Camera Raspberry Pi Camera V2 thông qua cổng CSI, thực hiện các bước tiền xử lý hình ảnh như resize về kích thước 640x640 pixel và tăng cường độ tương phản bằng thư viện OpenCV, sau đó truyền dữ liệu qua mạng Tailscale đến server để xử lý với mô hình YOLOv8. Ngoài ra, Raspberry Pi 4B còn điều khiển hiển thị kết quả nhận diện (ID container, loại lỗi, chỉ số hư hỏng) lên màn hình máy tính thông qua cổng HDMI, đồng thời hỗ trợ kết nối từ xa qua Tailscale và SSH Remote để quản lý và giám sát

hệ thống. Hệ điều hành Raspberry Pi OS (dựa trên Linux) được cài đặt trên thiết bị, cung cấp môi trường ổn định để chạy các phần mềm như OpenCV, Python, và các công cụ cần thiết cho việc triển khai hệ thống.



Hình 3.1: Raspberry Pi 4 Model B.

Một số thông số kỹ thuật chính của Raspberry Pi 4B được nêu dưới bảng 3.1 bên dưới.

Bảng 3.1: Thông số kỹ thuật của Raspberry Pi 4B.

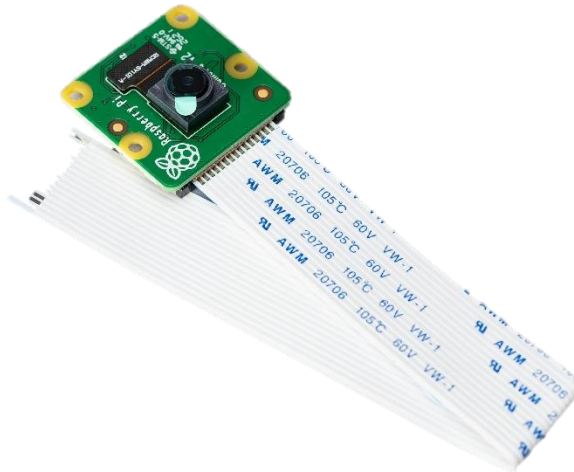
Thông số kỹ thuật
<ul style="list-style-type: none"> • Vi xử lý; Quad core 64-bit ARM-Cortex A72 (1.5GHz). • Ram: 1, 2 and 4 (GB), RAM LPDDR4. • Tích hợp 28 GPIO: UART, I2C, SPI, DPI, PCM, PWM, CPCLK. • Nguồn điện sử dụng: 5V/2.5A. • Màn hình HDMI lên đến 4Kp60.

3.2.2. Camera Raspberry Pi V2 IMX219

Camera Raspberry Pi V2 IMX219 8MP là phiên bản Camera Module dành cho Raspberry Pi mới nhất sử dụng cảm biến ảnh IMX219 8-megapixel từ Sony thay cho cảm biến cũ là OV5647. Với cảm biến IMX219 8-megapixel từ Sony, Camera Module cho Raspberry Pi đã có được sự nâng cấp vượt trội về cả chất lượng hình ảnh, video cũng như độ bền.

Camera Raspberry Pi V2 IMX219 8MP có thể sử dụng với Raspberry Pi để chụp hình, quay phim với chất lượng HD 1080p30, 720p60 hoặc VGA90, cách sử dụng cũng như lập trình với Camera Module trên Raspberry Pi cũng rất dễ dàng, chỉ cần cắm vào cổng Camera trên Raspberry Pi và qua 1 vài bước thiết lập là có thể dùng được.

Camera Raspberry Pi V2 IMX219 8MP có thể điều khiển thông qua MMAL và V4L APIs, có rất nhiều bộ thư viện được cộng đồng Raspberry Pi phát triển trên Python giúp cho việc tìm hiểu và sử dụng trở nên dễ dàng hơn rất nhiều.



Hình 3.2: Camera Raspberry Pi V2 IMX219.

Thông số kỹ thuật chi tiết của Camera Raspberry Pi V2 được nêu dưới bảng 3.2 bên dưới.

Bảng 3.2: Thông số kỹ thuật của Camera Raspberry Pi V2

Thông số kỹ thuật
<ul style="list-style-type: none">• Cảm biến: IMX219• Độ phân giải ảnh tĩnh :3280 x 2464 pixel• Độ phân giải video: 1080p @ 30 fps, 720p @ 60 fps, 640x480p @ 90 fps• Góc nhìn: 62.2° (đường chéo)• Kết nối :Cáp ribbon 15-pin với cổng CSI• Kích thước : 25mm x 24mm x 9mm• Điện năng: Cấp nguồn từ Raspberry Pi (khoảng 200-250mA)• Nhiệt độ hoạt động: -20°C đến 60°C

3.2.3. Nguồn 5V/3A(bao gồm mạch sạc pin18650 có UPS 5V/3A và 2 pin 18650 Panasonic 3400mAh)

Để tối ưu cho tính linh hoạt của hệ thống, nhóm quyết định chọn mạch sạc pin 18650 có UPS 5V/3A. Với dòng điện có định mức 5V/3A hoàn toàn có thể cung cấp năng lượng đầy đủ cho Pi hoạt động ổn định, tránh bị sụt áp trong thời gian dài làm việc ở bãi container. Ngoài ra mạch còn tích hợp khả năng sạc pin 18650 thông qua cổng usb type-c tích hợp giúp thuận tiện trong quá trình sử dụng.

Hai pin 18650 có dung lượng 3400mAh mỗi viên đảm bảo cho thời gian làm việc lâu dài. Ngoài ra việc sử dụng pin từ Panasonic là một nhà sản xuất lớn còn đảm bảo chu kỳ sạc/xả lớn, giúp kéo dài tuổi thọ của bộ nguồn,qua đó giảm thiểu chi phí thay thế và bảo trì trong suốt quá trình vận hành tại bãi container.



Hình 3.3: Mạch sạc pin 18650 với UPS 5V/3A.

Thông số kỹ thuật chi tiết của mạch sạc pin 18650 với UPS 5V/3A. được nêu dưới bảng 3.3 bên dưới.

Bảng 3.3: Thông số kỹ thuật của mạch sạc pin 18650 với UPS 5V/3A.

Thông số kỹ thuật
<ul style="list-style-type: none"> • Dung lượng pin: 13.000 mAh • Loại pin: Li-ion • Công suất tối đa: 15W • Đầu ra: 2 cổng USB-A (5V / 3A tổng cộng, tối đa 2.4A mỗi cổng) • Đầu vào: 1 cổng Micro USB (5V / 2A) • Công nghệ sạc: PowerIQ, VoltageBoost • Bảo vệ: Hệ thống an toàn MultiProtect (quá dòng, quá áp, ngắn mạch, kiểm soát nhiệt độ). • Kích thước: Khoảng 9.6 cm x 8.1 cm x 2.3 cm. • Trọng lượng: Khoảng 241g.

Bảng 3.4: Bảng thông số kỹ thuật của Pin 18650.

Thông số kỹ thuật
<ul style="list-style-type: none">• Model: NCR18650GA chính hãng Panasonic.• Kiểu pin: 18650 Li-ion rechargeable battery.• Điện áp trung bình 3.7VDC, sạc đầy 4.2VDC.• Dung lượng: 3400~3500mAh• Dòng xả tối đa liên tục: Max 10A (3C x 3500mAh)• Nội trở trung bình: khoảng 21~25mΩ• Số lần sạc xả: 1000 lần.• Kích thước: 18x65mm• Trọng lượng trung bình: 46g.

3.3. Thiết bị và thành phần hỗ trợ

3.3.1. Các ứng dụng hỗ trợ

- **Tailscale**

Tailscale là một ứng dụng mạng VPN mã nguồn mở, được phát triển dựa trên giao thức WireGuard, cung cấp một giải pháp kết nối an toàn và hiệu quả giữa các thiết bị trong cùng một mạng riêng ảo (VPN) qua Internet. Ứng dụng này nổi bật với khả năng thiết lập kết nối nhanh chóng mà không đòi hỏi cấu hình phức tạp, đồng thời đảm bảo bảo mật cao thông qua mã hóa đầu cuối (end-to-end encryption). Tailscale hỗ trợ truy cập từ xa tới Raspberry Pi 4 từ bất kỳ địa điểm nào có kết nối Internet, giúp loại bỏ các rào cản về địa lý trong quá trình triển khai hệ thống thực tế tại bãi container.

Trong đề tài, Tailscale được triển khai để thiết lập một kênh truyền dữ liệu an toàn giữa Raspberry Pi 4 và máy tính cá nhân (server), cho phép truyền hình ảnh container theo thời gian thực với tốc độ cao và độ tin cậy lớn. Địa chỉ IP động của

Raspberry Pi 4 được Tailscale quản lý tự động, loại bỏ nhu cầu sử dụng các công cụ quét mạng thủ công như Advanced IP Scanner, đồng thời hỗ trợ việc định tuyến dữ liệu một cách linh hoạt ngay cả khi mạng cục bộ thay đổi. Ngoài ra, Tailscale cung cấp giao diện quản lý đơn giản, cho phép người dùng theo dõi trạng thái kết nối, quản lý thiết bị, và phân quyền truy cập, giúp tối ưu hóa quy trình vận hành hệ thống.

Việc ứng dụng Tailscale trong hệ thống thực tế không chỉ đảm bảo truyền tải dữ liệu hình ảnh từ camera tới server một cách liên tục và bảo mật, mà còn hỗ trợ tích hợp với các nền tảng web và ứng dụng di động để hiển thị kết quả nhận diện lỗi hư hỏng container. Ngoài ra, Tailscale còn cho phép đồng bộ hóa file dữ liệu giữa Raspberry Pi 4 và server, tạo điều kiện thuận lợi cho việc lưu trữ và xử lý dữ liệu thô trên Google Colab, đặc biệt trong các giai đoạn huấn luyện mô hình YOLOv8.

- SSH Remote

SSH Remote là một công cụ hỗ trợ kết nối từ xa dựa trên giao thức SSH (Secure Shell), được thiết kế để cung cấp khả năng truy cập và quản lý các thiết bị như Raspberry Pi 4 từ xa một cách an toàn và hiệu quả. Giao thức SSH sử dụng mã hóa mạnh mẽ để bảo vệ dữ liệu truyền tải, bao gồm cả lệnh điều khiển và thông tin hệ thống, giúp ngăn chặn các cuộc tấn công nghe lén hoặc can thiệp trái phép. Công cụ này cho phép người dùng thực thi các lệnh dòng lệnh, giám sát trạng thái phần cứng, và điều chỉnh cấu hình hệ thống từ xa mà không cần giao diện đồ họa, phù hợp với các hệ thống nhúng như Raspberry Pi 4 trong môi trường thực tế.

Trong đề tài, SSH Remote được tích hợp để truy cập Raspberry Pi 4 thông qua địa chỉ IP được cung cấp bởi Tailscale, hỗ trợ thực hiện các tác vụ quản trị quan trọng như kiểm tra log hệ thống, cập nhật firmware, hoặc điều chỉnh thông số camera trong quá trình vận hành hệ thống nhận diện lỗi hư hỏng container theo thời gian thực. Ví dụ, người dùng có thể sử dụng SSH Remote để khởi động lại camera khi xảy ra lỗi kết nối, hoặc kiểm tra dung lượng bộ nhớ để đảm bảo dữ liệu hình ảnh không bị tràn bộ đệm. Sự kết hợp giữa SSH Remote và Tailscale tạo ra một giải pháp quản lý từ xa

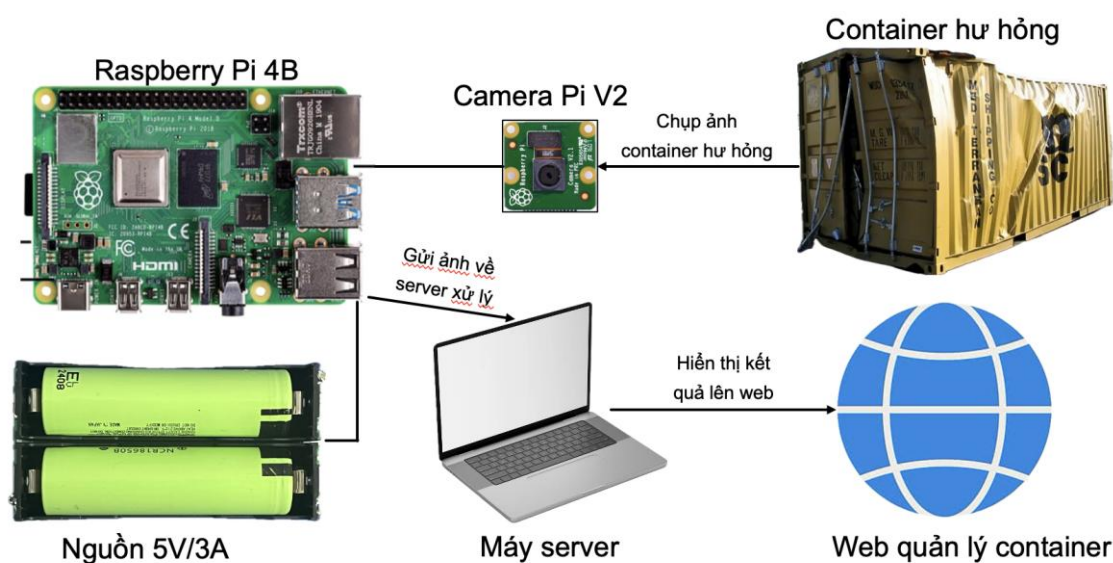
toàn diện, cho phép đội ngũ kỹ thuật can thiệp kịp thời ngay cả khi không có mặt trực tiếp tại bãi container.

Ngoài ra, SSH Remote còn hỗ trợ chuyển tiếp cổng (port forwarding), cho phép kết nối tới các dịch vụ khác trên Raspberry Pi 4, chẳng hạn như giao diện quản lý camera hoặc server cục bộ, từ bất kỳ máy tính nào trong mạng Tailscale. Điều này đặc biệt hữu ích trong việc debug hệ thống hoặc thử nghiệm các cập nhật mô hình YOLOv8 trước khi triển khai chính thức. Với khả năng này, nhóm nghiên cứu có thể tối ưu hóa hiệu suất hệ thống và đảm bảo tính ổn định trong quá trình thu thập dữ liệu liên tục, đồng thời chuẩn bị dữ liệu đầu vào cho các giai đoạn xử lý nâng cao trên server hoặc Google Colab.

3.4. Mô hình hệ thống

3.4.1. Mô tả kết nối hệ thống

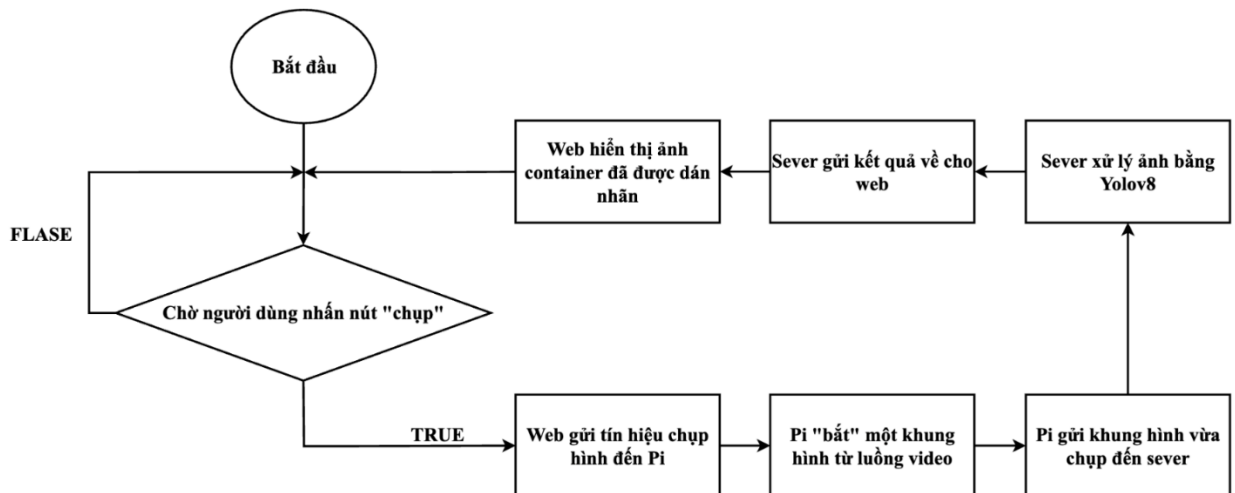
Sau khi tìm hiểu lý thuyết, các thiết bị chính và ứng dụng hỗ trợ, nhóm thực hiện xây dựng mô hình hệ thống nhằm để thu thập hình ảnh container và nhận diện lỗi hư hỏng theo thời gian thực. Hình 3.4 bên dưới cho thấy cách kết nối và các giao thức giữa các thành phần hệ thống.



Hình 3.4: Sơ đồ kết nối các thành phần của hệ thống.

Hệ thống được thiết kế với Raspberry Pi 4 đóng vai trò trung tâm, kết nối với Camera Raspberry Pi Camera V2 qua cổng CSI để thu nhận hình ảnh container liên tục theo thời gian thực. Raspberry Pi 4 kết nối với máy tính cá nhân (server) thông qua Tailscale, sử dụng VPN an toàn để truyền dữ liệu hình ảnh và nhận kết quả xử lý từ mô hình YOLOv8. Ngoài ra, SSH Remote được sử dụng để truy cập và quản lý Raspberry Pi 4 từ xa, hỗ trợ vận hành và giám sát hệ thống.

3.4.2. Lưu đồ giải thuật mô hình dự đoán lỗi hư hỏng container



Hình 3.5: Lưu đồ giải thuật mô hình dự đoán lỗi

Cách vận hành của hệ thống: hệ thống khởi động và liên tục thu nhận hình ảnh container từ Camera Raspberry Pi Camera V2 theo thời gian thực. Hình ảnh được tiền xử lý (resize và tăng cường độ tương phản) trên Raspberry Pi 4, sau đó gửi qua Tailscale đến server. Server sử dụng mô hình YOLOv8 đã huấn luyện để nhận diện lỗi hư hỏng (thùng, ri sét, lõm) và trích xuất biển số container bằng OCR (Tesseract), đồng thời tính toán chỉ số hư hỏng (độ sâu, diện tích). Kết quả, bao gồm ID container,

loại lỗi, biên số, và chỉ số hư hỏng, được trả về Raspberry Pi 4 và hiển thị trên màn hình LCD 16x2. Dữ liệu cũng được lưu trữ trên server và đóng gói để tích hợp với nền tảng web/app, hỗ trợ ước tính chi phí sửa chữa dựa trên tiêu chuẩn IICL. Mô hình YOLOv8 được huấn luyện từ bộ dữ liệu thô đã qua tiền xử lý, training và test trên Google Colab, với chi tiết trình bày ở Chương 4: Xây dựng mô hình máy học, và Chương 5: Kết quả thực nghiệm.

Chương 4. PHƯƠNG PHÁP NGHIÊN CỨU VÀ THIẾT KẾ HỆ THỐNG

Việc xây dựng mô hình phát hiện lỗi trên bề mặt container đòi hỏi một phương pháp luận nghiên cứu và thiết kế hệ thống tổng thể chặt chẽ, cùng với việc lựa chọn và áp dụng các thành phần kỹ thuật phù hợp. Quy trình triển khai giải pháp được mô tả chi tiết, bao gồm các giai đoạn then chốt như thu thập và xây dựng tập dữ liệu, tiền xử lý dữ liệu thô, lựa chọn và cấu hình mô hình học sâu, cũng như thiết kế kiến trúc hệ thống phục vụ triển khai thực tế. Tuân thủ phương pháp luận khoa học và thiết kế hệ thống chặt chẽ là yếu tố cốt lõi, đảm bảo tính hiệu quả, độ tin cậy và khả năng mở rộng của giải pháp trong bối cảnh ứng dụng công nghiệp.

Hệ thống sử dụng YOLOv8 để nhận diện lỗi hư hỏng của container gồm 4 bước chính:

- **Bước 1: Thu thập và xây dựng tập dữ liệu.** Bước này bao gồm việc xác định nguồn dữ liệu, phương pháp thu thập, quy trình gán nhãn, và tổ chức dữ liệu thô.
- **Bước 2: Tiền xử lý dữ liệu thô.** Giai đoạn này tập trung vào việc làm sạch, chuẩn hóa và biến đổi dữ liệu để phù hợp với yêu cầu của mô hình học máy.
- **Bước 3: Huấn luyện mô hình.** Đây là quá trình lựa chọn kiến trúc mô hình, cấu hình siêu tham số và đào tạo mô hình trên tập dữ liệu đã chuẩn bị.
- **Bước 4: Chạy thử nghiệm và đánh giá.** Bước cuối cùng nhằm kiểm tra hiệu suất của mô hình trong các điều kiện khác nhau và đánh giá khả năng ứng dụng thực tế.

4.1. Kiến trúc tổng thể hệ thống

Kiến trúc hệ thống đóng vai trò định hình cách các thành phần khác nhau của dự án tương tác và hoạt động cùng nhau để đạt được mục tiêu cuối cùng. Hệ thống phát hiện lỗi trên bề mặt container được thiết kế theo kiến trúc phân tán, tích hợp các công nghệ xử lý hình ảnh, kết nối mạng và giao diện người dùng, nhằm đảm bảo khả

năng hoạt động hiệu quả trong môi trường thực tế, đặc biệt là với mục tiêu triển khai trên các thiết bị nhúng.

4.1.1. Tổng quan về Kiến trúc Hệ thống

Hệ thống phát hiện lỗi được thiết kế theo kiến trúc phân tán, bao gồm các thành phần chính tương tác với nhau để tạo thành một giải pháp hoàn chỉnh và mạnh mẽ, đáp ứng các yêu cầu về tự động hóa và quản lý trong ngành logistics:

- **Thiết bị nhúng(Raspberry Pi 4):**Raspberry Pi đóng vai trò là điểm thu thập dữ liệu chính của hệ thống. Nhiệm vụ chính của nó là nhận hình ảnh trực tiếp từ camera CSI gắn trên Pi, sau đó truyền tải hình ảnh thô này về máy chủ trung tâm (server). Việc sử dụng Raspberry Pi 4 ở đây giúp hệ thống duy trì tính di động và khả năng thu thập dữ liệu linh hoạt tại hiện trường, đồng thời tối ưu hóa chi phí cho giai đoạn thu thập ban đầu, giảm thiểu băng thông mạng cần thiết cho việc truyền tải liên tục dữ liệu.
- **VPN:** Để đảm bảo kết nối ổn định và bảo mật giữa Raspberry Pi và các máy chủ (Server) hoặc thiết bị khác trong hệ thống, nhóm đã lựa chọn sử dụng Tailscale [20].
- **Server:** Nhiệm vụ chính của Server là nhận dữ liệu lỗi đã được phát hiện từ Module Xử lý Hình ảnh trên Raspberry Pi, lưu trữ dữ liệu này một cách có hệ thống, và xử lý các logic nghiệp vụ phức tạp liên quan đến quản lý dữ liệu lỗi.
- **Module Nhận dạng Ký tự Quang học (OCR):** Đây là một module hỗ trợ quan trọng và tích hợp, được thiết kế để tự động trích xuất thông tin số hiệu từ hình ảnh container.. Việc tích hợp OCR cho phép liên kết thông tin văn bản với dữ liệu lỗi được phát hiện, tăng cường khả năng theo dõi và quản lý container một cách tự động và hiệu quả hơn, đồng thời giảm thiểu đáng kể việc nhập liệu thủ công.
- **Giao diện Người dùng:** Để cung cấp khả năng tương tác và giám sát trực quan cho người dùng cuối (ví dụ: nhân viên kiểm tra, quản lý kho bãi), nhóm đã

thiết kế ra một giao diện web cho phép người dùng theo dõi trạng thái hoạt động của hệ thống theo thời gian thực, xem kết quả phát hiện lỗi một cách trực quan (bao gồm hình ảnh gốc với các hộp giới hạn lỗi được vẽ và thông tin chi tiết về từng lỗi), và quản lý các thông tin liên quan đến quy trình kiểm tra chất lượng.

4.1.2. Các Công nghệ và Công cụ được sử dụng

Để hiện thực hóa kiến trúc hệ thống đã đề xuất, một số công nghệ và công cụ chuyên biệt đã được lựa chọn và tích hợp một cách chiến lược, đảm bảo tính hiệu quả, độ tin cậy và khả năng mở rộng của từng module, đồng thời đáp ứng các yêu cầu về hiệu suất và khả năng triển khai trong môi trường công nghiệp:

- **OpenCV cho Xử lý Hình ảnh: OpenCV (Open Source Computer Vision Library)** [19] là một thư viện mã nguồn mở hàng đầu và mạnh mẽ trong lĩnh vực thị giác máy tính, được sử dụng rộng rãi cho các tác vụ xử lý ảnh và video. Trong hệ thống này, OpenCV đóng vai trò trung tâm trong Module Xử lý Hình ảnh trên Raspberry Pi, đảm nhiệm các chức năng quan trọng và tối ưu hóa hiệu suất:
 - **Thu thập và điều khiển camera:** Cung cấp các API mạnh mẽ để truy cập và điều khiển các camera kết nối với Raspberry Pi, cho phép thu thập luồng video chất lượng cao hoặc chụp ảnh tĩnh để phân tích.
 - **Tiền xử lý hình ảnh tức thời:** Thực hiện các thao tác tiền xử lý hình ảnh thời gian thực (on-the-fly) trước khi đưa vào mô hình học sâu. Các thao tác này bao gồm điều chỉnh kích thước đầu vào để khớp với yêu cầu của mô hình YOLOv8 (imgsz=1024 pixel), chuyển đổi không gian màu (ví dụ: RGB sang BGR nếu cần cho một số thư viện), và áp dụng các bộ lọc nhiễu/tăng cường (ví dụ: làm mịn ảnh để giảm nhiễu hạt, điều chỉnh độ tương phản cục bộ để làm nổi bật lỗi) để tối ưu hóa chất lượng hình ảnh đầu vào cho mô hình, đảm bảo tính nhất quán của dữ liệu.

- **Vẽ kết quả phát hiện:** Sau khi mô hình YOLOv8 trả về các hộp giới hạn và nhãn lỗi đã được dự đoán, OpenCV được sử dụng để vẽ trực quan các hộp giới hạn này lên hình ảnh gốc, cùng với nhãn lớp và điểm tự tin, giúp người dùng dễ dàng quan sát và xác định các lỗi đã được phát hiện một cách rõ ràng và trực quan.
- **Công cụ và Thư viện OCR:** Để thực hiện chức năng nhận dạng ký tự quang học, nhóm nghiên cứu đã lựa chọn sử dụng EasyOCR. Công cụ này đóng vai trò then chốt trong việc tự động hóa quá trình trích xuất thông tin văn bản từ hình ảnh container, giúp liên kết dữ liệu lỗi với các mã định danh cụ thể (ví dụ: mã số container), nâng cao hiệu quả quản lý thông tin và theo dõi lịch sử của từng container.
- **Phát triển giao diện Web:** Cung cấp một dashboard tổng quan toàn diện, khả năng quản lý dữ liệu lỗi, và các công cụ phân tích chuyên sâu. Các tính năng chính bao gồm dashboard hiển thị tổng quan về tình hình lỗi trên dây chuyền sản xuất (ví dụ: biểu đồ thống kê lỗi theo ngày/tuần, số lượng chi tiết đã kiểm tra), kho lưu trữ hình ảnh lỗi với khả năng tìm kiếm, chức năng quản lý cấu hình hệ thống hoặc tùy chỉnh ngưỡng phát hiện (nếu có), và báo cáo chi tiết về hiệu suất của hệ thống, hỗ trợ việc ra quyết định chiến lược và tối ưu hóa quy trình.

4.2. Thu thập và Xây dựng tập dữ liệu

Chất lượng và sự đa dạng của tập dữ liệu đóng vai trò nền tảng, quyết định khả năng học và hiệu suất hoạt động của mô hình. Trong đề tài, việc xây dựng tập dữ liệu được thực hiện cẩn thận, bao gồm các hình ảnh và nhãn tương ứng của các lỗi hư hỏng trên bề mặt container. Việc đảm bảo một tập dữ liệu đầy đủ, chính xác và đại diện là yếu tố quan trọng cho sự thành công của mô hình, ảnh hưởng trực tiếp đến khả năng tổng quát hóa và độ tin cậy của mô hình khi triển khai trong môi trường thực tế.


4.2.1. Tổng quan về Tập dữ liệu



Đầu ra của mô hình được thiết kế để nhận diện 3 lớp lỗi hư hỏng khác nhau trên bề mặt container. Các lớp lỗi này bao gồm:

- **Dent (móp méo):** Lỗi biến dạng, lõm vào hoặc lồi ra trên bề mặt container, thường do va chạm trong quá trình vận chuyển hoặc xếp dỡ, ảnh hưởng đến cấu trúc và thẩm mỹ.
- **Hole (thủng lỗ):** Lỗi xuyên thủng, tạo thành lỗ hổng trên thân container, có thể do ăn mòn nghiêm trọng, va đập mạnh hoặc lỗi sản xuất, gây nguy cơ rò rỉ hàng hóa.
- **Rusty (gỉ sét):** Lỗi ăn mòn kim loại, xuất hiện dưới dạng các mảng gỉ màu nâu đỏ trên bề mặt container, thường do tiếp xúc lâu dài với độ ẩm, nước biển và oxy, ảnh hưởng đến độ bền vật liệu.

Bảng 4.1 dưới đây tóm tắt các loại lỗi quan trọng của container trong tập dữ liệu của nghiên cứu:

Bảng 4.1: Các lỗi quan trọng của container trong tập dữ liệu

Loại lỗi hư hỏng	Hình ảnh
Dent	

Hole	
Rust	

Nhóm nghiên cứu đã thực hiện chia dữ liệu theo tỷ lệ sau, để đảm bảo cân bằng giữa các tập và tối ưu cho quá trình huấn luyện:

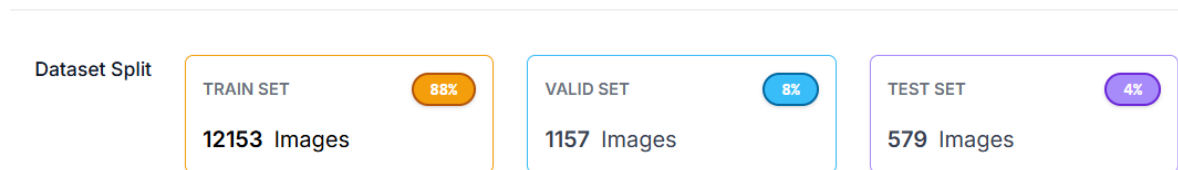
- Tập huấn luyện (Train Set): 4051 hình ảnh (chiếm 70% tổng số dữ liệu).
- Tập xác thực (Validation Set): 1157 hình ảnh (chiếm 20% tổng số dữ liệu).
- Tập kiểm tra (Test Set): 579 hình ảnh (chiếm 10% tổng số dữ liệu).

Khi nhóm thực hiện tăng cường dữ liệu thì đã được tỷ lệ của các tập như sau:

- Tập huấn luyện (Train Set): 12153 hình ảnh (chiếm 88% tổng số dữ liệu).
- Tập xác thực (Validation Set): 1157 hình ảnh (chiếm 8% tổng số dữ liệu).
- Tập kiểm tra (Test Set): 579 hình ảnh (chiếm 4% tổng số dữ liệu).

Tỷ lệ phân chia này được lựa chọn dựa trên sự cân bằng giữa việc cung cấp đủ dữ liệu cho mô hình học (88% cho huấn luyện) và đảm bảo các tập dữ liệu độc lập đủ lớn để đánh giá hiệu suất một cách tin cậy (8% và 4% cho xác thực và kiểm tra). Trong học sâu, việc dành phần lớn dữ liệu cho tập huấn luyện là ưu tiên hàng đầu, đặc biệt với các mô hình phức tạp như YOLOv8 và khi có một lượng lớn dữ liệu

(13827 hình ảnh). Điều này đảm bảo mô hình có đủ thông tin để học các đặc trưng phức tạp và quy luật ẩn sâu trong dữ liệu, tránh hiện tượng dưới khớp (underfitting). Đồng thời, tập xác thực và kiểm tra với kích thước lần lượt là 1157 và 579 hình ảnh vẫn đủ đa dạng và có ý nghĩa thống kê để đại diện cho các trường hợp thực tế và cung cấp các đánh giá tin cậy về khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy. Tỷ lệ phân chia dữ liệu được áp dụng sẽ được minh họa trong hình 4.1 dưới đây.



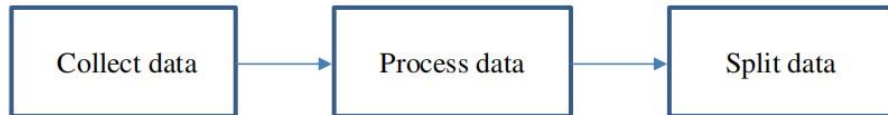
Hình 4.1: Tỷ lệ phân chia dữ liệu trong nghiên cứu.

Tập dữ liệu hình ảnh được sử dụng trong nghiên cứu này được thu thập từ các nguồn dữ liệu có sẵn trên nền tảng Roboflow [10], sau đó được nhóm nghiên cứu điều chỉnh, tinh lọc và bổ sung để phù hợp với mục tiêu đề tài. Dù không trực tiếp thu thập từ dây chuyền sản xuất của một đơn vị cụ thể, tập dữ liệu này vẫn được lựa chọn cẩn thận để nắm bắt các biến thể và điều kiện thực tế trong môi trường công nghiệp, nhằm tăng cường khả năng chống nhiễu và tổng quát hóa của mô hình. Các yếu tố môi trường được xem xét trong bộ dữ liệu bao gồm:

- Điều kiện ánh sáng khác nhau: Bộ dữ liệu chứa hình ảnh được chụp dưới nhiều điều kiện ánh sáng khác nhau (ví dụ: ánh sáng mạnh, bóng tối, ánh sáng yếu), giúp mô hình học cách nhận diện lỗi mà không bị ảnh hưởng bởi sự biến đổi về độ sáng hay phản xạ.
- Góc chụp đa dạng: Các hình ảnh trong bộ dữ liệu được chụp từ nhiều góc độ khác nhau, giúp mô hình nhận biết các lỗi dù chúng xuất hiện ở các vị trí, hướng hoặc phối cảnh khác nhau trên bề mặt.
- Bối cảnh và bề mặt khác nhau: Bộ dữ liệu bao gồm các hình ảnh với nhiều bối cảnh và đặc điểm bề mặt kim loại không phải lỗi (ví dụ: vết xước không phải lỗi, vân kim loại tự nhiên, bụi bẩn không đáng kể), giúp mô hình phân biệt lỗi với các đặc trưng nền không liên quan.

4.2.2. Quy trình Thu thập và Gán nhãn Dữ liệu

Quá trình xử lý dữ liệu từ khi thu thập đến khi sẵn sàng cho huấn luyện mô hình YOLOv8 bao gồm các bước quan trọng nhằm đảm bảo dữ liệu đạt chất lượng cao và đúng định dạng yêu cầu. Hình 4.2 mô tả tổng quan quy trình xử lý dữ liệu đã được thực hiện.



Hình 4.2: Quá trình xử lý dữ liệu

Các hình ảnh trong tập dữ liệu được thu thập từ hệ thống kiểm tra chất lượng tự động trên dây chuyền sản xuất các chi tiết kim loại (container). Quá trình thu thập này được thiết kế cẩn thận để nắm bắt các biến thể và điều kiện thực tế trong môi trường công nghiệp, nhằm tăng cường khả năng chống nhiễu và tổng quát hóa của mô hình. Các yếu tố môi trường được xem xét bao gồm:

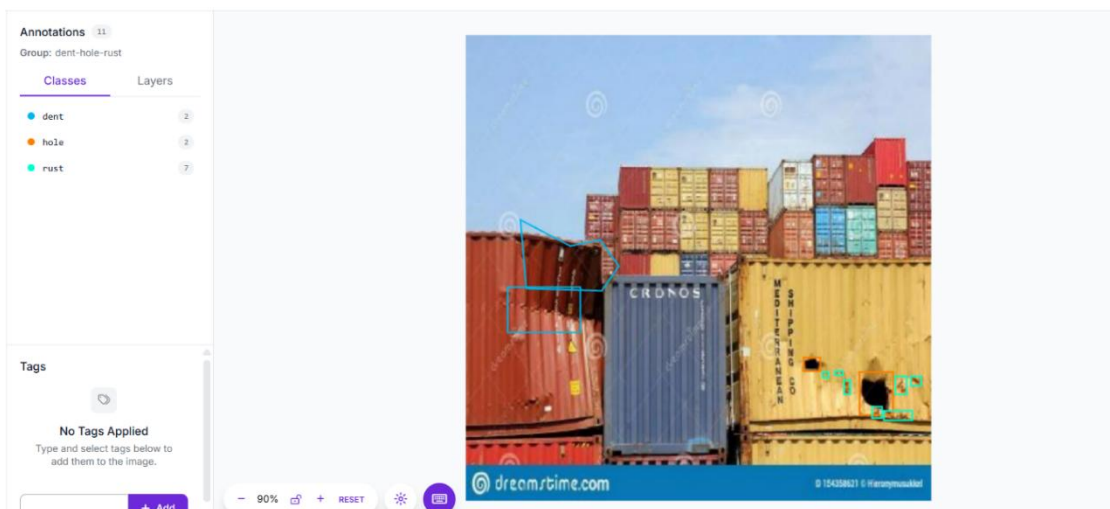
- Điều kiện ánh sáng khác nhau: Môi trường công nghiệp thường có sự thay đổi về cường độ và hướng chiếu sáng, từ ánh sáng mạnh trực tiếp đến các vùng bóng tối hoặc ánh sáng yếu. Việc thu thập hình ảnh dưới các điều kiện ánh sáng đa dạng giúp mô hình học cách nhận diện lỗi mà không bị ảnh hưởng bởi sự biến đổi về độ sáng hay phản xạ, từ đó giảm thiểu sự nhạy cảm với ánh sáng nền và nâng cao tính mạnh mẽ của hệ thống.
- Góc chụp đa dạng: Các chi tiết kim loại có thể được quan sát từ nhiều góc độ khác nhau trên dây chuyền sản xuất. Việc chụp ảnh từ các góc độ đa dạng giúp mô hình nhận biết các lỗi dù chúng xuất hiện ở các vị trí, hướng hoặc phối cảnh khác nhau trên bề mặt, đảm bảo khả năng phát hiện toàn diện và giảm thiểu bỏ sót lỗi do yếu tố hình học.
- Bối cảnh và bề mặt khác nhau: Mặc dù tập trung vào chi tiết kim loại của container, dữ liệu thu thập cũng bao gồm các bối cảnh xung quanh (ví dụ: các

bộ phận máy móc, môi trường nhà xưởng) hoặc các đặc điểm bề mặt kim loại không phải lỗi (ví dụ: vết xước không phải lỗi, vân kim loại tự nhiên, bụi bẩn không đáng kể). Việc này giúp mô hình học cách phân biệt rõ ràng giữa các loại lỗi mục tiêu và các đặc trưng nền không liên quan, nâng cao độ chính xác và giảm thiểu sai dương (phát hiện giả).

Việc thu thập dữ liệu đa dạng và phong phú dưới các điều kiện vận hành thực tế là yếu tố sống còn để đảm bảo tính tổng quát và khả năng chống nhiễu của mô hình khi triển khai trong môi trường sản xuất thực tế [14].

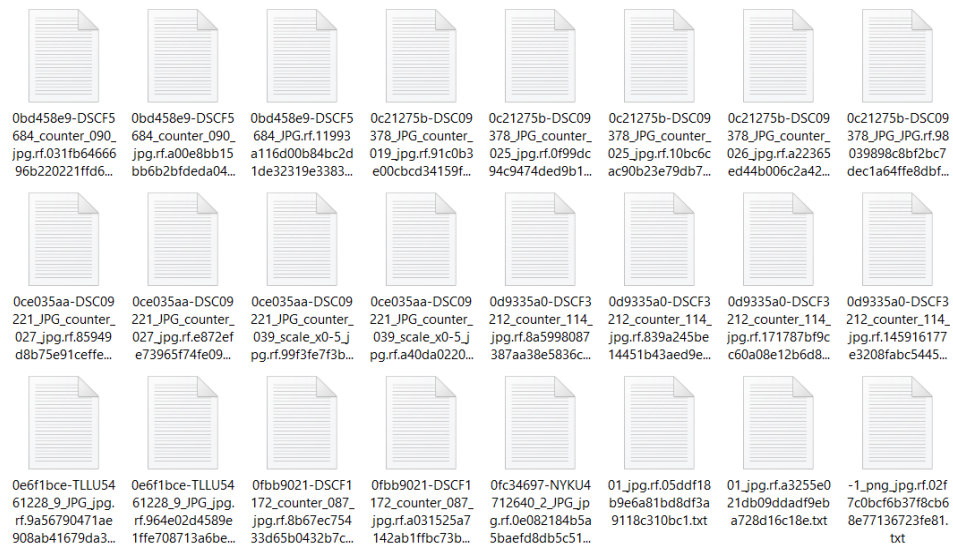
Sau khi thu thập bộ dữ liệu hình ảnh thô, nhóm nghiên cứu đã tiến hành gán nhãn cho các hình ảnh này thông qua giao diện trực quan của web Roboflow [10]. Roboflow là một nền tảng quản lý và tiền xử lý dữ liệu hiệu quả, cung cấp các công cụ mạnh mẽ để thực hiện việc gán nhãn một cách chính xác và hiệu quả cho các bài toán thị giác máy tính, đặc biệt là phát hiện đối tượng.

Hình 4.3 minh họa giao diện trên Roboflow, nơi các nhà nghiên cứu có thể vẽ các hộp giới hạn (bounding box) và gán nhãn cho từng đối tượng lỗi trên hình ảnh.



Hình 4.3: Giao diện trên ROBOFLOW

Sau khi hoàn tất quá trình gán nhãn trên Roboflow, hệ thống sẽ tự động xuất ra các tệp tọa độ nhãn tương ứng theo định dạng chuẩn của YOLO. Mỗi hình ảnh sẽ có một tệp nhãn (.txt) đi kèm, có tên giống với tệp hình ảnh nhưng với phần mở rộng là .txt. Hình 4.4 minh họa cấu trúc các tệp tọa độ dưới dạng .txt được tạo ra.



Hình 4.4: Các tệp tọa độ dưới dạng .txt

Mỗi tệp nhãn (.txt) chứa thông tin về tất cả các đối tượng lỗi có trong ảnh tương ứng. Mỗi dòng trong tệp nhãn sẽ biểu diễn thông tin của một đối tượng lỗi, bao gồm 5 giá trị: số đầu tiên (0, 1, 2) là chỉ số của lớp lỗi (tương ứng với Dent, Hole, Rust), và 4 số tiếp theo là tọa độ của hộp giới hạn theo định dạng YOLO (tọa độ trung tâm x, tọa độ trung tâm y, chiều rộng, chiều dài). Điều quan trọng cần lưu ý là các tọa độ đường bao này đã được chuẩn hóa về khoảng (0, 1), đây là yêu cầu chuẩn của mô hình YOLO để đảm bảo tính độc lập với kích thước ảnh gốc và giúp mô hình học các đặc trưng vị trí một cách tổng quát hơn, không phụ thuộc vào độ phân giải ảnh đầu vào.

Hình 4.5 minh họa cấu trúc thông tin tọa độ của một đối tượng trong tệp .txt mẫu.

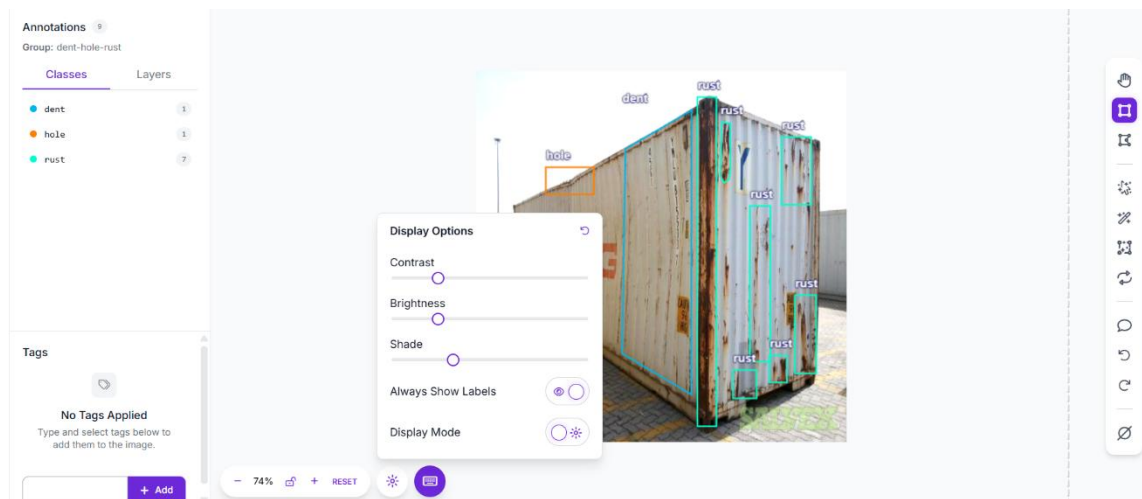
```
0 0.25859375 0.5375 0.1140625 0.19296875
1 0.26484375 0.5265625 0.0265625 0.053125
1 0.278125 0.36328125 0.334375 0.1203125
2 0.396875 0.7078125 0.115625 0.13984375
2 0.78359375 0.60234375 0.1484375 0.1859375
```

Hình 4.5: Thông tin tọa độ của đối tượng trong file .txt

4.2.3. Thách thức trong Quá trình Gán nhãn và Giải pháp

Quá trình gán nhãn dữ liệu thực tế luôn tiềm ẩn các thách thức có thể ảnh hưởng đến chất lượng tập dữ liệu, đặc biệt là với các lỗi nhỏ hoặc khó nhìn thấy trên bề mặt container. Việc nhận diện và giải quyết các thách thức này là rất quan trọng để đảm bảo tính chính xác và độ tin cậy của dữ liệu đã gán nhãn, từ đó tác động trực tiếp đến hiệu suất của mô hình học sâu:

- Kích thước lỗi hư hỏng nhỏ: Một số lỗi hư hỏng, đặc biệt là các lỗ thủng hoặc vết lõm nhỏ li ti, có kích thước rất bé trên hình ảnh độ phân giải cao. Điều này gây khó khăn đáng kể cho việc gán nhãn chính xác bằng mắt thường, dễ dẫn đến việc bỏ sót hoặc vẽ hộp giới hạn không chính xác. Giải pháp: Roboflow hỗ trợ các công cụ phóng to mạnh mẽ, cho phép người gán nhãn nhìn rõ hơn và vẽ hộp giới hạn tỉ mỉ ngay cả với các đối tượng cực nhỏ [10].
- Lỗi hư hỏng mờ hoặc có độ tương phản thấp: Trong một số hình ảnh, lỗi hư hỏng có thể bị mờ, bị che khuất một phần, hoặc không có sự khác biệt rõ rệt về độ tương phản so với bề mặt nền xung quanh. Điều này khiến việc nhận diện bằng mắt thường trở nên khó khăn và dễ dẫn đến bỏ sót hoặc gán nhãn sai. Giải pháp: Trong quá trình gán nhãn trên Roboflow, người gán nhãn đã chủ động sử dụng các tùy chọn hiển thị (Display Options) được cung cấp, bao gồm Contrast (Độ tương phản), Brightness (Độ sáng) và Shade (Đổ bóng) (như minh họa trong Hình 4.6). Việc điều chỉnh các thông số này giúp làm nổi bật các chi tiết lỗi mờ hoặc có độ tương phản thấp, từ đó tăng cường khả năng nhận diện thủ công và đảm bảo độ chính xác trong việc gán nhãn.



Hình 4.6: Tùy chọn hiển thị trong giao diện gắn nhãn của Roboflow

- Lỗi hư hỏng chồng chéo hoặc gần nhau: Các lỗi hư hỏng có thể xuất hiện rất gần nhau hoặc thậm chí chồng chéo lên nhau trên bề mặt container (ví dụ: nhiều vết gỉ sét nhỏ gần một vết móp lớn). Điều này đòi hỏi người gắn nhãn phải có khả năng phân định ranh giới rõ ràng cho từng hộp giới hạn, tránh việc gộp nhiều lỗi thành một hoặc tách một lỗi thành nhiều hộp, điều có thể gây nhầm lẫn cho mô hình trong quá trình học và làm giảm độ chính xác dự đoán. Giải pháp: Quy tắc gắn nhãn rõ ràng và việc kiểm tra chéo giữa các người gắn nhãn đã được áp dụng để đảm bảo tính nhất quán trong các trường hợp phức tạp này.

4.2.4. Phân tích Thống kê Tập dữ liệu

Để hiểu rõ hơn về tính chất của tập dữ liệu và chuẩn bị cho việc huấn luyện mô hình, việc phân tích thống kê số lượng thể hiện (instances) của từng loại lỗi trong các tập dữ liệu là cần thiết. Tổng số hình ảnh trong tập dữ liệu là **13827**, và phân bố các lỗi được thể hiện chi tiết trong Bảng 4.2 dưới đây:

Bảng 4.2: Phân bố số lượng thể instances của các lớp lỗi trong các tập dữ liệu.

Class Name	Total Count	Training Count	Validation Count	Test Count

rust	8091	5517	1860	714
dent	4744	3331	959	454
hole	1194	836	241	117

Như Bảng 4.2 cho thấy, lớp rust chiếm số lượng thể hiện lớn nhất trong tổng số dữ liệu (8091 instances), tiếp theo là dent (4744 instances) và hole (1194 instances). Sự phân bố này chỉ ra một sự **mất cân bằng đáng kể trong tập dữ liệu (class imbalance)**. Cụ thể, lớp hole là lớp thiểu số với số lượng mẫu ít nhất đáng kể so với rust và dent trên cả ba tập huấn luyện, xác thực và kiểm tra.

Sự mất cân bằng lớp là một thách thức phổ biến trong các bài toán phân loại và phát hiện đối tượng, nơi số lượng mẫu của một lớp (lớp thiểu số) ít hơn đáng kể so với các lớp khác (lớp đa số) [13], [14]. Nếu không được xử lý, mô hình có xu hướng thiên vị các lớp đa số, dẫn đến hiệu suất kém trên các lớp thiểu số do không có đủ mẫu để học các đặc trưng đa dạng của chúng. Để giải quyết vấn đề này, một số giải pháp tiềm năng đã được xem xét hoặc áp dụng trong quá trình tiền xử lý và tăng cường dữ liệu (được trình bày chi tiết tại Mục 4.3.2), bao gồm:

- **Tăng cường dữ liệu có trọng số:** Áp dụng các kỹ thuật tăng cường dữ liệu nhiều hơn cho các lớp thiểu số để tạo ra các biến thể mới của dữ liệu hiện có, từ đó tăng số lượng mẫu cho lớp đó một cách hiệu quả (ví dụ: thông qua các tham số tăng cường trong YOLOv8).
- **Điều chỉnh hàm mất mát (loss function):** Sử dụng các hàm mất mát có trọng số (weighted loss) hoặc các kỹ thuật như Focal Loss. Các hàm này gán trọng số cao hơn cho lỗi dự đoán sai trên các lớp thiểu số, khuyến khích mô hình tập trung học các đặc trưng của chúng.
- **Kỹ thuật lấy mẫu (sampling techniques):** Bao gồm oversampling (tăng số lượng mẫu lớp thiểu số bằng cách nhân bản hoặc tạo mẫu tổng hợp) hoặc

undersampling (giảm số lượng mẫu lớp đa số). Tuy nhiên, undersampling có thể dẫn đến mất mát thông tin quan trọng [14].

- **Thu thập thêm dữ liệu:** Cách hiệu quả nhất nhưng cũng tốn kém nhất là thu thập thêm hình ảnh thực tế cho các lớp thiểu số.

Việc hiểu rõ về phân bố lớp giúp đưa ra các điều chỉnh phù hợp trong quá trình huấn luyện và đánh giá để đạt được hiệu suất cân bằng cho tất cả các loại lỗi, đặc biệt là các lỗi ít phổ biến hơn nhưng có thể quan trọng trong thực tế.

4.3. Tiền xử lý dữ liệu và Tăng cường dữ liệu

Tiền xử lý dữ liệu và tăng cường dữ liệu là các bước không thể thiếu trong bất kỳ dự án học máy nào, đặc biệt là trong thị giác máy tính. Mục đích chính của tiền xử lý là chuẩn hóa dữ liệu đầu vào, loại bỏ nhiễu, làm nổi bật các đặc trưng quan trọng và đưa dữ liệu về định dạng phù hợp nhất cho mô hình học sâu [14], [15]. Tăng cường dữ liệu (Data Augmentation) giúp mở rộng tập huấn luyện một cách nhân tạo, tăng tính đa dạng của dữ liệu mà vẫn giữ nguyên nhãn lớp, từ đó giúp mô hình trở nên mạnh mẽ hơn, giảm thiểu hiện tượng quá khớp (overfitting) và cải thiện khả năng tổng quát hóa trên dữ liệu chưa từng thấy [11], [14].

4.3.1. Tiền xử lý dữ liệu

Quá trình tiền xử lý hình ảnh bao gồm một loạt các thao tác được áp dụng cho dữ liệu ảnh thô trước khi chúng được đưa vào mạng nơ-ron. Mục tiêu là chuẩn hóa kích thước, giảm nhiễu, tăng cường đặc trưng và tối ưu hóa tài nguyên tính toán. Trong nghiên cứu này, các bước tiền xử lý được thực hiện thông qua nền tảng **Roboflow**, cung cấp một quy trình tự động hóa và hiệu quả [10].

Các kỹ thuật tiền xử lý đã áp dụng và phân tích bao gồm:

- **Auto-Orient:** Chức năng này tự động xoay hình ảnh về đúng hướng chuẩn dựa trên dữ liệu EXIF metadata được lưu trữ trong tệp ảnh [10]. Đảm bảo tính nhất quán về hướng của tất cả các hình ảnh, loại bỏ sự cần thiết phải xoay thủ công từng hình ảnh.

- **Resize (Thay đổi kích thước):** Thay đổi kích thước hình ảnh đầu vào để phù hợp với kích thước yêu cầu của mô hình học sâu. Cụ thể, chúng tôi sử dụng `imgsz=1024` pixel với phương pháp **Fit** (giữ nguyên tỷ lệ khung hình gốc, đệm đen các vùng trống). Phương pháp này được ưu tiên hơn so với **Stretch** để giảm thiểu biến dạng đối tượng, điều này đặc biệt quan trọng với các lỗi có hình dạng đặc trưng như lỗ thủng hay vết lõm. Kích thước đầu vào 1024x1024 giúp mô hình có khả năng nhìn thấy nhiều chi tiết hơn, cải thiện khả năng phát hiện các lỗi nhỏ [9], [12].

4.3.2. Các Phương pháp Tăng cường Dữ liệu (Data Augmentations)

Tăng cường dữ liệu (Data Augmentation) là một kỹ thuật thiết yếu trong học sâu, đặc biệt khi làm việc với tập dữ liệu có kích thước hạn chế, hoặc để tăng tính tổng quát của mô hình. Nó mở rộng tập huấn luyện một cách nhân tạo bằng cách tạo ra các biến thể mới của dữ liệu hiện có mà vẫn giữ nguyên nhãn lớp, từ đó giúp mô hình trở nên mạnh mẽ hơn, giảm thiểu hiện tượng quá khớp (overfitting) và cải thiện khả năng tổng quát hóa trên dữ liệu chưa từng thấy [11], [14]. Trong bài toán phát hiện lỗi hư hỏng trên bề mặt kim loại, việc tăng cường dữ liệu đặc biệt quan trọng vì các lỗi có thể xuất hiện dưới nhiều điều kiện khác nhau (góc nhìn, ánh sáng, kích thước, biến dạng) trong môi trường công nghiệp thực tế.

- **Các Kỹ thuật Tăng cường Hình học (Geometric Augmentations):** Các phép biến đổi hình học làm thay đổi vị trí, kích thước, hoặc hình dạng của đối tượng trong hình ảnh, giúp mô hình học cách nhận diện đối tượng bất kể các biến đổi vị trí và không gian [11].
 - **Lật (Flip):** Lật hình ảnh theo chiều ngang (Horizontal) và/hoặc chiều dọc (Vertical) [10]. Giúp mô hình học các đặc trưng mà không phụ thuộc vào hướng cụ thể của đối tượng trong không gian 2D.
 - **Xoay (Rotation):** Xoay hình ảnh ngẫu nhiên trong một khoảng góc nhất định (ví dụ, từ -15° đến $+15^\circ$) [10]. Hữu ích khi các lỗi có thể xuất hiện với các hướng ngẫu nhiên trên bề mặt container.

- Cắt xén (Shear): Áp dụng biến dạng cắt xén (ngiên lệch) hình ảnh theo chiều ngang và/hoặc chiều dọc (ví dụ, $\pm 10^\circ$) [10]. Mô phỏng các góc nhìn hơi nghiêng hoặc biến dạng phối cảnh của vật thể.
- Bounding Box (cho các phép biến đổi trên): Đảm bảo bounding box cũng được điều chỉnh tương ứng với ảnh gốc sau khi Flip, Rotation, Shear.
- Các Kỹ thuật Tăng cường Màu sắc (Color Augmentations): Các phép biến đổi màu sắc làm thay đổi đặc điểm về ánh sáng, độ bão hòa, và sắc thái của hình ảnh, giúp mô hình học cách nhận diện đối tượng bất kể sự biến đổi về điều kiện ánh sáng hoặc màu sắc thực tế [11].
 - Điều chỉnh Sắc thái (Hue): Thay đổi sắc thái (màu cơ bản) của hình ảnh trong một khoảng nhất định (ví dụ, $hsv_h=0.015$) [9]. Giúp mô hình ít nhạy cảm hơn với sự thay đổi nhỏ về màu sắc.
 - Điều chỉnh Độ bão hòa (Saturation): Thay đổi độ bão hòa (cường độ màu) của hình ảnh trong một khoảng nhất định (ví dụ, $hsv_s=0.7$) [9]. Mô phỏng sự khác biệt về độ bão hòa màu sắc do cường độ ánh sáng hoặc các đặc tính vật liệu khác nhau.
 - Điều chỉnh Độ sáng (Brightness): Thay đổi độ sáng của hình ảnh trong một khoảng nhất định (ví dụ, $hsv_v=0.4$) [9]. Giúp mô hình học cách nhận diện đối tượng dưới các điều kiện chiếu sáng khác nhau (sáng hơn, tối hơn).
- Các Kỹ thuật Tăng cường Dữ liệu Nâng cao:
 - Mosaic Augmentation: Một kỹ thuật tăng cường dữ liệu đặc trưng và hiệu quả trong YOLOv8 [9]. Kỹ thuật này kết hợp bốn hình ảnh huấn luyện riêng biệt thành một hình ảnh duy nhất để tăng cường phát hiện đối tượng nhỏ và đa dạng hóa ngữ cảnh ($mosaic=1.0$). Về mặt lý thuyết, nó giúp tăng batch size hiệu quả mà không tốn thêm bộ nhớ GPU.

- Mixup Augmentation: Tạo ra các hình ảnh huấn luyện mới bằng cách kết hợp tuyến tính hai hình ảnh ngẫu nhiên cùng nhãn (mixup=0.2), giúp làm mịn đường biên quyết định và cải thiện khả năng tổng quát hóa [9].

4.4. Nhận dạng Ký tự Quang học (OCR) và Tích hợp

OCR tự động trích xuất văn bản từ ảnh container (mã số, thông tin vận chuyển), liên kết thông tin này với dữ liệu lỗi để nâng cao quản lý và theo dõi.

4.4.1. Tổng quan về OCR

OCR là lĩnh vực thị giác máy tính kết hợp xử lý ảnh và nhận dạng mẫu. Quá trình gồm tiền xử lý, phân đoạn, nhận dạng và hậu xử lý ký tự. Các hệ thống OCR hiện đại thường dùng CNN và RNN.

4.4.2. Phương pháp OCR áp dụng

Để nhận dạng ký tự, nhóm sử dụng **EasyOCR**. EasyOCR là một thư viện Python linh hoạt, hỗ trợ GPU và nhiều ngôn ngữ, thường cho kết quả tốt trên các văn bản đa dạng và có thể tích hợp dễ dàng vào các luồng xử lý hình ảnh.

- **Quy trình tích hợp EasyOCR:**

- **Thu thập vùng văn bản:** Sau khi hình ảnh container được chụp và xử lý bởi mô hình phát hiện đối tượng (nếu cần để xác định vị trí văn bản), vùng ảnh chứa văn bản cần nhận dạng được cắt ra.
- **Tiền xử lý văn bản cho OCR:** Vùng ảnh này có thể trải qua các bước tiền xử lý bổ sung như chuyển đổi sang ảnh xám, điều chỉnh độ tương phản, làm nhị phân hóa (binarization) hoặc làm thẳng ảnh bị nghiêng (deskewing) để tối ưu hóa đầu vào cho thuật toán EasyOCR, giúp cải thiện độ chính xác nhận dạng.
- **Nhận dạng ký tự:** Thuật toán EasyOCR được gọi để xử lý vùng ảnh đã tiền xử lý và trích xuất chuỗi ký tự. EasyOCR sử dụng các mô hình học sâu cho cả phát hiện vùng văn bản và nhận dạng ký tự.

- **Hậu xử lý và chuẩn hóa:** Chuỗi ký tự thô được trích xuất sẽ trải qua bước hậu xử lý để loại bỏ ký tự nhiễu, sửa lỗi chính tả dựa trên các quy tắc hoặc từ điển, và định dạng lại thông tin (ví dụ: chuẩn hóa mã số container theo định dạng BIC code) để đảm bảo tính chính xác và phù hợp cho việc lưu trữ và phân tích trong hệ thống quản lý.
- **Ví dụ kết quả OCR:**
- **Hình 4.6: Ví dụ về kết quả nhận dạng OCR và phát hiện lỗi (rust) trên bề mặt container.**

Hình 4.6 cho thấy một ví dụ thực tế về việc mô hình phát hiện lỗi rust với độ tin cậy 0.90 trên bề mặt container, đồng thời minh họa khả năng tích hợp OCR (khu vực trống bên dưới có thể dành cho kết quả OCR trích xuất thông tin liên quan).

4.4.3. Vai trò của OCR trong hệ thống

Module OCR đóng vai trò đa dạng và bổ trợ quan trọng trong hệ thống phát hiện lỗi container:

- **Liên kết thông tin lỗi với container cụ thể:** Bằng cách trích xuất mã định danh container (BIC code) hoặc số sê-ri từ hình ảnh, hệ thống có thể tự động liên kết các lỗi được phát hiện (Dent, Hole, Rusty) với container cụ thể. Điều này giúp theo dõi lịch sử lỗi của từng container, hỗ trợ công tác bảo trì và quản lý tài sản.
- **Tự động hóa nhập liệu:** Giảm sai sót và tăng tốc độ xử lý dữ liệu.
- **Hỗ trợ kiểm kê và logistics:** Tự động cập nhật trạng thái container.
- **Bằng chứng và báo cáo:** Lưu trữ hình ảnh kèm thông tin OCR và lỗi để kiểm toán hoặc phân tích.

4.5. Mô hình học sâu và Môi trường huấn luyện

4.5.1. Kiến trúc Mô hình và Lựa chọn YOLOv8m.pt

Trong nghiên cứu này, mô hình được lựa chọn để thực nghiệm là **YOLOv8m.pt**. Sự lựa chọn này dựa trên việc cân nhắc kỹ lưỡng giữa hiệu suất đạt được (độ chính xác và tốc độ) và yêu cầu về tài nguyên tính toán, đặc biệt là với mục tiêu triển khai trên thiết bị nhúng như Raspberry Pi 4. Ultralytics cung cấp một loạt các phiên bản YOLOv8 với các kích thước và độ phức tạp khác nhau (từ nhỏ gọn (n - nano, s - small) đến lớn và mạnh mẽ hơn (m - medium, l - large, x - xlarge)). Mỗi phiên bản đại diện cho một sự đánh đổi khác nhau giữa tốc độ và độ chính xác, cho phép lựa chọn phù hợp với các yêu cầu cụ thể của ứng dụng [9].

YOLOv8m.pt (m - medium) được chọn vì nó cung cấp một sự cân bằng tối ưu giữa tốc độ suy luận (inference speed) và độ chính xác (accuracy) cho bài toán phát hiện lỗi hư hỏng trên bề mặt kim loại. Các phiên bản nhỏ hơn như YOLOv8n hoặc YOLOv8s, mặc dù nhanh hơn và nhẹ hơn, có thể không đạt được độ chính xác mong muốn cho việc phát hiện các lỗi nhỏ hoặc khó nhận diện, vốn là đặc thù của các vết móp hay lỗ thủng li ti trên container. Ngược lại, các phiên bản lớn hơn như YOLOv8l hoặc YOLOv8x có thể mang lại độ chính xác cao hơn, nhưng đi kèm với chi phí tính toán lớn hơn và tốc độ suy luận chậm hơn đáng kể, làm cho việc triển khai trên thiết bị nhúng tài nguyên hạn chế như Raspberry Pi trở nên khó khăn hoặc không khả thi cho các ứng dụng yêu cầu thời gian thực [16], [17], [18]. YOLOv8m được kỳ vọng sẽ cung cấp đủ khả năng phát hiện mạnh mẽ mà vẫn duy trì tốc độ xử lý chấp nhận được, phù hợp với yêu cầu của hệ thống tự động kiểm tra.

YOLOv8m, mặc dù có kích thước lớn hơn các phiên bản nano hoặc small, vẫn nằm trong phạm vi các mô hình có thể được tối ưu hóa (ví dụ: thông qua lượng tử hóa sau huấn luyện) để chạy hiệu quả trên CPU hoặc GPU tích hợp của Raspberry Pi [16], [17], [18]. Sự cân bằng của YOLOv8m giúp nhóm có một mô hình đủ mạnh mẽ để đạt được độ chính xác cần thiết, đồng thời vẫn giữ được tiềm năng cho các bước tối ưu hóa sau huấn luyện để phù hợp với môi trường thiết bị nhúng.

Mô hình YOLOv8m.pt được khởi tạo với trọng số đã được huấn luyện trước trên các tập dữ liệu lớn và đa dạng như COCO (Common Objects in Context) [9]. Kỹ thuật này được gọi là Học chuyển giao (Transfer Learning) [1]. Việc sử dụng trọng số đã huấn luyện trước giúp mô hình không cần phải học lại từ đầu các đặc trưng cơ bản của hình ảnh (như cạnh, kết cấu, hình dạng) mà đã được học từ hàng triệu hình ảnh trong tập dữ liệu lớn và phong phú. Điều này mang lại nhiều lợi ích: Tăng tốc độ hội tụ của quá trình huấn luyện trên tập dữ liệu container chuyên biệt, cải thiện hiệu suất ngay cả khi tập dữ liệu riêng có kích thước không quá lớn, và giảm thiểu hiện tượng quá khớp do trọng số đã có tính tổng quát hóa tốt từ một miền dữ liệu rộng hơn.

4.5.2. Môi trường Huấn luyện

Quá trình huấn luyện mô hình được tiến hành trên nền tảng điện toán đám mây Google Colaboratory (Google Colab). Google Colab là một dịch vụ dựa trên Jupyter Notebook hoàn toàn miễn phí từ Google, cho phép người dùng viết và chạy mã Python trực tiếp trên trình duyệt mà không cần bất kỳ cấu hình phức tạp nào.

Ưu điểm chính của Google Colab cho các tác vụ học sâu là khả năng truy cập GPU miễn phí. Cụ thể trong nghiên cứu này, các phiên huấn luyện đã may mắn tận dụng được sức mạnh của GPU NVIDIA A100 [16], [17]. Đây là một trong những GPU hàng đầu thế giới của NVIDIA, được thiết kế đặc biệt cho các tác vụ tính toán hiệu năng cao và học sâu. A100 nổi bật với Tensor Cores thế hệ 3 (tăng tốc đáng kể các phép toán ma trận cho học sâu), bộ nhớ HBM2e dung lượng lớn và băng thông cao (giúp xử lý các mô hình lớn và lô dữ liệu lớn hiệu quả hơn), và khả năng Multi-Instance GPU (MIG) cho phép chia nhỏ A100 thành các instance nhỏ hơn để phục vụ nhiều người dùng hoặc tác vụ song song. Việc sử dụng A100 giúp giảm đáng kể thời gian huấn luyện mô hình (từ vài ngày xuống vài giờ), cho phép thực hiện nhiều thử nghiệm và tinh chỉnh siêu tham số trong một khoảng thời gian ngắn. Môi trường Colab được cấu hình sẵn với các thư viện học sâu phổ biến (TensorFlow, PyTorch, YOLOv8) và các driver cần thiết, giảm gánh nặng cài đặt cho người dùng. Khả năng

tích hợp dễ dàng với Roboflow để tải dữ liệu cũng tạo ra một quy trình làm việc liền mạch từ chuẩn bị dữ liệu đến huấn luyện mô hình.

4.5.3. Cấu hình và Siêu tham số Huấn luyện chi tiết

Việc tinh chỉnh các siêu tham số (hyperparameters) là một quá trình lặp lại và quan trọng để tối ưu hóa quá trình học của mô hình. Các siêu tham số này được lựa chọn và tinh chỉnh dựa trên các thực nghiệm ban đầu, kinh nghiệm từ các nghiên cứu trước đây về YOLOv8 [9], và mục tiêu đạt được hiệu suất tốt nhất cho bài toán phát hiện lỗi hư hỏng trên bề mặt kim loại. Dưới đây là giải thích chi tiết về từng tham số được sử dụng trong câu lệnh huấn luyện mô hình YOLOv8:

```
!yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml epochs=100 imgsz=1024 batch=16 lr=0.002 optimizer=SGD weight_decay=0.0005 momentum=0.937 iou=0.6 warmup_epochs=3 warmup_momentum=0.8 warmup_bias_lr=0.1 patience=10 cos_lr=True hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 translate=0.1 scale=0.5 flipr=0.5 mosaic=1.0 mixup=0.2 plots=True
```

- `task=detect`: Tham số này xác định nhiệm vụ chính của mô hình là phát hiện đối tượng (object detection) [9]. Điều này đảm bảo mô hình sẽ học cách xác định vị trí và phân loại đối tượng trong hình ảnh.
- `mode=train`: Tham số này chỉ định chế độ hoạt động của chương trình là huấn luyện (training) mô hình, cho phép mô hình học từ dữ liệu.
- `model=yolov8m.pt`: Tham số này chỉ định kiến trúc mô hình được sử dụng là YOLOv8 phiên bản cỡ trung bình (m - medium), được khởi tạo từ trọng số đã được huấn luyện trước (pre-trained weights) trên tập dữ liệu COCO [9]. Việc này tận dụng lợi ích của học chuyển giao.
- `data={dataset.location}/data.yaml`: Tham số này chỉ định đường dẫn đến tệp cấu hình dữ liệu (data.yaml), chứa tất cả thông tin cần thiết về tập dữ liệu, bao gồm đường dẫn đến các thư mục chứa hình ảnh và nhãn, cũng như danh sách tên và số lượng các lớp đối tượng (dent, hole, rust) [9].

- epochs=100: Xác định số lượng epoch (kỷ nguyên) mà mô hình sẽ được huấn luyện. Một epoch hoàn thành khi toàn bộ tập huấn luyện đã được đi qua một lần [14]. Lựa chọn 100 epoch đảm bảo mô hình có đủ thời gian để hội tụ.
- imgsz=1024: Thiết lập kích thước đầu vào của hình ảnh cho mạng nơ-ron là 1024x1024 pixel [9]. Kích thước đầu vào lớn hơn thường giúp mô hình nhìn thấy nhiều chi tiết hơn, đặc biệt quan trọng cho việc phát hiện các đối tượng nhỏ như các vết lỗi li ti trên bề mặt container [12].
- batch=16: Xác định kích thước lô (batch size) là 16. Mỗi lần cập nhật trọng số của mô hình sẽ dựa trên gradient được tính toán từ 16 hình ảnh cùng một lúc [14]. Kích thước lô vừa phải giúp cân bằng giữa tốc độ huấn luyện và khả năng tổng quát hóa.
- lr0=0.002: Thiết lập tốc độ học ban đầu (initial learning rate) là 0.002 [9]. Một tốc độ học được chọn cẩn thận để đảm bảo mô hình hội tụ hiệu quả mà không bị phân kỳ.
- optimizer=SGD: Chỉ định thuật toán tối ưu hóa được sử dụng là Stochastic Gradient Descent (SGD) [9]. SGD là một thuật toán phổ biến và hiệu quả cho các mạng nơ-ron lớn.
- weight_decay=0.0005: Thiết lập giá trị cho giảm trọng số (L2 regularization) là 0.0005 [9]. Tham số này giúp kiểm soát độ phức tạp của mô hình và ngăn ngừa quá khớp bằng cách hạn chế độ lớn của các trọng số.
- momentum=0.937: Thiết lập giá trị momentum cho thuật toán SGD là 0.937 [9]. Momentum giúp tăng tốc độ hội tụ bằng cách tích lũy gradient từ các bước trước, giúp vượt qua các điểm cực tiểu cục bộ và đi đến điểm tối ưu toàn cục nhanh hơn.
- iou=0.6: Xác định ngưỡng Intersection over Union (IoU) tối thiểu để một dự đoán hộp giới hạn được coi là đúng dương (True Positive) [9]. Ngưỡng IoU

cao hơn (ví dụ 0.6 thay vì 0.5) yêu cầu sự chính xác vị trí cao hơn để được tính là phát hiện đúng.

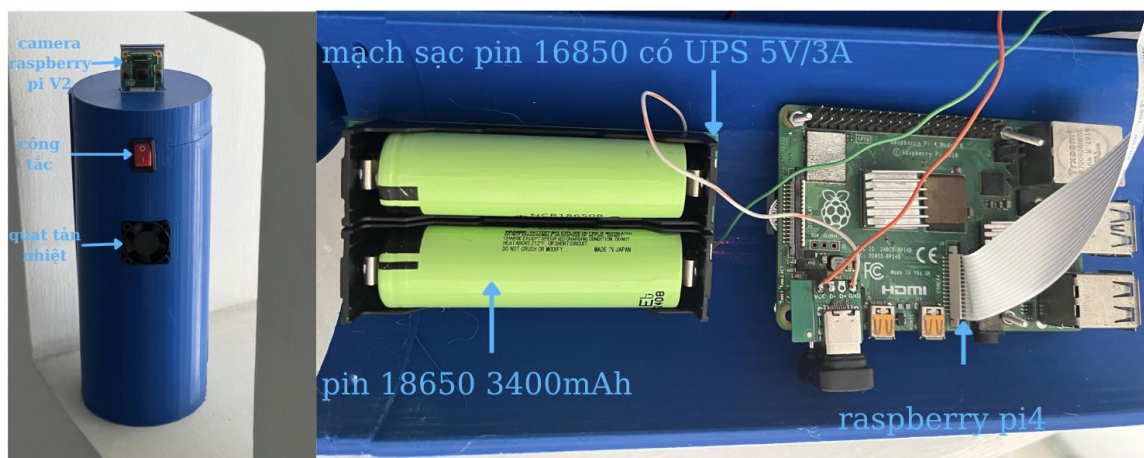
- `warmup_epochs=3`: Chỉ định số lượng epoch cho giai đoạn warm-up của tốc độ học là 3 [9]. Trong giai đoạn này, tốc độ học sẽ tăng dần từ một giá trị rất nhỏ lên `lr0`, giúp ổn định quá trình huấn luyện ban đầu.
- `warmup_momentum=0.8`: Thiết lập giá trị momentum ban đầu trong giai đoạn warm-up là 0.8 [9].
- `warmup_bias_lr=0.1`: Thiết lập tốc độ học ban đầu dành riêng cho các bias trong giai đoạn warm-up là 0.1 [9].
- `patience=10`: Thiết lập giá trị kiên nhẫn (patience) cho cơ chế dừng sớm (Early Stopping) là 10 [9]. Cơ chế này sẽ tự động dừng quá trình huấn luyện nếu hiệu suất trên tập xác thực không cải thiện trong 10 epoch liên tiếp, nhằm ngăn ngừa quá khớp.
- `cos_lr=True`: Kích hoạt lịch trình tốc độ học cosine annealing (Cosine Annealing Learning Rate Scheduler) [9]. Lịch trình này giúp tốc độ học giảm dần theo hàm cosine, cho phép các bước lớn hơn ở đầu quá trình để khám phá không gian tham số, sau đó giảm dần để tinh chỉnh các trọng số một cách cẩn thận hơn khi mô hình tiếp cận điểm tối ưu. Điều này góp phần vào sự giảm mất mát mịn màng và ổn định.
- `hsv_h=0.015`, `hsv_s=0.7`, `hsv_v=0.4`: Đây là các tham số điều khiển mức độ biến đổi trong các kỹ thuật tăng cường dữ liệu liên quan đến không gian màu HSV (Hue, Saturation, Value) [9]. Các biến đổi này giúp mô hình ít nhạy cảm hơn với sự thay đổi về điều kiện ánh sáng và màu sắc thực tế.
- `translate=0.1`, `scale=0.5`, `fliplr=0.5`: Đây là các tham số cho kỹ thuật tăng cường dữ liệu liên quan đến các biến đổi hình học cơ bản: dịch chuyển (ngẫu nhiên 10% ảnh), thay đổi tỷ lệ (ngẫu nhiên 50%), và lật ngang (xác suất 0.5)

[9]. Các biến đổi này giúp mô hình học cách nhận diện đối tượng bất kể vị trí, kích thước hay hướng của chúng.

- mosaic=1.0: Bất tính năng tăng cường dữ liệu Mosaic Augmentation với xác suất 1.0 (luôn áp dụng) [9]. Kỹ thuật này kết hợp bốn hình ảnh huấn luyện riêng biệt thành một hình ảnh duy nhất, giúp tăng cường phát hiện đối tượng nhỏ và đa dạng hóa ngữ cảnh.
- mixup=0.2: Bất tính năng tăng cường dữ liệu Mixup Augmentation với xác suất 0.2 [9]. Kỹ thuật này tạo ra các hình ảnh mới bằng cách kết hợp tuyến tính hai hình ảnh cùng nhãn, giúp làm mịn đường biên quyết định và cải thiện khả năng tổng quát hóa.

Chương 5. KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ

5.1. Hiện thực phần cứng



Hình 5.1: Hình ảnh bên ngoài và bên trong của phần cứng

Hình 5.1 là hình ảnh phần cứng được nhóm hiện thực hóa từ những thành phần hệ thống đã được tìm hiểu ở mục 3.2: Thành phần hệ thống. Camera Raspberry Pi V2 được đặt ở một vị trí thuận lợi để chụp ảnh hư hỏng.

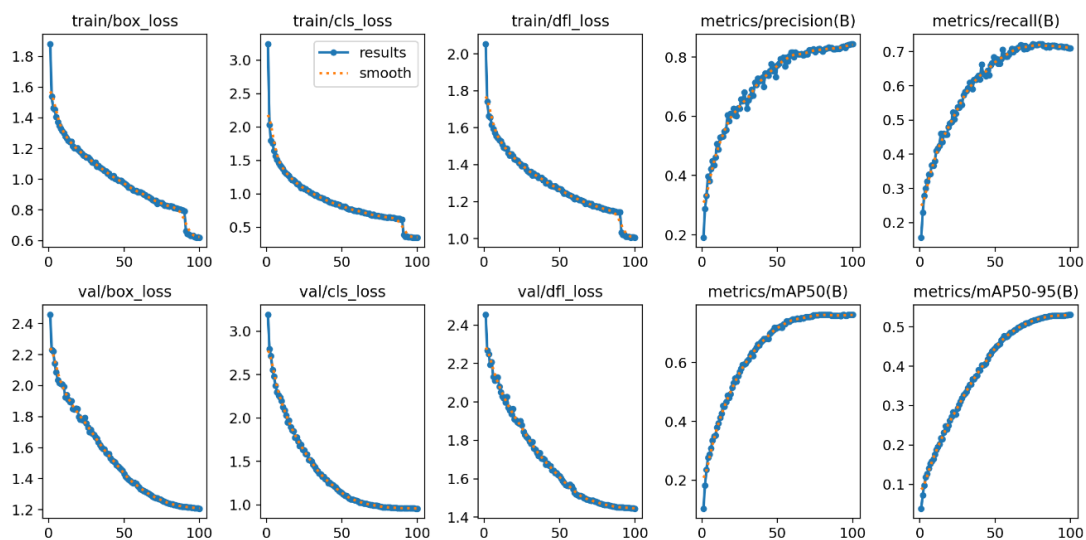
5.2. Kết quả Huấn luyện Mô hình

Phần này trình bày và phân tích chi tiết kết quả của toàn bộ quá trình huấn luyện mô hình YOLOv8m.pt. Mục tiêu là để đánh giá khả năng học hỏi của mô hình từ tập dữ liệu, xác định các hiện tượng kỹ thuật như quá khớp hay dưới khớp, và lý giải vai trò của các tham số đã được lựa chọn. Quá trình này được thực hiện qua hai giai đoạn chính: giai đoạn huấn luyện nền tảng và giai đoạn tinh chỉnh để lựa chọn mô hình cuối cùng.

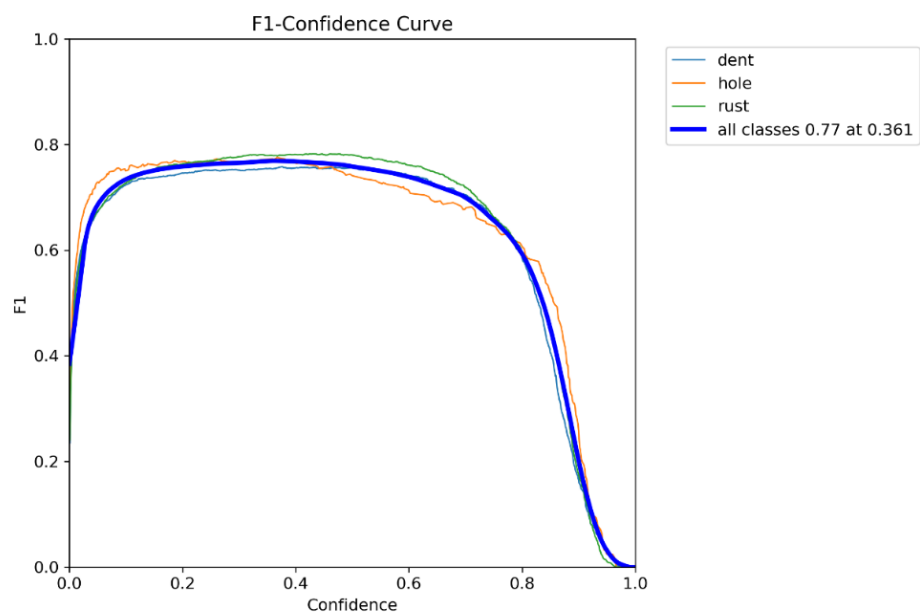
5.2.1. Quá trình Huấn luyện và Đường cong Mất mát (Loss Curves)

Giai đoạn đầu tiên tập trung vào việc huấn luyện mô hình trong 100 epoch từ trọng số khởi tạo của tập dữ liệu COCO, nhằm xây dựng một mô hình nền tảng với hiệu suất tốt. Quá trình này được theo dõi chặt chẽ thông qua các đường cong mất mát và chỉ số hiệu suất trên cả tập huấn luyện và tập xác thực. Các biểu đồ và nhật ký

huấn luyện trong Hình 5.2, Hình 5.3, và Hình 5.4 cung cấp cái nhìn toàn diện và trực quan về quá trình này.



Hình 5.2: Biểu đồ kết quả huấn luyện của mô hình YOLOv8m.pt.



Hình 5.3: Biểu đồ đường cong F1-Confidence.

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
98/100	17.3G	0.6192	0.3457	1.008	12	1024: 100% 760/760 [03:05<00:00, 4.09it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 37/37 [00:08<00:00, 4.34it/s]
all		1157	3060	0.843	0.712	0.763 0.53
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
99/100	17.2G	0.624	0.3438	1.007	25	1024: 100% 760/760 [03:05<00:00, 4.09it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 37/37 [00:08<00:00, 4.33it/s]
all		1157	3060	0.844	0.711	0.763 0.531
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
100/100	17.3G	0.6206	0.3454	1.005	20	1024: 100% 760/760 [03:05<00:00, 4.09it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 37/37 [00:08<00:00, 4.30it/s]
all		1157	3060	0.845	0.71	0.763 0.531

100 epochs completed in 5.468 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 52.1MB
Optimizer stripped from runs/detect/train/weights/best.pt, 52.1MB

Validating runs/detect/train/weights/best.pt...

Ultralytics YOLOv8.2.103 Python-3.11.13 torch-2.6.0+cu124 CUDA:0 (NVIDIA A100-SXM4-40GB, 40507MiB)
Model summary (fused): 218 layers, 25,841,497 parameters, 0 gradients, 78.7 GFLOPs

Class	Images	Instances	Box(P	R	mAP50	mAP50-95)
all	1157	3060	0.842	0.711	0.759	0.53
dent	564	959	0.814	0.709	0.737	0.519
hole	165	241	0.855	0.707	0.753	0.526
rust	486	1860	0.858	0.715	0.787	0.546

Speed: 0.3ms preprocess, 3.8ms inference, 0.0ms loss, 1.1ms postprocess per image
Results saved to runs/detect/train
Learn more at <https://docs.ultralytics.com/modes/train>

Hình 5.4: Nhật ký huấn luyện chi tiết và kết quả đánh giá cuối cùng.

• Phân tích Xu hướng Hội tụ của Hàm Mất Mát

Quan sát các đường cong mất mát trong Hình 5.2, bao gồm train/box_loss (mất mát hộp giới hạn), train/cls_loss (mất mát phân loại), và train/dfl_loss (mất mát phân bố đối tượng). Tất cả các đường cong này đều cho thấy một xu hướng giảm dần rõ rệt và ổn định theo số epoch, minh chứng rằng mô hình đang học hiệu quả từ dữ liệu huấn luyện.

- Đặc biệt, mất mát phân loại (cls_loss) cho cả tập huấn luyện và xác thực giảm nhanh chóng ở các epoch đầu và dần ổn định ở mức thấp, cho thấy mô hình nhanh chóng học được cách phân biệt các lớp đối tượng (dent, hole, rust).
- Tương tự, mất mát hộp giới hạn (box_loss) và mất mát phân bố đối tượng (dfl_loss) cũng giảm đều, chỉ ra rằng mô hình đang liên tục cải thiện khả năng dự đoán vị trí và kích thước của các đối tượng một cách chính xác hơn.
- Một dấu hiệu tích cực quan trọng là khoảng cách tương đối nhỏ và ổn định giữa các đường cong mất mát của tập huấn luyện và tập xác thực. Điều này cho thấy mô hình không chỉ "học thuộc lòng" dữ liệu huấn luyện mà còn có khả năng tổng quát hóa tốt trên dữ liệu mới chưa từng thấy, một yếu tố then chốt cho thấy hiện tượng quá khớp đang được kiểm soát hiệu quả.

- **Đánh giá Hiệu suất và các Hiện tượng Kỹ thuật**

Hiệu suất mô hình: Các chỉ số hiệu suất định lượng như Precision, Recall, và mAP (Mean Average Precision) đều có xu hướng tăng mạnh ở các epoch đầu và dần hội tụ ở các giá trị cao, khẳng định sự hội tụ và hiệu quả của mô hình. Cụ thể:

- Precision (metrics/precision(B)) tăng từ gần 0 lên xấp xỉ 0.85, cho thấy độ chính xác cao của các dự đoán.
- Recall (metrics/recall(B)) tăng từ gần 0 lên xấp xỉ 0.72, cho thấy mô hình có khả năng phát hiện phần lớn các đối tượng thực tế.
- mAP@0.5 (mAP ở ngưỡng IoU 0.5) đạt giá trị cuối cùng là 0.759, một kết quả rất tốt cho bài toán nhận diện đa đối tượng.
- Đặc biệt, chỉ số F1-Score, là thước đo cân bằng giữa Precision và Recall, được sử dụng để tìm ra điểm hoạt động tối ưu. Biểu đồ F1-Confidence trong Hình 5.3 cho thấy hiệu suất tổng thể của mô hình đạt F1-Score cao nhất là 0.77 tại ngưỡng tin cậy (confidence threshold) là 0.361. Điều này cung cấp một chỉ dẫn quan trọng để cấu hình mô hình khi triển khai thực tế nhằm đạt được sự cân bằng tốt nhất.

Phân tích Hiện tượng Quá khớp (Overfitting): Quan sát nhật ký huấn luyện chi tiết ở các epoch cuối trong Hình 5.4, các giá trị mất mát và chỉ số hiệu suất trên tập xác thực (val) vẫn duy trì ổn định hoặc có sự cải thiện nhẹ. Cụ thể, val/mAP50 là 0.763 và val/mAP50-95 là 0.531 ở các epoch 99 và 100. Các giá trị này không có dấu hiệu suy giảm, khẳng định mô hình không bị quá khớp nghiêm trọng. Việc mô hình hoàn thành đủ 100 epoch mà không bị cơ chế dừng sớm (patience=10) kích hoạt cũng chứng tỏ hiệu suất trên tập xác thực vẫn được duy trì tốt cho đến cuối.

Phân tích Hiện tượng Dưới khớp (Underfitting): Không có dấu hiệu dưới khớp được quan sát trong quá trình huấn luyện. Hiện tượng này xảy ra khi mô hình không đủ phức tạp hoặc không được huấn luyện đủ để nắm bắt các quy luật của dữ liệu, biểu hiện qua việc cả đường cong mất mát của tập huấn luyện và xác thực đều duy trì ở

mức cao. Ngược lại, trong nghiên cứu này, các đường cong mất mát đạt mức thấp và các chỉ số mAP, F1-Score đạt mức cao, cho thấy mô hình đã học đủ các đặc trưng cần thiết.

- **Thảo luận về Vai trò của các Siêu tham số**

Sự thành công của giai đoạn huấn luyện này phần lớn đến từ việc lựa chọn và tinh chỉnh cẩn thận các siêu tham số, giúp định hướng và tối ưu hóa quá trình học của mô hình:

- Tốc độ học ($lr=0.002$) và Lịch trình Cosine Annealing ($cos_lr=True$): Việc sử dụng tốc độ học ban đầu vừa phải cùng với giai đoạn "warm-up" giúp mô hình bắt đầu quá trình học một cách ổn định. Lịch trình Cosine Annealing sau đó giúp tốc độ học giảm dần một cách thông minh, cho phép mô hình khám phá không gian tham số hiệu quả ở giai đoạn đầu và tinh chỉnh các trọng số một cách cẩn thận hơn khi gần đến điểm tối ưu, góp phần quan trọng vào sự hội tụ mịn màng và ổn định.
- Thuật toán tối ưu hóa ($optimizer=SGD$) với $momentum=0.937$: SGD với momentum cao giúp tăng tốc độ hội tụ, vượt qua các vùng phẳng của hàm mất mát và các điểm cực tiểu cục bộ hiệu quả hơn.
- Giảm trọng số ($weight_decay=0.0005$): Đây là một kỹ thuật điều chuẩn L2 quan trọng, giúp kiểm soát độ phức tạp của mô hình bằng cách phạt các trọng số có giá trị lớn, từ đó ngăn ngừa quá khớp và giúp mô hình tổng quát hóa tốt hơn trên dữ liệu mới.
- Các kỹ thuật tăng cường dữ liệu (Mosaic, Mixup, v.v.): Đây là yếu tố then chốt giúp mô hình chống lại hiện tượng quá khớp. Bằng cách tạo ra các biến thể dữ liệu đa dạng một cách nhân tạo, chúng buộc mô hình phải học các đặc trưng cốt lõi và bất biến của đối tượng, thay vì chỉ ghi nhớ các mẫu cụ thể trong tập huấn luyện.

5.2.2. Huấn luyện Bổ sung và Lựa chọn Mô hình cuối cùng

Sau khi có được mô hình best.pt từ 100 epoch đầu tiên, nhóm đã tiến hành huấn luyện tiếp mô hình này thêm 28 epoch để kiểm tra khả năng cải thiện thêm hiệu suất. Quá trình huấn luyện bổ sung này được ghi lại trong nhật ký ở **Hình 5.5**.

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
24/100	17.3G	0.4895	0.2607	0.9304	17	1024: 100% 760/760 [03:06<00:00, 4.09it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 37/37 [00:08<00:00, 4.37it/s]
	all	1157	3060	0.86	0.713	0.767	0.556
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
25/100	17.4G	0.4899	0.2614	0.9295	18	1024: 100% 760/760 [03:05<00:00, 4.09it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 37/37 [00:08<00:00, 4.36it/s]
	all	1157	3060	0.843	0.723	0.763	0.557
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
26/100	17.4G	0.4864	0.2596	0.9288	19	1024: 100% 760/760 [03:05<00:00, 4.09it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 37/37 [00:08<00:00, 4.33it/s]
	all	1157	3060	0.852	0.718	0.765	0.56
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
27/100	17.3G	0.4834	0.2587	0.9268	17	1024: 100% 760/760 [03:05<00:00, 4.10it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 37/37 [00:08<00:00, 4.37it/s]
	all	1157	3060	0.854	0.716	0.767	0.559
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
28/100	17.4G	0.4812	0.2564	0.9256	27	1024: 100% 760/760 [03:05<00:00, 4.09it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 37/37 [00:08<00:00, 4.34it/s]
	all	1157	3060	0.851	0.724	0.765	0.559

Hình 5.5: Kết quả sau khi train tiếp 28 epochs từ file best.pt.

Trong quá trình này, các chỉ số hiệu suất như Precision (P), Recall (R) và đặc biệt là mAP@50 có dấu hiệu "chững lại", không còn tăng trưởng rõ rệt từ khoảng epoch thứ 24 của giai đoạn này. Đây là dấu hiệu cho thấy mô hình đã đạt đến trạng thái bão hòa.

Nhận thấy nguy cơ bị quá khớp (overfitting) nếu tiếp tục huấn luyện, nhóm đã quyết định dừng quá trình này lại, và chọn file best.pt mới (dung lượng 152MB) làm mô hình cuối cùng để sử dụng cho các bước đánh giá và triển khai sau đó của dự án.

5.3. Đánh giá Mô hình Dự đoán

5.3.1. Các Chỉ số Đánh giá Hiệu suất (Evaluation Metrics)

Để lượng hóa và so sánh hiệu suất của mô hình phát hiện đối tượng, các chỉ số đánh giá chuyên biệt là không thể thiếu. Việc hiểu rõ ý nghĩa và cách tính toán của từng chỉ số giúp đưa ra kết luận chính xác về khả năng hoạt động của mô hình trong thực tế.

Trước khi đi vào các chỉ số chi tiết, chúng ta cần định nghĩa bốn thành phần cơ bản của ma trận đánh giá hiệu suất phân loại:

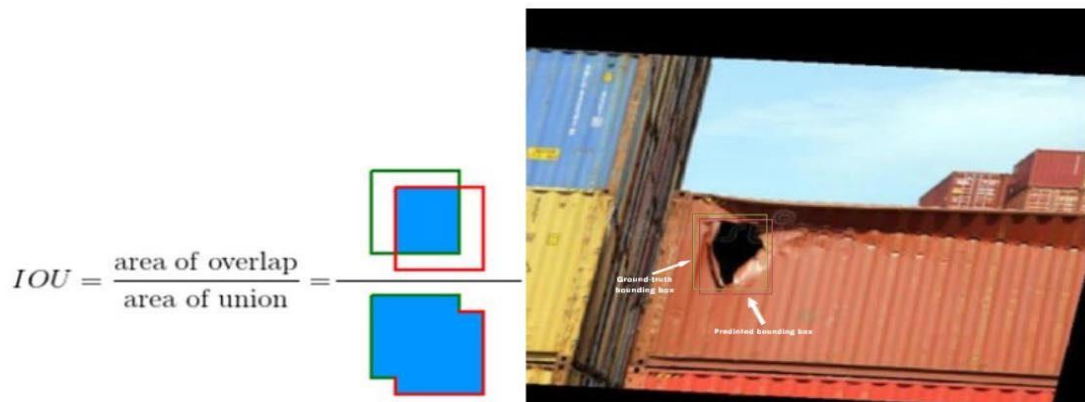
- True Positive (TP): Trường hợp dự đoán là Dương tính và kết quả thực tế là Dương tính. Ví dụ: mô hình dự đoán một vùng là 'rust' và thực tế đó đúng là một vết 'rust'.
- False Positive (FP): Trường hợp dự đoán là Dương tính nhưng kết quả thực tế là Âm tính (còn gọi là lỗi loại I hay "báo động giả"). Ví dụ: mô hình dự đoán một vùng là 'dent' nhưng thực tế đó chỉ là 'background'.
- False Negative (FN): Trường hợp dự đoán là Âm tính nhưng kết quả thực tế là Dương tính (còn gọi là lỗi loại II hay "bỏ sót"). Ví dụ: thực tế có một vết 'hole' nhưng mô hình không phát hiện ra.
- True Negative (TN): Trường hợp dự đoán là Âm tính và kết quả thực tế là Âm tính. Ví dụ: mô hình không phát hiện lỗi nào trên một vùng ảnh và thực tế vùng đó đúng là 'background'.

Bốn thành phần này là nền tảng để xây dựng các chỉ số đánh giá hiệu suất phức tạp hơn như Precision, Recall và F1-Score.

- Intersection over Union (IoU): Thước đo tỷ lệ chồng lấn giữa hai hộp giới hạn, tính bằng diện tích phân giao chia cho diện tích phân hợp.

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

IoU là chỉ số quan trọng để xác định mức độ chính xác của vị trí hộp giới hạn dự đoán và được sử dụng làm ngưỡng để phân loại True Positive (TP), False Positive (FP). Hình 5.6 bên dưới minh họa tỉ lệ của predicted boundingbox và ground-truth bounding box cùng với lỗi thùng lỗi.



Hình 5.6: Hình minh họa predicted bounding box với ground-truth bounding box. Trong đó:

- Area of overlap là diện tích phần giao giữa của hai khung
- Area of Union là diện tích phần hợp giữa hai khung.
- Với Ground-truth bounding box là do ta xác định trong lúc gắn nhãn, Predicted bounding box do mô hình xác định.

Ngưỡng của IOU thường có giá trị nhất định (0 đến 1). Một khung dự đoán được coi là đúng (true positive) nếu IoU của nó với khung thực tế lớn hơn hoặc bằng ngưỡng đã định (thường được đặt bằng 0,5).

Các tiêu chí đánh giá với IOU threshold:

- True positive (TP): Đối tượng được nhận dạng đúng có $IOU \geq \text{Ngưỡng}$.
- False positive (FP): Đối tượng được nhận dạng sai có $IOU < \text{Ngưỡng}$.
- False negative (FN): Đối tượng không nhận dạng được.
 - Precision (Độ chính xác): Tỷ lệ các dự đoán tích cực mà mô hình đưa ra là đúng. Nói cách khác, nó trả lời câu hỏi: "Trong số những gì mô hình nói là có lỗi, bao nhiêu phần trăm thực sự là lỗi?"

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

Precision cao cho thấy mô hình ít đưa ra các dự đoán sai (ít FP).

- Recall (Độ nhạy/Độ phủ): Tỷ lệ các đối tượng thực tế được mô hình phát hiện đúng. Nói cách khác, nó trả lời câu hỏi: "Trong số tất cả các lỗi thực tế, bao nhiêu phần trăm mô hình đã tìm thấy?"

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

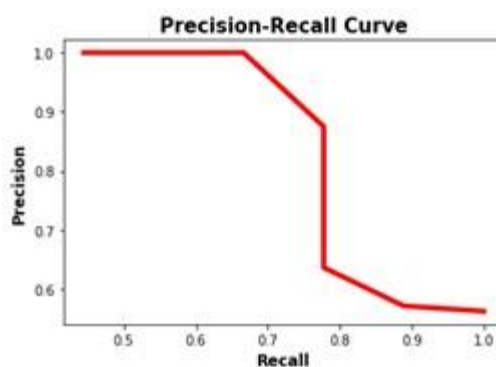
Recall cao cho thấy mô hình ít bỏ sót các đối tượng thực tế (ít FN).

- F1-Score: Trung bình điều hòa của Precision và Recall. Nó cung cấp một cái nhìn cân bằng về hiệu suất của mô hình khi có sự đánh đổi giữa Precision và Recall.

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

- Average Precision (AP) và Mean Average Precision (mAP):

Precision-Recall Curve (Đường cong Precision-Recall): Là một biểu đồ cho thấy mối quan hệ giữa Precision và Recall ở các ngưỡng điểm tự tin khác nhau từ 0 đến 1. Để quan sát tất cả các precision và recall tương ứng các threshold ta sử dụng Precision Recall Curve – đường đi qua tất các điểm với giá trị (recall, precision) ứng với từng ngưỡng. Khi ngưỡng tự tin giảm, recall thường tăng (phát hiện nhiều lỗi hơn) nhưng precision có thể giảm (tăng FP). Hình 5.7 dưới đây thể hiện mối quan hệ giữa Precision và Recall.



Hình 5.7: Precision-Recall Curve

- Average Precision (AP): AP là diện tích dưới đường cong Precision-Recall cho một lớp đối tượng cụ thể. Nó cung cấp một chỉ số tổng hợp về hiệu suất của mô hình cho lớp đó [14].

Được xác định bởi công thức:

$$AP = \sum_{k=1}^n (R_k - R_{k-1}) * P_k$$

Trong đó:

- R_k, P_k : lần lượt là Recall và Precision ứng với ngưỡng xác suất thứ k
- n : số lượng ngưỡng xác suất
- AP lớn \Rightarrow vùng AUC này lớn \Rightarrow đường cong có xu hướng ở góc trên bên phải \Rightarrow Ở các threshold khác nhau có Precision và Recall đều ở mức cao \Rightarrow mô hình có hiệu suất tốt.
- AP nhỏ \Rightarrow Precision và Recall đều khá thấp \Rightarrow mô hình có hiệu suất không tốt.
 - Mean Average Precision (mAP): mAP là giá trị trung bình của AP trên tất cả các lớp đối tượng trong tập dữ liệu [14]. Đây là chỉ số quan trọng và phổ biến nhất để đánh giá toàn diện hiệu suất của mô hình phát hiện đối tượng.

$$mAP = \frac{1}{n} \sum_{i \in C} AP(i)$$

Trong đó:

- C là tập hợp tất cả các class - n là số class

mAP càng lớn \Rightarrow AP của mỗi lớp càng lớn \Rightarrow mô hình có hiệu suất càng tốt. Vậy nên ta sẽ cố gắng train model để cho giá trị mAP là lớn nhất. Vì thế nên ta sẽ sử dụng mAP để đánh giá model.

- mAP@0.5: Đây là giá trị mAP được tính khi ngưỡng IoU cố định là 0.5 [18]. Tức là, một dự đoán được coi là đúng dương nếu IoU của nó với GT lớn hơn hoặc bằng 0.5.

- **mAP@0.5:0.95**: Đây là chỉ số mAP được sử dụng trong các cuộc thi như COCO, được tính bằng cách lấy giá trị trung bình của AP trên một loạt các ngưỡng IoU khác nhau, từ 0.5 đến 0.95 với bước nhảy 0.05 [18].
- **Frames Per Second (FPS)**: Đo lường tốc độ suy luận (inference speed) của mô hình, tức là số lượng khung hình mà mô hình có thể xử lý trong một giây. Đây là chỉ số cực kỳ quan trọng cho các ứng dụng thời gian thực (real-time applications) như hệ thống kiểm tra trực tuyến trên dây chuyền sản xuất hoặc triển khai trên các thiết bị nhúng như Raspberry Pi [2].

5.3.2. Hiệu suất Phát hiện đối tượng trên Tập Xác thực và Kiểm tra

Bảng 5.1 và Bảng 5.2 tóm tắt các chỉ số hiệu suất chính của mô hình YOLOv8m.pt đã huấn luyện trên tập xác thực (Validation Set).

Bảng 5.1: Tổng hợp các chỉ số mAP trên tập xác thực.

Chỉ số	Giá trị
mAP@0.5	0.759
mAP@0.5:0.95	0.530

Bảng 5.2: Chi tiết Precision, Recall, mAP cho từng lớp trên tập xác thực

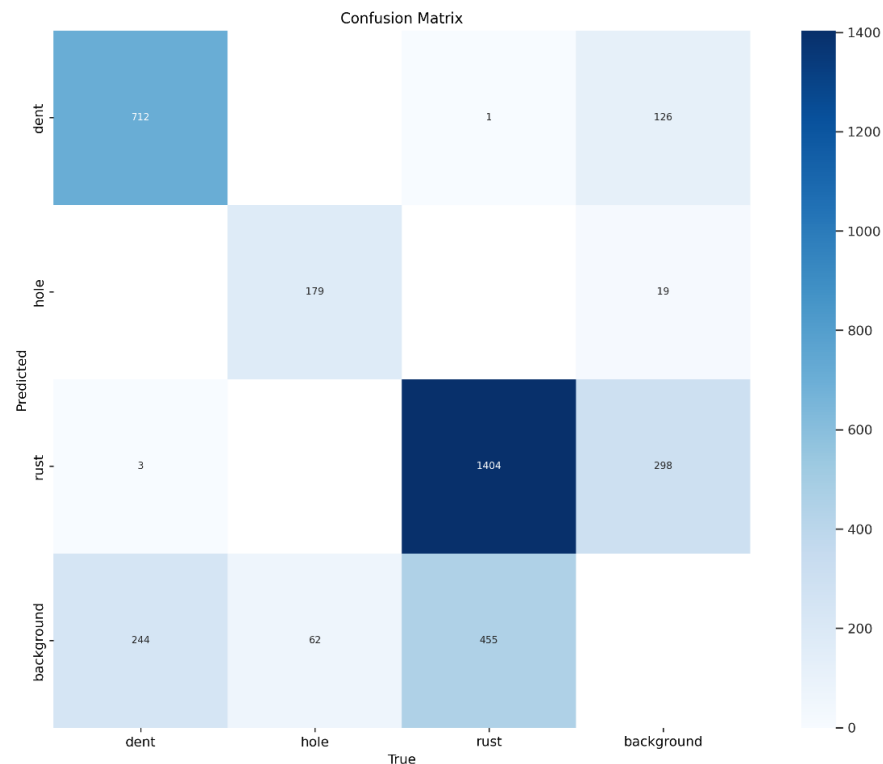
Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	1157	3060	0.842	0.711	0.759	0.530
dent	564	959	0.814	0.709	0.737	0.519
hole	165	241	0.855	0.707	0.753	0.526
rust	486	1860	0.858	0.715	0.787	0.546

Thông tin về số lượng tham số (Parameters) và kích thước mô hình (Model Size):

- **Model Summary (fused)**: 218 layers, 25,841,497 parameters, gradients, 78.7 GFLOPS.
- **Kích thước mô hình (best.pt)**: 152MB

5.3.3. Ma trận Nhầm lẫn (Confusion Matrix)

Ma trận nhầm lẫn là một công cụ đánh giá quan trọng, cung cấp cái nhìn chi tiết về hiệu suất phân loại của mô hình cho từng lớp đối tượng. Nó thể hiện số lượng các dự đoán đúng (True Positives, True Negatives) và sai (False Positives, False Negatives) của mô hình trên tập dữ liệu xác thực (validation set).



Hình 5.8: Ma trận nhầm lẫn của mô hình trên tập xác thực.

Phân tích dưới đây sẽ đi sâu vào các loại lỗi mà mô hình mắc phải thông qua ma trận nhầm lẫn (Hình 5.8), đồng thời sử dụng các chỉ số chính thức từ Bảng 5.2 để đánh giá hiệu suất tổng thể. Sự khác biệt về số liệu giữa hai nguồn này là do chúng được tính ở các ngưỡng tin cậy (confidence threshold) khác nhau, phản ánh sự đánh đổi giữa độ chính xác và độ nhạy của mô hình.

- Lớp 'dent' (Vết móp)
- Phân tích lỗi chi tiết (từ Ma trận nhầm lẫn):
 - True Positives (TP): 712
 - False Positives (FP): 3 (nhầm rust) + 244 (nhầm background) = 247

- False Negatives (FN): 1 (nhầm hole) + 126 (nhầm background) = 127
- Hiệu suất tính toán tại ngưỡng của ma trận nhầm lẫn:
 - Độ chính xác (Precision) = $712/(712+247) \approx 0.742$ (74.2%)
 - Độ nhạy (Recall) = $712/(712+127) \approx 0.849$ (84.9%)
- Hiệu suất chính thức (từ Bảng 5.2 tại ngưỡng tối ưu):
 - Precision = 0.814 (81.4%)
 - Recall = 0.709 (70.9%)

Đánh giá: Ma trận nhầm lẫn cho thấy ở một ngưỡng tin cậy thấp, mô hình có thể đạt Recall rất cao (84.9%) nhưng Precision chỉ ở mức khá (74.2%). Tuy nhiên, ở ngưỡng hoạt động tối ưu, mô hình đã hy sinh một phần Recall để đạt được Precision cao hơn hẳn (81.4%), cho thấy sự đánh đổi để có kết quả đáng tin cậy hơn.

- Lớp 'hole' (Lỗ thùng)
- Phân tích lỗi chi tiết (từ Ma trận nhầm lẫn):
 - True Positives (TP): 179
 - False Positives (FP): 1 (nhầm dent) + 62 (nhầm background) = 63
 - False Negatives (FN): 19 (nhầm rust)
- Hiệu suất tính toán tại ngưỡng của ma trận nhầm lẫn:
 - Độ chính xác (Precision) = $179/(179+63) \approx 0.740$ (74.0%)
 - Độ nhạy (Recall) = $179/(179+19) \approx 0.904$ (90.4%)
- Hiệu suất chính thức (từ Bảng 5.2 tại ngưỡng tối ưu):
 - Precision = 0.855 (85.5%)
 - Recall = 0.707 (70.7%)

Đánh giá: Tương tự, mô hình có thể đạt Recall tới 90.4% nhưng phải chấp nhận Precision thấp. Ở chế độ tối ưu, Precision được đẩy lên rất cao (85.5%), làm cho việc dự đoán lớp 'hole' trở nên rất đáng tin cậy.

- Lớp 'rust' (Gỉ sét)
- Phân tích lỗi chi tiết (từ Ma trận nhầm lẫn):
 - True Positives (TP): 1404
 - False Positives (FP): 19 (nhầm hole) + 455 (nhầm background) = 474

- False Negatives (FN): 3 (nhầm dent) + 298 (nhầm background) = 301
- Hiệu suất tính toán tại ngưỡng của ma trận nhầm lẫn:
 - Độ chính xác (Precision) = $1404/(1404+474) \approx 0.748$ (74.8%)
 - Độ nhạy (Recall) = $1404/(1404+301) \approx 0.823$ (82.3%)
- Hiệu suất chính thức (từ Bảng 5.2 tại ngưỡng tối ưu):
 - Precision = 0.858 (85.8%)
 - Recall = 0.715 (71.5%)

Đánh giá: Vấn đề lớn nhất của mô hình là phân biệt rust với background. Ma trận nhầm lẫn chỉ ra rõ điều này với số lượng FP và FN rất lớn liên quan đến background. Để đạt được Precision cao (85.8%) trong kết quả cuối cùng, mô hình đã phải giảm Recall xuống còn 71.5%, chấp nhận bỏ sót một số trường hợp để các dự đoán đưa ra là chắc chắn nhất.

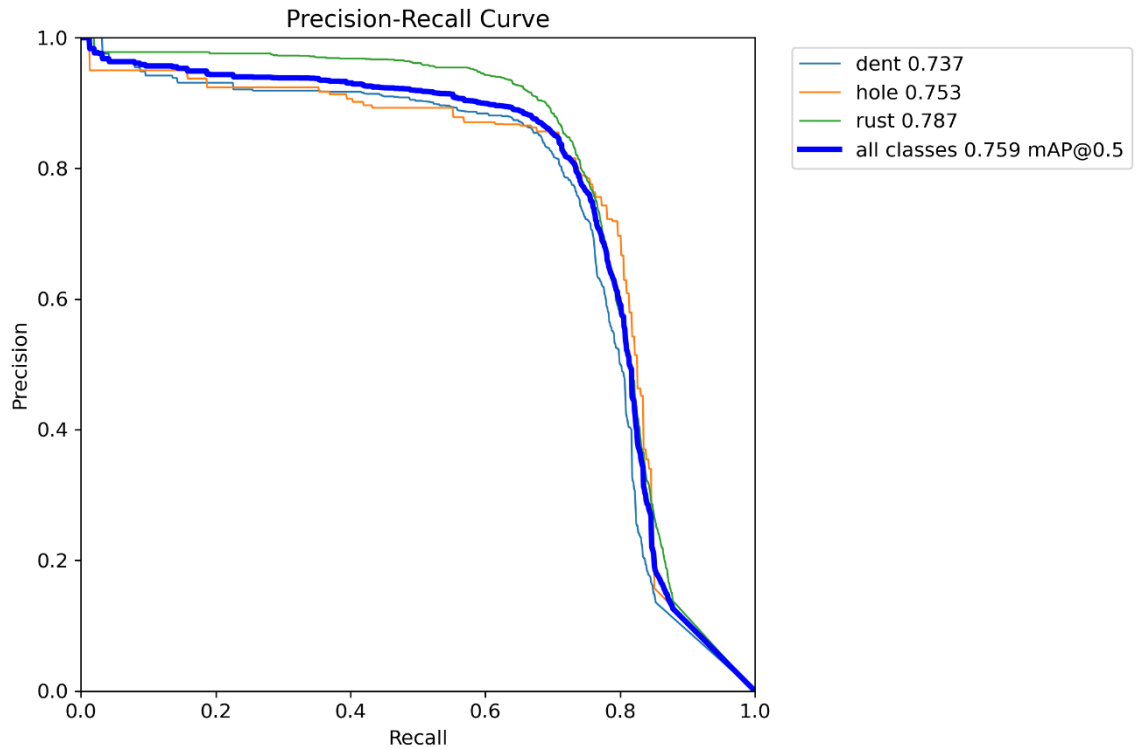
Từ việc kết hợp cả hai bảng phân tích, có thể rút ra kết luận toàn diện:

Mô hình đạt được độ chính xác (Precision) cao cho tất cả các lớp lỗi (đều trên 81%), đặc biệt là hole và rust (gần 86%). Điều này có nghĩa là các cảnh báo lỗi mà mô hình đưa ra có độ tin cậy cao. Tuy nhiên, điểm yếu cốt lõi nằm ở độ nhạy (Recall), chỉ đạt khoảng 71%. Nguyên nhân chính được chỉ ra rõ trong ma trận nhầm lẫn: mô hình gặp khó khăn trong việc phân biệt ranh giới giữa các vùng lỗi (đặc biệt là dent và rust) với các đặc điểm của nền (background). Điều này dẫn đến việc mô hình bỏ sót một phần đáng kể các lỗi thực tế. Để cải thiện, hướng đi quan trọng nhất là tăng cường và đa dạng hóa dữ liệu huấn luyện, tập trung vào các hình ảnh có nền phức tạp, nhiễu, hoặc có đặc điểm dễ gây nhầm lẫn với các loại lỗi, nhằm nâng cao khả năng phân biệt của mô hình.

5.3.4. Đường cong Precision-Recall (PR Curve)

Đường cong Precision-Recall (PR Curve) là một biểu đồ quan trọng khác để đánh giá hiệu suất của mô hình phát hiện đối tượng, đặc biệt hữu ích khi tập dữ liệu có sự mất cân bằng lớp hoặc khi quan tâm đến hiệu suất của các lớp thiểu số. Đường cong này minh họa mối quan hệ giữa Precision và Recall khi ngưỡng tin cậy (confidence threshold) thay đổi. Diện tích dưới đường cong PR (Area

Under Curve - AUC) chính là Average Precision (AP), một chỉ số tổng hợp về hiệu suất của mô hình cho một lớp cụ thể.



Hình 5.9: Đường cong Precision-Recall cho từng lớp và tổng thể.

Quan sát Hình 5.9

- Đường cong màu xanh đậm (all classes) đại diện cho hiệu suất tổng thể, với giá trị mAP@0.5 là 0.759. Giá trị này thể hiện một hiệu suất phát hiện đối tượng khá tốt, đặc biệt khi yêu cầu độ chính xác vị trí tương đối ($\text{IoU} > 0.5$).
- Lớp 'rust' (đường màu xanh lá cây) thể hiện AP cao nhất là 0.787. Đường cong này duy trì Precision cao (trên 0.9) trong một phạm vi Recall rộng (đến khoảng 0.7), sau đó giảm dần khi Recall tiến gần 1.0. Điều này cho thấy mô hình rất tự tin và chính xác khi phát hiện lỗi gỉ sét, phù hợp với việc lớp rust có số lượng thể hiện lớn nhất trong tập dữ liệu, cung cấp đủ mẫu đa dạng cho mô hình học.

- Lớp 'hole' (đường màu cam) có AP là 0.753. Mặc dù đây là lớp thiểu số (chỉ 241 instances trong tập xác thực), mô hình vẫn đạt được hiệu suất tốt. Đường cong của 'hole' cũng duy trì Precision khá cao ở các mức Recall trung bình, cho thấy khả năng phân biệt tốt các lỗ thủng, có thể do đặc trưng hình học của chúng thường rõ ràng và dễ phân biệt hơn.
- Lớp 'dent' (đường màu xanh dương) có AP là 0.737. Đây là lớp có AP thấp nhất trong ba lớp, cho thấy đây là thách thức lớn nhất đối với mô hình. Đường cong của dent giảm Precision nhanh hơn khi Recall tăng lên (đặc biệt rõ rệt sau khi Recall vượt quá 0.7), ngụ ý mô hình kém tự tin hơn và dễ mắc lỗi hơn khi cố gắng tìm tất cả các vết lõm. Điều này có thể do tính chất đa dạng và đôi khi không rõ ràng của các vết lõm, dễ bị nhầm lẫn với các đặc điểm bề mặt hoặc nhiễu.

Phân tích đường cong PR giúp hiểu rõ sự đánh đổi giữa Precision và Recall. Trong ứng dụng thực tế, sự lựa chọn ngưỡng tin cậy sẽ phụ thuộc vào yêu cầu cụ thể của bài toán: nếu ưu tiên không bỏ sót lỗi (Recall cao, ví dụ trong kiểm tra an toàn nghiêm ngặt), người dùng có thể chấp nhận Precision thấp hơn một chút (tức là có thể có thêm báo động giả). Ngược lại, nếu ưu tiên giảm báo động giả (Precision cao, tránh lãng phí thời gian kiểm tra), Recall có thể bị ảnh hưởng (mô hình sẽ bỏ sót một số lỗi).

5.3.5. Ảnh hưởng của Ngưỡng Tin cậy (Confidence Threshold) và IoU

Threshold trong Post-training

Trong quá trình đánh giá hiệu suất mô hình sau huấn luyện (post-training validation), việc lựa chọn các ngưỡng conf (confidence threshold) và iou (Intersection over Union threshold) có vai trò quan trọng trong việc tinh chỉnh kết quả và phù hợp với yêu cầu thực tế của ứng dụng. IoU threshold (iou=0.6) được cố định để xác định khi nào một dự đoán được coi là đúng vị trí so với ground truth. Trong khi đó, conf threshold ảnh hưởng trực tiếp đến số lượng dự đoán được chấp nhận và do đó ảnh hưởng đến Precision và Recall.

Dưới đây là bảng tổng hợp các chỉ số hiệu suất với các ngưỡng conf khác nhau, giữ iou=0.6:

Bảng 5.3: Hiệu suất mô hình YOLOv8m.pt với các số conf khác nhau (IoU=0.6)

Confidence (conf)	Precision (Box(P))	Recall (R)	mAP50	mAP50-95
0.25	0.852	0.723	0.795	0.621
0.3	0.852	0.723	0.793	0.621
0.5	0.875	0.694	0.782	0.592
0.7	0.906	0.619	0.754	0.611
0.9	0.941	0.187	0.561	0.500

Phân tích ảnh hưởng của ngưỡng tin cậy:

- Ngưỡng conf=0.25 và iou=0.6:
 - Đạt được mAP50 = 0.795 và mAP50-95 = 0.621. Precision tổng thể là 0.852 và Recall là 0.723. Đây là cấu hình cho ra kết quả mAP tổng thể tốt nhất trong các thử nghiệm với ngưỡng tin cậy này. Ngưỡng tin cậy thấp hơn này cho phép mô hình chấp nhận nhiều dự đoán hơn, dẫn đến Recall cao hơn, nhưng vẫn duy trì Precision ở mức chấp nhận được, làm cho mAP tổng thể cao.
- Ngưỡng conf=0.3 và iou=0.6:
 - Tương tự conf=0.25, đạt mAP50 = 0.793 và mAP50-95 = 0.621. Precision và Recall vẫn cao, cho thấy hiệu suất ổn định trong khoảng ngưỡng này.
- Ngưỡng conf=0.5 và iou=0.6:

- Khi tăng conf lên 0.5, Precision tăng lên 0.875 (giảm FP), nhưng Recall giảm xuống 0.694. Điều này phù hợp với đặc tính của ngưỡng tin cậy: tăng ngưỡng sẽ chấp nhận ít dự đoán hơn, làm tăng độ chính xác của các dự đoán được chấp nhận nhưng có thể bỏ sót nhiều đối tượng hơn. mAP50 và mAP50-95 cũng giảm nhẹ so với ngưỡng 0.25.
- Ngưỡng conf=0.7 và iou=0.6:
 - Precision tiếp tục tăng lên đáng kể (0.906), cho thấy các dự đoán được chấp nhận là rất tin cậy và ít sai sót. Tuy nhiên, Recall giảm mạnh xuống 0.619, tức là mô hình bỏ sót nhiều lỗi thực tế hơn. mAP tổng thể cũng giảm xuống 0.754 và 0.611.
- Ngưỡng conf=0.9 và iou=0.6:
 - Với ngưỡng rất cao này, Precision đạt mức cao nhất (0.941), nhưng Recall giảm rất sâu (0.187). Điều này có nghĩa là mô hình chỉ báo cáo những gì nó cực kỳ chắc chắn, nhưng bỏ sót gần như tất cả các lỗi khác. Kết quả là mAP giảm rất mạnh (0.561 và 0.500), cho thấy hiệu suất tổng thể không còn tốt.

Kết luận về ngưỡng tin cậy:

Dựa trên phân tích này, việc lựa chọn conf=0.25 và iou=0.6 cho quá trình post-training (như đã chọn trong cấu hình !yolo val) là một quyết định hợp lý vì nó mang lại mAP tổng thể cao nhất, thể hiện sự cân bằng tốt giữa Precision và Recall cho bài toán phát hiện lỗi container. Ngưỡng này cho phép mô hình phát hiện được phần lớn các lỗi mà vẫn duy trì độ chính xác chấp nhận được cho các dự đoán của nó. Việc hiểu rõ ảnh hưởng của ngưỡng tin cậy giúp tinh chỉnh hệ thống để phù hợp với yêu cầu cụ thể của ứng dụng (ví dụ: nếu an toàn là tối thượng thì cần Recall cao, nếu tránh báo động giả là tối thượng thì cần Precision cao).

5.3.6. So sánh hiệu suất mô hình với mô hình huấn luyện trên nền tảng Roboflow

Để đánh giá một cách toàn diện hiệu suất của mô hình YOLOv8m.pt đã được huấn luyện tùy chỉnh trên Google Colab, nhóm nghiên cứu đã tiến hành so sánh kết quả với một mô hình tương tự được huấn luyện trực tiếp trên nền tảng **Roboflow** sử dụng cùng tập dữ liệu. Việc so sánh này nhằm làm nổi bật đóng góp của quá trình tinh chỉnh siêu tham số và cấu hình huấn luyện thủ công.

- **Thông tin và Kết quả của Mô hình Roboflow:**

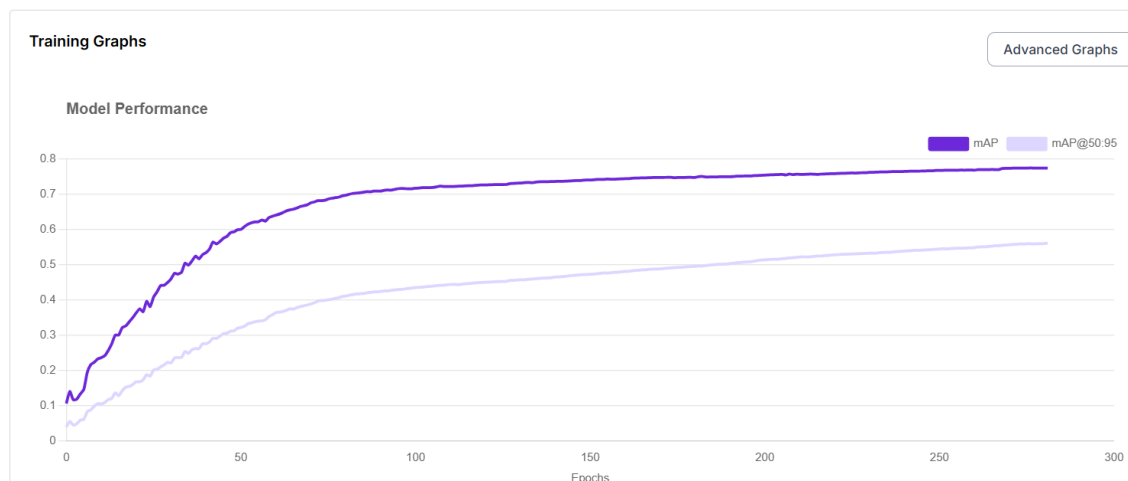
Thông tin về mô hình Roboflow:

- **Model URL:** container-damaged-detection-last/1
- **Checkpoint:** COCOs (cho thấy đây là mô hình được huấn luyện dựa trên trọng số pre-trained của COCO).
- **Dataset Version:** 2025-06-06 8:28pm
- **Model Type:** Roboflow 3.0 Object Detection (Accurate)

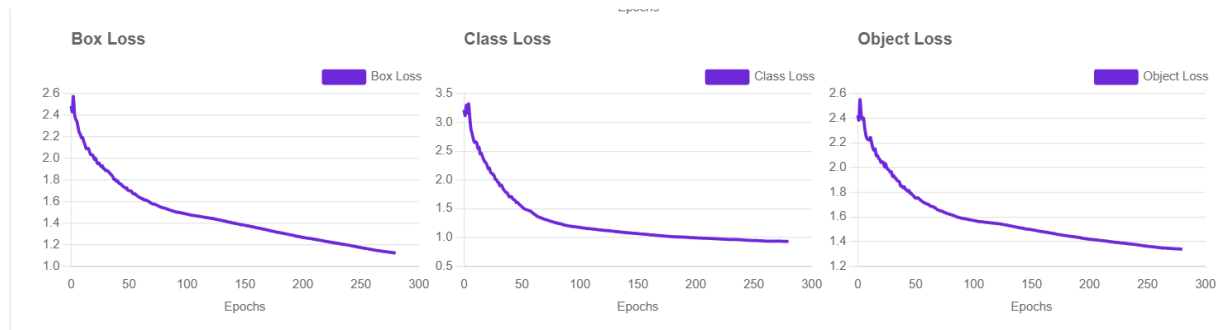
Kết quả đánh giá cuối cùng của mô hình Roboflow là:

- **mAP@50:** 77.5 %
- **Precision:** 84.0 %
- **Recall:** 73.2 %
- **Biểu đồ huấn luyện của mô hình Roboflow:**

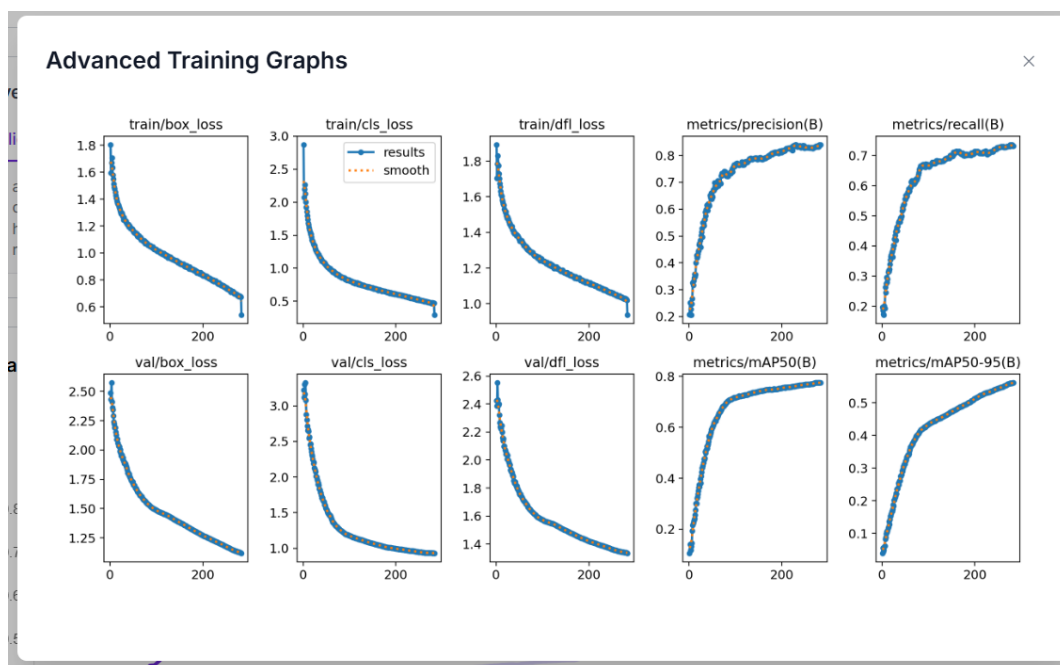
Các biểu đồ dưới đây minh họa quá trình huấn luyện và hiệu suất của mô hình được huấn luyện trên Roboflow.



Hình 5.10: Biểu đồ hiệu suất huấn luyện của mô hình Roboflow.



Hình 5.11: Biểu đồ đường cong mất mát của mô hình Roboflow



Hình 5.12: Biểu đồ hiệu suất huấn luyện nâng cao của mô hình Roboflow

Bảng 5.4: So sánh giữa Mô hình Tự Huấn luyện và Mô hình Roboflow

Chỉ số	Mô hình Tự Huấn luyện (Colab, YOLOv8m.pt)	Mô hình Roboflow	Nhận xét
mAP@0.5	0.759	0.775	Mô hình Roboflow có mAP@0.5 cao hơn một chút (77.5% so với 75.9%). Điều này cho thấy cấu hình mặc định hoặc tự động tối ưu của Roboflow đã rất hiệu quả.
Precision	0.842	0.840	Precision của hai mô hình tương đương, cho thấy cả hai đều có khả năng đưa ra dự đoán đúng với tỷ lệ False Positive tương tự.
Recall	0.711	0.732	Mô hình Roboflow có

			Recall cao hơn một chút (73.2% so với 71.1%). Điều này có thể do Roboflow tìm thấy nhiều lỗi thực tế hơn.
mAP@0.5:0.95	0.530	(Không hiển thị rõ)	Cần thông tin mAP@0.5:0.95 từ Roboflow để so sánh đầy đủ về độ chính xác định vị.
Đường cong mất mát (Box, Class, Object Loss)	Giảm ổn định, hội tụ tốt (Hình 5.4)	Giảm ổn định, hội tụ tốt (Hình 5.11)	Cả hai mô hình đều cho thấy quá trình học tập hiệu quả, mất mát giảm dần.
Đường cong mAP theo Epoch	Tăng ổn định và hội tụ (Hình 5.4)	Tăng ổn định và hội tụ (Hình 5.12)	Cả hai mô hình đều học tốt và hiệu suất cải thiện theo thời gian.

Phân tích và Thảo luận:

Kết quả so sánh cho thấy mô hình được huấn luyện trực tiếp trên nền tảng Roboflow đạt hiệu suất tổng thể (mAP@0.5 và Recall) tốt hơn một chút so với mô hình được huấn luyện tùy chỉnh trên Google Colab với cấu hình đã nêu. Điều này có thể được giải thích bởi một số yếu tố:

- **Tối ưu hóa nội bộ của Roboflow:** Nền tảng Roboflow có thể áp dụng các chiến lược huấn luyện, siêu tham số, hoặc thậm chí kiến trúc mô hình (ví dụ: các biến thể YOLO riêng, hoặc các kỹ thuật điều chỉnh/tăng cường dữ liệu nâng cao) được tối ưu hóa đặc biệt cho dữ liệu và loại mô hình cụ thể mà họ hỗ trợ. Các tối ưu hóa này có thể vượt trội hơn so với cấu hình siêu tham số thủ công đã được sử dụng trong nghiên cứu này.
- **Kỹ thuật tăng cường dữ liệu:** Mặc dù nghiên cứu đã áp dụng nhiều kỹ thuật tăng cường dữ liệu mạnh mẽ, Roboflow có thể sử dụng một pipeline tăng cường dữ liệu tự động hoặc các kỹ thuật mà Roboflow gọi là "Augmentations" phức tạp hơn hoặc khác biệt hơn, phù hợp hơn với đặc điểm của tập dữ liệu container.
- **Quy trình huấn luyện:** Roboflow có thể sử dụng các thuật toán tối ưu hóa hoặc lịch trình tốc độ học tinh vi hơn, hoặc có thời gian huấn luyện lâu hơn (Hình 5.11 cho thấy mô hình Roboflow huấn luyện đến gần 300 epoch, trong khi mô hình tự huấn luyện chỉ 100 epoch), điều này cho phép mô hình học sâu hơn từ dữ liệu.

Mặc dù mô hình Roboflow có mAP@0.5 cao hơn, nhưng mô hình tự huấn luyện trên Google Colab vẫn đạt được hiệu suất rất cạnh tranh và đáng tin cậy. Điều quan trọng là nghiên cứu này đã chứng minh được khả năng triển khai mô hình học sâu một cách độc lập và kiểm soát toàn bộ quá trình, từ chuẩn bị dữ liệu đến huấn luyện và đánh giá. Việc hiểu rõ các siêu tham số và tác động của chúng là một đóng góp quan trọng. Kết quả này cũng mở ra hướng nghiên cứu sâu hơn

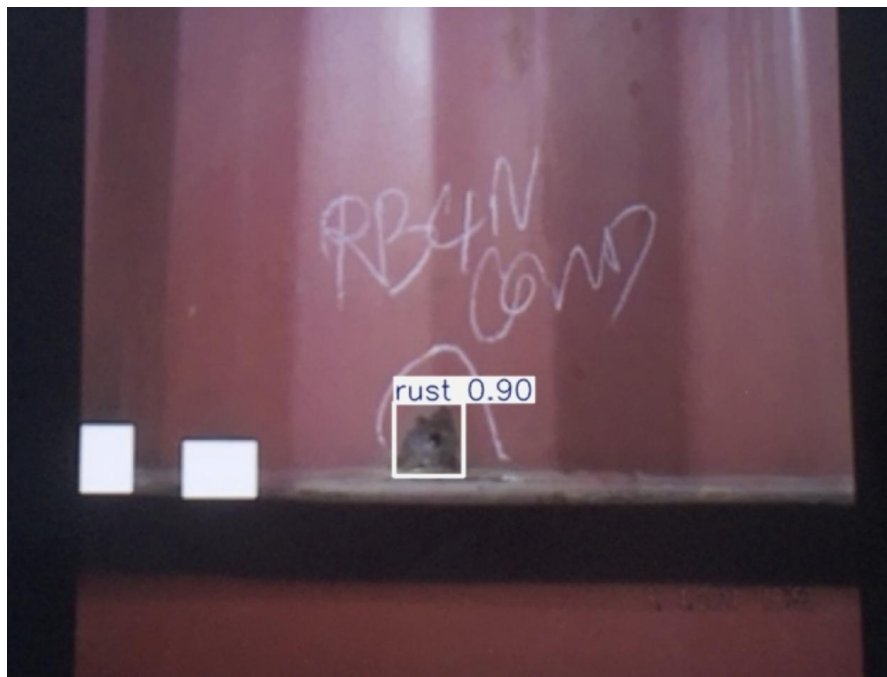
về việc tối ưu hóa siêu tham số bằng các kỹ thuật tự động (ví dụ: Auto-ML, Bayesian Optimization) để vượt qua hiệu suất của nền tảng sẵn có trong tương lai.

5.4. Phân tích Định tính (Qualitative Analysis)

Phân tích định tính được thực hiện bằng cách kiểm tra trực quan các hình ảnh từ tập kiểm tra với các hộp giới hạn dự đoán của mô hình. Điều này giúp hiểu sâu hơn về khả năng và hạn chế của mô hình trong các tình huống thực tế, bổ sung cho các đánh giá định lượng.

Các trường hợp phát hiện thành công:

Mô hình cho thấy khả năng phát hiện tốt các lỗi có đặc trưng rõ ràng, kích thước đủ lớn và độ tương phản cao so với nền. Ví dụ, các vết rỉ sét lớn, đậm màu, hoặc các lỗ thủng có đường biên sắc nét thường được phát hiện với điểm tự tin cao và hộp giới hạn chính xác.

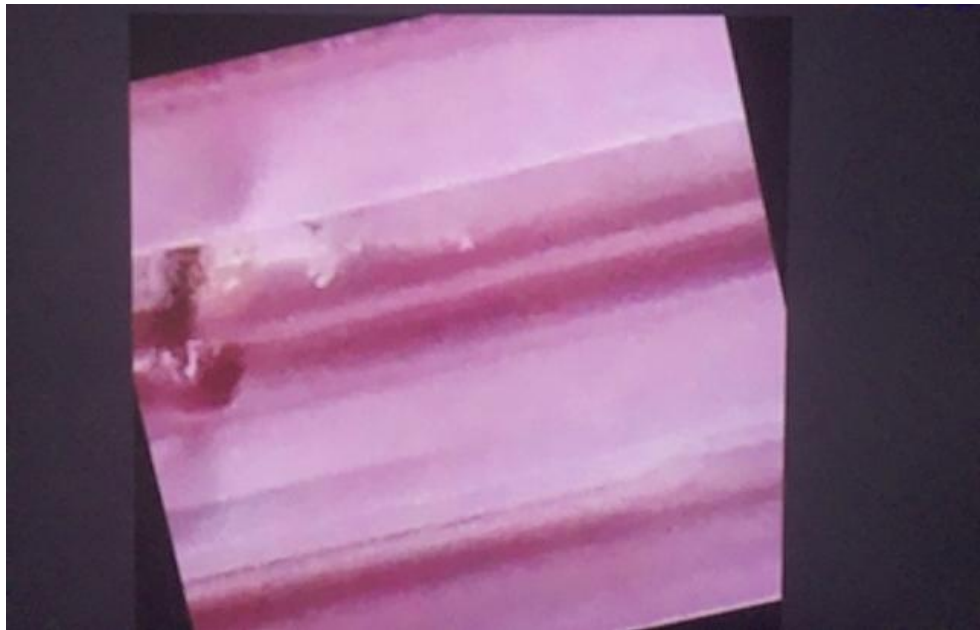


Hình 5.13: Ví dụ về trường hợp phát hiện lỗi thành công (Vết rỉ sét rõ ràng).

Các trường hợp mô hình gặp khó khăn/thất bại:

Mặc dù có hiệu suất tốt, mô hình vẫn gặp khó khăn trong một số trường hợp, dẫn đến các loại lỗi sau:

- **Sai dương (FP):** Mô hình đôi khi có thể nhầm lẫn các đặc điểm tự nhiên của bề mặt kim loại (như vết bẩn, vân kim loại, hoặc phản xạ ánh sáng) với các lỗi hư hỏng.
- **Sai âm (FN):** Các lỗi bị bỏ sót thường là các lỗi rất nhỏ, mờ, hoặc có độ tương phản cực thấp, khiến chúng khó nhận diện ngay cả bằng mắt thường.
- **Phân loại sai:** Trong một số trường hợp, mô hình có thể phát hiện đúng vị trí của lỗi nhưng gán nhãn sai loại (ví dụ: một dent lớn có thể bị nhầm thành rust).



Hình 5.14: Ví dụ về trường hợp phát hiện lỗi thất bại (Lỗi nhỏ bị bỏ sót).

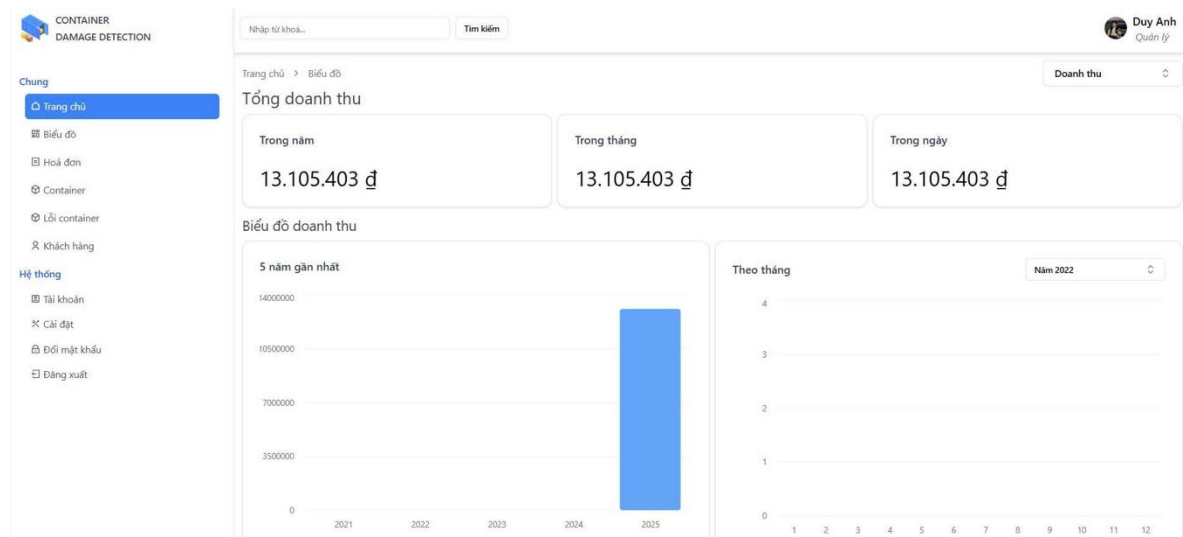
Nguyên nhân chính dẫn đến các sai sót này đến từ sự kết hợp của nhiều yếu tố. Chủ yếu là do đặc tính vật lý của một số lỗi quá khó để nhận diện (ví dụ: quá nhỏ, mờ, độ tương phản thấp) và do điều kiện ánh sáng phức tạp tại hiện trường (chói,

bóng tối) đã che khuất các đặc trưng quan trọng. Bên cạnh đó, dù đã được tăng cường, tập dữ liệu vẫn có thể chưa bao quát hết mọi biến thể lỗi hiếm gặp, và sự thiếu nhất quán trong quá trình gán nhãn ban đầu cũng có thể ảnh hưởng đến khả năng học của mô hình.

5.5. Triển khai trên website

5.5.1. Trang quản lý container

Khi truy cập hệ thống, người dùng sẽ được đưa đến trang Dashboard chính, cung cấp cái nhìn tổng quan về trạng thái hoạt động của hệ thống và các thống kê cơ bản.



Hình 5.15: Giao diện Dashboard .

Giao diện bao gồm 2 phần: Tổng doanh thu và Biểu đồ doanh thu.

- Tổng doanh thu: thể hiện rõ doanh thu 3 hạng mục: trong ngày, trong tháng, trong năm
- Biểu đồ doanh thu: có thể theo dõi biểu đồ doanh thu trong 5 năm gần nhất kèm một biểu đồ theo dõi chi tiết doanh thu của từng năm

5.5.2. Quy trình thêm và chụp ảnh container

Để bắt đầu quy trình kiểm tra, người dùng sẽ nhấn vào nút thêm mới ở tab container sẽ có một giao diện cho phép người dùng thêm số hiệu container bằng cách thủ công hoặc quét tự động qua camera.

Thêm container mới

Mã container

Nhập mã container... **Quét bảng hiệu**

Mô tả

Nhập mô tả...

Vai trò

Chọn khách hàng...

Đóng **Lưu**

Hình 5.16: Giao diện thêm mới một container

Trước khi bắt đầu chụp ảnh container người dùng sẽ phải thêm phiên sửa mới để tiện cho việc theo dõi theo ngày tháng và dễ dàng trích xuất hình ảnh khi sửa chữa.

Danh sách phiên sửa container mã số #123456

Mã container

123456

Ngày sửa

Chọn ngày hết hiệu lực

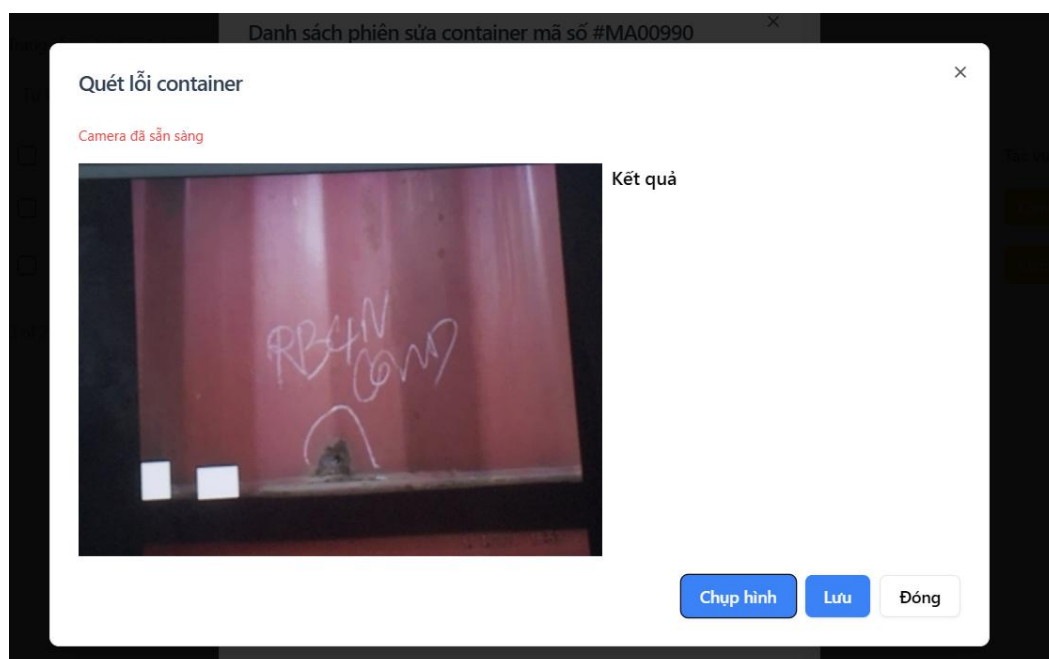
Người sửa

Chọn khách hàng...

Lưu **Đóng**

Hình 5.17: Chọn phiên sửa cho container

Sau khi thêm phiên sửa sẽ tiến đến quét lỗi hư hỏng của container

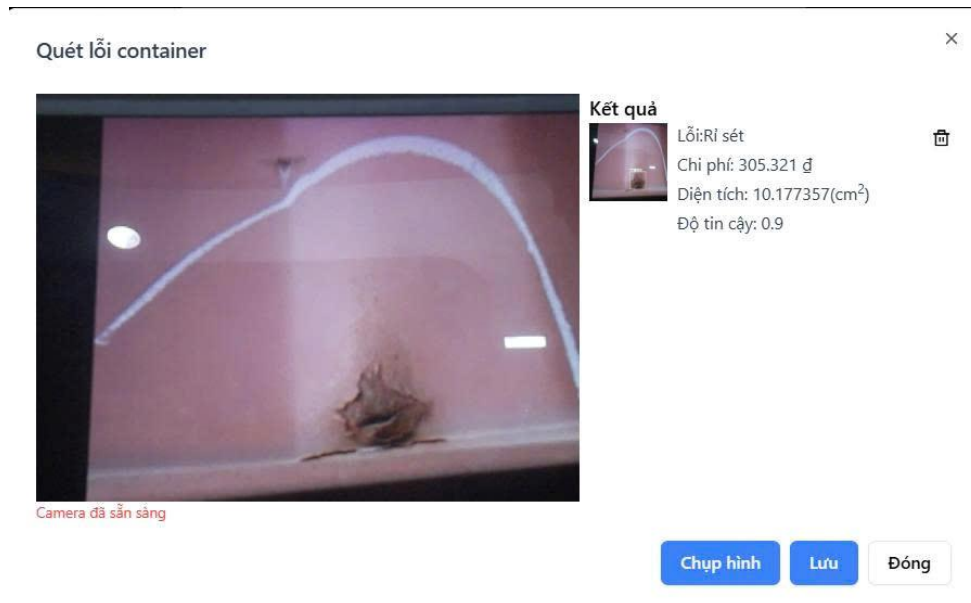


Hình 5.18: Chụp ảnh hư hỏng container

Tại đây, người dùng thấy luồng hình ảnh trực tiếp từ camera (kết nối với Raspberry Pi) và có hai nút "Chụp hình" và "Kết thúc". Khi nhấn "Chụp hình", Raspberry Pi sẽ bắt một khung hình và gửi lên server để xử lý bởi mô hình YOLOv8.

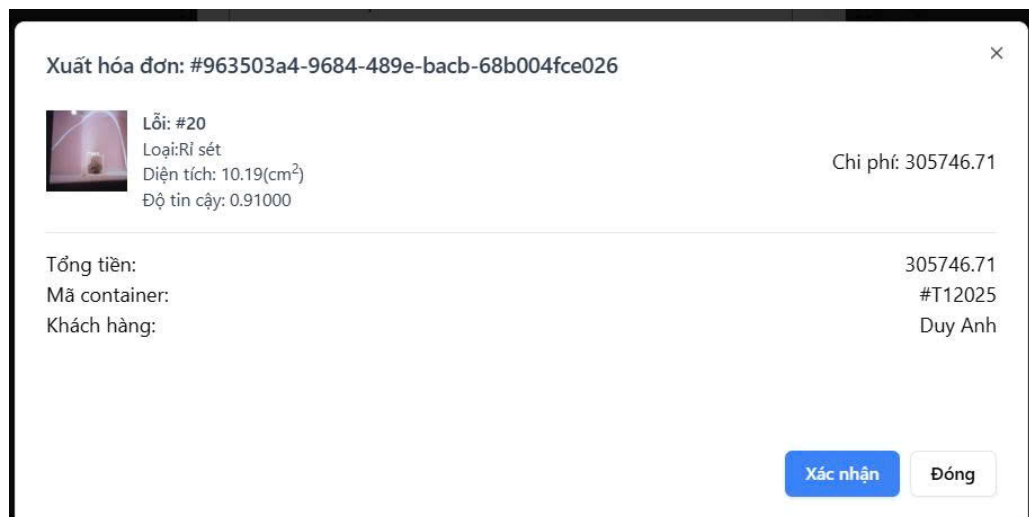
5.5.3. Hiển thị kết quả phát hiện lỗi và quản lý ảnh

Sau khi ảnh được chụp và xử lý, kết quả phát hiện lỗi sẽ được hiển thị trực tiếp trên giao diện web, cho phép người dùng xem chi tiết và quản lý ảnh. Đồng thời hiển thị các thông tin về lỗi, tính toán diện tích, và tính ra giá tiền (dựa trên lý thuyết) mà người dùng phải chi trả để sửa chữa loại lỗi này



Hình 5.19: Giao diện hiển thị kết quả phát hiện lỗi trên ảnh.

Trên hình ảnh container, lỗi rỉ sét đã được phát hiện và khoanh vùng bằng hộp giới hạn, kèm theo độ tin cậy 0.9. Người dùng có thể "Lưu" hoặc tiếp tục chụp cho đến khi đủ hình ảnh các góc độ của container. Sau khi nhận diện được lỗi người dùng có thể chọn in hoá đơn giá tiền sửa chữa.



Hình 5.20: Giao diện in hoá đơn sửa chữa

Duy Anh
Quản lý

Trang chủ > Container

Hiển thị

<input type="checkbox"/>	Mã container	Ngày tạo ↑↓	Khách hàng ↑↓	Mô tả ↑↓	Tác vụ
<input type="checkbox"/>	MA00990	2025-07-08T14:45:50	Duy Anh	Container xoi	<input type="button" value="Chỉnh sửa"/> <input type="button" value="Phiên sửa"/> <input type="button" value="Xóa"/>
<input type="checkbox"/>	21520558	2025-07-09T18:52:05	Duy Anh		<input type="button" value="Chỉnh sửa"/> <input type="button" value="Phiên sửa"/> <input type="button" value="Xóa"/>
<input type="checkbox"/>	123456	2025-07-10T06:25:44	Duy Anh		<input type="button" value="Chỉnh sửa"/> <input type="button" value="Phiên sửa"/> <input type="button" value="Xóa"/>
<input type="checkbox"/>	122222	2025-07-10T06:36:55	Duy Anh	container 2	<input type="button" value="Chỉnh sửa"/> <input type="button" value="Phiên sửa"/> <input type="button" value="Xóa"/>

0 of 4 items

Trước
Sau

Hình 5.21: Giao diện danh sách container với ảnh minh họa.

Hình 5.21 cho thấy giao diện quản lý container,. Giao diện có các trường "Mã container", "Ngày tạo", "Khách hàng", "Mô tả", và "Tác vụ". Các tác vụ bao gồm "Chỉnh sửa", "Phiên sửa", và "Xóa".

Chương 6. KẾT LUẬN

6.1. Kết quả đạt được

Nghiên cứu đã thành công trong việc xây dựng và triển khai một hệ thống phát hiện lỗi trên bề mặt container dựa trên kiến trúc học sâu YOLOv8. Các kết quả thực nghiệm đã chứng minh tính khả thi và hiệu suất đáng kể của mô hình trong việc tự động hóa quy trình kiểm tra chất lượng:

Xây dựng và tiền xử lý tập dữ liệu toàn diện: Nhóm đã thu thập và tổ chức một tập dữ liệu lớn và đa dạng gồm 13827 hình ảnh container, tập trung vào 3 lớp lỗi chính (dent, hole, rust). Quá trình gán nhãn được thực hiện tỉ mỉ bằng Roboflow [10], đảm bảo độ chính xác cao. Các kỹ thuật tiền xử lý hình ảnh (Auto-Orient, Resize Fit, Auto-Adjust Contrast) và tăng cường dữ liệu tiên tiến (Geometric, Color, Mosaic, Mixup) [11], [18] đã được áp dụng hiệu quả, nâng cao chất lượng và tính tổng quát của dữ liệu.

Huấn luyện mô hình YOLOv8m hiệu quả: Mô hình YOLOv8m.pt đã được huấn luyện trên môi trường điện toán đám mây với GPU NVIDIA A100. Quá trình huấn luyện cho thấy sự hội tụ ổn định của các đường cong mất mát và các chỉ số hiệu suất, chứng tỏ mô hình học hiệu quả từ dữ liệu. Các chỉ số hiệu suất trên tập xác thực đạt được ấn tượng: mAP@0.5 là 0.759 và mAP@0.5:0.95 là 0.530 (Bảng 5.4). Những con số này minh chứng rằng mô hình không chỉ có khả năng phát hiện đúng đối tượng lỗi mà còn xác định vị trí của chúng một cách chính xác cao, điều rất quan trọng trong các ứng dụng kiểm tra chất lượng.

Hiệu suất phân loại lỗi đáng kể và chi tiết theo từng lớp: Mô hình thể hiện hiệu suất đặc biệt tốt đối với lỗi rust và hole, với các chỉ số Precision và Recall cao (Bảng 5.5). Mặc dù lỗi dent vẫn còn một số thách thức do tính chất khó nhận diện (phụ thuộc vào ánh sáng và góc nhìn), nhưng tổng thể, mô hình đã cung cấp một giải pháp tự động hóa kiểm tra lỗi đáng tin cậy, giảm thiểu sự chủ quan của con người.

Tốc độ suy luận cao và tiềm năng triển khai biên: Trong môi trường huấn luyện với GPU NVIDIA A100, mô hình đạt tốc độ suy luận ấn tượng khoảng 178 FPS (tức 5.6ms/ảnh). Mặc dù tốc độ này sẽ giảm trên thiết bị nhúng như Raspberry Pi 4, nhưng

phân tích đã chỉ ra rằng với các bước tối ưu hóa sau huấn luyện (như lượng tử hóa mô hình), hiệu suất chấp nhận được (khoảng 10-20 FPS) là hoàn toàn khả thi cho việc triển khai tại biên (edge deployment) [6], [7], [8], mang lại khả năng xử lý tại chỗ và giảm thiểu phụ thuộc vào điện toán đám mây liên tục.

Thiết kế kiến trúc hệ thống tích hợp và toàn diện: Nghiên cứu đã đề xuất và hiện thực hóa một kiến trúc hệ thống tổng thể bao gồm module xử lý hình ảnh trên Raspberry Pi 4, kênh truyền thông an toàn sử dụng Tailscale [15], và các giao diện người dùng (ứng dụng di động, giao diện web). Kiến trúc này không chỉ đảm bảo luồng dữ liệu thông suốt và bảo mật mà còn là nền tảng vững chắc cho việc triển khai thực tế hệ thống kiểm tra tự động, nâng cao hiệu quả vận hành.

Khả năng nhận dạng ký tự quang học (OCR): Việc tích hợp module OCR sử dụng EasyOCR (Mục 4.4) cho thấy tiềm năng trong việc tự động trích xuất thông tin văn bản từ container, cho phép liên kết thông tin lỗi với các mã định danh cụ thể, từ đó nâng cao khả năng theo dõi và quản lý container trong chuỗi cung ứng.

Phát triển giao diện Web trực quan và chức năng quản lý: Hệ thống web được triển khai cung cấp các tính năng quan trọng như Dashboard tổng quan, giao diện nhập liệu số hiệu container và chụp ảnh trực tiếp, cũng như hiển thị kết quả phát hiện lỗi chi tiết trên ảnh và quản lý danh sách container (Mục 5.2). Điều này cải thiện đáng kể trải nghiệm người dùng và khả năng quản lý dữ liệu.

6.2. Những điều đạt được khi thực hiện đề tài

Đã xây dựng được một hệ thống nhúng có thể nhận diện được lỗi hư hỏng của container một cách nhanh chóng với độ chính xác cao.

Hiểu biết hơn về những công nghệ, phương pháp liên quan về nhận diện lỗi hư hỏng container. Ứng dụng được những lý thuyết nhóm tìm hiểu được vào đề tài.

Hiểu biết hơn về phương pháp học sâu, nắm rõ quy trình của một bài toán học sâu. Tìm hiểu được nhiều thuật toán học sâu giải quyết vấn đề của từng bước bài toán trong học sâu. Tìm hiểu được nhiều loại vi xử lý, vi điều khiển, máy tính nhúng cũng như các loại phần cứng, linh kiện điện tử công nghệ cao có thể tìm thấy trên thị trường hiện nay

6.3. Những khó khăn trong quá trình thực hiện đề tài

Khó khăn lớn nhất và tốn nhiều công sức nhất đến từ dữ liệu. Việc dán nhãn thủ công cho số lượng ảnh lớn đòi hỏi nhiều thời gian và công sức. Tiếp đến là việc thu thập dữ liệu từ thực tế gặp phải nhiều khó khăn từ việc xin phép vào các depot để chụp ảnh hư hỏng và thiếu số lượng container hư hỏng với các lỗi đa dạng để thu thập.

Về mặt kỹ thuật, nhóm không thể triển khai mô hình trực tiếp trên Pi. Với nguồn tài nguyên phần cứng (RAM, CPU) hạn chế so với máy chủ đám mây, không thể duy trì hiển thị màn hình chụp ảnh >15fps trong một thời gian dài. Điều này gắn liền với quá trình tối ưu hóa và tinh chỉnh mô hình, một công việc lặp đi lặp lại và tốn kém tài nguyên để tìm ra bộ siêu tham số tốt nhất cho thuật toán.

Vấn đề cuối cùng là việc tích hợp và đảm bảo hệ thống hoạt động ổn định. Do mô hình hoạt động dưới hình thức (client-Sever) nên việc duy trì kết nối mạng là bắt buộc. Thiết lập kết nối mạng wifi cho Pi và giữ cho Pi không bị mất kết nối gây ra một chút khó khăn cho không có màn hình hiển thị. Tuy nhiên đây chỉ là một vấn đề nhỏ và có thể cải thiện sớm trong thời gian gần nhất.

6.4. Hướng phát triển chính cho đề tài

Hiện tại hệ thống là một phần nhỏ của hệ thống khung chụp ảnh lỗi container lớn được sử dụng ở các cảng. Hệ thống có tiềm năng phát triển thêm thành một mô hình lớn, cung cấp nhiều camera với nhiều góc độ khác nhau. Các camera sẽ chụp đồng thời hình ảnh container với đầy đủ các góc theo tiêu chuẩn ICLL và cho ra kết quả hư hỏng nhanh hơn hiện tại.

Cải thiện nâng cao độ chính xác bằng một lượng tệp dữ liệu lớn hơn được huấn luyện trên một môi trường đủ mạnh để cho ra được chỉ số dự đoán trên 0.9. Có thể áp dụng model YoloV12 để huấn luyện mô hình, lợi thế mới ra mắt và được tối ưu một cách tốt nhất bởi Ultralytics có thể nâng cao được độ chính xác của mô hình mà không phải huấn luyện quá nhiều epoch và tiết kiệm được thời gian huấn luyện hơn so với YoloV8

Tập trung vào việc nâng cấp phần mềm và khả năng tích hợp. Hệ thống có thể được bổ sung các thuật toán xử lý ảnh nâng cao sử dụng OpenCV để làm nổi bật các đặc trưng của lỗi trước khi đưa vào mô hình. Giao diện người dùng cũng sẽ được phát triển thành một dashboard tương tác, cung cấp các biểu đồ phân tích xu hướng lỗi và cảnh báo thông minh theo mức độ nghiêm trọng.

TÀI LIỆU THAM KHẢO

Tài liệu tham khảo

- [1] C. Indranath and C. Gyusung, "Port container terminal quay crane allocation based on simulation and machine learning method," *Sens. Mater.*, vol. 34, no. 1, pp. 1–11, 2022.
- [2] Daga, R. P., et al., "Performance analysis of object detection models on Raspberry Pi for real-time applications," in *Proc. Int. Conf. Adv. Comput. Commun. Technol. (ICACCT)*, 2021, pp. 1-6.
- [3] Institute of International Container Lessors, *Container Inspection Guidelines*, IICL Standards, 2021.
- [4] Kim, H. C. and Kwon, Y. R., "Addressing class imbalance in deep learning for image classification: A survey," *IEEE Access*, vol. 9, pp. 110410-110427, 2021.
- [5] K. Magdalena, L. Dorota, B. Karolina, et al., "Review of the container ship loading model—cause analysis of cargo damage and/or loss," *Polish Maritime Res.*, vol. 29, no. 1, pp. 26–35, 2022.
- [6] Lin, Y., et al., "Optimized YOLOv5s model for real-time detection of insulator defects based on edge computing," *Optik*, vol. 259, p. 168995, 2022.
- [7] Liu, M., et al., "A lightweight YOLOv5 model for real-time defect detection of power lines," *IEEE Access*, vol. 10, pp. 110931-110941, 2022.
- [8] M. Deepak, K. Prashant, S. Sunil, et al., "A metaphor analysis on vehicle license plate detection using YOLO-NAS and YOLOv8," *J. Electr. Syst.*, vol. 20, no. 1, pp. 152–164, 2024.
- [9] OpenCV. "OpenCV Documentation," [Online]. Available: <https://docs.opencv.org/>. [Accessed: Jun. 10, 2025].

- [10] Roboflow, "Roboflow Documentation," [Online]. Available: <https://docs.roboflow.com/>. [Accessed: Jun. 10, 2025].
- [11] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 779-788.
- [12] Shorten, A. and Khoshgoftaar, T., "A survey on image data augmentation for deep learning," J. Big Data, vol. 6, no. 1, p. 60, 2019.
- [13] Szeliski, R., Computer Vision: Algorithms and Applications. Springer, 2010.
- [14] Raschka, S. and Mirjalili, V., Python Machine Learning. Packt Publishing Ltd, 2019.
- [15] Tailscale. "Tailscale Documentation," [Online]. Available: <https://tailscale.com/docs/>. [Accessed: Jun. 10, 2025].
- [16] T. Nguyen Thi Phuong and G. S. Cho, "Automating container damage detection with the YOLO-NAS deep learning model," Science Progress, vol. 108, no. 1, pp. 1–21, Jan. 2025.
- [17] T. V. Anh, "Logistics challenges for SMEs in Vietnam," Vietnam Logist. Rev., vol. 8, no. 4, pp. 89–95, 2023.
- [18] Ultralytics, "YOLOv8 Documentation," [Online]. Available: <https://docs.ultralytics.com/>. [Accessed: Jun. 10, 2025].
- [19] W. Zixin, G. Jing, Z. Qingcheng, et al., "Multitype damage detection of containers using CNN based on transfer learning," Math. Probl. Eng., vol. 2021, pp. 1–12, 2021.
- [20] Yogesh, K. and Pankaj, K., "Comparative study of YOLOv8 and YOLO-NAS for agriculture application," in Proc. Int. Conf. Signal Process. Integr. Net., 2024, pp. 72–77.