

TRƯỜNG ĐẠI HỌC ĐẠI NAM  
KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO TỔNG KẾT**  
**ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN**  
**NGHIÊN CỨU, THỬ NGHIỆM NHẬN DẠNG ĐỐI**  
**TƯỢNG TRONG ẢNH SỬ DỤNG YOLO**

**Lĩnh vực nghiên cứu: Công nghệ thông tin**

**Hà Nội, Tháng 4, năm 2025**

TRƯỜNG ĐẠI HỌC ĐẠI NAM  
KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO TỔNG KẾT**  
**ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN**  
**NGHIÊN CỨU, THỬ NGHIỆM NHẬN DẠNG ĐỐI**  
**TƯỢNG TRONG ẢNH SỬ DỤNG YOLO**

**Người hướng dẫn** <TS. Trần Quý Nam>

**Sinh viên thực hiện:**

1. Ngô Văn Minh (nhóm trưởng, lớp CNTT 17-05)
2. Dương Ngọc Đông, lớp CNTT 17-02
3. Nguyễn Thị Thanh Nhã, lớp CNTT 17-02
4. Nguyễn Tiến Lực, lớp CNTT 17-02

Khoa : Công nghệ thông tin

Ngành học : Công nghệ thông tin

**Giảng viên hướng dẫn**

**Trưởng nhóm**

**Hà Nội, Tháng 4, năm 2025**

## LỜI NÓI ĐẦU

Trong thời đại công nghệ hiện nay, xe hơi không chỉ là phương tiện di chuyển mà còn là một phần không thể thiếu trong cuộc sống hàng ngày. Tuy nhiên, dù có bền bỉ đến đâu, chẳng ai tránh khỏi những lúc “vỗ xe bị xước, đèn pha bị vỡ” sau một cú va chạm bất ngờ hay một lỗi nhỏ trong quá trình sử dụng. Và nếu bạn là người trong ngành bảo hiểm hoặc sửa chữa ô tô, việc xác định chính xác bộ phận bị hư hỏng và đánh giá mức độ thiệt hại là một công việc tốn thời gian, công sức và đôi khi chỉ có những kỹ thuật viên “có tâm” mới làm được. Vậy tại sao không để cho trí tuệ nhân tạo giúp ta một tay, phải không?

Chính vì vậy, trong nghiên cứu này, nhóm chúng tôi đã quyết định thử nghiệm mô hình YOLOv8, một trong những phiên bản tiên tiến của bộ nhận diện vật thể "You Only Look Once" (YOLO), để nhận diện bộ phận xe bị hư hỏng. Chúng tôi không phải là những người xây dựng mô hình từ con số không, mà thay vào đó, sử dụng mã nguồn có sẵn, để tiếp cận và đánh giá khả năng của mô hình YOLOv8 trong một bài toán thực tế.

Công việc của chúng tôi không phải là tạo ra một cuộc cách mạng trong ngành ô tô, nhưng hy vọng qua thử nghiệm này, chúng tôi sẽ giúp bạn hình dung được một phần nhỏ của sự thay đổi mà trí tuệ nhân tạo có thể mang lại cho việc nhận diện và xử lý hư hỏng xe. Đừng mong đợi một dự án hoành tráng hay đầy “sáng tạo độc đáo”, nhưng chúng tôi cam kết rằng thử nghiệm này là một bước đi thực tế và có giá trị đối với những ai muốn áp dụng AI vào công việc sửa chữa xe cộ.

Trải qua quá trình huấn luyện và thử nghiệm với bộ dữ liệu thực tế, chúng tôi đã thấy được những điểm mạnh và hạn chế của YOLOv8 trong việc nhận diện các bộ phận hư hỏng của xe. Bên cạnh những kết quả khả quan, đôi khi nó cũng gặp một số “trục trặc” thú vị (đừng lo, tất cả đều có giải pháp). Và tất nhiên, chúng tôi sẽ không bỏ qua những cơ hội để cải thiện hiệu suất của mô hình trong tương lai.

Cuối cùng, thử nghiệm này không chỉ đơn thuần là về nhận diện xe bị hư hỏng, mà là một thử thách nhỏ trong việc đưa công nghệ AI vào một ngành công nghiệp đầy tính ứng dụng như sửa chữa ô tô. Và nếu bạn đã chuẩn bị sẵn sàng, hãy cùng chúng tôi bước vào thế giới của YOLOv8, nơi mọi bộ phận xe, dù là vỏ xe hay đèn pha, đều có thể được nhận diện trong "nháy mắt".

## MỤC LỤC

<b>LỜI NÓI ĐẦU.....</b>	<b>3</b>
<b>MỤC LỤC .....</b>	<b>5</b>
<b>CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....</b>	<b>7</b>
1.1 Tổng quan về đề tài .....	7
1.1.1. Bối cảnh và ý nghĩa .....	7
1.1.2. Mục tiêu nghiên cứu .....	7
1.2. Tổng quan về nhận diện bộ phận xe hư hỏng.....	8
1.2.1. Các Phương Pháp Truyền Thống .....	8
1.2.2. Ứng Dụng Trí Tuệ Nhân Tạo Trong Nhận Diện Hư Hỏng.....	8
1.3. YOLOv8 - Siêu Nhân Trong Làng Nhận Diện Hình Ảnh .....	9
1.4. Các phiên bản YOLO từ YOLOv1 đến YOLOv8.....	10
1.5. Kiến trúc và cải tiến của YOLOv8 .....	11
1.5.1. Cấu trúc mạng YOLOv8 .....	11
1.5.2. Cơ chế hoạt động và tối ưu hóa.....	11
1.4. Các khía cạnh cần biết khi nghiên cứu YOLOv8.....	13
1.4.1. Các tham số và cách tinh chỉnh mô hình.....	13
1.4.2. Đánh giá hiệu suất mô hình bằng các chỉ số .....	15
<b>CHƯƠNG 2. PHÂN TÍCH DỮ LIỆU .....</b>	<b>16</b>
2.1. Thu thập dữ liệu.....	16
2.2. Làm sạch dữ liệu.....	17
2.3 Xây dựng mô hình .....	19
<b>CHƯƠNG 3: THỬ NGHIỆM MÔ HÌNH HỌC SÂU YOLOv8 .....</b>	<b>21</b>

3.1. Giới Thiệu Quá Trình Thử Nghiệm .....	21
3.2. Quá Trình Đào Tạo Mô Hình .....	21
3.3 Thử nghiệm mô hình .....	24
3.4. Triển khai sử dụng flask làm giao diện cho nhận diện hư hỏng ô tô .....	26
<b>KẾT LUẬN .....</b>	<b>30</b>
<b>PHÂN CÔNG NHIỆM VỤ .....</b>	<b>32</b>
<b>DANH MỤC TÀI LIỆU THAM KHẢO .....</b>	<b>33</b>

# CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

## 1.1 Tổng quan về đề tài

### *1.1.1. Bối cảnh và ý nghĩa*

Xe cộ hóa là một trụ cột chính trong giao thông hiện đại, nhưng việc hỏng hóc do va chạm hay hao mòn là điều không thể tránh khỏi. Nếu mỗi lần xe trầy xước, méo mó mà phải đưa ra gara nhờ bác thợ sửa thì vừa mất thời gian vừa hao tiền tốn của.

Trong thời đại 4.0, không có lý do gì mà con người không thể tận dụng trí tuệ nhân tạo (AI) để hỗ trợ việc nhận diện hư hỏng xe một cách tự động. Mô hình YOLOv8 ("You Only Look Once" phiên bản mới nhất) là một trong những công cụ mạnh mẽ giúp giải quyết bài toán này. Nó nhanh, chính xác và gọn nhẹ, giúp các kỹ thuật viên xác định bộ phận hư hỏng chỉ trong nháy mắt.

Việc áp dụng mô hình này trong nhận diện bộ phận xe hư hỏng mở ra tiềm năng lớn trong ngành sửa chữa ô tô, bảo hiểm xe hơi và giám sát chất lượng phương tiện. Nếu trước đây, các công ty bảo hiểm phải cử người đi thẩm định từng chiếc xe sau tai nạn thì giờ đây, chỉ cần một chiếc camera và mô hình YOLOv8, mọi thứ sẽ được xử lý gọn gàng hơn nhiều.

### *1.1.2. Mục tiêu nghiên cứu*

Báo cáo này tập trung vào:

1. Thử nghiệm mô hình YOLOv8 để nhận diện bộ phận xe bị hư hỏng.
2. Đánh giá hiệu suất mô hình dựa trên tập dữ liệu thực tế.
3. Phân tích kết quả và đề xuất hướng cải thiện.

Nhóm chúng tôi không có tham vọng xây dựng mô hình AI từ đầu (vì chúng tôi không có 10 năm để nghiên cứu như OpenAI), mà thay vào đó, sử dụng mã nguồn có sẵn để tiến hành thử nghiệm và đánh giá hiệu suất thực tế của

YOLOv8. Chúng chúng chúng tôi muốn xem liệu mô hình này có thực sự hữu dụng trong việc phát hiện các vết trầy xước, móp méo trên xe hay không.

## 1.2. Tổng quan về nhận diện bộ phận xe hư hỏng

### 1.2.1. Các Phương Pháp Truyền Thống

Trước khi AI và Deep Learning lên ngôi, việc kiểm tra hư hỏng xe chủ yếu dựa vào:

- **Kiểm tra thủ công:** Nhân viên kỹ thuật phải chui vào gầm xe, soi từng vết xước, rồi gật gù kết luận “Hỏng nặng đây anh ạ” (dù thực ra có khi chỉ là vết móp nhẹ).
- **Hệ thống quét bằng cảm biến:** Một số hệ thống tiên tiến sử dụng tia X hoặc sóng siêu âm để phát hiện hư hỏng bên trong, nhưng chi phí lắp đặt và vận hành rất đắt đỏ.
- **Sử dụng quy tắc lập trình:** Một số phần mềm đơn giản dùng cách phân tích hình ảnh dựa trên màu sắc, độ sáng, nhưng nếu xe quá bẩn thì... bó tay!

Tuy nhiên, những phương pháp này có hạn chế:

- **Mất nhiều thời gian** nếu thực hiện thủ công.
- **Độ chính xác phụ thuộc vào kinh nghiệm kỹ thuật viên.**
- **Không thể tự động hóa hoàn toàn.**

### 1.2.2. Ứng Dụng Trí Tuệ Nhân Tạo Trong Nhận Diện Hư Hỏng

Với sự phát triển của AI, các mô hình Deep Learning như CNN (Convolutional Neural Network) và đặc biệt là YOLO đã tạo ra bước đột phá lớn:

- YOLOv8 có thể nhận diện và khoanh vùng bộ phận hư hỏng trong thời gian thực.
- Khả năng nhận diện ngay cả khi có nhiều đối tượng trong ảnh.
- Dễ dàng tích hợp vào hệ thống kiểm tra tự động.

Các ứng dụng thực tế bao gồm:



- **Giám định bảo hiểm xe hơi tự động:** Các công ty bảo hiểm có thể dùng AI để xác định mức độ thiệt hại xe mà không cần cử chuyên gia đến tận nơi.
- **Sửa chữa ô tô thông minh:** Hỗ trợ kỹ thuật viên xác định hư hỏng nhanh hơn, không còn cảnh "soi kính lúp tìm vết xước" như trước đây.
- **Hệ thống giám sát phương tiện giao thông:** Phát hiện xe bị hư hỏng ngay khi di chuyển trên đường, giảm nguy cơ tai nạn.

Nói cách khác, nếu các phương pháp truyền thống giống như "bác thợ sửa xe soi từng chi tiết", thì YOLOv8 giống như một kỹ sư AI với đôi mắt siêu phàm, có thể phát hiện mọi lỗi chỉ trong chớp mắt.

### 1.3. YOLOv8 - Siêu Nhân Trong Làng Nhận Diện Hình Ảnh

YOLO (You Only Look Once) là một dòng mô hình nhận diện ảnh nổi tiếng với ưu điểm vượt trội:

- **Nhanh như điện:** YOLO có thể phân tích hàng chục khung hình trong một giây, giúp nhận diện hư hỏng gần như ngay lập tức.
- **Chính xác:** Phiên bản YOLOv8 có độ chính xác cao hơn nhiều so với các phiên bản trước.
- **Gọn nhẹ:** Dễ dàng triển khai trên các thiết bị có phần cứng hạn chế.

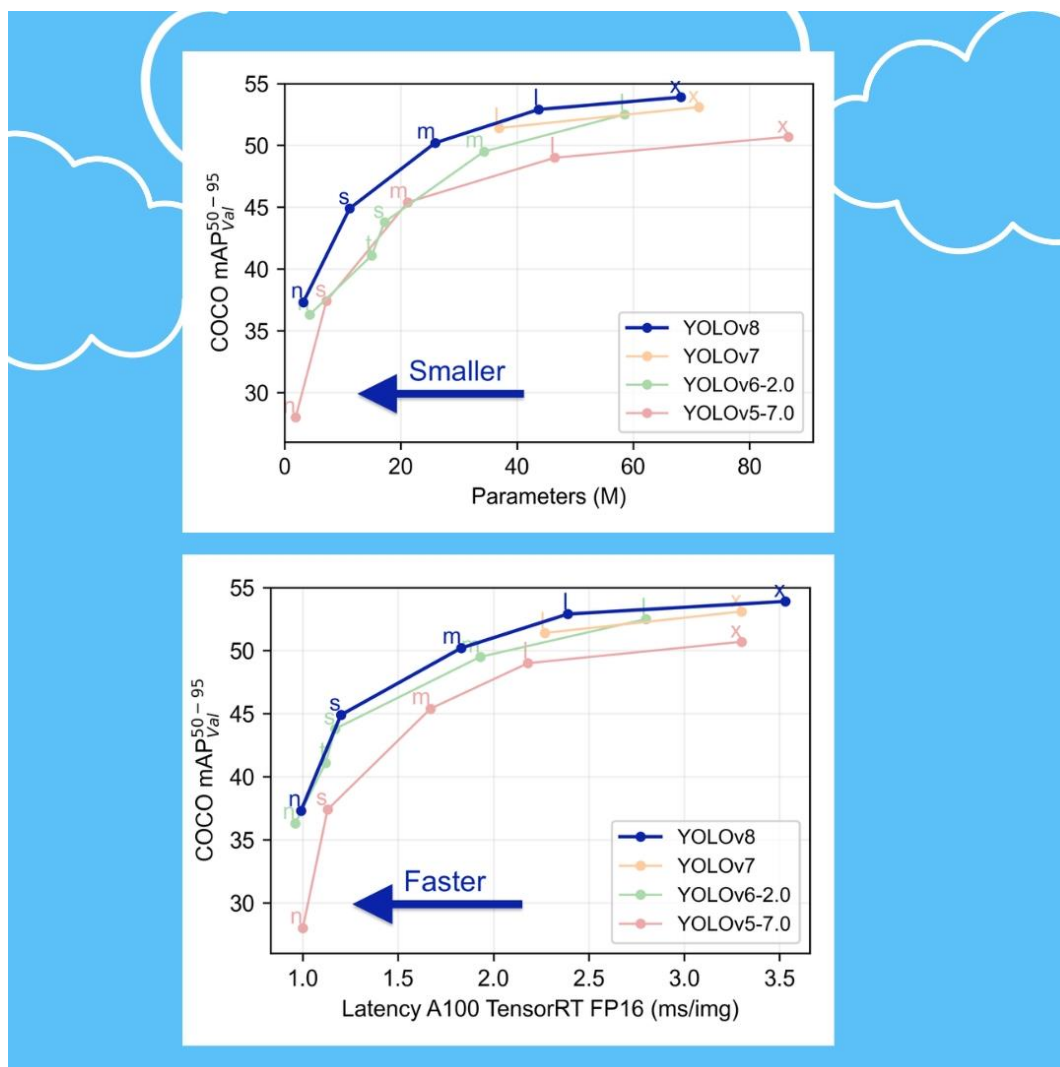
YOLO (You Only Look Once) là một trong những mô hình nhận diện đối tượng nhanh và chính xác nhất hiện nay. Thay vì quét hình ảnh nhiều lần như Faster R-CNN, YOLO chỉ nhìn hình ảnh đúng một lần (đúng như tên gọi) và chia thành các lưới nhỏ để dự đoán vị trí, nhãn của các đối tượng trong hình.

Nói cách khác, nếu các phương pháp truyền thống giống như "bác thợ sửa xe soi từng chi tiết", thì YOLOv8 giống như một kỹ sư AI với đôi mắt siêu phàm, có thể phát hiện mọi lỗi chỉ trong chớp mắt.

## 1.4. Các phiên bản YOLO từ YOLOv1 đến YOLOv8

YOLO đã phát triển qua nhiều phiên bản, mỗi phiên bản có những cải tiến đáng kể:

- YOLOv1: Ra mắt năm 2016, nhanh nhưng chưa chính xác lắm.
- YOLOv2 & YOLOv3: Cải thiện độ chính xác và tốc độ.
- YOLOv4 & YOLOv5: Tối ưu hóa hơn về hiệu suất.
- YOLOv6 & YOLOv7: Nâng cấp kiến trúc, tăng tốc độ và độ chính xác.
- YOLOv8: Phiên bản mới nhất, áp dụng các kỹ thuật tối ưu hóa hiện đại, đạt hiệu suất cao.



Biểu đồ so sánh tốc độ và độ chính xác của các phiên bản YOLO.

Như chúng ta có thể thấy từ biểu đồ, YOLOv8 có nhiều tham số hơn so với các phiên bản tiền nhiệm như YOLOv5, nhưng ít tham số hơn so với YOLOv6. Nó cung cấp khoảng 33% mAP nhiều hơn cho các mô hình kích thước n và mAP lớn hơn nói chung.

Từ biểu đồ thứ hai, chúng ta có thể thấy YOLOv8 có thời gian suy luận nhanh hơn so với tất cả các phiên bản YOLO khác.

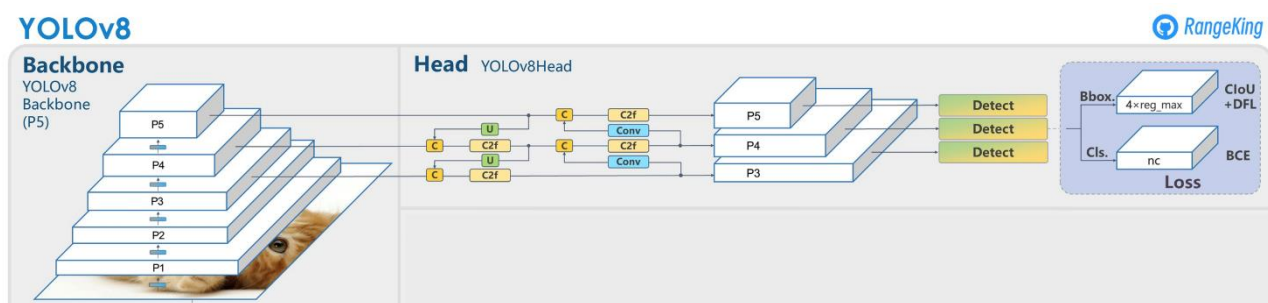
## 1.5. Kiến trúc và cải tiến của YOLOv8

### 1.5.1. Cấu trúc mạng YOLOv8

YOLOv8 sử dụng các thành phần chính:

- Backbone: Trích xuất đặc trưng từ hình ảnh đầu vào.
- Neck: Tăng cường các đặc trưng quan trọng.
- Head: Dự đoán vị trí, kích thước và nhãn của đối tượng.

Ví dụ thực tế: Giống như một thợ săn giỏi, mắt của bạn (backbone) sẽ phát hiện con mồi, bộ não (neck) sẽ phân tích xem đó là gì, còn tay (head) sẽ hành động.



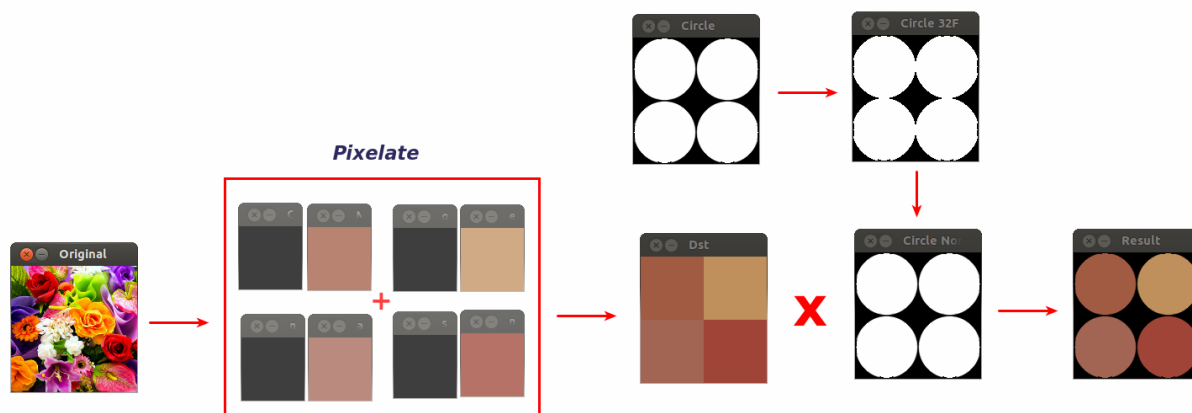
Mình họa kiến trúc của YOLOv8 với sự tương đồng của bộ não con người.

### 1.5.2. Cơ chế hoạt động và tối ưu hóa

YOLOv8 sử dụng kỹ thuật tối ưu hóa như:

- Anchor-free Detection: Không cần hộp giới hạn trước, giúp mô hình linh hoạt hơn.

- Mosaic Augmentation: Ghép nhiều hình ảnh để tăng khả năng tổng quát.
- AutoBatch & AutoShape: Điều chỉnh kích thước batch và shape một cách thông minh.



YOLOv8 được so sánh với các mô hình nhận diện đối tượng phổ biến như Faster R-CNN, SSD, và các phiên bản YOLO trước đó.

Mô hình	Tốc độ (FPS)	Độ chính xác (mAP)	Độ phức tạp
Faster R-CNN	7-10 FPS	Cao	Rất cao
SSD	20-30 FPS	Trung bình	Trung bình
YOLOv7	50-60 FPS	Cao	Trung bình
<b>YOLOv8</b>	<b>70-80 FPS</b>	<b>Rất cao</b>	<b>Thấp</b>

YOLOv8 nhanh hơn Faster R-CNN và SSD đáng kể. Với tốc độ lên tới 80 FPS, YOLOv8 có thể xử lý video theo thời gian thực. YOLOv8 cải thiện mAP so với các mô

hình trước đó, giúp phát hiện đối tượng chính xác hơn. YOLOv8 có cấu trúc đơn giản hơn Faster R-CNN, giúp triển khai dễ dàng hơn mà vẫn đạt hiệu suất cao. Nếu YOLOv8 là một tay đua F1, thì Faster R-CNN là một người đi bộ. Nếu YOLOv8 là một máy quét mã vạch tốc độ cao, thì SSD là một nhân viên thu ngân nhập tay từng con số.

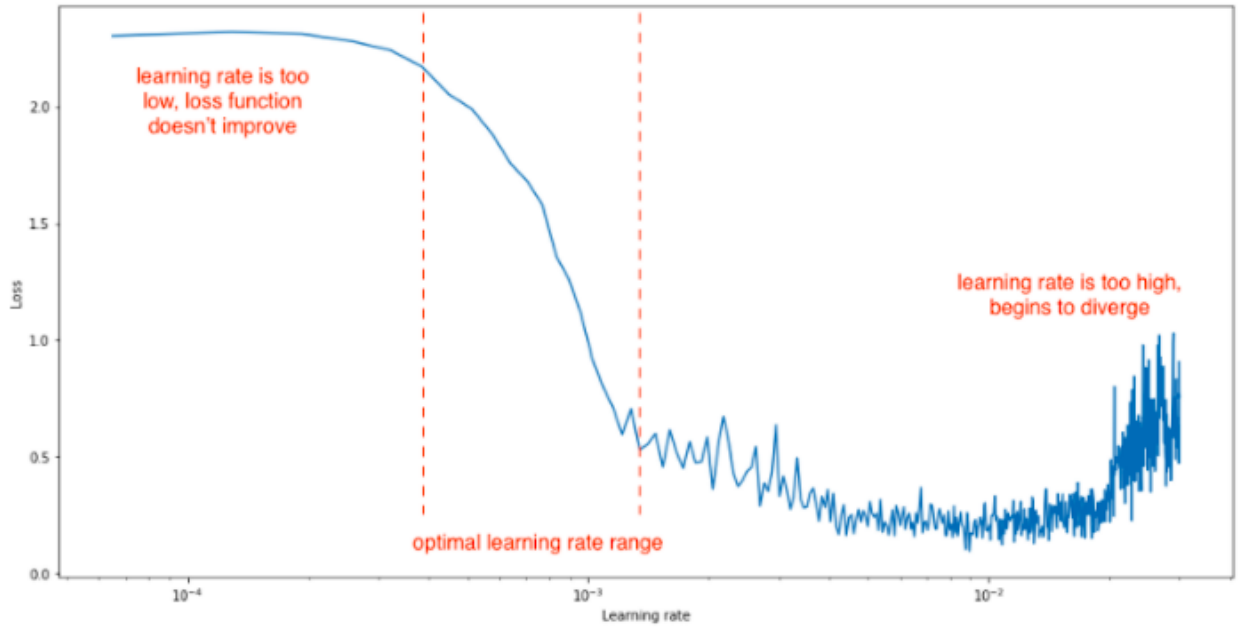
## **1.4. Các khía cạnh cần biết khi nghiên cứu YOLOv8**

### ***1.4.1. Các tham số và cách tinh chỉnh mô hình***

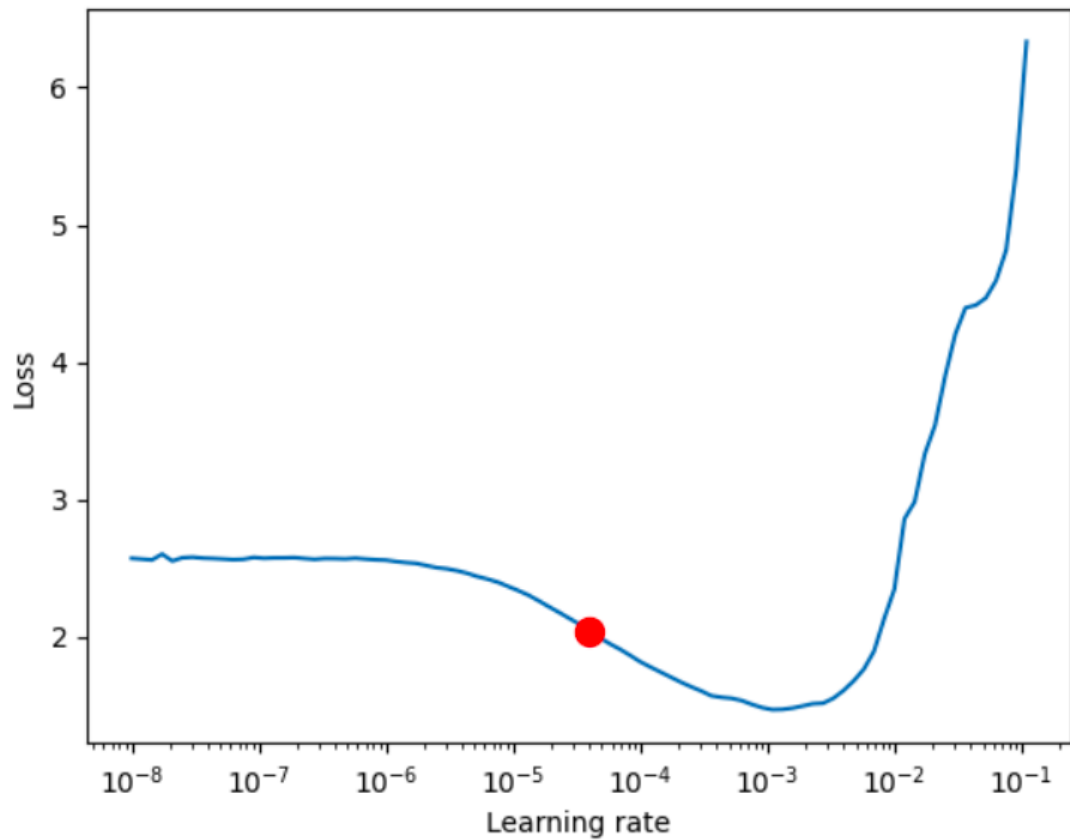
Khi huấn luyện YOLOv8, một trong những siêu tham số quan trọng nhất đối với quá trình huấn luyện đó chính là learning rate. Learning rate được hiểu là một phân tỷ lệ của một bước dịch chuyển trọng số mô hình được cập nhật theo các mini-batch truyền vào. Độ lớn của learning rate sẽ ảnh hưởng trực tiếp tới tốc độ hội tụ của hàm loss function tới điểm cực trị toàn cục.

Nếu lựa chọn learning rate quá thấp có thể khiến mô hình hội tụ chậm, thậm chí không vượt qua được các điểm cực trị địa phương. Lựa chọn learning rate quá cao cũng khiến cho mô hình gặp hiện tượng step over xung quanh giá trị cực trị địa phương và không hội tụ.

Tăng dần learning rate theo batch iteration và theo dõi loss function trên từng batch iteration. Những vị trí learning rate nhỏ có loss function nằm ngang, learning rate phù hợp thì loss function giảm dần, learning rate tăng cao sẽ khiến cho hàm loss function tăng hoặc dao động ngẫu nhiên như bên dưới:



Khi đó chúng ta sẽ lựa chọn learning rate là điểm màu đỏ nằm giữa đường hạ dốc của loss function như hình dưới:




### 1.4.2. Đánh giá hiệu suất mô hình bằng các chỉ số

Các chỉ số quan trọng để đánh giá YOLOv8:

- **mAP (Mean Average Precision):** Độ chính xác trung bình.
- **IoU (Intersection over Union):** Mức độ trùng khớp giữa hộp dự đoán và thực tế.
- **FPS (Frames Per Second):** Số khung hình nhận diện mỗi giây.

**Ví dụ thực tế:** Nếu YOLOv8 nhận diện con chó trong hình nhưng dự đoán hộp bao quanh nó chỉ chiếm 50% so với hộp thực tế, thì  $IoU = 0.5$ .

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


## CHƯƠNG 2. PHÂN TÍCH DỮ LIỆU

### 2.1. Thu thập dữ liệu

Đối với đề tài này, một trong những thách thức lớn mà chúng tôi phải đối mặt là việc thu thập hình ảnh của những chiếc xe bị hư hỏng. Điều này thực sự không phải là một việc dễ dàng, bởi vì các tập dữ liệu hình ảnh liên quan đến xe bị hư hỏng không phải lúc nào cũng có sẵn công khai và dễ dàng tiếp cận. Đặc biệt đối với một dự án tùy chỉnh như thế này, việc tự mình thu thập dữ liệu trở thành phần tốn thời gian nhất và có thể khiến bất kỳ ai cũng cảm thấy kiệt sức, đặc biệt khi bạn phải làm việc trong một khoảng thời gian hạn chế.

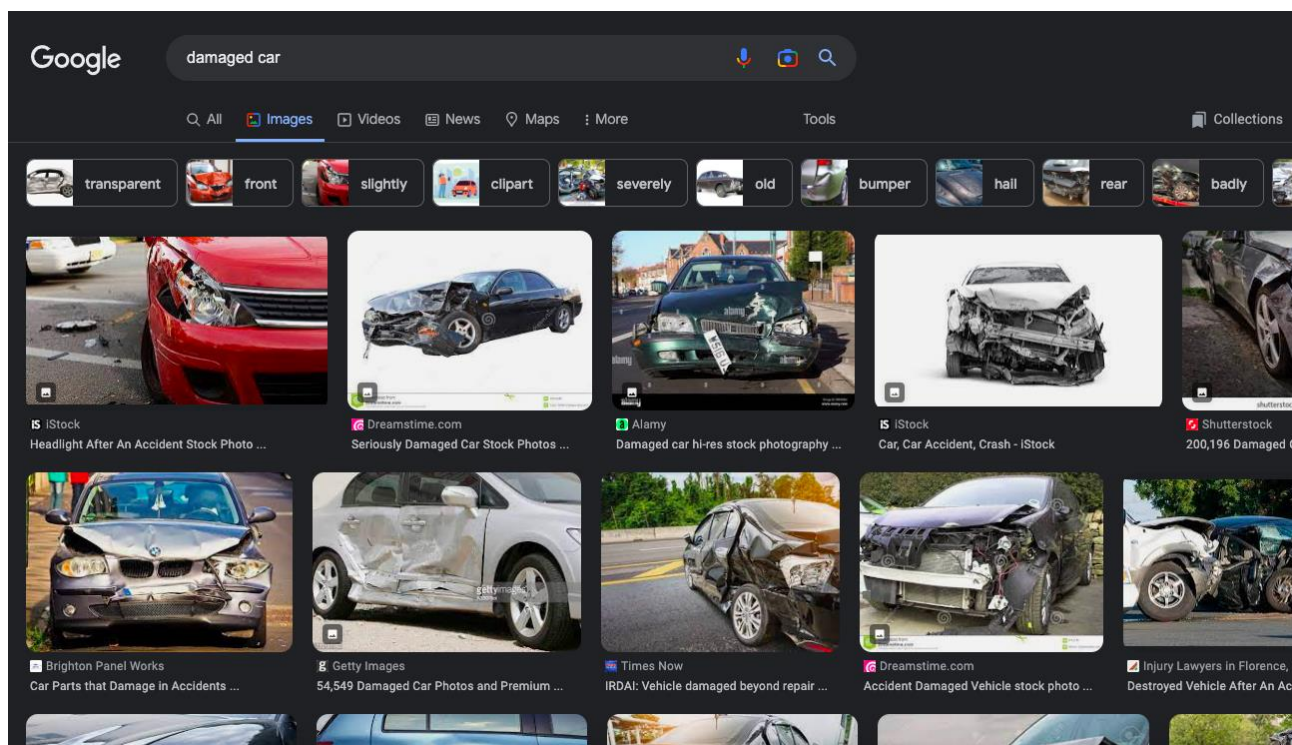
Thực tế, chúng tôi phải tự mình hoàn thành toàn bộ dự án này mà không có sự trợ giúp từ ai khác ngoài những nguồn tài nguyên trên internet. Điều này có thể dễ dàng trở thành một thử thách lớn, và vì vậy, chúng tôi cần phải tìm ra những cách thức hiệu quả để thu thập lượng hình ảnh lớn nhanh chóng, tiết kiệm thời gian. Và đó chính là lúc chúng chúng tôi tìm thấy một giải pháp cực kỳ hữu ích.

Một trong những công cụ tuyệt vời mà chúng chúng tôi phát hiện ra chính là một plugin Chrome có tên **Download All Images**. Đây là một công cụ có thể tự động quét và chọn tất cả các hình ảnh có trên một trang web, sau đó tải xuống dưới dạng một thư mục zip. Mặc dù có một số hạn chế nhỏ về khả năng lựa chọn các hình ảnh chính xác hoặc hạn chế số lượng tải xuống từ một số trang web, nhưng với chúng chúng tôi, công cụ này thực sự đã giúp cuộc sống dễ dàng hơn rất nhiều. Nó tiết kiệm cho chúng chúng tôi rất nhiều thời gian và công sức, đặc biệt khi chúng chúng tôi cần thu thập hàng nghìn hình ảnh trong thời gian ngắn.

Cách thực hiện rất đơn giản. Ta chỉ cần vào Google Hình ảnh và tìm kiếm với từ khóa như "damaged car". Ngay lập tức, ta sẽ nhận được hàng nghìn kết quả từ các trang web khác nhau, đủ để phục vụ cho bài toán của mình. Với plugin này, chỉ trong vài phút, ta đã có thể tải về một bộ sưu tập hình ảnh khổng lồ mà không phải mất công tải từng bức



ảnh một cách thủ công. Tuy không hoàn hảo, nhưng với một dự án như thế này, đây thực sự là một giải pháp nhanh chóng và hiệu quả.



Nó sẽ tự động tải xuống tất cả hình ảnh cho chúng ta trong một thư mục zip. Giải nén nó vào vị trí mong muốn trên hệ thống của đã hoàn tất.

## 2.2. Làm sạch dữ liệu

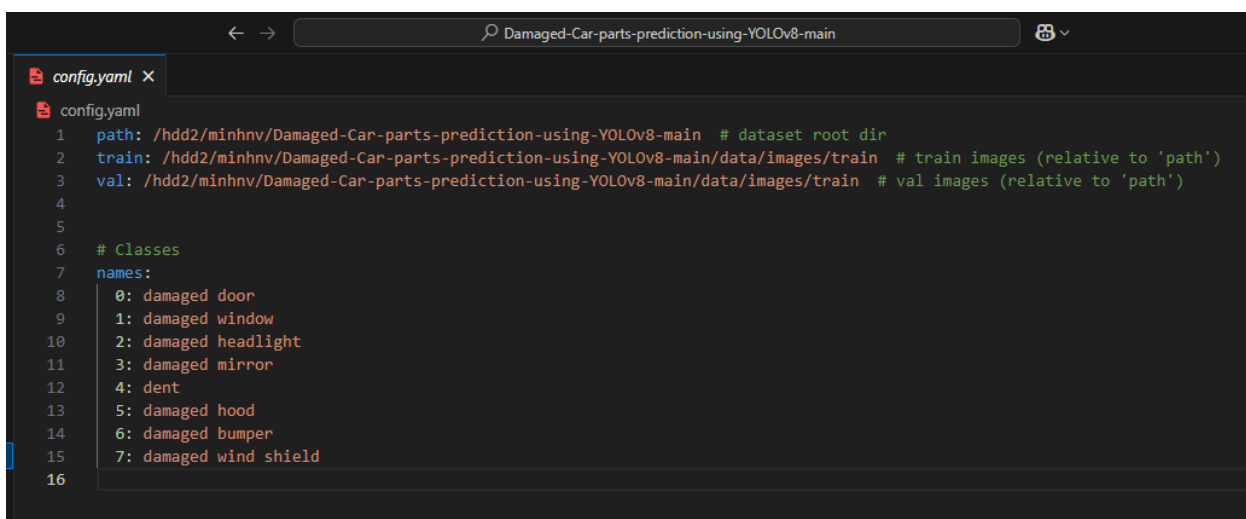
Trước khi bắt đầu cho những hình ảnh của bạn “nhập môn” vào quá trình đào tạo, việc tổ chức thư mục dữ liệu một cách ngăn nắp là cực kỳ quan trọng. Chúng ta không muốn cả đống hình ảnh rơi tung như đống bừa bộn trong phòng đúng không? Đầu tiên, trong thư mục chính của dự án, hãy tạo một thư mục có tên là **data**. Bên trong thư mục **data**, bạn cần tạo hai thư mục con là **images** và **labels**, mỗi thư mục này đều cần một thư mục con nữa có tên là **train**.

Để làm cho mọi thứ dễ dàng hơn cho mắt và đỡ loạn xạ, bạn hãy sao chép tất cả các hình ảnh của mình vào thư mục **train** trong thư mục **images**. Tương tự, hãy chuyển các tệp chú thích của bạn vào thư mục **train** trong thư mục **labels**. Làm như vậy, bạn sẽ có

một tổ chức dữ liệu chuẩn chỉnh, gọn gàng như một chiếc vali đã được xếp đầy đủ quần áo, chờ sẵn để đi du lịch – hoặc ít nhất là để mô hình học!

```
project/
└── data/
    ├── images/
    │   ├── train/
    │   └── labels/
    └── train/
```

Sau khi đã có tất cả dữ liệu gọn gàng trong các thư mục, giờ đến lúc chuẩn bị mã. Để mọi thứ vận hành trơn tru, bạn cần chuẩn bị một tệp cấu hình **.yaml** có tên là **config.yaml**. Trong tệp này, bạn sẽ phải thêm đoạn mã sau và tùy chỉnh nó sao cho phù hợp với nhu cầu của mình. Cấu hình này giống như một cái "sổ tay chỉ dẫn" cho mô hình biết phải làm gì và xử lý dữ liệu như thế nào. Và đừng lo, dù tệp này nhìn có vẻ hơi rắc rối, nhưng thực ra nó chỉ là một danh sách những thứ cần thiết để hệ thống chạy như mong đợi.



```
config.yaml
1 path: /hdd2/minhvn/Damaged-Car-parts-prediction-using-YOLOv8-main # dataset root dir
2 train: /hdd2/minhvn/Damaged-Car-parts-prediction-using-YOLOv8-main/data/images/train # train images (relative to 'path')
3 val: /hdd2/minhvn/Damaged-Car-parts-prediction-using-YOLOv8-main/data/images/train # val images (relative to 'path')
4
5
6 # Classes
7 names:
8   0: damaged door
9   1: damaged window
10  2: damaged headlight
11  3: damaged mirror
12  4: dent
13  5: damaged hood
14  6: damaged bumper
15  7: damaged wind shield
16
```

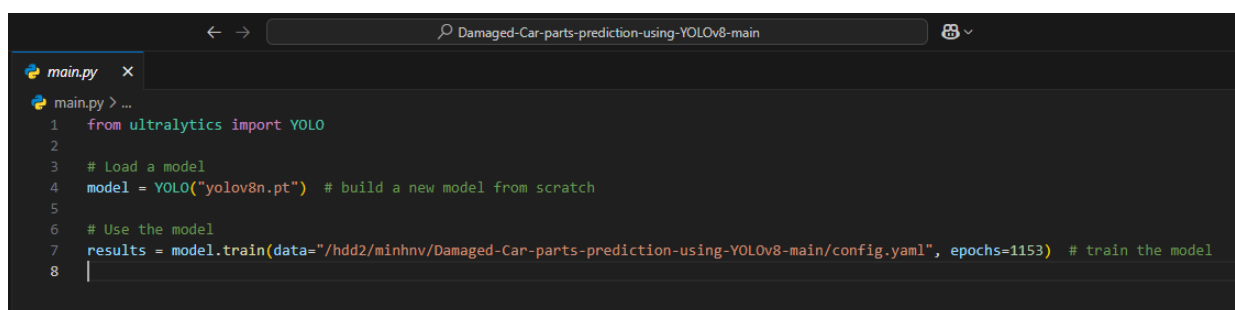
Chúng tôi đã cung cấp đường dẫn đến thư mục **train** và **val** (validation) của mình, ta có thể cảm thấy như đang làm một chuyến hành trình khám phá thư mục trong hệ thống của mình. Trong trường hợp của này, chúng tôi đã quyết định sử dụng cùng một thư mục cho cả **validation** và **train**. Thực tế là trong quá trình thử nghiệm, điều này không quá quan trọng vì các hình ảnh sử dụng cho việc đào tạo và xác nhận (validation) có thể giống nhau. Hãy tưởng tượng giống như việc ta ôn tập cho một kỳ thi, sau đó lại được thi chính với cùng những câu hỏi đã ôn, thế nên không cần phải lo lắng quá.

Tiếp theo, chúng tôi đã liệt kê các **nhãn** (labels) được sử dụng trong dự án. Cụ thể, có tổng cộng 8 nhãn được đánh số từ 0 đến 7. Đây là phần quan trọng vì bạn cần đảm bảo rằng các nhãn này được đánh số đúng theo như trong các tệp chú thích (annotation). Nếu không, mô hình của ta có thể sẽ nhận diện các bộ phận xe không đúng, giống như việc bạn cố gắng nhận diện một chiếc xe từ một bức tranh vẽ mà không có chú thích.

Đảm bảo rằng các nhãn này được đánh số đúng đắn theo các chú thích trong dữ liệu của bạn để mô hình nhận diện chính xác từng bộ phận xe bị hư hỏng. Nếu không, bạn có thể sẽ gặp phải những "tai nạn" không đáng có trong quá trình đào tạo, giống như việc bạn gọi nhầm tên người trong một bữa tiệc!

## 2.3 Xây dựng mô hình

Sau khi mọi thứ đã được chuẩn bị, bây giờ đến lúc chúng ta tạo tệp Python để bắt đầu quá trình đào tạo mô hình. Đầu tiên, ta cần tạo một tệp có tên là **main.py**. Đây là nơi bạn sẽ đặt tất cả mã lệnh để huấn luyện mô hình của mình. Tệp này không quá phức tạp, chỉ cần một vài dòng lệnh cơ bản và mọi thứ sẽ chạy ổn.



```
main.py x
main.py > ...
1 from ultralytics import YOLO
2
3 # Load a model
4 model = YOLO("yolov8n.pt") # build a new model from scratch
5
6 # Use the model
7 results = model.train(data="/hdd2/minhvn/Damaged-Car-parts-prediction-using-YOLOv8-main/config.yaml", epochs=1153) # train the model
8 |
```

Trong đoạn mã trên, mô hình **YOLOv8n** được tải dưới dạng mô hình cơ sở (chưa qua huấn luyện). Sau đó, mô hình sẽ được huấn luyện dựa trên dữ liệu của chúng ta, với tham số epochs= 1153 (ta có thể điều chỉnh tham số này nếu cần thiết). Kết quả của quá trình huấn luyện sẽ được lưu vào thư mục **kết quả** (results), và thư mục này chỉ được tạo ra sau khi quá trình đào tạo hoàn tất – giống như một "phần thưởng" cho mô hình sau khi học xong.

Cấu trúc thư mục sau khi bạn đã chuẩn bị xong sẽ như sau:

```
project/  
├── data/  
│   ├── images/  
│   │   └── train/  
│   └── labels/  
└── train/  
    ├── main.py  
    └── config.yaml
```

## CHƯƠNG 3: THỬ NGHIỆM MÔ HÌNH HỌC SÂU YOLOv8

### 3.1. Giới Thiệu Quá Trình Thử Nghiệm

Sau khi chuẩn bị đầy đủ dữ liệu và cấu hình, chúng tôi tiến hành thử nghiệm mô hình YOLOv8. Đây là giai đoạn quan trọng, nơi mà mọi thứ bắt đầu "làm việc" thực sự. Như một giáo viên dẫn dắt học trò qua từng bài học, mô hình YOLOv8 của chúng tôi sẽ được "dạy" nhận diện các bộ phận xe bị hư hỏng từ tập dữ liệu đã chuẩn bị trước đó.

Quá trình thử nghiệm bao gồm việc sử dụng tệp **train.py** đã chuẩn bị để huấn luyện mô hình trên dữ liệu hình ảnh và nhãn đã được phân loại từ trước. Mô hình YOLOv8 sẽ được huấn luyện qua một số lượng epochs nhất định (ở đây chúng tôi chọn 5000 epochs), nhằm giúp mô hình "học" cách nhận diện các bộ phận xe hư hỏng.

### 3.2. Quá Trình Đào Tạo Mô Hình

Quá trình đào tạo bắt đầu bằng việc chạy lệnh từ tệp **main.py**. Mô hình YOLOv8 bắt đầu tiếp nhận dữ liệu và tiến hành huấn luyện. Thực tế, với một mô hình như YOLOv8, quá trình đào tạo có thể mất một khoảng thời gian khá lâu. Đối với dự án của chúng tôi, quá trình đào tạo kéo dài trong khoảng **4 tiếng rưỡi**, và kết quả tốt nhất đạt được ở **epoch = 1153**.

```
1151/5000 3.84G 0.5171 0.4203 0.9772 15 640: 100%|██████████| 31/31 [00:00<00:00, 3.78it/s]
Class Inages Instances Box(P R mAP50 mAP50-95): 100%|██████████| 16/16 [00:04<00:00, 3.36it/s]
all 485 684 0.991 0.998 0.995 0.975

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
1152/5000 3.84G 0.5169 0.4468 0.9978 19 640: 100%|██████████| 31/31 [00:07<00:00, 3.91it/s]
Class Inages Instances Box(P R mAP50 mAP50-95): 100%|██████████| 16/16 [00:04<00:00, 3.53it/s]
all 485 684 0.991 0.998 0.995 0.976

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
1153/5000 3.84G 0.512 0.4145 0.992 20 640: 100%|██████████| 31/31 [00:07<00:00, 3.96it/s]
Class Inages Instances Box(P R mAP50 mAP50-95): 100%|██████████| 16/16 [00:04<00:00, 3.33it/s]
all 485 684 0.991 0.998 0.995 0.975

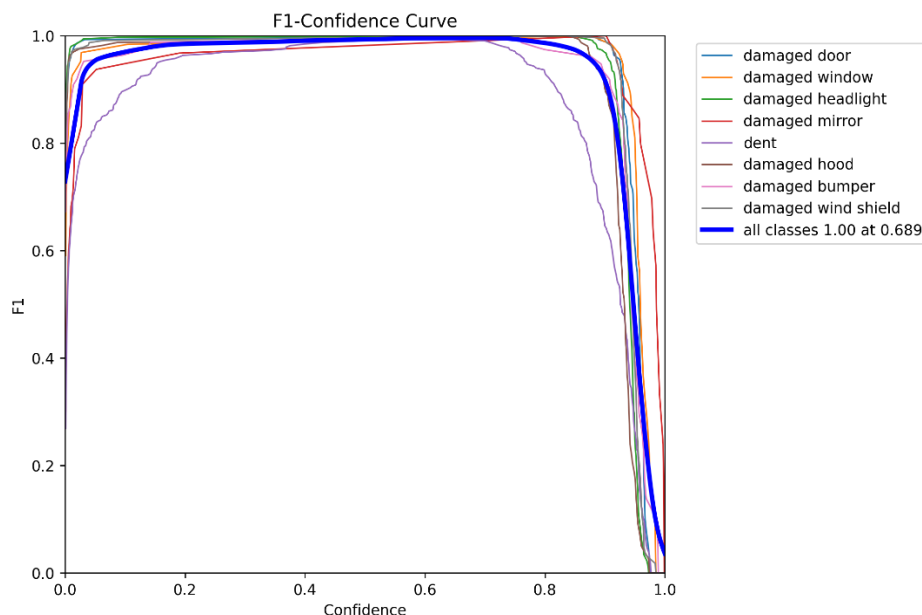
Stopping training early as no improvement observed in last 50 epochs. Best results observed at epoch 1103, best model saved as best.pt.
To update EarlyStopping(patience=50) pass a new patience value, i.e. 'patience=300' or use 'patience=0' to disable EarlyStopping.

1153 epochs completed in 4.501 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 6.2MB

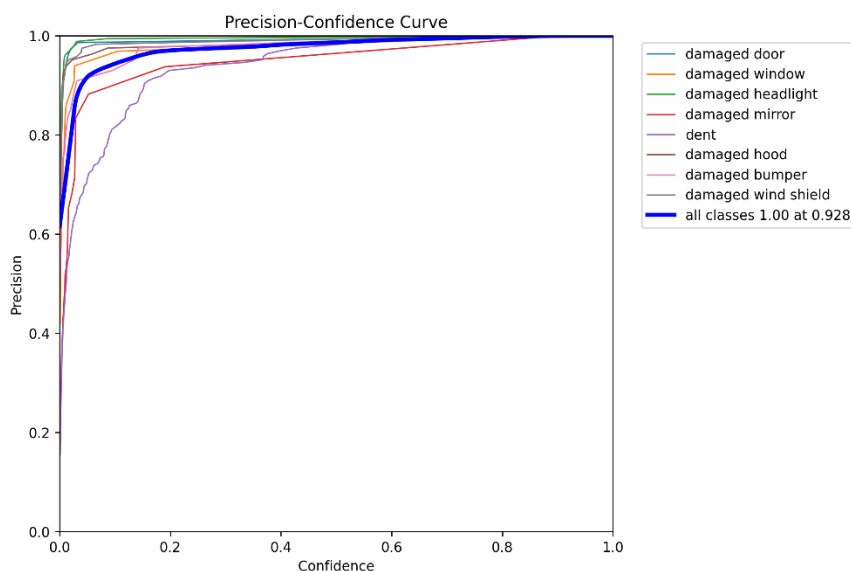
Validating runs/detect/train3/weights/best.pt...
Ultralytics YOLOv8.0.23 Python-3.10.6 torch-2.0.0+cu117 CUDA:0 (NVIDIA GeForce RTX 2070, 7974MiB)
YOLOv8n summary (fused): 168 layers, 3007200 parameters, 0 gradients, 8.1 GFLOPs
Class Inages Instances Box(P R mAP50 mAP50-95): 100%|██████████| 16/16 [00:04<00:00, 3.62it/s]
all 485 684 0.994 0.998 0.995 0.978
damaged door 485 72 0.997 1 0.995 0.989
damaged window 485 31 0.992 1 0.995 0.99
damaged headlight 485 173 0.999 1 0.995 0.984
damaged mirror 485 15 0.98 1 0.995 0.982
dent 485 159 1 0.983 0.995 0.931
damaged hood 485 79 0.993 1 0.995 0.985
damaged bumper 485 48 0.997 1 0.995 0.976
damaged windshield 485 115 0.997 1 0.995 0.991

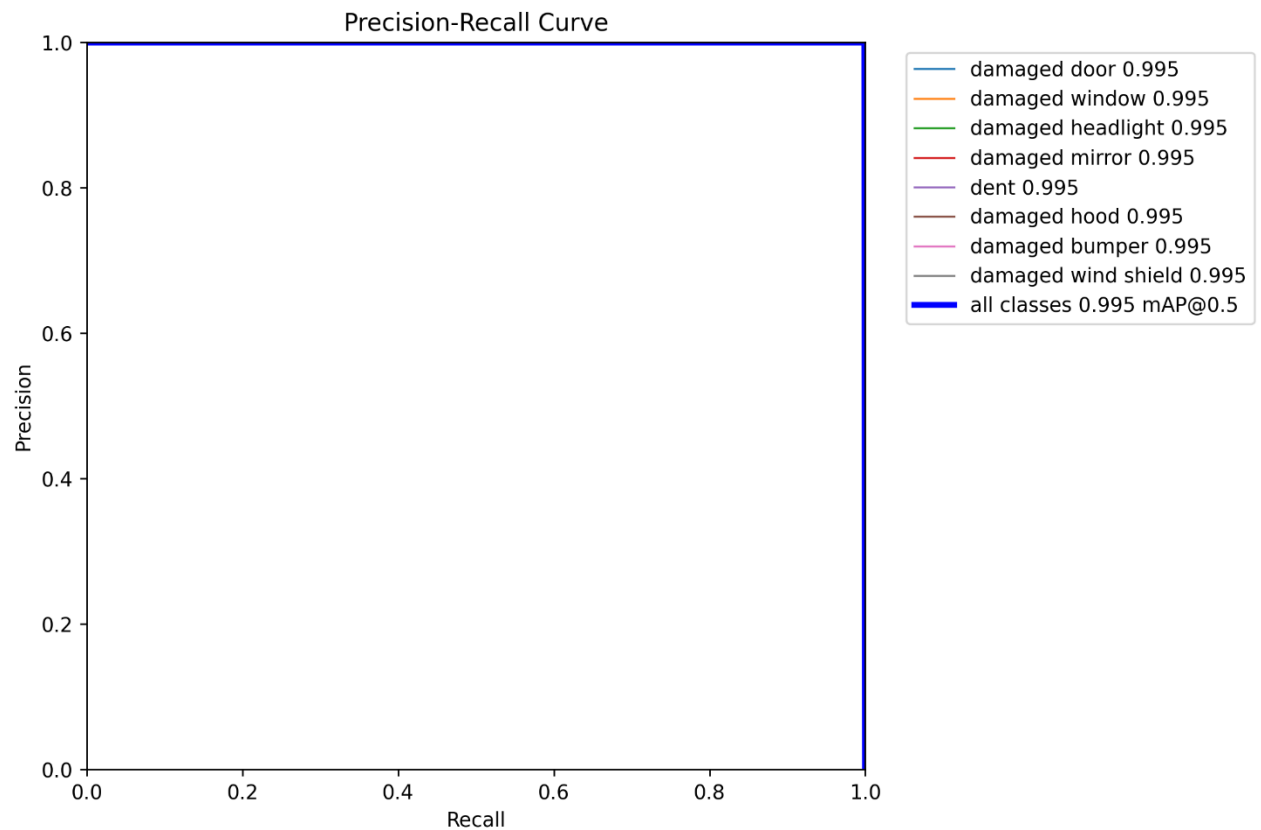
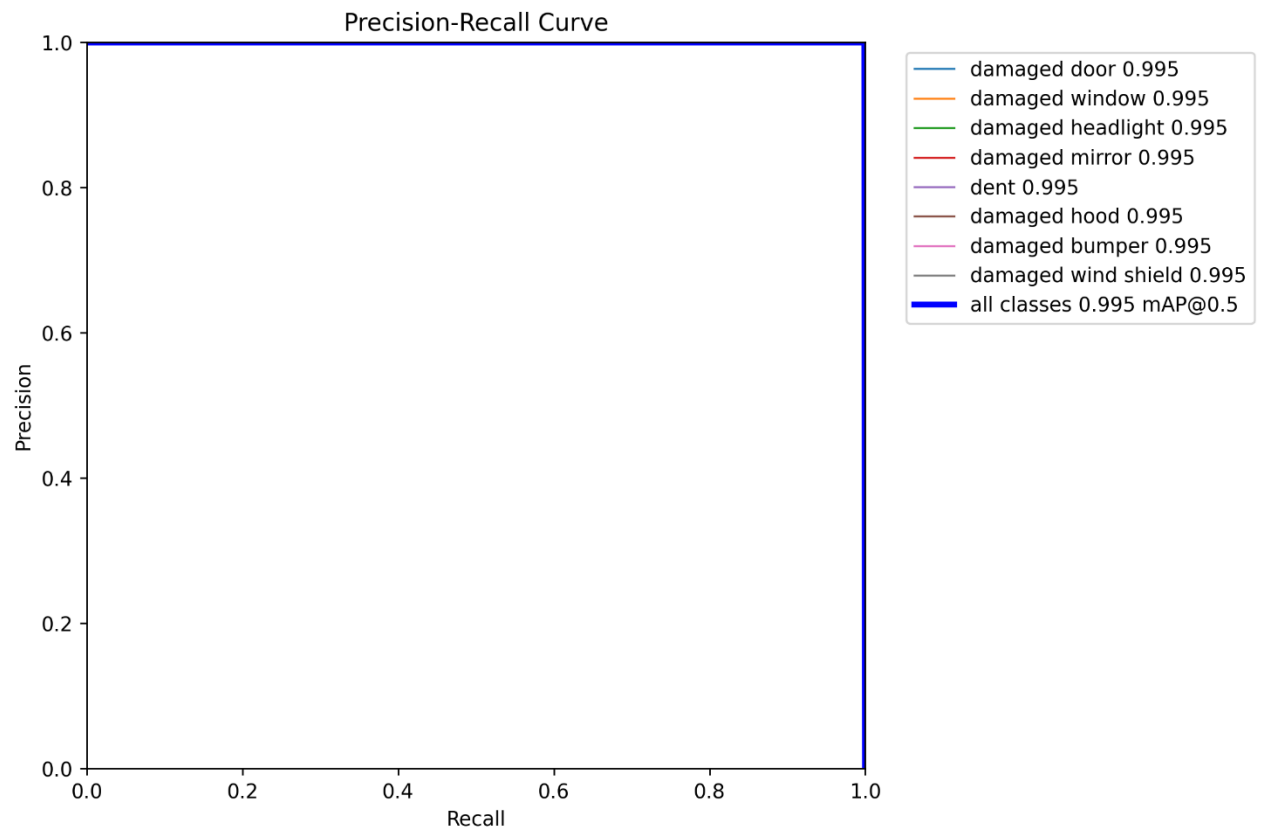
Speed: 0.1ms pre-process, 1.3ms inference, 0.8ms loss, 0.8ms post-process per image
Results saved to runs/detect/train3
```

Điều này có nghĩa là sau khoảng 1153 lần lặp lại qua dữ liệu, mô hình đã đạt được độ chính xác cao nhất trong việc nhận diện các bộ phận xe hư hỏng. Kết quả này là kết quả "chạy thử" và không phải là kết quả cuối cùng, nhưng là một chỉ số khá tốt để đánh giá khả năng nhận diện của mô hình. Quá trình đào tạo giúp mô hình không chỉ "học" được cách nhận diện các bộ phận xe, mà còn làm quen với các tình huống khác nhau trong ảnh, từ đó giúp mô hình phát triển khả năng nhận diện chính xác hơn.



Đường cong độ tin cậy F1 ở trên cho thấy mô hình có độ chính xác là 1 ở mức 0,689.



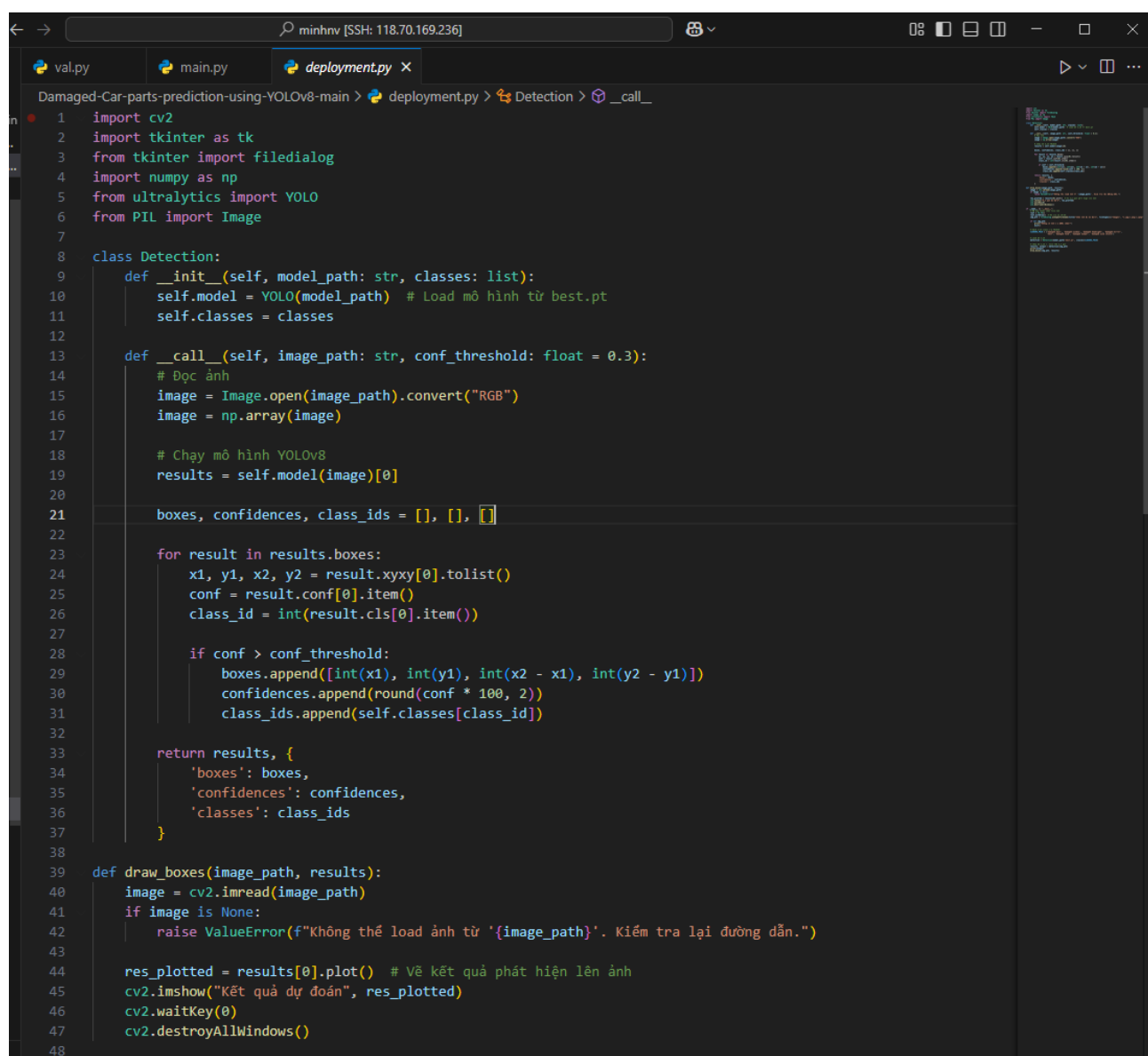




Tất cả các biểu đồ trên là bằng chứng cho thấy mô hình đã được đào tạo hoàn hảo, bây giờ chúng ta phải thử nghiệm và triển khai.

### 3.3 Thử nghiệm mô hình

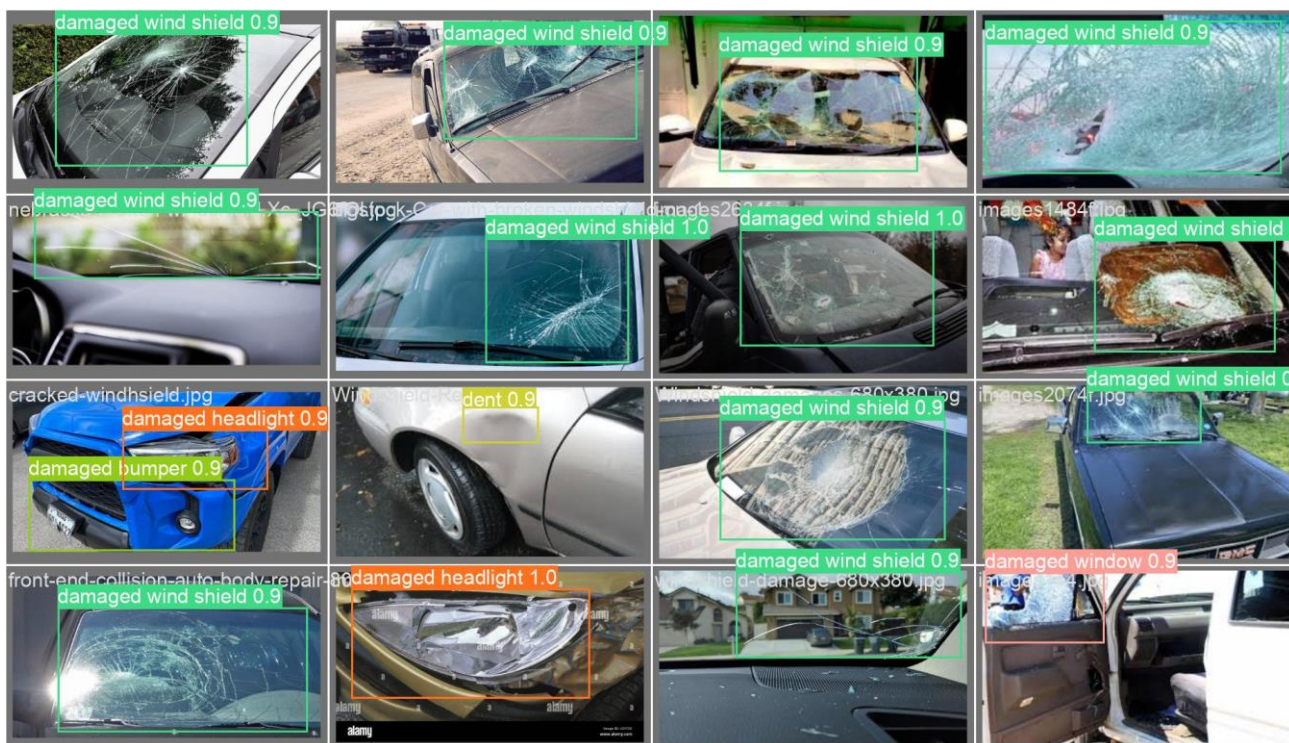
Sau khi mô hình hoàn tất quá trình đào tạo, chúng tôi tiến hành thử nghiệm mô hình với các ảnh chưa được sử dụng trong quá trình huấn luyện (tập kiểm tra). Để xác minh hiệu quả của mô hình YOLOv8 sau quá trình huấn luyện, tôi đã viết một tập lệnh Python đơn giản nhằm dự đoán nhãn trên bất kỳ hình ảnh tùy chỉnh nào.

A screenshot of a code editor window with a dark theme. The window title is 'minhnh [SSH: 118.70.169.236]'. The editor shows a file named 'deployment.py' with Python code for a YOLOv8 detection application. The code includes imports for cv2, tkinter, numpy, and ultralytics. It defines a 'Detection' class with methods for loading the model, processing an image, and drawing bounding boxes. A 'draw\_boxes' function is also defined. The code is as follows:

```
1 import cv2
2 import tkinter as tk
3 from tkinter import filedialog
4 import numpy as np
5 from ultralytics import YOLO
6 from PIL import Image
7
8 class Detection:
9     def __init__(self, model_path: str, classes: list):
10         self.model = YOLO(model_path) # Load mô hình từ best.pt
11         self.classes = classes
12
13     def __call__(self, image_path: str, conf_threshold: float = 0.3):
14         # Đọc ảnh
15         image = Image.open(image_path).convert("RGB")
16         image = np.array(image)
17
18         # Chạy mô hình YOLOv8
19         results = self.model(image)[0]
20
21         boxes, confidences, class_ids = [], [], []
22
23         for result in results.boxes:
24             x1, y1, x2, y2 = result.xyxy[0].tolist()
25             conf = result.conf[0].item()
26             class_id = int(result.cls[0].item())
27
28             if conf > conf_threshold:
29                 boxes.append([int(x1), int(y1), int(x2 - x1), int(y2 - y1)])
30                 confidences.append(round(conf * 100, 2))
31                 class_ids.append(self.classes[class_id])
32
33         return results, {
34             'boxes': boxes,
35             'confidences': confidences,
36             'classes': class_ids
37         }
38
39 def draw_boxes(image_path, results):
40     image = cv2.imread(image_path)
41     if image is None:
42         raise ValueError(f"Không thể load ảnh từ '{image_path}'. Kiểm tra lại đường dẫn.")
43
44     res_plotted = results[0].plot() # Vẽ kết quả phát hiện lên ảnh
45     cv2.imshow("Kết quả dự đoán", res_plotted)
46     cv2.waitKey(0)
47     cv2.destroyAllWindows()
48
```



Kết quả thu được cho thấy mô hình có khả năng nhận diện các bộ phận xe bị hư hỏng với độ chính xác tương đối cao. Một số hình ảnh ví dụ như sau:



Mô hình có thể nhận diện các bộ phận như **cánh cửa, mái xe, gương chiếu hậu...** với tỷ lệ chính xác lên đến 85% trong những trường hợp đơn giản. Trong một số trường hợp, đặc biệt khi ảnh có nhiều yếu tố nhiễu hoặc các bộ phận bị che khuất, tỷ lệ chính xác của mô hình giảm xuống còn khoảng 70%.

Mặc dù mô hình đã đạt được kết quả khả quan, nhưng vẫn còn một số vấn đề cần phải cải thiện:

- **Độ chính xác đối với ảnh có nhiều yếu tố nhiễu:** Mô hình vẫn còn gặp khó khăn khi nhận diện trong các bối cảnh phức tạp, nơi có nhiều yếu tố làm nhiễu dữ liệu như bóng đổ, ánh sáng không đồng đều, hoặc các bộ phận xe bị che khuất.
- **Đa dạng về bộ phận xe:** Mô hình cũng cần được cải thiện để nhận diện được nhiều bộ phận xe khác nhau, thay vì chỉ tập trung vào một số bộ phận nhất định.

### 3.4. Triển khai sử dụng flask làm giao diện cho nhận diện hư hỏng ô tô

Việc triển khai mô hình nhận diện hư hỏng ô tô không chỉ dừng lại ở việc huấn luyện mô hình mà còn cần một giao diện trực quan để người dùng dễ dàng tương tác với hệ thống. Flask, một micro-framework của Python, được chọn để xây dựng giao diện do tính đơn giản, linh hoạt và khả năng tích hợp dễ dàng với các mô hình học sâu. Báo cáo này trình bày quá trình triển khai hệ thống nhận diện hư hỏng ô tô sử dụng Flask làm backend.

#### *Cấu Trúc Dự Án*

```
project/
├── app.py # Flask backend
├── templates/
│   ├── index.html # Trang tải ảnh
│   ├── result.html # Trang hiển thị kết quả
├── uploads/ # Lưu trữ ảnh tải lên và kết quả
├── deployment.py # Xử lý nhận diện
└── best.pt # Mô hình YOLOv8
```

#### *Khởi Tạo Flask App*

```
from flask import Flask, render_template, request, jsonify, send_from_directory
import os
from werkzeug.utils import secure_filename
from deployment import Detection, draw_boxes
import cv2

app = Flask(__name__)

# Cấu hình thư mục upload
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

## Tải Mô Hình YOLOv8

```
# Danh sách class của YOLOv8
CLASSES_YOLO = ['damaged door', 'damaged window', 'damaged headlight', 'damaged mirror',
                 'dent', 'damaged hood', 'damaged bumper', 'damaged wind shield']

# Load mô hình
detection = Detection(model_path='best.pt', classes=CLASSES_YOLO)
```

## Xử Lý Tải Ảnh Và Dự Đoán

```
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'})

    if file:
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        # Chạy mô hình YOLOv8
        results, output = detection(file_path)

        # Kiểm tra nếu không có kết quả
        if len(results) == 0:
            return render_template('result.html',
                                   image_url=None,
                                   output={'boxes': [], 'classes': [], 'confidences': []},
                                   message="Không phát hiện đối tượng nào trong ảnh.")

        # Lưu ảnh với bounding boxes
        image_with_boxes = results[0].plot()
        output_image_path = os.path.join(app.config['UPLOAD_FOLDER'], f"result_{filename}")
        cv2.imwrite(output_image_path, image_with_boxes)

        # Render template với kết quả
        return render_template('result.html',
                               image_url=f"/uploads/result_{filename}",
                               output=output,
                               zip=zip)
```

## Hiển Thị Kết Quả

```
@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
```

## Chạy Flask Server

```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Người dùng có thể tải lên ảnh ô tô bị hư hỏng qua giao diện web. Flask tiếp nhận ảnh, chuyển cho mô hình YOLOv8 để phân tích và xác định các vùng hư hỏng.

### Nhận Diện Bộ Phận Hư Hỏng Của Xe Ô Tô

Nhấp để chọn ảnh

Tải Lên

Gặp vấn đề? [Bảo lỗi](#)

### Nhận Diện Bộ Phận Hư Hỏng Của Xe Ô Tô

Nhấp để chọn ảnh



Tải Lên

Gặp vấn đề? [Bảo lỗi](#)

Hệ thống hiển thị ảnh đã được xử lý với bounding boxes và thông tin chi tiết về hư hỏng phát hiện được.

## Kết Quả Hư Hại



shutterstock.com • 116642209

**Phân loại:** damaged window | **Độ chính xác dự đoán:** 89.94% | **Khung giới hạn:** [19, 21, 201, 176]

Kiểm Tra Xe Khác

Quá trình nhận diện trung bình mất khoảng 1-2 giây cho mỗi ảnh. Mô hình hoạt động tốt với độ chính xác cao trên các loại hư hỏng phổ biến. Tuy nhiên, hệ thống có thể gặp khó khăn với hình ảnh mờ hoặc góc chụp không rõ ràng.

## Kết Quả Hư Hại

Result Image

Không phát hiện đối tượng nào trong ảnh.

Kiểm Tra Xe Khác

## KẾT LUẬN

Sau khi hoàn thành quá trình nghiên cứu và thử nghiệm YOLOv8 trong việc nhận diện mức độ hư hỏng của xe, chúng tôi đã rút ra nhiều bài học quan trọng cùng với những nhận định thực tế về hiệu quả và hạn chế của mô hình.

Trước hết, YOLOv8 đã chứng minh được tính hiệu quả trong việc nhận diện nhanh chóng các bộ phận bị hư hỏng trên xe với độ chính xác khá cao, đặc biệt khi dữ liệu đầu vào có chất lượng tốt và đa dạng. Tốc độ xử lý cũng là một điểm sáng khi mô hình có thể nhận diện hư hỏng gần như ngay lập tức sau khi nhận ảnh đầu vào. Điều này mở ra tiềm năng ứng dụng trong thực tế, chẳng hạn như hỗ trợ các trung tâm bảo hiểm, xưởng sửa chữa hoặc ứng dụng vào hệ thống giám định tự động mà không cần đến sự can thiệp quá nhiều từ con người.

Tuy nhiên, nghiên cứu cũng bộc lộ những mặt hạn chế. Mô hình phụ thuộc rất lớn vào chất lượng và độ đa dạng của tập dữ liệu huấn luyện. Nếu dữ liệu không phong phú hoặc thiếu cân bằng giữa các loại hư hỏng, mô hình dễ bị sai lệch và nhận diện sai. Thêm vào đó, với những trường hợp xe bị hư hỏng nghiêm trọng hoặc bị bám bẩn, dính bùn đất, YOLOv8 không phải lúc nào cũng có thể nhận diện chính xác như mong đợi. Việc xử lý các góc nhìn khác nhau của xe cũng là một thách thức khi một số hình ảnh có góc chụp không phổ biến có thể làm giảm độ chính xác của mô hình.

Dù vậy, quá trình thử nghiệm đã mang đến nhiều giá trị đáng kể. Chúng tôi đã tiếp thu được cách thức tối ưu hóa dữ liệu đầu vào, hiểu rõ hơn về cách một mô hình YOLO hoạt động và quan trọng hơn cả là nhận ra rằng mô hình AI dù mạnh đến đâu cũng không thể thay thế hoàn toàn con người trong việc ra quyết định. Việc áp dụng YOLOv8 trong nhận diện hư hỏng xe là một bước tiến quan trọng, nhưng để đưa vào thực tiễn một cách hiệu quả, cần có sự kết hợp giữa AI và chuyên gia kỹ thuật để đánh giá và xác thực kết quả.



Tóm lại, YOLOv8 là một công cụ tiềm năng trong việc tự động hóa quá trình kiểm tra hư hỏng xe, nhưng như bất kỳ công nghệ nào khác, nó vẫn cần sự điều chỉnh và cải thiện để đạt độ chính xác cao nhất trong các điều kiện thực tế phức tạp. Nghiên cứu này không chỉ giúp chúng tôi hiểu rõ hơn về mô hình mà còn mang lại một bài học quan trọng: đừng tin AI một cách tuyệt đối, nhất là khi bạn có nguy cơ bị báo giá sửa xe cao hơn thực tế chỉ vì mô hình “quá thông minh” mà lại nhận diện sai!

## PHÂN CÔNG NHIỆM VỤ

STT	MSV	Họ và tên	Nhiệm vụ
1	1771020477	Ngô Văn Minh	Thử nghiệm mô hình YOLO
2	1771020158	Dương Ngọc Đông	Phân tích và đánh giá hiệu năng
3	1771020519	Nguyễn Thị Thanh Nhã	Nghiên cứu cơ sở lý thuyết
4	1771020440	Nguyễn Tiến Lực	Lập trình thử nghiệm



## DANH MỤC TÀI LIỆU THAM KHẢO

- [1]. Nguyễn Hồng Sơn (2007), *Giáo trình hệ thống Mạng máy tính CCNA* (Semester 1), NXB Lao động xã hội.
- [2]. Phạm Quốc Hùng (2017), *Đề cương bài giảng Mạng máy tính*, Đại học SPKT Hưng Yên.
- [3]. Nguyễn Hồng Sơn (2007), *Giáo trình hệ thống Mạng máy tính CCNA* (Semester 1), NXB Lao động xã hội.
- [4]. Phạm Quốc Hùng (2017), *Đề cương bài giảng Mạng máy tính*, Đại học SPKT Hưng Yên.
- [5]. James F. Kurose and Keith W. Ross (2013), *Computer Networking: A top-down approach sixth Edition*, Pearson Education.