BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH

TRƯƠNG ĐÌNH NHẬT CƯỜNG

XÂY DỰNG MÔ HÌNH DỰ ĐOÁN TÌNH TRẠNG HƯ HỎNG XE Ô TÔ SỬ DỤNG CNN

ĐỒ ÁN NGÀNH

NGÀNH CÔNG NGHỆ THÔNG TIN

TP. HỒ CHÍ MINH, 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH

TRƯƠNG ĐÌNH NHẬT CƯỜNG

XÂY DỰNG MÔ HÌNH DỰ ĐOÁN TÌNH TRẠNG HƯ HỎNG XE Ô TÔ SỬ DỤNG CNN

Mã số sinh viên: 2151050045

ĐỒ ÁN NGÀNH

NGÀNH CÔNG NGHỆ THÔNG TIN

Giảng viên hướng dẫn: TS. TRƯƠNG HOÀNG VINH

TP. HỒ CHÍ MINH, 2024

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn sâu sắc nhất đến thầy Trương Hoàng Vinh, người đã tận tình hướng dẫn và đồng hành cùng em trong suốt quá trình thực hiện đồ án ngành. Sự kiên nhẫn, tận tụy và những đóng góp quý báu của thầy đã giúp em vượt qua những khó khăn và hoàn thành đồ án này.

Em cũng xin bày tỏ lòng biết ơn chân thành đến tất cả các thầy cô trong Khoa Công Nghệ Thông Tin nói riêng và Trường Đại học Mở Thành phố Hồ Chí Minh nói chung. Những kiến thức mà thầy cô đã truyền đạt không chỉ giúp em hoàn thiện kỹ năng chuyên môn mà còn mang đến cho em nhiều bài học thực tiễn quý giá trong suốt quá trình học tập.

Bên cạnh đó, em xin gửi lời cảm ơn chân thành đến các bạn cùng lớp, những người đã luôn sát cánh, chia sẻ khó khăn và động viên em trên con đường học tập. Sự đồng hành của các bạn đã tiếp thêm động lực cho em hoàn thành tốt chặng đường này.

Cuối cùng, em xin kính chúc thầy cô và các bạn thật nhiều sức khỏe, hạnh phúc và thành công trong mọi lĩnh vực. Em trân trọng và biết ơn những kỷ niệm đẹp cùng thầy cô và bạn bè trong suốt quãng thời gian học tập vừa qua.

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TÓM TẮT ĐỒ ÁN NGÀNH

Trong thời đại công nghệ số, việc tự động phát hiện và phân tích hư hỏng xe ô tô từ hình ảnh đang trở thành một xu hướng thiết yếu trong ngành bảo hiểm, bảo dưỡng và kiểm định xe. Hiện nay, có nhiều mô hình tiên tiến đã giải quyết khá tốt trong việc phát hiện và phân vùng đối tượng, tiêu biểu là các mô hình như Fast R-CNN, Faster R-CNN và Mask R-CNN. Trong phạm vi đồ án, em tập trung nghiên cứu sâu về kiến trúc mạng Mask R-CNN – một trong những mô hình mạnh mẽ nhất hiện nay, nổi bật với khả năng không chỉ phát hiện đối tượng mà còn phân đoạn chi tiết vùng đối tượng trên hình ảnh. Đặc biệt, em sẽ nghiên cứu và áp dụng mô hình Mask R-CNN của Detectron2 – một một framework mã nguồn mở mạnh mẽ chuyên về các tác vụ thị giác máy tính, giúp mô hình có thể phát hiện và phân đoạn các đối tượng trên hình ảnh một cách hiệu quả. Nhằm hướng tới xây dựng một hệ thống có khả năng tự động nhận diện và phân vùng các vùng bị hư hỏng trên xe thông qua việc sử dụng mô hình học sâu Mask R-CNN, và trực quan hóa kết quả trên một nền tảng web hiện đại.

Đồ án ngành của em sẽ nghiên cứu và xây dựng một mô đun có khả năng phát hiện và phân đoạn vùng hư hỏng xe ô tô thông qua mô hình Mask R-CNN của Detectron2 và trực quan mô đun đạt được lên hệ thông web được viết bằng Flask Python. Giải pháp được triển khai bao gồm việc thu thập và xử lý bộ dữ liệu hình ảnh xe ô tô. Mô hình Mask R-CNN của Detectron2, được sử dụng để huấn luyện và tinh chỉnh, nhằm cải thiện độ chính xác trong việc dự đoán các vùng hư hỏng. Sau khi hoàn thiện mô hình, hệ thống được triển khai trên một nền tảng web sử dụng Flask Python, cho phép người dùng dễ dàng tải lên hình ảnh xe và nhận kết quả phân tích trực quan qua giao diện web.

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	**Từ đầy đủ**
AI	Artificial Intelligence
CNN	Convolutional Neural Networks
R-CNN	Region-based Convolutional Neural Networks
Mask R-CNN	Mask Region-based Convolutional Neural Networks
FPN	Feature Pyramid Network
COCO	Common Objects in Context
LVIS	Large Vocabulary Instance Segmentation
ReLU	Rectified Linear Unit
ROI	Region of Interest
VGG	Visual Geometry Group
SVM	Support Vector Machines
JSON	JavaScript Object Notation
FC	Fully Connected
RPN	Region Proposal Network
AP	Average Precision
APs	Average Precision for small objects
APm	Average Precision for medium objects
APl	Average Precision for large objects
bbox	bounding box
segm	segmentation mask
IoU	Intersection over Union

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG

MỞ ĐẦU

Trong những năm gần đây, sự phát triển nhanh chóng của công nghệ, trí tuệ nhân tạo (AI) và học sâu (deep learning) đang dần trở thành những công cụ đột phá trong nhiều lĩnh vực khác nhau, bao gồm cả ngành công nghiệp ô tô. Việc sử dụng các hệ thống tự động để phát hiện và đánh giá hư hỏng trên xe ô tô đang trở thành một hướng nghiên cứu và ứng dụng tiềm năng, giúp tiết kiệm thời gian, chi phí và tăng tính chính xác so với các phương pháp thủ công truyền thống.

Hiện nay, việc kiểm tra và đánh giá tình trạng hư hỏng của xe ô tô chủ yếu được thực hiện thủ công bởi các chuyên gia hoặc kỹ thuật viên, với những bước kiểm tra phức tạp và nhiều khả năng dẫn đến sai sót. Điều này không chỉ gây mất thời gian mà còn ảnh hưởng đến chi phí và hiệu quả của toàn bộ quy trình bảo dưỡng hoặc giám định xe. Vì vậy, nhu cầu phát triển một hệ thống tự động, có thể phân tích và dự đoán hư hỏng thông qua hình ảnh đã trở thành một yêu cầu cấp thiết, đặc biệt trong các ngành bảo hiểm, sửa chữa và dịch vụ ô tô.

Trên cơ sở đó, đề tài Phát hiện hư hỏng trên xe ô tô thông qua phân tích hình ảnh sử dụng Detectron2 được xây dựng với mục tiêu phát triển một hệ thống ứng dụng học sâu để nhận diện các vùng hư hỏng trên xe một cách nhanh chóng và chính xác. Hệ thống này không chỉ dựa vào mô hình Detectron2 để xử lý dữ liệu hình ảnh mà còn được triển khai trên nền tảng web thân thiện với người dùng bằng Flask Python. Qua đó, người dùng có thể dễ dàng tải lên hình ảnh xe, nhận kết quả phân tích chi tiết về các vùng hư hỏng ngay lập tức, góp phần tự động hóa quy trình kiểm tra và đánh giá tình trạng xe.

Chương 1. **TỔNG QUAN ĐỀ TÀI**

1.1. Tổng quan đề tài

Trong bối cảnh phát triển mạnh mẽ của công nghệ trí tuệ nhân tạo (AI), các mô hình học sâu đã và đang trở thành công cụ đắc lực trong việc giải quyết những vấn đề phức tạp liên quan đến nhận diện và phân tích hình ảnh. Một trong những ứng dụng tiềm năng của học sâu là khả năng phát hiện hư hỏng trên xe ô tô thông qua phân tích hình ảnh, hỗ trợ đáng kể trong các lĩnh vực như bảo hiểm xe, dịch vụ bảo trì và sửa chữa. Việc tự động hóa quá trình phát hiện hư hỏng không chỉ giúp tăng tính chính xác mà còn giảm bớt sự phụ thuộc vào con người, mang lại hiệu quả về mặt thời gian và chi phí. Nhằm khắc phục những hạn chế này, em đã nghiên cứu thu thập thông tin, dữ liệu để xây dựng một hệ thống có thể áp dụng mô hình học sâu để phát hiện và nhận diện hư hỏng từ hình ảnh nhằm mang lại sự tiện lợi tối đa cho người dùng trong việc kiểm tra và đánh giá tình trạng xe.

1.2. Lý do chọn đề tài

Việc đánh giá và phát hiện hư hỏng trên xe ô tô là một quy trình phức tạp và đòi hỏi nhiều thời gian. Trong những tình huống khẩn cấp, chẳng hạn như sau tai nạn hoặc kiểm tra bảo hiểm, việc chờ đợi các chuyên gia kỹ thuật đánh giá tình trạng xe có thể kéo dài, gây phiền toái cho người dùng và tăng chi phí vận hành cho doanh nghiệp. Hơn nữa, tính chính xác của quá trình kiểm tra phụ thuộc vào yếu tố con người, dễ dẫn đến sai sót hoặc chênh lệch trong kết quả đánh giá.

Trong bối cảnh đó, việc phát triển một hệ thống tự động hóa, sử dụng công nghệ học sâu để phân tích hình ảnh và phát hiện hư hỏng trên xe ô tô trở nên cần thiết hơn bao giờ hết. Trong quá trình nghiên cứu về đề tài này, em nhận thấy rằng Detectron2 là một công cụ mạnh mẽ trong việc nhận diện đối tượng và phân vùng, sẽ là nền tảng cốt lõi để giải quyết bài toán này. Việc áp dụng mô hình học sâu không chỉ giảm thiểu sai sót mà còn cải thiện tốc độ, độ chính xác và tính nhất quán trong quá trình kiểm tra xe. Đó cũng chính là lý do chọn đề tài này là nhằm ứng dụng công nghệ hiện đại vào giải quyết một vấn đề thực tiễn trong ngành công nghiệp ô tô.

1.3. Mục tiêu và phạm vi đề tài

1.3.1. Mục tiêu đề tài

Mục tiêu của đề tài là phát triển một hệ thống tự động hóa dựa trên công nghệ học sâu (deep learning) nhằm phát hiện hư hỏng trên xe ô tô thông qua phân tích hình ảnh. Cụ thể:

- **Phát triển mô hình Detectron2:** Huấn luyện mô hình Mask R-CNN trên bộ dữ liệu hình ảnh hư hỏng xe ô tô, nhằm phát hiện và khoanh vùng chính xác các khu vực bị hư hỏng.
- **Đảm bảo hiệu suất cao:** Tối ưu hóa mô hình để đạt được độ chính xác và tốc độ xử lý tốt, đảm bảo hệ thống có thể hoạt động hiệu quả trong môi trường thực tế.
- **Tích hợp hệ thống vào nền tảng web:** Xây dựng giao diện web thân thiện với người dùng bằng Flask Python, giúp người dùng dễ dàng tải lên hình ảnh và nhận kết quả phân tích hư hỏng trực quan.

1.3.2. Phạm vi đề tài

1.3.2.1. Con người và xã hội

Hệ thống ứng dụng tập trung vào các doanh nghiệp và cá nhân nhỏ có nhu cầu kiểm tra và đánh giá tình trạng xe thông qua hình ảnh để có thể nhận diện được các vùng hư hỏng trên xe một cách nhanh chóng và chính xác.

1.3.2.2. Công nghệ

Áp dụng các kỹ thuật hiện đại trong trí tuệ nhân tạo và học sâu, cụ thể là mô hình Mask R-CNN của Detectron2 và được triển khai trên nền tảng web với Flask Python. Cụ thể như sau:

- **Về mô hình học sâu (Deep Learning):** Sử dụng mô hình Mask R-CNN trong Detectron2 để phát hiện và khoanh vùng các hư hỏng trên xe ô tô thông qua hình ảnh. Quá trình này đòi hỏi việc xử lý dữ liệu, lựa chọn siêu tham số và huấn luyện mô hình để đạt hiệu suất tối ưu.
- **Về nền tảng web với Flask Python:** Hệ thống sẽ được triển khai trên nền tảng web sử dụng Flask Python, cho phép người dùng tải lên hình ảnh và nhận phản

hồi tức thì dưới dạng hình ảnh đã phân tích. Việc này đảm bảo tính tiện lợi, dễ dàng tích hợp và mở rộng trong các hệ thống thực tế.

1.4. Phương pháp thực hiện

Các giai đoạn thực hiện:

- **Phân tích và thu thập dữ liệu:** Dữ liệu hình ảnh xe ô tô hư hỏng sẽ được thu thập từ nhiều nguồn đáng tin cậy. Các hình ảnh sẽ được xử lý, gán nhãn và định dạng lại theo tiêu chuẩn COCO để phù hợp với mô hình Detectron2.
- **Huấn luyện mô hình:** Detectron2 với kiến trúc Mask R-CNN sẽ được sử dụng để huấn luyện mô hình phát hiện hư hỏng. Quá trình này bao gồm lựa chọn siêu tham số, tinh chỉnh mô hình và đánh giá trên tập kiểm thử.
- **Dự đoán kiểm tra và đánh giá mô hình:** Sau khi hoàn thành mô hình thì sẽ được kiểm thử trên các tập dữ liệu kiểm tra để đánh giá hiệu suất về độ chính xác và thời gian xử lý. Các kết quả sẽ được phân tích và tối ưu hóa nếu cần.
- **Xây dựng nền tảng web:** Flask Python sẽ được sử dụng để xây dựng giao diện web. Hệ thống sẽ cho phép người dùng tải hình ảnh từ máy tính cá nhân và hiển thị kết quả phân tích dưới dạng trực quan với các vùng hư hỏng được đánh dấu.

1.5. Bố cục báo cáo

Đồ án gồm các chương sau:

- Chương 1: Nêu tổng quan đề tài gồm: Giới thiệu đề tài, lý do chọn đề tài, mục tiêu và phạm vi đề tài, phương pháp thực hiện và bố cục báo cáo
- Chương 2: Cơ sở lý thuyết: Giới thiệu về mạng nơ ron tích chập CNN, mạng Mask R-CNN, framework Detectron2 và ứng dụng lên nền tảng web bằng Python Flask
- Chương 3: Triển khai mô hình bài toán: Giới thiệu mô hình được sử dụng trong nghiên cứu này, quá trình sử dụng và triển khai mô hình nhằm đưa ra kết quả tổng quan.
- Chương 4: Kết luận và hướng phát triển: Kết quả đã đạt được, những điểm còn hạn chế và hướng phát triển cho bài toán.

Chương 2. CƠ SỞ LÝ THUYẾT

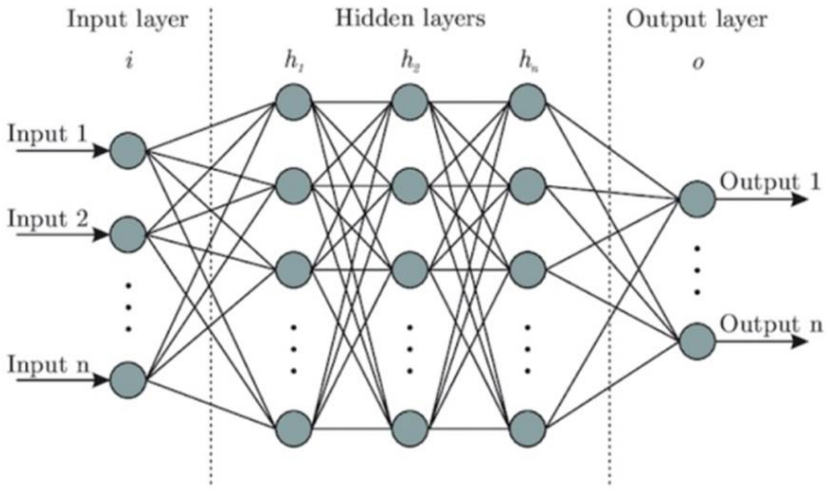
2.1. Giới thiệu chung về mạng nơ ron tích chập (Convolutional Neural Networks - CNN)

2.1.1. Mạng nơ ron nhân tạo (Neural Network)

Mạng nơ ron nhân tạo là một mô hình toán học mô phỏng cách thức hoạt động của bộ não con người. Nó được cấu tạo từ một tập các phần tử xử lý đơn giản được kết nối với nhau, gồm nhiều lớp đơn vị xử lý (neuron), trong đó mỗi neuron nhận thông tin từ các nguồn đầu vào, thực hiện một phép biến đổi và truyền kết quả đến các lớp tiếp theo.

Cấu trúc của mạng nơ ron gồm ba thành phần chính [1]:

- **Lớp đầu vào (Input Layer):** Đây là lớp nhận dữ liệu từ bên ngoài và truyền chúng vào mạng. Trong bài toán xử lý hình ảnh, lớp này thường nhận dữ liệu dưới dạng các pixel từ ảnh.
- **Lớp ẩn (Hidden Layers):** Các lớp này nằm giữa lớp đầu vào và lớp đầu ra, thực hiện quá trình tính toán và biến đổi tín hiệu. Mỗi lớp ẩn có thể gồm nhiều neuron, và mỗi neuron có thể áp dụng một hàm kích hoạt (activation function) để tính toán đầu ra của nó.
- **Lớp đầu ra (Output Layer):** Lớp này tạo ra kết quả cuối cùng của mạng, chẳng hạn như dự đoán về các lớp đối tượng trong ảnh hoặc vị trí của đối tượng cần nhận diện.



Hình 2-1: Cấu trúc mạng nơ ron tích chập

2.1.2. Mạng nơ ron tích chập CNN (Convolutional Neural Networks)

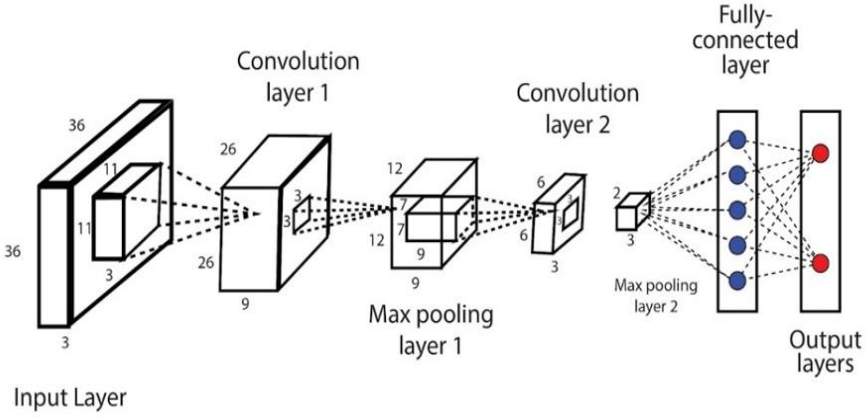
Mạng nơ ron tích chập (CNN) [2] là một mạng nơ ron nhân tạo chuyên dụng cho việc xử lý dữ liệu hình ảnh, dựa trên nguyên tác tích chập (convolution). CNN có khả năng học một lượng lớn các dữ liệu trong khoảng thời gian ngắn hơn so với mạng nơ ron truyền thống (Fully Connected Neural Networks) và trích xuất các đặc trưng quan trọng từ ảnh, giúp cho mô hình nhận diện các đối tượng với độ chính xác cao hơn.

Dưới đây là kiến trúc và cơ chế hoạt động của CNN:

2.1.2.1. Kiến trúc của CNN

- **Lớp Đầu Vào (Input Layer):** Đây là nơi hình ảnh đầu vào được cung cấp cho mạng. Hình ảnh thường là một ma trận 2D (cho ảnh đen trắng) hoặc ma trận 3D (cho ảnh màu với ba kênh RGB).
- **Lớp Tích Chập (Convolutional Layer):** Là cốt lõi của CNN, nơi các bộ lọc (filters) hay còn gọi là nhân tích chập (kernels) quét qua hình ảnh đầu vào để trích xuất các đặc trưng. Mỗi bộ lọc có kích thước cố định (ví dụ 3x3 hoặc 5x5 pixel) và di chuyển qua toàn bộ hình ảnh để tính toán tích chập, tạo ra một ma trận đặc trưng (feature map). Quá trình tích chập là phép toán học giữa bộ lọc và vùng của hình ảnh đầu vào. Kết quả của phép toán này là một giá trị duy nhất cho mỗi vị trí của bộ lọc, được lặp lại cho toàn bộ ảnh. Điều này giúp xác định các đặc điểm cụ thể như cạnh, góc, hoặc kết cấu.
- **Lớp kích hoạt (Activation Layer):** Các lớp kích hoạt được áp dụng để thêm phi tuyến tính vào mạng nơ ron, cho phép mạng học và mô phỏng các mối quan hệ phức tạp hơn. Ví dụ hàm kích hoạt phổ biến như hàm ReLU (Rectified Linear Unit) chuyển đổi các giá trị âm thành 0 và giữ nguyên giá trị dương, giúp tăng tốc độ huấn luyện và cải thiện hiệu suất của mạng.
- **Lớp Pooling (Pooling Layer):** Lớp này giảm kích thước không gian của các ma trận đặc trưng. Điều này giúp giảm số lượng tham số, tăng tốc độ tính toán và giảm nguy cơ overfitting (quá khớp). Các phương pháp pooling như:
  - **Max Pooling:** Chọn giá trị lớn nhất từ một vùng nhỏ của ma trận đặc trưng.
  - **Average Pooling:** Tính trung bình của các giá trị trong vùng nhỏ của ma trận đặc trưng.

- **Lớp Fully Connected (Fully Connected Layer):** Các lớp fully connected kết nối tất cả các neuron từ lớp trước đó với các neuron trong lớp hiện tại. Lớp này thường được sử dụng ở cuối mạng để kết hợp các đặc trưng đã học và đưa ra dự đoán cuối cùng. Trong bài toán phân loại, lớp fully connected có thể tạo ra đầu ra là xác suất cho từng lớp (class).
- **Lớp đầu ra (Output Layer):** Lớp này đưa ra kết quả dự đoán của mạng, có thể là một xác suất cho từng lớp trong bài toán phân loại hoặc các tọa độ cho bài toán phát hiện đối tượng.



Hình 2-2 Cấu trúc mạng nơ ron tích chập CNN [2]

2.1.2.2. Cơ chế hoạt động của CNN

Mạng nơ ron tích chập (CNN) hoạt động theo nguyên tắc chính là trích xuất và học các đặc trưng quan trọng từ hình ảnh đầu vào thông qua nhiều lớp khác nhau. Cơ chế hoạt động của CNN được tóm tắt theo các bước như sau:

Bước 1: Tiền xử lý dữ liệu

Trước khi đưa vào mô hình, hình ảnh thường được thay đổi kích thước để đồng nhất và chuẩn hóa để có thể đảm bảo được tính nhất quán và đôi khi áp dụng các kỹ thuật tăng cường dữ liệu (data augmentation) nhằm cải thiện khả năng học của mô hình.

Bước 2: Tích chập (Convolution)

Lớp tích chập sử dụng các bộ lọc (kernels) để trích xuất các đặc trưng từ ảnh đầu vào. Các bộ lọc này sẽ quét qua ảnh và thực hiện phép tính tích chập để

tạo ra ma trận đặc trưng (feature map). Kết quả là các đặc trưng cục bộ như cạnh, góc, và chi tiết nhỏ của ảnh được phát hiện.

Bước 3: Kích hoạt (Activation)

Sau mỗi lớp tích chập, hàm kích hoạt phi tuyến (thường là ReLU) được áp dụng để tăng độ phức tạp của mạng và giúp mạng học được các quan hệ phi tuyến.

Bước 4: Giảm kích thước (Pooling)

Lớp pooling (thường là max pooling) giúp giảm kích thước của ma trận đặc trưng, giữ lại các thông tin quan trọng và giảm số lượng tham số, đồng thời tăng khả năng khái quát hoá của mô hình.

Bước 5: Kết nối đầy đủ (Fully Connected Layer)

Sau khi dữ liệu đã được giảm kích thước qua các lớp tích chập và pooling, chúng sẽ được chuyển thành một vectơ 1D để đưa vào lớp fully connected. Lớp này sẽ đưa ra các dự đoán cuối cùng dựa trên những đặc trưng đã học được từ hình ảnh.

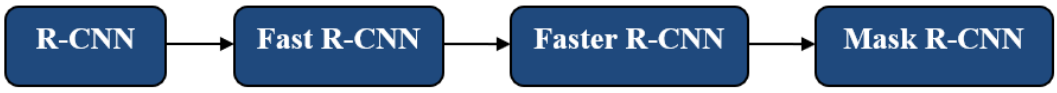
Bước 6: Lan truyền ngược (Backpropagation)

Sau khi mạng đưa ra kết quả dự đoán, thuật toán lan truyền ngược sẽ tính toán sai số giữa giá trị dự đoán và giá trị thực tế, từ đó điều chỉnh các trọng số của mạng nhằm giảm thiểu sai số.

2.1.3. Kiến trúc của CNN trong bài toán phát hiện và phân đoạn đối tượng

Ở nội dung trên em đã giải thích chi tiết về kiến trúc và cơ chế hoạt động của CNN. Trong phần này thì em sẽ trình bày nội dung kiến trúc cụ thể sử dụng trong bài toán phát hiện và phân đoạn đối tượng cho chủ đề đồ án này.

Với chủ đích của đồ án là sử dụng Detectron2 để giải quyết bài toán phát hiện và phân đoạn đối tượng (cụ thể là hư hỏng xe ô tô). Trong Detectron2 sử dụng một mô hình mạnh mẽ là Mask R-CNN để giải quyết bài toán trên. Và để hiểu được kiến trúc Mask R-CNN thì em sẽ đi theo thứ tự như sau:



2.1.3.1. R-CNN (Regions with CNN features)

R-CNN là một trong những mô hình tiên phong trong việc phát hiện đối tượng dựa trên hình ảnh bằng cách sử dụng các phương pháp học sâu. Được phát triển bởi Ross Girshick và các cộng sự vào năm 2014 [3], R-CNN đã mang lại bước tiến lớn trong lĩnh vực phát hiện đối tượng (object detection), đặc biệt khi kết hợp Mạng nơ ron tích chập (CNN) với các phương pháp truyền thống như Selective Search.

Cơ chế hoạt động của R-CNN

R-CNN hoạt động dựa trên việc chia hình ảnh thành nhiều vùng nhỏ và xử lý từng vùng để phát hiện các đối tượng có thể có trong mỗi vùng. Quy trình được thực hiện qua ba bước chính như sau:

1. **Region Proposal (Đề xuất vùng):**

Đầu tiên, R-CNN sử dụng một phương pháp truyền thống, cụ thể là **Selective Search**, để đề xuất các vùng quan tâm (Region of Interest - ROI) [3] trong ảnh. Selective Search là một thuật toán tìm kiếm không giám sát, giúp xác định các vùng có khả năng chứa đối tượng trong hình ảnh. Các đề xuất này không phân loại đối tượng, chỉ đơn thuần là tạo ra các vùng có thể chứa đối tượng.

Sau khi chạy Selective Search, hàng ngàn đề xuất vùng (thường là khoảng 2000) được tạo ra.

2. **Feature Extraction (Trích xuất đặc trưng):**

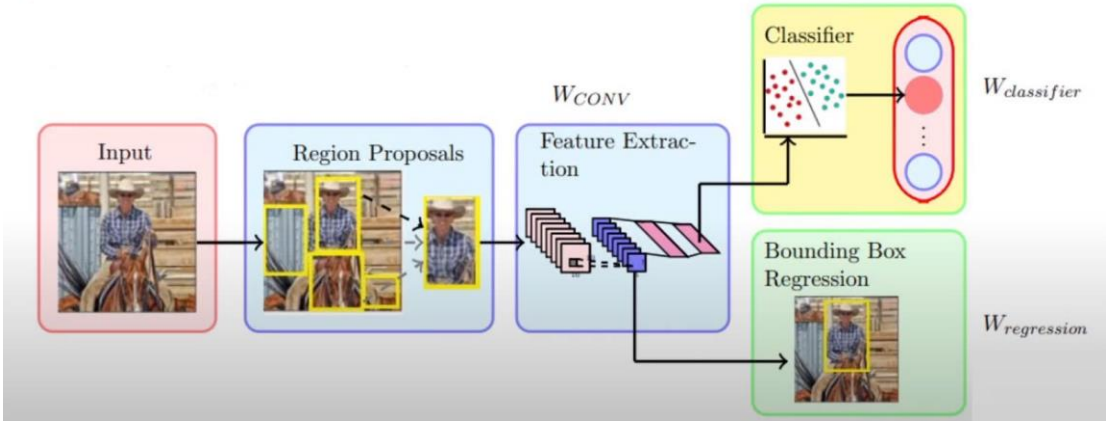
Với mỗi vùng đề xuất (region proposals), R-CNN thực hiện một bước trích xuất đặc trưng bằng cách sử dụng mạng nơ ron tích chập (CNN) đã được huấn luyện, ví dụ như mô hình AlexNet, VGG-16, đã được huấn luyện trước trên các tập dữ liệu lớn như ImageNet.

Các vùng đề xuất này được biến đổi thành cùng một kích thước trước khi đưa qua mạng CNN, do đó mỗi vùng sẽ được ánh xạ thành một vector đặc trưng cố định.

3. Classification (Phân loại):

Sau khi trích xuất đặc trưng, mỗi vector đặc trưng từ vùng đề xuất sẽ được đưa vào một bộ phân loại (thường là một SVM - **Support Vector Machine**). Bộ phân loại này sẽ dự đoán xem vùng đề xuất đó có chứa đối tượng thuộc một lớp cụ thể (ví dụ: xe, người, động vật, v.v.) hay không, và nếu có thì đối tượng thuộc lớp nào

Ngoài ra, một hồi quy tuyến tính được sử dụng để dự đoán lại hộp giới hạn (bounding box regression), cải thiện độ chính xác trong việc xác định vị trí đối tượng trong ảnh. (Tinh chỉnh bouding box)



Hình 2-3: Cấu trúc mạng R-CNN

Ưu điểm của R-CNN là có tính chính xác cao và hiệu quả với các hình ảnh phức tạp. Nhưng R-CNN phải xử lý hàng ngàn vùng đề xuất riêng lẻ và mỗi vùng cần phải chạy qua CNN, điều này làm cho mô hình cực kỳ chậm và tốn nhiều tài nguyên tính toán.

2.1.3.2. Fast R-CNN

Fast R-CNN ra đời nhằm giải quyết những hạn chế về tốc độ và hiệu suất của R-CNN, mang lại sự cải tiến đáng kể trong việc phát hiện đối tượng trong ảnh. Được giới thiệu bởi Ross Girshick vào năm 2015, Fast R-CNN đã đổi mới quá trình phân tích hình ảnh bằng cách trích xuất đặc trưng từ toàn bộ ảnh chỉ một lần duy nhất, thay vì xử lý từng vùng đề xuất (region proposals) như trong R-CNN. Điều này giúp giảm đáng kể thời gian tính toán và cải thiện hiệu suất mô hình.

Cơ chế hoạt động của Fast R-CNN

1. **Trích xuất đặc trưng từ toàn bộ hình ảnh**:

Thay vì trích xuất đặc trưng cho từng vùng đề xuất như trong R-CNN, Fast R-CNN chỉ trích xuất đặc trưng một lần duy nhất cho toàn bộ ảnh đầu vào bằng cách sử dụng một mạng CNN (ví dụ như VGG16, ResNet). Mạng CNN này nhận toàn bộ ảnh và sinh ra một bản đồ đặc trưng (feature map) duy nhất đại diện cho tất cả các thông tin quan trọng trong ảnh.

Sau khi có bản đồ đặc trưng từ toàn bộ ảnh, Fast R-CNN sử dụng một thuật toán sinh vùng đề xuất (như Selective Search) để tạo ra các vùng đề xuất có khả năng chứa đối tượng.

2. **RoI Pooling (Region of Interest Pooling)**:

Sau khi trích xuất đặc trưng của toàn bộ ảnh, Fast R-CNN sử dụng RoI Pooling để lấy các đặc trưng của từng vùng đề xuất từ bản đồ đặc trưng và chia chúng về kích thước cố định, giúp quá trình xử lý các vùng đề xuất trở nên nhất quán và dễ dàng hơn.

RoI Pooling là quá trình ánh xạ các vùng đề xuất lên các đặc trưng đã được tính toán trước đó, giúp tăng tốc độ xử lý rất nhiều.

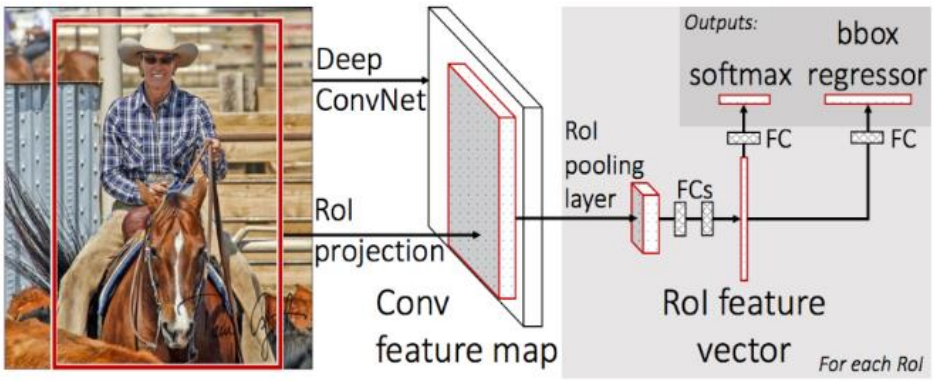
3. **Phân loại đối tượng và điều chỉnh hộp giới hạn (Bounding Box Regression)**:

Sau khi RoI Pooling, các đặc trưng của từng vùng đề xuất được đưa qua một mạng **fully connected** (FC) để tiếp tục xử lý. Mạng FC này học các đặc trưng phức tạp hơn và thực hiện hai nhiệm vụ song song là: Phân loại đối tượng trong từng vùng đề xuất và Tinh chỉnh bounding box.

Fast R-CNN có hai đầu ra từ mạng FC:

- Phân loại đối tượng: Đầu ra này bao gồm một danh sách các lớp (classes) cho mỗi vùng đề xuất.
- Bounding Box Regression: Đầu ra này là một bộ tọa độ điều chỉnh (gồm 4 thông số) dùng để tinh chỉnh vị trí và kích thước của khung giới hạn quanh đối tượng trong vùng đề xuất.

Cấu trúc cụ thể của Fast R-CNN được mô tả trong hình 2-4 dưới đây:



Hình 2-4: Cấu trúc mô hình Fast R-CNN [4]

Ưu điểm của Fast R-CNN là đã cải thiện tốc độ, hiệu quả tính toán cao hơn, độ chính xác cao hơn so với R-CNN. Nhưng Fast R-CNN vẫn còn phải phụ thuộc vào Selective Search để đề xuất vùng quan tâm, gây ảnh hưởng đến tốc độ. Mặc dù đã nhanh hơn R-CNN, nhưng Selective Search vẫn là bước tính toán tốn kém và không tối ưu.

2.1.3.3. Faster R-CNN

Faster R-CNN là một cải tiến tiếp theo của Fast R-CNN, được giới thiệu vào năm 2015 bởi Shaoqing Ren và các cộng sự. Thay vì sử dụng phương pháp Selective Search, Faster R-CNN giới thiệu Region Proposal Network (RPN), một mạng nơ ron tích chập riêng biệt để tự động đề xuất các vùng đối tượng, giúp cải thiện tốc độ và độ chính xác đáng kể.

Cơ chế hoạt động của Faster R-CNN

1. **Region Proposal Network (RPN)**:

RPN là một mạng nơ-ron tích chập (CNN) được thiết kế để tự động sinh ra các đề xuất vùng (RoIs) thay vì dựa vào các phương pháp đề xuất vùng thủ công như Selective Search. Cơ chế hoạt động của RPN bao gồm:

- **Trích xuất đặc trưng từ ảnh đầu vào**: Ảnh được đưa qua mạng CNN để tạo ra bản đồ đặc trưng. Từ đây, RPN sẽ lấy những đặc trưng này để phân tích.
- **Tạo ra các anchor boxes**: Tại mỗi vị trí trong ảnh, RPN tạo ra một tập hợp các anchor boxes với kích thước và tỷ lệ khác nhau. Những anchor boxes này hoạt động như các hộp giả định chứa đối tượng trong ảnh.

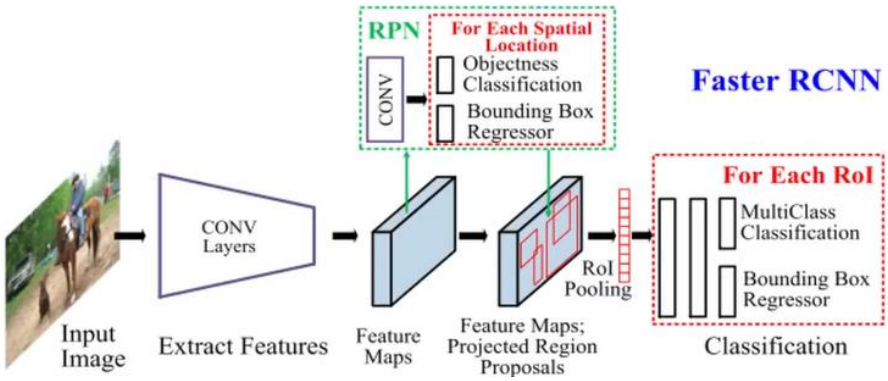
- **Phân loại và lọc anchor boxes**: Mỗi anchor box sẽ được phân loại là chứa đối tượng hoặc không chứa đối tượng. Những anchor có khả năng chứa đối tượng cao sẽ được giữ lại làm đề xuất vùng (RoIs) để tiếp tục xử lý.

2. **RoI Pooling và Phân loại đối tượng**:

Các vùng đề xuất (RoIs) từ RPN sau đó được chuyển qua lớp RoI Pooling, giúp chuẩn hóa kích thước của từng vùng để có thể đưa qua mạng CNN mà vẫn giữ nguyên tính nhất quán.

Sau khi các đặc trưng vùng đã được trích xuất, chúng sẽ đi qua các lớp fully connected (FC) để thực hiện hai nhiệm vụ chính:

- **Phân loại đối tượng**: Mỗi vùng được dự đoán thuộc về lớp đối tượng nào.
- **Tinh chỉnh bounding box**: Điều chỉnh kích thước và vị trí của hộp giới hạn (Bounding Box Regression) để phù hợp chính xác hơn với đối tượng được phát hiện.



Hình 2-5: Cấu trúc mô hình Faster R-CNN [5]

Ưu điểm của Faster R-CNN là nhanh hơn nhiều so với Fast R-CNN vì đã loại bỏ Selective Search và thay thế bằng RPN – một quá trình tự động và tích hợp trực tiếp trong mô hình. Học đặc trưng tốt hơn vì RPN giúp học các đặc trưng và đề xuất các vùng quan tâm tối ưu hơn, nhờ đó tăng cường hiệu suất phát hiện đối tượng. Nhưng bên cạnh đó Faster R-CNN lại đòi hỏi tài nguyên tính toán lớn vì yêu cầu lượng lớn tài nguyên GPU để xử lý, do cả RPN và mạng phân loại đều sử dụng CNN

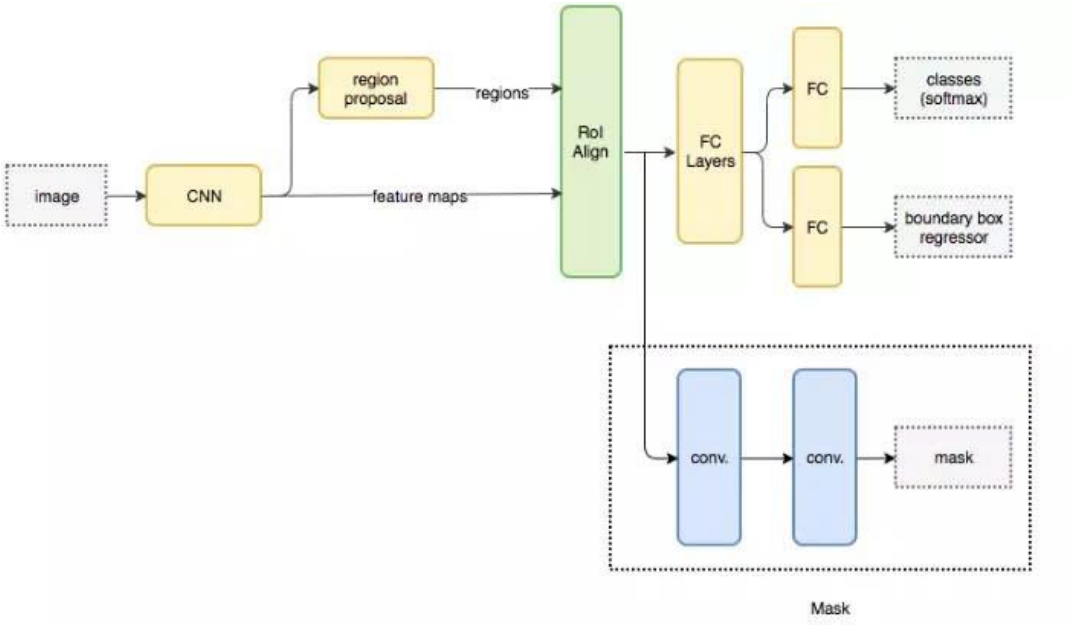
2.2. Mạng Mask R-CNN

Mask R-CNN là một phiên bản mở rộng của Faster R-CNN, được giới thiệu bởi Kaiming He và các cộng sự vào năm 2017 [6]. Mô hình này không chỉ phát hiện các đối tượng trong hình ảnh (như Faster R-CNN) mà còn thực hiện phân đoạn đối tượng (**instance segmentation**), tức là xác định chính xác vùng biên của mỗi đối tượng thay vì chỉ bao quanh chúng bằng các hộp giới hạn (bounding box).

Mask R-CNN là một khuôn khổ gồm hai giai đoạn: Giai đoạn đầu tiên là quét hình ảnh và tạo ra các đề xuất (proposals). Giai đoạn thứ hai là phân loại các đề xuất và tạo ra các hợp giới hạn (bounding boxes) và các mặt nạ (masks)

Cấu trúc của Mask R-CNN

Cấu trúc Mask R-CNN là một phần mở rộng của Faster R-CNN, với một nhánh bổ sung cho việc dự đoán các mask (mặt nạ) nhị phân cho từng đối tượng.



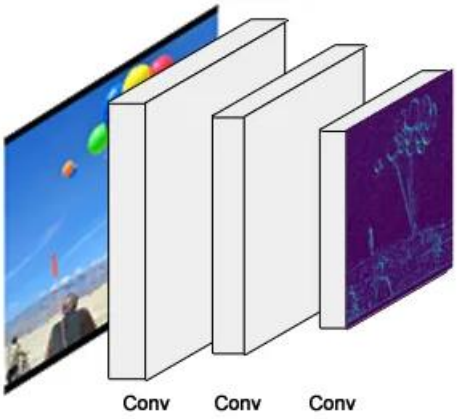
Hình 2-6: Cấu trúc mô hình Mask R-CNN [6]

Mask R-CNN bao gồm các thành phần chính sau:

2.2.1. Backbone

Backbone đóng vai trò là một mạng trích xuất đặc trưng tổng quan từ hình ảnh, thường là một mạng nơ rơn tích chập (CNN) như ResNet, VGG. Backbone không thực hiện trực tiếp các tác vụ như phân loại hay phân đoạn, mà thay vào đó nó mã hóa

hình ảnh thành một bản đồ đặc trưng (feature map) chứa các thông tin có giá trị giúp mô hình xử lý các bước tiếp theo.



Hình 2-7: Minh họa về mạng backbone

Trong Mask R-CNN sử dụng một kiến trúc **backbone** để trích xuất đặc trưng từ ảnh đầu vào. Hai kiến trúc phổ biến là ResNet-50 và ResNet-101 [7]:

- **ResNet-50**: Gồm 50 lớp, là một phiên bản nhẹ hơn của ResNet, cân bằng tốt giữa tốc độ và độ chính xác. Đây là kiến trúc thông dụng trong các ứng dụng thời gian thực khi cần hiệu suất cao nhưng yêu cầu xử lý nhanh.
- **ResNet-101**: Có 101 lớp, sâu hơn ResNet-50, cung cấp khả năng trích xuất đặc trưng mạnh hơn, phù hợp với các bài toán đòi hỏi độ chính xác cao hơn, nhưng bù lại tốc độ xử lý sẽ chậm hơn.

Trong Mask R-CNN, Backbone thường được sử dụng kết hợp với một kiến trúc mạng phức hợp khác như **FPN (Feature Pyramid Network)** để tạo ra các bản đồ đặc trưng với độ phân giải khác nhau, giúp phát hiện các đối tượng ở nhiều kích thước khác nhau

Feature Pyramid Network (FPN):

FPN được thêm vào để cải thiện khả năng phát hiện đối tượng ở các kích thước khác nhau [8]. FPN trích xuất các đặc trưng từ nhiều mức độ phân giải khác nhau trong mạng CNN, từ đó tạo ra các đặc trưng tổng hợp từ các lớp khác nhau. Điều này đặc biệt quan trọng với các đối tượng nhỏ, giúp cải thiện độ chính xác trong việc phát hiện và phân đoạn các đối tượng ở nhiều kích thước.

2.2.2. Region Proposal Network (RPN)

Region Proposal Network (RPN) là một phần quan trọng trong kiến trúc của Mask R-CNN, đóng vai trò quyết định trong việc tự động phát hiện các vùng có khả năng chứa đối tượng (Region of Interest - RoI) từ hình ảnh đầu vào. Cơ chế hoạt động của RPN chi tiết như sau:

- **Trích xuất đặc trưng từ backbone:** Backbone (ví dụ: ResNet kết hợp với FPN) trích xuất đặc trưng từ ảnh đầu vào và tạo ra bản đồ đặc trưng (feature map). Bản đồ này sẽ được sử dụng để tạo ra các vùng đề xuất.
- **Dự đoán anchor boxes**: Tại mỗi điểm trên bản đồ đặc trưng, RPN tạo ra một tập hợp các anchor boxes, tức là các hộp với kích thước và tỉ lệ khác nhau (ví dụ: 3 tỉ lệ và 3 kích thước, tổng cộng 9 anchor boxes cho mỗi điểm). Các anchor này đại diện cho các vùng khả dĩ chứa đối tượng.
- **Phân loại vùng**: RPN sử dụng các lớp tích chập để phân loại các anchor box thành hai loại: "có đối tượng" và "không có đối tượng". Đây là một bước quan trọng để loại bỏ những vùng không quan trọng, giúp giảm thiểu số lượng vùng cần xử lý ở bước sau
- **Tinh chỉnh bounding boxes**: Đồng thời, RPN cũng thực hiện hồi quy (regression) để tinh chỉnh vị trí và kích thước của các anchor boxes, nhằm dự đoán vùng chứa đối tượng chính xác hơn. Những hộp này sẽ được điều chỉnh dựa trên sự tương đồng với các đối tượng thực tế trong ảnh.
- **Non-Maximum Suppression (NMS)**: Sau khi có danh sách các hộp được phân loại là "có đối tượng", RPN áp dụng kỹ thuật NMS để loại bỏ những hộp chồng lấn quá nhiều và giữ lại những hộp tốt nhất, tránh trùng lặp. Kết quả là một tập hợp nhỏ các RoIs chất lượng cao.

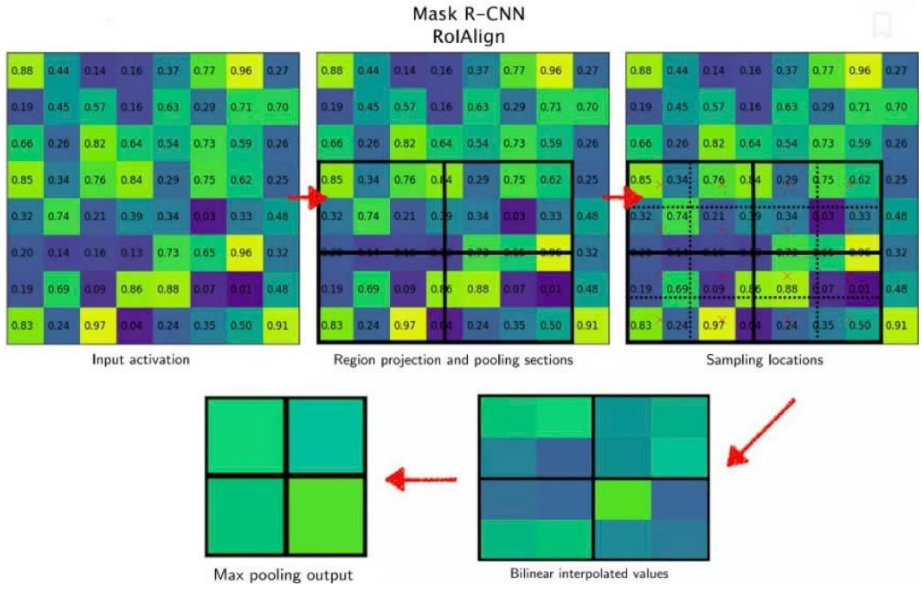
2.2.3. RoI Align

Một cải tiến quan trọng của Mask R-CNN so với Faster R-CNN là việc thay thế RoI Pooling bằng RoI Align [9]. RoI Align giúp khắc phục vấn đề làm tròn các tọa độ trong RoI Pooling:

- RoI Align giữ nguyên các tọa độ không gian của đối tượng, cho phép mô hình dự đoán chính xác hơn các đặc trưng không gian của đối tượng.

- Điều này đặc biệt quan trọng đối với các bài toán phân đoạn, nơi cần phân chia chi tiết từng pixel của đối tượng.

Cụ thể như sau: Thay vì lượng tử hóa vị trí các điểm đặc trưng như RoI Pooling, RoI Align sử dụng nội suy bilinear (nội suy hai chiều) để tính toán giá trị tại các vị trí chính xác hơn trên đặc trưng đầu vào. Phương pháp này tránh mất mát thông tin do làm tròn tọa độ, cung cấp kết quả phân vùng với độ chính xác cao hơn. Hình minh họa 2-8 dưới đây cho thấy quá trình từ ma trận đặc trưng đầu vào, các vùng được chia nhỏ và nội suy để lấy giá trị tại các vị trí mẫu.



Hình 2-8: Mô tả phương pháp RoIAlign [9]

2.2.4. Phân loại và Hồi quy (Classifier & Bounding Box Regressor)

Sau khi RoI Align thực hiện trích xuất đặc trưng từ các vùng đề xuất, các đặc trưng này được đưa vào hai nhánh: [10]

- **Classifier (Phân loại)**: Nhánh này dự đoán nhãn của đối tượng trong từng vùng đề xuất (RoI). Mục tiêu là xác định loại đối tượng.
- **Bounding Box Regressor (Hồi quy hộp giới hạn)**: Nhánh này tinh chỉnh tọa độ của các hộp giới hạn để dự đoán chính xác vị trí của đối tượng. Đây là bước quan trọng giúp cải thiện độ chính xác của các hộp giới hạn, đảm bảo rằng chúng ôm sát đối tượng một cách chính xác.

Hai nhánh này làm việc song song để cung cấp cả thông tin về loại đối tượng và vị trí của chúng.

2.2.5. Segmentation Masks

Phần bổ sung của Mask R-CNN so với Faster R-CNN là nhánh Segmentation Mask, được thiết kế để dự đoán các mặt nạ phân đoạn nhị phân cho từng đối tượng.

- **Input**: Các đặc trưng từ RoI Align được đưa vào một mạng con gồm nhiều lớp tích chập (convolutional layers) để tạo ra mặt nạ cho đối tượng.
- **Output**: Kết quả là một mặt nạ nhị phân (binary mask) với kích thước đã định sẵn, xác định pixel nào thuộc về đối tượng và pixel nào không. Mặt nạ này chỉ được tạo cho các đối tượng đã được phân loại.

Hình 2-9: Ví dụ về dự đoán Mask cho tất cả các đối tượng

2.3. Detectron2

2.3.1. Giới thiệu về Detectron2

Detectron2 là một framework mã nguồn mở, một mô hình học sâu dược xây dựng trên nền tảng Pytorch được phát triển bởi nhóm nghiên cứu AI của Facebook (FAIR – Facebook AI Research) [11]. Được thiết kế để thực hiện các tác vụ về thị giác máy tính, bao gồm các bài toán về phân đoạn đối tượng, nhận diện đối tượng, phát hiện đối tượng và nhiều ứng dụng khác. Detectron2 là sự kế thừa của phiên bản Detectron ban đầu, với nhiều cải tiến đáng kể về hiệu suất, khả năng mở rộng và dễ sử dụng.

Detectron2 là một mô hình học sâu được xây dựng trên nền tảng Pytorch và cung cấp nhiều tùy chọn kiến trúc backbone, bao gồm ResNet, ResNeXt và MobileNet. Ba thành phần chính trong kiến trúc của Detectron2 bao gồm:

- **Backbone Network:** Backbone trong Detectron2 có nhiệm vụ trích xuất các bản đồ đặc trưng (feature maps) ở các cấp độ khác nhau từ hình ảnh đầu vào.
- **Regional Proposal Network (RPN):** Từ các đặc trưng đa cấp, mạng RPN phát hiện các vùng có thể chứa đối tượng trong ảnh.
- **ROI Heads:** Xử lý các bản đồ đặc trưng (feature maps) được tạo từ các vùng được chọn trong hình ảnh. Điều này được thực hiện bằng cách trích xuất và chuyển đổi các feature maps dựa trên các hộp đề xuất thành các đặc trưng cố định có kích thước khác nhau, sau đó tinh chỉnh vị trí hộp và phân loại đối tượng thông qua các lớp fully connected.

Với cấu trúc trên, Detectron2 cung cấp một nền tảng mạnh mẽ và linh hoạt cho việc phát hiện, phân loại và phân đoạn đối tượng trong hình ảnh.

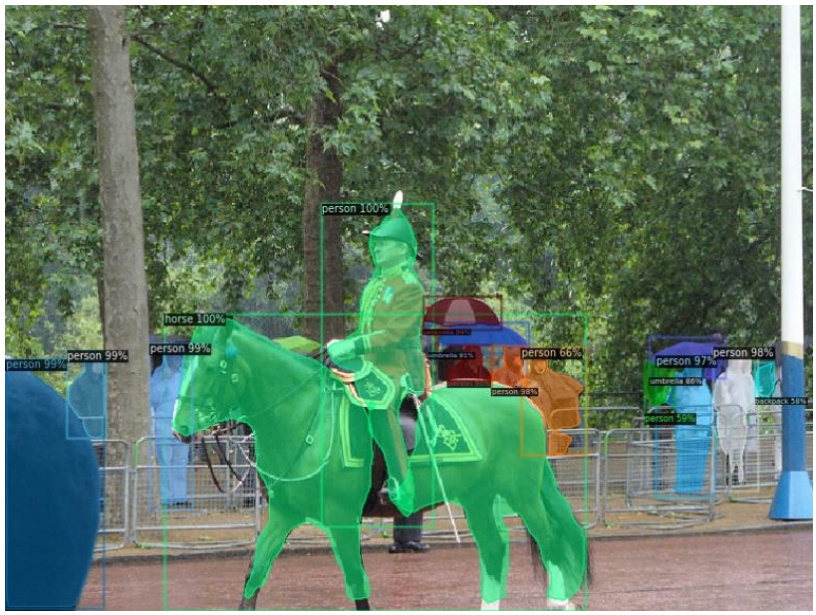
Các tính năng nổi bật của Detectron2:

- **Thiết Kế Modular:** Kiến trúc modular của Detectron2 cho phép người dùng dễ dàng thử nghiệm với các cấu trúc mô hình, hàm mất mát và kỹ thuật huấn luyện khác nhau.
- **Hiệu Suất Cao:** Detectron2 đạt được hiệu suất vượt trội trên nhiều tiêu chuẩn khác nhau, bao gồm cả bộ dữ liệu COCO (Common Objects in Context) và LVIS (Large Vocabulary Instance Segmentation), với hơn 1,5 triệu đối tượng và 250.000 ví dụ về dữ liệu điểm chính.
- **Hỗ Trợ Tập Dữ Liệu Tùy Chỉnh:** Khung làm việc của Detectron2 cung cấp các công cụ hữu ích cho việc làm việc với các tập dữ liệu tùy chỉnh, giúp người dùng dễ dàng tích hợp dữ liệu riêng của mình.
- **Mô Hình Được Huấn Luyện Sẵn:** Bộ sưu tập mô hình của Detectron2 bao gồm nhiều mô hình được huấn luyện sẵn cho từng nhiệm vụ trong thị giác máy tính, giúp người dùng tiết kiệm thời gian và công sức trong việc phát triển mô hình.

- **Inference Hiệu Quả:** Detectron2 bao gồm các tối ưu hóa cho quá trình inference, đảm bảo hoạt động tốt trong các môi trường sản xuất với yêu cầu thời gian thực hoặc độ trễ thấp.
- **Cộng Đồng Phát Triển Năng Động:** Nhờ tính chất mã nguồn mở, Detectron2 có một cộng đồng phát triển sôi nổi với sự đóng góp từ người dùng trên toàn thế giới, giúp thúc đẩy sự cải tiến và đổi mới liên tục.

Dưới đây là một ví dụ sử dụng Detectron2 xử lý bài toán Phát hiện đối tượng và phân đoạn đối tượng (Object Detetion và Segmentation):

Hình 2-10: Hình ảnh trước khi sử dụng Detectron2



Hình 2-11: Hình ảnh đã được phát hiện và phân loại đối tượng khi sử dụng Detectron2

2.3.2. Ứng dụng của Detectron2 trong đồ án

Trong đồ án **phát hiện và phân đoạn các hư hỏng trên xe ô tô thông qua hình ảnh**, Detectron2 được ứng dụng như một công cụ quan trọng để thực hiện bài toán phân đoạn đối tượng, cụ thể ở đây sẽ sử dụng mô hình **Mask R-CNN**. Với khả năng nhận diện và phân đoạn chính xác, Detectron2 giúp giải quyết một cách hiệu quả bài toán phát hiện các khu vực bị hư hỏng của xe trên ảnh chụp.

Vai trò và ứng dụng cụ thể của Detectron2 trong đồ án:

- Phát hiện đối tượng và phân đoạn hư hỏng.
- Sử dụng tích hợp mô hình Mask R-CNN với **ResNet và FPN**: Với ResNet cho phép mô hình học được các đặc trưng phức tạp từ hình ảnh, còn FPN giúp phát hiện hư hỏng ở các kích thước khác nhau, từ những vết xước nhỏ đến những phần thiệt hại lớn.
- Quy trình huấn luyện và dự đoán tối ưu: Detectron2 cho phép huấn luyện mô hình trên bộ dữ liệu hình ảnh lớn về xe ô tô và các loại hư hỏng khác nhau, kết hợp với việc tinh chỉnh các siêu tham số như learning rate, batch size, và số epoch nhằm tối ưu hiệu suất của mô hình.
- Khả năng tăng cường dữ liệu: Detectron2 hỗ trợ nhiều kỹ thuật **data augmentation**, bao gồm xoay, cắt, thay đổi độ sáng và độ tương phản, giúp cải thiện khả năng tổng quát hóa của mô hình. Điều này giúp mô hình có thể xử lý dữ liệu một cách hiệu quả.
- Triển khai trên nền tảng web: Một trong những ứng dụng quan trọng trong đồ án này là triển khai mô hình đã được đào tạo qua Detectron2 lên một nền tảng web. Điều này giúp cho người dùng tải hình ảnh xe cần kiểm tra, sau đó hệ thống sẽ tự động phát hiện và hiển thị các vùng bị hư hỏng, giúp đưa ra đánh giá nhanh chóng về tình trạng của xe.

2.3.3. Cài đặt Detectron2

Dưới đây là các bước cài đặt Detectron2:

2.3.3.1. Yêu cầu phần cứng và phần mềm

- **Hệ điều hành**: Linux hoặc Windows.

- **CUDA:** Để tận dụng sức mạnh của GPU trong việc huấn luyện và dự đoán thì CUDA cần phải được cài đặt (Phiên bản tương thích với Phiên bản PyTorch). Detectron2 hoạt động tốt với CUDA 10.1, 11.0 hoặc cao hơn.
- **Python:** Detectron2 hỗ trợ Python từ 3.7 đến 3.10.
- **PyTorch:** Detectron2 được xây dựng dựa trên PyTorch nên cần phải cài đặt phiên bản tương thích (ví dụ PyTorch 1.7.1 trở lên).

Chi tiết cài đặt trong đồ án của em:

Hệ điều hành	CUDA	Python	PyTorch
Windows	12.4	3.10	2.4.0

Bảng 2-1: Chi tiết các phiên bản cài đặt

Vì em huấn luyện trên local, nên em đã nâng cấp bộ nhớ máy lên 16GB RAM.

2.3.3.2. Cài đặt Detectron2 từ kho lưu trữ (GitHub)

- Cài đặt PyTorch 2.4.0 với CUDA 12.4 qua pip:
  python -m pip install torch==2.4.0+cu124 torchvision torchaudio --index-url https://download.pytorch.org/whl/cu124
- Cài đặt qua pip:
  python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'

Kết quả sau khi tải các phiên bản trên:

```
import torch, torchvision
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print(torch.__version__, torch.cuda.is_available())
print("detectron2:", detectron2.__version__)
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Tue_Feb_27_16:28:36_Pacific_Standard_Time_2024
Cuda compilation tools, release 12.4, V12.4.99
Build cuda_12.4.r12.4/compiler.33961263_0
2.4.0+cu124 True
detectron2: 0.6
```

Hình 2-12: Kiểm tra phiên bản của các cài đặt

2.4. Python Flask

2.4.1. Giới thiệu về Flask

Flask được phát triển bởi Armin Ronacher và được ra mắt vào năm 2010. Đây là một micro web framework dành cho Python, được thiết kế để phát triển các ứng dụng web nhỏ gọn nhưng mạnh mẽ. Flask nổi bật nhờ tính năng đơn giản, dễ học hỏi và khả năng mở rộng linh hoạt thông quá các extension.

2.4.2. Cấu trúc cơ bản của ứng dụng Flask

- **App Object (Đối tượng ứng dụng)**: Mọi ứng dụng Flask đều bắt đầu bằng việc khởi tạo một đối tượng Flask. Đây là điểm khởi đầu của tất cả các thành phần trong ứng dụng.
- **Route (Định tuyến)**: Route là cách Flask xác định URL và ánh xạ nó đến một hàm xử lý tương ứng. Mỗi URL sẽ tương ứng với một đoạn mã để xử lý yêu cầu từ người dùng.
- **Request và Response**: Flask sử dụng đối tượng request để nhận dữ liệu từ người dùng, và trả về response chứa kết quả. Flask hỗ trợ nhiều loại phương thức HTTP như GET, POST, PUT, DELETE, v.v.
- **Template (Giao diện động)**: Flask sử dụng Jinja2 làm engine template, cho phép tạo ra các giao diện HTML động dựa trên dữ liệu.

2.4.3. Python Flask trong dự án phát hiện hư hỏng xe ô tô

Trong đồ án phát hiện hư hỏng xe ô tô thông qua hình ảnh, Flask được sử dụng để xây dựng giao diện web cho phép người dùng tải lên hình ảnh xe ô tô cần kiểm tra. Sau đó, ảnh sẽ được xử lý bởi mô hình Mask R-CNN đã được huấn luyện, và kết quả phân vùng hư hỏng sẽ được hiển thị lại trên giao diện web.

Quy trình hoạt động:

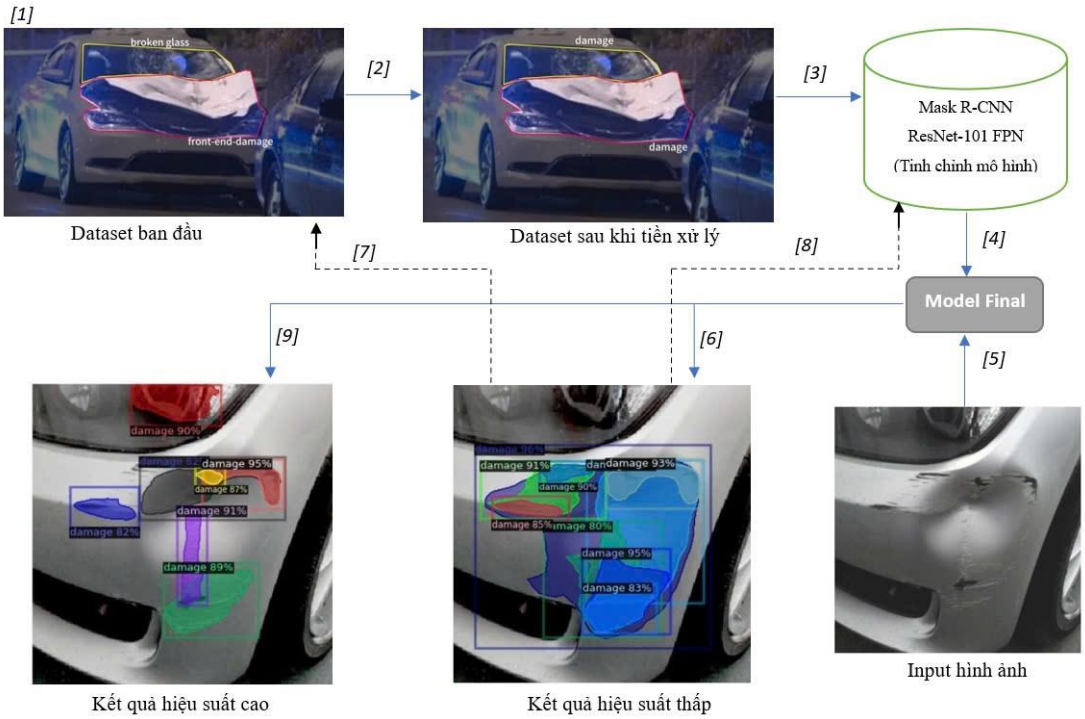
1. Người dùng tải lên hình ảnh xe ô tô
2. Flask nhận và chuyển hình ảnh đến mô hình đã huấn luyện
3. Mô hình Mask R-CNN đã được huấn luyện dự đoán vùng hư hỏng
4. Kết quả sẽ được Flask hiển thị trên giao diện web

Chương 3. TRIỂN KHAI MÔ HÌNH PHÁT HIỆN VÀ PHÂN ĐOẠN HƯ HỎNG XE Ô TÔ

3.1. Giới thiệu mô hình đề xuất nghiên cứu cho bài toán

Mô hình được em sử dụng trong bài toán phát hiện và phân đoạn hư hỏng xe ô tô ở đây là Mask R-CNN sử dụng ResNet-101 là backbone và FPN để cải thiện khả năng phát hiện đối tượng ở nhiều kích thước khác nhau.

Mô hình mô tả quy trình xử lý bài toán như sau:



Hình 3-1: Mô hình mô tả quy trình xử lý

Giải thích các mục:

[1] Input dataset ban đầu: Hình ảnh xe ô tô và dữ liệu chú thích (Data Annotation dạng JSON)

[2] Tiền xử lý dữ liệu:

- Đối với những dataset chứa nhiều nhãn (Classes) thì chuyển về một nhãn duy nhất là **'damage'**
- Chia bộ dữ liệu thành 3 tập train/val/test
- Đăng ký dataset (cả 3 tập train/val/test) vào hệ thống Detectron2

[3] Sử dụng Mô hình: Sử dụng mô hình Mask R-CNN với kiến trúc ResNet-101 FPN

[4] Huấn luyện mô hình: *nhằm đưa ra mô đun cuối cùng*

- Tập train sau khi đã đăng ký vào hệ thống Detectron2 được sử dụng để huấn luyện mô hình
- Điều chỉnh các siêu tham số (Batch, ROI, Epoch, ITER,…)
- Tập test để đánh giá cuối cùng

[5] Dự đoán:

- Input là một hình ảnh cần dự đoán hư hỏng của xe
- Sử dụng mô đun đã được huấn luyện ở trên để dự đoán

[6], [9] Đánh giá mô hình

- Đánh giá dựa trên các chỉ số theo hai loại dự đoán và bbox (bounding box) và segm (segmentation mask)
    - **AP**: Chỉ số trùng bình chính xác trên tất cả các ngưỡng IoU từ 0,5 đến 0,95
    - **AP50**: Chỉ số trung bình chính xác tại ngưỡng IoU=0,5
    - **AP75**: Chỉ số trung bình chính xác tại ngưỡng IoU=0,75
    - **APs, APm, APl**: Chỉ số chính xác trên các đối tượng nhỏ, trung bình, lớn
- Đối với kết quả có hiệu suất thấp **[6]** → Thì cần: **[7]** Thêm dataset để tăng cường dữ liệu, hoặc **[8]** Tinh chỉnh/ thay đổi mô hình.
- Đối với kết quả có hiệu suất cao **[9]** → Mô đun đạt yêu cầu.

3.2. Quá trình sử dụng mô hình

3.2.1. Chuẩn bị bộ dữ liệu

3.2.1.1. Thu thập dữ liệu (Dataset Collection)

Em thu thập các bộ dữ liệu từ trang web Roboflow Universe với các hình ảnh cho các loại hư hỏng ô tô khác nhau như: hư hỏng do va chạm, kính vỡ, đèn pha vỡ, đèn hậu vỡ, trầy xước, móp méo các bộ phận khác của xe,... Những bộ dữ

liệu này được xây dựng từ nhiều nguồn khác nhau, đảm bảo sự đa dạng và phong phú về kiểu dáng, màu sắc và vị trí hư hỏng. Mỗi bộ dữ liệu được phân loại thành nhiều lớp (classes) khác nhau, cho phép mô hình học hỏi và nhận diện các loại hư hỏng cụ thể. Đặc biệt, các hình ảnh trong bộ dữ liệu đều được đảm bảo có độ phân giải cao và chất lượng tốt, nhằm phục vụ cho việc huấn luyện mô hình một cách hiệu quả nhất. Bên cạnh đó các bộ dữ liệu thu thập được đều đi kèm với tập chú thích dữ liệu (Data Annotation) chi tiết, trong này sẽ phân định rõ ràng khu vực bị hư hỏng trong mỗi hình ảnh và được lưu trữ ở định dạng COCO JSON.

3.2.1.2. Dữ liệu chú thích (Data Annotation)

Dữ liệu chú thích (Data Annotation) là quá trình gán nhãn cho dữ liệu, giúp các mô hình máy học hiểu và phân tích dữ liệu tốt hơn. Trong lĩnh vực thị giác máy tính, nó đóng vai trò quan trọng trong việc huấn luyện các mô hình học sâu để nhận diện và phân đoạn các đối tượng trong hình ảnh.

Một định dạng phổ biến cho dữ liệu chú thích là COCO JSON, cung cấp cách lưu trữ hiệu quả và hỗ trợ phát triển các mô hình nhận diện hình ảnh. Mỗi hình ảnh trong bộ dữ liệu đi kèm với các chú thích rõ ràng về vị trí và kích thước các vùng bị hư hỏng, với các lớp hư hỏng được đánh dấu bằng nhãn tương ứng. Điều này tạo điều kiện thuận lợi cho việc huấn luyện mô hình Mask R-CNN. Việc sử dụng dữ liệu chú thích chính xác và đầy đủ là rất quan trọng, vì nó ảnh hưởng trực tiếp đến khả năng nhận diện và phân loại các loại hư hỏng của mô hình cũng như đánh giá hiệu suất của nó trong quá trình kiểm tra và tinh chỉnh.

Cấu trúc cơ bản của một file dữ liệu chú thích cho bài toán phát hiện đối tượng gồm: (Ở định dạng COCO JSON):

- **“images”:** Chứa thông tin về hình ảnh trong tập dữ liệu bao gồm như:
    + file_name: Tên của file hình ảnh.
    + height: Chiều cao của hình ảnh.
    + width: Chiều rộng của hình ảnh.
    + id: Mã định danh cho hình ảnh trong tập dữ liệu (giúp liên kết hình ảnh với các chú thích trong phần annotations)
- **“annotations”:** Chứa thông tin chi tiết về các đối tượng được chú thích trong hình ảnh bao gồm như:

+ category_id: Mã định danh cho loại đối tượng.
+ iscrowd: Biến nhị phân cho biết đối tượng có phải là một đám đông hay không. Nếu giá trị là 1, điều này có nghĩa là đối tượng có thể bao gồm nhiều thành phần, trong khi 0 chỉ ra rằng đây là một đối tượng đơn lẻ.
+ bbox: Danh sách chứa tọa độ của hộp giới hạn (bounding box) bao quanh đối tượng như: Tọa độ x và y, chiều rộng và chiều cao của hộp.
+ area: Diện tích của hộp giới hạn.
+ segmentation: Một mảng chứa các tọa độ của các điểm phân đoạn để xác định chính xác hình dạng của đối tượng.
+ Id: Mã định danh duy nhất cho đối tượng chú thích trong phần này.
+ image_id: Mã định danh của hình ảnh mà đối tượng này thuộc về

- **“categories”:** Chứa thông tin về các loại đối tượng có trong tập dữ liệu bao gồm:
    + id: Mã định danh duy nhất cho một loại đối tượng trong tập dữ liệu.
    + name: Tên của đối tượng, cụ thể là classes


```
{
  "images": {
    "file_name": "Car damages 1.png",
    "height": 476,
    "width": 881,
    "id": 1
  },
  "annotations": {
    "category_id": 1,
    "iscrowd": 0,
    "bbox": [ 210.0, 233.0, 12.0, 18.0 ],
    "area": 44.5,
    "segmentation": [
      [
        220.0,
        251.0,
        220.0,
        244.0,
        215.0,
        239.0,
        210.0
      ]
    ],
    "id": 1,
    "image_id": 0
  },
  "categories": [
    {
      "id": 1,
      "name": "scratch"
    }
  ]
}
```


Hình 3-2: Ví dụ về dữ liệu chú thích

3.2.2. Tiền xử lý bộ dữ liệu

Dataset của bài toán này được em thu thập từ nhiều nguồn khác nhau với mỗi bộ dữ liệu được phân loại thành nhiều lớp (classes) khác nhau như: hư hỏng do va chạm, kính vỡ, đèn pha vỡ, đèn hậu vỡ, trầy xước, móp méo các bộ phận khác của xe,... Với mục đích là xác định hư hỏng của xe ô tô nên em chỉ cần sử dụng một lớp để xác định hư hỏng là : classes = "damage". Do đó cần tổng hợp các hình ảnh, các dữ liệu chú thích về chung một file dataset, đồng thời chuyển về một lớp duy nhất là "damage".

Vì cấu trúc dữ liệu mà Detectron2 yêu cầu bao gồm các trường thông tin chính trong dạng COCO JSON như "images", "annotations" và "categories". Do đó cần đăng ký tập dữ liệu cho Detectron2 thông qua hàm **DatasetCatalog.register** để thêm tập dữ liệu của mình vào danh sách các tập dữ liệu có sẵn trong Detectron2. Đăng ký các loại dữ liệu khác nhau (train, val, test).

Việc đăng ký tập dữ liệu giúp người dùng dễ dàng quản lý và sử dụng các tập dữ liệu tùy chỉnh, đồng thời cho phép mô hình hoạt động hiệu quả hơn nhờ vào cấu trúc rõ ràng và có hệ thống của dữ liệu.

3.2.3. Định nghĩa cấu hình cho việc tinh chỉnh mô hình Detectron2

Việc định nghĩa cấu hình là một bước quan trọng trong quá trình tinh chỉnh mô hình Detectron2, giúp xác định cách mà mô hình sẽ được huấn luyện, đánh giá và dự đoán. Cấu hình trong Detectron2 được quản lý thông qua các tệp JSON hoặc thông qua các đối tượng Python, cho phép người dùng điều chỉnh các thông số theo yêu cầu của bài toán cụ thể. Dưới đây là một số khía cạnh cần chú ý khi định nghĩa cấu hình:

3.2.3.1. Cấu hình của Detectron2 bao gồm nhiều thành phần chính như:

- **MODEL**: Các thông số liên quan đến mô hình, bao gồm loại kiến trúc (ví dụ: Mask R-CNN, Faster R-CNN), backbone (ResNet, ResNeXt), và các hyperparameters khác.
- **DATASETS**: Tên các tập dữ liệu được sử dụng cho huấn luyện và kiểm tra, giúp mô hình xác định nguồn dữ liệu đầu vào.

- **SOLVER**: Các thông số liên quan đến quá trình tối ưu hóa, bao gồm learning rate, số epoch, kích thước batch, và các thông số khác để điều chỉnh quá trình huấn luyện.
- **DATALOADER**: Các thông số cho bộ tải dữ liệu, như số lượng worker và cách thức lấy mẫu dữ liệu.

3.2.3.2. Cụ thể trong bài như sau:

Cấu hình mẫu từ Model Zoo của Detectron2 đó là **Mask R-CNN sử dụng ResNet-101 là backbone và FPN**

Tập dữ liệu huấn luyện (Train) và kiểm tra (Test): Sử dụng tập dữ liệu đã được đăng ký dữ liệu cho Detectron2

Ý nghĩa của các siêu tham số được cấu hình trong mô hình Detectron2:

- **cfg.SOLVER.IMS_PER_BATCH:** Đây là số lượng hình ảnh được sử dụng trong mỗi lần cập nhật trọng số của mô hình (mỗi batch). Giá trị này ảnh hưởng đến tốc độ huấn luyện và cách mô hình học từ dữ liệu.
- **cfg.SOLVER.BASE_LR:** Đây là learning rate (tỷ lệ học) cơ bản cho quá trình tối ưu hóa. Tỷ lệ học xác định mức độ điều chỉnh trọng số của mô hình sau mỗi lần huấn luyện.
- **cfg.SOLVER.MAX_ITER:** Đây là tổng số bước huấn luyện mà mô hình sẽ thực hiện. Một bước huấn luyện tương ứng với một lần cập nhật trọng số dựa trên một batch dữ liệu.
- **cfg.SOLVER.WEIGHT_DECAY:** Đây là một tham số điều chỉnh cho quá trình regularization nhằm ngăn chặn overfitting bằng cách giảm độ lớn của các trọng số trong mô hình.
- **cfg.SOLVER.CHECKPOINT_PERIOD:** Đây là số bước mà mô hình sẽ tự động lưu lại trạng thái hiện tại của nó trong quá trình huấn luyện. Điều này rất quan trọng để có thể tiếp tục huấn luyện sau này nếu cần thiết.
- **cfg.SOLVER.WARMUP_ITERS:** Đây là số bước mà learning rate sẽ được tăng dần từ giá trị nhỏ đến giá trị base learning rate đã chỉ định. Quá trình này giúp mô hình ổn định trong những bước đầu tiên của huấn luyện.

- **cfg.SOLVER.WARMUP_FACTOR:** Đây là tỷ lệ mà learning rate sẽ được khởi tạo trong giai đoạn warmup. Ví dụ, nếu tỷ lệ này là 0.001, learning rate sẽ bắt đầu từ 0.001 và tăng dần lên giá trị base learning rate.
- **cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE:** Đây là số lượng đối tượng được xử lý trong một hình ảnh trong mỗi lần tính toán cho các ROI heads.
- **cfg.MODEL.ROI_HEADS.NUM_CLASSES:** Đây là số lượng lớp mà mô hình sẽ phân loại. Trong trường hợp của bạn, giá trị này là 1, có nghĩa là mô hình chỉ cần nhận diện một lớp duy nhất là "damage".

3.2.4. Huấn luyện

Em xây dựng một lớp để huấn luyện mô hình là **MyTrainer**, là một lớp tùy chỉnh kế thừa từ **DefaultTrainer** trong thư viện Detectron2. Lớp này được em thiết kế để hỗ trợ huấn luyện mô hình với khả năng xử lý số thực hỗn hợp (Mixed Precision Training), giúp cải thiện hiệu suất và giảm tài nguyên cần thiết trong quá trình huấn luyện. Chi tiết bên trong MyTrainer như sau:

- **Được kế thừa từ DefaultTrainer:** Có thể sử dụng tất cả các phương thức và thuộc tính đã được định nghĩa trong lớp cha.
- **Các hàm khởi tạo:**
  - Khởi tạo GradScaler: Được sử dụng để hỗ trợ việc huấn luyện với số thực hỗn hợp (mixed precision). Nó tự động điều chỉnh độ lớn của gradient trong quá trình tính toán, giúp cải thiện độ chính xác và hiệu suất.
  - Khởi tạo logger: Nhằm ghi lại thông tin về quá trình huấn luyện, giúp theo dõi và phân tích hiệu suất mô hình trong thời gian thực.
- **Các phương thức tùy chỉnh:**
  - Build_evaluator: Cho phép đánh giá mô hình trên tập dữu liệu cụ thể
  - Build_train_loader: Tạo một Data Loader cho quá trình huấn luyện. Dữ liệu được nạp vào theo cách đã được chỉ định, cùng với các phép biến đổi dữ liệu cần thiết để tăng cường hiệu suất của mô hình.
  - Run_step: Đây là phương thức chính thực hiện một bước huấn luyện

- _write_metrics: ghi lại các thông số mất mát và thời gian mỗi khi đạt đến một số bước nhất định. Điều này cung cấp thông tin hữu ích về hiệu suất của mô hình trong quá trình huấn luyện.

3.2.5. Đánh giá và tinh chỉnh

Sau khi quá trình huấn luyện hoàn tất và em có được một mô đun chứa kết quả, bước tiếp theo là tiến hành đánh giá mô hình đã được đào tạo. Đánh giá mô hình đóng vai trò quan trọng trong quy trình phát triển và triển khai các hệ thống nhận diện hình ảnh. Bước này không chỉ giúp xác định hiệu suất của mô hình mà còn cung cấp những thông tin quý giá để đưa ra các quyết định hợp lý cho các bước phát triển tiếp theo. Dưới đây là những nội dung cốt lõi liên quan đến quá trình đánh giá mô hình.

3.2.5.1. Đánh giá mô hình

Mô hình được đánh giá dựa trên các chỉ số chính, tập trung vào hai loại dự đoán: bounding box (bbox) và segmentation mask (segm). Các chỉ số quan trọng bao gồm:

- **AP (Average Precision)**: Là chỉ số trung bình chính xác trên tất cả các ngưỡng Intersection over Union (IoU) từ 0,5 đến 0,95. AP cho phép đánh giá toàn diện khả năng của mô hình trong việc phát hiện các đối tượng với độ chính xác khác nhau.
- **AP50**: Là chỉ số trung bình chính xác tại ngưỡng IoU = 0,5. Chỉ số này thường được sử dụng để đánh giá hiệu suất tối thiểu của mô hình trong việc phát hiện các đối tượng.
- **AP75**: Là chỉ số trung bình chính xác tại ngưỡng IoU = 0,75. Chỉ số này giúp đánh giá khả năng của mô hình trong việc phát hiện các đối tượng với độ chính xác cao hơn.
- **APs, APm, APl**: Là các chỉ số trung bình chính xác cho các đối tượng nhỏ (APs), trung bình (APm), và lớn (APl). Các chỉ số này giúp xác định khả năng của mô hình trong việc phát hiện các kích thước đối tượng khác nhau.

3.2.5.2. Phân tích hiệu suất nằm tinh chỉnh mô hình

Kết quả có hiệu suất thấp:

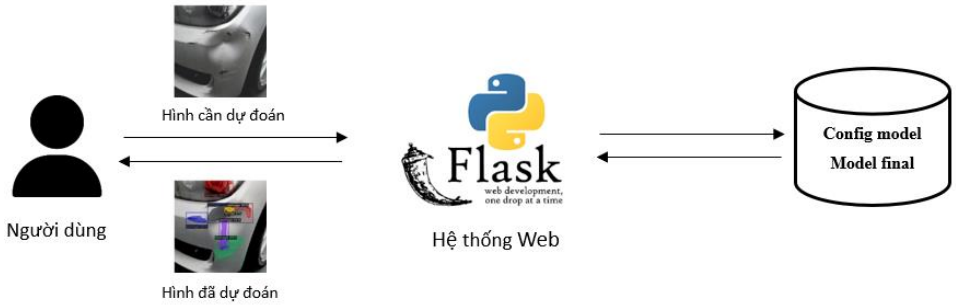
- Nếu mô hình không đạt yêu cầu và có hiệu suất thấp, cần xem xét các giải pháp như:
    - **Thêm Dataset**: Tăng cường dữ liệu bằng cách mở rộng tập dữ liệu huấn luyện với nhiều hình ảnh và chú thích hơn. Điều này có thể giúp mô hình học được nhiều đặc trưng hơn và cải thiện khả năng phát hiện.
    - **Tinh chỉnh/Thay đổi Mô Hình**: Điều chỉnh các siêu tham số hoặc thay đổi kiến trúc của mô hình có thể giúp cải thiện hiệu suất. Việc này bao gồm thử nghiệm với các mô hình khác nhau, điều chỉnh learning rate, số lượng epochs, hoặc số lượng layer trong mô hình.

Kết quả có hiệu suất cao:

- Nếu mô hình đạt yêu cầu với các chỉ số cao, điều này cho thấy mô hình hoạt động hiệu quả. Tuy nhiên, việc tiếp tục theo dõi hiệu suất và điều chỉnh mô hình khi cần thiết vẫn là điều quan trọng để đảm bảo rằng mô hình có thể duy trì hiệu suất cao trong các tình huống thực tế.

3.2.6. Triển khai

Triển khai mô hình là bước quan trọng sau khi đạt được một mô đun đáp ứng các yêu cầu về hiệu suất và độ chính xác. Khi mô hình hoạt động hiệu quả, việc triển khai lên một nền tảng web sẽ cho phép người dùng dễ dàng truy cập và sử dụng tính năng nhận diện hình ảnh mà mô hình cung cấp. Trong trường hợp này, em sử dụng Python Flask, một framework nhẹ và linh hoạt, để xây dựng ứng dụng web.



Hình 3-3: Mô hình hệ thống triển khai lên web

Quá trình triển khai gồm các bước sau:

- **Thiết lập môi trường**: Đầu tiên, cần đảm bảo rằng môi trường phát triển đã được cài đặt đầy đủ các thư viện cần thiết, bao gồm Flask và các thư viện hỗ trợ khác để tích hợp mô hình học sâu.
- **Xây dựng ứng dụng Flask**: Tạo một ứng dụng Flask cho phép người dùng tải lên hình ảnh để mô hình có thể xử lý và trả về kết quả. Giao diện người dùng sẽ được thiết kế thân thiện, giúp người dùng dễ dàng tương tác.
- **Tích hợp mô hình**: Mô hình đã được huấn luyện sẽ được tích hợp vào ứng dụng Flask, cho phép nó thực hiện suy luận trên các hình ảnh được tải lên. Việc này bao gồm việc nạp trọng số của mô hình và xử lý đầu vào để tạo ra các dự đoán.
- **Triển khai trang web**: Cuối cùng, ứng dụng Flask sẽ được triển khai lên một trang web, cho phép người dùng có thể truy cập qua trình duyệt để sử dụng tính năng nhận diện hình ảnh mà mô hình cung cấp.

3.3. Triển khai mô hình giải quyết bài toán

3.3.1. Thu thập và tiền xử lý bộ dữ liệu

Em thu thập năm bộ dữ liệu từ trang web Roboflow Universe và được 20.448 hình tất cả [12] [13] [14] [15] [16]. Trong đó được chia ra 16.447 hình cho tập train, 2.648 hình cho tập val và 1.353 hình cho tập test. Và với mỗi tập đều có file dữ liệu chú thích tương ứng.

Chi tiết các bước tổng hợp và tiền xử lý dữ liệu như sau:

Bước 1: Chuyển các dataset về chung một class duy nhất là **"damage"**

```
import json

def convert_coco_labels(input_json_path, output_json_path):
    # Đọc file COCO JSON
    with open(input_json_path, 'r') as f:
        coco_data = json.load(f)

    # Tạo mapping từ các category_id cũ về category_id mới 'damage'
    old_label_ids = [1, 2]  # Các ID hiện có trong file JSON
    new_label_id = 1  # ID cho nhãn mới 'damage'

    # Sửa đổi các annotations
    for annotation in coco_data['annotations']:
        if annotation['category_id'] in old_label_ids:
            annotation['category_id'] = new_label_id

    # Sửa đổi danh sách categories để chỉ có nhãn 'damage'
    coco_data['categories'] = [{'id': new_label_id, 'name': 'damage'}]

    # Ghi lại file COCO JSON đã chỉnh sửa
    with open(output_json_path, 'w') as f:
        json.dump(coco_data, f, indent=4)

# Áp dụng cho cả test, train và valid

convert_coco_labels('D:/DoAn/Test/input/NewCarDame/train/_annotations.coco.json', 'D:/DoAn/Test/input/NewCarDame/train/NCD_anno_train.json')
convert_coco_labels('D:/DoAn/Test/input/NewCarDame/valid/_annotations.coco.json', 'D:/DoAn/Test/input/NewCarDame/valid/NCD_anno_val.json')
convert_coco_labels('D:/DoAn/Test/input/NewCarDame/test/_annotations.coco.json',  'D:/DoAn/Test/input/NewCarDame/test/NCD_anno_test.json')
```

Hình 3-4: Code chuyển nhiều classes về một class duy nhất "damage"

Hình trên là của bộ dữ liệu "NewCarDame" đang có hai classes khác nhau cần chuyển về một class duy nhất. Tương tự với những bộ dữ liệu khác.

Bước 2: Tổng hợp bộ chú thích của năm bộ dữ liệu thành một file chú thích

```python
import json

def merge_coco_jsons(json_files, output_file):
    merged_data = {
        "images": [],
        "annotations": [],
        "categories": []
    }

    image_id_offset = 0
    annotation_id_offset = 0

    for json_file in json_files:
        with open(json_file, 'r') as f:
            data = json.load(f)

            if not merged_data["categories"]:
                merged_data["categories"] = data["categories"]

            for img in data["images"]:
                img["id"] += image_id_offset
                merged_data["images"].append(img)

            for ann in data["annotations"]:
                ann["id"] += annotation_id_offset
                ann["image_id"] += image_id_offset
                merged_data["annotations"].append(ann)

        image_id_offset += len(data["images"])
        annotation_id_offset += len(data["annotations"])

    with open(output_file, 'w') as f:
        json.dump(merged_data, f)


json_files = ['D:/DoAn/Test/input/CarDamageV5/train/CDV5_anno_train.json', 'D:/DoAn/Test/input/Car Damage Img/train/CDI_anno_train.json',
              'D:/DoAn/Test/input/CarDDV1/train/CDDV1_anno_train.json', 'D:/DoAn/Test/input/NewCarDame/train/NCD_anno_train.json',
              'D:/CarDamageDetectron_DoAn/working/data/train/CDD_anno_train.json' ]

output_file = 'D:/CodeDoAn_NhatCuong/Allmerged_data_train.json'

merge_coco_jsons(json_files, output_file)
```

Hình 3-5: Code tổng hợp cả năm bộ dữ liệu thành một cho tập train

Hình trên là tổng hợp bộ chú thích cho tập train, tương tự với bộ dữ liệu tập val

Kết quả sau khi xử lý xong bộ dữ liệu:

Tập train:

```python
# Đường dẫn tới file merged_data.json
merged_data_train_json = 'D:/CodeDoAn_NhatCuong/working/data/train/Allmerged_data_train.json'

# Đọc nội dung của file JSON
with open(merged_data_train_json, 'r') as f:
    merged_data = json.load(f)

# Đếm tổng số phần tử trong từng danh mục
total_images = len(merged_data['images'])
total_annotations = len(merged_data['annotations'])
total_categories = len(merged_data['categories'])

# Hiển thị kết quả
print(f"Tổng số images: {total_images}")
print(f"Tổng số annotations: {total_annotations}")
print(f"Tổng số categories: {total_categories}")
```

✓ 0.3s

```
Tổng số images: 16447
Tổng số annotations: 29931
Tổng số categories: 1
```

Hình 3-6: Hiển thị thông tin chú thích của tập Train

Tập val:

```
# Đường dẫn tới file merged_data.json
merged_data_val_json = 'D:/CodeDoAn_NhatCuong/working/data/val/Allmerged_data_val.json'

# Đọc nội dung của file JSON
with open(merged_data_val_json, 'r') as f:
    merged_data = json.load(f)

# Đếm tổng số phần tử trong từng danh mục
total_images = len(merged_data['images'])
total_annotations = len(merged_data['annotations'])
total_categories = len(merged_data['categories'])

# Hiển thị kết quả
print(f"Tổng số images: {total_images}")
print(f"Tổng số annotations: {total_annotations}")
print(f"Tổng số categories: {total_categories}")
```

✓ 0.1s

```
Tổng số images: 2648
Tổng số annotations: 4815
Tổng số categories: 1
```

Hình 3-7: Hiển thị thông tin chú thích của tập Val

Bước 3: Đăng ký tập dữ liệu chú thích vào hệ thống của Detectron2

```
register_coco_instances("car_dataset_train", {}, ann_train_alldata, image_root=img_dir)
register_coco_instances("car_dataset_val", {}, ann_val_alldata, image_root=img_dir)
```

✓ 0.0s

Hình 3-8: Đăng ký vào hệ thống của Detectron2

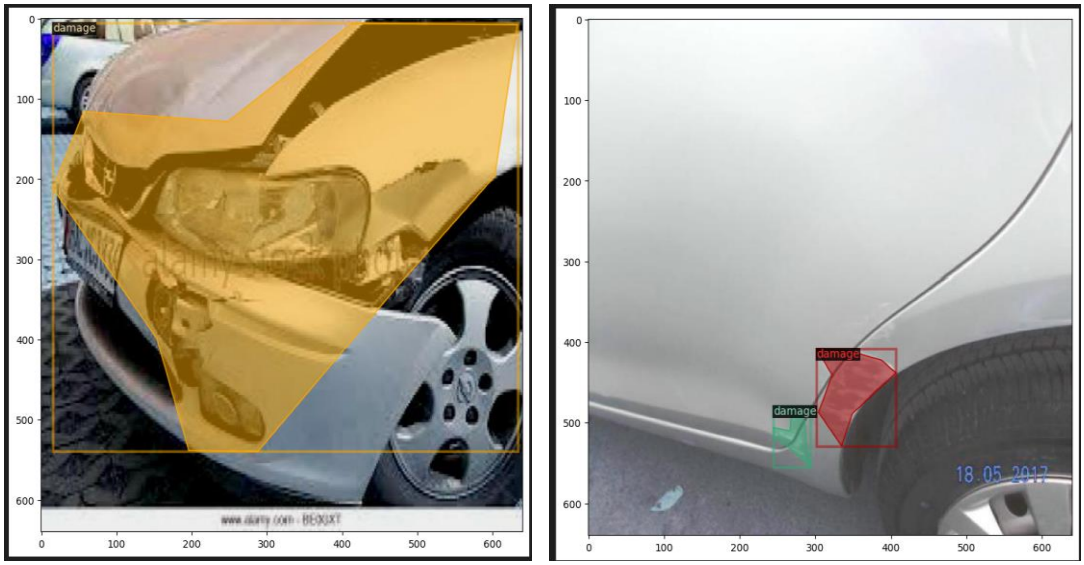
```
dataset_dicts = DatasetCatalog.get("car_dataset_train")
metadata_dicts= MetadataCatalog.get("car_dataset_train")

print(dataset_dicts)
print(metadata_dicts)
```

✓ 1.3s

```
[10/03 23:25:35 d2.data.datasets.coco]: Loaded 16447 images in COCO format from D:/CodeDoAn_NhatCuong/working/data/train/Allmerged_data_train.json
```

Hình 3-9: Kiểm tra và xác minh đã được đăng ký vào hệ thống của Detectron2

Bước 4: Trực quan hóa chú thích nhằm kiểm tra chất lượng dữ liệu tập train



Nhận thấy chất lượng dữ liệu chú thích của tập train khá tốt, sẵn sàng cho việc huấn luyện mô hình.

3.3.2. Huấn luyện mô hình

Bước 1: Xây dựng lớp MyTrainer để huấn luyện mô hình

```python
# Tùy chỉnh Trainer với Mixed Precision
class MyTrainer(DefaultTrainer):
    def __init__(self, cfg):
        super().__init__(cfg)
        self.scaler = GradScaler()
        self._data_loader_iter = iter(self.data_loader)

        # Khởi tạo logger
        self.logger = logging.getLogger("detectron2.trainer")
        self.logger.setLevel(logging.INFO)
        handler = logging.StreamHandler()
        formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
        handler.setFormatter(formatter)
        self.logger.addHandler(handler)

        self.step_counter = 0  # Biến đếm số bước

    @classmethod
    def build_evaluator(cls, cfg, dataset_name):
        return COCOEvaluator(dataset_name, cfg, False, output_dir=cfg.OUTPUT_DIR)

    @classmethod
    def build_train_loader(cls, cfg):
        return build_detection_train_loader(
            cfg,
            mapper=DatasetMapper(cfg, is_train=True, augmentations=train_aug())
        )
```

```python
def run_step(self):
    assert self.model.training, "[MyTrainer] Model was changed to eval mode!"
    start = time.perf_counter()
    self.optimizer.zero_grad()

    # Mixed Precision Training
    with autocast():
        data = next(self._data_loader_iter)
        loss_dict = self.model(data)
        losses = sum(loss_dict.values())

    # Tăng biến đếm
    self.step_counter += 1

    # Ghi lại các chỉ số (metrics) mỗi 20 bước
    if self.step_counter % 20 == 0:
        self._write_metrics(loss_dict, time.perf_counter() - start)

    # Scale gradients và thực hiện backpropagation
    self.scaler.scale(losses).backward()
    self.scaler.step(self.optimizer)
    self.scaler.update()

def _write_metrics(self, loss_dict, time_elapsed):
    # Tính toán tổng số mất mát
    loss = sum(loss_dict.values())

    # Tạo một chuỗi với các thông số trên một hàng
    metrics = (
        f"loss: {loss:.3f} | "
        f"time: {time_elapsed:.3f}s | "
        f"loss_cls: {loss_dict.get('loss_cls', 0.0):.3f} | "
        f"loss_box_reg: {loss_dict.get('loss_box_reg', 0.0):.3f} | "
        f"loss_mask: {loss_dict.get('loss_mask', 0.0):.3f} | "
        f"loss_rpn_cls: {loss_dict.get('loss_rpn_cls', 0.0):.3f} | "
        f"loss_rpn_loc: {loss_dict.get('loss_rpn_loc', 0.0):.3f}"
    )

    # Ghi lại chỉ số vào logger
    self.logger.info(metrics)
```

Hình 3-10: Code xây dựng lớp MyTrainer

Mục đích ở đoạn code trên là xây dựng lớp MyTrainer để mở rộng và cải thiện quá trình huấn luyện mô hình thông qua các tính năng như sau:

- **Huấn luyện với độ chính xác hỗn hợp**: Tăng hiệu suất huấn luyện bằng cách giảm tải bộ nhớ mà vẫn giữ độ chính xác cao.
- **Ghi nhật ký quá trình huấn luyện**: Theo dõi tiến trình huấn luyện và các chỉ số cụ thể để phân tích kết quả và phát hiện lỗi.
- **Tích hợp đánh giá mô hình**: Sử dụng COCOEvaluator để đánh giá mô hình với các chỉ số phổ biến trong phát hiện và phân đoạn đối tượng.
- **Tăng cường dữ liệu**: Cải thiện quá trình huấn luyện bằng các kỹ thuật tăng cường dữ liệu.
- **Quản lý quá trình huấn luyện**: Điều khiển từng bước huấn luyện, thực hiện tính toán loss và cập nhật trọng số một cách chính xác và hiệu quả.

Bước 2: Định nghĩa cấu hình huấn luyện

```
# 1. Định nghĩa cấu hình (Define the config)
cfg = get_cfg()
cfg.OUTPUT_DIR = f"{model_dir}outputDfTD_{datetime.datetime.now().strftime('%Y%m%d_%T').replace(':', '')}"

# 2. Tải cấu hình mẫu từ Model Zoo
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml"))

# 3. Thiết lập tập dữ liệu huấn luyện và kiểm tra
cfg.DATASETS.TRAIN = ("car_dataset_train",)
cfg.DATASETS.TEST = ("car_dataset_val",)
cfg.DATALOADER.NUM_WORKERS = 4  # Số luồng (threads) để tải dữ liệu.

# 4. Thiết lập trọng số khởi tạo (pretrained weights) và thiết bị
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml")
cfg.MODEL.DEVICE = 'cuda'

# 5. Thiết lập các siêu tham số của Solver
cfg.SOLVER.IMS_PER_BATCH = 32  # Số img trong mỗi batch huấn luyện
cfg.SOLVER.BASE_LR = 0.00025  # Learning rate (tốc độ học)
cfg.SOLVER.MAX_ITER = 2000  # Số bước huấn luyện
cfg.SOLVER.WEIGHT_DECAY = 0.0001  # Thêm trọng số decay để giảm overfitting
cfg.SOLVER.CHECKPOINT_PERIOD = 500  # Lưu checkpoint mỗi 500 bước

# 6. Thiết lập các tham số cho ROI Heads của mô hình Mask R-CNN
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 8  # Số lượng đề xuất (proposals)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1

# 7. Tạo thư mục đầu ra nếu chưa tồn tại
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

# 8. Huấn luyện mô hình sử dụng MyTrainer (trainer tùy chỉnh)
trainer = MyTrainer(cfg)

# Giữ thứ tự hook
periodic_writer_hook = [hook for hook in trainer._hooks if isinstance(hook, hooks.PeriodicWriter)]
all_other_hooks = [hook for hook in trainer._hooks if not isinstance(hook, hooks.PeriodicWriter)]
trainer._hooks = all_other_hooks + periodic_writer_hook

trainer.resume_or_load(resume=False)  # Tải lại trạng thái mô hình nếu có từ trước
trainer.train()
```

Hình 3-11: Code định nghĩa cấu hình huấn luyện

Trong quá trình huấn luyện mô hình Mask R-CNN dựa trên nền tảng Detectron2, em đã thực hiện một số bước cấu hình và tùy chỉnh cụ thể để phân đoạn đối tượng hư hỏng trên xe ô tô. Mô hình được lựa chọn là **Mask R-CNN sử dụng ResNet-101 kết hợp với FPN,** đây là một kiến trúc mạnh mẽ cho bài toán phân đoạn đối tượng.

Tập dữ liệu huấn luyện và kiểm tra lần lượt được chỉ định là **"car_dataset_train"** và **"car_dataset_val",** cả hai đều đã được đăng ký vào Detectron2. Để tăng tốc độ tải dữ liệu trong quá trình huấn luyện, em đã thiết lập 4 luồng xử lý song song (workers = 4). Việc huấn luyện mô hình diễn ra trên GPU (thiết bị 'cuda'), tận dụng khả năng tính toán mạnh mẽ của nó.

Các siêu tham số quan trọng như **batch size** được thiết lập là 32, nghĩa là trong mỗi vòng lặp huấn luyện, 32 hình ảnh sẽ được xử lý đồng thời. **Learning rate** – tốc độ học – được đặt ở mức 0.00025 để điều chỉnh việc cập nhật trọng số trong quá trình tối ưu hóa. Em thiết lập số bước huấn luyện - **iter** là 2000 bước, đồng thời thêm **weight decay** 0.0001 nhằm giảm thiểu hiện tượng overfitting. Để đảm bảo an toàn trong trường hợp xảy ra sự cố, mô hình sẽ lưu lại **checkpoint** sau mỗi 500 bước huấn luyện, giúp tránh mất mát dữ liệu.

Cuối cùng, với cấu hình **ROI Heads**, số lượng đề xuất (proposals) được đặt là 8, và mô hình được định nghĩa để phát hiện một lớp đối tượng chính là **"damage"** (hư hỏng). Sau khi hoàn tất quá trình huấn luyện, mô hình **"model_final"** sẽ được lưu tại thư mục đầu ra đã định trước, sẵn sàng cho các bước đánh giá và triển khai tiếp theo.

3.3.3. Suy luận và đánh giá mô hình

Sau khi huấn luyện mô hình với các thông số trên, tổng thời gian huấn luyện là 1 ngày 12 tiếng thì có được kết quả như sau:

```
...
[09/29 01:55:00 d2.evaluation.testing]: copypaste: 11.3470,29.1342,7.0485,1.5808,5.8269,15.3005
[09/29 01:55:00 d2.evaluation.testing]: copypaste: Task: segm
[09/29 01:55:00 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[09/29 01:55:00 d2.evaluation.testing]: copypaste: 8.7914,24.1108,4.8475,0.5258,2.5295,13.3555
```

Kết quả trên là kết quả tốt nhất trong quá trình em huấn luyện. Đã có nhiều kết quả trước đó nhưng vẫn không đảm bảo được chất lượng, cụ thể chi tiết:

```
...
[09/11 22:26:26 d2.evaluation.testing]: copypaste: 6.0030,14.8125,3.9613,4.3255,11.8824,14.0539
[09/11 22:26:26 d2.evaluation.testing]: copypaste: Task: segm
[09/11 22:26:26 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[09/11 22:26:26 d2.evaluation.testing]: copypaste: 4.4013,10.9637,2.8509,2.2946,6.4986,15.9240


[09/19 22:11:52 d2.evaluation.testing]: copypaste: 9.9238,23.1145,7.1698,2.6766,7.1007,27.5456
[09/19 22:11:52 d2.evaluation.testing]: copypaste: Task: segm
[09/19 22:11:52 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[09/19 22:11:52 d2.evaluation.testing]: copypaste: 7.6945,19.2838,5.0575,1.0507,4.2695,22.6627
```

…..

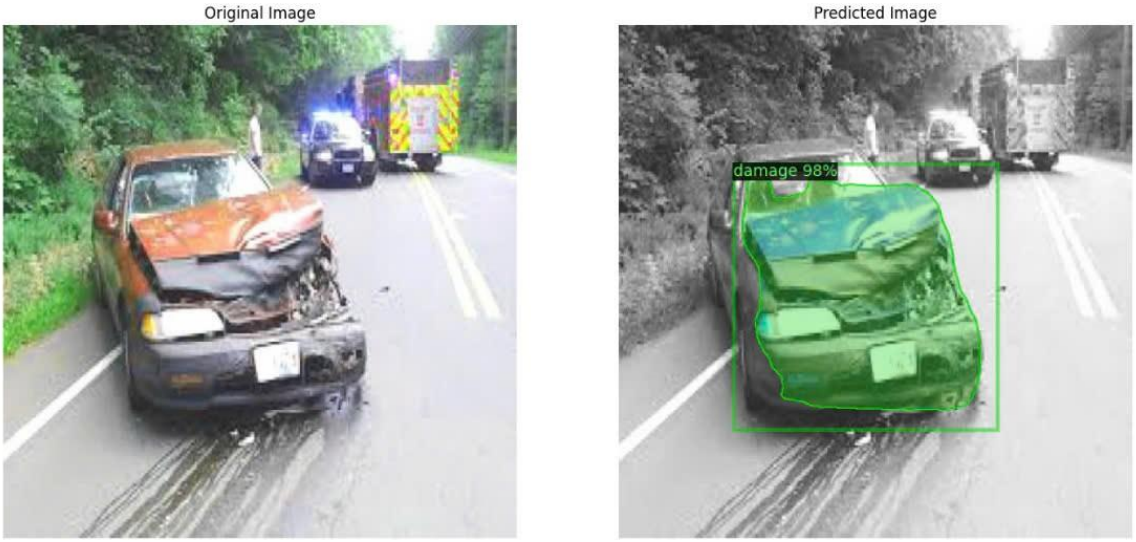
Dataset (img)	**Cấu hình**	**AP50 (bbox)**	**AP50 (segm)**
Train: 6288 Val: 1688	Batch: 4 Iter: 800	21.5553	18.0396
Train: 6288 Val: 1688	Batch: 32 Iter: 2000	24.2300	19.1894
…	…	…	…
Train: 16447 Val: 2648	Batch: 16 Iter: 2000	22.7487	18.4936
Train: 16447 **Val: 2648**	**Batch: 32** **Iter: 2000**	**29.1342**	**24.1108**

Bảng 3-1: Bảng chi tiết quá trình huấn luyện

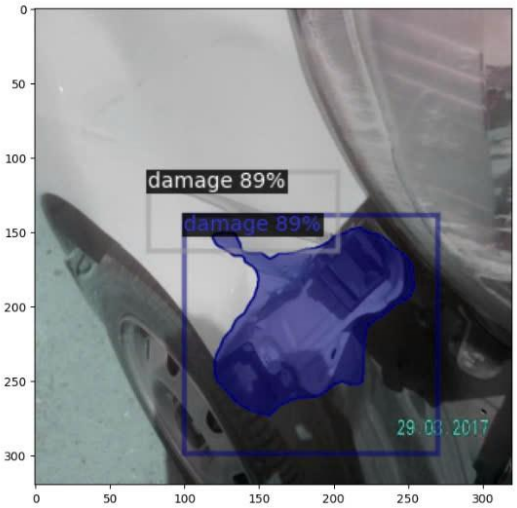
	AP	**AP50**	**AP75**	**APs**	**APm**	**APl**
bbox	11.3470	29.1342	7.0485	1.5808	5.8269	15.3005
segm	8.7914	24.1108	4.8475	0.5258	2.5295	13.3555

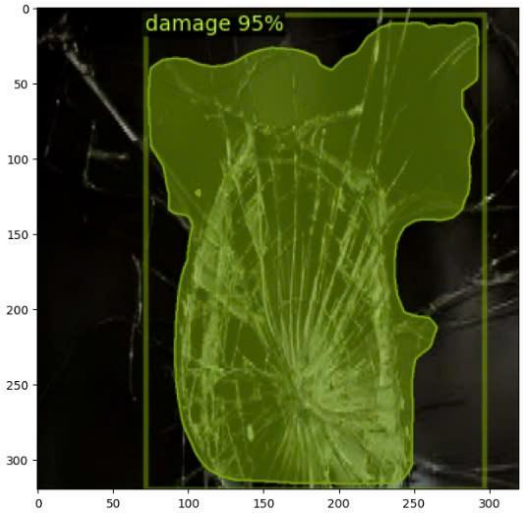
Bảng 3-2: Bảng kết quả chi tiết của mô đun cuối cùng

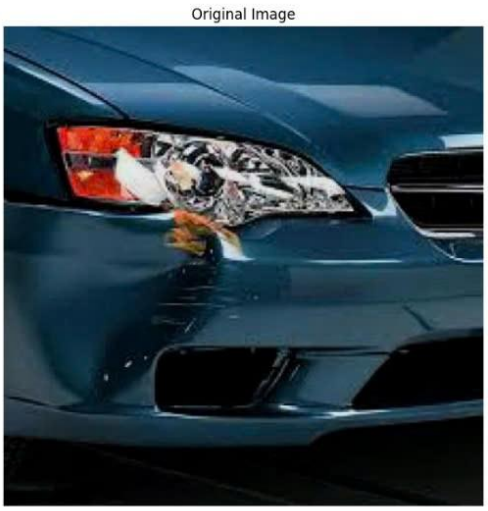
Kết quả trực quan hóa lên bộ dữ liệu val:

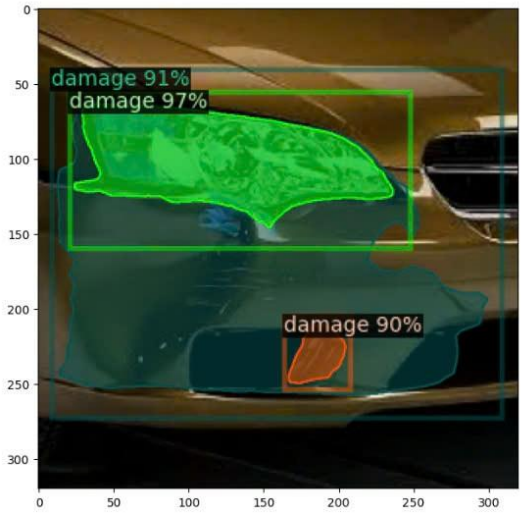


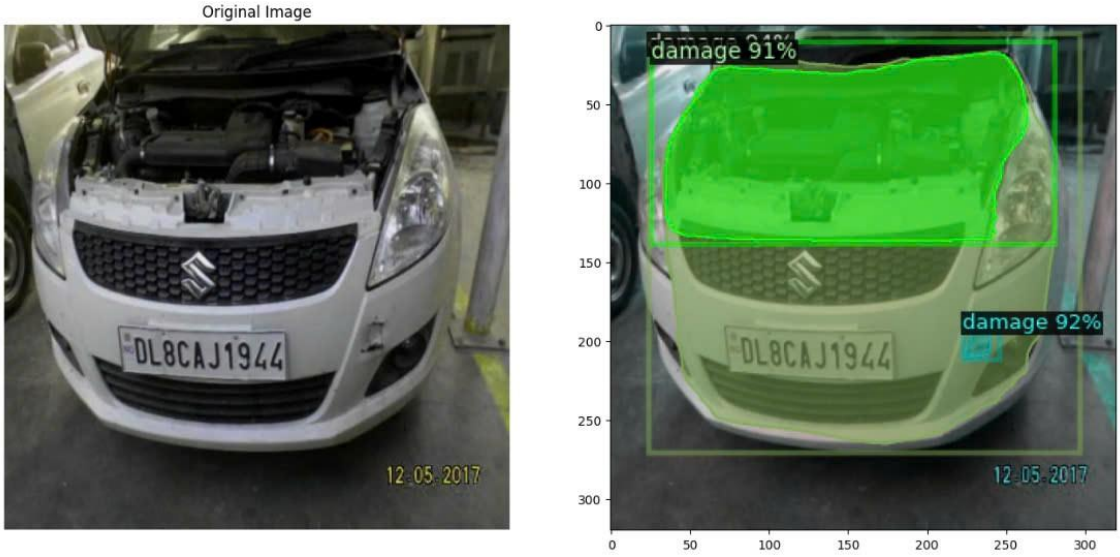
Kết quả trực quan hóa lên bộ dữ liệu test:











Nhận xét: Dựa vào kết quả trực quan từ các hình ảnh dự đoán của hai tập dữ liệu val và test thì có thể đánh giá sơ bộ như sau: Mô hình đã thành công trong việc nhận diện các vùng hư hỏng chính và khoanh được các vùng bằng các khung chữ nhật cũng như tạo mặt nạ phân đoạn khu vực bị hư hỏng. Đối với các vùng hư hỏng lớn, mô hình đưa ra dự đoán với xác suất cao, thể hiện sự tự tin trong việc nhận diện những chi tiết nổi bật. Tuy nhiên, với các hư hỏng nhỏ hoặc ít rõ ràng, mô hình vẫn bỏ sót hoặc không nhận diện chính xác cao, cho thấy khả năng của mô hình trong việc phát hiện cấc chi tiết tinh vi còn hạn chế. Điều này gợi ý rằng, mặc dù mô hình có khả năng nhận diện tốt các đặc điểm lớn, nó chưa hoàn toàn đủ mạnh mẽ để xử lý các chi tiết nhỏ.

Mặc dù đã đạt được một số tiến bộ sau nhiều lần huấn luyện, mô hình vẫn chưa đạt được hiệu suất như mong muốn. Em nhận thấy rằng mô hình hoạt động tốt hơn khi tăng số lượng dữ liệu huấn luyện và điều chỉnh batch size. Tuy nhiên, hạn chế lớn nhất là dung lượng bộ nhớ GPU. Do máy tính của em chỉ có 16GB RAM và quá trình huấn luyện được thực hiện trên môi trường local, em gặp phải tình trạng đầy bộ nhớ GPU khi cố gắng tăng cường thêm dữ liệu và điều chỉnh cấu hình để cải thiện mô hình.

Việc huấn luyện tốn rất nhiều thời gian, mỗi lần từ 1 đến 2 ngày, và mặc dù em đã thử các cấu hình tối ưu từ nhiều nguồn tham khảo quốc tế, hiệu suất cho bài toán phân đoạn đối tượng này vẫn chỉ dừng lại ở mức dưới 30% [17] [18] [19] [20]. Do đó, em đã quyết định chốt kết quả cuối cùng với tập dữ liệu huấn luyện 16,447 ảnh và tập

kiểm tra 2,648 ảnh, đạt được hiệu suất AP(50) là 29.1342 cho dự đoán bounding box (bbox) và 24.1108 cho phân đoạn đối tượng (segm).

Kết quả này, dù chưa hoàn hảo, vẫn phản ánh những nỗ lực đáng kể trong việc giải quyết bài toán, đồng thời cho thấy thách thức lớn của bài toán này và những giới hạn về tài nguyên phần cứng trong quá trình huấn luyện mô hình.

3.3.4. Triển khai lên hệ thống Web

Sau khi hoàn tất quá trình huấn luyện và đạt được mô hình cuối cùng, bước tiếp theo trong dự án là phát triển mô hình lên hệ thống web. Việc phát triển này không chỉ giúp cho người dùng dễ dàng truy cập và sử dụng hình ảnh mà vẫn tạo ra một giao diện trực quan để hiển thị kết quả phát hiện hư hỏng ô tô.

Để phát triển hệ thống web, em sử dụng Python Flask cho phép xây dựng ứng dụng web nhanh chóng và tích hợp dễ dàng cho các mô hình học sâu đã được huấn luyện.

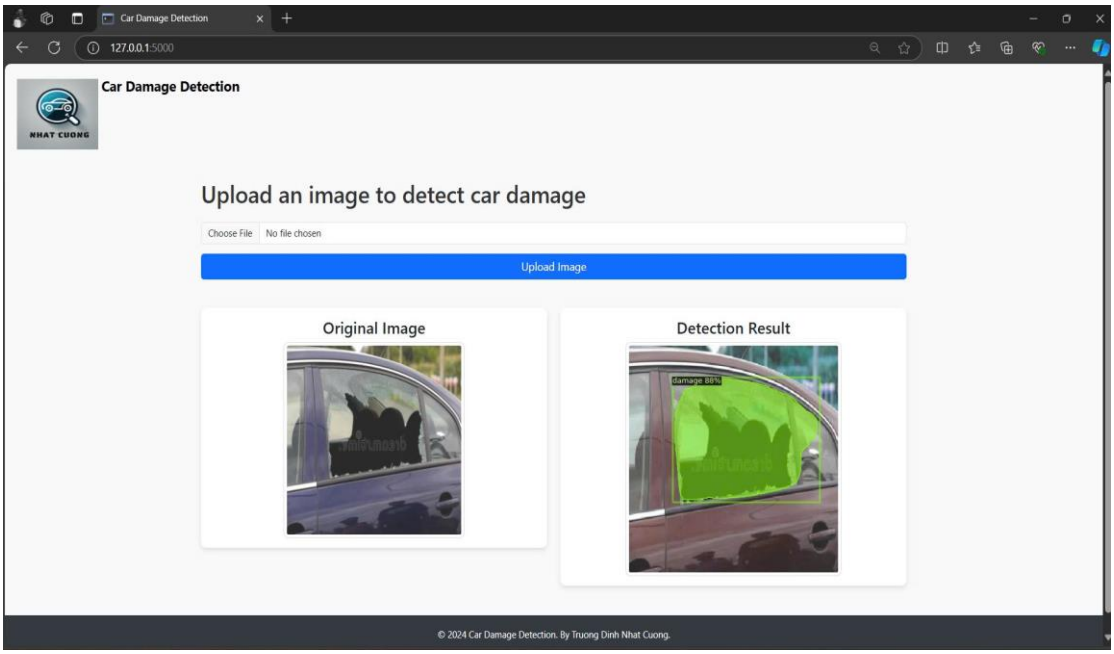
Cấu trúc dự án triển khai web:

Web_RealizationOF_CarDamageModel

- **model/**: Chứa các tệp liên quan đến mô hình Detectron2.
    - config.yaml: Tệp cấu hình của mô hình.
    - model_final.pth: Trọng số cuối cùng của mô hình sau khi huấn luyện.
- **static/**: Chứa các tệp tĩnh như hình ảnh, tệp CSS, và các tệp upload.
    - **uploads/**: Thư mục chứa các hình ảnh được người dùng tải lên và các hình ảnh kết quả sau khi dự đoán hư hỏng.
        - Các tệp hình ảnh được tải lên và kết quả dự đoán của chúng.
    - LogoNC.png: Logo của trang web.
    - style.css: Tệp CSS để định dạng giao diện trang web.
- **templates/**: Chứa các tệp HTML cho giao diện trang web.
    - index.html: Trang chính để người dùng tải lên hình ảnh.
    - result.html: Trang hiển thị kết quả của việc dự đoán.

- **app.py**: Tệp chính của ứng dụng Flask, nơi xử lý logic tải lên hình ảnh, gọi mô hình Detectron2 để dự đoán, và hiển thị kết quả.

Giao diện trang web:



Hình 3-12: Giao diện trang web

Chương 4. **Kết luận và hướng phát triển**

4.1. Kết luận

Trong quá trình thực hiện và nghiên cứu đồ án ngành, em đã có cơ hội tiếp cận sâu hơn với lĩnh vực thị giác máy tính và học máy, đặc biệt là việc ứng dụng mạng nơ-ron tích chập (CNN). Qua đó, em không chỉ nắm vững các kiến thức lý thuyết mà còn thực hành thành thạo quy trình xây dựng và triển khai một mô hình học sâu hiệu quả. Thông qua dự án, em đã nghiên cứu và áp dụng thành công các kỹ thuật tiên tiến, đặc biệt là mô hình Mask R-CNN của Detectron2 – một công cụ mạnh mẽ cho việc phát hiện và phân đoạn đối tượng trong hình ảnh. Dự án đã mang lại nhiều kết quả đáng khích lệ, góp phần giải quyết bài toán phát hiện hư hỏng trên xe ô tô một cách chính xác và hiệu quả.

Các kết luận chính từ dự án được tóm tắt như sau:

4.1.1. Kết quả đạt được

Phát triển mô hình phát hiện hư hỏng ô tô: Dự án đã thành công trong việc phát triển một mô hình phát hiện và phân đoạn hư hỏng ô tô dựa trên nền tảng Detectron2. Mô hình có khả năng nhận diện các vùng hư hỏng lớn trên xe ô tô như thân xe móp, đèn xe bị vỡ, hoặc các chi tiết rõ ràng trên bề mặt xe. Nhờ vào việc huấn luyện trên tập dữ liệu tùy chỉnh, mô hình đã đạt được hiệu suất nhận diện tốt đối với các hư hỏng có kích thước lớn, tạo cơ sở vững chắc để triển khai vào thực tiễn.

Tích hợp và triển khai web: Mô hình phát hiện hư hỏng ô tô đã được tích hợp vào một ứng dụng web sử dụng Flask, cho phép người dùng tải lên hình ảnh xe và nhận kết quả dự đoán ngay lập tức trên trình duyệt. Kết quả dự đoán hiển thị trực quan các vùng hư hỏng trên hình ảnh, mang lại trải nghiệm người dùng mượt mà và thuận tiện, mở ra tiềm năng ứng dụng trong các hệ thống bảo hiểm xe cộ hoặc garage sửa chữa tự động.

Chỉ số đánh giá: Mô hình đã đạt được các chỉ số đánh giá cơ bản như AP (Average Precision) và IoU (Intersection over Union) ở mức trung bình khá, đặc biệt trong việc nhận diện các hư hỏng lớn. Mặc dù còn hạn chế đối với các chi tiết nhỏ, nhưng các chỉ số này cho thấy mô hình đã đạt yêu cầu cơ bản về độ chính xác, đảm bảo khả năng tổng quát hóa cho nhiều loại hư hỏng xe ô tô.

4.1.2. Những điểm còn hạn chế

Mặc dù mô hình đã đạt được kết quả khả quan, nhưng vẫn tồn tại một số hạn chế nhất định:

- **Hiệu suất nhận diện**: Chỉ số đánh giá của mô hình chỉ đạt mức trung bình, chưa cao. Mặc dù các hư hỏng lớn có thể được nhận diện tương đối chính xác, nhưng đối với những chi tiết nhỏ hoặc phức tạp hơn như các vết trầy xước nhẹ, mô hình vẫn còn khó khăn trong việc nhận diện và phân đoạn chính xác. Điều này có thể do dữ liệu huấn luyện chưa được đa dạng hóa và thiếu các mẫu dữ liệu về các loại hư hỏng nhỏ hoặc hiếm gặp.
- **Dữ liệu huấn luyện chưa tối ưu**: Tập dữ liệu sử dụng để huấn luyện mô hình còn hạn chế về số lượng và tính đa dạng. Các hình ảnh hư hỏng chưa bao quát hết các trường hợp có thể xảy ra trong thực tế, đặc biệt là các hư hỏng khó phát hiện bằng mắt thường hoặc các chi tiết nhỏ, như vết nứt nhỏ, vết xước nhẹ.
- **Thời gian huấn luyện dài và yêu cầu tài nguyên lớn**: Một trong những hạn chế lớn nhất trong quá trình thực hiện đồ án đó là thời gian huấn luyện rất dài, kéo dài từ 1 đến 2 ngày cho mỗi lần huấn luyện mô hình, điều này đòi hỏi nhiều tài nguyên, đặc biệt là GPU với bộ nhớ lớn. Việc sử dụng bộ nhớ GPU nhiều dẫn đến thời gian chờ đợi dài và khó khăn trong việc liên tục tinh chỉnh mô hình.

4.2. Hướng phát triển

Để cải thiện và phát triển hệ thống trong tương lai, có một số hướng đi chính có thể thực hiện:

- **Mở rộng và tối ưu hóa dữ liệu huấn luyện**: Cần bổ sung thêm các hình ảnh hư hỏng xe với nhiều dạng kích thước và mức độ chi tiết khác nhau để mô hình có thể học tốt hơn các trường hợp hư hỏng nhỏ hoặc hiếm gặp. Việc thu thập thêm dữ liệu từ các nguồn khác nhau, đảm bảo tính đa dạng về loại hư hỏng, góc chụp, môi trường và điều kiện ánh sáng sẽ giúp cải thiện đáng kể hiệu suất của mô hình. Ngoài ra, có thể sử dụng các kỹ thuật data augmentation (phóng to, thu nhỏ, xoay, làm mờ) để tăng cường tập dữ liệu và làm mô hình học tốt hơn trong những điều kiện khác nhau.

- **Tinh chỉnh và tối ưu mô hình**: Sử dụng các kỹ thuật tiên tiến để tối ưu hóa việc huấn luyện như giảm kích thước batch size, điều chỉnh lại các siêu tham số (hyperparameters) như learning rate, weight decay để giúp mô hình hội tụ nhanh hơn và đạt hiệu suất cao hơn. Đồng thời, có thể cân nhắc việc sử dụng các mô hình khác có độ chính xác cao hơn hoặc tích hợp thêm các kỹ thuật như learning rate scheduler, gradient clipping để ổn định quá trình huấn luyện.
- **Tối ưu hóa tài nguyên và rút ngắn thời gian huấn luyện**: Thời gian huấn luyện hiện tại là một trở ngại lớn, do đó có thể tìm kiếm các phương pháp như huấn luyện phân tán trên nhiều GPU hoặc sử dụng dịch vụ cloud computing để tối ưu tài nguyên. Bên cạnh đó, việc triển khai mô hình với kỹ thuật Mixed Precision Training (huấn luyện với độ chính xác pha trộn) có thể giảm lượng bộ nhớ tiêu thụ, từ đó tăng tốc độ huấn luyện. Một số giải pháp khác bao gồm sử dụng các phương pháp giảm nhẹ trọng số mô hình hoặc sử dụng các mô hình nhẹ hơn để tăng tốc độ dự đoán.
- **Ứng dụng thực tiễn**: Tiếp tục phát triển hệ thống ứng dụng phát hiện hư hỏng ô tô, không chỉ dừng lại ở web-based mà còn mở rộng ra các nền tảng di động hoặc tích hợp vào các hệ thống quản lý bảo hiểm, kiểm tra xe tại các garage để tự động phát hiện và đánh giá hư hỏng, giúp tiết kiệm thời gian và tăng tính chính xác trong quy trình sửa chữa và bảo trì xe cộ.

TÀI LIỆU THAM KHẢO

1. I. H. Sarker, Y. B. Abushark, A. I. Khan, M. M. Alam, and R. Nowrozy, "Mobile deep learning: Exploring deep neural network for predicting context-aware smartphone usage," *Journal of Big Data*, vol. 8, no. 1, pp. 1–22, 2021

2. R. Venkatesan and B. Li, *Convolutional Neural Networks in Visual Computing: A Concise Guide*. Boca Raton, FL, USA: CRC Press, 2017. ISBN: 978-1-351-65032-8.

3. R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, Jan. 2016, doi: 10.1109/TPAMI.2015.2437384.

4. T. V. Huy, "Fast R-CNN Understanding," *GitHub Pages*, 20-Jul-2021. [Online]. Available: https://huytranvan2010.github.io/Fast-R-CNN-Understanding/. [Accessed: 01-Oct-2024].

5. K. Patnaik, "Annotated RPN, ROI Pooling and ROI Align," *GitHub Pages*, 04-Jul-2020. [Online]. Available: https://kaushikpatnaik.github.io/annotated/papers/2020/07/04/ROI-Pool-and-Align-Pytorch-Implementation.html. [Accessed: 01-Oct-2024].

6. K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, Oct. 2017, doi: 10.1109/ICCV.2017.322. ISBN: 978-1-5386-1032-9.

7. W. Abdulla, "Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow," *Matterport Engineering*, 20-Mar-2018. [Online]. Available: https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46. [Accessed: 01-Oct-2024].

8. T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *arXiv*, vol. 2017, no. 1, pp. 1-14, 2017. [Online]. Available: https://arxiv.org/abs/1612.03144. [Accessed: 02-Oct-2024].

9. Firiuza, "ROI pooling vs. ROI align," *Medium*, 08-Jan-2020. [Online]. Available: https://firiuza.medium.com/roi-pooling-vs-roi-align-65293ab741db. [Accessed: 02-Oct-2024].

10. A. Rosebrock, "Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning," *PyImageSearch*, 05-Oct-2020. [Online]. Available: https://pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning. [Accessed: 02-Oct-2024].

11. Facebook Research, "Detectron2," *GitHub*, 2019. [Online]. Available: https://github.com/facebookresearch/detectron2. [Accessed: 06-Oct-2024].

12. Roboflow user, "Car Damage V5," *Roboflow Universe*, generated on Oct. 17, 2023. [Online]. Available: https://universe.roboflow.com/car-damage-kadad/car-damage-v5/dataset/6. [Accessed: 01-Oct-2024].

13. Roboflow user, "Car Damage Images," *Roboflow Universe*, generated on Mar. 29, 2023. [Online]. Available: https://universe.roboflow.com/car-damage-kadad/car-damage-images/dataset/3. [Accessed: 01-Oct-2024].

14. Roboflow user, "Car Damage Detection SKLXM," *Roboflow Universe*, generated on Jul. 11, 2024. [Online]. Available: https://universe.roboflow.com/aiclub-pcphh/car-damage-detection-sklxm/dataset/3. [Accessed: 01-Oct-2024].

15. Roboflow user, "Car Damages V3GYZ," *Roboflow Universe*, generated on Feb. 28, 2023. [Online]. Available: https://universe.roboflow.com/project-p5nyc/car-damages-v3gyz/dataset/5. [Accessed: 01-Oct-2024].

16. Roboflow user, "Car Damage Detection HA5MM," *Roboflow Universe*, generated on Apr. 4, 2023. [Online]. Available: https://universe.roboflow.com/college-gxdrt/car-damage-detection-ha5mm/dataset/1. [Accessed: 01-Oct-2024].

17. L. Plenka, "Detectron2 Car Damage Detection," *Kaggle*, 4 years ago. [Online]. Available: https://www.kaggle.com/code/lplenka/detectron2-car-damage-detection/notebook. [Accessed: 04-Oct-2024].

18. A. Wu, "Detectron2 Car Damage Detection," *Kaggle*, 3 years ago. [Online]. Available: https://www.kaggle.com/code/arashilen/detectron2-car-damage-detection. [Accessed: 04-Oct-2024].

19. Whitish, "Car Damage Detection with Detectron2," *Kaggle*, 6 months ago. [Online]. Available: https://www.kaggle.com/code/whitish/car-damage-detection-with-detectron2/notebook. [Accessed: 04-Oct-2024].

20. E. Osorio, "Detectron2 Car Damage Detection," *Kaggle*, 3 years ago. [Online]. Available: https://www.kaggle.com/code/emilioosorioemilio/detectron2-car-damage-detection. [Accessed: 04-Oct-2024].