

TRƯỜNG ĐẠI HỌC NGOẠI NGỮ-TIN HỌC TP. HỒ CHÍ MINH  
KHOA: CÔNG NGHỆ THÔNG TIN

## KHÓA LUẬN TỐT NGHIỆP

# Nghiên cứu và cải tiến Mô hình nhận diện hình ảnh YOLOV8

GIẢNG VIÊN HƯỚNG DẪN: Phan Thị Ngọc Hân

SINH VIÊN THỰC HIỆN: Nguyễn Cao Hoàng - 21DH112490

TP. HỒ CHÍ MINH – 8-2024

## Lời cảm ơn

Trước tiên, nhóm chúng em gửi lời cảm ơn chân thành và sâu sắc đến cô Phan Thị Ngọc Hân, là giảng viên đã tận tình hướng dẫn, chỉ bảo, giúp đỡ em trong suốt thời gian em nghiên cứu khóa luận. Và cũng là người đưa ra những ý tưởng, kiểm tra sự phù hợp của khóa luận.

Em cũng xin gửi lời cảm ơn đến toàn thể giảng viên trường Đại học Ngoại ngữ Tin học thành phố Hồ Chí Minh, trong thời gian qua đã giảng dạy, và tạo điều kiện cho chúng em trong quá trình học tập và nghiên cứu tại trường. Những kiến thức mà chúng em nhận được sẽ là hành trang giúp chúng em vững bước trong tương lai.

Cuối cùng, em xin cảm ơn gia đình, bạn bè, người thân đã luôn ở bên để động viên và là nguồn cổ vũ lớn lao, là động lực giúp em hoàn thành khóa luận này. Mặc dù đã cố gắng hoàn thành khóa luận trong phạm vi và khả năng có thể. Tuy nhiên sẽ không tránh khỏi những thiếu sót. Em rất mong nhận được sự cảm thông và tận tình chỉ bảo của quý thầy cô và toàn thể các bạn.

Em xin chân thành cảm ơn.

## **Lời cam đoan**

Em xin cam đoan đề tài này là một công trình nghiên cứu độc lập với toàn bộ nội dung và kết quả là sản phẩm mà em đã nỗ lực nghiên cứu trong quá trình học tập tại trường cũng như dưới sự dẫn dắt của giảng viên hướng dẫn.

Trong quá trình viết bài có sự tham khảo một số tài liệu có nguồn gốc rõ ràng và đã được trích dẫn đầy đủ.

Em xin cam đoan nếu có vấn đề gì em xin chịu hoàn toàn trách nhiệm.

## MỤC LỤC

Lời cảm ơn .....	i
Lời cam đoan.....	ii
Mục lục.....	iii
Danh mục các từ viết tắt.....	vi
Danh mục các hình vẽ.....	vii
Danh mục các bảng .....	xii
Chương 1: Mở đầu .....	12
1.1 Lý do chọn đề tài .....	12
1.2 Nội dung nghiên cứu.....	12
Chương 2: Tổng quan .....	13
2.1 Lịch sử YOLO .....	13
2.1.1 YOLO (You Only Look Once) - 2015 .....	13
2.1.2 YOLOv2 (YOLO9000) - 2016.....	13
2.1.3 YOLOv3 - 2018.....	13
2.1.4 YOLOv4 - 2020.....	14
2.1.5 Scaled-YOLOv4 - 2020.....	14
2.1.6 YOLOv5 - 2020.....	14
2.1.7 PP-YOLO - 2020.....	15
2.1.8 PP-YOLOv2 - 2021 .....	15
2.1.9 YOLOR - 2021 .....	15
2.1.10 YOLOX - 2021.....	15
2.1.11 YOLOv6 - 2022.....	16
2.1.12 YOLOv7 - 2022.....	16
2.1.13 PP-YOLOE - 2022 .....	16
2.1.14 YOLOv8 - 2023.....	17
2.2 Kiến trúc YOLOv8 .....	17
2.2.1 Tổng quan.....	17
2.2.2 Cải tiến quan trọng .....	21
2.2.3 Ưu, nhược điểm.....	22

Chương 3: Phương pháp đề xuất.....	23
3.1 Điều chỉnh siêu tham số.....	23
3.1.1 Learning Rate (Tốc độ học).....	23
3.1.2 Batch Size.....	24
3.1.3 Epochs .....	24
3.1.4 Optimizer (Bộ tối ưu hóa): .....	25
3.1.5 Momentum .....	25
3.1.6 Dropout Rate .....	25
3.1.7 Hiệu quả của sự thay đổi các siêu tham số.....	26
3.2 Tăng cường dữ liệu.....	26
3.2.1 HSV (Hue, Saturation, Value) Augmentation (hsv_h, hsv_s, hsv_v).....	26
3.2.2 Degrees (Rotation) .....	27
3.2.3 Translate .....	28
3.2.4 Scale .....	28
3.2.5 Mosaic .....	29
3.2.6 Mixup .....	29
3.2.7 Flipud (Flip Up-Down) .....	30
3.2.8 Fliplr (Flip Left-Right) .....	30
3.2.9 Shear .....	31
3.2.10 Perspective.....	31
3.2.11 Ảnh hưởng của tăng cường dữ liệu .....	32
3.3 Thay đổi kiến trúc YOLOv8.....	32
3.4 TensorRT .....	39
3.4.1 Khái niệm .....	39
3.4.2 Ưu, nhược điểm .....	40
Chương 4: Thực nghiệm .....	42
4.1 Thiết lập thực nghiệm .....	42
4.1.1 Chuẩn bị Dữ Liệu: .....	42
4.1.2 Huấn luyện mô hình nguyên bản của YOLOv8.....	42
4.1.3 Điều chỉnh siêu tham số .....	45

4.1.4 Tăng cường dữ liệu.....	48
4.1.5 Thay đổi kiến trúc YOLOv8 .....	50
4.1.6 TensorRT.....	58
4.2 Kết quả .....	60
Chương 5: Kết luận .....	65
TÀI LIỆU THAM KHẢO.....	66

## Danh mục các từ viết tắt

STT	Từ viết tắt	Mô tả
1	API	Application Programming Interface – phương thức trung gian kết nối

## Danh mục các hình vẽ

Hình 1: Kiến trúc YOLOv8.....	18
Hình 2: Ảnh hưởng của learning rate.....	23
Hình 3: Tìm kiếm learning rate thích hợp.....	23
Hình 4: SGD với momentum .....	25
Hình 5: Box plot và hình ảnh minh họa của các giá trị hsv_h, hsv_s, hsv_v ..	27
Hình 6: Kĩ thuật quay ảnh .....	28
Hình 7: Kĩ thuật translate .....	28
Hình 8: Kĩ thuật scale.....	29
Hình 9: Kĩ thuật mosiac .....	29
Hình 10: Kĩ thuật mixup .....	30
Hình 11: Kĩ thuật flipup .....	30
Hình 12: Kĩ thuật fliplr.....	31
Hình 13: Kĩ thuật shear .....	31
Hình 14: Kĩ thuật perspective .....	32
Hình 15: Những thay đổi kiến trúc YOLOv8 cho kích thước nhỏ .....	33
Hình 16: Kiến trúc YOLOv8 cho kích thước nhỏ.....	34
Hình 17: Những thay đổi kiến trúc YOLOv8 cho kích thước trung bình.....	35
Hình 18: Kiến trúc YOLOv8 cho kích thước trung bình .....	36
Hình 19: Những thay đổi kiến trúc YOLOv8 cho kích thước lớn .....	37
Hình 20: Kiến trúc YOLOv8 cho kích thước lớn .....	38
Hình 21: Sử dụng API để huấn luyện fm_original .....	42

Hình 22: Kết quả huấn luyện của fm_original .....	43
Hình 23: Các chỉ số đo độ chính xác của fm_original .....	43
Hình 24: Tốc độ của fm_original .....	43
Hình 25: Sử dụng API để huấn luyện fruit_original .....	44
Hình 26: Kết quả huấn luyện của fruit_original .....	44
Hình 27: Các chỉ số đo độ chính xác của fruit_original.....	44
Hình 28: Tốc độ của fruit_original .....	45
Hình 29: Sử dụng API để huấn luyện fm_v1 .....	45
Hình 30: Kết quả huấn luyện của fruit_original .....	45
Hình 31: Các chỉ số đo độ chính xác của fruit_original.....	46
Hình 32: Tốc độ của fruit_original .....	46
Hình 33: Sử dụng API để huấn luyện fruit_v1 .....	46
Hình 34: Kết quả huấn luyện của fruit_v1 .....	47
Hình 35: Các chỉ số đo độ chính xác của fruit_v1 .....	47
Hình 36: Tốc độ của fruit_v1 .....	47
Hình 37: Sử dụng API để huấn luyện fm_v2.....	48
Hình 38: Kết quả huấn luyện của fm_v2 .....	48
Hình 39: Các chỉ số đo độ chính xác của fm_v2 .....	48
Hình 40: Tốc độ của fm_v2 .....	49
Hình 41: Sử dụng API để huấn luyện fruit_v3 .....	49
Hình 42: Kết quả huấn luyện của fruit_v3 .....	49

Hình 43: Các chỉ số đo độ chính xác của fruit_v3 .....	50
Hình 44: Tốc độ của fruit_v3 .....	50
Hình 45: Histogram plot của tập train thuộc tập fm .....	51
Hình 46: Histogram plot của tập val thuộc tập fm.....	51
Hình 47: Histogram plot của tập test thuộc tập fm .....	52
Hình 48: Nội dung file YOLOv8l-s_m.yaml .....	53
Hình 49: Sử dụng API để huấn luyện fm_v3 .....	53
Hình 50: Kết quả huấn luyện của fm_v3 .....	53
Hình 51: Các chỉ số đo độ chính xác của fm_v3 .....	54
Hình 52: Tốc độ của fm_v3 .....	54
Hình 53: Histogram plot của tập train thuộc tập fruit_fr .....	55
Hình 54: Histogram plot của tập val thuộc tập fruit_fr.....	55
Hình 55: Histogram plot của tập test thuộc tập fruit_fr .....	56
Hình 56: Nội dung file YOLOv8l-m.yaml.....	57
Hình 57: Sử dụng API để huấn luyện fruit_v2 .....	57
Hình 58: Kết quả huấn luyện của fruit_v2 .....	57
Hình 59: Các chỉ số đo độ chính xác của fruit_v2 .....	58
Hình 60: Tốc độ của fruit_v2 .....	58
Hình 61: Xuất fm_original sang kiểu TensorRT .....	58
Hình 62: Các chỉ số đo độ chính xác của fm_tensorrt .....	59
Hình 63: Tốc độ của fm_tensorrt .....	59

Hình 64: Xuất fruit_original sang kiểu TensorRT .....	59
Hình 65: Các chỉ số đo độ chính xác của fruit_tensorrt.....	60
Hình 66: Tốc độ của fruit_tensorrt.....	60
Hình 67: Biểu đồ so sánh các chỉ số đo độ chính xác của các model đào tạo từ fm	61
Hình 68: Biểu đồ so sánh tốc độ của các model đào tạo từ fm.....	62
Hình 69: Biểu đồ so sánh các chỉ số đo độ chính xác của các model đào tạo từ fruit_fr .....	63
Hình 70: Biểu đồ so sánh tốc độ của các model đào tạo từ fruit_fr.....	64

## **Danh mục các bảng**

Bảng 1: Các chỉ số đo độ chính xác của mô hình đào tạo từ fm.....	60
Bảng 2: Tốc độ của mô hình đào tạo từ fm.....	61
Bảng 3: Các chỉ số đo độ chính xác của mô hình đào tạo từ fruit_fr.....	62
Bảng 4: Các chỉ số đo độ chính xác của mô hình đào tạo từ fruit_fr.....	63

## Chương 1: Mở đầu

### 1.1 Lý do chọn đề tài

- Lý do chọn đề tài này là do sự phát triển vượt bậc của các mô hình YOLO (You Only Look Once) trong lĩnh vực nhận diện và phát hiện vật thể. YOLOv8 là phiên bản với nhiều cải tiến đáng kể so với các phiên bản cũ hơn có tốc độ xử lý nhanh hơn và độ chính xác cao, việc nghiên cứu và cải thiện mô hình này có thể đóng góp lớn cho các ứng dụng thực tế.
- Sự phổ biến của YOLO trong các lĩnh vực như an ninh, xe tự hành, và công nghiệp đã chứng minh tiềm năng của mô hình này. Việc cải thiện hiệu suất của YOLOv8 sẽ giúp nâng cao độ chính xác và tốc độ xử lý, từ đó ứng dụng rộng rãi hơn.

### 1.2 Nội dung nghiên cứu

- Nghiên cứu và hiểu rõ về mô hình YOLOv8.
- Thực hiện các thí nghiệm và cải thiện mô hình này.
- Xây dựng một demo minh họa về hiệu quả của các cải tiến đã thực hiện.

## Chương 2: Tổng quan

### 2.1 Lịch sử YOLO

- YOLO (You Only Look Once) là một dòng mô hình mạng nơ-ron nhân tạo nổi tiếng trong lĩnh vực phát hiện vật thể, được phát triển bởi Joseph Redmon và các đồng sự. Dưới đây là lịch sử phát triển của các phiên bản YOLO từ đầu tới phiên bản YOLOv8:

#### 2.1.1 YOLO (You Only Look Once) - 2015

- **Giới thiệu:** Joseph Redmon và cộng sự phát triển.

- **Đặc điểm nổi bật:**

- + Phương pháp thông nhất: sử dụng một mạng CNN duy nhất để dự đoán bounding boxes và class probabilities.
- + Xử lý tốc độ cao: 45 FPS.

#### 2.1.2 YOLOv2 (YOLO9000) - 2016

- **Giới thiệu:** Cải thiện độ chính xác và tốc độ.

- **Đặc điểm nổi bật:**

- + Chuẩn hóa theo batch.
- + Anchor boxes.
- + Bộ phân loại có độ phân giải cao.
- + YOLO9000: Kết hợp phát hiện đối tượng trên cả COCO và ImageNet.

#### 2.1.3 YOLOv3 - 2018

- **Giới thiệu:** Tăng cường khả năng phát hiện đối tượng nhỏ.

- **Đặc điểm nổi bật:**

- + Dự đoán đa kích thước.
- + Darknet-53.
- + Dự đoán bounding box bằng logistic regression.

#### 2.1.4 YOLOv4 - 2020

- **Giới thiệu:** Cải tiến đáng kể về tốc độ và độ chính xác.

- **Đặc điểm nổi bật:**

- + CSPDarknet53.
- + Bag of freebies và bag of specials.
- + Sử dụng nhiều kỹ thuật tối ưu hóa và tăng cường dữ liệu.

#### 2.1.5 Scaled-YOLOv4 - 2020

- **Giới thiệu:** Phiên bản mở rộng của YOLOv4.

- **Đặc điểm nổi bật:**

- + Kiến trúc đa mô hình (P5, P6, P7).
- + Hợp nhất đường dẫn đặc trưng (PAN).
- + Tăng cường Mosaic, đào tạo tự đối kháng.

#### 2.1.6 YOLOv5 - 2020

- **Giới thiệu:** Tối ưu hóa cho sử dụng trong thực tế.

- **Đặc điểm nổi bật:**

- + Triển khai trong PyTorch.
- + Cải thiện tăng cường dữ liệu.
- + Nhiều kích thước mô hình (s, m, l, x).

### 2.1.7 PP-YOLO - 2020

- **Giới thiệu:** Được phát triển bởi Baidu.

- **Đặc điểm nổi bật:**

+ ResNet50-vd.

+ Tối ưu hóa phần cứng.

+ Tăng cường dữ liệu MixUp, CutMix.

### 2.1.8 PP-YOLOv2 - 2021

- **Giới thiệu:** Phiên bản cải tiến của PP-YOLO.

- **Đặc điểm nổi bật:**

+ ResNet50-vd và CSPDarknet.

+ IoU Loss, Grid Sensitive, Matrix NMS.

+ Hiệu suất vượt trội.

### 2.1.9 YOLOR - 2021

- **Giới thiệu:** Kết hợp phát hiện đối tượng và phân loại.

- **Đặc điểm nổi bật:**

+ Học đại diện hợp nhất.

+ Phân loại và phát hiện đối tượng.

+ Hiệu suất cao.

### 2.1.10 YOLOX - 2021

- **Giới thiệu:** Mô hình YOLO tiên tiến.

- **Đặc điểm nổi bật:**

+ Không có neo.

+ Gán nhãn động.

+ Đa quy mô (s, m, l, x).

### 2.1.11 YOLOv6 - 2022

- **Giới thiệu:** Tập trung vào ứng dụng công nghiệp.

- **Đặc điểm nổi bật:**

+ Tăng cường dữ liệu nâng cao.

+ Cải thiện kiến trúc mạng.

+ Tập trung vào hiệu quả triển khai.

### 2.1.12 YOLOv7 - 2022

- **Giới thiệu:** Được cho là nhanh nhất và chính xác nhất trong dòng YOLO.

- **Đặc điểm nổi bật:**

+ E-ELAN.

+ Hàm mắt mát nâng cao.

+ Độ chính xác cao và hiệu suất thời gian thực.

### 2.1.13 PP-YOLOE - 2022

- **Giới thiệu:** Phiên bản mới của PP-YOLOv2.

- **Đặc điểm nổi bật:**

+ Kiến trúc mở rộng.

+ Mạng tổng hợp đường dẫn.

+ Hợp nhất tính năng không gian thích ứng.

+ Hiệu suất cao.

### 2.1.14 YOLOv8 - 2023

- **Giới thiệu:** Phiên bản nổi bật với khả năng kết hợp giữa tốc độ và độ chính xác.

- **Đặc điểm nổi bật:**

- + Cải tiến trong thiết kế mạng.
- + Tăng cường dữ liệu hiện đại.
- + Tập trung vào triển khai thực tế.

## 2.2 Kiến trúc YOLOv8

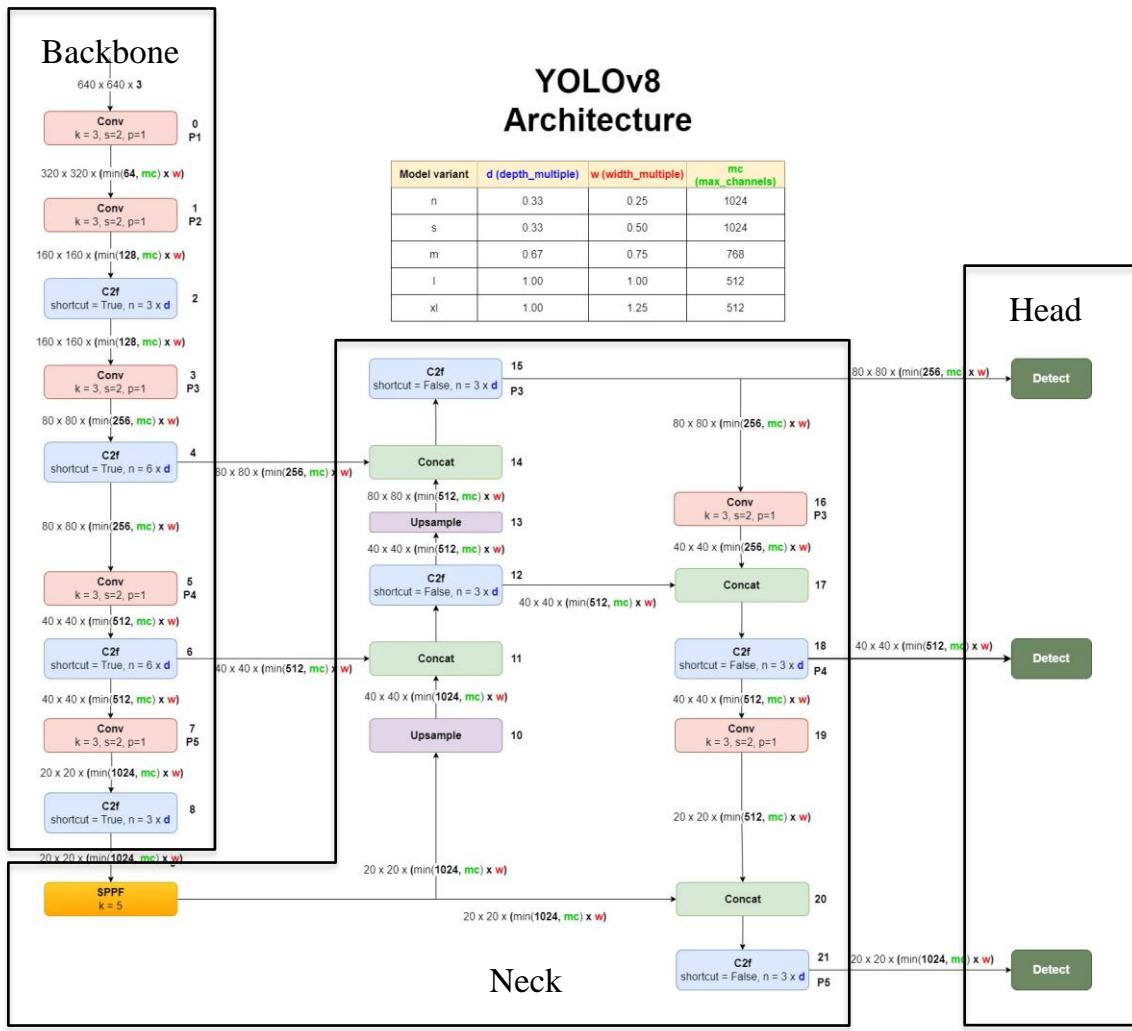
### 2.2.1 Tổng quan

- YOLOv8 là một phiên bản cải tiến đáng kể của dòng mô hình YOLO, với nhiều thay đổi và bổ sung quan trọng trong kiến trúc để nâng cao hiệu suất và độ chính xác. Kiến trúc YOLOv8 có thể được chia thành các phần chính sau:

- + Backbone: Là phần đầu vào của mạng, YOLOv8 sử dụng một phiên bản cải tiến của Darknet hoặc một mạng backbone khác như CSPNet để trích xuất các đặc trưng từ ảnh đầu vào.
- + Neck: Gồm các lớp convolution và các cơ chế attention để tiếp tục xử lý và kết hợp các đặc trưng được trích xuất từ Backbone.
- + Head: Phần cuối cùng của mạng, chứa các lớp phát hiện để dự đoán các hộp giới hạn và xác suất lớp cho mỗi vật thể trong ảnh. Các công thức toán học quan trọng:
  - + Hàm mất mát: YOLOv8 sử dụng một hàm mất mát tổng hợp bao gồm các thành phần cho lỗi vị trí (location loss), lỗi kích thước (size loss), lỗi phân loại (classification loss), và lỗi IOU (Intersection over Union loss).
  - + Hàm kích hoạt: Sử dụng các hàm kích hoạt như Leaky ReLU, Mish, và các biến thể khác để cải thiện khả năng học của mạng.

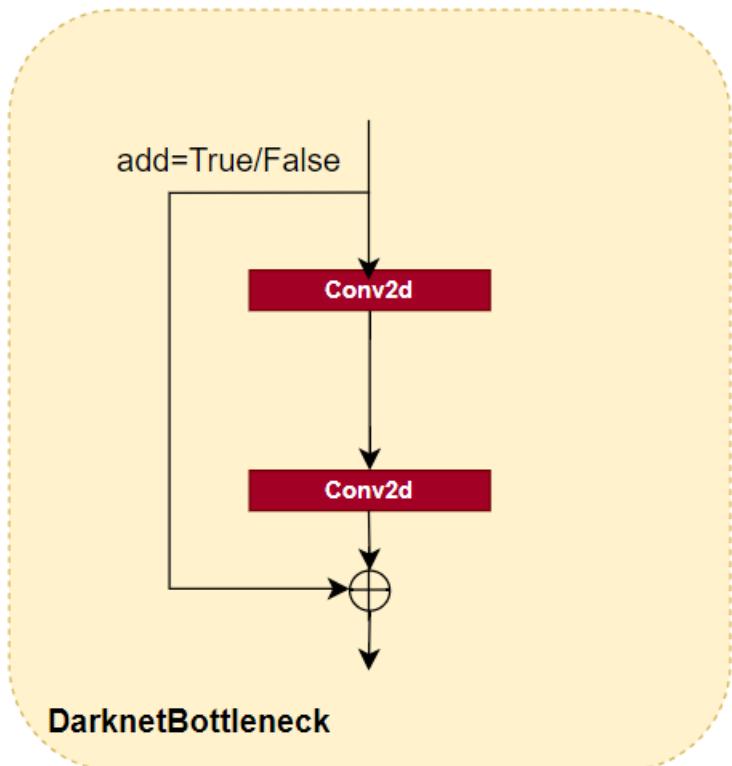
## Chương 2: Tổng quan

- + Cơ chế attention: Sử dụng các cơ chế như Squeeze-and-Excitation (SE) và Convolutional Block Attention Module (CBAM) để tăng cường khả năng chú ý vào các vùng quan trọng trong ảnh.
- Kiến trúc của mô hình YOLOv8 với 2 khối SPFF và C2f được truyền cảm hứng từ kiến trúc khối DarknetBottleneck:

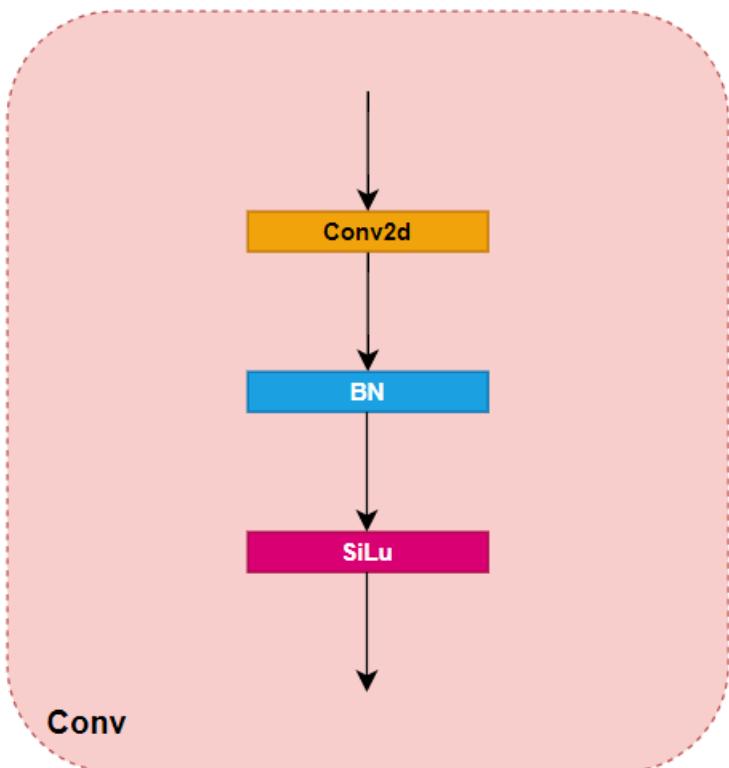


Hình 1: Kiến trúc YOLOv8.

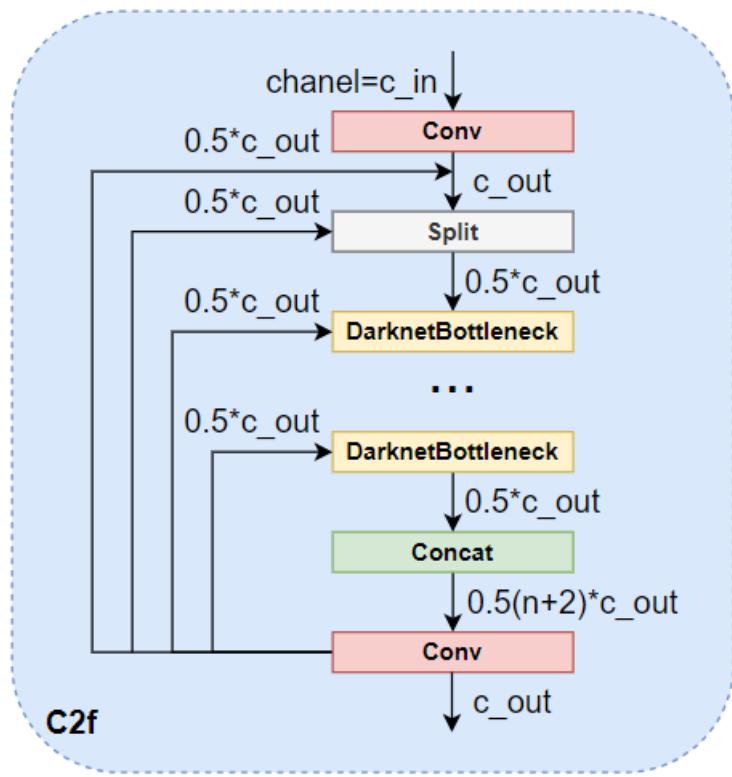
- + DarknetBottleneck:



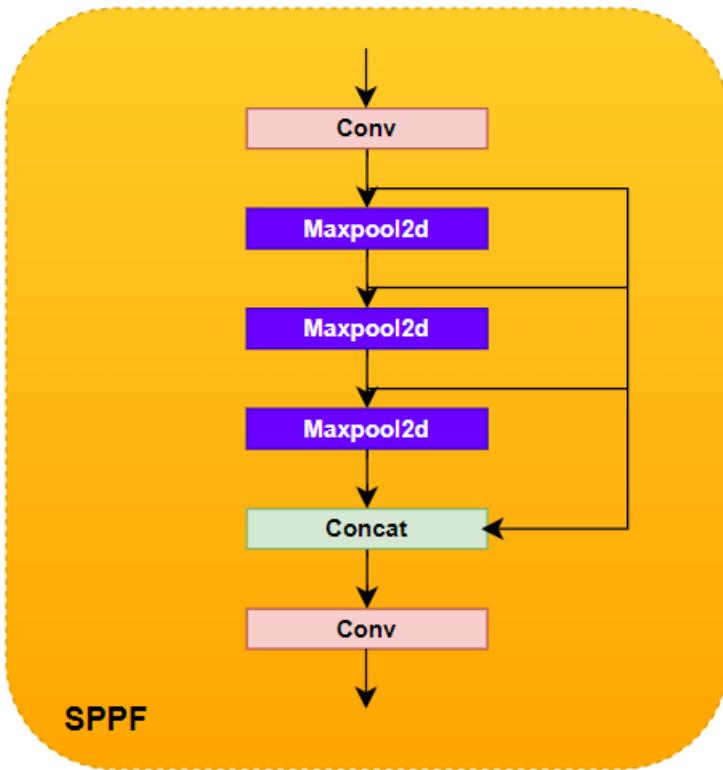
+ Conv:



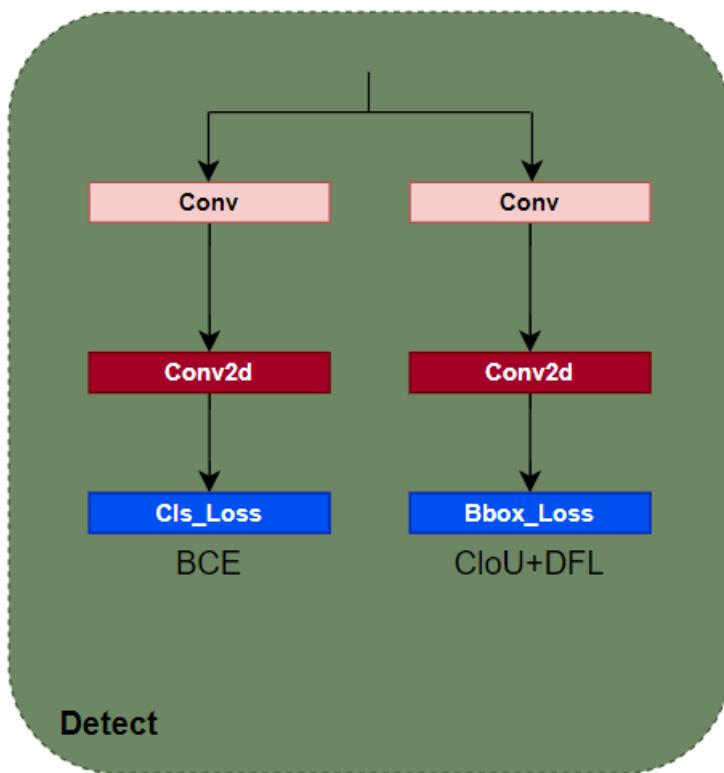
+ C2f:



+ SPFF:



+ Detect:



### 2.2.2 Cải tiến quan trọng

- YOLOv8 mang lại nhiều cải tiến quan trọng so với các phiên bản trước, bao gồm:
  - + Kiến trúc CSPNet: Sử dụng CSPNet để cải thiện hiệu suất và giảm chi phí tính toán bằng cách chia nhỏ các feature. Các chỉ số đo độ chính xác và kết hợp chúng lại sau khi thực hiện các phép biến đổi.
  - + Các lớp convolution sâu hơn: Sử dụng các lớp convolution sâu hơn và phức tạp hơn để cải thiện khả năng trích xuất đặc trưng.
  - + Attention mechanism: Áp dụng các cơ chế attention để tập trung vào các vùng quan trọng của ảnh, giúp cải thiện độ chính xác trong việc phát hiện các vật thể nhỏ và khó nhận diện.
  - + Hyperparameter optimization: Tối ưu hóa các siêu tham số thông qua các kỹ thuật như grid search và Bayesian optimization để tìm ra các cấu hình tốt nhất cho mạng.

### 2.2.3 Ưu, nhược điểm

- Ưu điểm:

- + Tốc độ cao: YOLOv8 vẫn giữ được tốc độ xử lý nhanh, giúp nó phù hợp với các ứng dụng thời gian thực.
- + Độ chính xác cao: Nhờ vào các cải tiến về kiến trúc và thuật toán, YOLOv8 đạt được độ chính xác cao hơn trong việc phát hiện vật thể.
- + Khả năng nhận diện vật thể nhỏ: Với các cơ chế attention và các lớp convolution sâu, YOLOv8 cải thiện đáng kể khả năng nhận diện các vật thể nhỏ và khó phát hiện.

- Nhược điểm:

- + Chi phí tính toán cao: Các cải tiến về kiến trúc và sử dụng các lớp convolution sâu hơn dẫn đến chi phí tính toán cao hơn, yêu cầu phần cứng mạnh mẽ hơn.
- + Độ phức tạp: Kiến trúc phức tạp hơn đòi hỏi nhiều thời gian và công sức trong việc huấn luyện và tối ưu hóa mô hình.
- + Cần nhiều dữ liệu: Để đạt được hiệu suất tốt nhất, YOLOv8 yêu cầu một lượng lớn dữ liệu huấn luyện chất lượng cao.

## Chương 3: Phương pháp đề xuất

### 3.1 Điều chỉnh siêu tham số

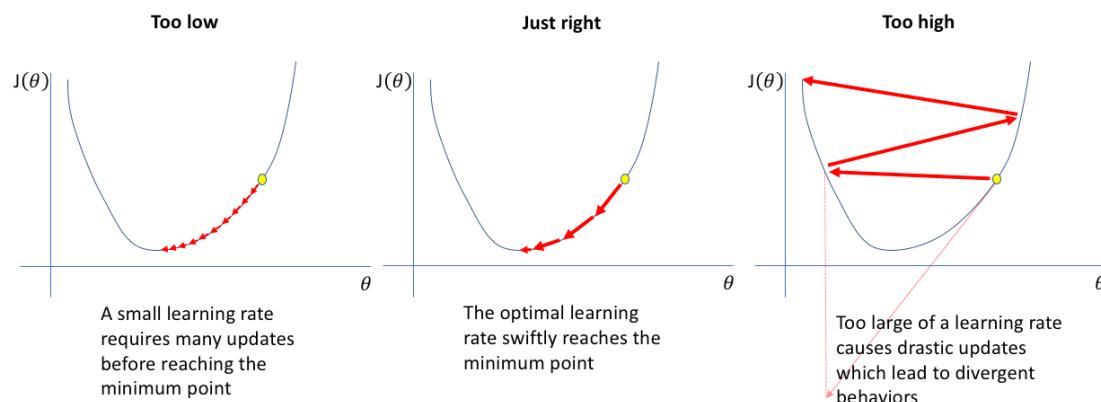
- Siêu tham số (hyperparameter) là những tham số được thiết lập trước khi quá trình huấn luyện bắt đầu và không được cập nhật trong quá trình huấn luyện của mô hình. Chúng có tác động lớn đến hiệu suất của mô hình và quá trình học của mạng nơ-ron nhân tạo. Dưới đây là một số siêu tham số phổ biến trong mô hình học sâu:

#### 3.1.1 Learning Rate (Tốc độ học)

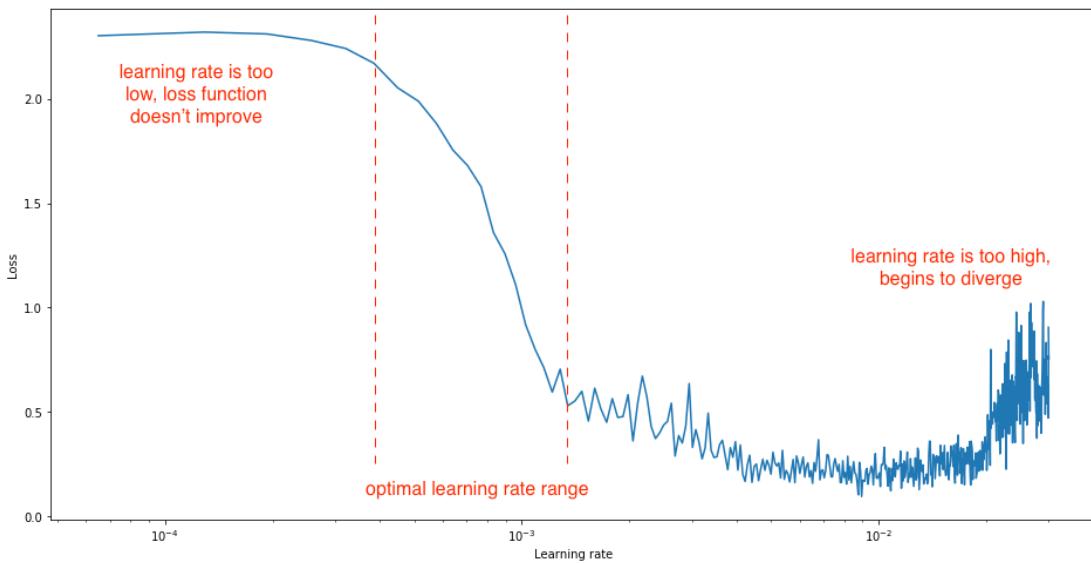
- **Định nghĩa:** Là một hệ số xác định mức độ điều chỉnh trọng số của mạng nơ-ron đổi với mỗi bước cập nhật dựa trên độ dốc của hàm mất mát.

- **Ý nghĩa thay đổi:**

- + **Learning rate quá cao:** Mô hình có thể dao động xung quanh giá trị tối ưu và không hội tụ.
- + **Learning rate quá thấp:** Quá trình huấn luyện diễn ra chậm và mô hình có thể bị kẹt tại các cực trị cục bộ.



Hình 2: [1] Ảnh hưởng của learning rate.



Hình 2: [2] Tìm kiếm learning rate thích hợp.

### 3.1.2 Batch Size

- **Định nghĩa:** Số lượng mẫu được sử dụng để tính toán mỗi bước cập nhật của trọng số.

- **Ý nghĩa thay đổi:**

+ **Batch size lớn:** Giúp tính toán gradient chính xác hơn nhưng yêu cầu bộ nhớ lớn và thời gian huấn luyện lâu hơn.

+ **Batch size nhỏ:** Giảm nhu cầu bộ nhớ và tăng tốc độ huấn luyện, nhưng gradient tính toán ít chính xác hơn.

### 3.1.3 Epochs

- **Định nghĩa:** Số lần toàn bộ tập dữ liệu huấn luyện được đưa qua mô hình.

- **Ý nghĩa thay đổi:**

+ **Số lượng epochs quá ít:** Mô hình chưa học đủ, dẫn đến underfitting.

+ **Số lượng epochs quá nhiều:** Mô hình có thể học quá mức, dẫn đến overfitting.

### 3.1.4 Optimizer (Bộ tối ưu hóa):

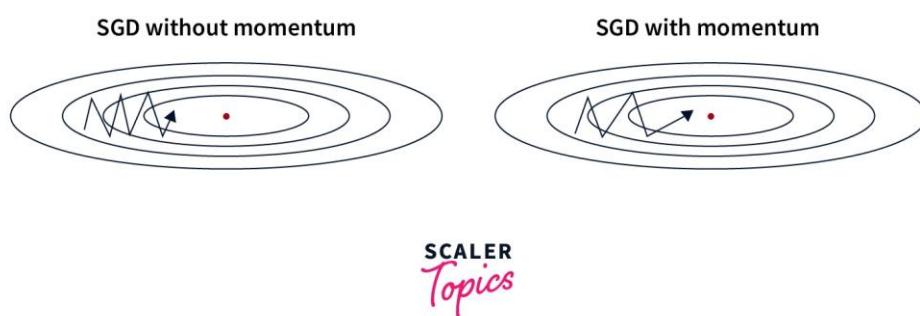
- **Định nghĩa:** Thuật toán được sử dụng để điều chỉnh trọng số của mô hình nhằm tối thiểu hóa hàm mất mát.
- **Ý nghĩa thay đổi:** Các bộ tối ưu hóa khác nhau (như SGD, Adam, RMSprop) có các chiến lược cập nhật trọng số khác nhau, ảnh hưởng đến tốc độ và khả năng hội tụ của mô hình.

### 3.1.5 Momentum

- **Định nghĩa:** Tham số này giúp tăng tốc độ hội tụ của mô hình bằng cách thêm một phần của gradient trước đó vào gradient hiện tại.

#### - Ý nghĩa thay đổi:

- + **Momentum quá cao:** Có thể dẫn đến dao động quanh giá trị tối ưu.
- + **Momentum quá thấp:** Có thể làm chậm quá trình hội tụ.



Hình 4: [3] SGD với momentum.

### 3.1.6 Dropout Rate

- **Định nghĩa:** Tỷ lệ bỏ qua ngẫu nhiên các đơn vị (neurons) trong quá trình huấn luyện để ngăn ngừa overfitting.
- **Ý nghĩa thay đổi:**

- + **Dropout rate quá cao:** Mô hình khó học được các đặc trưng quan trọng.
- + **Dropout rate quá thấp:** Không đủ hiệu quả trong việc ngăn chặn overfitting.

### 3.1.7 Hiệu quả của sự thay đổi các siêu tham số

- **Hiệu suất mô hình:** Điều chỉnh siêu tham số có thể cải thiện hoặc làm giảm hiệu suất của mô hình trên tập kiểm tra.
- **Tốc độ hội tụ:** Các siêu tham số như learning rate và batch size ảnh hưởng trực tiếp đến tốc độ hội tụ của mô hình.
- **Overfitting và Underfitting:** Việc thiết lập các siêu tham số hợp lý giúp ngăn ngừa overfitting và underfitting, từ đó cải thiện khả năng tổng quát hóa của mô hình.
- **Sử dụng tài nguyên:** Siêu tham số như batch size và number of epochs ảnh hưởng đến tài nguyên tính toán và thời gian huấn luyện.
- Việc chọn lựa và điều chỉnh siêu tham số là một quá trình quan trọng và đòi hỏi sự thử nghiệm, thường thông qua các phương pháp như grid search, random search, hoặc các thuật toán tối ưu hóa bayesian để tìm ra cấu hình tốt nhất cho mô hình.

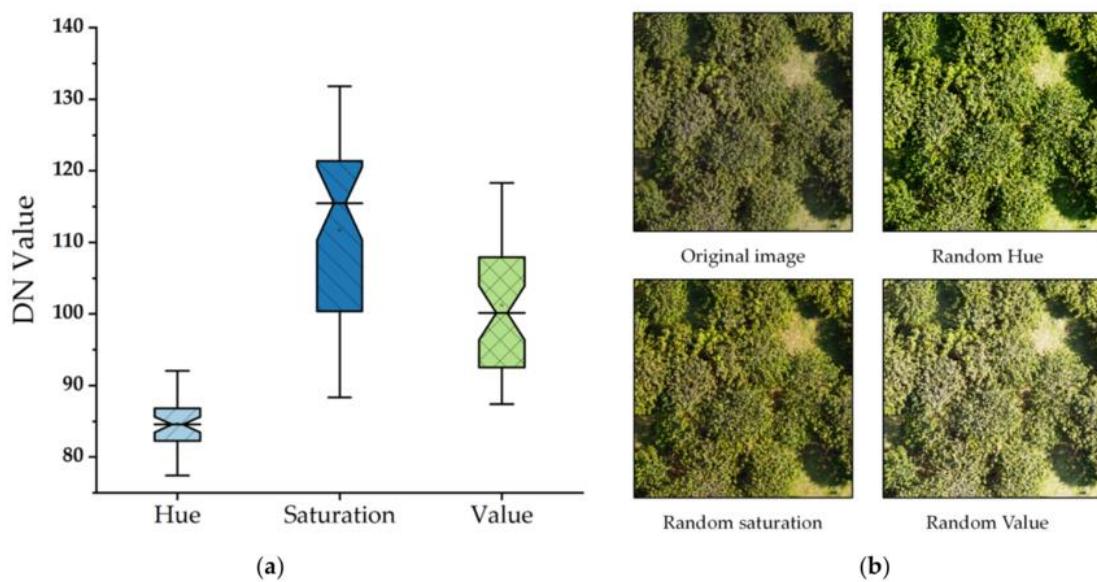
## 3.2 Tăng cường dữ liệu

- Tăng cường dữ liệu (data augmentation) là một kỹ thuật được sử dụng để tạo ra các biến thể của dữ liệu huấn luyện hiện có nhằm cải thiện khả năng tổng quát hóa của mô hình học sâu. Mục tiêu của việc tăng cường dữ liệu là giúp mô hình học được nhiều đặc trưng hơn từ dữ liệu bằng cách làm phong phú thêm tập huấn luyện mà không cần thu thập thêm dữ liệu mới. Sau đây là 1 số tham số và ý nghĩa của chúng:

### 3.2.1 HSV (Hue, Saturation, Value) Augmentation (hsv\_h, hsv\_s, hsv\_v)

- **hsv\_h (Hue Shift):** Điều chỉnh sắc độ của ảnh. Tham số này thay đổi màu sắc của ảnh mà không thay đổi độ sáng hoặc độ bão hòa.

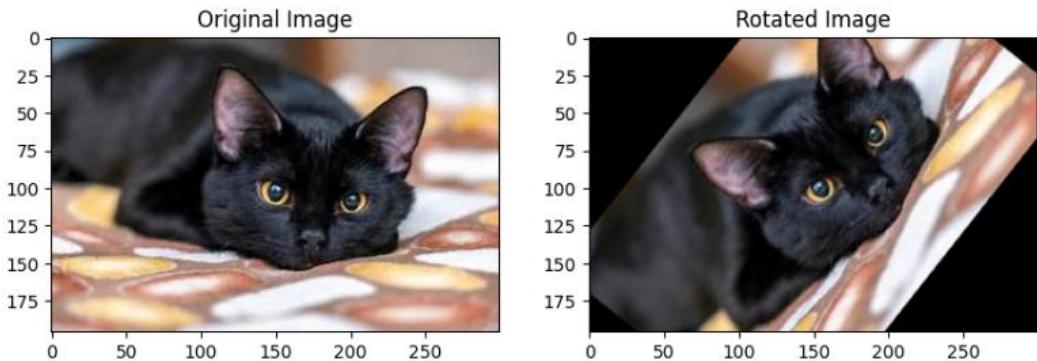
- **hsv\_s (Saturation Shift)**: Điều chỉnh độ bão hòa của ảnh. Tham số này thay đổi độ rực rỡ của màu sắc trong ảnh.
- **hsv\_v (Value Shift)**: Điều chỉnh độ sáng của ảnh. Tham số này thay đổi độ sáng tổng thể của ảnh.
- **Ý nghĩa**: Những thay đổi này giúp mô hình học được cách nhận diện các đối tượng dưới các điều kiện ánh sáng và màu sắc khác nhau.



Hình 5: [4] Box plot và hình ảnh minh họa của các giá trị  $hsv_h$ ,  $hsv_s$ ,  $hsv_v$

### 3.2.2 Degrees (Rotation)

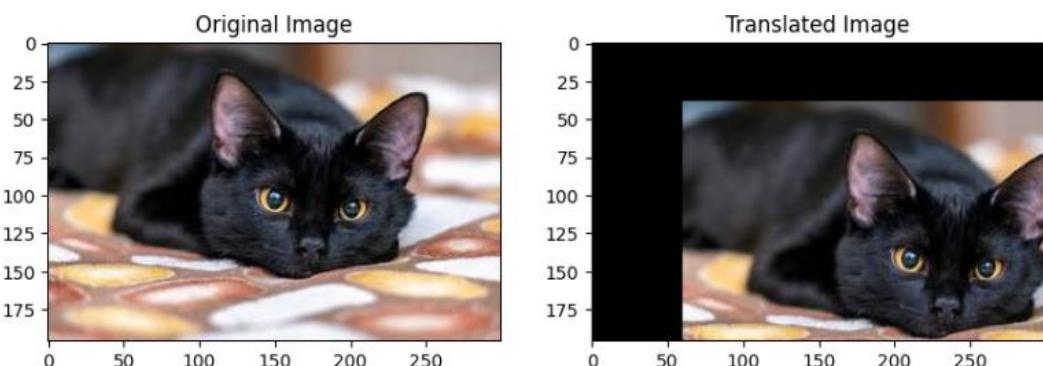
- **Định nghĩa**: Xoay ảnh theo một góc ngẫu nhiên trong khoảng giá trị cho trước.
- **Ý nghĩa**: Giúp mô hình học được cách nhận diện đối tượng từ nhiều góc nhìn khác nhau.



Hình 6: Kỹ thuật quay ảnh.

### 3.2.3 Translate

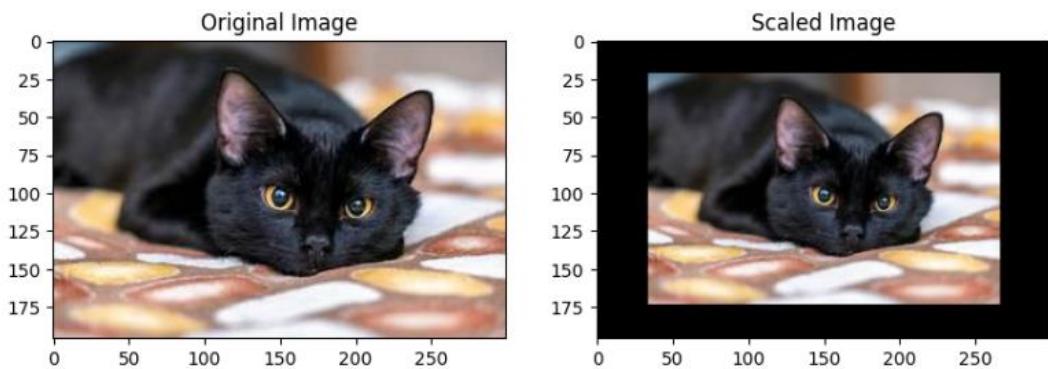
- **Định nghĩa:** Dịch chuyển ảnh theo chiều ngang hoặc chiều dọc với một khoảng ngẫu nhiên.
- **Ý nghĩa:** Giúp mô hình nhận diện đối tượng ngay cả khi chúng không nằm ở vị trí trung tâm của ảnh.



Hình 7: Kỹ thuật translate.

### 3.2.4 Scale

- **Định nghĩa:** Thay đổi kích thước của ảnh bằng cách phóng to hoặc thu nhỏ theo một tỷ lệ ngẫu nhiên.
- **Ý nghĩa:** Giúp mô hình học cách nhận diện đối tượng ở các kích thước khác nhau.



Hình 8: Kỹ thuật scale.

### 3.2.5 Mosaic

- **Định nghĩa:** Kết hợp nhiều ảnh lại với nhau để tạo thành một ảnh duy nhất.
- **Ý nghĩa:** Tăng tính đa dạng của tập dữ liệu bằng cách tạo ra các bối cảnh phức tạp hơn, giúp mô hình nhận diện đối tượng trong nhiều tình huống khác nhau.



Hình 9: Kỹ thuật mosaic.

### 3.2.6 Mixup

- **Định nghĩa:** Trộn lẫn hai ảnh với nhau bằng cách lấy trung bình có trọng số của các giá trị pixel.

- **Ý nghĩa:** Giúp mô hình tổng quát hóa tốt hơn bằng cách tạo ra các mẫu mới từ các mẫu hiện có.

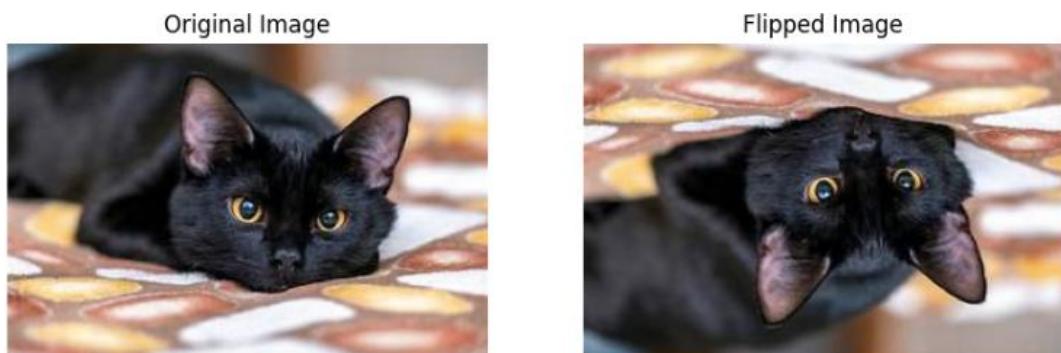


Hình 10: Kỹ thuật MixUp.

### 3.2.7 Flipud (Flip Up-Down)

- **Định nghĩa:** Lật ngược ảnh theo chiều dọc.

- **Ý nghĩa:** Tăng cường tính đa dạng của tập dữ liệu bằng cách thêm các biến thể lật ngược của ảnh.



Hình 11: Kỹ thuật flipud.

### 3.2.8 Fliplr (Flip Left-Right)

- **Định nghĩa:** Lật ngược ảnh theo chiều ngang.

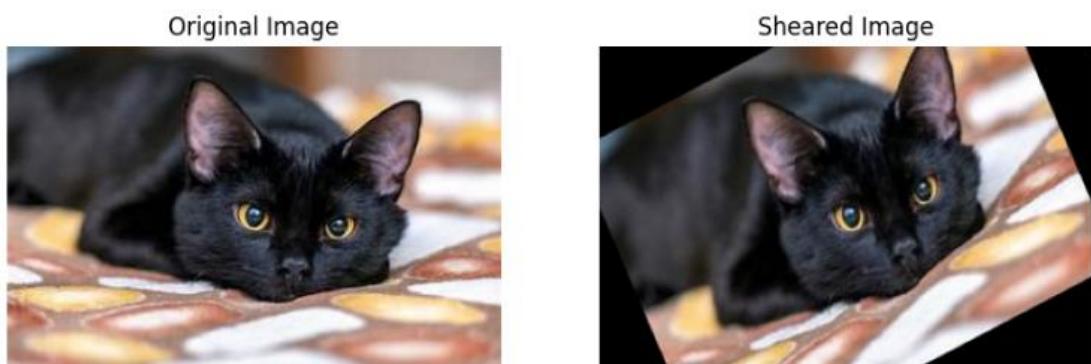
- **Ý nghĩa:** Tăng cường tính đa dạng của tập dữ liệu bằng cách thêm các biến thể lật ngang của ảnh.



Hình 12: Kĩ thuật *flplr.*

### 3.2.9 Shear

- **Định nghĩa:** Biến dạng ảnh theo một hướng cố định, tạo ra hiệu ứng kéo dài hoặc nén.
- **Ý nghĩa:** Giúp mô hình học được cách nhận diện đối tượng ngay cả khi chúng bị biến dạng.



Hình 13: Kĩ thuật *shear.*

### 3.2.10 Perspective

- **Định nghĩa:** Thay đổi phối cảnh của ảnh, tạo ra hiệu ứng như nhìn từ một góc khác.
- **Ý nghĩa:** Giúp mô hình nhận diện đối tượng dưới các góc nhìn phối cảnh khác nhau.



Hình 14: Kỹ thuật perspective.

### 3.2.11 Ảnh hưởng của tăng cường dữ liệu

- **Cải thiện khả năng tổng quát hóa:** Tăng cường dữ liệu giúp mô hình học được nhiều đặc trưng hơn từ dữ liệu, từ đó cải thiện khả năng tổng quát hóa của mô hình đối với dữ liệu chưa từng thấy trước đó.
- **Giảm overfitting:** Bằng cách tạo ra nhiều biến thể của dữ liệu huấn luyện, tăng cường dữ liệu giúp ngăn ngừa mô hình học quá mức vào các đặc trưng cụ thể của tập huấn luyện.
- **Tăng kích thước tập dữ liệu:** Tăng cường dữ liệu giúp tăng kích thước tập dữ liệu huấn luyện mà không cần thu thập thêm dữ liệu mới, từ đó cải thiện hiệu suất huấn luyện mô hình.
- **Đa dạng hóa dữ liệu:** Các kỹ thuật tăng cường dữ liệu giúp mô hình học được cách nhận diện đối tượng trong nhiều tình huống và điều kiện khác nhau, từ đó cải thiện hiệu suất tổng thể của mô hình.
- Việc sử dụng các kỹ thuật tăng cường dữ liệu một cách hợp lý có thể mang lại hiệu suất tốt hơn cho mô hình học sâu, đặc biệt là khi dữ liệu huấn luyện bị hạn chế.

## 3.3 Thay đổi kiến trúc YOLOv8

- Kiến trúc YOLOv8 sử dụng 3 đầu detect cho từng tỉ lệ kích cỡ vật thể cần nhận diện, cùng với đó là những khối tương ứng hỗ trợ. Có thể chia ra làm 3 kiến trúc nhỏ hơn là small, medium, large:

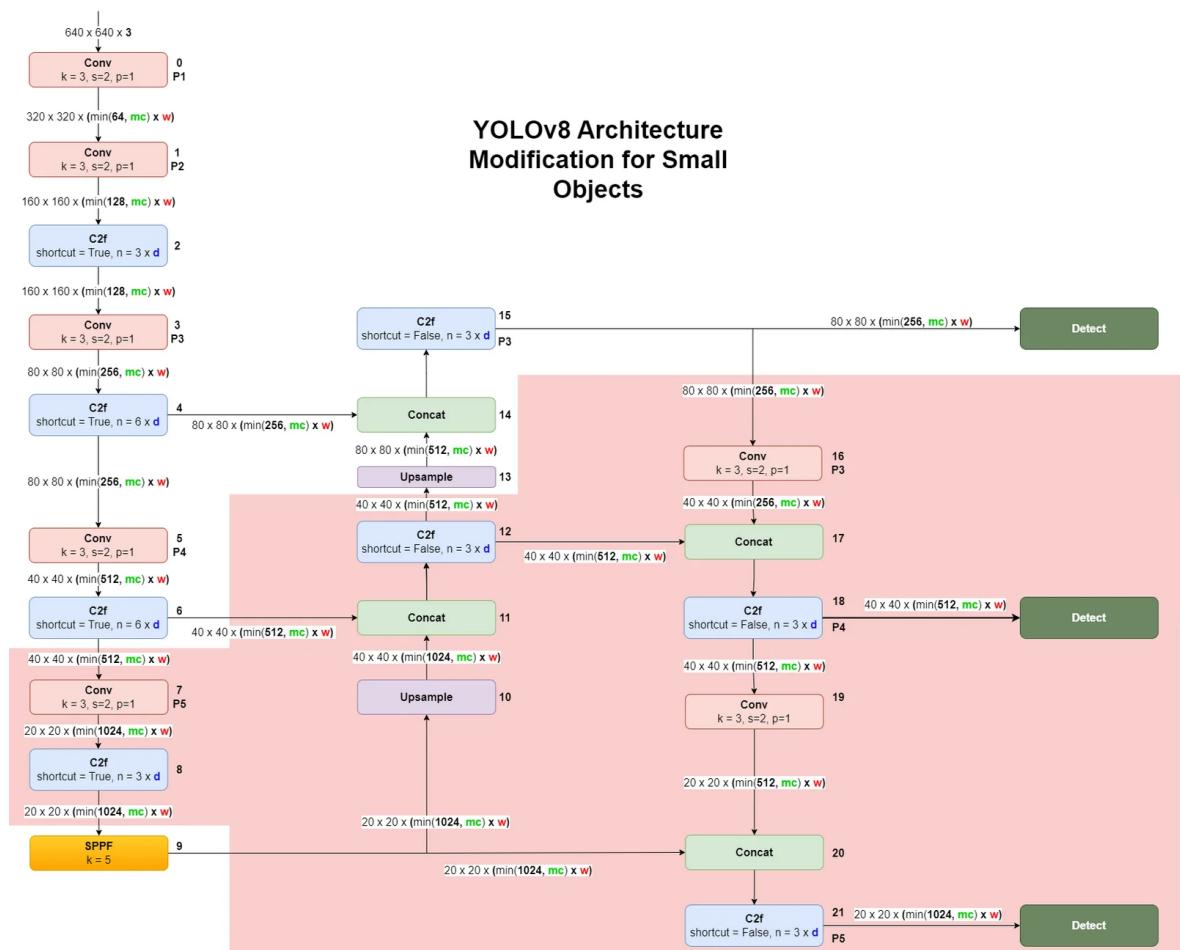
### Chương 3: Phương pháp đề xuất

- +  $80 \times 80 \times (\min(256, mc) \times w)$  cho kích thước nhỏ (small).
- +  $40 \times 40 \times (\min(512, mc) \times w)$  cho kích thước trung bình (medium).
- +  $20 \times 20 \times (\min(1024, mc) \times w)$  cho kích thước lớn (large).

- Với những công việc có sự chính xác và lặp lại cao như sản xuất và chế tạo trong công nghiệp sẽ tạo ra hình ảnh có góc chụp không đổi và tỉ lệ kích cỡ tương đối giống nhau.

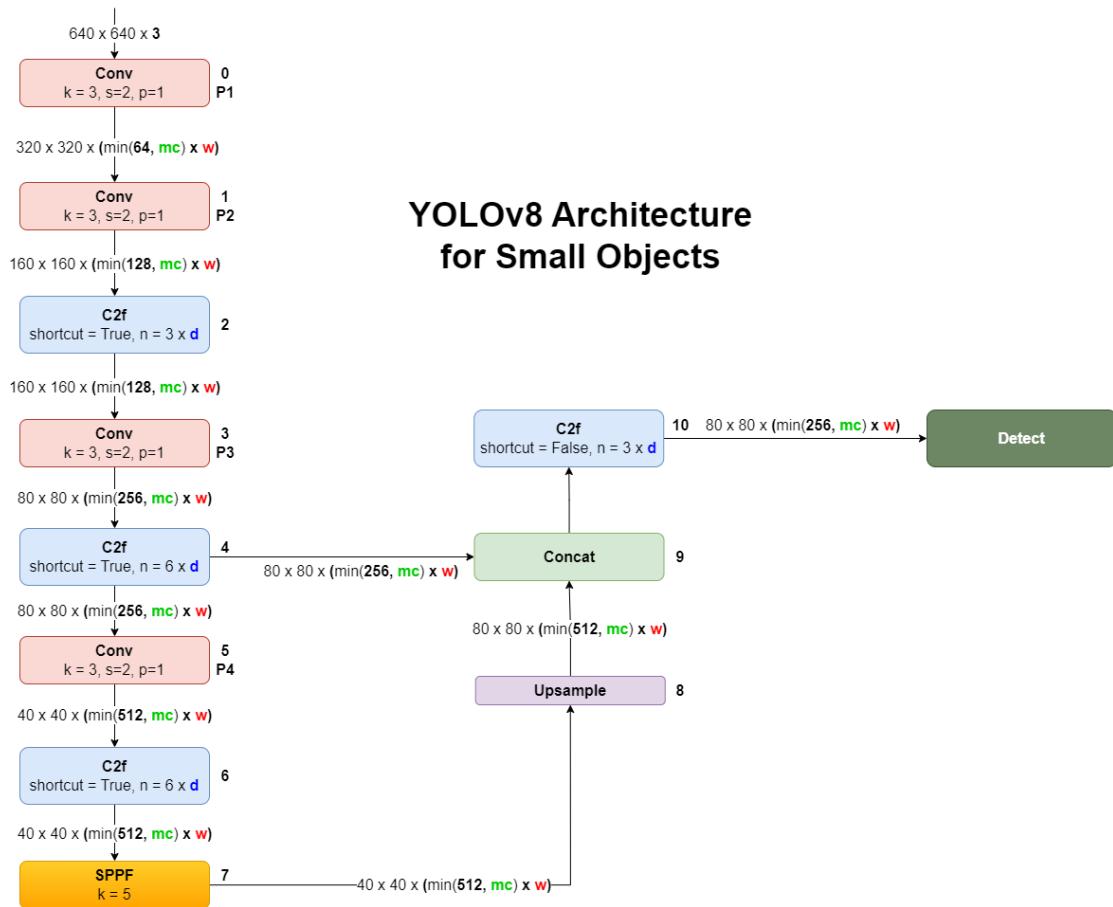
- Từ hiện thực nêu trên, em đề xuất rằng với từng công việc khác nhau tạo ra những bức ảnh có tỉ lệ tương đồng, chúng ta sẽ loại bỏ những đầu detect những khối hỗ trợ tương ứng không cần thiết với tỉ lệ vật thể cần nhận diện để giảm số lượng tham số cần huấn luyện của mô hình từ đó tăng tốc độ detect của model:

+ Small:



### Chương 3: Phương pháp đề xuất

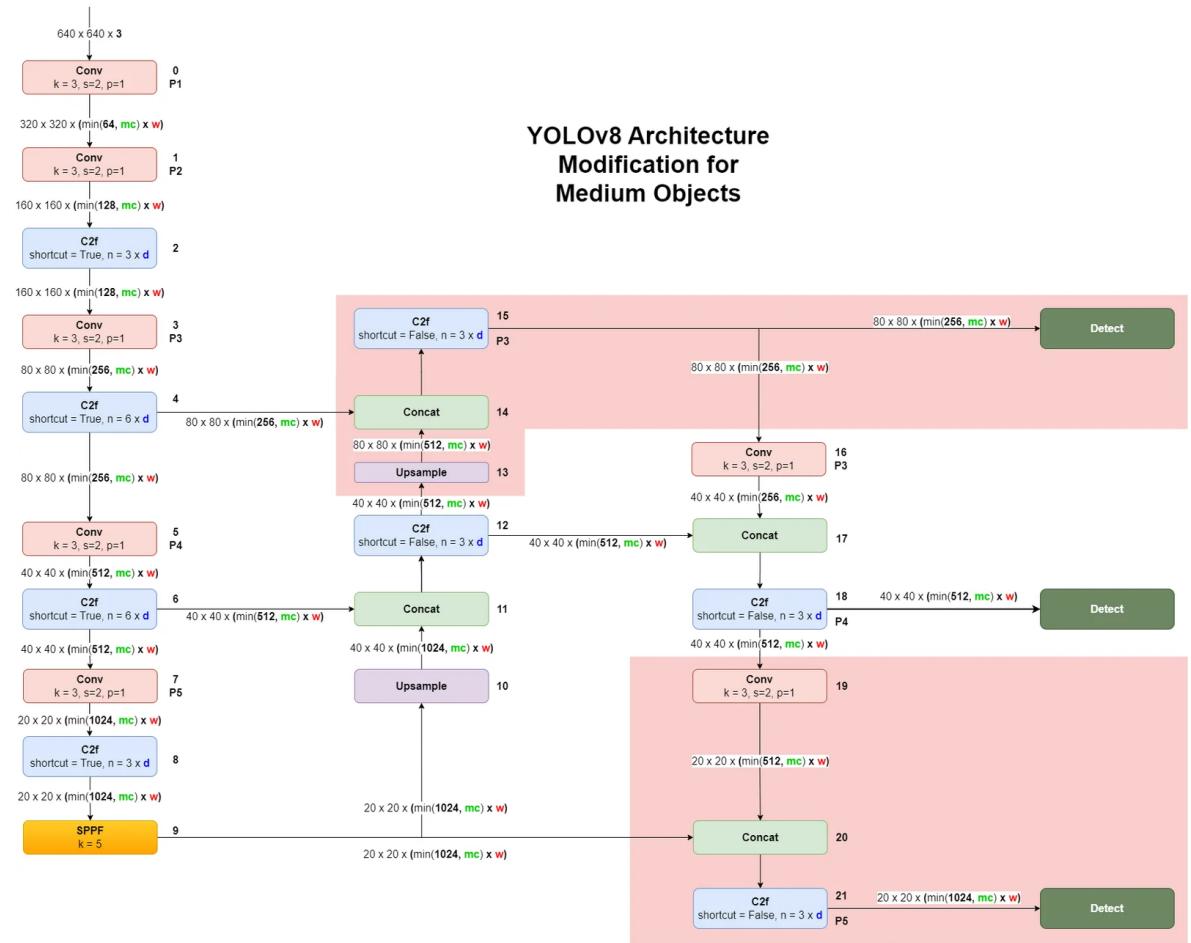
Hình 15: Những thay đổi kiến trúc YOLOv8 cho kích thước nhỏ.



Hình 16: Kiến trúc YOLOv8 cho kích thước nhỏ.

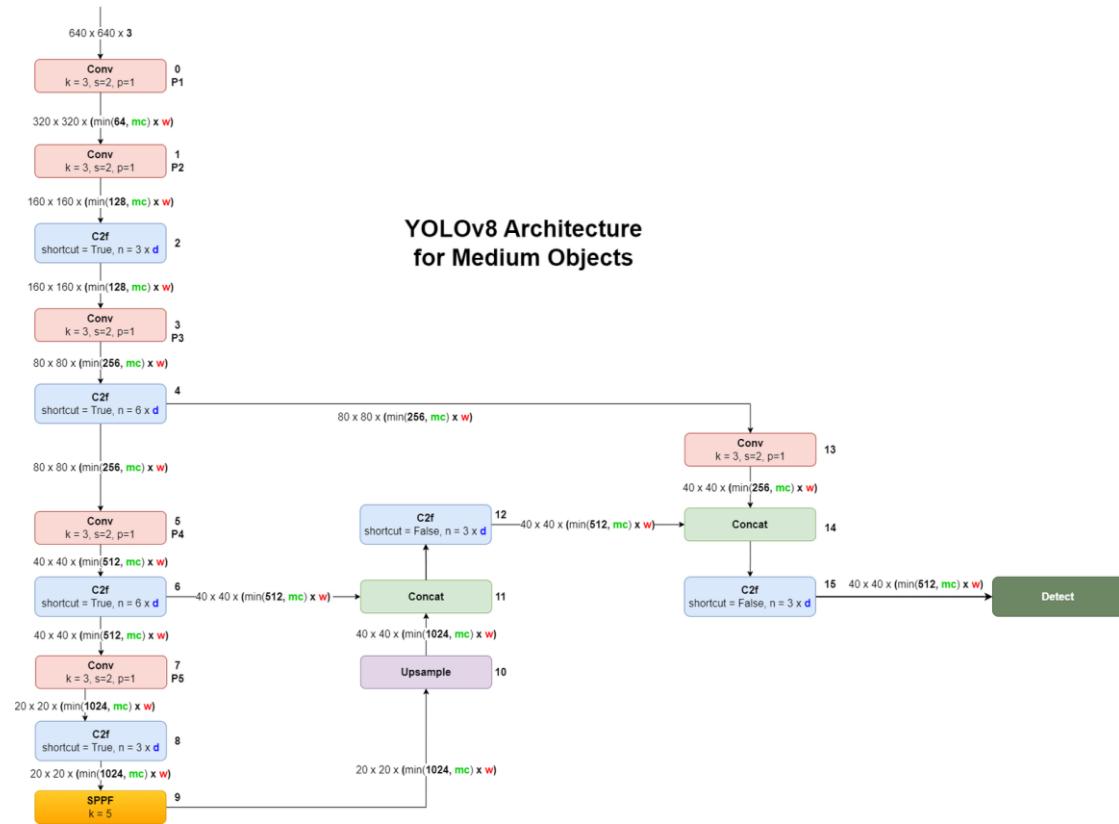
+ Medium:

### Chương 3: Phương pháp đề xuất



*Hình 17: Những thay đổi kiến trúc YOLOv8 cho kích thước trung bình.*

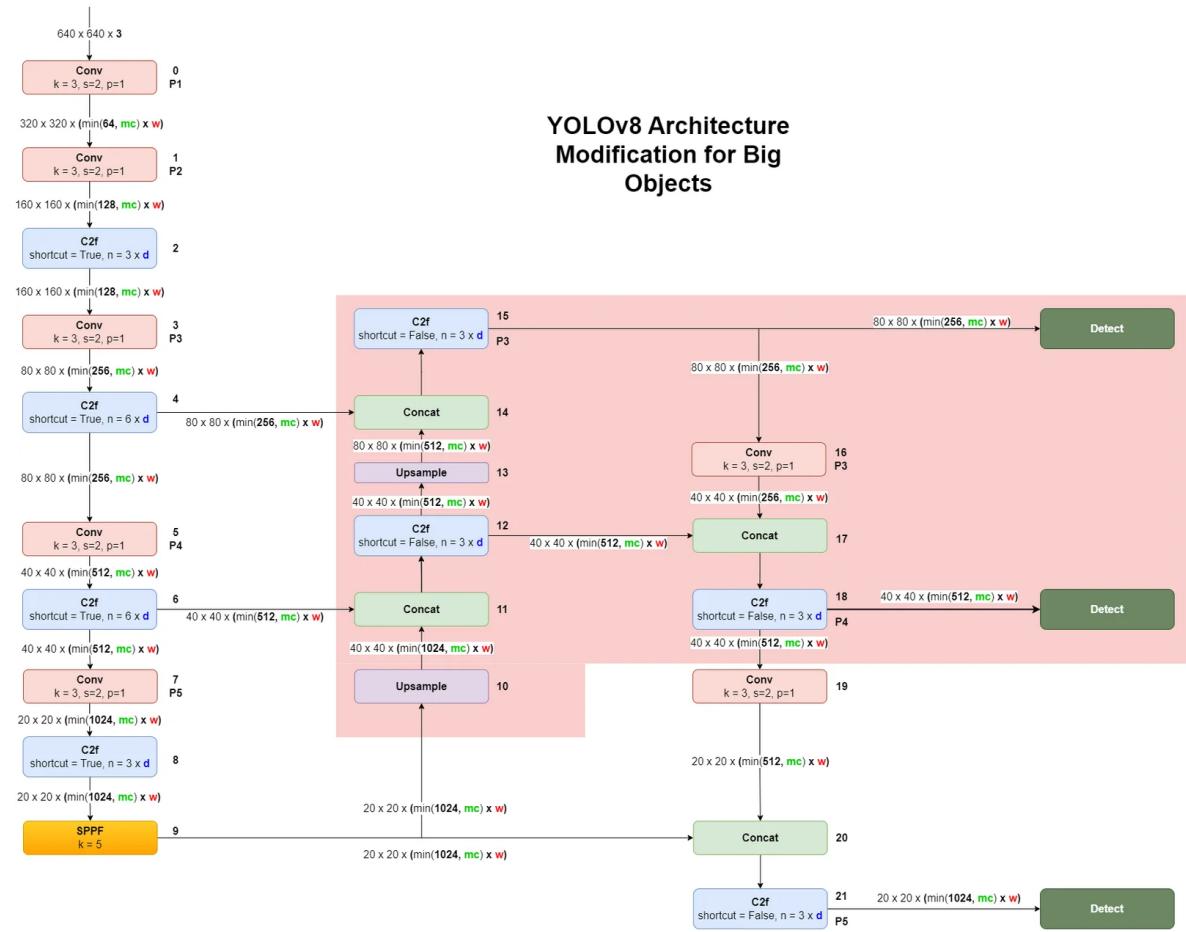
### Chương 3: Phương pháp đề xuất



Hình 18: Kiến trúc YOLOv8 cho kích thước trung bình.

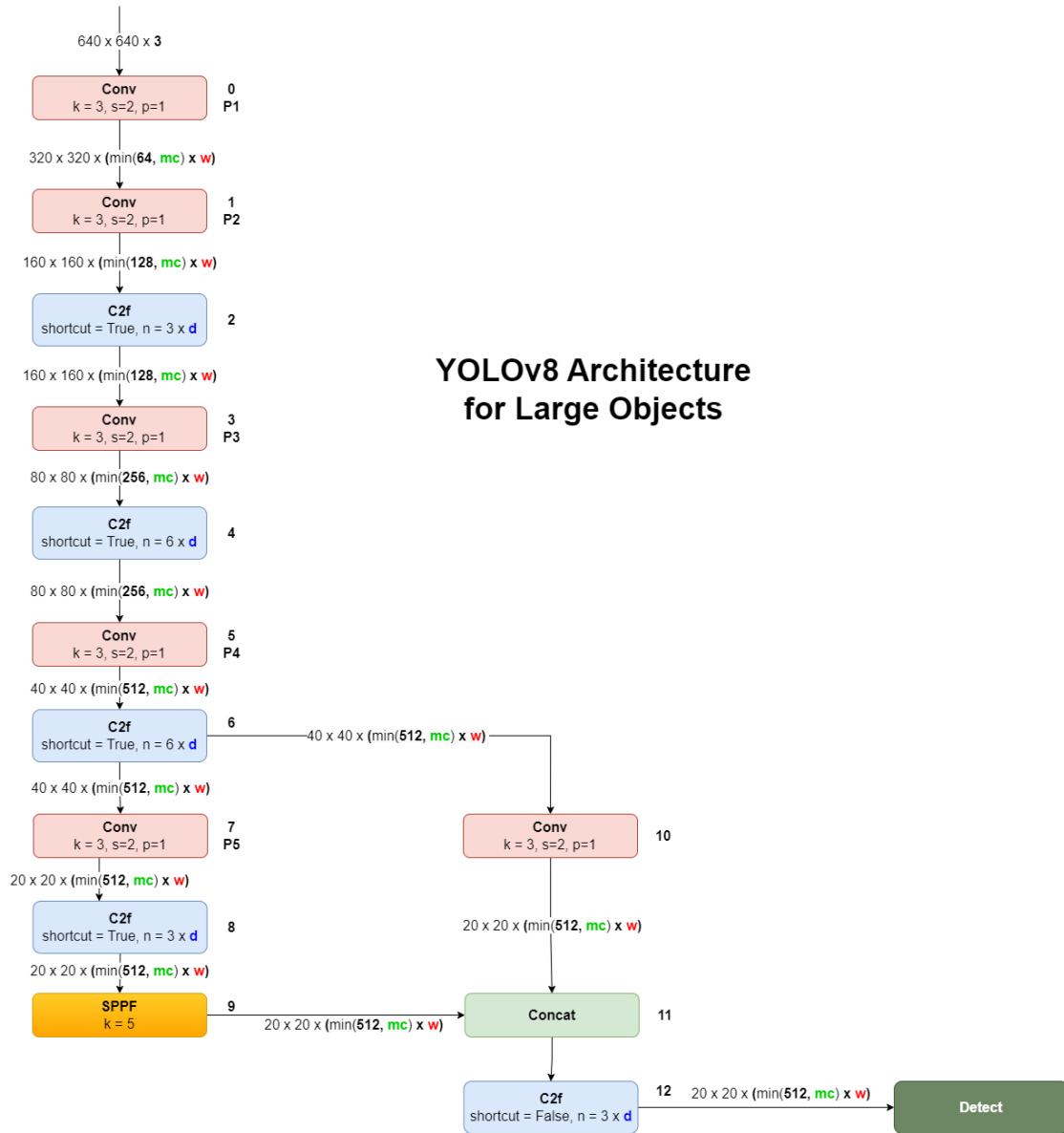
+ Large:

### Chương 3: Phương pháp đề xuất



Hình 19: Những thay đổi kiến trúc YOLOv8 cho kích thước lớn.

### Chương 3: Phương pháp đề xuất



Hình 20: Kiến trúc YOLOv8 cho kích thước lớn.

- Tùy thuộc vào tỉ lệ vật cần detect so với khung hình, chúng ta có thể sử dụng riêng hoặc kết hợp các kiến trúc trên để huấn luyện mô hình.
- Tuy nhiên kỹ thuật này cũng tồn tại 1 vài nhược điểm:
  - + Sự đánh đổi độ chính xác với tốc độ phụ thuộc rất nhiều vào sự nhất quán của tỉ lệ vật cần detect so với khung hình trong tập dữ liệu, bỏ đi các kiến trúc dùng để nhận diện các vật thể tỉ lệ khác đồng nghĩa với việc khó hoặc gần như không thể nhận diện các vật thể tỉ lệ đó.

- + Chưa có khoảng cụ thể để quyết định sẽ chọn kiến trúc nào, dẫn đến việc chỉ có thể dự đoán thông qua trung bình tỉ lệ vật thể với ảnh và phải kiểm chứng với huấn luyện mô hình rất tốn thời gian.

### 3.4 TensorRT

- Có rất nhiều thư viện hỗ trợ cho mô hình YOLOv8 như TensorRT và OpenVINO, sử dụng các thuật toán tối ưu hóa như hiệu chỉnh các lớp và hiệu chỉnh độ chính xác như FP16, INT18 để cải thiện hiệu suất của các mô hình học sâu. Tuy nhiên trong báo cáo này chúng ta chỉ đề cập tương ứng 1 thư viện và đó chính là TensorRT.

#### 3.4.1 Khái niệm

- TensorRT là một thư viện và bộ công cụ được phát triển bởi NVIDIA. Nó được thiết kế để tối ưu hóa và triển khai các mô hình mạng nơ-ron trên GPU của NVIDIA, tăng tốc độ và hiệu quả của các hoạt động suy luận:

##### - Các tính năng chính của TensorRT:

- + **Tối ưu mô hình:** TensorRT thực hiện nhiều tối ưu hóa trên các mô hình mạng nơ-ron đã huấn luyện, như hợp nhất các lớp, hiệu chỉnh độ chính xác (FP32, FP16, INT8), và tự động điều chỉnh nhân, để tối đa hóa hiệu suất trên phần cứng của NVIDIA.
- + **Suy luận hiệu suất cao:** Bằng cách tối ưu hóa mô hình, TensorRT tăng tốc suy luận, giảm cả độ trễ và tài nguyên tính toán cần thiết. Điều này rất quan trọng cho các ứng dụng thời gian thực như lái xe tự động, robot và phân tích video.
- + **Hiệu chỉnh độ chính xác:** TensorRT cho phép các mô hình chạy ở các chế độ độ chính xác giảm (FP16 và INT8), giúp cải thiện hiệu suất đáng kể và giảm sử dụng bộ nhớ mà không làm giảm độ chính xác quá nhiều.
- + **Quản lý bộ nhớ động Tensor:** TensorRT quản lý bộ nhớ tensor một cách động, giúp sử dụng bộ nhớ GPU hiệu quả, cho phép triển khai các mô hình phức tạp hơn hoặc chạy nhiều mô hình đồng thời.

- + **Tích hợp với hệ sinh thái NVIDIA:** TensorRT được thiết kế để hoạt động liền mạch với các khung và thư viện học sâu của NVIDIA, bao gồm CUDA, cuDNN và DeepStream SDK, tạo điều kiện thuận lợi cho quy trình triển khai từ đầu đến cuối.
- + **Hỗ trợ các khung chính:** TensorRT hỗ trợ các mô hình từ các khung học sâu phổ biến như TensorFlow, PyTorch, ONNX và Caffe. Điều này giúp dễ dàng triển khai các mô hình được huấn luyện trên các khung này lên GPU của NVIDIA.

### 3.4.2 Ưu, nhược điểm

#### - Ưu điểm:

- + **Tối ưu hiệu suất:** TensorRT cung cấp các cải tiến về tốc độ đáng kể cho suy luận thông qua tối ưu hóa các lớp mạng, hợp nhất lớp, và lựa chọn độ chính xác tốt nhất (FP32, FP16, INT8) cho hiệu suất.
- + **Giảm độ trễ:** TensorRT giảm độ trễ của các tác vụ suy luận, rất quan trọng cho các ứng dụng thời gian thực như nhận diện đối tượng trong xe tự lái hoặc giám sát video.
- + **Sử dụng tài nguyên hiệu quả:** TensorRT tối ưu hóa việc sử dụng bộ nhớ GPU và tài nguyên tính toán, cho phép triển khai nhiều mô hình hơn hoặc xử lý nhiều dữ liệu hơn trong cùng một hạn chế phần cứng.
- + **Hiệu chỉnh độ chính xác:** TensorRT hỗ trợ độ chính xác INT8, giúp giảm đáng kể bộ nhớ và tăng thông lượng mà không làm giảm độ chính xác nhiều, đặc biệt hữu ích cho triển khai trên các thiết bị biên.
- + **Tích hợp hệ thống:** TensorRT tích hợp tốt với phần cứng và phần mềm của NVIDIA, bao gồm CUDA, cuDNN và DeepStream SDK, tạo điều kiện thuận lợi cho các quy trình triển khai từ đầu đến cuối.
- + **Bộ nhớ Tensor động:** TensorRT phân bổ và giải phóng bộ nhớ tensor động, giúp quản lý bộ nhớ GPU hiệu quả hơn và cho phép xử lý các kích thước đầu vào khác nhau.

**- Nhược Điểm:**

- + **Độ phức tạp:** Quá trình chuyển đổi mô hình sang TensorRT có thể phức tạp và đòi hỏi hiểu biết tốt về cả kiến trúc mô hình và TensorRT. Việc khắc phục sự cố phát sinh trong quá trình chuyển đổi có thể khó khăn.
- + **Vấn đề tương thích:** TensorRT có thể gặp vấn đề tương thích với một số lớp hoặc thao tác trong mô hình gốc, đòi hỏi các triển khai tùy chỉnh hoặc điều chỉnh mô hình.
- + **Hỗ trợ khung hạn chế:** Mặc dù TensorRT hỗ trợ nhiều khung phổ biến như TensorFlow và PyTorch, nhưng hỗ trợ cho các khung mới hoặc ít phổ biến hơn có thể bị chậm hoặc cần nỗ lực bổ sung để tích hợp.
- + **Dánh đổi độ chính xác:** Mặc dù độ chính xác thấp hơn (như INT8) mang lại lợi ích về hiệu suất, nhưng có thể dẫn đến giảm độ chính xác của mô hình. Cần hiệu chỉnh và thử nghiệm kỹ lưỡng để đảm bảo hiệu suất chấp nhận được.
- + **Yêu cầu tài nguyên:** Tối ưu hóa và suy luận sử dụng TensorRT có thể yêu cầu nhiều tài nguyên, đòi hỏi GPU mạnh mẽ, điều này có thể không khả thi cho tất cả các môi trường triển khai.
- + **Tối ưu hóa tĩnh:** Các tối ưu hóa của TensorRT được thực hiện tại thời điểm tạo engine. Điều này có nghĩa là các thay đổi động của mô hình trong thời gian chạy (ví dụ: cập nhật mô hình) yêu cầu chuyển đổi và tối ưu hóa lại.
- + **Khóa Vendor:** TensorRT là sản phẩm của NVIDIA, điều này ràng buộc việc triển khai vào phần cứng của NVIDIA. Đây có thể là một hạn chế nếu cần triển khai trên phần cứng của các nhà cung cấp khác.

## Chương 4: Thực nghiệm

### 4.1 Thiết lập thực nghiệm

- Thực hiện các thực nghiệm trên các tập dữ liệu là Fruit Freshness Detection Dataset với 12410 ảnh thuộc 4 lớp gồm có 'Fresh Apple', 'Fresh Banana', 'Rotten Apple', 'Rotten Banana' được thu thập bởi Brac University [5] và fm với 801 ảnh thuộc 3 lớp gồm có 'Mask', 'No Mask', 'Bad Mask'. Các bước thực hiện chủ yếu thông qua API của Ultralytics [6] và triển khai trên Google Colab bao gồm:

#### 4.1.1 Chuẩn bị Dữ Liệu:

- Đổi tên tập dữ liệu Fruit Freshness Detection Dataset thành fruit\_fr để tiện việc sử dụng sau này và sau đó chia cả 2 tập dữ liệu thành các tập huấn luyện, xác thực và kiểm tra.

- Các dữ liệu được chuẩn bị với các đặc trưng sau:

- + Huấn luyện: 70% mẫu
- + Xác thực: 20% mẫu
- + Kiểm tra: 10% mẫu

#### 4.1.2 Huấn luyện mô hình nguyên bản của YOLOv8

##### 4.1.2.1 fm

- Sử dụng tập dữ liệu fm để huấn luyện mô hình nguyên bản của YOLOv8, đặt tên là fm\_original:

```
#FaceMask: fm_original
!yolo detect train model=yolov8l.pt data=datasets/fm_data.yaml workers=2 batch=12 device=0 epochs=100 patience=50 name=fm_original
```

Hình 21: Sử dụng API để huấn luyện fm\_original.

- Kết quả huấn luyện:

## Chương 4: Thực nghiệm

```
100 epochs completed in 1.074 hours.  
Optimizer stripped from runs/detect/fm_original/weights/last.pt, 87.6MB  
Optimizer stripped from runs/detect/fm_original/weights/best.pt, 87.6MB  
  
Validating runs/detect/fm_original/weights/best.pt...  
Ultralytics YOLOv8.2.71 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPS  
    Class      Images Instances   Box(P)      R      mAP50    mAP50-95: 100% 7/7 [00:09<00:00, 1.33s/it]  
    all       160        866   0.918     0.825   0.911     0.632  
    Mask      142        709   0.977     0.873   0.958     0.67  
    No Mask    54        133   0.926     0.852   0.945     0.65  
    Bad Mask   20         24   0.851     0.75    0.829     0.576  
Speed: 0.6ms preprocess, 13.2ms inference, 0.0ms loss, 3.3ms postprocess per image
```

Hình 22: Kết quả huấn luyện của fm\_original.

- Các chỉ số đo độ chính xác:

```
#fm_original  
!yolo detect val model=runs/detect/fm_original/weights/best.pt data=datasets/fm_data.yaml iou=0.5 imgsz=640 half=True name=val_fm_original  
  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPS  
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...  
100% 755k/755k [00:00<00:00, 27.7MB/s]  
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fm/labels/val... 160 images, 0 backgrounds, 0 corrupt: 100% 160/160 [01:07<00:00, 2.36it/s]  
val: New cache created: /content/gdrive/MyDrive/yolov8_KLTN/datasets/fm/labels/val.cache  
    Class      Images Instances   Box(P)      R      mAP50    mAP50-95: 100% 10/10 [00:12<00:00, 1.29s/it]  
    all       160        866   0.919     0.825   0.911     0.633  
    Mask      142        709   0.979     0.873   0.959     0.672  
    No Mask    54        133   0.926     0.852   0.946     0.651  
    Bad Mask   20         24   0.851     0.75    0.829     0.577  
Speed: 1.1ms preprocess, 29.0ms inference, 0.1ms loss, 15.2ms postprocess per image
```

Hình 23: Các chỉ số đo độ chính xác của fm\_original.

- Tốc độ:

```
#fm_original  
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test.jpg save=True name=fm_img_test project=result/fm_original  
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test2.jpeg save=True name=fm_img_test2 project=result/fm_original  
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test3.jpg save=True name=fm_img_test3 project=result/fm_original  
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test4.jpg save=True name=fm_img_test4 project=result/fm_original  
  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPS  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test.jpg: 384x640 19 Masks, 75.1ms  
Speed: 3.2ms preprocess, 75.1ms inference, 510.0ms postprocess per image at shape (1, 3, 384, 640)  
Results saved to result/fm_original/fm_img_test  
💡 Learn more at https://docs.ultralytics.com/modes/predict  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPS  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test2.jpeg: 448x640 8 Masks, 10 No Masks, 129.8ms  
Speed: 5.6ms preprocess, 129.8ms inference, 979.9ms postprocess per image at shape (1, 3, 448, 640)  
Results saved to result/fm_original/fm_img_test2  
💡 Learn more at https://docs.ultralytics.com/modes/predict  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPS  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test3.jpg: 384x640 14 Masks, 1 No Mask, 1 Bad Mask, 71.8ms  
Speed: 2.3ms preprocess, 71.8ms inference, 557.5ms postprocess per image at shape (1, 3, 384, 640)  
Results saved to result/fm_original/fm_img_test3  
💡 Learn more at https://docs.ultralytics.com/modes/predict  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPS  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test4.jpg: 384x640 10 Masks, 72.8ms  
Speed: 2.5ms preprocess, 72.8ms inference, 516.3ms postprocess per image at shape (1, 3, 384, 640)  
Results saved to result/fm_original/fm_img_test4  
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 24: Tốc độ của fm\_original.

### 4.1.2.2 fruit\_fr

- Sử dụng tập dữ liệu fruit\_fr để huấn luyện mô hình nguyên bản của YOLOv8, đặt tên là fruit\_original:

```
#fruit_original
!yolo detect train model=yolov8l.pt data=datasets/data.yaml workers=2 batch=16 device=0 epochs=100 patience=50 name=fruit_original
```

Hình 25: Sử dụng API để huấn luyện fruit\_original.

- Kết quả huấn luyện:

```
100 epochs completed in 15.832 hours.
Optimizer stripped from runs/detect/fruit_original4/weights/last.pt, 87.7MB
Optimizer stripped from runs/detect/fruit_original4/weights/best.pt, 87.7MB

Validating runs/detect/fruit_original4/weights/best.pt...
Ultralytics YOLOv8.2.41 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43609692 parameters, 0 gradients, 164.8 GFLOPs
      Class   Images Instances   Box(P)      R    mAP50    mAP50-95: 100% 33/33 [00:25<00:00,  1.29it/s]
          all     1030     1309    0.909    0.843    0.886    0.629
        Fresh Apple    281     303    0.966    0.943    0.976    0.819
        Fresh Banana   220     385    0.84     0.666    0.769    0.443
        Rotten Apple   278     315    0.937    0.99     0.969    0.773
        Rotten Banana  249     306    0.891    0.771    0.828    0.48
Speed: 0.3ms preprocess, 16.3ms inference, 0.0ms loss, 2.2ms postprocess per image
```

Hình 26: Kết quả huấn luyện của fruit\_original.

- Các chỉ số đo độ chính xác:

```
#fruit_original
!yolo detect val model=runs/detect/fruit_original/weights/best.pt data=datasets/data.yaml iou=0.5 imgsz=640 half=True name=val_fruit_original

Ultralytics YOLOv8.2.46 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43609692 parameters, 0 gradients, 164.8 GFLOPs
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 138MB/s]
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fruit_fr/labels/val.cache... 1030 images, 2 backgrounds, 0 corrupt: 100% 1030/1030 [00:00<?, ?it/s]
      Class   Images Instances   Box(P)      R    mAP50    mAP50-95: 100% 65/65 [01:10<00:00,  1.08s/it]
          all     1030     1309    0.928    0.842    0.891    0.633
        Fresh Apple    281     303    0.969    0.944    0.976    0.821
        Fresh Banana   220     385    0.897    0.654    0.778    0.446
        Rotten Apple   278     315    0.938    0.99     0.969    0.776
        Rotten Banana  249     306    0.907    0.778    0.841    0.489
Speed: 0.2ms preprocess, 15.0ms inference, 0.0ms loss, 2.0ms postprocess per image
```

Hình 27: Các chỉ số đo độ chính xác của fruit\_original.

- Tốc độ:

## Chương 4: Thực nghiệm

```
#fruit_original
!yolo detect predict model=runs/detect/fruit_original/weights/best.pt source=inference/fruit_img_test.jpg save=True name=fruit_img_test project=result/fruit_original
!yolo detect predict model=runs/detect/fruit_original/weights/best.pt source=inference/fruit_img_test2.jpg save=True name=fruit_img_test2 project=result/fruit_original
!yolo detect predict model=runs/detect/fruit_original/weights/best.pt source=inference/fruit_img_test3.jpg save=True name=fruit_img_test3 project=result/fruit_original
!yolo detect predict model=runs/detect/fruit_original/weights/best.pt source=inference/fruit_img_test4.jpg save=True name=fruit_img_test4 project=result/fruit_original

Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test.jpg: 640x640 1 Fresh Apple, 62.8ms
Speed: 5.0ms preprocess, 62.8ms inference, 711.6ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_original/fruit_img_test
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test2.jpg: 512x640 1 Rotten Apple, 128.8ms
Speed: 5.0ms preprocess, 128.8ms inference, 580.4ms postprocess per image at shape (1, 3, 512, 640)
Results saved to result/fruit_original/fruit_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test3.jpg: 448x640 1 Rotten Banana, 105.0ms
Speed: 4.3ms preprocess, 105.0ms inference, 529.8ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fruit_original/fruit_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test4.jpg: 448x640 1 Fresh Banana, 103.3ms
Speed: 3.8ms preprocess, 103.3ms inference, 565.0ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fruit_original/fruit_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 28: Tốc độ của fruit\_original.

### 4.1.3 Điều chỉnh siêu tham số

#### 4.1.3.1 fm

- Điều chỉnh các siêu tham số của YOLOv8 để huấn luyện mô hình fm\_v1:

```
#FaceMask_TunedHyperparameters: fm_v1
!yolo detect train model=yolov8l.pt data=datasets/fm_data.yaml workers=2 batch=12 device=0 epochs=100 patience=30 name=fm_v1 lr0=0.01 lrf=0.01 momentum=0.937

weight_decay=0.0001 warmup_epochs=10 warmup_momentum=0.5 warmup_bias_lr=0.1 optimizer=SGD
```

Hình 29: Sử dụng API để huấn luyện fm\_v1.

- Kết quả huấn luyện:

```
EarlyStopping: Training stopped early as no improvement observed in last 30 epochs. Best results observed at epoch 32, best model saved as best.pt.
To update EarlyStopping(patience=30) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

62 epochs completed in 0.647 hours.
Optimizer stripped from runs/detect/fm_v1/weights/last.pt, 87.6MB
Optimizer stripped from runs/detect/fm_v1/weights/best.pt, 87.6MB

Validating runs/detect/fm_v1/weights/best.pt...
Ultralytics YOLOv8.2.71 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs
    Class   Images   Instances   Box(P     R     mAP50     mAP50-95): 100% 7/7 [00:11<00:00,  1.63s/it]
           all       160      866     0.879     0.884     0.916     0.643
           Mask      142      709     0.948     0.914     0.959     0.682
           No Mask    54      133     0.86      0.92     0.946     0.658
           Bad Mask   20       24     0.831     0.817     0.843     0.589
Speed: 1.8ms preprocess, 15.1ms inference, 0.0ms loss, 8.0ms postprocess per image
```

Hình 30: Kết quả huấn luyện của fm\_v1.

- Các chỉ số đo độ chính xác:

## Chương 4: Thực nghiệm

```
#fm_v1
!yolo detect val model=runs/detect/fm_v1/weights/best.pt data=datasets/fm_data.yaml iou=0.5 imgs=640 half=True name=val_fm_v1

Ultralytics YOLOv8.2.72 🦄 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fm_labels/val.cache...
    Class   Images Instances   Box(P)      R      mAP50  mAP50-95: 100% 10/10 [00:10<00:00,  1.08s/it]
        all     160       866    0.881    0.883    0.915    0.643
        Mask    142       709    0.952    0.913    0.957    0.682
        No Mask  54       133    0.86     0.921    0.948    0.66
        Bad Mask 20        24    0.83     0.817    0.841    0.586
Speed: 1.8ms preprocess, 36.5ms inference, 0.0ms loss, 15.4ms postprocess per image
```

Hình 31: Các chỉ số đo độ chính xác của fm\_v1.

- Tốc độ:

```
#fm_v1
!yolo detect predict model=runs/detect/fm_v1/weights/best.pt source=inference/fm_img_test.jpg save=True name=fm_img_test project=result/fm_v1
!yolo detect predict model=runs/detect/fm_v1/weights/best.pt source=inference/fm_img_test2.jpeg save=True name=fm_img_test2 project=result/fm_v1
!yolo detect predict model=runs/detect/fm_v1/weights/best.pt source=inference/fm_img_test3.jpg save=True name=fm_img_test3 project=result/fm_v1
!yolo detect predict model=runs/detect/fm_v1/weights/best.pt source=inference/fm_img_test4.jpg save=True name=fm_img_test4 project=result/fm_v1

Ultralytics YOLOv8.2.72 🦄 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test.jpg: 384x640 18 Masks, 94.0ms
Speed: 4.5ms preprocess, 94.0ms inference, 569.2ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_v1/fm_img_test
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🦄 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test2.jpeg: 448x640 9 Masks, 5 No Masks, 83.5ms
Speed: 5.5ms preprocess, 83.5ms inference, 708.9ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fm_v1/fm_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🦄 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test3.jpg: 384x640 22 Masks, 2 No Masks, 2 Bad Masks, 98.6ms
Speed: 3.5ms preprocess, 98.6ms inference, 843.5ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_v1/fm_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🦄 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test4.jpg: 384x640 10 Masks, 100.8ms
Speed: 3.7ms preprocess, 100.8ms inference, 559.2ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_v1/fm_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 32: Tốc độ của fm\_v1.

### 4.1.3.2 fruit\_fr

- Điều chỉnh các siêu tham số của YOLOv8 để huấn luyện mô hình fruit\_v1:

```
#FruitFreshness_TunedHyperparameters: fruit_v1
!yolo detect train model=yolov8l.pt data=datasets/data.yaml workers=2 batch=16 device=0 epochs=100 patience=30 name=fruit_v1 lr=0.01 lrf=0.01 momentum=0.937

weight_decay=0.0001 warmup_epochs=10 warmup_momentum=0.5 warmup_bias_lr=0.1 optimizer=SGD
```

Hình 33: Sử dụng API để huấn luyện fruit\_v1.

- Kết quả huấn luyện:

## Chương 4: Thực nghiệm

```
66 epochs completed in 10.585 hours.  
Optimizer stripped from runs/detect/fruit_v1/weights/last.pt, 87.7MB  
Optimizer stripped from runs/detect/fruit_v1/weights/best.pt, 87.7MB  
  
Validating runs/detect/fruit_v1/weights/best.pt...  
Ultralytics YOLOv8.2.42 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43609692 parameters, 0 gradients, 164.8 GFLOPs  
    Class     Images Instances   Box(P      R      mAP50      mAP50-95): 100% 33/33 [00:27<00:00,  1.20it/s]  
        all      1030     1309   0.922     0.829     0.89      0.636  
    Fresh Apple    281      303   0.979     0.941     0.985     0.819  
    Fresh Banana    220      385   0.873     0.642     0.782     0.456  
    Rotten Apple    278      315   0.928     0.99      0.972     0.78  
    Rotten Banana    249      306   0.907     0.745     0.821     0.489  
Speed: 0.3ms preprocess, 16.4ms inference, 0.0ms loss, 2.6ms postprocess per image
```

Hình 34: Kết quả huấn luyện của fruit\_v1.

- Các chỉ số đo độ chính xác:

```
#fruit_v1  
!yolo detect val model=runs/detect/fruit_v1/weights/best.pt data=datasets/data.yaml iou=0.5 imgs=640 half=True name=val_fruit_v1  
  
Ultralytics YOLOv8.2.46 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43609692 parameters, 0 gradients, 164.8 GFLOPs  
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fruit_fr/labels/val.cache... 1030 images, 2 backgrounds, 0 corrupt: 100% 1030/1030 [00:00<?, ?it/s]  
    Class     Images Instances   Box(P      R      mAP50      mAP50-95): 100% 65/65 [00:31<00:00,  2.08it/s]  
        all      1030     1309   0.926     0.832     0.893     0.639  
    Fresh Apple    281      303   0.979     0.947     0.981     0.819  
    Fresh Banana    220      385   0.901     0.64      0.798     0.462  
    Rotten Apple    278      315   0.925     0.99      0.972     0.781  
    Rotten Banana    249      306   0.898     0.749     0.822     0.495  
Speed: 0.7ms preprocess, 15.3ms inference, 0.0ms loss, 2.4ms postprocess per image
```

Hình 35: Các chỉ số đo độ chính xác của fruit\_v1.

- Tốc độ:

```
#fruit_v1  
!yolo detect predict model=runs/detect/fruit_v1/weights/best.pt source=inference/fruit_img_test.jpg save=True name=fruit_img_test project=result/fruit_v1  
!yolo detect predict model=runs/detect/fruit_v1/weights/best.pt source=inference/fruit_img_test2.jpg save=True name=fruit_img_test2 project=result/fruit_v1  
!yolo detect predict model=runs/detect/fruit_v1/weights/best.pt source=inference/fruit_img_test3.jpg save=True name=fruit_img_test3 project=result/fruit_v1  
!yolo detect predict model=runs/detect/fruit_v1/weights/best.pt source=inference/fruit_img_test4.jpg save=True name=fruit_img_test4 project=result/fruit_v1  
  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test.jpg: 640x640 1 Fresh Apple, 62.8ms  
Speed: 5.6ms preprocess, 62.8ms inference, 1065.7ms postprocess per image at shape (1, 3, 640, 640)  
Results saved to result/fruit_v1/fruit_img_test  
💡 Learn more at https://docs.ultralytics.com/modes/predict  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test2.jpg: 512x640 1 Rotten Apple, 115.3ms  
Speed: 15.0ms preprocess, 115.3ms inference, 746.5ms postprocess per image at shape (1, 3, 512, 640)  
Results saved to result/fruit_v1/fruit_img_test2  
💡 Learn more at https://docs.ultralytics.com/modes/predict  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test3.jpg: 448x640 1 Rotten Banana, 134.5ms  
Speed: 7.2ms preprocess, 134.5ms inference, 756.0ms postprocess per image at shape (1, 3, 448, 640)  
Results saved to result/fruit_v1/fruit_img_test3  
💡 Learn more at https://docs.ultralytics.com/modes/predict  
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs  
  
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test4.jpg: 448x640 1 Fresh Banana, 265.7ms  
Speed: 4.2ms preprocess, 265.7ms inference, 865.3ms postprocess per image at shape (1, 3, 448, 640)  
Results saved to result/fruit_v1/fruit_img_test4  
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 36: Tốc độ của fruit\_v1.

#### 4.1.4 Tăng cường dữ liệu

##### 4.1.4.1 fm

- Áp dụng các kĩ thuật tăng cường dữ liệu để huấn luyện mô hình fm\_v2:

```
#FaceMask_DataAugmentation: fm_v2
!yolo detect train model=yolov8l.pt data=datasets/fm_data.yaml workers=2 batch=12 device=0 epochs=100 patience=30 name=fm_v2 hsv_h=0.015 hsv_s=0.8 hsv_v=0.1
degrees=0.15 translate=0.1 scale=0.7 mosaic=0 mixup=1 flipud=0 fliplr=0.5 shear=25 perspective=0.001
```

Hình 37: Sử dụng API để huấn luyện fm\_v2.

- Kết quả huấn luyện:

```
100 epochs completed in 1.380 hours.
Optimizer stripped from runs/detect/fm_v2/weights/last.pt, 87.6MB
Optimizer stripped from runs/detect/fm_v2/weights/best.pt, 87.6MB

Validating runs/detect/fm_v2/weights/best.pt...
Ultralytics YOLOv8.2.71 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs
    Class      Images Instances   Box(P)      R      mAP50      mAP50-95: 100% 7/7 [00:10<00:00,  1.50s/it]
        all       160      866     0.942     0.849     0.929     0.607
        Mask      142      709     0.954     0.909     0.965     0.604
        No Mask    54      133     0.874     0.889     0.95      0.608
        Bad Mask   20       24     0.996     0.75     0.873     0.608
Speed: 1.0ms preprocess, 16.0ms inference, 0.3ms loss, 9.0ms postprocess per image
```

Hình 38: Kết quả huấn luyện của fm\_v2.

- Các chỉ số đo độ chính xác:

```
#fm_v2
!yolo detect val model=runs/detect/fm_v2/weights/best.pt data=datasets/fm_data.yaml iou=0.5 imgsz=640 half=True name=val_fm_v2

Ultralytics YOLOv8.2.71 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fm_labels/val.cache... 160 images, 0 backgrounds, 0 corrupt: 100% 160/160 [00:00<?, ?it/s]
    Class      Images Instances   Box(P)      R      mAP50      mAP50-95: 100% 10/10 [00:13<00:00,  1.31s/it]
        all       160      866     0.942     0.849     0.928     0.607
        Mask      142      709     0.955     0.908     0.965     0.607
        No Mask    54      133     0.874     0.888     0.945     0.604
        Bad Mask   20       24     0.997     0.75     0.874     0.609
Speed: 1.9ms preprocess, 47.9ms inference, 0.0ms loss, 11.6ms postprocess per image
```

Hình 39: Các chỉ số đo độ chính xác của fm\_v2.

- Tốc độ:

## Chương 4: Thực nghiệm

```
#fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test.jpg save=True name=fm_img_test project=result/fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test2.jpeg save=True name=fm_img_test2 project=result/fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test3.jpg save=True name=fm_img_test3 project=result/fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.pt source=inference/fm_img_test4.jpg save=True name=fm_img_test4 project=result/fm_original

Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test.jpg: 384x640 19 Masks, 75.1ms
Speed: 3.2ms preprocess, 75.1ms inference, 510.0ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_original/fm_img_test
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test2.jpeg: 448x640 8 Masks, 10 No Masks, 129.8ms
Speed: 5.6ms preprocess, 129.8ms inference, 979.9ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fm_original/fm_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test3.jpg: 384x640 14 Masks, 1 No Mask, 1 Bad Mask, 71.8ms
Speed: 2.3ms preprocess, 71.8ms inference, 557.5ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_original/fm_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,608,921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test4.jpg: 384x640 10 Masks, 72.8ms
Speed: 2.5ms preprocess, 72.8ms inference, 516.3ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_original/fm_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 40: Tốc độ của fm\_v2.

### 4.1.4.2 fruit\_fr

- Áp dụng các kĩ thuật tăng cường dữ liệu để huấn luyện mô hình fruit\_v3:

```
#FruitFreshness_DataAugmentation: fruit_v3
!yolo detect train model=yolov8l.pt data=data/datasets/data.yaml workers=2 batch=16 device=0 epochs=100 patience=30 name=fruit_v3 hsv_h=0.015 hsv_s=0.8 hsv_v=0.1

degrees=0.15 translate=0.1 scale=0.7 mosaic=0 mixup=1 flipud=1 fliplr=0.5 shear=25 perspective=0.001
```

Hình 41: Sử dụng API để huấn luyện fruit\_v3.

- Kết quả huấn luyện:

```
100 epochs completed in 19.428 hours.
Optimizer stripped from runs/detect/fruit_v3/weights/last.pt, 87.7MB
Optimizer stripped from runs/detect/fruit_v3/weights/best.pt, 87.7MB

Validating runs/detect/fruit_v3/weights/best.pt...
Ultralytics YOLOv8.2.48 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43609692 parameters, 0 gradients, 164.8 GFLOPs
    Class   Images  Instances      Box(P       R      mAP50      mAP50-95): 100% 33/33 [00:26<00:00,  1.25it/s]
          all      1030     1309     0.901     0.822     0.876     0.569
          Fresh Apple    281     303     0.911     0.934     0.967     0.754
          Fresh Banana   220     385     0.87     0.681     0.767     0.397
          Rotten Apple   278     315     0.95     0.94     0.971     0.732
          Rotten Banana   249     306     0.872     0.734     0.8     0.394
Speed: 0.3ms preprocess, 15.9ms inference, 0.0ms loss, 2.4ms postprocess per image
```

Hình 42: Kết quả huấn luyện của fruit\_v3.

- Các chỉ số đo độ chính xác:

## Chương 4: Thực nghiệm

```
#fruit_v3
yolo detect val model=runs/detect/fruit_v3/weights/best.pt data=datasets/data.yaml iou=0.5 imgs=640 half=True name=val_fruit_v3

Ultralytics YOLOv8.2.48 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43609692 parameters, 0 gradients, 164.8 GFLOPs
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fruit_fr/labels/val.cache... 1030 images, 2 backgrounds, 0 corrupt: 100% 1030/1030 [00:00<?, ?it/s]
    Class      Images Instances   Box(P)      R      mAP@0      mAP@50-95: 100% 65/65 [00:23<00:00,  2.73it/s]
        all       1030      1309     0.898     0.829     0.876     0.567
    Fresh Apple    281      303     0.895     0.937     0.967     0.756
    Fresh Banana    220      385     0.876     0.677     0.767     0.389
    Rotten Apple    278      315     0.946     0.948     0.973     0.736
    Rotten Banana    249      306     0.875     0.753     0.795     0.388
Speed: 0.3ms preprocess, 15.5ms inference, 0.0ms loss, 2.1ms postprocess per image
```

Hình 43: Các chỉ số đo độ chính xác của fruit\_v3.

- Tốc độ:

```
#fruit_v3
yolo detect predict model=runs/detect/fruit_v3/weights/best.pt source=inference/fruit_img_test.jpg save=True name=fruit_img_test project=result/fruit_v3
yolo detect predict model=runs/detect/fruit_v3/weights/best.pt source=inference/fruit_img_test2.jpg save=True name=fruit_img_test2 project=result/fruit_v3
yolo detect predict model=runs/detect/fruit_v3/weights/best.pt source=inference/fruit_img_test3.jpg save=True name=fruit_img_test3 project=result/fruit_v3
yolo detect predict model=runs/detect/fruit_v3/weights/best.pt source=inference/fruit_img_test4.jpg save=True name=fruit_img_test4 project=result/fruit_v3

Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test.jpg: 640x640 1 Fresh Apple, 70.2ms
Speed: 8.1ms preprocess, 70.2ms inference, 1121.5ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_v3/fruit_img_test
    Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test2.jpg: 512x640 1 Rotten Apple, 99.9ms
Speed: 4.7ms preprocess, 99.9ms inference, 509.2ms postprocess per image at shape (1, 3, 512, 640)
Results saved to result/fruit_v3/fruit_img_test2
    Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test3.jpg: 448x640 1 Rotten Banana, 186.2ms
Speed: 4.3ms preprocess, 186.2ms inference, 541.6ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fruit_v3/fruit_img_test3
    Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43,609,692 parameters, 0 gradients, 164.8 GFLOPs

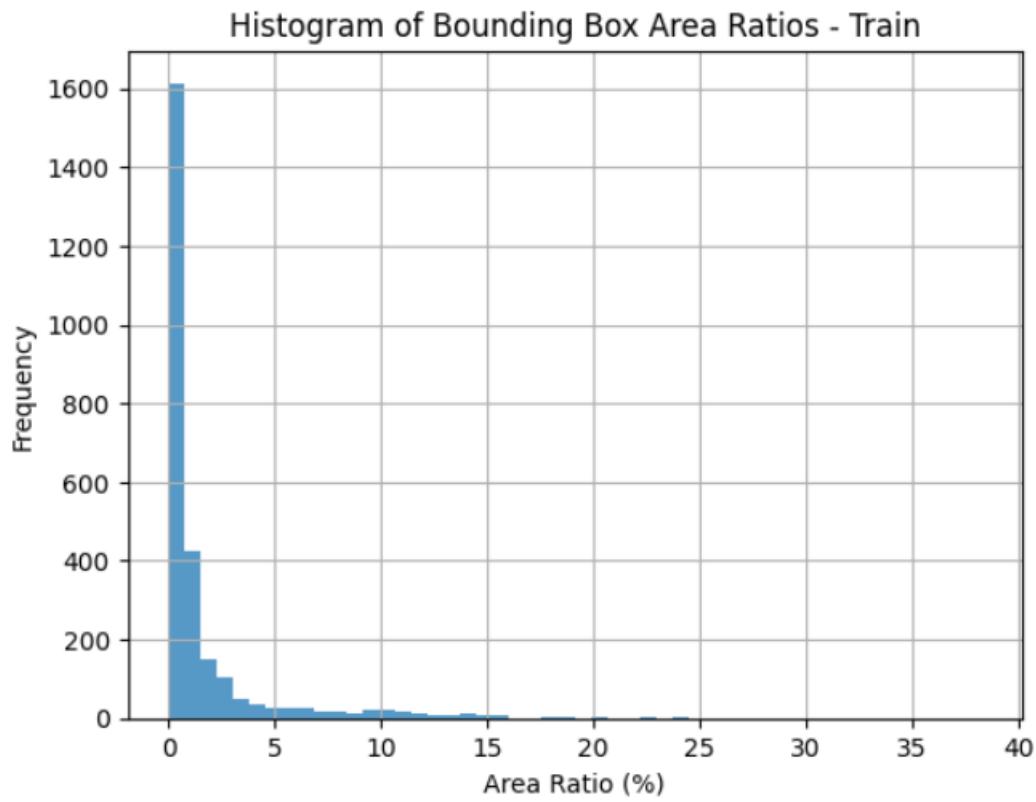
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test4.jpg: 448x640 1 Fresh Banana, 102.3ms
Speed: 3.7ms preprocess, 102.3ms inference, 547.6ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fruit_v3/fruit_img_test4
    Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 44: Tốc độ của fruit\_v3.

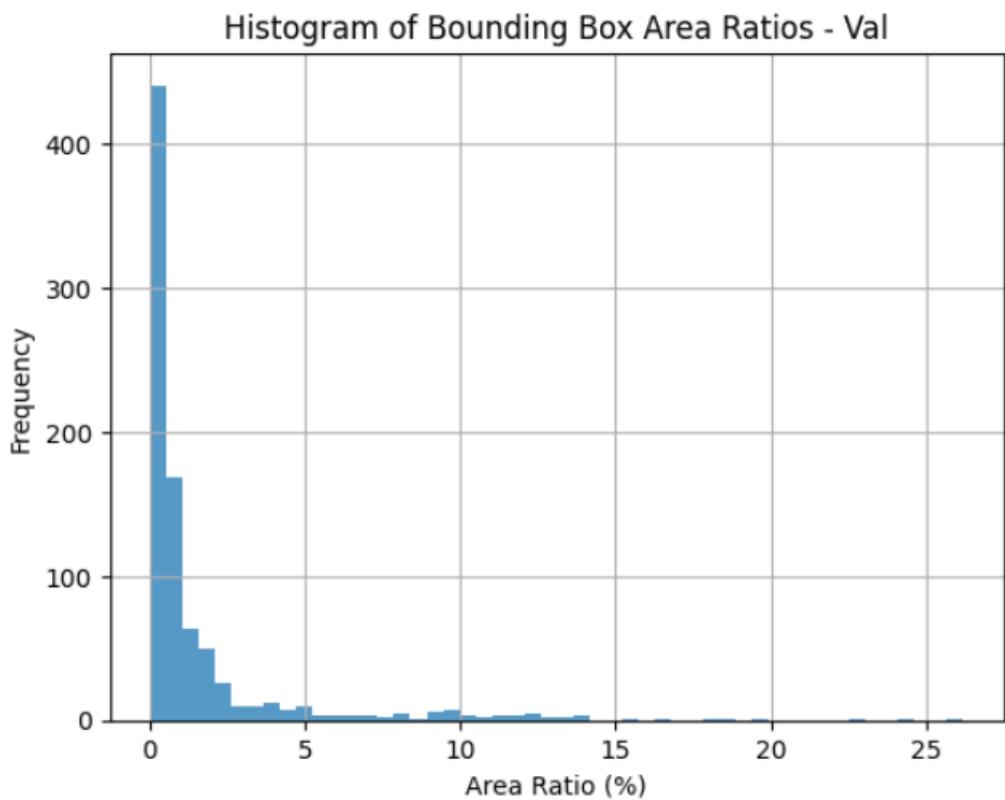
### 4.1.5 Thay đổi kiến trúc YOLOv8

#### 4.1.5.1 fm

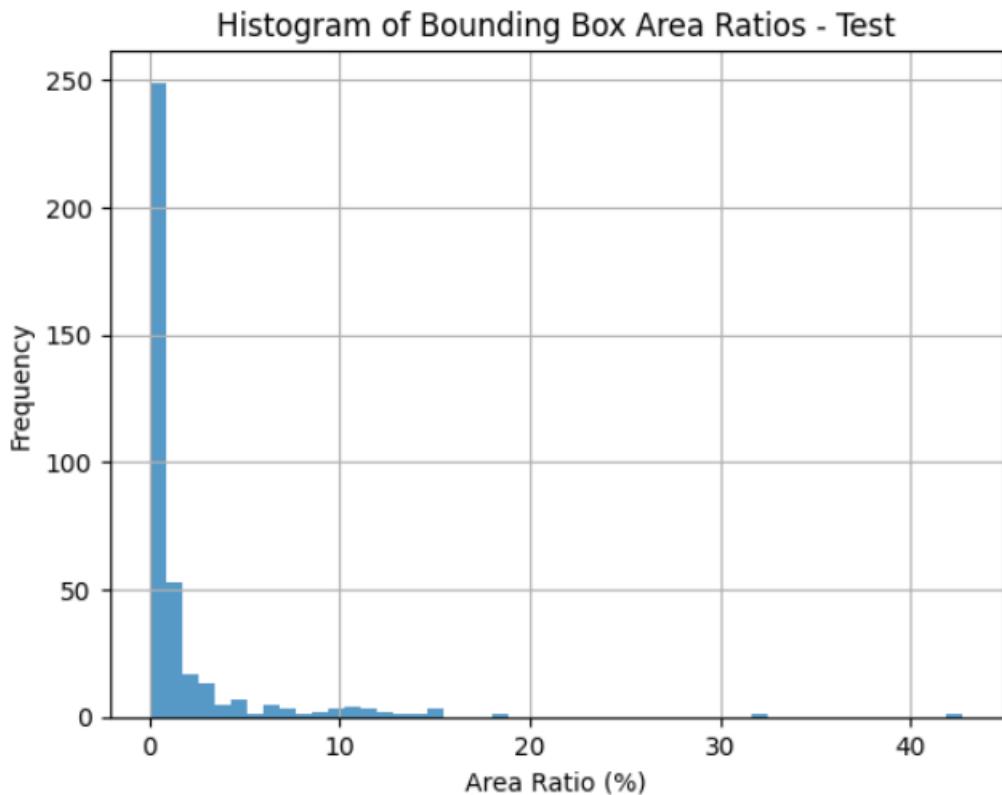
- Plot biểu đồ histogram về tỉ lệ hình ảnh:



*Hình 45: Histogram plot của tập train thuộc tập fm.*



Hình 46: Histogram plot của tập val thuộc tập fm.



Hình 47: Histogram plot của tập test thuộc tập fm.

- Tạo file YOLOv8l-s\_m.yaml:

## Chương 4: Thực nghiệm

```
# Ultralytics YOLO 🚀, AGPL-3.0 license
# YOLOv8 object detection model with P3-P5 outputs. For Usage examples see https://docs.ultralytics.com/tasks/detect

# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolov8n.yaml' will call yolov8.yaml with scale 'n'
# [depth, width, max_channels]
n: [0.33, 0.25, 1024] # YOLOv8n summary: 225 layers, 3157200 parameters, 3157184 gradients, 8.9 GFLOPs
s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers, 11166560 parameters, 11166544 gradients, 28.8 GFLOPs
m: [0.67, 0.75, 768] # YOLOv8m summary: 295 layers, 25902640 parameters, 25902624 gradients, 79.3 GFLOPs
l: [1.00, 1.00, 512] # YOLOv8l summary: 365 layers, 43691520 parameters, 43691504 gradients, 165.7 GFLOPs
x: [1.00, 1.25, 512] # YOLOv8x summary: 365 layers, 68229648 parameters, 68229632 gradients, 258.5 GFLOPs

# YOLOv8.0n backbone
backbone:
    # [from, repeats, module, args]
    - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
    - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
    - [-1, 3, C2f, [128, True]]
    - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
    - [-1, 6, C2f, [256, True]]
    - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
    - [-1, 6, C2f, [512, True]]
    - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
    - [-1, 3, C2f, [1024, True]]
    - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
    - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
    - [[-1, 6], 1, Concat, [1]] # cat backbone P4
    - [-1, 3, C2f, [512]] # 12

    - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
    - [[-1, 4], 1, Concat, [1]] # cat backbone P3
    - [-1, 3, C2f, [256]] # 15 (P3/8-small)

    - [-1, 1, Conv, [256, 3, 2]]
    - [[-1, 12], 1, Concat, [1]] # cat head P4
    - [-1, 3, C2f, [512]] # 18 (P4/16-medium)

    # - [-1, 1, Conv, [512, 3, 2]]
    # - [[-1, 9], 1, Concat, [1]] # cat head P5
    # - [-1, 3, C2f, [1024]] # 21 (P5/32-large)

    - [[15, 18], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

Hình 48: Nội dung file YOLOv8l-s\_m.yaml

- Huấn luyện mô hình fm\_v3:

```
#FaceMask_ArchitectureModified[small + medium]: fm_v3
yolo detect train model=yolov8l-s_m.yaml data=datasets/fm_data.yaml workers=2 batch=12 device=0 epochs=100 patience=30 name=fm_v3
```

Hình 49: Sử dụng API để huấn luyện fm\_v3.

- Kết quả huấn luyện:

```
100 epochs completed in 1.022 hours.
Optimizer stripped from runs/detect/fm_v3/weights/last.pt, 69.2MB
Optimizer stripped from runs/detect/fm_v3/weights/best.pt, 69.2MB

Validating runs/detect/fm_v3/weights/best.pt...
Ultralytics YOLOv8.2.71 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-s_m summary (fused): 232 layers, 34,421,142 parameters, 0 gradients, 157.5 GFLOPs
      Class   Images Instances   Box(P       R       mAP50       mAP50-95): 100% 7/7 [00:11<00:00,  1.61s/it]
          all      160     866   0.893     0.812     0.894     0.593
          Mask     142     709   0.95      0.891     0.946     0.654
          No Mask    54     133   0.877     0.827     0.902     0.607
          Bad Mask   20      24   0.851     0.717     0.833     0.517
Speed: 1.0ms preprocess, 16.6ms inference, 0.0ms loss, 10.0ms postprocess per image
```

## Chương 4: Thực nghiệm

Hình 50: Kết quả huấn luyện của fm\_v3.

- Các chỉ số đo độ chính xác:

```
#fm_v3
!yolo detect val model=runs/detect/fm_v3/weights/best.pt data=datasets/fm_data.yaml iou=0.5 imgs=640 half=True name=val_fm_v3

Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-s_m summary (fused): 232 layers, 34,421,142 parameters, 0 gradients, 157.5 GFLOPs
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fm_labels/val.cache...
    Class   Images Instances   Box(P)      R      mAP50  mAP50-95: 100% 10/10 [00:10<0:00, 1.08s/it]
        all     160      866    0.895    0.814    0.893    0.593
        Mask    142      709    0.951    0.891    0.948    0.656
    No Mask    54      133    0.883    0.827    0.906    0.609
    Bad Mask   20       24    0.852    0.723    0.826    0.513
Speed: 1.4ms preprocess, 29.0ms inference, 0.0ms loss, 15.3ms postprocess per image
```

Hình 51: Các chỉ số đo độ chính xác của fm\_v3.

- Tốc độ:

```
#fm_v3
!yolo detect predict model=runs/detect/fm_v3/weights/best.pt source=inference/fm_img_test.jpg save=True name=fm_img_test project=result/fm_v3.1
!yolo detect predict model=runs/detect/fm_v3/weights/best.pt source=inference/fm_img_test2.jpeg save=True name=fm_img_test2 project=result/fm_v3.1
!yolo detect predict model=runs/detect/fm_v3/weights/best.pt source=inference/fm_img_test3.jpg save=True name=fm_img_test3 project=result/fm_v3.1
!yolo detect predict model=runs/detect/fm_v3/weights/best.pt source=inference/fm_img_test4.jpg save=True name=fm_img_test4 project=result/fm_v3.1

Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-s_m summary (fused): 232 layers, 34,421,142 parameters, 0 gradients, 157.5 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test.jpg: 384x640 21 Masks, 1 No Mask, 67.3ms
Speed: 3.2ms preprocess, 67.3ms inference, 633.9ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_v3.1/fm_img_test
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-s_m summary (fused): 232 layers, 34,421,142 parameters, 0 gradients, 157.5 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test2.jpeg: 448x640 10 Masks, 6 No Masks, 1 Bad Mask, 67.6ms
Speed: 4.6ms preprocess, 67.6ms inference, 558.9ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fm_v3.1/fm_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-s_m summary (fused): 232 layers, 34,421,142 parameters, 0 gradients, 157.5 GFLOPs

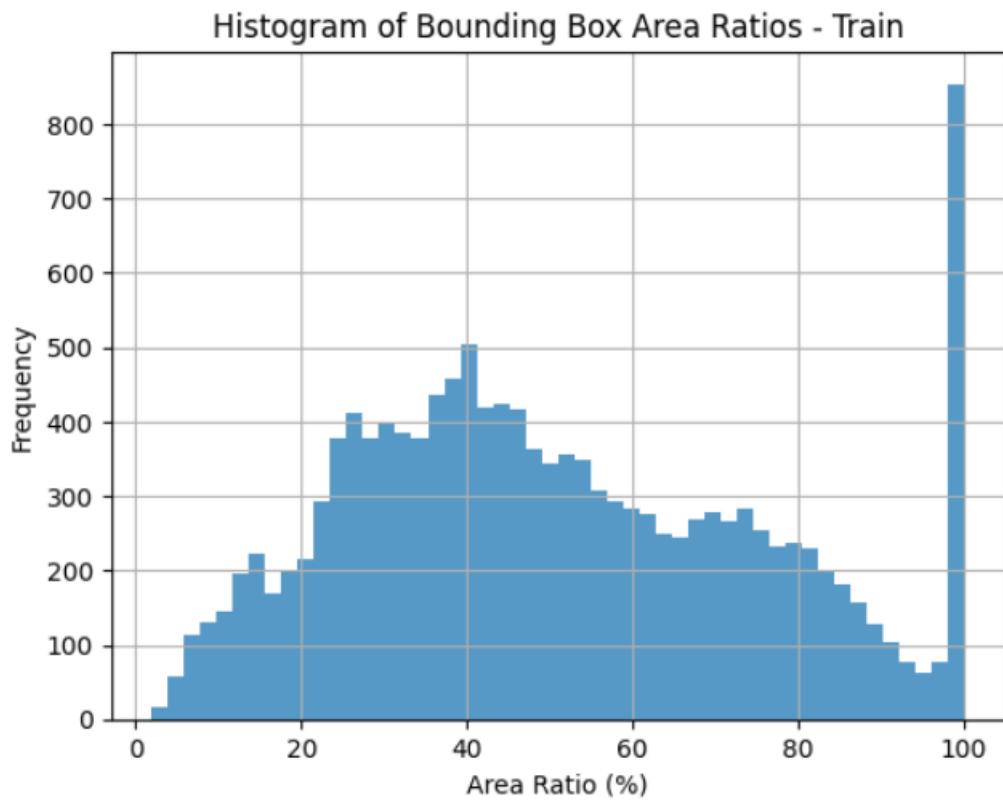
image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test3.jpg: 384x640 11 Masks, 1 No Mask, 1 Bad Mask, 67.1ms
Speed: 2.3ms preprocess, 67.1ms inference, 670.7ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_v3.1/fm_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-s_m summary (fused): 232 layers, 34,421,142 parameters, 0 gradients, 157.5 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test4.jpg: 384x640 9 Masks, 67.0ms
Speed: 2.7ms preprocess, 67.0ms inference, 551.6ms postprocess per image at shape (1, 3, 384, 640)
Results saved to result/fm_v3.1/fm_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

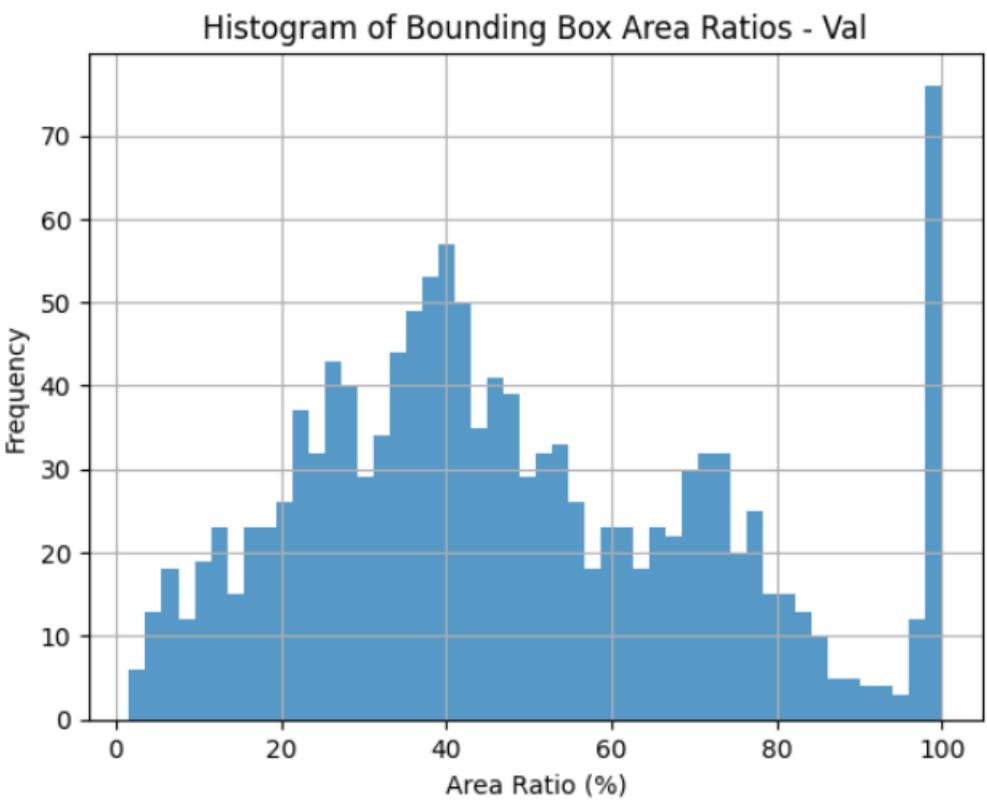
Hình 52: Tốc độ của fm\_v3.

### 4.1.5.2 fruit\_fr

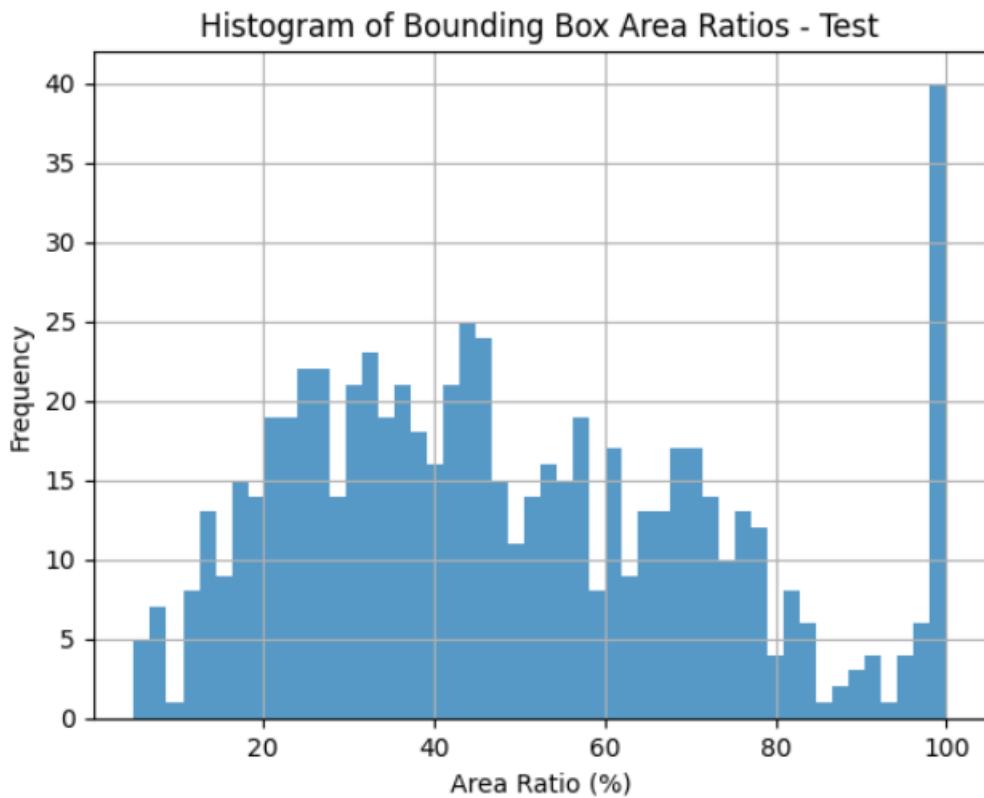
- Plot biểu đồ histogram về tỉ lệ hình ảnh:



*Hình 53: Histogram plot của tập train thuộc tập fruit\_fr.*



Hình 54: Histogram plot của tập val thuộc tập fruit\_fr.



Hình 55: Histogram plot của tập test thuộc tập fruit\_fr.

- Tạo file YOLOv8l-m.yaml:

## Chương 4: Thực nghiệm

```

# Ultralytics YOLO 🚀, AGPL-3.0 license
# YOLOv8 object detection model with P3-P5 outputs. For Usage examples see https://docs.ultralytics.com/tasks/detect

# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolov8n.yaml' will call yolov8.yaml with scale 'n'
  # [depth, width, max_channels]
n: [0.33, 0.25, 1024] # YOLOv8n summary: 225 layers, 3157200 parameters, 3157184 gradients, 8.9 GFLOPs
s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers, 11166560 parameters, 11166544 gradients, 28.8 GFLOPs
m: [0.67, 0.75, 768] # YOLOv8m summary: 295 layers, 25902640 parameters, 25902624 gradients, 79.3 GFLOPs
l: [1.00, 1.00, 512] # YOLOv8l summary: 365 layers, 43691520 parameters, 43691504 gradients, 165.7 GFLOPs
x: [1.00, 1.25, 512] # YOLOv8x summary: 365 layers, 68229648 parameters, 68229632 gradients, 258.5 GFLOPs

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 3, C2f, [512]] # 12

  # - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  # - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  # - [-1, 3, C2f, [256]] # 15 (P3/8-small)

  - [4, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]] # cat head P4
  - [-1, 3, C2f, [512]] # 18 (P4/16-medium) -> 15

  # - [-1, 1, Conv, [512, 3, 2]]
  # - [[-1, 9], 1, Concat, [1]] # cat head P5
  # - [-1, 3, C2f, [1024]] # 21 (P5/32-large)

  - [[15], 1, Detect, [nc]] # Detect(P3, P4, P5)

```

Hình 56: Nội dung file YOLOv8l-m.yaml

- Huấn luyện mô hình fruit\_v2:

```
#FruitFreshness: fruit_v2
!yolo detect train model=yolov8l-m.yaml data=datasets/data.yaml workers=2 batch=16 device=0 epochs=100 patience=30 name=fruit_v2
```

Hình 57: Sử dụng API để huấn luyện fruit\_v2.

- Kết quả huấn luyện:

```

EarlyStopping: Training stopped early as no improvement observed in last 30 epochs. Best results observed at epoch 51, best model saved as best.pt.
To update EarlyStopping(patience=30) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

81 epochs completed in 10.848 hours.
Optimizer stripped from runs/detect/fruit_v23/weights/last.pt, 70.6MB
Optimizer stripped from runs/detect/fruit_v23/weights/best.pt, 70.6MB

Validating runs/detect/fruit_v23/weights/best.pt...
Ultralytics YOLOv8.2.46 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA-0 (Tesla T4, 15102MiB)
YOLOv8l-medium summary (fused): 197 layers, 35165780 parameters, 0 gradients, 134.8 GFLOPs
      Class   Images  Instances   Box(P      R      mAP50      mAP50-95): 100% 33/33 [00:26<00:00,  1.23it/s]
      all     1030    1309    0.912    0.821    0.885    0.485
      Fresh Apple    281    303    0.963    0.94    0.972    0.569
      Fresh Banana    220    385    0.869    0.605    0.778    0.315
      Rotten Apple    278    315    0.92    0.968    0.976    0.666
      Rotten Banana    249    306    0.895    0.771    0.815    0.391
Speed: 0.4ms preprocess, 12.7ms inference, 0.0ms loss, 3.3ms postprocess per image

```

## Chương 4: Thực nghiệm

Hình 58: Kết quả huấn luyện của fruit\_v2.

- Các chỉ số đo độ chính xác:

```
#fruit_v2
!yolo detect val model=runs/detect/fruit_v2/weights/best.pt data=datasets/data.yaml iou=0.5 imgsz=640 half=True name=val_fruit_v2

Ultralytics YOLOv8.2.48 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-medium summary (fused): 197 layers, 35165780 parameters, 0 gradients, 134.8 GFLOPs
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fruit_fr/labels/val.cache...
    Class   Images Instances   Box(P)      R      mAP50  mAP50-95: 100% 1030/1030 [00:00<?, ?it/s]
    all     1030    1309   0.921    0.827    0.89    0.489
    Fresh Apple  281    303   0.963    0.944    0.967    0.569
    Fresh Banana 220    385   0.985    0.619    0.795    0.318
    Rotten Apple 278    315   0.911    0.978    0.977    0.67
    Rotten Banana 249    306   0.905    0.768    0.822    0.397
Speed: 0.5ms preprocess, 12.7ms inference, 0.0ms loss, 2.7ms postprocess per image
```

Hình 59: Các chỉ số đo độ chính xác của fruit\_v2.

- Tốc độ:

```
#fruit_v2
!yolo detect predict model=runs/detect/fruit_v2/weights/best.pt source=inference/fruit_img_test.jpg save=True name=fruit_img_test project=result/fruit_v2.1
!yolo detect predict model=runs/detect/fruit_v2/weights/best.pt source=inference/fruit_img_test2.jpg save=True name=fruit_img_test2 project=result/fruit_v2.1
!yolo detect predict model=runs/detect/fruit_v2/weights/best.pt source=inference/fruit_img_test3.jpg save=True name=fruit_img_test3 project=result/fruit_v2.1
!yolo detect predict model=runs/detect/fruit_v2/weights/best.pt source=inference/fruit_img_test4.jpg save=True name=fruit_img_test4 project=result/fruit_v2.1

Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-medium summary (fused): 197 layers, 35,165,780 parameters, 0 gradients, 134.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test.jpg: 640x640 1 Fresh Apple, 50.2ms
Speed: 15.0ms preprocess, 50.2ms inference, 756.4ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_v2.1/fruit_img_test
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-medium summary (fused): 197 layers, 35,165,780 parameters, 0 gradients, 134.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test2.jpg: 512x640 1 Rotten Apple, 62.0ms
Speed: 3.5ms preprocess, 62.0ms inference, 561.2ms postprocess per image at shape (1, 3, 512, 640)
Results saved to result/fruit_v2.1/fruit_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-medium summary (fused): 197 layers, 35,165,780 parameters, 0 gradients, 134.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test3.jpg: 448x640 1 Rotten Banana, 83.2ms
Speed: 4.5ms preprocess, 83.2ms inference, 858.1ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fruit_v2.1/fruit_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.73 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8l-medium summary (fused): 197 layers, 35,165,780 parameters, 0 gradients, 134.8 GFLOPs

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test4.jpg: 448x640 1 Fresh Banana, 53.5ms
Speed: 2.7ms preprocess, 53.5ms inference, 687.7ms postprocess per image at shape (1, 3, 448, 640)
Results saved to result/fruit_v2.1/fruit_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 60: Tốc độ của fruit\_v2.

### 4.1.6 TensorRT

#### 4.1.6.1 fm

- Chuyển đổi fm\_original sang TensorRT:

```
# Export YOLOv8 Model to Tensorrt: fm_original
!yolo export model=/content/gdrive/MyDrive/yolov8_KLTN/runs/detect/fm_original/weights/best.pt format=engine half=True device=0 workspace=8
```

Hình 61: Xuất fm\_original sang kiểu TensorRT.

## Chương 4: Thực nghiệm

- Các chỉ số đo độ chính xác:

```
#fm_tensorrt
!yolo detect val model=runs/detect/fm_original/weights/best.engine data=datasets/fm_data.yaml iou=0.5 imgsz=640 half=True name=val_fm_tensorrt

Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fm_original/weights/best.engine for TensorRT inference...
[08/04/2024-15:27:12] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-15:27:12] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fm/labels/val.cache...
    Class   Images  Instances   Box(P      R      mAP50      mAP50-95: 100% 160/160 [00:06<00:00, 26.32it/s]
        all     160     866   0.913   0.824   0.909   0.627
        Mask    142     709   0.978   0.873   0.955   0.669
      No Mask    54     133   0.911   0.85    0.944   0.642
      Bad Mask   20      24   0.85    0.75   0.828   0.571
Speed: 1.2ms preprocess, 13.0ms inference, 0.1ms loss, 6.8ms postprocess per image
```

Hình 62: Các chỉ số đo độ chính xác của fm\_tensorrt.

- Tốc độ:

```
#fm_tensorrt
!yolo detect predict model=runs/detect/fm_original/weights/best.engine source=inference/fm_img_test.jpg save=True name=fm_img_test project=result/fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.engine source=inference/fm_img_test2.jpg save=True name=fm_img_test2 project=result/fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.engine source=inference/fm_img_test3.jpg save=True name=fm_img_test3 project=result/fm_original
!yolo detect predict model=runs/detect/fm_original/weights/best.engine source=inference/fm_img_test4.jpg save=True name=fm_img_test4 project=result/fm_original

Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fm_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:32:28] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:32:28] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test.jpg: 640x640 19 Masks, 17.4ms
Speed: 36.1ms preprocess, 17.4ms inference, 620.8ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fm_original/fm_img_test5
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fm_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:32:34] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:32:34] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test2.jpg: 640x640 8 Masks, 17.4ms
Speed: 35.7ms preprocess, 17.4ms inference, 614.0ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fm_original/fm_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fm_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:32:40] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:32:40] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test3.jpg: 640x640 14 Masks, 1 No Mask, 1 Bad Mask, 17.4ms
Speed: 33.6ms preprocess, 17.4ms inference, 698.1ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fm_original/fm_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fm_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:32:46] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:32:46] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fm_img_test4.jpg: 640x640 10 Masks, 17.4ms
Speed: 34.4ms preprocess, 17.4ms inference, 733.4ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fm_original/fm_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 63: Tốc độ của fm\_tensorrt.

### 4.1.6.2 fruit\_fr

- Chuyển đổi fruit\_original sang TensorRT:

```
# Export YOLOv8 Model to Tensorrt
!yolo export model=/content/gdrive/MyDrive/yolov8_KLTN/runs/detect/fruit_original/weights/best.pt format=engine half=True device=0 workspace=8
```

Hình 64: Xuất fruit\_original sang kiểu TensorRT.

## Chương 4: Thực nghiệm

- Các chỉ số đo độ chính xác:

```
#fruit_tensorrt
yolo detect val model=runs/detect/fruit_original/weights/best.engine data=datasets/data.yaml iou=0.5 imgsz=640 half=True name=val_fruit_tensorrt

Ultralytics YOLOv8.2.70 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fruit_original/weights/best.engine for TensorRT inference...
[08/01/2024-16:08:59] [TRT] [I] Loaded engine size: 85 MiB
[08/01/2024-16:09:00] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 22.8MB/s]
val: Scanning /content/gdrive/MyDrive/yolov8_KLTN/datasets/fruit_fr/labels/val.cache...
1030 images, 2 backgrounds, 0 corrupt: 100% 1030/1030 [00:00<?, ?it/s]
    Class   Images Instances   Box(P      R      mAP50      mAP50-95)
    all     1030      1309   0.911   0.829   0.872   0.579
    Fresh Apple   281      303   0.939   0.937   0.973   0.784
    Fresh Banana   220      385   0.877   0.629   0.745   0.387
    Rotten Apple   278      315   0.937   0.99    0.967   0.721
    Rotten Banana   249      306   0.893   0.76    0.805   0.424
Speed: 0.8ms preprocess, 10.9ms inference, 0.0ms loss, 2.8ms postprocess per image
```

Hình 65: Các chỉ số đo độ chính xác của fruit\_tensorrt.

- Tốc độ:

```
#fruit_tensorrt
yolo detect predict model=runs/detect/fruit_original/weights/best.engine source=inference/fruit_img_test.jpg save=True name=fruit_img_test project=result/fruit_tensorrt
yolo detect predict model=runs/detect/fruit_original/weights/best.engine source=inference/fruit_img_test2.jpg save=True name=fruit_img_test2 project=result/fruit_tensorrt
yolo detect predict model=runs/detect/fruit_original/weights/best.engine source=inference/fruit_img_test3.jpg save=True name=fruit_img_test3 project=result/fruit_tensorrt
yolo detect predict model=runs/detect/fruit_original/weights/best.engine source=inference/fruit_img_test4.jpg save=True name=fruit_img_test4 project=result/fruit_tensorrt

Ultralytics YOLOv8.2.72 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fruit_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:24:47] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:24:47] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test.jpg: 640x640 1 Fresh Apple, 17.4ms
Speed: 35.1ms preprocess, 17.4ms inference, 529.7ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_tensorrt/fruit_img_test
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fruit_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:24:54] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:24:54] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test2.jpg: 640x640 1 Rotten Apple, 17.4ms
Speed: 33.4ms preprocess, 17.4ms inference, 682.1ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_tensorrt/fruit_img_test2
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fruit_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:24:59] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:24:59] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test3.jpg: 640x640 1 Rotten Banana, 17.4ms
Speed: 35.7ms preprocess, 17.4ms inference, 658.3ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_tensorrt/fruit_img_test3
💡 Learn more at https://docs.ultralytics.com/modes/predict
Ultralytics YOLOv8.2.72 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Loading runs/detect/fruit_original/weights/best.engine for TensorRT inference...
[08/04/2024-19:25:05] [TRT] [I] Loaded engine size: 85 MiB
[08/04/2024-19:25:05] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +48, now: CPU 0, GPU 133 (MiB)

image 1/1 /content/gdrive/MyDrive/yolov8_KLTN/inference/fruit_img_test4.jpg: 640x640 1 Fresh Banana, 19.6ms
Speed: 64.8ms preprocess, 19.6ms inference, 819.8ms postprocess per image at shape (1, 3, 640, 640)
Results saved to result/fruit_tensorrt/fruit_img_test4
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Hình 66: Tốc độ của fruit\_tensorrt.

## 4.2 Kết quả

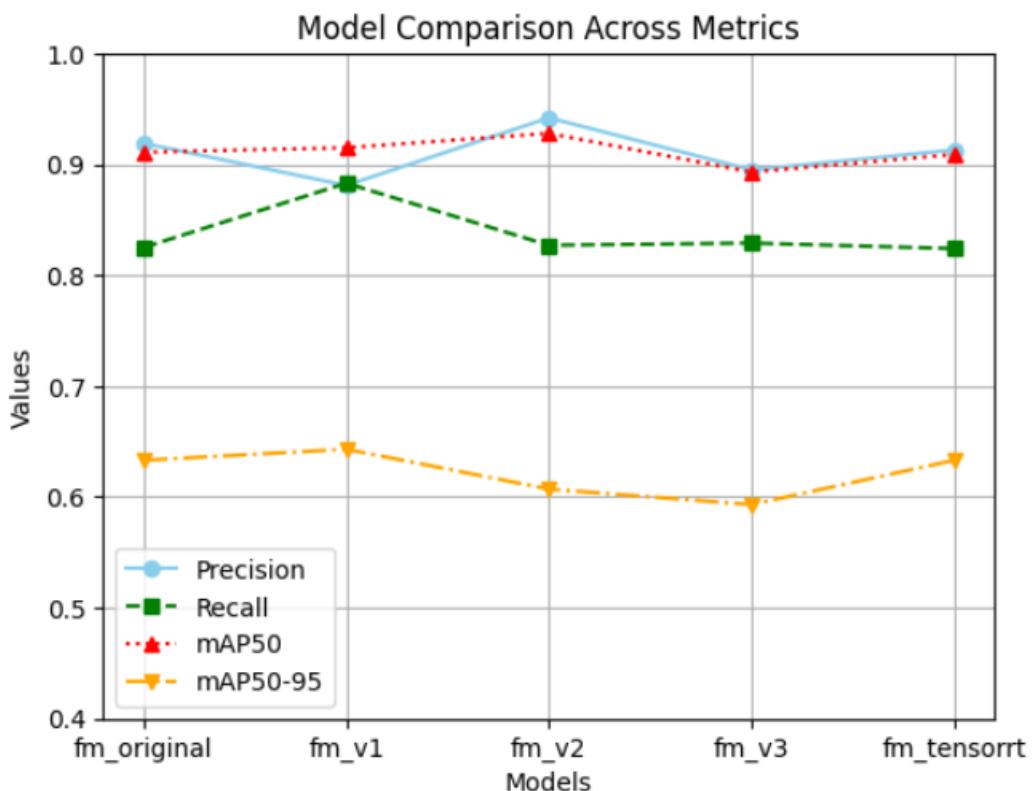
- Kết quả các phiên bản của mô hình đào tạo từ tập dữ liệu fm:

+ Các chỉ số đo độ chính xác:

Model	Precision	Recall	mAP50	mAP50-95
fm_original	0.919	0.825	0.911	0.633

fm_v1	0.881 (-0.038)	0.883 (+0.058)	0.915 (+0.004)	0.643 (+0.01)
fm_v2	0.942 (+0.023)	0.827 (+0.002)	0.928 (+0.017)	0.607 (-0.026)
fm_v3	0.895 (-0.024)	0.829 (+0.004)	0.893 (-0.018)	0.593 (-0.04)
fm_tensorrt	0.913 (-0.006)	0.824 (-0.001)	0.909 (-0.002)	0.633

Bảng 1: Các chỉ số đo độ chính xác của mô hình đào tạo từ fm.

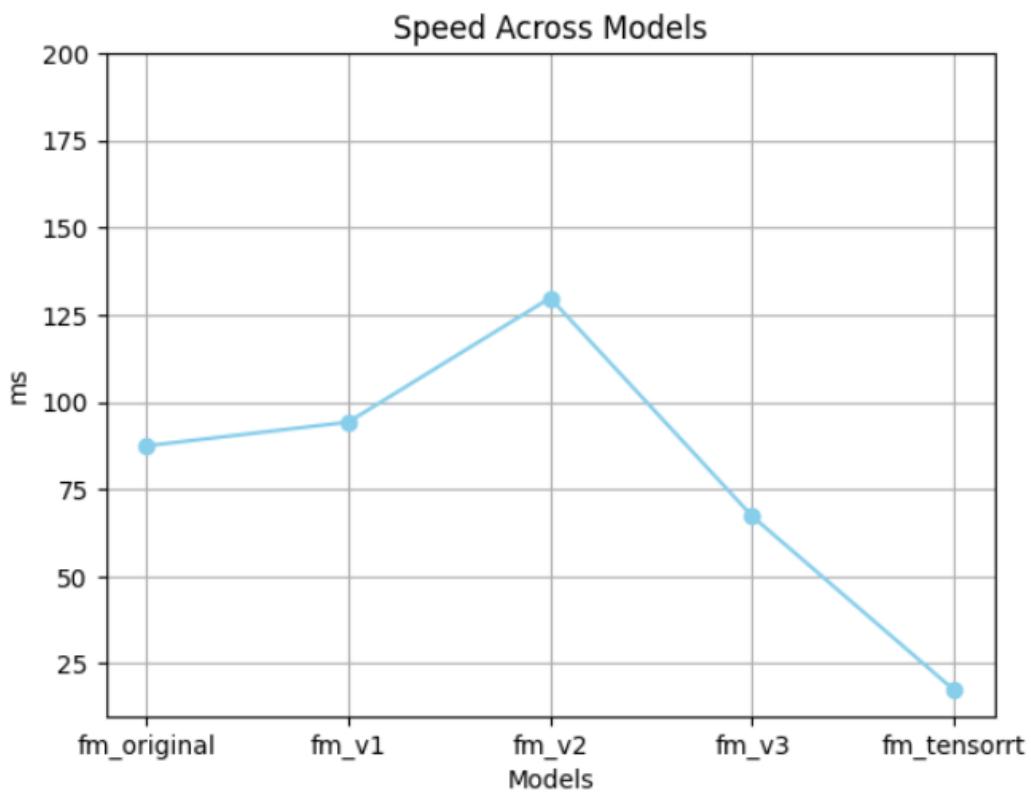


Hình 67: Biểu đồ so sánh các chỉ số đo độ chính xác của các model đào tạo từ fm.

+ Tốc độ (trung bình từ 4 lần detect ảnh):

Model	Inference
fm_original	87.4
fm_v1	94.2 (+6.8)
fm_v2	129.8 (+42.4)
fm_v3	67.25 (-20.05)
fm_tensorrt	17.4 (-70)

Bảng 2: Tốc độ của mô hình đào tạo từ fm.



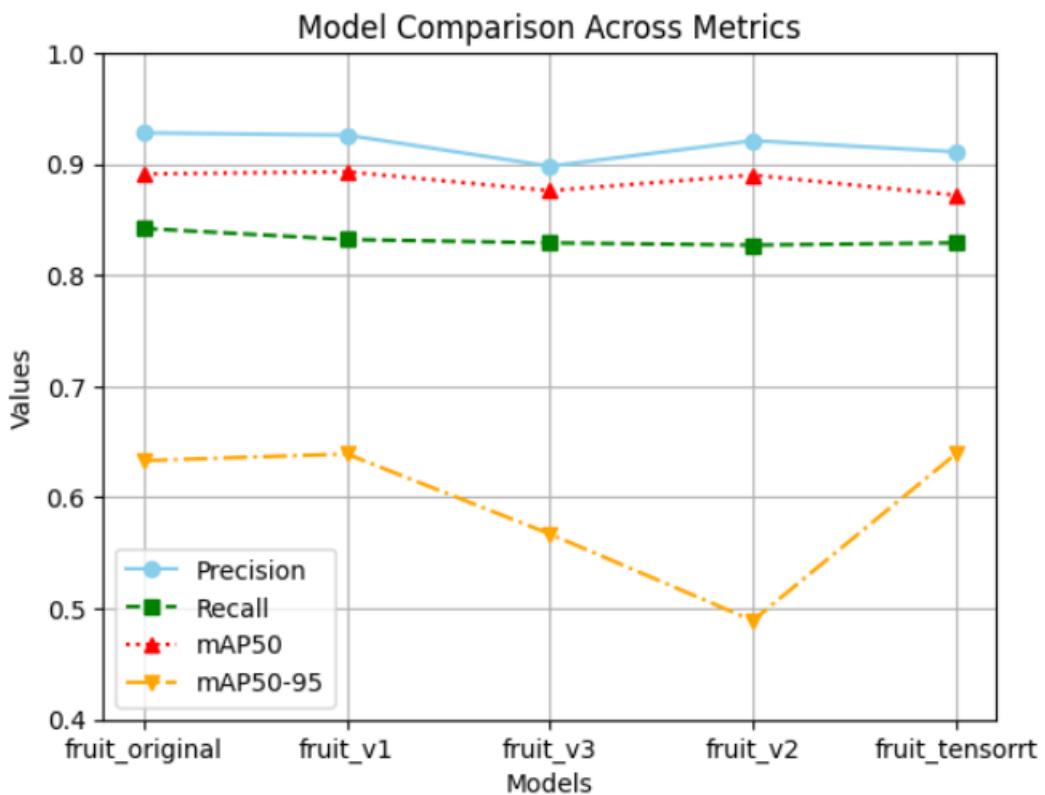
Hình 68: Biểu đồ so sánh tốc độ của các model đào tạo từ fm.

- Kết quả các phiên bản của mô hình đào tạo từ tập dữ liệu fruit\_fr:

+ Các chỉ số đo độ chính xác:

Model	Precision	Recall	mAP50	mAP50-95
fruit_original	0.928	0.842	0.891	0.633
fruit_v1	0.926 (-0.002)	0.832 (-0.01)	0.893 (+0.002)	0.639 (+0.006)
fruit_v2	0.921 (-0.007)	0.827 (-0.015)	0.89 (-0.001)	0.489 (-0.144)
fruit_v3	0.898 (-0.03)	0.829 (-0.013)	0.876 (-0.015)	0.567 (-0.066)
fruit_tensorrt	0.911 (-0.017)	0.829 (-0.013)	0.872 (-0.019)	0.639 (+0.066)

Bảng 3: Các chỉ số đo độ chính xác của mô hình đào tạo từ fruit\_fr.



Hình 67: Biểu đồ so sánh các chỉ số đo độ chính xác của các model đào tạo từ fruit\_fr.

+ Tốc độ (trung bình từ 4 lần detect ảnh):

Model	Inference
fruit_original	99.98
fruit_v1	144.58 (+44.6)
fruit_v2	62.225 (-37.755)
fruit_v3	114.65 (+14.67)
fruit_tensorrt	17.84 (-82.14)

Bảng 4: Tốc độ của mô hình đào tạo từ fruit\_fr.



*Hình 70: Biểu đồ so sánh tốc độ của các model đào tạo từ fruit\_fr.*

## Chương 5: Kết luận

- Từ kết quả thực nghiệm chúng ta có thể kết luận rằng:
  - + Điều chỉnh siêu tham số làm tốt về giảm tài nguyên với việc dừng sớm tại fm\_v1 và tăng độ chính xác với fruit\_fr.
  - + Tăng cường dữ liệu tuy là những phương pháp có thể tăng cường hiệu suất nhưng có nhược điểm là tốn thời gian để có thể tìm ra các giá trị phù hợp, minh chứng rằng tăng cường dữ liệu tăng độ chính xác tại fm\_v2 nhưng lại giảm tại fruit\_v3 do dữ liệu phần lớn là sử dụng các kỹ thuật tăng cường dữ liệu.
  - + Thay đổi kiến trúc YOLOv8 tuy phụ thuộc vào sự nhất quán của dữ liệu nhưng tốc độ và việc giảm tài nguyên để huấn luyện do giảm số lượng param rất đáng để đánh đổi với độ chính xác.
  - + Chuyển đổi mô hình sang TensorRT mang lại tốc độ vượt trội so với thông thường nhưng phải đánh đổi với độ chính xác tuy nhiên giảm rất ít, có thể cân nhắc sử dụng.

## TÀI LIỆU THAM KHẢO

[1] Jeremy Jordan, Setting the learning rate of your neural network. (1/3/2018).

Ngày truy cập : 5/8/2024. [Hình ảnh trực tuyến]

<https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png>

[2] Jeremy Jordan, Learning rate finder - Setting the learning rate of your neural network. (1/3/2018). Ngày truy cập: 5/8/2024. [Hình ảnh trực tuyến]

[https://www.jeremyjordan.me/content/images/2018/02/lr\\_finder.png](https://www.jeremyjordan.me/content/images/2018/02/lr_finder.png)

[3] Cathrine Jeeva, SGD with momentum - Optimizers in Deep Learning.

(11/1/2024). Ngày truy cập: 5/8/2024. [Hình ảnh trực tuyến]

<https://www.scaler.com/topics/images/sgd-with-momentum.webp>

[4] Zhangxi Ye, Spectral augmentation, a HSV boxplot b spectral transformation - Extraction of Olive Crown Based on UAV Visible Images and the U2-Net Deep Learning Model. (21/3/2022). Ngày truy cập: 5/8/2024. [Hình ảnh trực tuyến]

<https://www.researchgate.net/publication/359401395/figure/fig5/AS:11431281212511218@1702672691169/Spectral-augmentation-a-HSV-boxplot-b-spectral-transformation.tif>

[5] Brac University, Fruit Freshness Detection Dataset. (9/2022). Ngày truy cập:

5/8/2024. [Tập dữ liệu] <https://universe.roboflow.com/brac-university-v9w2y/fruit-freshness-detection-08shj>

[6] Glenn Jocher - CEO của Ultralytics, Ultralytics, (2014). Ngày truy cập:

5/8/2024. [Thư viện] <https://github.com/ultralytics/ultralytics>