

AI VIET NAM – AI COURSE 2024

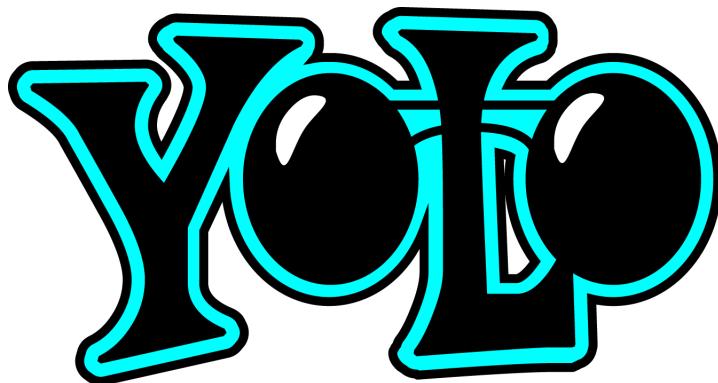
Tutorial: Phát hiện đối tượng trong ảnh với YOLOv10

Dinh-Thang Duong, Nguyen-Thuan Duong, Minh-Duc Bui và
Quang-Vinh Dinh

Ngày 30 tháng 5 năm 2024

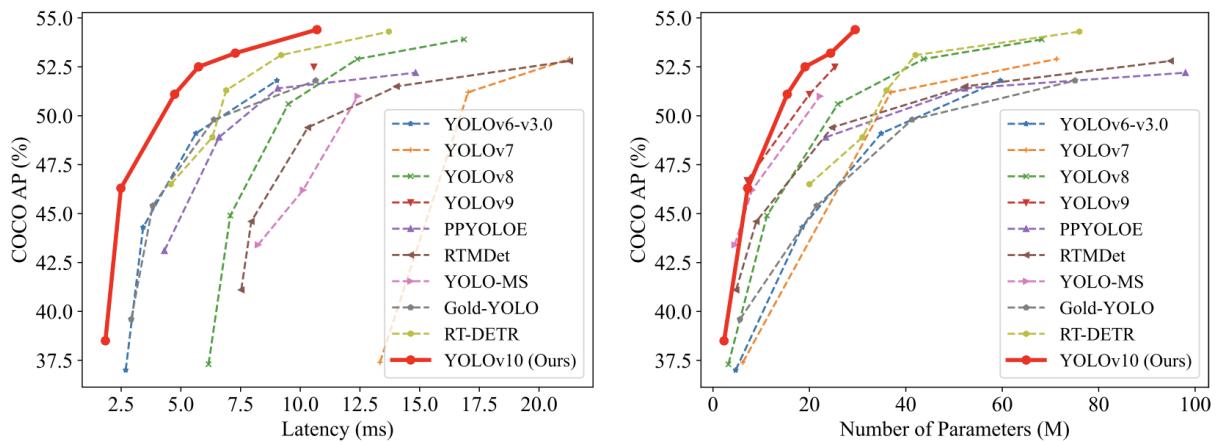
I. Giới thiệu

Object Detection (Tạm dịch: Phát hiện đối tượng) là một bài toán cổ điển thuộc lĩnh vực Computer Vision. Mục tiêu của bài toán này là tự động xác định vị trí của các đối tượng trong một tấm ảnh. Tính tới thời điểm hiện tại, đã có rất nhiều phương pháp được phát triển nhằm giải quyết hiệu quả bài toán này. Trong đó, các phương pháp thuộc họ YOLO (You Only Look Once) thu hút được sự chú ý rất lớn từ cộng đồng nghiên cứu bởi độ chính xác và tốc độ thực thi mà loại mô hình này mang lại.



Hình 1: Logo của mô hình YOLO. Ảnh: [link](#).

Thời gian vừa qua, Ao Wang và các cộng sự tại Đại học Thanh Hoa (Tsinghua University) đã đề xuất mô hình **YOLOv10** trong bài báo **YOLOv10: Real-Time End-to-End Object Detection** [10]. Với những cải tiến mới, mô hình đã đạt được hiệu suất vượt trội hơn so với các phiên bản YOLO trước đó ở các khía cạnh khác nhau, tăng cường khả năng phát hiện đối tượng theo thời gian thực (real-time object detection).



Hình 2: Hiệu suất của YOLOv10 khi so sánh với các mô hình khác. Trên tập dữ liệu COCO, YOLOv10 đạt được kết quả tốt nhất về khía cạnh Độ trễ (Latency) và Số lượng tham số mô hình (Number of parameters) trong khi vẫn giữ được độ chính xác (COCO AP) cao. Ảnh: [10].

Trong bài viết này, chúng ta sẽ cùng nhau tìm hiểu về YOLOv10 và cách sử dụng mô hình này. Thông qua đó, nhóm cũng sẽ trình bày sơ lược về bài toán Object Detection cũng như tóm tắt ngắn gọn các phiên bản YOLO trước đó để bạn đọc có một cái nhìn tổng quan hơn về nội dung này.

Theo đó, bài viết được bô cục như sau:

- **Phần I:** Giới thiệu về nội dung bài viết.
- **Phần II:** Tóm tắt về bài toán Object Detection và các phiên bản YOLO đời trước.
- **Phần III:** Trình bày nội dung YOLOv10.
- **Phần IV:** Hướng dẫn cách cài đặt, huấn luyện và sử dụng YOLOv10.
- **Phần V:** Trích dẫn tài liệu.

II. Bài toán Object Detection và các phiên bản YOLO đời trước

II.I. Bài toán Object Detection

Trong Computer Vision, bài toán Object Detection hướng đến xây dựng một chương trình có thể tự động xác định vị trí và nhận diện tên (class) của các vật thể trong một bức ảnh. Tổng hợp hai thông tin đầu ra này còn được gọi với tên là bounding box. Từ đây, ta có thể mô tả Input/Output của một chương trình Object Detection như sau:

- **Input:** Một bức ảnh.
- **Output:** Bounding box của các vật thể cần phát hiện trong ảnh.



Hình 3: Minh họa Input/Output của bài toán Object Detection.

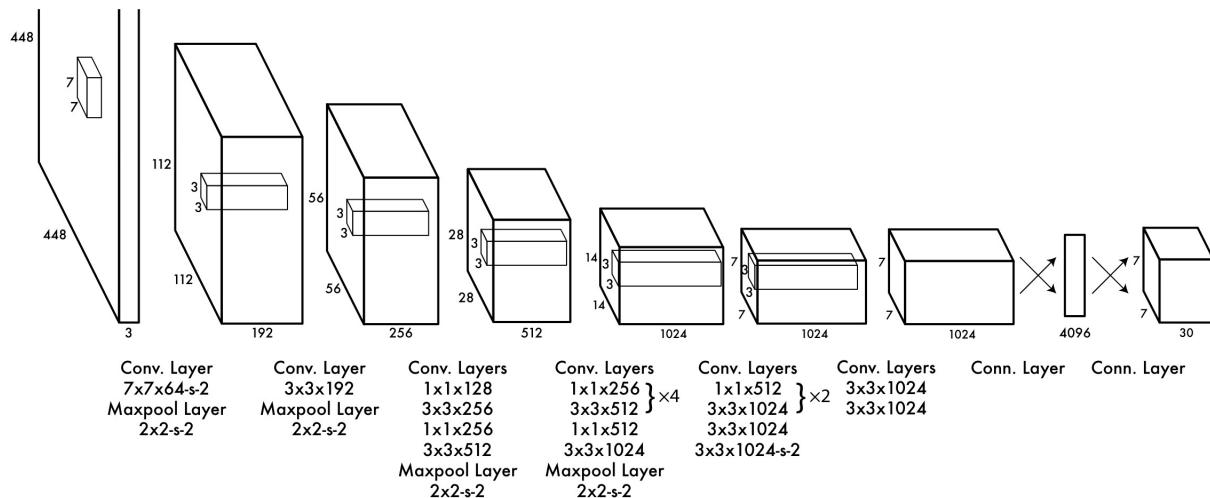
Đến thời điểm hiện tại, các phương pháp sử dụng mạng Deep Learning cho thấy hiệu suất vượt trội. Ta có thể tóm tắt các hướng tiếp cận theo ba dạng như sau:

1. **One-stage Object Detection:** Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện trên một bước duy nhất. Diễn hình cho hướng tiếp cận này có thể kể đến SSD [13] và YOLO [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
2. **Two-stage Object Detection:** Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện riêng biệt. Diễn hình cho hướng tiếp cận này có thể kể đến RCNN [14] và Faster RCNN [15].
3. **End-to-end Object Detection:** Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được dự đoán bởi một mô hình duy nhất (không sử dụng các bước tiền và hậu xử lý bounding box). Diễn hình cho hướng tiếp cận này có thể kể đến DETR [16], DINO [17], và DeFCN [18].

Ở phần sau, chúng ta sẽ tập trung điểm qua các phiên bản YOLO (từ v1 đến v9).

II.II. YOLOv1

YOLOv1 [1] là mô hình one-stage (hoặc single-stage) real-time object detection được giới thiệu vào năm 2016.

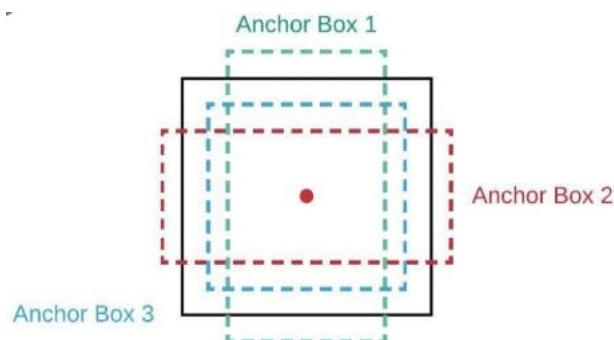


Hình 4: Kiến trúc mô hình YOLOv1 với 24 lớp conv và 2 lớp mlp. Ảnh: [1].

- **Điểm mới:** YOLOv1 sử dụng một mạng neural đơn để dự đoán cả vị trí và tên class của các object trực tiếp từ ảnh đầu vào.
- **Ưu điểm:** Tốc độ nhanh, khả năng object detection theo thời gian thực.
- **Nhược điểm:** Độ chính xác không cao với các object nhỏ hoặc bị che khuất.

II.III. YOLOv2

YOLOv2 [2], còn được gọi là YOLO9000, được giới thiệu vào năm 2017 với nhiều cải tiến so với YOLOv1.

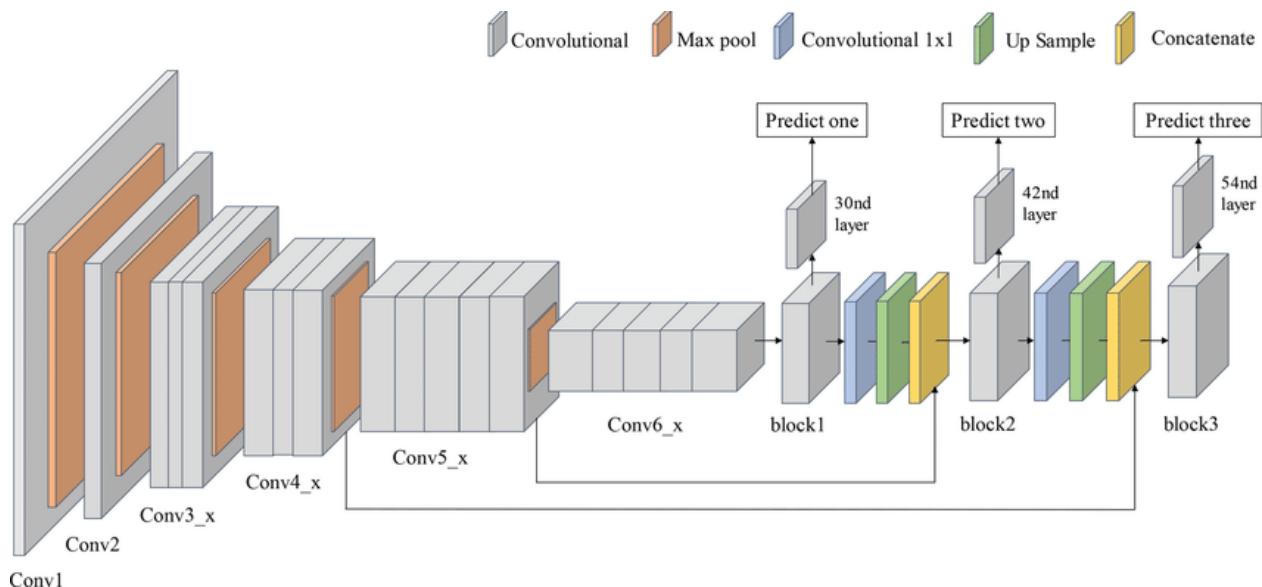


Hình 5: Hình ảnh minh họa về anchor boxes. Ảnh: [Zixuan Zhang](#).

- **Điểm mới:** Sử dụng anchor boxes, mạng Darknet-19, và tăng training data để tăng độ chính xác.
- **Ưu điểm:** Tăng độ chính xác và khả năng nhận diện nhiều object trong 1 cell.
- **Nhược điểm:** Phức tạp hơn, cần nhiều tài nguyên tính toán, và khó detect các object nhỏ.

II.IV. YOLOv3

YOLOv3 [3] ra mắt năm 2018, tiếp tục cải tiến từ YOLOv2.



Hình 6: Kiến trúc mô hình YOLOv3. Ảnh: [3].

- **Điểm mới:** Sử dụng mạng Darknet-53 và detect object ở ba cấp độ khác nhau (multi-scale detection) để cải thiện độ chính xác.
- **Ưu điểm:** Độ chính xác cao hơn, khả năng phát hiện object nhỏ tốt hơn.
- **Nhược điểm:** Tốc độ chậm hơn so với các phiên bản trước do sự phức tạp của mô hình.

II.V. YOLOv4

YOLOv4 [4] ra mắt năm 2020, với mục tiêu cải thiện cả độ chính xác và tốc độ.

- **Điểm mới:** Sử dụng nhiều kỹ thuật mới như CSPDarknet53, PANet, và nhiều cải tiến khác.
- **Ưu điểm:** Cân bằng tốt giữa tốc độ và độ chính xác, dễ dàng sử dụng và triển khai.
- **Nhược điểm:** Yêu cầu phần cứng mạnh để đạt hiệu năng tối ưu.

II.VI. YOLOv5

YOLOv5 [5], không phải do tác giả gốc phát triển, nhưng được cộng đồng sử dụng rộng rãi từ năm 2020.

- **Điểm mới:** Tập trung vào tối ưu hóa và dễ dàng sử dụng với các framework như PyTorch. Sử dụng CSPNet làm backbone và PANet để fusion giúp cải thiện độ chính xác của mô hình.
- **Ưu điểm:** Dễ dàng triển khai, tối ưu hóa tốt, cộng đồng hỗ trợ mạnh mẽ.
- **Nhược điểm:** Yêu cầu tài nguyên tính toán cao và khó detect được các object nhỏ.

II.VII. YOLOv6

YOLOv6 [6] là phiên bản tiếp theo với nhiều cải tiến về tốc độ và độ chính xác.

- **Điểm mới:** Sử dụng backbone mới EfficientRep và Rep-PAN Neck để tối ưu hóa và tăng hiệu năng của mô hình. SimOTA, một phương pháp Label Assignment, cũng được sử dụng để tăng tính ổn định khi training.
- **Ưu điểm:** Hiệu năng cao hơn, tốc độ nhanh hơn.
- **Nhược điểm:** Yêu cầu tài nguyên tính toán cao hơn.

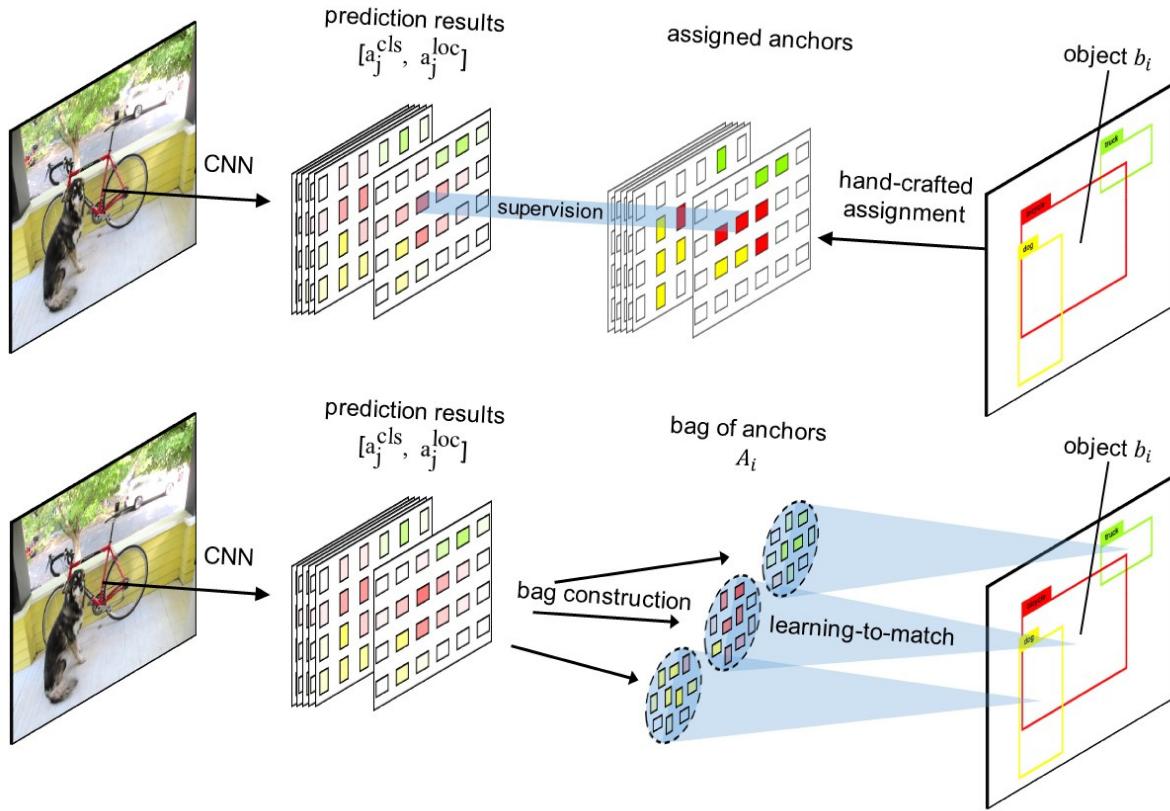
II.VIII. YOLOv7

YOLOv7 [7] tiếp tục phát triển với các cải tiến về mô hình và thuật toán.

- **Điểm mới:** Sử dụng backbone E-ELAN kết hợp với phương pháp trainable bag-of-freebies để tăng độ chính xác của mô hình mà không làm tăng chi phí tính toán.
- **Ưu điểm:** Tăng độ chính xác và khả năng nhận diện trong các điều kiện phức tạp.
- **Nhược điểm:** Mức độ phức tạp cao, cần nhiều thời gian và tài nguyên để huấn luyện.

II.IX. YOLOv8

YOLOv8 [8] được giới thiệu vào năm 2023 bởi [Ultralytics](#). Mô hình này cải thiện độ chính xác và tốc độ so với YOLOv7 và giới thiệu nhiều tính năng mới như anchor-free detection.

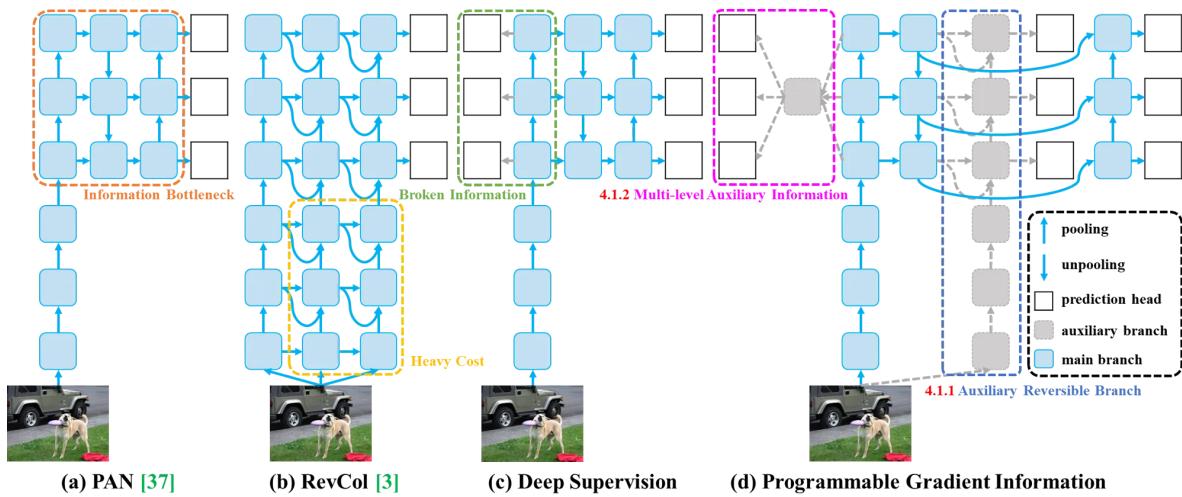


Hình 7: So sánh 2 phương pháp hand-crafted anchor (trên) và anchor-free (dưới). Ảnh: [11].

- **Điểm mới:** Sử dụng anchor-free detection, giúp đơn giản hóa kiến trúc mô hình và cải thiện hiệu suất.
- **Ưu điểm:**
 - + Độ chính xác cao hơn: YOLOv8 đạt mAP 50.2% trên bộ dữ liệu COCO, cao hơn so với YOLOv7.
 - + Dễ sử dụng: YOLOv8 có giao diện Python và CLI dễ sử dụng, giúp người dùng dễ dàng triển khai và huấn luyện mô hình.
- **Nhược điểm:** Yêu cầu tài nguyên tính toán cao: Mặc dù có nhiều cải tiến, YOLOv8 vẫn yêu cầu nhiều tài nguyên tính toán, đặc biệt là khi xử lý hình ảnh độ phân giải cao.

II.X. YOLOv9

YOLOv9 [9] được giới thiệu vào năm 2024 bởi Chien-Yao Wang, I-Hau Yeh, và Hong-Yuan Mark Liao. Mô hình này cải thiện độ chính xác và tốc độ so với YOLOv8 và giới thiệu nhiều kỹ thuật mới như Programmable Gradient Information (PGI) và Generalized Efficient Layer Aggregation Network (GELAN).



Hình 8: PGI và các kiến trúc tương tự. Ảnh: [9].

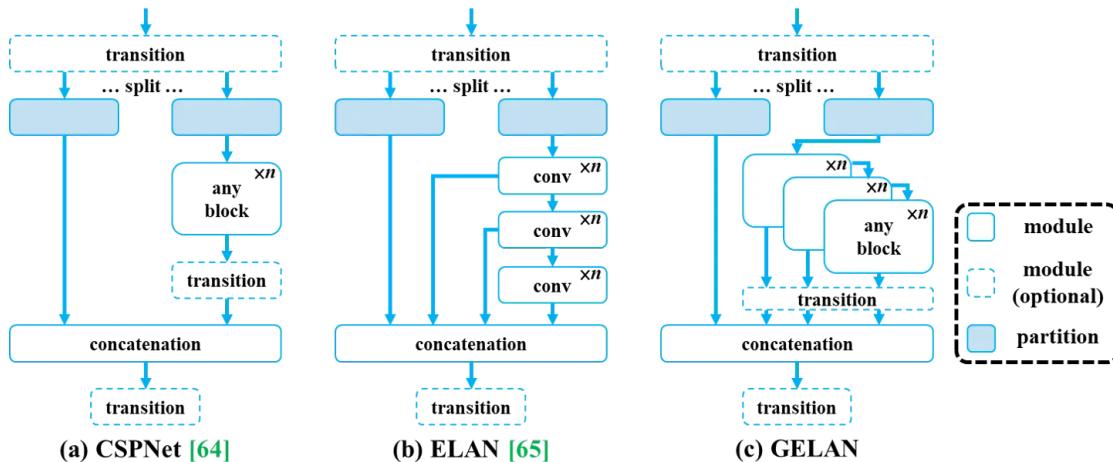
- **Điểm mới:** YOLOv9 sử dụng PGI và GELAN để cải thiện độ chính xác và hiệu suất của mô hình.

- **Ưu điểm:**

- + Kiến trúc tiên tiến: Sử dụng PGI và GELAN giúp mô hình duy trì thông tin quan trọng và tối ưu hóa quá trình huấn luyện, làm cho YOLOv9 trở nên mạnh mẽ và linh hoạt hơn trong nhiều ứng dụng khác nhau.
- + Tốc độ nhanh hơn: YOLOv9 có thể xử lý hình ảnh nhanh hơn so với YOLOv8 nhờ vào các cải tiến trong kiến trúc mạng.

- **Nhược điểm:**

- + Mặc dù nhanh hơn YOLOv8, YOLOv9 vẫn yêu cầu nhiều tài nguyên tính toán, đặc biệt là khi xử lý hình ảnh độ phân giải cao.
- + Mặc dù cải thiện so với YOLOv8, YOLOv9 vẫn gặp khó khăn trong việc phát hiện các object rất nhỏ.



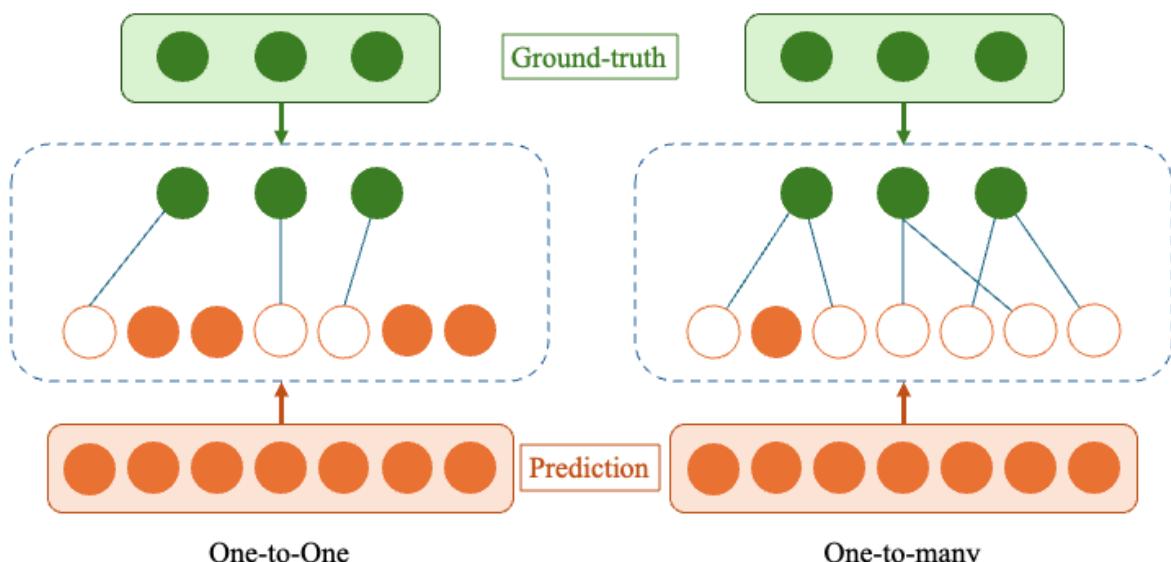
Hình 9: GELAN và các kiến trúc tương tự. Ảnh: [9].

III. YOLOv10: Real-Time End-to-End Object Detection

Ao Wang và các cộng sự đã đặt nghi vấn về sự tối ưu trong việc phụ thuộc vào kỹ thuật hậu xử lý Non-maximum Suppresion (NMS) và cách thiết kế mô hình của các phiên bản YOLO trước đó. Với các hạn chế quan sát được từ hai điều trên và mục tiêu xây dựng một mô hình object detection thời gian thực, YOLOv10 đã được đề xuất với những thay đổi mới. Theo đó, có hai điểm nhấn chính trong phương pháp mà nhóm tác giả YOLOv10 đề xuất bao gồm:

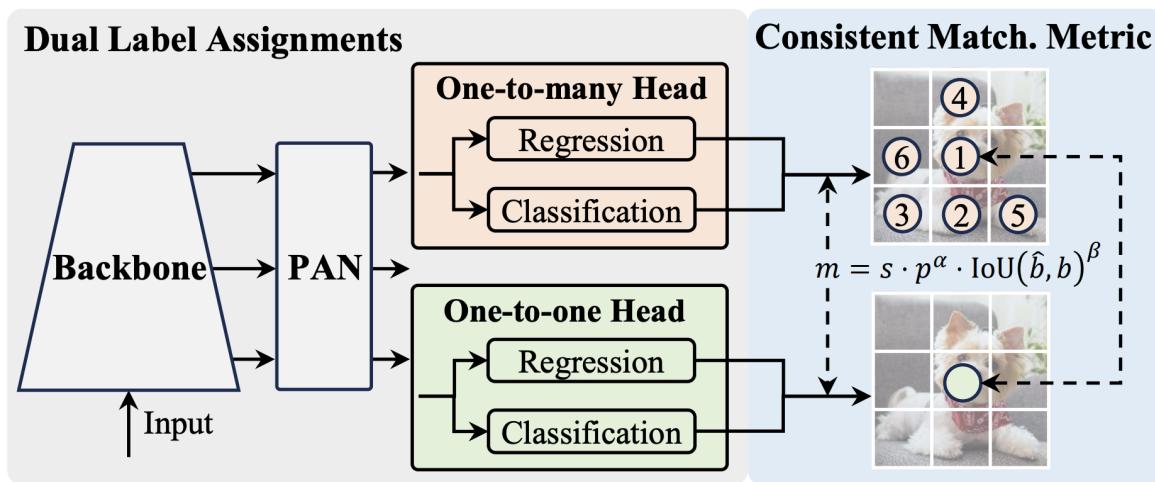
1. **Consistent Dual Assignments for NMS-free Training:** Trong quá trình dự đoán của các mạng YOLO đời trước, rất nhiều bounding box được mô hình đưa ra (ví dụ: anchors box,...) và nhiệm vụ của chúng ta là tìm ra đại diện chính xác nhất cho mỗi vật thể có trong ảnh. Để tận dụng tối đa các đề xuất bounding box đúng trong việc huấn luyện, các phương pháp thường ứng dụng kỹ thuật Task Alignment Learning (TAL). Trong đó, chiến lược one-to-many label assignment được áp dụng để gán các bounding box “positive” (bounding box chính xác) vào ground-truth của vật thể tương ứng để tăng cường khả năng nhận biết vật thể của mô hình. Tuy vậy, việc này lại gây ra độ trễ (latency) lớn trong quá trình inference của mô hình bởi việc phụ thuộc vào thuật toán NMS để lọc các dự đoán thừa.

Một cách tiếp cận khác đó là sử dụng chiến lược one-to-one label assignment, bằng cách chỉ gán một đề xuất bounding box “positive” với ground-truth của vật thể tương ứng, qua đó tránh việc hậu xử lý với NMS. Tuy vậy, chiến lược này lại dẫn đến hiệu suất mô hình không được tốt.



Hình 10: Minh họa chiến lược one-to-one và one-to-many label assignments.

Để khắc phục trình trạng của hai cách nêu trên, YOLOv10 cài đặt một chiến lược huấn luyện mới là sự kết hợp của one-to-one và one-to-many, mang tên Dual label assignments. Chiến lược này được minh họa theo như hình sau:

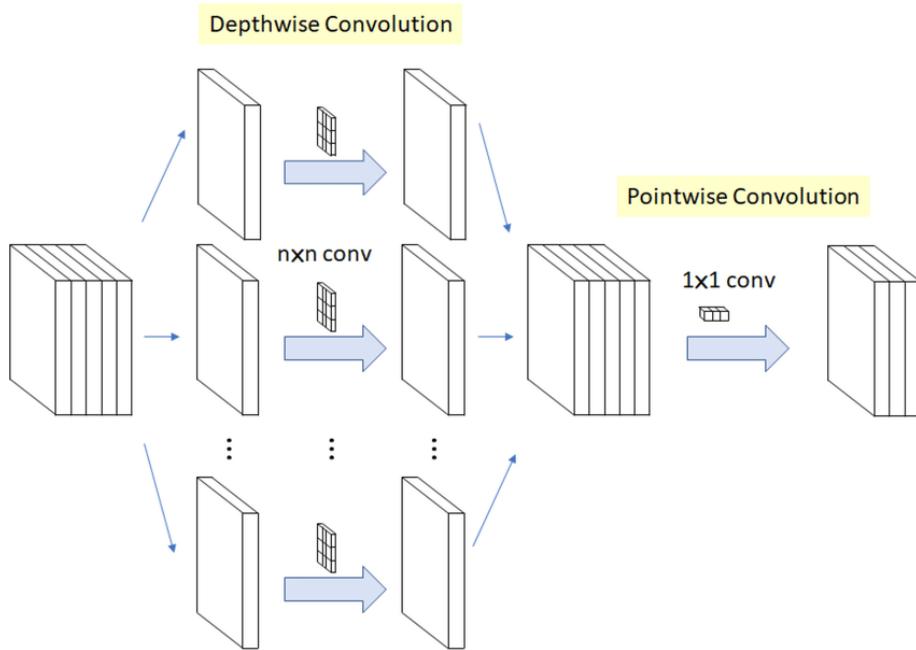


Hình 11: Minh họa chiến lược Dual label assignments. Ảnh: [10].

Về cơ bản, trong quá trình huấn luyện, tác giả sử dụng thông tin của cả hai chiến lược. Đến quá trình inference, nhánh one-to-many sẽ được bỏ đi để tránh việc sử dụng NMS. Về cách bắt cặp ground-truth và bounding box dự đoán, cả hai chiến lược đều sử dụng chung một độ đo là Consistent Matching Metric.

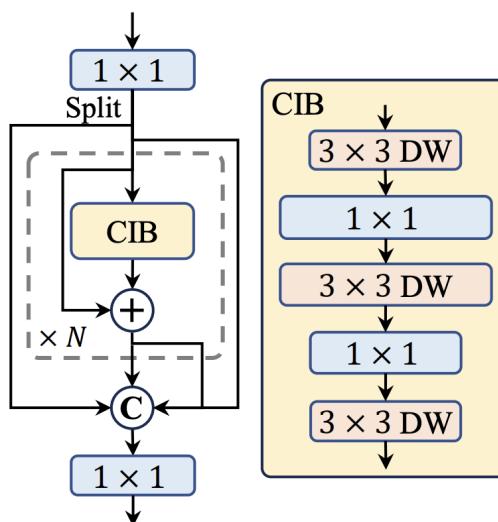
2. **Holistic Efficiency-Accuracy Driven Model Design:** Bên cạnh kỹ thuật huấn luyện, việc thiết kế kiến trúc mô hình cho YOLO vẫn còn đó những thách thức và hạn chế để khắc phục theo tiêu chí độ hiệu quả (efficiency) và độ chính xác (accuracy). Về độ hiệu quả, dựa trên kiến trúc YOLO của bản trước (YOLOv8), nhóm tác giả thực hiện hiệu chỉnh các nội dung sau:

- **Lightweight classification head:** Nhóm tác giả thực hiện giảm bớt độ lớn về kích thước của nhánh Classification khi nhận thấy với cùng một kiến trúc, nhánh Regression cho thấy mức độ ảnh hưởng lớn đến hiệu suất của YOLO hơn. Vì vậy, tác giả sử dụng hai layer depth-wise convolution với kernel 3x3 đi kèm với point-wise convolution với kernel 1x1. Điều này sẽ làm giảm đáng kể số lượng tham số của kiến trúc mô hình cũng như thời gian xử lý.
- **Spatial-channel decoupled downsampling:** Các mạng YOLO thường sử dụng layer convolution với kernel 3x3 và stride=2 để giảm kích thước feature map xuống. Điều này được nhóm tác giả quan sát cho thấy chi phí tính toán còn lớn. Vì vậy, tương tự như với classification head, YOLOv10 cũng sử dụng kết hợp phép point-wise và depth-wise convolution để thay thế phương thức thông thường.



Hình 12: Minh họa về phép depth-wise convolution và point-wise convolution để thay thế phép convolution thông thường. Ảnh: [link](#).

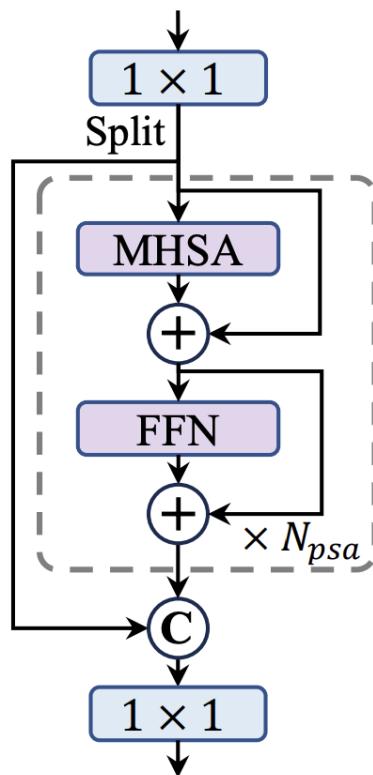
- **Rank-guided block design:** Quan sát YOLOv8, tác giả nhận thấy toàn bộ các stage trong kiến trúc đều sử dụng chung một building block. Tuy nhiên, thông qua việc tính intrinsic rank của mỗi stage, tác giả chỉ ra rằng ở các stage gần cuối có sự dư thừa (redundancy) về mặt tham số lớn, dẫn đến sự không tối ưu về chi phí tính toán và lưu trữ. Vì vậy, để khắc phục, nhóm tác giả áp dụng chiến lược: duyệt qua các stage trong một mô hình theo thứ tự tăng dần về intrinsic rank, thực hiện thay thế basic block bằng một block được đề xuất là Compact Inverted Block (CIB). Các bạn có thể quan sát thành phần của block này ở ảnh dưới đây:



Hình 13: Minh họa cấu trúc của Compact Inverted Block (CIB). Ảnh: [10].

Về độ chính xác, dựa trên ý tưởng liên quan đến receptive field và phép self-attention, nhóm tác giả thực hiện hiệu chỉnh các nội dung sau:

- **Large-kernel convolution:** Ở các phiên bản YOLOv10 kích thước nhỏ, nhóm tác giả thực hiện tăng kích thước kernel từ 3x3 lên 7x7 của phép depth-wise convolution trong CIB nhằm cải thiện receptive field. Việc tăng này chỉ được áp dụng ở các stage cuối.
- **Partial self-attention (PSA):** Để tận dụng sức mạnh của phép self-attention. Kể từ sau stage 4 của mô hình, nhóm tác giả tách features sau phép point-wise convolution của CIB làm hai phần. Một phần sẽ được đẩy vào N_{PSA} block bao gồm sự kết hợp của lớp Multi-head self-attention (MHSA) và Feed-forward network (FFN), khá giống với Transformer Encoder. Phần LayerNorm sẽ được thay thế bằng BatchNorm để tăng tốc độ xử lý. Sau đó, kết quả của bước này sẽ được kết hợp với phần tách còn lại bởi phép point-wise convolution.



Hình 14: Minh họa Partial self-attention (PSA). Ảnh: [10].

IV. Cài đặt chương trình và đánh giá

Trong phần này, nhóm sẽ trình bày cách cài đặt, sử dụng và huấn luyện YOLOv10 trên bộ dữ liệu mới. Đồng thời, nhóm cũng thực hiện một thực nghiệm nhỏ nhằm so sánh hiệu suất của YOLOv10 so với hai phiên bản gần nhất là YOLOv8 và YOLOv9. Môi trường lập trình nhóm sử dụng là Google Colab.

IV.I. Cài đặt và sử dụng pre-trained model

Một cách nhanh chóng để sử dụng được YOLOv10 đó là sử dụng pre-trained model (mô hình đã được huấn luyện sẵn trên bộ dữ liệu COCO - một bộ dữ liệu rất lớn). Để sử dụng pre-trained model, các bạn làm như sau:

1. **Cài đặt các thư viện cần thiết:** Tải về mã nguồn của YOLOv10 và cài đặt các thư viện trong file requirements.txt bằng các đoạn code sau:

```
1 !git clone https://github.com/THU-MIG/yolov10.git
2 %cd yolov10
3 !pip install -q -r requirements.txt
4 !pip install -e .
```

2. **Tải trọng số của pre-trained models:** Để sử dụng được pre-trained models, chúng ta cần tải về file trọng số (file .pt). Các bạn chạy đoạn code sau để tải về file trọng số phiên bản YOLOv10n:

```
1 !wget https://github.com/THU-MIG/yolov10/releases/download/v1.1/
      yolov10n.pt
```

3. **Khởi tạo mô hình:** Để khởi tạo mô hình với trọng số vừa tải về, các bạn chạy đoạn code sau:

```
1 from ultralytics import YOLOv10
2
3 model = YOLOv10("yolov10n.pt")
```

4. **Tải ảnh cần dự đoán:** Chúng ta sẽ test mô hình trên một ảnh bất kỳ. Các bạn có thể tự chọn ảnh của riêng mình hoặc sử dụng ảnh tại [đây](#). Các bạn có thể chạy đoạn code sau để tải ảnh này vào colab tự động:

```
1 !gdown "1tr9PSRRd1C2pNir7jsYugpSMG-7v32VJ" -O "./images/"
```

5. **Dự đoán:** Để chạy dự đoán cho ảnh đã tải về, các bạn truyền đường dẫn ảnh vào mô hình như đoạn code sau:

```
1 image_path = "./images/HCMC_Street.jpg"
2 result = model(source=image_path)[0]
```



Hình 15: Ảnh cần dự đoán.

6. **Lưu kết quả dự đoán:** Để lưu lại ảnh đã được dự đoán, các bạn chạy đoạn code sau:

```
1 result.save("./images/HCMC_Street_predict.png")
```



Hình 16: Kết quả dự đoán của mô hình YOLOv10 phiên bản **nano** (yolov10n.pt).

7. **Dự đoán youtube video:** Để dự đoán với input là youtube video, các bạn chỉ cần thay thế **image_path** bằng đường dẫn youtube video như đoạn code sau:

```

1 youtube_video_path = "https://youtu.be/wqPSsu7XQ74"
2 video_result = model(source=youtube_video_path)

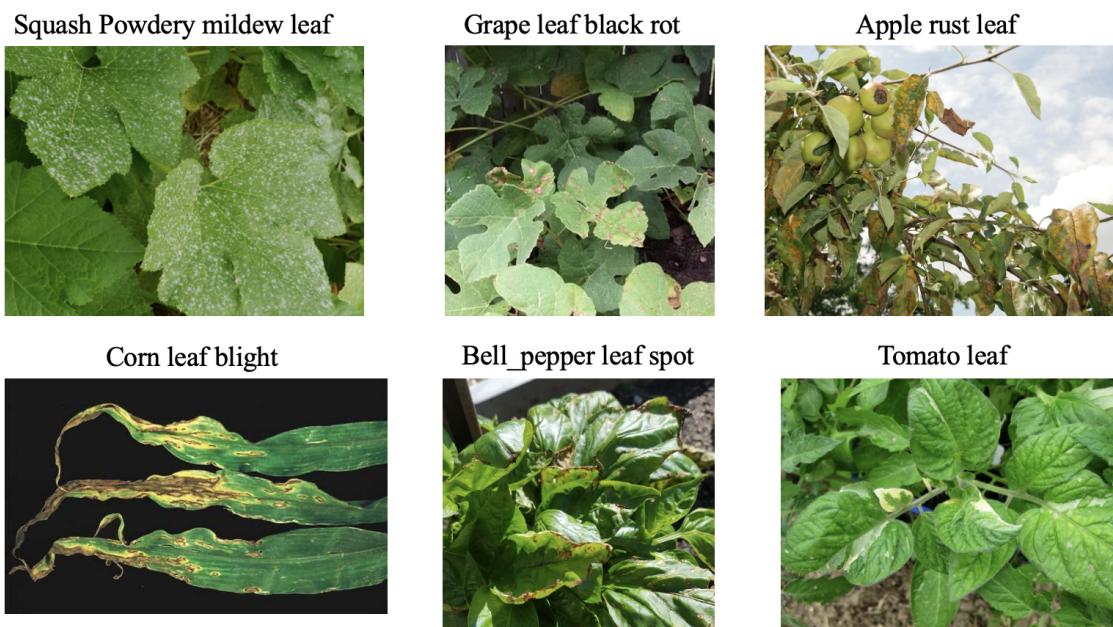
```

Kết quả dự đoán sẽ là một video được lưu dưới dạng .avi trong thư mục: /content/yolov10/runs/detect/predict

IV.II. Huấn luyện YOLOv10 trên tập dữ liệu mới

Trong phần này, chúng ta sẽ thực hiện huấn luyện mô hình YOLOv10 (fine-tuning) trên một bộ dữ liệu với các class mới. Để tránh sự nhầm lẫn, phần này sẽ được thực hiện ở một file colab khác so với phần trước. Các bước thực hiện như sau:

1. **Tải bộ dữ liệu:** Chúng ta sẽ giải quyết bài toán phát hiện các loại lá được phân biệt theo trình trạng bệnh của chúng. Bộ dữ liệu được sử dụng trong bài toán này là PlantDoc [12]. Để dễ hình dung, các bạn có thể quan sát ảnh minh họa sau:



Hình 17: Một vài mẫu dữ liệu trong bộ dữ liệu về bệnh của lá.

Để tải bộ dữ liệu trên, các bạn hãy chạy đoạn code sau:

```

1 !gdown "1LBpKKXFcfvUVgyk3tgQH6YxNp1KXX0Va"

```

Giải nén bộ dữ liệu vào folder **datasets** và xóa file nén không còn dùng đến. Các bạn thực thi đoạn code sau:

```

1 !mkdir datasets
2 !unzip -q "/content/PlantDocv4.zip" -d "/content/datasets/PlantDocv4"
3 !rm /content/PlantDocv4.zip

```

Quan sát thư mục giải nén, có thể thấy bộ dữ liệu này đã được gán nhãn và đưa vào format cấu trúc dữ liệu training theo yêu cầu của YOLO. Vì vậy, chúng ta sẽ không cần thực hiện bước chuẩn bị dữ liệu ở bài này.

2. **Cài đặt và import các thư viện cần thiết:** Tương tự như phần trước, các bạn chạy các đoạn code sau để cài đặt các gói thư viện để sử dụng được YOLOv10:

```

1 !git clone https://github.com/THU-MIG/yolov10.git
2 %cd yolov10
3 !pip install -q -r requirements.txt
4 !pip install -e .

```

3. **Khởi tạo mô hình YOLOv10:** Chúng ta sẽ khởi tạo mô hình YOLOv10 với phiên bản **nano (n)** từ trọng số đã được huấn luyện trên bộ dữ liệu COCO. Để tải trọng số yolov10n.pt, các bạn chạy đoạn code sau:

```

1 !wget https://github.com/THU-MIG/yolov10/releases/download/v1.1/
      yolov10n.pt

```

Sau đó, để khởi tạo mô hình từ trọng số đã tải về, các bạn chạy đoạn code sau:

```

1 from ultralytics import YOLOv10
2
3 model = YOLOv10("yolov10n.pt")

```

4. **Huấn luyện mô hình:** Chúng ta tiến hành huấn luyện YOLOv10 trên bộ dữ liệu PlantDoc với 100 epochs và kích thước ảnh là 640. Các bạn chạy đoạn code sau:

```

1 model.train(data="..../datasets/PlantDocv4/data.yaml",
2               epochs=100,
3               imgsz=640)

```

Epoch	GPU_mem	box_om	cls_om	dfl_om	box_oo	cls_oo	dfl_oo	Instances	Size
1/5	3.53G	1.259	4.349	1.438	1.154	5.433	1.345	65	640: 100% [██████████] 126/126 [01:30<00:00, 1.84it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% [██████████]	10/10 [00:05<00:00,	
Epoch	GPU_mem	box_om	cls_om	dfl_om	box_oo	cls_oo	dfl_oo	Instances	Size
2/5	3.49G	1.225	3.863	1.368	1.121	4.863	1.258	33	640: 100% [██████████] 126/126 [01:27<00:00, 1.83it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% [██████████]	10/10 [00:05<00:00,	
Epoch	GPU_mem	box_om	cls_om	dfl_om	box_oo	cls_oo	dfl_oo	Instances	Size
3/5	3.56G	1.29	3.444	1.429	1.16	4.464	1.28	56	640: 100% [██████████] 126/126 [01:25<00:00, 1.56it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% [██████████]	10/10 [00:06<00:00,	
Epoch	GPU_mem	box_om	cls_om	dfl_om	box_oo	cls_oo	dfl_oo	Instances	Size
4/5	3.54G	1.283	3.128	1.425	1.159	4.182	1.271	34	640: 100% [██████████] 126/126 [01:21<00:00, 1.11it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% [██████████]	10/10 [00:09<00:00,	
Epoch	GPU_mem	box_om	cls_om	dfl_om	box_oo	cls_oo	dfl_oo	Instances	Size
5/5	3.31G	1.278	2.9	1.425	1.149	3.973	1.27	48	640: 100% [██████████] 126/126 [01:21<00:00, 2.02it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100% [██████████]	10/10 [00:04<00:00,	
	all	314	1191	0.341	0.31	0.234	0.171		

Hình 18: Quá trình huấn luyện mô hình YOLOv10 trên tập dữ liệu PlantDoc.

5. **Đánh giá mô hình:** Để thực hiện đánh giá mô hình trên tập test, các bạn chạy đoạn code sau:

```

1 model = YOLOv10("./runs/detect/train/weights/best.pt")
2
3 model.val(data='../datasets/PlantDocv4/data.yaml',
4             imgsz=640,
5             split="test")

```

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95): 100%
all	246	484	0.597	0.653	0.659	0.498
Apple Scab leaf	246	13	0.581	0.749	0.745	0.57
Apple leaf	246	10	0.356	0.9	0.724	0.487
Apple rust leaf	246	12	0.738	0.667	0.761	0.551
Bell_pepper leaf spot	246	11	0.444	0.909	0.552	0.369
Bell_pepper leaf	246	15	0.606	0.31	0.458	0.38
Blueberry leaf	246	42	0.691	0.524	0.594	0.367
Cherry leaf	246	19	0.856	0.313	0.569	0.411
Corn Gray leaf spot	246	4	0.228	0.5	0.58	0.462
Corn leaf blight	246	12	0.785	0.913	0.848	0.762
Corn rust leaf	246	11	0.877	1	0.995	0.831
Peach leaf	246	10	0.757	0.626	0.682	0.511
Potato leaf late blight	246	17	0.326	0.412	0.381	0.286
Potato leaf	246	17	0.367	0.529	0.514	0.406
Raspberry leaf	246	17	0.708	0.824	0.885	0.777
Soyabean leaf	246	20	0.791	0.6	0.699	0.624
Squash Powdery mildew leaf	246	9	0.644	0.889	0.857	0.726
Strawberry leaf	246	30	0.826	1	0.977	0.817
Tomato Early blight leaf	246	19	0.485	0.421	0.372	0.215
Tomato Septoria leaf spot	246	24	0.473	0.625	0.575	0.368
Tomato leaf bacterial spot	246	27	0.376	0.667	0.428	0.255
Tomato leaf late blight	246	14	0.147	0.429	0.366	0.252
Tomato leaf mosaic virus	246	14	0.395	0.714	0.622	0.471
Tomato leaf yellow virus	246	36	0.652	0.194	0.526	0.34
Tomato leaf	246	42	0.618	0.619	0.651	0.3
Tomato mold leaf	246	16	0.476	0.5	0.539	0.35
grape leaf black rot	246	15	0.924	0.813	0.895	0.704
grape leaf	246	8	1	0.995	0.995	0.85

Hình 19: Đánh giá mô hình sau khi huấn luyện trên tập test.

IV.III. Đánh giá

Nhóm thực hiện đánh giá mô hình YOLO qua các phiên bản v8, v9 và v10. Bằng cách lựa chọn cả 3 phiên bản có cùng số lượng tham số khoảng 25M tương ứng là YOLOv8-M, YOLOv9-C và YOLOv10-L. Các thử nghiệm được thực hiện trên cùng một thiết bị, python version, random seed và một số hyperparameter như: batch_size=16, image_size=640,... Sau khi quá trình huấn luyện kết thúc, thực hiện đánh giá trên tập test và ghi lại kết quả vào bảng 1 dưới đây:

Bảng 1: Bảng thực nghiệm kết quả trên tập test của các mô hình YOLO phiên bản YOLOv8-M, YOLOv9-C, YOLOv10-M, YOLOv10-L sau khi fine-tuning.

Model	params	GFLOPs	layers	inference time	mAP@50	mAP@50-95
YOLOv8-M	25.9M	78.9	295	6.5 ms	0.614	0.476
YOLOv9-C	25.3M	102.1	618	8.1 ms	0.653	0.503
YOLOv10-M	16.5 M	64.5	498	5.5 ms	0.626	0.479
YOLOv10-L	24.4M	120.3	628	7.8 ms	0.659	0.498

Quan sát kết quả trên, ta có thể thấy trên cùng một phiên bản, YOLOv10 có mức độ tối ưu tốt hơn về mặt tham số mô hình cũng như độ trễ trong inference trong khi vẫn giữ được độ chính xác ngang hoặc hơn so với các phiên bản trước.

V. Trích dẫn

- [1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. ArXiv. [/abs/1506.02640](https://arxiv.org/abs/1506.02640)
- [2] Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. ArXiv. [/abs/1612.08242](https://arxiv.org/abs/1612.08242)
- [3] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. ArXiv. [/abs/1804.02767](https://arxiv.org/abs/1804.02767)
- [4] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. ArXiv. [/abs/2004.10934](https://arxiv.org/abs/2004.10934)
- [5] Jocher, G. (2020). YOLOv5 by Ultralytics. Zenodo. [/record/3908559](https://zenodo.org/record/3908559)
- [6] Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., & Wei, X. (2022). YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. ArXiv. [/abs/2209.02976](https://arxiv.org/abs/2209.02976)
- [7] Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. ArXiv. [/abs/2207.02696](https://arxiv.org/abs/2207.02696)
- [8] Jocher, G., Stoken, A., Borovec, J., Christopher, S. T. A. N., Laughing, L. C., & Ultralytics. (2023). YOLOv8 by Ultralytics. GitHub. [/ultralytics/ultralytics](https://github.com/ultralytics/ultralytics)
- [9] Wang, C., Yeh, I., & Liao, H. (2024). YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. ArXiv. [/abs/2402.13616](https://arxiv.org/abs/2402.13616)
- [10] Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024). YOLOv10: Real-Time End-to-End Object Detection. ArXiv. [/abs/2405.14458](https://arxiv.org/abs/2405.14458)
- [11] Zhang, X., Wan, F., Liu, C., Ji, R., Ye, Q. (2019). FreeAnchor: Learning to Match Anchors for Visual Object Detection. ArXiv. [/abs/1909.02466](https://arxiv.org/abs/1909.02466)
- [12] Singh, Davinder and Jain, Naman and Jain, Pranjali and Kayal, Pratik and Kumawat, Sudhakar and Batra, Nipun (2020). PlantDoc: A Dataset for Visual Plant Disease Detection. <https://doi.org/10.1145/3371158.3371196>
- [13] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (2015). SSD: Single Shot MultiBox Detector. ArXiv. https://doi.org/10.1007/978-3-319-46448-0_2
- [14] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. ArXiv. [/abs/1311.2524](https://arxiv.org/abs/1311.2524)
- [15] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. ArXiv. [/abs/1506.01497](https://arxiv.org/abs/1506.01497)
- [16] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. ArXiv. [/abs/2005.12872](https://arxiv.org/abs/2005.12872)

- [17] Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., & Joulin, A. (2021). Emerging Properties in Self-Supervised Vision Transformers. ArXiv. /abs/2104.14294
- [18] Wang, J., Song, L., Li, Z., Sun, H., Sun, J., & Zheng, N. (2020). End-to-End Object Detection with Fully Convolutional Network. ArXiv. /abs/2012.03544

- *Hết* -