## Giới thiệu đơn giản về YoLo trong bài toán object dectection

### Oject Dection là gì?

Object Detection là một bài toán quan trọng trong lĩnh vực Computer Vision, thuật toán Object Detection được chia thành 2 nhóm chính:

* Họ các mô hình RCNN (Region-Based Convolutional Neural Networks) để giải quyết các bài toán về định vị và nhận diện vật thể.
* Họ các mô hình về YOLO (You Only Look Once) dùng để nhận dạng đối tượng được thiết kế để nhận diện các vật thể real-time

## Định nghĩa về YoLo

YoLo là một mô hình mạng CNN dành cho nhiệm vụ nhận diện, phát hiện, phân loại đối tượng. YoLo được tạo ra từ việc kết hợp các convolutional layers và connected layers.

Trong đó:

* Convolutional layers sẽ trích xuất ra các feature của ảnh
* Fully - connected layers sẽ dự đoán ra xác suất đó và tọa độ của đối tượng.

## Đặc điểm

YoLo xử lý toàn bộ ảnh trong một lần truyền (single pass), khác với các mô hình như R-CNN đòi hỏi nhiều bước, giúp tăng tốc độ và phù hợp cho ứng dụng thực thời gian.

## Input tensor

Input tensor của YoLo là một mảng 3D đại diện cho ảnh đầu vào.

Có định dạng [H, W, C]

Trong đó:

* H là chiều cao
* W là chiều rộng
* C là số kênh màu (thường là 3 cho RGB)



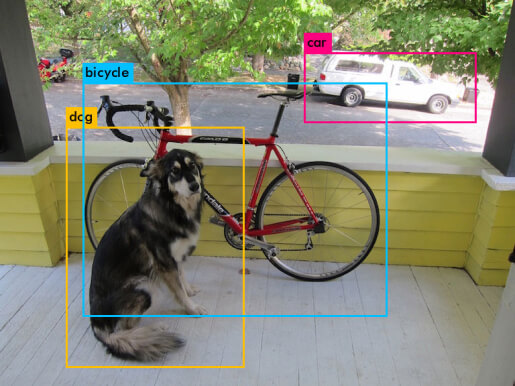
## Output tensor

Output tensor của YoLo là một mảng 3D với kích thước

[S, S, (B x 5 + C)]

Trong đó

* S là kích thước ô lưỡi (S x S)
* B x 5: Bao gồm x, y (tọa độ trung tâm), w, h (chiều rộng và chiều cao) và object score.
* C: Xác xuất thuộc các lớp (ví dụ như C=0.93 thuộc lớp dog, C=0.3 thuộc lớp bicyle, C=0.1 thuộc lớp Car)



Ảnh lấy từ: [The history of YOLO: The origin of the YOLOv1 algorithm | SuperAnnotate](https://www.superannotate.com/blog/yolov1-algorithm)

## Kiến trúc mô hình YOLO: 24 Convolutions

A diagram of a diagram of a number of boxes

AI-generated content may be incorrect.

Ảnh lấy từ: [1506.02640](https://arxiv.org/pdf/1506.02640)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STT | Tầng (Layer) | Loại | Kích thước kernel/ Stride | Số Filters | Kích thước đầu ra (Output size) |
| 1 | Input | - | - | - | 448x448 |
| 2 | Convolution | Conv | 7 x 7 / 2 | 64 | 224x224x64 |
| 3 | MaxPooling | MaxPool | 2 x 2 /2 | - | 112x112x64 |
| 4 | Convolution | Conv | 3 x 3 / 1 | 192 | 112x112x192 |
| 5 | MaxPooling | MaxPool | 2 x 2 / 2 | - | 56x56x192 |
| 6 | Convolution | Conv | 1 x 1/ 1 | 128 | 56x56x128 |
| 7 | Convolution | Conv | 3 x 3 / 1 | 256 | 56x56x256 |
| 8 | Convolution | Conv | 1 x 1 / 1 | 256 | 56x56x256 |
| 9 | Convolution | Conv | 3 x 3 / 1 | 512 | 56x56x512 |
| 10 | MaxPooling | MaxPool | 2 x 2 / 2 | - | 28x28x512 |
| 11-16 | Convolution block | Conv (x4) | 1 x 1 và 3 x 3 xen kẽ nhau | 256-512 | 28x28x512 |
| 17 | Convolution | Conv | 1 x 1 / 1 | 512 | 28x28x512 |
| 18 | Convolution | Conv | 3 x 3 / 1 | 1024 | 28x28x1024 |
| 19 | MaxPooling | MaxPool | 2 x 2 / 2 | - | 14x14x1024 |
| 20-21 | Convolution | Conv (x2) | 3 x 3 / 1 | 1024 | 14x14x1024  =>7x7x1204 |
| 22 | Fully Connected | FC | - | - | 4096 |
| 23 | Dropout | Dropout | - | - | 4096 |
| 24 | Fully Connected | FC (Output) | - | - | 7 x 7 x30 =1470 |

## Cách hoạt động của YoLo

* **Chia lưới hình ảnh**: Hình ảnh 448x448 được chia thành lưới 7 × 7. Mỗi ô lưới chịu trách nhiệm dự đoán đối tượng có tâm nằm trong ô đó.
* **Dự đoán hộp giới hạn**: Mỗi ô dự đoán 2 hộp giới hạn (tọa độ, kích thước, và confidence). Confidence phản ánh khả năng có đối tượng và độ chính xác của hộp.
* **Dự đoán lớp**: Mỗi ô dự đoán xác suất cho C lớp (dùng softmax để chuẩn hóa).
* **Loại bỏ trùng lặp**: Áp dụng Non-Maximum Suppression **(NMS)** để loại bỏ các hộp giới hạn chồng lấn, giữ lại hộp có confidence cao nhất.

A diagram of a diagram of a dog and a bicycle

AI-generated content may be incorrect.

Ảnh lấy từ: [YOLO — Intuitively and Exhaustively Explained | by Daniel Warfield | Intuitively and Exhaustively Explained | Medium](https://medium.com/intuitively-and-exhaustively-explained/yolo-intuitively-and-exhaustively-explained-83143925c7a9)

## Hàm Lỗi

Yolo sử dụng một hàm mất mát tổng hợp gồm 3 thành phần lỗi sau:

1. Classification Loss:

Mục đích: Đo sai lệch giữa xác xuất của lớp được dự đoán và lớp (class) ground truth của đối tượng trong mỗi grid cell có chứa đối tượng.

Công thức:

A black and white symbol

AI-generated content may be incorrect.

Ảnh lấy từ: [YOLO — Intuitively and Exhaustively Explained | by Daniel Warfield | Intuitively and Exhaustively Explained | Medium](https://medium.com/intuitively-and-exhaustively-explained/yolo-intuitively-and-exhaustively-explained-83143925c7a9)

1. Localization Loss

Mục đích: Đo sai lệch giữa tọa độ bounding box được dự đoán và ground-truth box, bao gồm:

* Center coordinates: x, y→ tương đối trong cell (giá trị từ 0–1)
* Width & height: w, h → sử dụng căn bậc hai: để tránh ưu tiên cho các object lớn.

Công thức:

A black text with a plus and a cross

AI-generated content may be incorrect.

Ảnh lấy từ: [YOLO — Intuitively and Exhaustively Explained | by Daniel Warfield | Intuitively and Exhaustively Explained | Medium](https://medium.com/intuitively-and-exhaustively-explained/yolo-intuitively-and-exhaustively-explained-83143925c7a9)

A math equations and formulas

AI-generated content may be incorrect.

Ảnh lấy từ: [YOLO — Intuitively and Exhaustively Explained | by Daniel Warfield | Intuitively and Exhaustively Explained | Medium](https://medium.com/intuitively-and-exhaustively-explained/yolo-intuitively-and-exhaustively-explained-83143925c7a9)

1. Confidence Loss

Mục đích: Đo sự khác biệt giữa:

* Objectness score được dự đoán (P (obj)\*IoU)
* Và ground- truth (1 nếu có object, o nếu không có object)
* Tính cho cả:
  + Grid cell có object => yêu cầu dự đoán confidence gần 1.
  + Grid cell không có object => yêu cầu dự đoán confidence gần 0.

Công thức:

A group of mathematical equations

AI-generated content may be incorrect.

Ảnh từ: [YOLO — Intuitively and Exhaustively Explained | by Daniel Warfield | Intuitively and Exhaustively Explained | Medium](https://medium.com/intuitively-and-exhaustively-explained/yolo-intuitively-and-exhaustively-explained-83143925c7a9)

## Các phiên bản của YoLo

### YoLov2

Để giải quyết các vấn đề còn tồn đọng ở YoLov1 tác giả đã phiên bản cải tiến và mang tính bước ngoặt trong YoLo.

#### InPut

* Kích thước: Thường là 416 × 416 pixel, một bội số của 32 để phù hợp với stride của mạng tích chập. YOLOv2 linh hoạt hơn YOLOv1 (cố định 448 × 448), hỗ trợ nhiều kích thước như 320 × 320, 608 × 608 trong quá trình huấn luyện (multi-scale training).
* Định dạng: Hình ảnh RGB, được chuẩn hóa giá trị pixel (thường chia cho 255 để đưa về khoảng [0, 1]).
* Chia lưới: Hình ảnh được chia thành lưới S × S, với S phụ thuộc vào kích thước đầu vào và stride. Với đầu vào 416 × 416 và stride 32, lưới là 13 × 13 (416 ÷ 32 = 13). Mỗi ô lưới chịu trách nhiệm dự đoán các đối tượng có tâm nằm trong ô đó.

#### OutPut

Đầu ra của YoLov2 cũng là một tensor chứa thông tin về hộp giới, xác xuất đối tượng, xác suất lớp. Cụ thể như sau:

Kích thước tensor: Với lưới (Grid) 13x13, B= 5 hộp neo (anchor boxes), và lớp C lớp (ví dụ, C = 20 với PASCAL VOC), tensor đầu ra có kích thước 13 x 13 x (B x (5+C))

Nội dung trong mỗi ô lưới:

* B là hộp neo bao gồm các thông số:
  + (tx, ty): Tọa độ tâm, chuẩn hóa bằng hàm sigmoid để nằm trong khoảng [0, 1] so với ô lưới.
  + (tw, th): Kích thước hộp, được điều chỉnh dựa trên anchor boxes
  + Confidence: Xác xuất tồn tại đối tượng, tính bằng Pr (Object) \* IoU (trong đó IoU là giao giữa hộp dự đoán và thực tế)
* C là xác xuất lớp: Xác suất đối tượng thuộc từng lớp.
* Anchor boxes: YOLOv2 sử dụng hộp neo được tối ưu hóa bằng thuật toán K-means trên tập dữ liệu, giúp dự đoán chính xác hơn các hộp giới hạn với tỷ lệ và kích thước đa dạng.

#### Kiến trúc mô hình YoLov2

A diagram of a diagram of a number of boxes

AI-generated content may be incorrect.

Ảnh lấy từ: [YOLO v2 Comprehensive Tutorial: Building on YOLO v1 Mistakes | by Sachinsoni | Medium](https://medium.com/@sachinsoni600517/yolo-v2-comprehensive-tutorial-building-on-yolo-v1-mistakes-aa7912292c1a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STT | Tầng (Layer) | Loại | Kích thước kernel/ Stride | Số Filters | Kích thước đầu ra (Output size) |
| 1 | Conv1 | Convolutional | 3x3/ 1 | 32 | 416x416x32 |
| 2 | Pool1 | MaxPooling | 2x2/ 2 | - | 208x208x32 |
| 3 | Conv2 | Convolutional | 3x3/ 1 | 64 | 208x208x64 |
| 4 | Pool2 | MaxPooling | 2x2/ 1 | - | 104x104x64 |
| 5 | Conv3 | Convolutional | 3x3/ 1 | 128 | 104x104x128 |
| 6 | Conv4 | Convolutional | 1x1/ 1 | 64 | 104x104x64 |
| 7 | Conv5 | Convolutional | 3x3/ 1 | 128 | 104x104x128 |
| 8 | Pool3 | MaxPooling | 2x2/ 1 | - | 52x52x128 |
| 9 | Conv6 | Convolutional | 3x3/ 1 | 256 | 52x52x256 |
| 10 | Conv7 | Convolutional | 1x1/ 1 | 128 | 52x52x128 |
| 11 | Conv8 | Convolutional | 3x3/ 1 | 256 | 52x52x256 |
| 12 | Pool4 | MaxPooling | 2x2/ 2 | - | 26x26x256 |
| 13 | Conv9 | Convolutional | 3x3/ 1 | 512 | 26x26x512 |
| 14 | Conv10 | Convolutional | 1x1/ 1 | 256 | 26x26x256 |
| 15 | Conv11 | Convolutional | 3x3/ 1 | 512 | 26x26x512 |
| 16 | Conv12 | Convolutional | 1x1/ 1 | 256 | 26x26x256 |
| 17 | Conv13 | Convolutional | 3x3/ 1 | 512 | 26x26x512 |
| 18 | Pool5 | MaxPooling | 2x2/ 2 | - | 13x13x512 |
| 19 | Conv14 | Convolutional | 3x3/ 1 | 1024 | 13x13x1024 |
| 20 | Conv15 | Convolutional | 1x1/ 1 | 512 | 13x13x512 |
| 21 | Conv16 | Convolutional | 3x3/ 1 | 1024 | 13x13x1024 |
| 22 | Conv17 | Convolutional | 1x1/ 1 | 512 | 13x13x512 |
| 23 | Conv18 | Convolutional | 3x3/ 1 | 1024 | 13x13x1024 |
| 24 | Conv19 | Convolutional | 3x3/ 1 | 1024 | 13x13x1024 |
| 25 | Conv20 | Convolutional | 1x1/ 1 | 125 | 13x13x125 |

#### Cách hoạt động của YoLov2

#### Điểm cải tiến hơn so với YoLov1

* Chuẩn hóa dữ liệu theo lô(Batch normalization):

YOLOv2 áp dụng batch normalization (BN) sau mỗi tầng tích chập trong backbone Darknet-19.

Lợi ích:

* + Ổn định huấn luyện: Giảm hiện tượng gradient vanishing/ exploding, giúp mạng hội tụ nhanh hơn.
  + Loại bỏ dropout: BN có tác dụng như một hình thức chính quy hóa (regularization), nên YOLOv2 không cần dùng dropout như YOLOv1.
  + Tăng độ chính xác: BN giảm sự phụ thuộc vào giá trị khởi tạo trọng số, cải thiện mAP (mean Average Precision) của YOLOv2 khoảng 2% trên tập PASCAL VOC.
* Phân loại độ phân giải cao:

Cải tiến: YOLOv2 huấn luyện trước backbone Darknet-19 trên tập ImageNet với độ phân giải cao (448 × 448) để học các đặc trưng phân loại tốt hơn, trước khi tinh chỉnh (fine-tune) cho phát hiện đối tượng với đầu vào 416 × 416.

Sau khi huấn luyện trước, mạng được tinh chỉnh thêm 10 epoch trên ImageNet với độ phân giải 448 × 448 để thích nghi với độ phân giải cao.

Lợi ích:

* + Cải thiện phân loại: Độ phân giải cao giúp mạng học được các đặc trưng chi tiết hơn, tăng độ chính xác khi phân loại đối tượng.
  + Tăng mAP: Độ phân giải cao cải thiện khả năng nhận diện các đối tượng nhỏ và phức tạp.
* Đặc trung phân giải cao:

Cải tiến

* + YOLOv2 sử dụng passthrough layer để kết hợp đặc trưng độ phân giải cao từ tầng trước (26 × 26 × 512) với tầng cuối (13 × 13 × 1024).
  + Passthrough layer tổ chức bản đồ đặc trưng 26 × 26 × 512 thành 13 × 13 × 2048 bằng cách chia mỗi kênh thành 4 phần, sau đó nối với bản đồ đặc trưng tầng cuối.

Lợi ích:

* + Phát hiện đối tượng nhỏ: Đặc trưng độ phân giải cao (26 × 26) chứa thông tin chi tiết hơn, giúp YOLOv2 nhận diện các đối tượng nhỏ, vốn là điểm yếu của YOLOv1 (lưới 7 × 7 quá thô).
  + Tăng mAP: Kết hợp đặc trưng đa tỷ lệ cải thiện độ chính xác, đặc biệt trên các tập dữ liệu như PASCAL VOC và COCO.
* Lớp Truyền tiếp

Cải tiến:

* + Passthrough layer là một kỹ thuật cụ thể để khai thác đặc trưng phân giải cao, như đã đề cập ở trên. Cụ thể là Nó lấy bản đồ đặc trưng từ tầng conv13 (26 × 26 × 512), tổ chức lại thành 13 × 13 × 2048, rồi nối với bản đồ đặc trưng tầng cuối (13 × 13 × 1024) để tạo ra tensor 13 × 13 × 3072 trước khi dự đoán.

Lợi ích:

* Cải thiện phát hiện chi tiết: Giúp YOLOv2 nhận diện các đối tượng nhỏ hoặc các chi tiết phức tạp, khắc phục hạn chế của YOLOv1.
* Tăng tính tổng quát: Kết hợp đặc trưng đa tỷ lệ giúp mô hình xử lý tốt hơn các cảnh có nhiều đối tượng với kích thước khác nhau.

A diagram of a diagram of a layer of cubes

AI-generated content may be incorrect.

Ảnh lấy từ: [YOLO v2 Comprehensive Tutorial: Building on YOLO v1 Mistakes | by Sachinsoni | Medium](https://medium.com/@sachinsoni600517/yolo-v2-comprehensive-tutorial-building-on-yolo-v1-mistakes-aa7912292c1a)

* Giải quyết các vấn đề với Bounding Box:

1. Loại bỏ đi fully connected

Cải tiến:

YOLOv2 loại bỏ hoàn toàn các tầng kết nối đầy đủ (fully connected layers) được sử dụng trong YOLOv1, thay vào đó chỉ dùng các tầng tích chập trong backbone Darknet-19. Tầng cuối (conv20) trực tiếp dự đoán tensor 13 × 13 × 125 dựa trên bản đồ đặc trưng thay vì sử dụng fully connected như YoLov1.

Lợi ích:

* + Giảm tham số giúp tăng tốc độ
  + Linh hoạt hơn: Mạng chi dùng tích chập hỗ trợ đầu vào kích thước khác nhau, dùng để multi scale training

1. Dự đoán đối tượng dựa vào bouding box (giải quyết vấn đề một ô grid cell chỉ được dự đoán một đối tượng)

Cải tiến:

* + YOLOv2 giới thiệu anchor boxes (hộp neo), với mỗi ô lưới 13 × 13 dự đoán 5 hộp neo, thay vì chỉ 2 hộp cố định như YOLOv1.
  + Anchor boxes được tối ưu bằng thuật toán K-means trên tập dữ liệu để phù hợp với các tỷ lệ đối tượng phổ biến.
  + Mỗi ô lưới dự đoán nhiều hộp neo, cho phép phát hiện nhiều đối tượng trong cùng một ô.

Lợi ích:

* + Tăng số lượng đối tượng phát hiện: Một ô lưới có thể dự đoán nhiều đối tượng với các tỷ lệ khác nhau, khắc phục hạn chế của YOLOv1 (mỗi ô chỉ dự đoán một lớp).
  + Cải thiện mAP: Anchor boxes giúp dự đoán hộp giới hạn chính xác hơn, đặc biệt với các đối tượng có tỷ lệ không đồng đều.

1. Giải quyết vấn đề lỗi cao và mất ổn định khi huấn luyện

Cải tiến: Điều chỉnh hàm mất mát để cân bằng tốt hơn giữa lỗi tọa độ, lỗi confidence, và lỗi xác suất lớp. Cụ thể:

* + Sử dụng sigmoid để dự đoán tọa độ tâm (tx, ty), giới hạn trong ô lưới.
  + Dự đoán kích thước hộp (tw, th) dựa trên anchor boxes, giảm sai lệch lớn.
  + Cân bằng trọng số giữa ô có/không có đối tượng, giảm lỗi từ các ô không có đối tượng (vấn đề lớn ở YOLOv1).

Lợi ích:

* + Giảm lỗi định vị: Dự đoán dựa trên anchor boxes giúp hộp giới hạn chính xác hơn.

1. Giải quyết vấn đề liên quan đến hộp neo (Ứng dụng kết hợp cả cách cũ và mới đó là dùng grid cell để dự đoán tâm của hộp neo, và vẫn dùng anchor boxes để tìm w, h)

Cải tiến:

YOLOv2 kết hợp cách tiếp cận của YOLOv1 (dùng grid cell để dự đoán tâm hộp) với kỹ thuật anchor boxes (dự đoán kích thước hộp). Cụ thể:

* + Tâm hộp (tx, ty): Dự đoán bằng sigmoid, chuẩn hóa trong phạm vi ô lưới [0, 1].
  + Kích thước hộp (tw, th): Dự đoán dựa trên anchor boxes

Lợi ích:

* + Dự đoán linh hoạt hơn: Kết hợp grid cell (cho tâm) và anchor boxes (cho kích thước) giúp YOLOv2 dự đoán hộp giới hạn chính xác hơn ở các tỷ lệ khác nhau.
  + Tăng khả năng phát hiện đa dạng: Có thể xử lý nhiều đối tượng trong cùng ô lưới.

#### Hàm lỗi

Loss Function Yolov2 là hàm tổng hợp nhiều thành phần, mỗi thành phần tập trung vào một khía cạnh cụ thể của bài toán phát hiện đối tượng. Cụ thể như sau:

* Localization Loss: Đảm bảo dự đoán vị trí và kích thước của bouding box.
* Confidence Loss: Dự đoán xác xuất có đối tượng trong một ô lưới (Grid cell).
* Classification Loss: Dự đoán đúng lớp của đối tượng.
* Anchor boxes Aligment: Đảm bảo các anchorbox được căn chỉnh phù hợp với dự đoán.

Công thức tổng hợp:

A diagram of mathematical equations

AI-generated content may be incorrect.

### YoLov3

* **Bounding box prediction**: dự đoán điểm đối tượng cho mỗi hộp giới hạn bằng cách sử dụng hồi quy logistic
* **Class Prediction:** dùng binary cross-entropy thay vì sofmax để đào tạo các bộ phân loại logistic độc lập và đưa ra vấn đề như một phân loại đa nhãn. Cho phép gán nhiều nhãn cho cùng một hộp, giúp xử lý các loại dữ liệu phức tạp với các nhãn chồng chéo. Ví dụ: cùng một đối tượng có thể là “a Person” và “a Man”
* **Newbackbone:** YOLOv3 sử dụng **DarkNet-53** làm backbone để trích xuất đặc trưng.
* Bổ sung ý tưởng về **FPN** để tận dụng lợi ích từ tất cả các tính toán trước đó
* **Spatial pyramid pooling (SPP)**
* **Multi-scalePredictions: tương tự như Feature Pyramid Networks.** Dự đoán ở 3 cấp độ (13×13, 26×26, 52×52) nhằm cải thiện đối tượng nhỏ
* **Bounding box priors:** dùng k-means để tìm bounding box trước anchor boxes. Sử dụng 3 boxes trước cho 3 thang đo khác nhau.

#### Các cải tiến chính

### YoLov4

Backbone: CSPDarknet-53 + Mish

Neck: Spatial Pyramid Pooling (SPP) + Modified PANet + SAM

* SPP: tương tự YOLOV3 cho multi-scale feature pooling
* PANet thay vì FPN như V3, mục đích là tạo ra một hệ thống phân cấp tính năng nhiều giai đoạn phong phú để phát hiện đối tượng mạnh mẽ
* SAM: vì nó cải thiện độ chính xác và giảm độ trễ suy luận
  + SPP giúp tăng trường tiếp nhận, PANet giúp tổng hợp tính năng tốt hơn

Head: YOLOV3 head

Self-adversarial Training (SAT): giúp mô hình mạnh mẽ hơn trước sự nhiễu loạn.

Bag-of-Freebies (BoF): các kỹ thuật sửa đổi phương pháp đào tạo hoặc tăng chi phí đào tạo mà không ảnh hưởng đến chi phí trong quá trình suy luận. Ví dụ điển hình là tăng cường dữ liệu.

Bag-of-Specials (BoS): Ngược lại với BoF, là các phương pháp làm tăng chi phí một chút tại thời điểm suy luận nhưng cải thiện đáng kể độ chính xác. Ví dụ: SPP và PANet

Cross Stage Partial Network: được thiết kế để suy luận nhanh hơn trong khi vẫn giữ được độ chính xác của mô hình ban đầu

#### Các cải tiến chính

* An Enhanced Architecture with **Bag-of-Specials (BoS)**
* Integrating **bag-of-freebies (BoF)** for an Advanced Training Approach
* **Self-adversarial Training (SAT)**
* **Hyperparameter Optimization with Genetic Algorithms**: để thay đổi tốc độ học trong quá trình đào tạo. Nó bắt đầu giảm tốc độ học chậm, sau đó giảm nhanh ở giữa quá trình đào tạo và kết thúc bằng giảm nhẹ

### YoLov5

Mô hình YOLOv5 được phát triển trong Pytorch thay vì Darknet kết hợp với thuật toán Ultralytics tên là AutoAnchor.

Backbone: **CSP-DarkNet53:** được sửa đổi, bắt đầu bằng 1 Stem, một lớp tích chập có strided với kích thước cửa sổ lớn để giảm bộ nhứ và chi phí tính toán.

Neck: **SPPF** and **CSP-PANet**

* **Spatial Pyramid Pooling Fast (SPPF):** thay vì chuyển đầu vào đến ba lớp max-pool khác nhau, đầu ra của một khối max-pool sẽ được đưa đến lớp max-pool tiếp theo, điều này làm cho quá trình nhanh hơn.
* **CSP-PANet**: giống như PANet. Thay vào đó, họ thêm một vài lớp CSP vào giữa PANet. Sự thay đổi này dẫn đến tốc độ xử lý nhanh hơn gấp đôi

Head: YOLOv3 Head

\* **AutoAnchors**: mô hình bắt đầu với các anchor boxes trước đó được tạo ra từ việc chạy k-means trên các ground truth bounding boxes. Sau đó, các bounding boxes này được cập nhật bằng *thuật toán genetic evolution (GE).* Thuật toán GE phát triển các anchors này qua 1.000 thế hệ, sử dụng CIoU loss và Best Possible Recall để đánh giá mức độ phù hợp của nó.

#### Các cải tiến chính

### YoLoR

YOLOR là viết tắt của You Only Learn One Representation. Họ đã phát triển một cách tiếp cận **Multi-task Learning**.

**Multi-task Learning (MTL):** được đào tạo để thực hiện nhiều nhiệm vụ khác nhau cùng một lúc, thay vì đào tạo các mô hình nhiệm vụ cụ thể. Bằng cách đào tạo song song một mô hình cho nhiều nhiệm vụ, người ta có thể cải thiện khả năng khái quát hóa của mô hình

YOLOR đặt mục tiêu khắc phục các phương pháp học chung truyền thống dẫn đến việc tạo tính năng không tối ưu bằng cách mã hóa kiến thức ngầm của mạng nơ-ron để áp dụng cho nhiều tác vụ, tương tự như cách con người sử dụng kinh nghiệm trong quá khư để tiếp cận các vấn đề mới.

* + Kết quả cho thấy việc đưa kiến thức ngầm vào mạng nơ-ron mang lại lợi ích cho tất cả các tác vụ.

#### Các cải tiến chính

### YoLoX

Được phát triển trong Pytorch và sử dụng YOLO3 từ Ultralytics làm điểm khởi đầu với 5 thay đổi chính.

#### Các cải tiến chính

Anchor-free: giúp đơn giản hóa quả trình huấn luyện và giải mã.

Multi positives: để bù đắp cho sự mất cân bằng lớn do thiếu anchors tạo ra

Decoupled head: YOLOX tách phần head thành 2 head: một cho nhiệm vụ phân loại và một cho nhiệm vụ hồi quy

Advanced label assignment: simOTA – một phiên bản đơn giản hóa của Optimal Transport (OT)

Strong augmentations: dùng MixUP và tăng cường Mosaic

### YoLo v6

YOLOv6 là một kiến trúc anchor-free, decoupled head với backbone là **EfficientRep**

**EfficientRep** bao gồm một **RepVGGBlock stem,** tiếp theo là 4 khối ERBlock. ERBlock được tạo thành từ RepVGGBlock và RepBlock. Lớp cuối cùng của ERBlock có RepVGGBlock và RepBlock, ngoại trừ **SimSPPF**

Neck: Rep-PAN. không có gì khác PANet, chỉ thay thế CSPBlock bằng **RepBlock** (đối với kiểu máy nhỏ) hoặc **CSPStackRep Block** (đối với kiểu máy lớn)

Head: sử dụng phiên bản tùy chỉnh của decoupled head từ YOLOX với tên là **hybrid-channel strategy** ( giảm một lớp conv 3×3 và cùng chia tỷ lệ chiều rộng của head theo hệ số chiều rộng cho backbone và neck)

Ngoài ra còn các kỹ thuật Label Assignment techniques như simOTA, TAL(Task alignment learning), hàm loss functions mới để giúp mô hình nhanh hơn và chính xác hơn

#### Các cải tiến chính

A new backbone based on RepVGG: EfficientRep

Label assignment: dùng Task alignment learning

New classification and regression losses: dùng classification VariFocal loss và SIoU/GIoU regression loss.

Aself-distillation: cho nhiệm vụ hồi quy và phân loại

Aquantization scheme: phát hiện bằng RepOptimizer và channel-wise distillation

### YoLo v7

Được huấn luyện chỉ bằng tập dữ liệu MS COCO với backbones không được huấn huấn luyện trước. YOLOv7 đã đề xuất một vài thay đổi về kiến trúc và 1 loạt các bag-of-freebies giúp tăng độ chính xác mà không ảnh hưởng đến tốc độ suy luận, chỉ ảnh hưởng đến thời gian đào tạo.

#### Các cải tiến chính

Extended efficient layer aggregation network (E-ELAN): là một mạng tổng hợp lớp hiệu quả mở rộng, một **biến thể của ELAN.** ELAN được lấy cảm hứng từ VoVNet và CSPNet

Model scaling for concatenation-based models: YOLOv7 đã đề xuất một chiến lược mới để mở rộng quy mô các mô hình dựa trên nối kết trong đó độ sâu và chiều rộng của khối được mở rộng theo cùng một hệ số để duy trì cấu trúc tối ưu của mô hình.

### So sánh các mô hình từ V3 đến V7 trên tập dữ liệu MS COCO

|  |  |
| --- | --- |
| **Phiên bản** | **AP** |
| YOLOv3 | 36,2% |
| YOLOv4 | 43,5% |
| YOLOv5 | 50,7% |
| YOLOR | 55,4% |
| YOLOX | 50,1% |
| YOLOv7 – E6 | 55,9% |
| YOLOv6 | 57,2% |

### YoLo v8

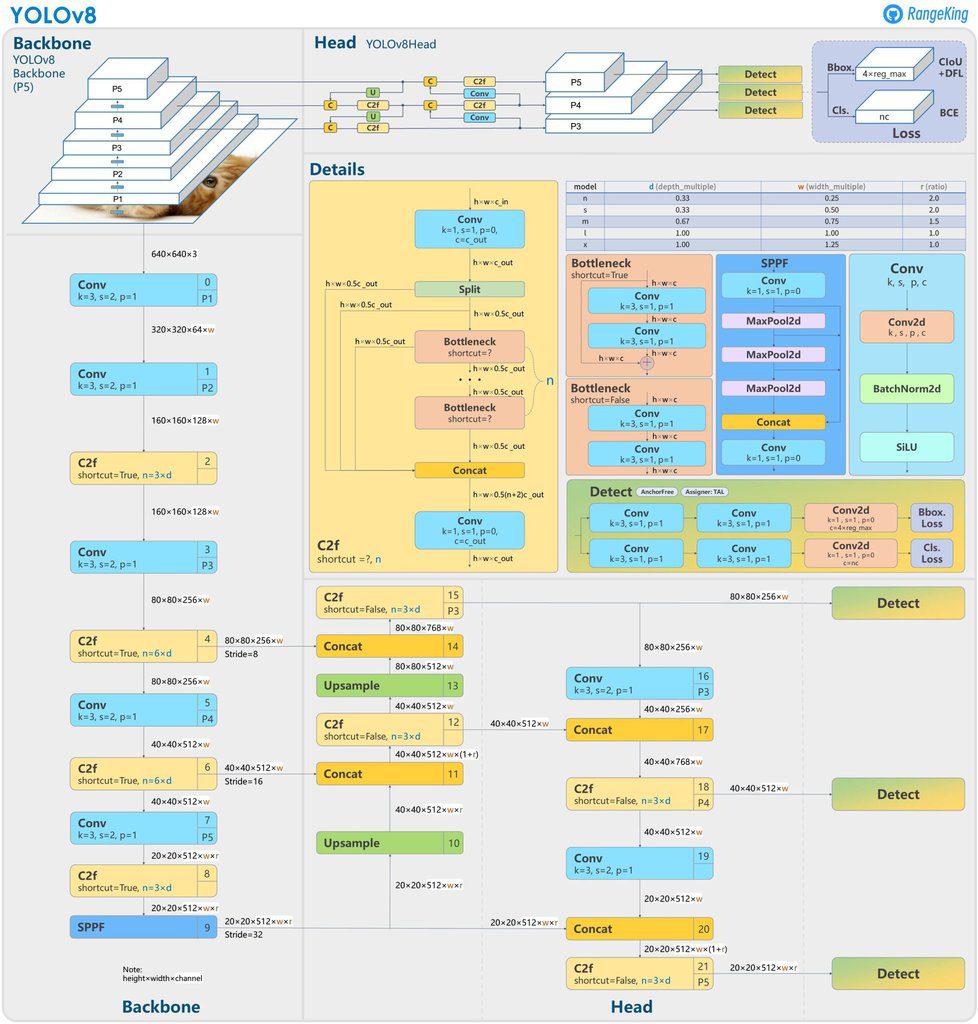
#### Input

* Mô hình YOLOv8 nhận ảnh RGB làm đầu vào với kích thước chuẩn hóa về (640×640×3)
* Trước khi đưa vào mô hình:
* Ảnh được resize về kích thước 640×640
* Chuẩn hóa giá trị pixel về khoảng [0, 1]
* Biến đổi ảnh thành tensor: **[B, 3, 640, 640]** trong đó:
  + B: batch size (số ảnh trong một lần forward)
  + 3: số kênh màu RGB

#### Output

* YOLOv8 sử dụng Anchor-Free Head, do đó đầu ra sẽ không dựa vào anchor box cố định như các phiên bản cũ.
* Đầu ra của mô hình là tensor có dạng: [N, 4 + 1 + C], trong đó:
  + N: số điểm đặc trưng (grid cells) trên feature maps, ví dụ tổng cộng 8400 điểm từ 3 cấp độ (P3, P4, P5)
  + 4: tọa độ bounding box (x\_center, y\_center, width, height)
  + 1: objectness score – xác suất tồn tại đối tượng tại điểm đó
  + C: số lớp phân loại

#### Kiến trúc YoLov8



Ảnh lấy từ: [Mastering All YOLO Models from YOLOv1 to YOLOv12](https://learnopencv.com/mastering-all-yolo-models/#yolov8)

BackBone: Trích xuất đặc trưng từ ảnh gốc sử dụng

Neck: Kết hợp đặc trưng từ nhiều tầng (PAN-FPN), giúp mô hình có thể tăng khả năng phát hiện đối tượng ở nhiều kích thước.

A diagram of a diagram

AI-generated content may be incorrect.

Ảnh lấy từ: [PANet: Path Aggregation Network In YOLOv4 | by Miracle R | Clique Community | Medium](https://medium.com/clique-org/panet-path-aggregation-network-in-yolov4-b1a6dd09d158)

Head: Dự đoán bounding box, class, objectness. Anchor-free, đơn giản hơn các phiên bản YOLO trước.

#### Cách hoạt động của YoLov8

* Ảnh đầu vào → backbone (trích đặc trưng)
* Đặc trưng trung gian qua neck (PAN-FPN) để kết hợp theo chiều xuôi + ngược
* Đặc trưng đầu ra từ các tầng P3, P4, P5 → head → dự đoán trực tiếp các:
* Toạ độ bbox
* Xác suất object
* Phân lớp đối tượng
* Output được lọc qua NMS để giữ lại bbox hợp lệ nhất

#### Inference

* Model nhận ảnh [1, 3, 640, 640].
* Trả về list các bounding boxes có tọa độ, nhãn, confidence score.
* Sử dụng mô-đun ultralytics.

#### Các bước train mô hình YoLov8

1. Chuẩn bị dữ liệu

Thu thập dữ liệu:

* Thu thập hình ảnh liên quan đến bài toán (ví dụ: ảnh xe hơi, người đi bộ, động vật, v.v.).
* Đảm bảo đa dạng
* Số lượng cần thiết: Lý tưởng là 1000+

Gắn nhãn cho dữ liệu:

* Sử dụng công cụ đánh nhãn như LabelIMG, Roboflow ...
* Định dạng nhãn: Yolov8 dùng định dạng YoLo txt ( mỗi file .txt chứa [Class id, xcenter,ycenter, width, height]

Tổ chức dữ liệu:

* Cấu trúc thư mục (theo chuẩn của YoLov8)
* Tăng cường dữ liệu (nếu cần)
* File data.yaml định nghĩa dataset:

Ví dụ:

train: ../train/images

val: ../valid/images

test: ../test/images

nc: 1

names: ['car']

1. Cài đặt môi trường

Cài đặt Python

Cài đặt Pytorch

Cài đặt Ultralytics YOLOv8

!pip install ultralytics

Cài đặt thư viện

from ultralytics import YOLO

import numpy as np

from PIL import Image

import requests

from io import BytesIO

import cv2

from google.colab.patches import cv2\_imshow

1. Chọn mô hình Yolov8

model = YOLO("yolov8n.pt")

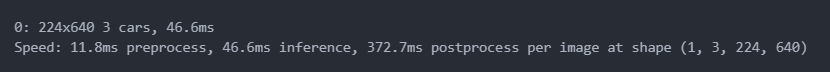
dự đoán trên hình ảnh

image = cv2.imread("/content/drive/MyDrive/Khóa Luận/Data\_Raw/2011\_09\_26/2011\_09\_26\_drive\_0015\_sync/image\_02/data/0000000000.png")

image = np.asarray(image)

image = image.copy()

results = model.predict(image)



Hiển thị bounding boxes

def box\_label(*image*, *box*, *label*='', *color*=(128, 128, 128), *txt\_color*=(255, 255, 255)):

  lw = max(round(sum(*image*.shape) / 2 \* 0.003), 2)

  p1, p2 = (int(*box*[0]), int(*box*[1])), (int(*box*[2]), int(*box*[3]))

  cv2.rectangle(*image*, p1, p2, *color*, *thickness*=lw, *lineType*=cv2.LINE\_AA)

  if *label*:

    tf = max(lw - 1, 1)  *# font thickness*

    w, h = cv2.getTextSize(*label*, 0, *fontScale*=lw / 3, *thickness*=tf)[0]  *# text width, height*

    outside = p1[1] - h >= 3

    p2 = p1[0] + w, p1[1] - h - 3 if outside else p1[1] + h + 3

    cv2.rectangle(*image*, p1, p2, *color*, -1, cv2.LINE\_AA)  *# filled*

    cv2.putText(*image*,

*label*, (p1[0], p1[1] - 2 if outside else p1[1] + h + 2),

                0,

                lw / 3,

*txt\_color*,

*thickness*=tf,

*lineType*=cv2.LINE\_AA)

def plot\_bboxes(*image*, *boxes*, *labels*=[], *colors*=[], *score*=True, *conf*=None):

*#Define COCO Labels*

  if *labels* == []:

*labels* = {0: u'\_\_background\_\_', 1: u'person', 2: u'bicycle',3: u'car', 4: u'motorcycle', 5: u'airplane', 6: u'bus', 7: u'train', 8: u'truck', 9: u'boat', 10: u'traffic light', 11: u'fire hydrant', 12: u'stop sign', 13: u'parking meter', 14: u'bench', 15: u'bird', 16: u'cat', 17: u'dog', 18: u'horse', 19: u'sheep', 20: u'cow', 21: u'elephant', 22: u'bear', 23: u'zebra', 24: u'giraffe', 25: u'backpack', 26: u'umbrella', 27: u'handbag', 28: u'tie', 29: u'suitcase', 30: u'frisbee', 31: u'skis', 32: u'snowboard', 33: u'sports ball', 34: u'kite', 35: u'baseball bat', 36: u'baseball glove', 37: u'skateboard', 38: u'surfboard', 39: u'tennis racket', 40: u'bottle', 41: u'wine glass', 42: u'cup', 43: u'fork', 44: u'knife', 45: u'spoon', 46: u'bowl', 47: u'banana', 48: u'apple', 49: u'sandwich', 50: u'orange', 51: u'broccoli', 52: u'carrot', 53: u'hot dog', 54: u'pizza', 55: u'donut', 56: u'cake', 57: u'chair', 58: u'couch', 59: u'potted plant', 60: u'bed', 61: u'dining table', 62: u'toilet', 63: u'tv', 64: u'laptop', 65: u'mouse', 66: u'remote', 67: u'keyboard', 68: u'cell phone', 69: u'microwave', 70: u'oven', 71: u'toaster', 72: u'sink', 73: u'refrigerator', 74: u'book', 75: u'clock', 76: u'vase', 77: u'scissors', 78: u'teddy bear', 79: u'hair drier', 80: u'toothbrush'}

*#Define colors*

  if *colors* == []:

*#colors = [(6, 112, 83), (253, 246, 160), (40, 132, 70), (205, 97, 162), (149, 196, 30), (106, 19, 161), (127, 175, 225), (115, 133, 176), (83, 156, 8), (182, 29, 77), (180, 11, 251), (31, 12, 123), (23, 6, 115), (167, 34, 31), (176, 216, 69), (110, 229, 222), (72, 183, 159), (90, 168, 209), (195, 4, 209), (135, 236, 21), (62, 209, 199), (87, 1, 70), (75, 40, 168), (121, 90, 126), (11, 86, 86), (40, 218, 53), (234, 76, 20), (129, 174, 192), (13, 18, 254), (45, 183, 149), (77, 234, 120), (182, 83, 207), (172, 138, 252), (201, 7, 159), (147, 240, 17), (134, 19, 233), (202, 61, 206), (177, 253, 26), (10, 139, 17), (130, 148, 106), (174, 197, 128), (106, 59, 168), (124, 180, 83), (78, 169, 4), (26, 79, 176), (185, 149, 150), (165, 253, 206), (220, 87, 0), (72, 22, 226), (64, 174, 4), (245, 131, 96), (35, 217, 142), (89, 86, 32), (80, 56, 196), (222, 136, 159), (145, 6, 219), (143, 132, 162), (175, 97, 221), (72, 3, 79), (196, 184, 237), (18, 210, 116), (8, 185, 81), (99, 181, 254), (9, 127, 123), (140, 94, 215), (39, 229, 121), (230, 51, 96), (84, 225, 33), (218, 202, 139), (129, 223, 182), (167, 46, 157), (15, 252, 5), (128, 103, 203), (197, 223, 199), (19, 238, 181), (64, 142, 167), (12, 203, 242), (69, 21, 41), (177, 184, 2), (35, 97, 56), (241, 22, 161)]*

*colors* = [(89, 161, 197),(67, 161, 255),(19, 222, 24),(186, 55, 2),(167, 146, 11),(190, 76, 98),(130, 172, 179),(115, 209, 128),(204, 79, 135),(136, 126, 185),(209, 213, 45),(44, 52, 10),(101, 158, 121),(179, 124, 12),(25, 33, 189),(45, 115, 11),(73, 197, 184),(62, 225, 221),(32, 46, 52),(20, 165, 16),(54, 15, 57),(12, 150, 9),(10, 46, 99),(94, 89, 46),(48, 37, 106),(42, 10, 96),(7, 164, 128),(98, 213, 120),(40, 5, 219),(54, 25, 150),(251, 74, 172),(0, 236, 196),(21, 104, 190),(226, 74, 232),(120, 67, 25),(191, 106, 197),(8, 15, 134),(21, 2, 1),(142, 63, 109),(133, 148, 146),(187, 77, 253),(155, 22, 122),(218, 130, 77),(164, 102, 79),(43, 152, 125),(185, 124, 151),(95, 159, 238),(128, 89, 85),(228, 6, 60),(6, 41, 210),(11, 1, 133),(30, 96, 58),(230, 136, 109),(126, 45, 174),(164, 63, 165),(32, 111, 29),(232, 40, 70),(55, 31, 198),(148, 211, 129),(10, 186, 211),(181, 201, 94),(55, 35, 92),(129, 140, 233),(70, 250, 116),(61, 209, 152),(216, 21, 138),(100, 0, 176),(3, 42, 70),(151, 13, 44),(216, 102, 88),(125, 216, 93),(171, 236, 47),(253, 127, 103),(205, 137, 244),(193, 137, 224),(36, 152, 214),(17, 50, 238),(154, 165, 67),(114, 129, 60),(119, 24, 48),(73, 8, 110)]

*#plot each boxes*

  for box in *boxes*:

*#add score in label if score=True*

    if *score* :

      label = *labels*[int(box[-1])+1] + " " + str(round(100 \* float(box[-2]),1)) + "%"

    else :

      label = *labels*[int(box[-1])+1]

*#filter every box under conf threshold if conf threshold setted*

    if *conf* :

      if box[-2] > *conf*:

        color = *colors*[int(box[-1])]

        box\_label(*image*, box, label, color)

    else:

      color = *colors*[int(box[-1])]

      box\_label(*image*, box, label, color)

*#show image*

*image* = cv2.cvtColor(*image*, cv2.COLOR\_BGR2RGB)

  try:

    import google.colab

    IN\_COLAB = True

  except:

    IN\_COLAB = False

  if IN\_COLAB:

    cv2\_imshow(*image*) *#if used in Colab*

  else :

    cv2.imshow(*image*) *#if used in Python*

plot\_bboxes(image, results[0].boxes.data, *score*=False)

A road with a green sign

AI-generated content may be incorrect.

1. Cấu hình training

model = YOLO("yolov8n.pt")

Cấu hình bao gồm tham số huấn luyện và file data. yaml

Tham số huấn luyện (trong code hoặc file config)

Epochs: Số vòng lặp (thường 50-300, tùy dataset)

batch\_size: Kích thước batch (8-32, tùy GPU, ví dụ: 16 cho 8GB VRAM).

imgsz: Kích thước ảnh đầu vào (mặc định 640x640).

optimizer: SGD hoặc Adam (mặc định AdamW trong YOLOv8).

lr0: Learning rate ban đầu (thường 0.01).

patience: Số epoch chờ nếu không cải thiện (mặc định 50).

1. Training mô hình

Chạy lệnh huấn luyện

results = model.train(*data*="coco128.yaml", *epochs*=10)

A screenshot of a computer

AI-generated content may be incorrect. A black and white screen with white text

AI-generated content may be incorrect.

1. Đánh giá (eval)

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A person and person giving each other a high five

AI-generated content may be incorrect.

1. Tối ưu hóa (nếu cần thiết)