Chiara Solito and
Tommaso Del Prete

# Comparison between Faster RCNN and YOLOv5

Computer Vision and
Deep Learning Course
Project

Chiara Solito
VR487795

Tommaso Del Prete
VR488382

# Table of Contents

# Introduction

Object detection is a subfield of computer vision that focuses on identifying and localizing objects within digital images or video frames. It involves the development of algorithms and models that can automatically detect and classify different objects of interest in a given scene.

During the project conceptualization phase, we opted to concentrate on conducting a performance analysis between two influential Object Detection algorithms, namely YOLO (specifically version v5) and Faster RCNN (with Backbone Resnet 50v2). This evaluation will be carried out on a novel and less-explored dataset.

# State of the Art

We briefly report here some of the detectors that are most famous and that we took an interest in during the course of Computer Vision and Deep Learning.

## CNN BASED

Convolutional Neural Network (CNN) based object detectors are typically classified into two categories, two-stage, and one-stage detection methods.

## ONE STAGE

One-stage object detectors use a single network pass for both classification and regression, without a separate proposal stage. They predict class probabilities, bounding box locations, and confidence scores at all image positions. They are divided into anchor-based (using predefined boxes) and keypoint-based (treating objects as points) detectors.

## TWO STAGE

Two-stage object detection methods like Two-Stage Detectors and RCNN involve a Region Proposal Network (RPN) for proposing regions of interest (ROIs), with subsequent classification and regression heads. RCNN uses selective search to find potential object regions, passing them through a CNN. Its successor, Fast-RCNN, improves speed and training by splitting losses.

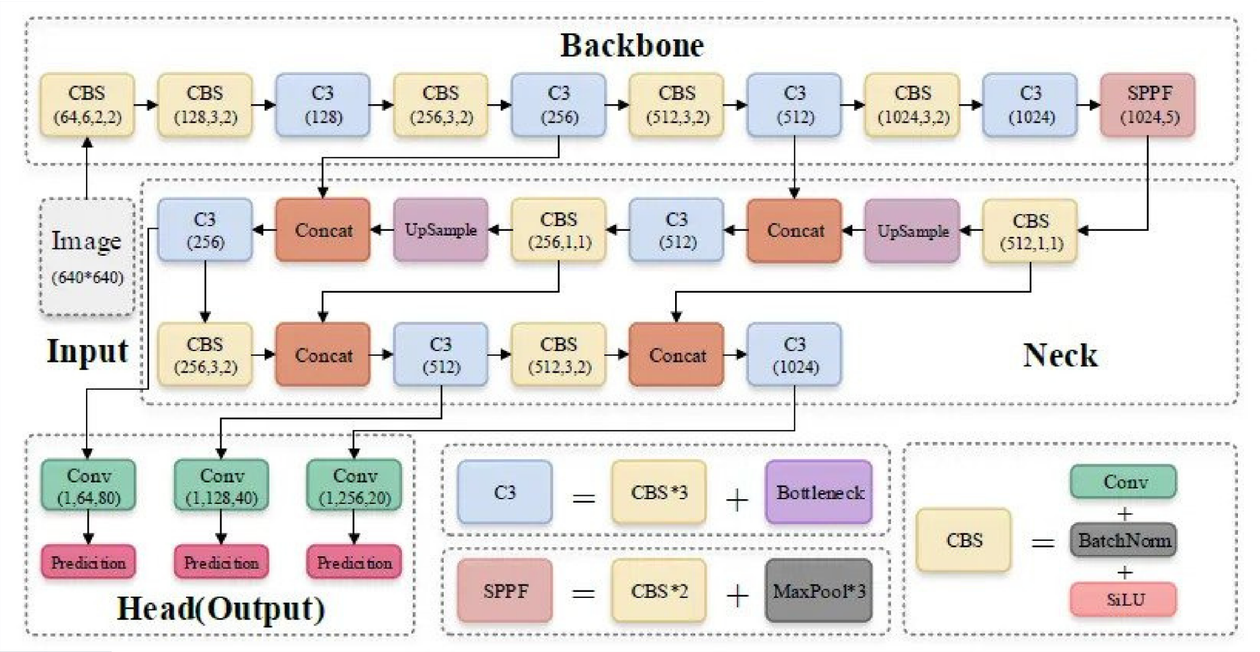

# Objectives

Our project endeavors to analyze a novel dataset and evaluate the suitability of two architectural models discussed in class. By exploring both pretrained and non-pretrained models, we aim to ascertain which one best aligns with our requirements.

**YOLO (You Only Look Once) and Faster R-CNN are widely used object detection algorithms, differing in approach and performance.** YOLO uses a single-stage approach, predicting bounding boxes and class probabilities for image grid cells, enabling real-time inference for fast applications. Yet, YOLO struggles with small object localization and occasional false results. On the other hand, Faster R-CNN uses a two-stage approach with a proposal network for potential bounding boxes and subsequent classification/refinement, yielding higher accuracy and localization, especially for small objects. However, its two-stage design leads to slower inference compared to YOLO.

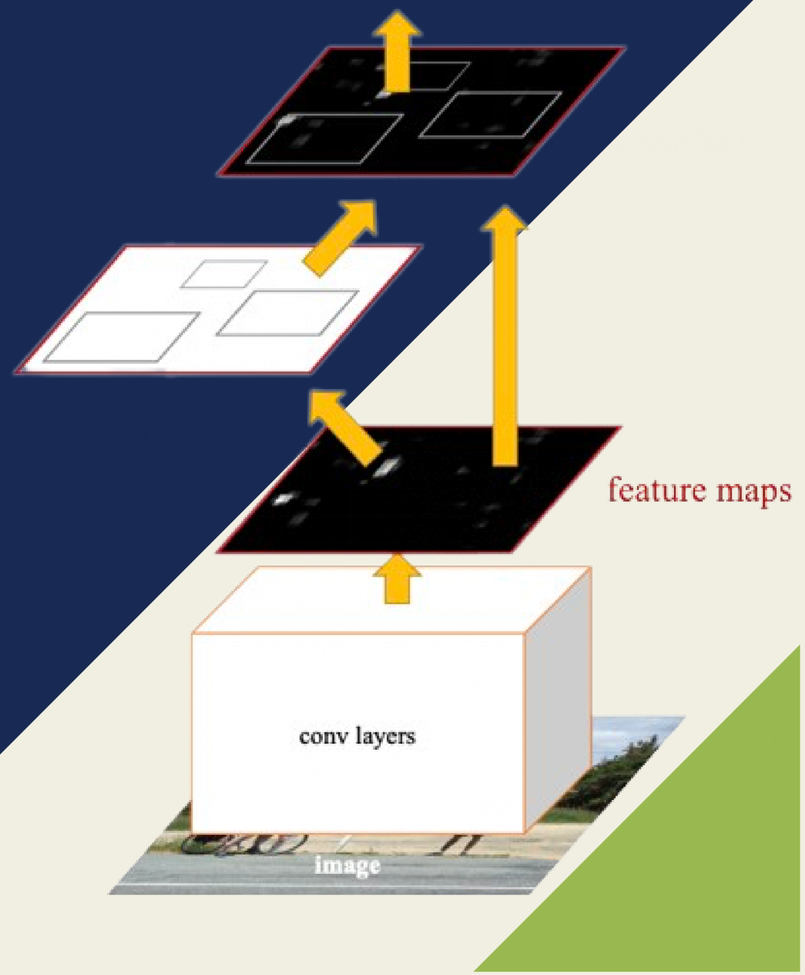
# Methodology

# YOLO v5




A one stage state-of-the-art model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility:

- It takes an input image and processes it through an input layer before passing it to the backbone for feature extraction.
- Produce three final feature maps for detecting small, medium, and large objects in the image.
- Once the three feature maps are obtained, they are sent to the prediction head, generating a multi-dimensional array called BBoxes.
- To filter out irrelevant information in the BBoxes array, corresponding thresholds (confthreshold, objthreshold) are set.




# Faster R-CNN

A modern two-stage object object detector that consists in:

- A region proposal algorithm to generate “bounding boxes” or locations of possible objects in the image
- A feature generation stage to obtain features of these objects, usually using a CNN
- A classification layer to predict which class this object belongs to
- A regression layer to make the coordinates of the object bounding box more precise

# Experiments and Results

# Dataset choice

The dataset is taken from roboflow and comprises a total of 338 plastic bottle images, each annotated using the YOLOv5 standard:

The specification for each line is as follows.

- One row per object
- Each row is class x_center y_center width height format.
- Box coordinates must be normalized by the dimensions of the image (i.e. have values between 0 and 1)
- Class numbers are zero-indexed (start from 0).

# Data Split

For training and evaluation purposes, we have three distinct subsets, we actually used the first and the third ones:.

## TRAINING SET

Comprising 88% of the total data, the training set was used to train our object detection model. It contains a diverse array of images, ensuring that our model learns to detect plastic bottles across different contexts.

## VALIDATION SET

Consisting of 8% of the data, the validation set serves as a crucial tool for hyperparameter tuning and model evaluation during the training process.

## TEST SET

This set, accounting for 4% of the dataset, was kept separate and untouched during model development. It was used exclusively to assess the model's generalization and performance on unseen data.

# Metrics for YOLOv5

The F1 curve we report is produced by calculating, for every threshold, precision, recall and F1 value and plot it

## MEAN AVERAGE PRECISION

It’s calculated using the mean of the Average Precision calculated for every class summed.
The mAP compares the ground-truth bounding box to the detected box and returns a score.

## RECALL

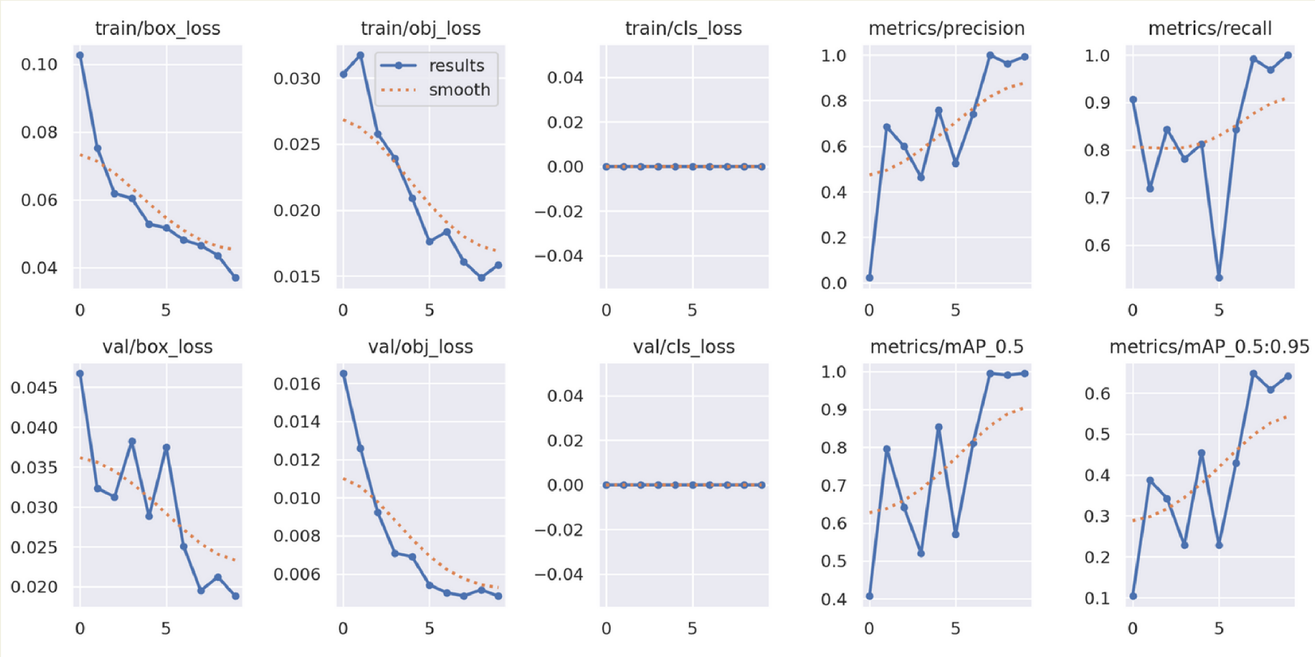
The recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect Positive samples.

## PRECISION

The precision is calculated as the ratio between the number of Positive samples correctly classified to the total number of samples classified as Positive (either correctly or incorrectly). The precision measures the model's accuracy in classifying a sample as positive.

# YOLOv5 - pretrained

YOLOv5's first experiment was done using a pre-trained YOLO model, specifically from pre-trained checkpoint YOLOv5m, the medium-sized network, which has very high performances.
On this base, we then create a custom model to detect the objects of our dataset and deploy it into the wild to make predictions. The custom model is trained for 10 epochs on batches of 16 images.




**10**

Epochs

**16**

Batch size

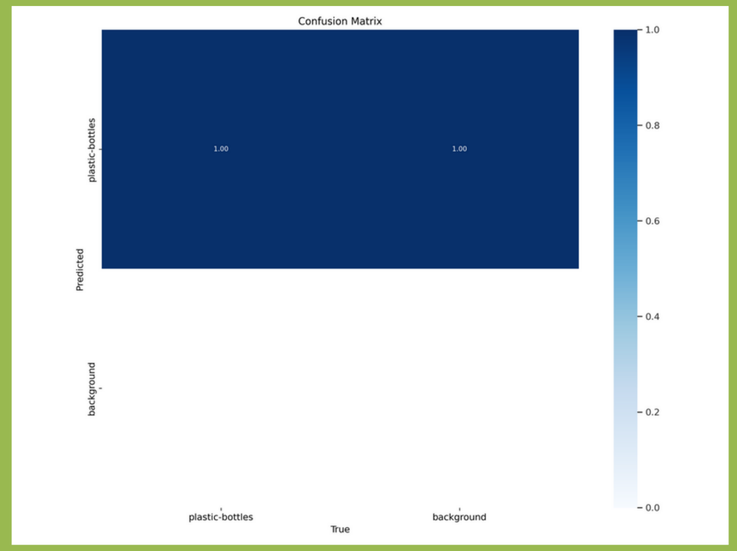
# Results

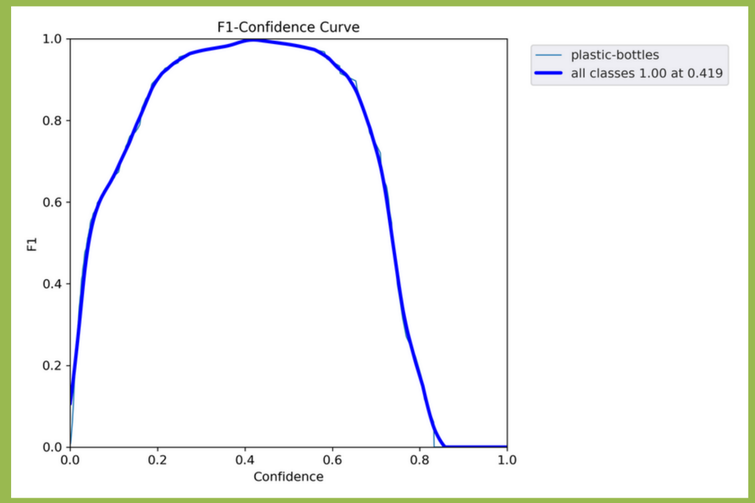
1. **MAP 0.5**
   (0.4071, 0.995)

2. **MAP 0.5:0.95**
   (0.1038, 0.6481)

3. **PRECISION**
   (0.0226, 1.0)

4. **RECALL**
   (0.5312, 1.0)



# Example

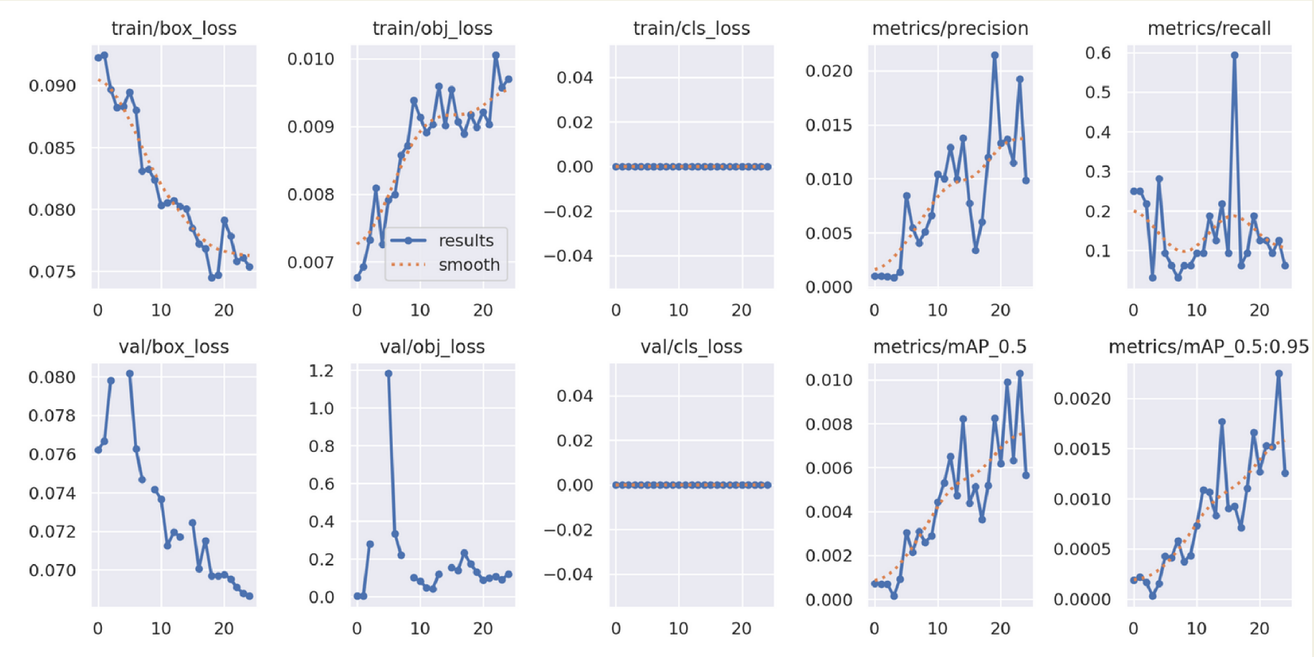
Example of predictions made on a batch of example images.
Every image in a test set was correctly predicted, except for this one:


# YOLOv5 - from scratch

We loaded the model from Pytorch Hub to train from scratch, with no pretraining or autoshape. We tried two different approaches:

1. The first approach consisted of using the same script, as Ultralytics suggested, but eliminating the pre-trained weights.
2. The second approach was to use a newly written training script.

Using the same layout and characteristics we obtained comparable results, but we chose to report the results from the original Ultralytics script since they had more comprehensive charts and information about results.




**25**

Epochs

**5**

Batch size

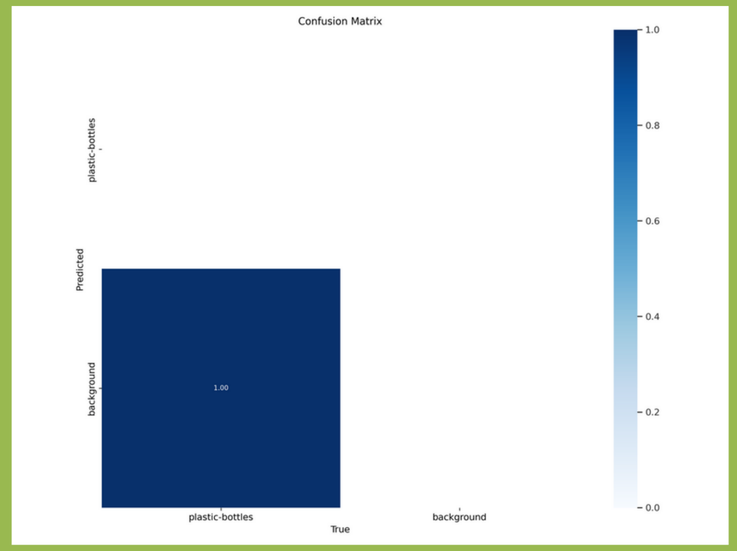
# Results

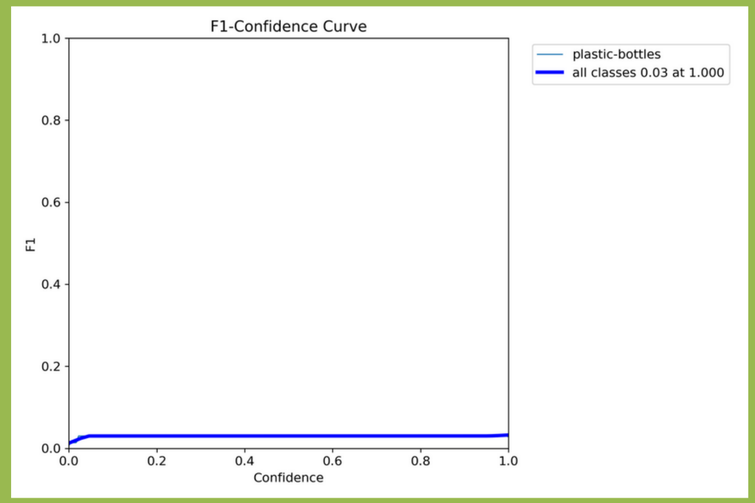
1. **MAP 0.5**
   (0.0001, 0.0103)

2. **MAP 0.5:0.95**
   (2.9848, 0.0022)

3. **PRECISION**
   (0.0008, 0.0214)

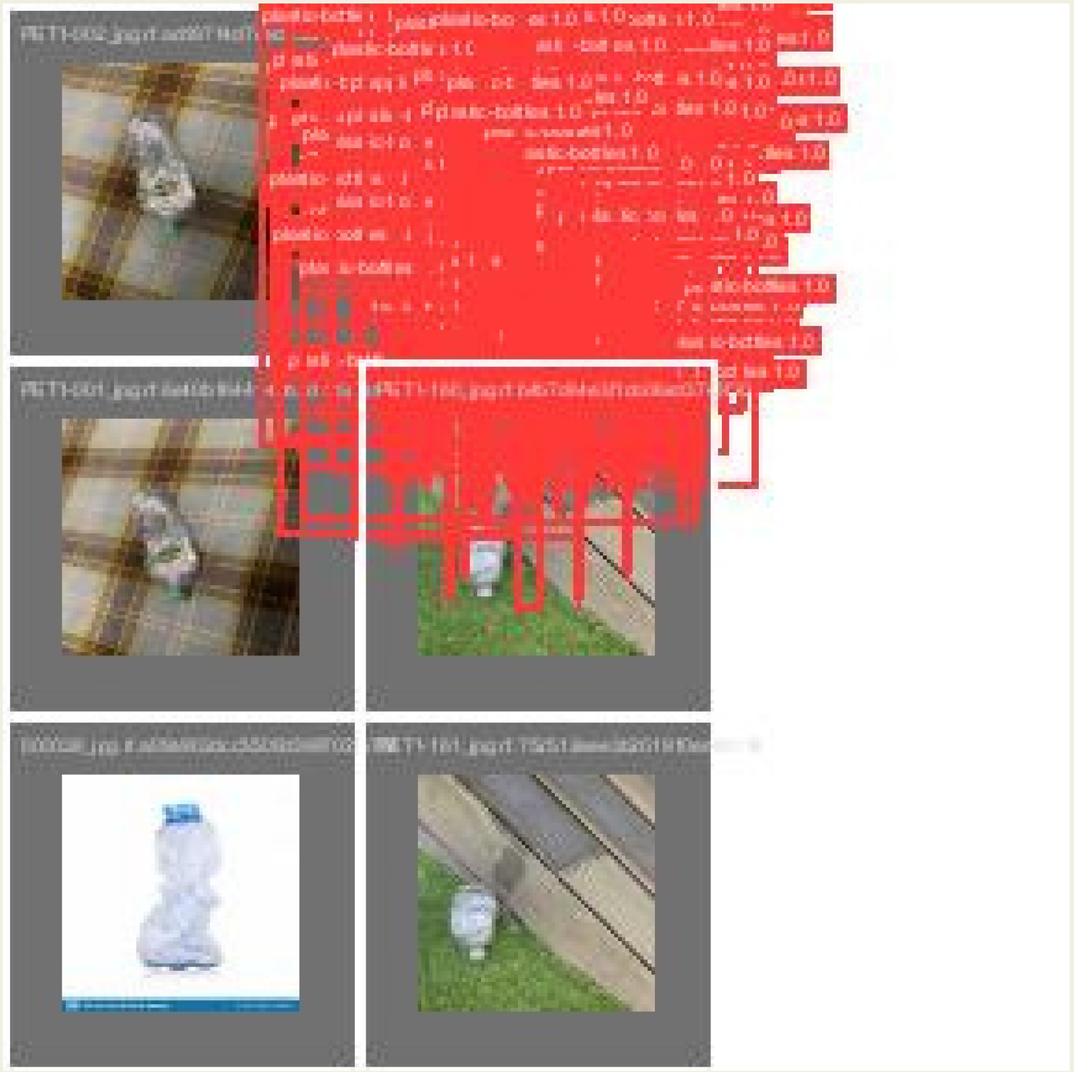
4. **RECALL**
   (0.0312, 0.5937)



# Example

Example of predictions made on a batch of example images.
Every image in a test set was wrongly predicted.
This was exactly what was expected. We have to consider that, to train from scratch, YOLO expects a very large dataset, trained for a long time, with at least 1500 images per class recommended and more than 10,000 instances (labeled objects) per class.



# Metrics for Faster RCNN

### AVERAGE PRECISION

It’s calculated by taking the area under the Precision-Recall curve, which represents how the precision and recall change as the detection threshold is varied.

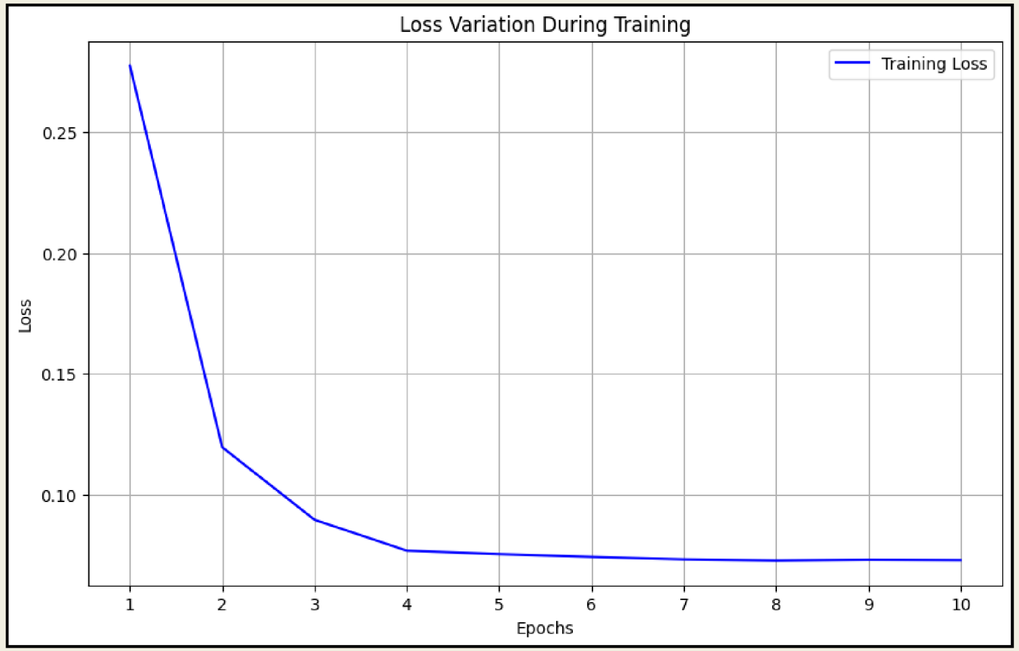
### AVERAGE RECALL

It's calculated by taking the average recall over different levels of confidence scores.

**Both AP and AR consider different IoU thresholds and areas**

# Faster RCNN - pretrained

The model initialization gives the user the possibility to load a pretrained network, obtained by training the net on the COCO dataset, a large-scale image recognition dataset for object detection




**10**

Epochs

**3**

Batch size

**SGD**

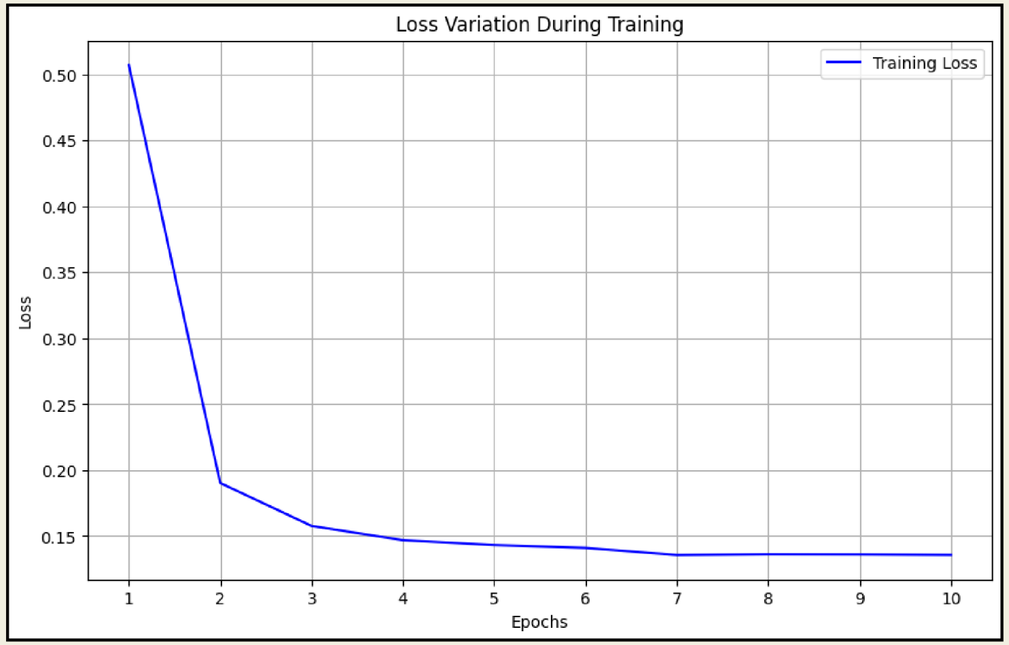
Optimizer

# Results

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.741
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.667
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.758
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.769
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.769
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.769
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.700
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.782
```

0.22 s per image

# Faster RCNN - from scratch




**10**

Epochs

**3**

Batch size

**SGD**

Optimizer

# Results

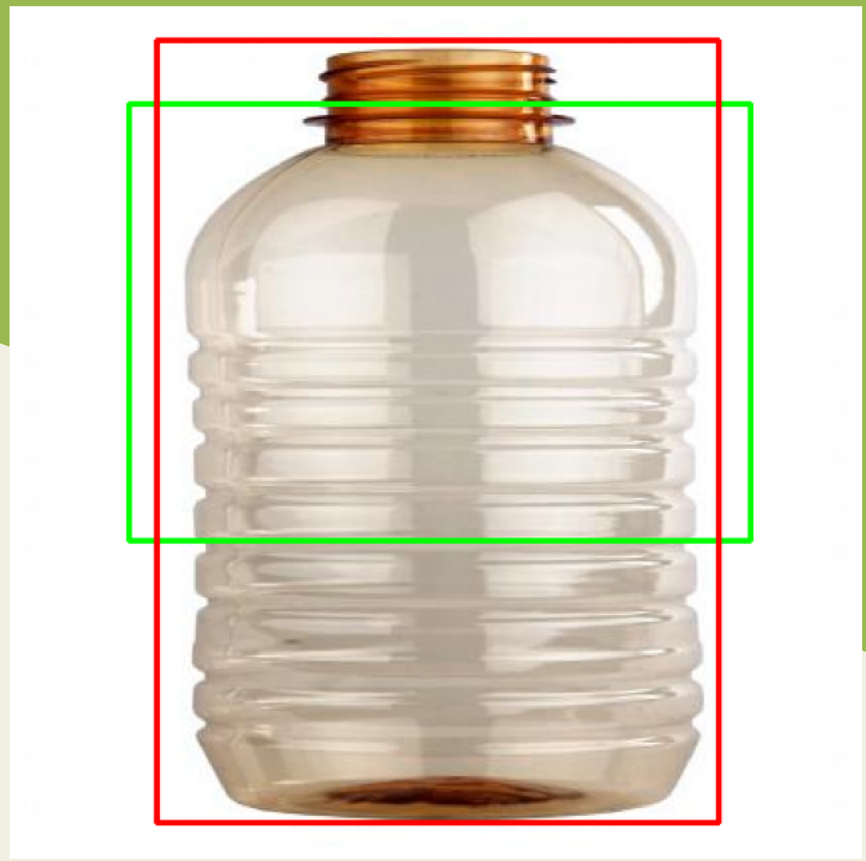
```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.558
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.946
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.673
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.270
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.613
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.531
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.623
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.623
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.400
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.664
```

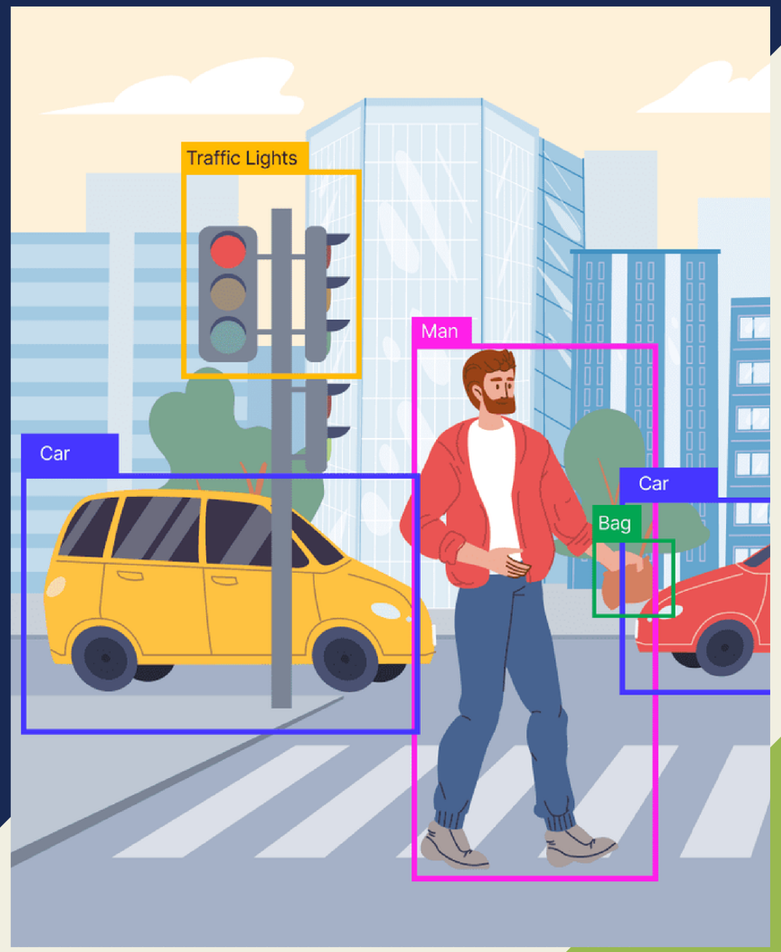
0.22 s per image

# Example

Here’s an example of a wrong  prediction of our model, obtained as an output of the non pre-trained version of Faster RCNN:

- Green: Modified bounding box
- Red: Ground truth boundingbox






# Conclusions

Our findings from comparing the two neural networks partially corroborate our initial assumptions:

- **Faster-RCNN Dominates**: Faster-RCNN tends to yield superior results overall, particularly when evaluating the performance of non-pre-trained networks.
- **YOLO's Speed Advantage**: Notably, YOLO demonstrates its speed advantage, with an inference time of under 2 milliseconds per image, compared to Faster-RCNN's 0.22 seconds per image.
- **Further Analysis Needed**: To gain a comprehensive understanding of YOLO's performance and to optimize both networks for our specific application, further analysis and fine-tuning are essential.