University of Verona

Master Degree in Artificial Intelligence

Computer Vision and Deep Learning

Comparison between Faster RCNN and YOLOv5

Authors:

Tommaso Del Prete
VR488382

Chiara Solito
VR487795

Professors:

Vittorio Murino

September 2023

Contents

1 Introduction

1.1 Context

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.

The goal of object detection is to not only recognize the presence of objects but also provide their precise spatial location through bounding boxes or pixel-level segmentation. This enables a more detailed understanding of the visual content and various applications such as autonomous driving, surveillance systems, image retrieval, robotics, and augmented reality use this method.

Traditional approaches to object detection relied on handcrafted features and classifiers, but recent advancements in deep learning have revolutionized the field. Deep learning-based object detection methods, such as the popular Faster R-CNN, SSD, and YOLO, leverage convolutional neural networks (CNNs) to extract hierarchical representations from input images and predict object classes and locations.

1.2 Motivation and Rationale

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 50). **This evaluation will be carried out on a novel and less-explored dataset.**

Over the past ten years, there has been remarkable advancement in object detection using deep learning techniques.[1] Nevertheless, these approaches often rely on extensive training data, making it challenging to employ them in real-world situations where unfamiliar objects are involved, not included in commonly used object detection datasets.

The process of annotating a substantial number of images for object detection is both expensive and laborious. In certain instances, such as medical applications, obtaining abundant images may even be unfeasible, we will try to understand the limitations of the two models on a small dataset, experimenting with both pre-trained and non pre-trained networks.

1.3 State of the Art

To present the state-of-the-art in object detection we refer to a recent survey [2]. The authors conduct a comprehensive study on real-time object detection. The experiments are done on multiple detectors and on diverse datasets, along with the genetic object detection benchmarks. 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.

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

1.3.1 Two-stage detection

Two-stage detectors in object detection employ a separate Region Proposal Network (RPN) in addition to classification. The RPN extracts features from proposed Regions of Interest (ROIs) and passes them to the classification head for determining class labels and the regression head for determining bounding boxes.[2]

The Region-based Convolutional Neural Network (RCNN) employs a selective search algorithm to identify potential object regions in an image. These selected candidates are then fed through CNN. The features extracted from the CNN serve as inputs to a Support Vector Machine (SVM) for region classification and a regressor for bounding box prediction. RCNN requires progressive multi-stage training and is relatively slow. To address these limitations, Fast-RCNN introduced specific modifications, such as splitting the loss functions into classification (Cls) and localization (Loc) losses. In the realm of convolution layer types, various options are utilized, including vanilla 2D convolution (V), deformable convolution (DCN), bilinear upsampling (UP), depth-wise separable convolution (DWS), and transposed convolution (T). [2]

1.3.2 One-stage detection

One-stage detectors are characterized by a unified feedforward network that handles classification and regression tasks in a single pass. These detectors do not involve a separate stage for generating proposals; instead, they consider all positions in the image as potential proposals. Each position is utilized to predict class probabilities, bounding box locations, and confidence scores, which indicate the network's confidence in its class prediction.[2]

The one-stage detectors can be categorized into anchor-based and keypoint-based detectors. Anchor-based detectors employ predefined anchor boxes or priors to aid in the prediction process. Prominent examples of anchor-based detectors include You Only Look Once (YOLO) by Redmon et al. (2016) and Redmon & Farhadi (2017, 2018), as well as Single Shot Detector (SSD) by Liu et al. (2016).[2]
Keypoint-based methods treat objects as points rather than modeling them as bounding boxes. Keypoints, such as corners or object centers, are estimated, and the width and height of the objects are regressed from these points, eliminating the need for predetermined anchors. Several keypoint-based networks have been introduced, including CornerNet, CenterNet, FCOS, NanoDet, and TTFNet (Law & Deng, 2018; Zhou et al., 2019a; Tian et al., 2019; Lyu, 2020; Liu et al., 2020).[2]

2 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.

2.1 Comparison between YOLO and Faster RCNN

YOLO (You Only Look Once) and Faster R-CNN are both popular object detection algorithms, but they differ in their approach and performance characteristics, as we briefly introduced in the previous section. Let's dive more profound into the matter.

YOLO takes a single-stage approach, directly predicting bounding boxes and class probabilities for grid cells in the input image. This allows YOLO to achieve real-time inference speed, making it suitable for applications requiring fast detection. However, YOLO may struggle with accurately localizing small objects and occasionally produces false positives or negatives.

In contrast, Faster R-CNN follows a two-stage approach. It uses a region proposal network (RPN) to generate potential bounding box proposals and then classifies and refines them for final detections. Faster R-CNN generally offers better accuracy and localization precision, particularly for small objects. **However, its two-stage architecture typically results in slower inference speed compared to YOLO.**

Regarding object classes, YOLO predicts classes globally for the entire image without considering object relationships, sometimes leading to diffi-

culties in distinguishing closely grouped objects of the same class. In contrast, Faster R-CNN predicts classes at the region proposal level, allowing it to consider context and handle complex scenes and overlapping objects more effectively.

Training complexity differs as well. YOLO is generally simpler to train, requiring fewer computational resources and training time. In contrast, Faster R-CNN has a more complex training process due to its two-stage architecture, often requiring more resources and training data.

For application scenarios, YOLO's real-time performance makes it suitable for video analysis, autonomous vehicles, and real-time surveillance systems. **Faster R-CNN's emphasis on accuracy and precise localization makes it well-suited for medical imaging, fine-grained object recognition, and detailed object analysis.**

We aim to show these differences, especially in terms of accuracy and speed, providing two examples of detection, with Faster R-CNN with ResNet50 + FPN and YOLOv5.

3 Methodology

3.1 YOLO v5

3.1.1 The architecture

Ultralytics YOLOv5 is a cutting-edge, 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. [4]

The YOLOv5 algorithm follows the same design concept as previous versions of YOLO. It takes an input image and processes it through an input layer before passing it to the backbone (Figure 1) for feature extraction. The backbone generates feature maps of various sizes, which are then combined using the feature fusion network (neck - Figure 2) to produce three final feature maps: P3, P4, and P5. These feature maps, represented by dimensions 80×80, 40×40, and 20×20 in YOLOv5, are responsible for detecting small, medium, and large objects in the image, respectively.[3]

Once the three feature maps are obtained, they are sent to the prediction head (head - also in Figure 2). At the prediction head, each pixel in the feature maps undergoes confidence calculation and bounding-box regression using predetermined prior anchors. This process generates a multi-dimensional array called BBoxes, which contains information such as object class, class confidence, box coordinates, width, and height.

To filter out irrelevant information in the BBoxes array, corresponding thresholds (confthreshold, objthreshold) are set. The resulting output represents the detection results.
The process of converting the input image into the BBoxes array is known as the inference process (Figure 3), while the subsequent thresholding and NMS operations are referred to as post-processing. It's important to note that the post-processing stage does not involve altering the network structure itself.[3]

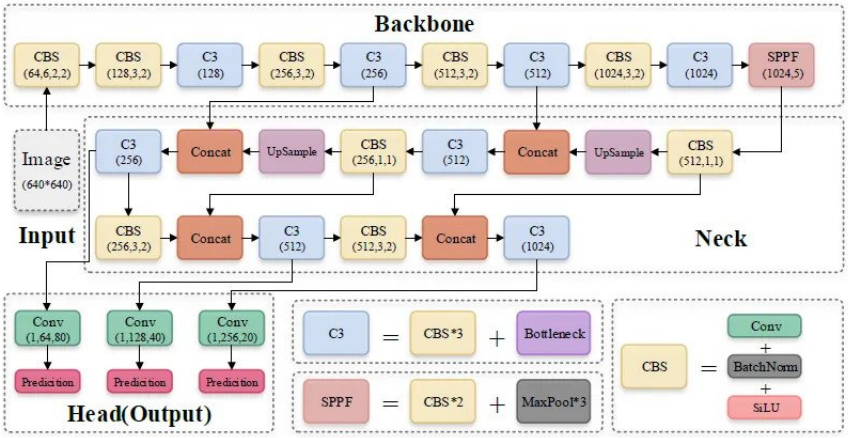


Figure 1: Backbone of Yolov5

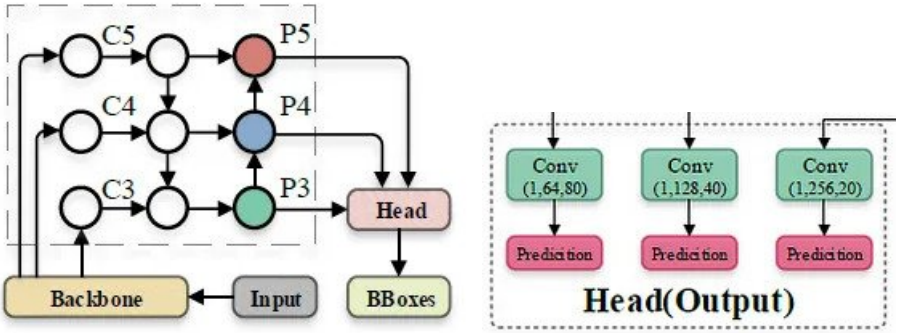


Figure 2: Neck and Head of Yolov5

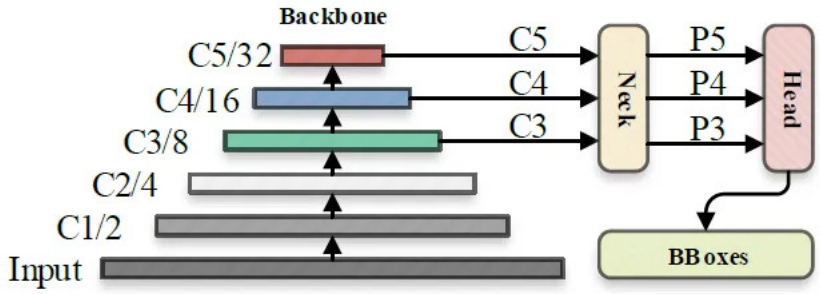


Figure 1. The default inference flowchart of YOLOv5.

Figure 3: Inference Process of Yolov5

3.1.2 Loss Functions

The loss function employed in object detectors comprises two distinct components: **Bounding Box Regression**, which evaluates the accuracy of predicted bounding boxes in capturing the ground truth bounding boxes, and **Cross Entropy Loss**, which assesses the detector's ability to correctly predict the class.
In this case, the authors implemented a IoU Loss for the Bounding Box Regression and two Focal Losses that use a Binary Cross Entropy With Logits Loss for objectness and class scores.

IoU Loss The IoU loss function quantifies the spatial overlap between predicted and ground truth bounding boxes. IoU, short for intersection over union, is calculated as the ratio of the intersection area to the union area of the two bounding boxes. This loss function is employed to optimize the coordinates of the bounding boxes and offers the benefit of aligning closely with the evaluation metric commonly used in object detection, which is typically based on IoU.

$$\mathcal{L}_{loss} = 1 - \frac{\text{area of intersection}}{\text{union area}}$$

However, it's important to note that IoU loss can be susceptible to instability and sensitivity to outliers. This is because it can exhibit significant gradients when the overlap between the predicted and ground truth bounding boxes is small or zero.

Focal Loss The Focal Loss is designed to address the one-stage object detection scenario in which there is an extreme imbalance between foreground and background classes during training (e.g., 1:1000). [7]

$$FL(p_t) = -(1 - p_t)^{\gamma} log(p_t)$$

Focal loss applies a modulating term to the cross entropy loss in order to focus learning on hard misclassified examples. It is a dynamically scaled cross entropy loss, where the scaling factor decays to zero as confidence in the correct class increases.

Binary Cross Entropy with Logits Loss This loss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability. [6]

$$\ell(x, y) + L + \{l_1, \ldots, l_N\}^T, l_n = -w_n[y_n \cdot log\sigma(x_n) + (1 - y_n) \cdot log(1 - \sigma(x_n))]$$

3.1.3 Implementation and training script

To implement and train YOLO, we followed the guidelines provided by Ultralytics, which involved the following steps:

- Cloning the Yolov5 repository from Ultralytics' GitHub
- Prepare the custom dataset
- Execute the training script

The results of this implementation can be found in section 4 (Experiments and Results). This approach aimed to maximize the model's potential and enable a comparison between the two architectures. Various experiments were conducted using both pre-trained weights and randomly initialized weights.

Architecture Implementation As previously said, the implementation used for these experiments is the original one from Ultralytics: the input is expected to be a list of tensors (images), while labels are loaded directly from the cache of the dataset split. The targets (`tcls`, `tbox`, `indices`, and `anchors`) are built as tensors (to compute the loss, which expects input:

targets(image, class, x, y, w, h)), loading the information from the `.yaml` file. During inference instead, the model only requires input images (processed as tesnsors) and returns post-processed predictions, directly saved.

3.2 Faster RCNN

3.2.1 The architecture

The architecture used in this project is directly provided by the official PyTorch documentation, more precisely, the Faster R-CNN (Region-based Convolutional Neural Network) with ResNet-50 is a popular and powerful object detection model that combines the Faster R-CNN architecture with a ResNet-50 backbone network, as described in the paper "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"[5].

FPN (Feature Pyramid Network) is often combined with the ResNet-50 architecture to create a more powerful and effective object detection model. ResNet-50 serves as the backbone network for feature extraction, while FPN enhances the multi-scale capabilities of the features extracted by ResNet-50.

The input to the model is expected to be a list of tensors, each of shape [C, H, W], one for each image, and should be in 0-1 range (different images can have different sizes).

The model's behavior varies depending on whether it is in training or evaluation mode.

- During training, the model expects input tensors and targets, which are a list of dictionaries. The targets include the ground-truth boxes in [x1, y1, x2, y2] format (where x1 and y1 are the top-left coordinates, and x2 and y2 are the bottom-right coordinates), along with the corresponding class labels. The model provides a dictionary of tensors as output during training, containing the classification and regression losses for both the RPN (Region Proposal Network) and the R-CNN (Region-based Convolutional Neural Network).

- During inference or evaluation, the model only requires input tensors and returns post-processed predictions as a list of dictionaries. Each dictionary in the list corresponds to an input image and contains the following fields:

  - **boxes**: The predicted bounding boxes in [x1, y1, x2, y2] format, representing the coordinates of the predicted boxes. The values satisfy the conditions 0 ¡= x1 ¡ x2 ¡= W and 0 ¡= y1 ¡ y2 ¡= H, where W and H are the width and height of the image.

- **labels**: The predicted class labels for each detection, indicating the predicted object class.
- **scores**: The scores or confidence levels associated with each detection, representing the model's confidence in its predictions. The higher the score, the more confident the model is about the detection.

The model builder also accepts pre-trained weights.

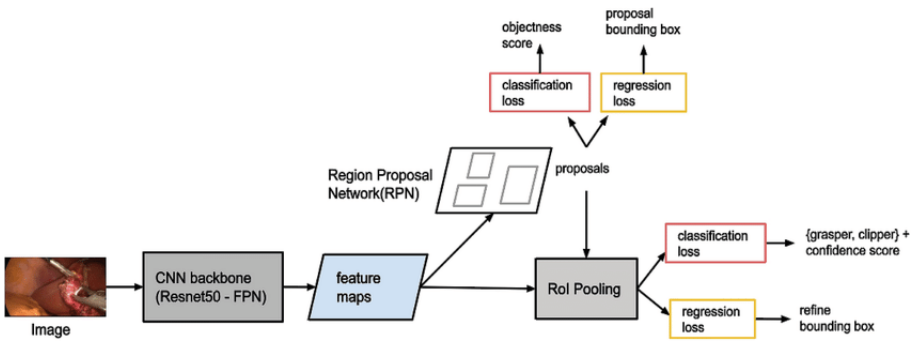


Figure 4: Structure of the FasterRCNN with Resnet50+FPN as backbone.

3.2.2 Loss functions

FRCNN uses two different losses, one to classify the detected object and another to adjust the proposals given by RPN.

The classification and regression losses are typically combined into a joint loss, which is used to train the Faster R-CNN model with ResNet-50. During training, the model aims to minimize this combined loss by adjusting its parameters through backpropagation and gradient descent optimization techniques. By optimizing these loss functions, the model learns to accurately classify objects and regress their bounding box coordinates, resulting in improved object detection performance.

Classification Loss The classification loss is responsible for ensuring that the model accurately predicts the presence or absence of objects and assigns them the correct class labels. It measures the dissimilarity between the predicted class probabilities and the ground truth class labels. The most commonly used loss function for classification in Faster R-CNN is the softmax cross-entropy loss.

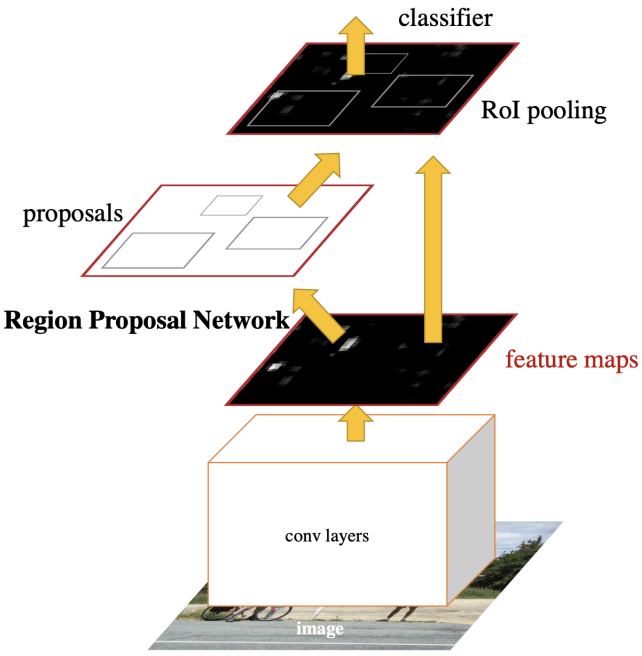


Figure 5: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

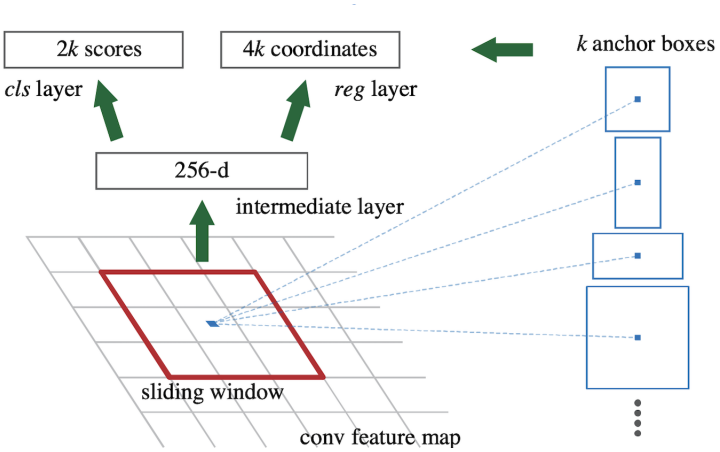


Figure 6: Region Proposal Network (RPN)

During training, for each region of interest (RoI), the model predicts the probabilities of different object classes using a classification head. These predicted probabilities are then compared with the ground truth class labels associated with the RoIs. The softmax cross-entropy loss computes the difference between the predicted probabilities and the true class labels, encouraging the model to correctly classify objects. The goal is to minimize this classification loss, thereby improving the model's ability to identify object classes accurately.

Regression Loss The regression loss is responsible for enabling the model to accurately localize objects by refining the predicted bounding box coordinates. It measures the discrepancy between the predicted bounding box coordinates and the ground truth bounding box coordinates. The most commonly used regression loss function in Faster R-CNN is the smooth L1 loss.

During training, for each RoI, the model predicts the coordinates of the bounding box that tightly encloses the object. These predicted coordinates are then compared with the ground truth bounding box coordinates associated with the RoIs. The smooth L1 loss function computes the error between the predicted and ground truth bounding box coordinates. It uses a combination of L1 and L2 loss functions, which makes it less sensitive to outliers compared to the traditional L1 loss. The objective is to minimize this regression loss, enabling the model to accurately localize objects by adjusting the predicted bounding box coordinates.

3.2.3 Implementation and training script

The implementation follows the one provided during the lab session on "Object Detection" using FasterRCNN.

Firstly we imported the model from torchvision.models.detection and then set the right number of features and classes to predict.

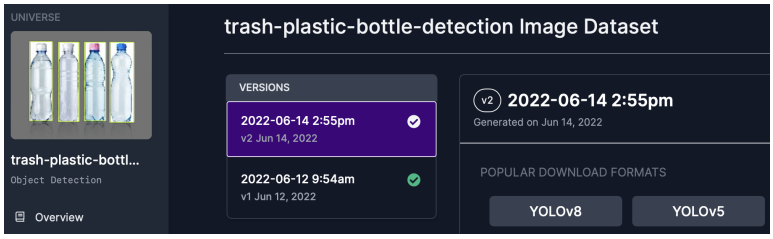
Then we processed the data to ensure the right format to give to the architecture.

4 Experiments and Results

4.1 Dataset Choice

This dataset holds significant value for individuals engaged in object detection, particularly those concerned with combating plastic waste and pol-

lution. By offering a diverse collection of annotated images, this dataset enables the development of accurate and efficient models capable of identifying plastic bottles in real-world environments.



The dataset is taken from roboflow and comprises a total of 338 images, each meticulously annotated using the YOLOv5 standard. This annotation format is renowned for its accuracy and compatibility with state-of-the-art object detection models. The use of YOLOv5 annotations ensures that the dataset is well-prepared for training and evaluation, allowing for seamless integration with cutting-edge object detection algorithms.

Data Split For training and evaluation purposes, we have three distinct subsets:

- **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 served 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.

YOLOv5 uses all the splits, including the Validation Set, but for Faster RCNN we did not validate.

4.2 YOLO v5

4.2.1 Initialized Weights

YOLOv5 first experiment was done using a pre-trained YOLO model, specifically from pre-trained checkpoint YOLOv5m, the medium-size network,

which has very high performances, as described in Table 1. The models are all compound-scaled variants[1] of the same architecture, following the EfficientDet compound scaling model, minus the image scaling.

size (pixels)	640
mAPval 50-95	45.4
mAPval 50	64.1
Speed CPU b1 (ms)	224
Speed V100 b1 (ms)	8.2
Speed V100 b32 (ms)	1.7
params (M)	21.2
FLOPs @640 (B)	49.0

Table 1: YOLO v5m performances

Notes on checkpoint

- The used checkpoint is trained to 300 epochs with default settings.
- mAPval values are for single-model single-scale on the COCO val2017 dataset.
- Speed averaged over COCO val images using an AWS p3.2xlarge instance.

On this base, we then create a custom model to detect the objects of our dataset and deploy it to make predictions. The custom model is trained for 10 epochs on batches of 16 images.

Metrics

- **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.**

[1]The compound scaling method is based on the idea of balancing dimensions of width, depth, and resolution by scaling with a constant ratio. This allows to maintain the properties that the model had at the initial design and thus maintain the optimal structure

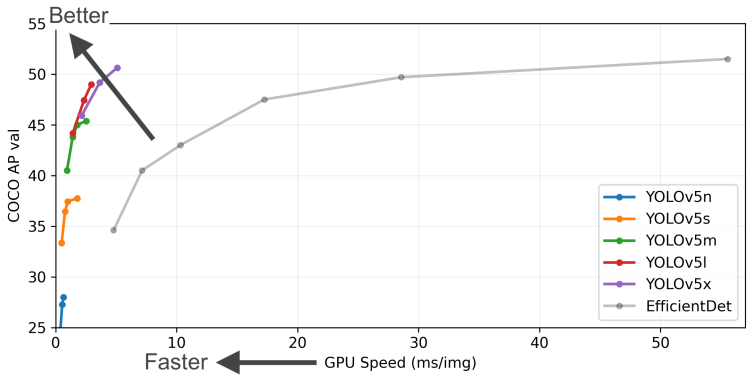


Figure 7: Evaluation of Yolov5 Checkpoints
COCO AP val denotes mAP@0.5:0.95 metric and GPU Speed measures average inference time per image on COCO val2017.

- **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.**

- **Mean Average Precision**: It's calculated using the mean of the Average Precision calculated for every class summed [2] **The mAP compares the ground-truth bounding box to the detected box and returns a score.**

Results file `results.csv` is updated after each epoch during training and then plotted as `results.png`, reported here in Figure 8. Note that class loss is zero because the task is single class.

The results of the Experiment with pre-trained weights, on the Validation Set, can be found in Table 2.

We can also take a look at the quantitative results: in Figure 9 we have the Confusion Matrix and F1 curve, while in Figure 10 we have the Precision and Recall Curves. All results come from the Validation.

[2] Average precision computes the average precision value for recall value over 0 to 1.

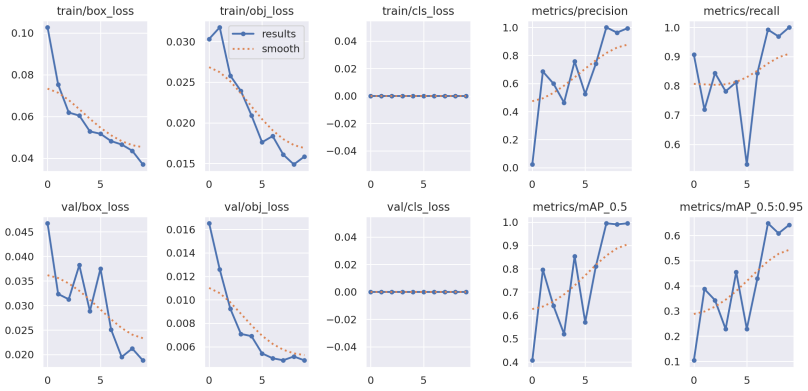


Figure 8: Metrics of YOLOv5 with pre-initialized weights

Metric	Value
Class	All (1)
Images	26
Instances	32
Precision	1
Recall	0.992
mAP50	0.995
mAP50-95	0.648

Table 2: Validation Metrics - YOLO with pre-trained weights

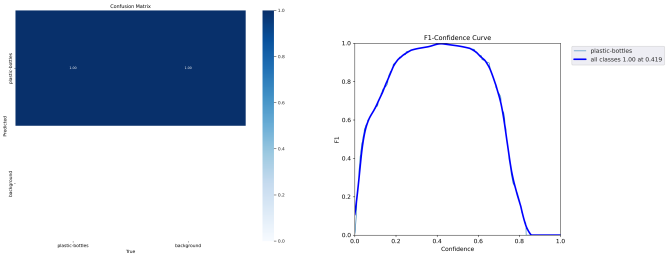


Figure 9: Confusion Matrix and F1 Curve of YOLOv5 with pre-initialized weights

Practical Examples Let's take for instance the Validation Batch represented in Figure 11. The predictions of the model are very good, even if we notice a mistake: a bottle is recognized two times.

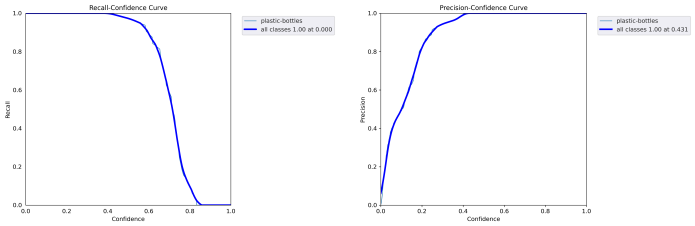


Figure 10: Recall and Precision Curve of YOLOv5 with pre-initialized weights

4.2.2 Testing

The test set was composed of 13 images. The experiment was run with speed: 0.7ms pre-process, 26.9ms inference, 9.8ms NMS per image at shape (1, 3, 640, 640).
Every image in the test set is correctly recognized, with the exception of the image in Figure 12:

4.2.3 Training from scratch

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

1. The first approach consisted of using the same script used before, as suggested by Ultralytics, 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.

The metrics used are indeed the same as before, the results for the experiment can be found graphically in Figure 13 and metrics from the validation step in Table 3. We train with a batch size of 5 images, for 25 epochs.

In Figure 14 we reported the Confusion Matrix and F1 curve, while in Figure 15 we have the Precision and Recall Curves.

Practical Examples Let's take again, for instance, one of the Validation Batches, represented in Figure 16. The prediction of the model is terrible!

Figure 11: Labeled and Predicted Batch Images

Considerations As we can see the performance of YOLOv5 is really poor without pre-initialized weight: this was exactly what was expected. In fact, we have to consider that, to train from scratch, YOLO expects a very large dataset, trained for a long time:

- **Images per class**: $\geq$ 1500 images per class recommended
- **Instances per class**: $\geq$ 10000 instances (labeled objects) per class recommended

Figure 12: Uncorrectly recognized image

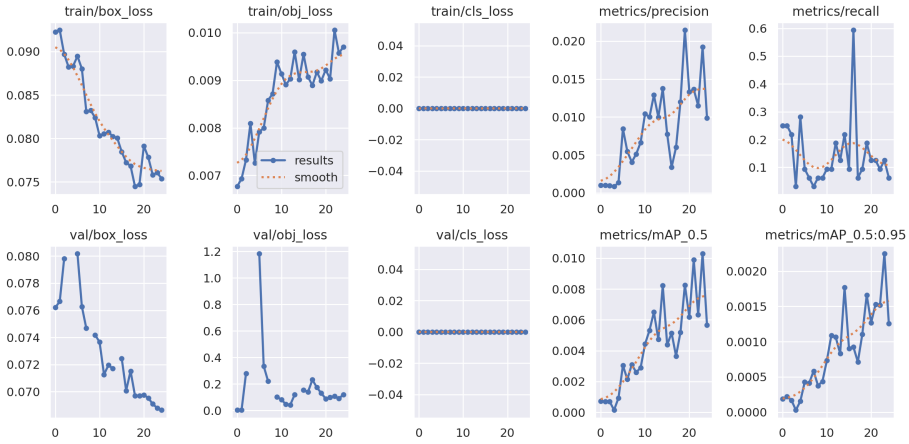


Figure 13: Metrics of YOLOv5 with non-initialized weights

Metric	Value
Class	All (1)
Images	26
Instances	32
Precision	0.000801
Recall	0.0625
mAP50	0.000504
mAP50-95	9.62e-05

Table 3: Metrics of evaluation - YOLO trained from scratch

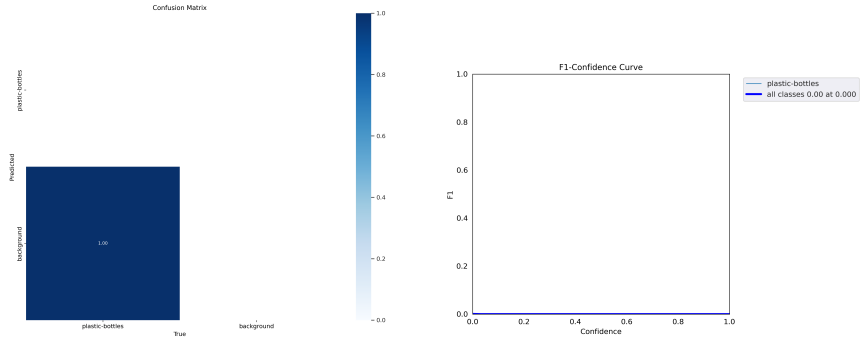


Figure 14: Confusion Matrix and F1 Curve of YOLOv5 with non-initialized weights

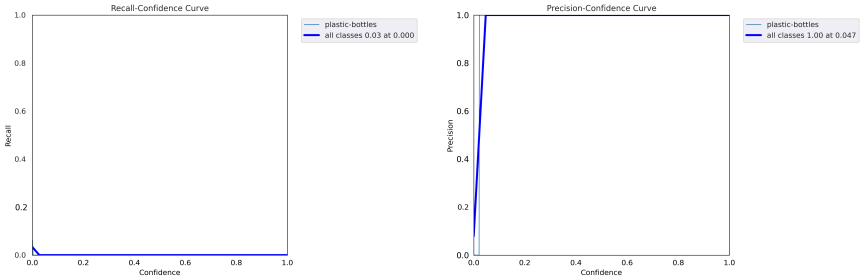


Figure 15: Recall and Precision Curve of YOLOv5 with pre-initialized weights

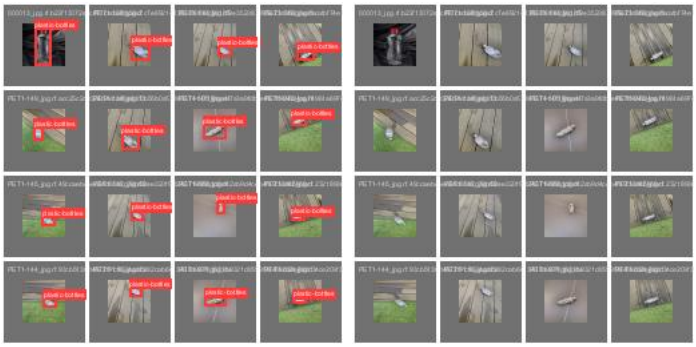


Figure 16: Labeled and Predicted Validation Batch Images

- **Image variety**: Must be representative of the deployed environment.

We also experimented with a higher number of epochs, but the model did not improve after 40 epochs. The results were the same as the ones reported in this report and also this was to be expected, given what we said in this paragraph.

4.3 Faster RCNN

4.3.1 Initialized Weights

The model initialization gives the user the possibility to load a pre-trained network instead of starting with random/zero weights.

More precisely, the default set of weights is obtained by training the net on the COCO dataset, a large-scale image recognition dataset for object detection, segmentation, and captioning tasks that contains over 330,000 images, each annotated with 80 object categories. The details about pre-trained weights are listed in Table 4.

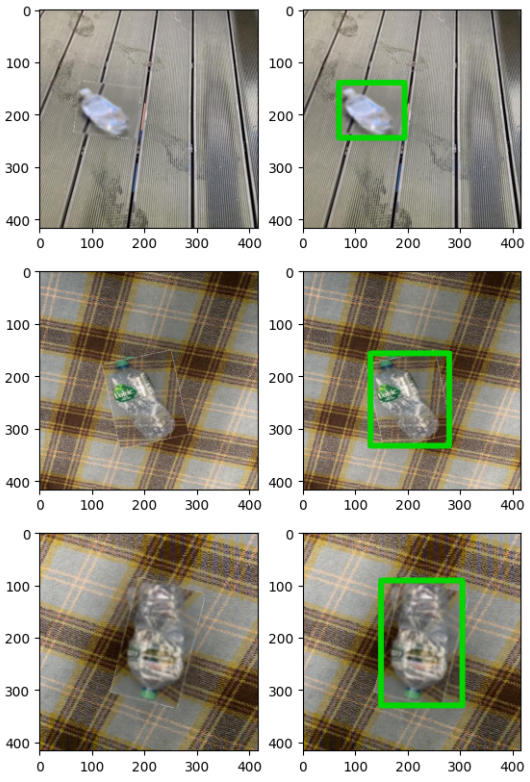


Figure 17: Example of labeled images with Faster RCNN.

BoxMap (on COCO-val2017)	46.7
Categories	88
MinSize	height=1, width=1
NumParams	43712278
GFLOPS	280.37
File size	167.1 MB

Table 4: Details about weights.COCOV1

Setting The optimizer is SGD (6Stochastic Gradient Descent), imported from pyTorch and with the following setup:

- lr=0.0005,
- momentum=0.9,
- weight decay=0.0005

To avoid vanishing gradient, which could become NaN during the training process, we used a batch size of 3 and a total of 10 epochs

```
IoU metric: bbox
 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
```

Figure 18: Metrics on our dataset with weights initialization

Results The results in Fig.18 show great performances:

- The Average Precision (AP) measures the accuracy of object detection by considering both precision and recall. Precision is the ratio of true positive detections to the total number of detections, while recall is the ratio of true positive detections to the total number of actual objects in the image. It is calculated by taking the area under the Precision-Recall curve, which represents how the precision and recall change as

the detection threshold is varied.
The final AP is 1.0 for all the images in the test set if we consider the boxes with an IoU of 0.5 or more. Worse results are obtained if we discriminate on the box areas and if the threshold is set higher than 0.75. That means the model is able to detect and localize all objects in our images accurately and without any false positives or false negatives when the threshold is not too strict. While the value -1.000 refers to the non-existing data since no box is small enough to compute the metric.

- The Average Recall (AR) is a metric that measures the ability of a model to recall a certain fraction of true positive detections. It's calculated by taking the average recall over different levels of confidence scores. In our case it's calculated by averaging the recall values at different IoU thresholds (0.50:0.95), different object sizes (small, medium, large), and different maximum detections per image (1, 10, 100). The results are not so great as AP but still good, with an average value of 0.769, so our model has a 76.9% chance of detecting real positives.

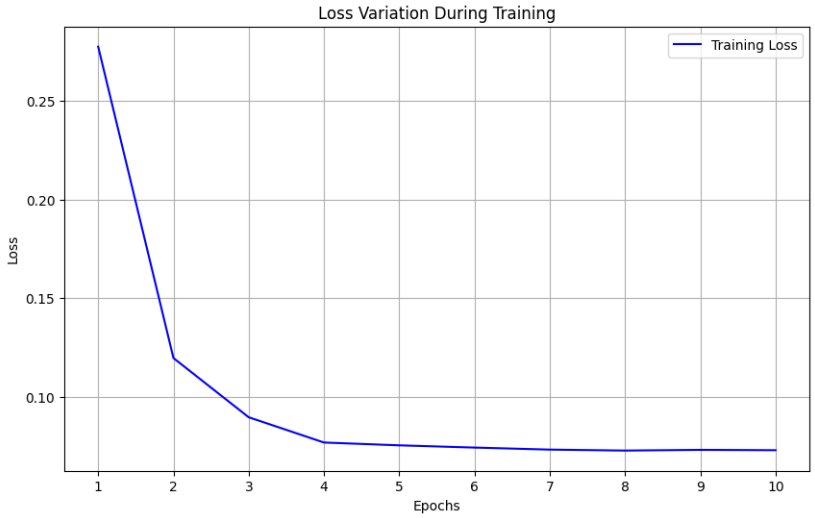


Figure 19: Loss variation during the training with pre-initialized weights. The convergence is near the value of 0.07, so better than training the model from scratch.

The loss variation can be seen in Fig. 19, the convergence value is near 0.07. Moreover, the time for inference is 0.22 seconds per image, hence not

as fast as YOLO!

4.3.2 Training from scratch

The network could also be trained without a weight initialization, obviously, we don't expect better results.

Setting The optimizer is the same of the previous experiment, and so are the number of epochs and the batch size.

```
IoU metric: bbox
 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
```

Figure 20: Metrics obtained after the evaluation on the Faster-RCNN trained from scratch.

Results From the results listed in Fig. 20 we can notice that:

- The Average Precision (AP) is worse than the previous one for every entry, but the results are robust even without the weight initialization (an example of a non perfect prediction is in Fig. 22)
- The Average Recall (AR) is as well not so good as the pretrained model, but the average is pretty decent (0.4 is the worst value).

The loss convergence is reached after 6 epochs and it's near the value of 0.10 (Fig. 21)

Practical Example In the Fig. 22 we can see an example of a predicted bounding box that doesn't perfectly fit the ground truth box, in this case, the model predicts a box that is smaller than the real one and so the result will be classified as positive or negative if we consider different thresholds.

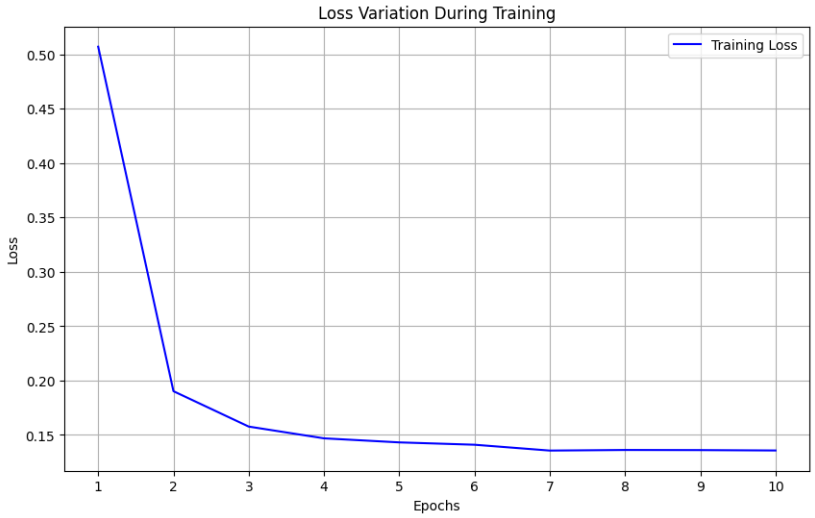


Figure 21: Loss variation during the training with non pre-initialized weights. The convergence is near the value of 0.10.

5 Conclusions

In summary, our findings from comparing the two neural networks partially corroborate our initial assumptions. Faster RCNN tends to yield superior results overall, particularly when we evaluate the performance of non-pre-trained networks. 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.

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.

Finally, it's interesting to notice that, even if these architectures are not usually written from scratch or evaluated without pre-trained weights, Faster RCNN is able to reach satisfying results nonetheless.

Figure 22: Example of a nonperfect bounding box, inferred by a non pre-trained Faster RCNN, in red the ground truth while in green the predicted bbox.

References

[1] Mona Köhler, Markus Eisenbach, Horst-Michael Gross (2022) *Few-Shot Object Detection: A Comprehensive Survey*, arXiv: 2112.11699

[2] Elahe Arani, Shruthi Gowda, Ratnajit Mukherjee, Omar Magdy, Senthilkumar Kathiresan, Bahram Zonooz (2022) *A Comprehensive Study of Real-Time Object Detection Networks Across Multiple Domains: A Survey*, Transactions on Machine Learning Research.

[3] Liu H., Sun F., Gu J., Deng L. (2022) *SF-YOLOv5: A Lightweight Small Object Detection Algorithm Based on Improved Feature Fusion Mode.* Sensors https://doi.org/10.3390/s22155817

[4] https://github.com/ultralytics/yolov5

[5] Shaoqing Ren and Kaiming He and Ross Girshick and Jian Sun (2016) *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*,

[6] https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

[7] T. -Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár (1 Feb. 2020), *Focal Loss for Dense Object Detection*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 2, pp. 318-327, doi: 10.1109/TPAMI.2018.2858826.