POLITECNICO DI TORINO

Master degree course in Data Science and Engineering

Master Degree Thesis

AI-powered Weed Identification in cultivated fields: an Object Detection approach

Supervisor
prof. Andrea Bottino

Candidate
Beatrice Macchia
matricola:292178

Internship Tutor
Data Reply's Manager Alessandro Piovano

Anno accademico 2022-2023

Summary

Over the last decades, Artificial Intelligence has attracted much attention and it is becoming increasingly common in our lives. AI is entering our physical spaces in the form of autonomous vehicles, drones or domestic devices. Among the different AI applications, in the last years part of the research community has focused on Computer Vision, a sub-field that teaches machines to analyze and extract relevant information from images and videos, performing multiple tasks such as classification, object recognition, mapping of the position and movement of human body joints. Thanks to the great improvements in storage systems, processing speeds, and analytic techniques, these types of algorithms are getting even more precise in analysis and decision-making.

Computer Vision includes multiple tasks, differing in complexity and goals. Among them, Object Detection works to identify and locate objects within an image or video by drawing bounding boxes around them. It can be useful for several real-life applications, from self-driving cars to medicine or agriculture.

Regarding the last one, weeds represent one of the most invasive issues affecting agricultural production. It is very difficult for the farmers to manually identify and pollute or extrapolate each weed singularly, hence chemical substances like herbicides are commonly sprayed all over the cultivated fields, resulting in massive waste and pollution of the farmland's ecological environment. With the continuous improvements in the agricultural production level, accurately distinguishing crops from weeds has become very important.

This thesis elaborates on the possibility of using object detection methods to solve weed extirpation problems. It provides an overview of various systems for weed detection and analyzes the advantages and disadvantages of state-of-the-art algorithms. In particular, two object detection algorithms will be further analyzed: YOLO (You Only Look Once) and Faster R-CNN. YOLO is a one-stage object detector, which means that its model architecture is built in order to directly predict objects' bounding boxes in the images. This leads to a simpler and faster architecture, although it can sometimes

lose some accuracy. Instead, Faster R-CNN is a two-stage detector: its structure is more complex as it is composed of a first stage, where the image is screened out and some regions of interest are generated and passed to the second stage, where these regions are filtered and refined.

In this project I deeply analyze the differences between the two detectors in terms of architecture and potential, then I test both models on agricultural data. I personally built the dataset using the images that the farmers sent us, which represent multiple typologies of cultivated fields, and I labeled them using the LabelImg tool. The data collection phase took a significantly long time since an unexpected drought led to the seedlings taking longer than usual to germinate, hence we had to wait. Additionally, the labeling process took me some weeks considering that images could be very rich in small crops and weeds presence: each of them needed to be bordered inside a specific bounding box. In the images the following elements were labeled: crops, which are the good plants; weeds, the infesting plants; roots of the weeds, where we want the pistons to specifically hit; rows, which are the lines of crops.

Once the dataset was set up, I could run the various experiments with YOLO and Faster R-CNN. The complete dataset has been split into three parts: training, validation and test sets. The two object detection models have been fine-tuned on the training set, their hyperparameters have been tuned and evaluated on the validation set, and the final performances have been assessed on the test set. The observed results show that YOLO is the most suitable detector for the purpose of this project: it outperforms Faster R-CNN especially in recognizing weeds and their roots, that is the main focus of this project. Therefore YOLO is chosen as a reference model to evaluate how the system would act in a real-life scenario. In particular I estimate the real dimension of the weeds represented in the images and analyze whether their root would be successfully extirpated by the mechanical piston, when the weed is recognized by the model but the root is not. For this purpose, I compared and measured the distance between the center of the weed bounding box and the actual root position. When the weed root is situated within the piston's range of action, then it would be successfully extirpated. As a result, we could observe that the vast majority of roots would be extrapolated.

Inference time has a key role as well in this project as the model is deployed in a real-time context: the final step is optimizing its speed in recognizing the elements of interest in the images. Unfortunately, the computer utilized by the cultivators is not provided with a powerful GPU and internet connection is often quite weak, then it is not possible to exploit online tools such as Google Colab. For this reason, I use OpenVINO (Open Visual Inference and Neural Network Optimization) and ONNX (Open Neural Network Exchange): OpenVINO is an open-source toolkit used to optimize models' inference on Intel products, while ONNX is usually used as an intermediate format to convert models to other frameworks, but is also able to reduce the time required by the model to perform its tasks. Even though ONNX and OpenVINO bring a moderate improvement in terms of inference time, they also lead to a significant accuracy loss. Once again, this proves that we cannot simply speed up the model while keeping the same performances: it is a trade-off that needs to be carefully assessed taking into consideration the context requirements. In this sense, the choice of the model needs to be balanced with respect to the speed reached by the tractor and the time required by the piston to be activated and hit the detected weeds.

This thesis demonstrates that an object detection approach for weed recognition in a real-time scenario is feasible and brings satisfactory results. The use of a larger dataset with images from various crop types could potentially lead to an improvement in the model performances, especially for what concerns generalizability. Currently the model has only been trained on specific and limited crop types, therefore a bigger and more diversified dataset would make it possible for the model to learn how to recognize new crops and weeds that it has not encountered before. Moreover, a bigger training dataset can enable the use of semantic segmentation techniques, leading to more accurate and detailed predictions. In conclusion, Computer Vision proves to be a promising tool in agriculture and, more in general, AI-powered solutions are able not only to help farmers produce more with fewer resources, but also to increase crop quality and reduce the impact of large-scale production on the environment.

Acknowledgements

Thank you Riccardo for being the best wingman I could ask for. This is a huge achievement for me and I will never thank you enough for being there during the whole path: it was fundamental.

Thank you to my parents and my grandparents: I am very proud of being the first graduate woman in my family, and it would have not been possible without your sacrifices. Thank you to my brother for being a reference point during my life.

Finally, thank you to my best friends for being an example to look up to, and for always supporting me.

Contents

List of Tables 9

List of Figures 11

1 Introduction 13

1.1 Artificial Intelligence 13

1.2 Computer Vision 14

1.3 Object detection 14

1.4 Weed Detection 15

1.5 Data Reply 15

2 Literature 17

2.1 Computer Vision 17

2.1.1 Convolutional Neural Networks 19

2.2 Object Detection 20

2.2.1 Faster R-CNN 22

2.2.2 YOLO 24

2.3 ONNX 29

2.4 OpenVINO 30

2.4.1 Model Optimizer 31

3 Problem statement and Solution 33

3.1 Context 33

3.2 Camera features 34

3.3 Dataset 35

3.3.1 Labeling 37

3.4 Object Detection Training 39

3.4.1 Faster R-CNN 40

3.4.2 YOLOv5 40

3.5 Real performances estimation 41
3.6 Inference optimization with ONNX and OpenVINO 43

4 Results 45
4.1 Metrics 45
4.1.1 IoU 45
4.1.2 Precision and Recall 47
4.1.3 Mean Average Precision 47
4.2 Faster R-CNN Results 48
4.3 YOLO Results 49
4.4 Real performances estimation 52
4.5 Inference optimization with ONNX and OpenVINO 54

5 Conclusion 57

Bibliography 59

List of Tables

3.1 Flir Blackfly S camera main features. 34

4.1 Complete performances of the model evaluated on the validation set, fine-tuning Faster R-CNN with ResNet-50 backbone on 100 epochs. 48

4.2 Complete performances of the model evaluated on the validation set, fine-tuning Faster R-CNN with ResNet-50 backbone on 100 epochs, excluding rows of crops from the dataset. . . . 49

4.3 Comparison of the performances of YOLOv5s, YOLOv5m and YOLOv5l evaluated in terms of recall of the weeds and of the roots of the weeds. Training processes were run using 100 epochs and the default settings. 49

4.4 YOLOv5m results gotten by tuning the image size and the batch size. Experiments are evaluated in terms of recall of the weeds and of the weeds roots. Training processes were run using 100 epochs 50

4.5 Complete performances of the model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8 51

4.6 Complete performances of the model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8, without the rows of crops. 51

4.7 Estimation of the actual performances of the model in extirpating the weeds' roots, considering a piston with a 1 cm diameter. 53

4.8 Mean inference time for best-performing model in different formats: Pytorch, ONNX, OpenVINO. Statistics are computed on Intel CPU. 54

4.9 Complete performances of the ONNX model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8. 54

4.10 Complete performances of the OpenVINO model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8. 55

4.11 Mean inference time for the best-performing model in different formats: Pytorch, ONNX, OpenVINO. Statistics are computed on NVIDIA GPU. 55

List of Figures

1.1 Multi-Object Detection And Classification. 14

2.1 Computer Vision main tasks [1]. 18

2.2 Convolutional Neural Network example structure. 19

2.3 Two-staged vs One-stage Detectors Diagram. [3] 21

2.4 R-CNN 22

2.5 Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network. 24

2.6 YOLO divides the image into an S × S grid and for each grid cell predicts B bounding boxes. Only the bounding boxes with the highest confidence are kept. 25

2.7 YOLOv5 architecture. 27

2.8 YOLOv5 equation change. 28

2.9 The workflow of OpenVINO. 30

3.1 Tractor with camera attached. 35

3.2 Dataset composition in terms of crops. 36

3.3 Dataset composition in terms of weeds. 37

3.4 Labeled objects: crops in blue, weeds in red, weeds' roots in orange, row of crops in purple. 38

3.5 YOLOv5 different model sizes, where FP16 stands for the half floating-point precision, V100 is an inference time in milliseconds on the Nvidia V100 GPU, and mAP based on the original COCO dataset. 40

3.6 Image taken putting a meter on the ground, in order to collect information about crops and weeds' real dimensions. 42

4.1 Intersection over Union formula. 46

4.2 Case example of how to define TP, TN, FP, FN, when the IoU threshold is set to 0.7. 47

4.3 Confusion Matrix of the best-performing model. On the horizontal axis there is the ground truth, while the predictions are reported on the vertical axis. In the squares of the matrix we observe the corresponding percentages. 52

Chapter 1

Introduction

Weeds are one of the main agricultural problems, as they limit the yield and quality of cultivated fields by competing with crops for light, water, fertilizer, and space. Chemical herbicides have become the primary tool for farmland weeds management worldwide because of their high efficiency. However, manually detecting every single weed would be difficult and time-consuming, hence they are sprayed over large areas, resulting in overuse, low utilization, and serious pollution. Even though herbicides can directly kill the wild plants, excessive use will decrease the yield and quality of agricultural products, cause serious environmental pollution, and reduce the efficiency of agricultural production.

In this project, I propose an alternative approach based on the recognition and localization of weeds in fields, in order to be consequentially extracted through the use of pistons.

1.1 Artificial Intelligence

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn. It encompasses a wide range of technologies and techniques, including machine learning, deep learning, natural language processing, and computer vision, among others. AI systems can perform tasks that typically require human intelligence, such as understanding natural language, recognizing images, and making decisions. The ultimate goal of AI research is to create machines that can perform any intellectual task that a human being can, and some experts believe that this goal is attainable in the future.

1.2 Computer Vision

Machine learning is a particular type of algorithm where machines learn patterns directly from data, perform reasoning about a specific task and extract meaningful information. Deep learning can be considered as a subset of machine learning that makes extensive use of neural networks. The architecture of neural networks is strongly inspired by the structure and functioning of a biological brain. Basically, an artificial neural network is composed of a variable number of neuron layers connected to each other, hence the name "deep" neural networks. The information is elaborated by traveling from layer to layer until the last layer is reached.

1.3 Object detection

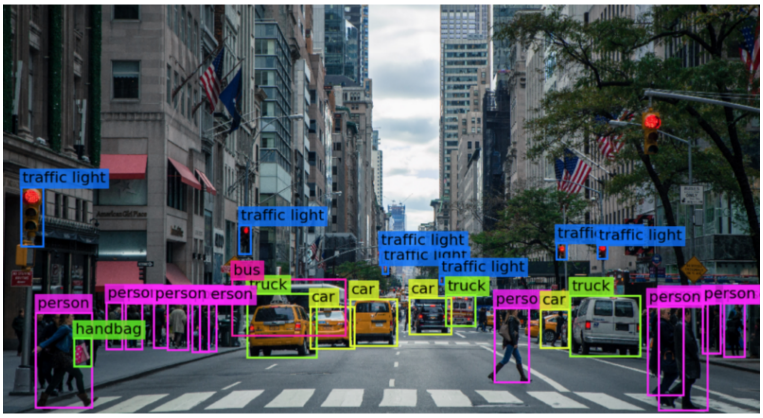


Figure 1.1. Multi-Object Detection And Classification.

Object detection is a computer vision problem that concerns the identification of class instance objects in an image (or video), and locating the actual position of said object in the picture. The spatial position of the detected object is identified by a surrounding rectangular bounding box that determines its height and width. Thus, object detection is far more powerful than image classification, not only because it "draws" a box where the object is located, but also because it can identify multiple elements in a single image, while classification models have the limit of labeling only the one predominant object in the scene.

1.4 Weed Detection

Technology is improving the ways, methods and approaches many old and tedious tasks are being performed. Agriculture deals with the cultivation of crops and rearing of animals for both human and industrial consumption. Due to this fundamental role, it has been subject to different stages of technological development throughout the ages due to humans' need to reduce man's intervention and improve crop yield. One symbolic example of this technological improvement is the invention of machineries such as tractors, harvesters, planters, sprayers, etc..

Regarding weeds, they are generally found among crops, they compete with other plants for resources such as water, nutrients, air and space thus limiting the growth of desired plants. However, since farmers dislike to record loss in the overall result of their farm investment, they exploit multiple approaches to eliminate the presence of weeds and their damages on desired plants: the application of herbicides is the most widespread method. These traditional ways have a significant level of damaging effect on the desired plants and on the surrounding environment. This is where the AI comes in our aid: in order to ensure preciseness and improve accuracy in weed detection, agricultural organizations together with research institutes are now incorporating artificial intelligence principles through various technological tools.

1.5 Data Reply

Reply is a global consulting company that specializes in digital transformation and innovation. The company was founded in Italy in 1996 and has since expanded to have a presence in over 20 countries around the world.

The company offers a wide range of services, including strategy consulting, technology consulting, digital marketing, data analytics, and customer experience design. They work with clients across a variety of industries, including finance, retail, automotive, telecommunications, and more.

One of Reply's key strengths is their focus on emerging technologies, such as artificial intelligence, blockchain, and the Internet of Things (IoT). They have a team of experts dedicated to researching and developing solutions that leverage these technologies to help their clients stay ahead of the curve.

Specifically, I worked on this project at Data Reply, that is a subsidiary of the society group Reply. It focuses specifically on data engineering, data science, and data analytics in general. Their main products are based on researching and developing solutions that include technologies such as artificial intelligence, machine learning, and big data.

Overall, Data Reply's main goal is to help their clients make sense of the vast amounts of data they generate and collect, and turn it into actionable insights that can drive business success.

Chapter 2

Literature

In this chapter, I introduce the background and focus area of this thesis, which is Computer Vision. In particular I will highlight the strengths of object detection and compare it with other Computer Vision tasks. I will analyse YOLO's architecture and explore its potential in recognizing objects in the images and in creating bounding box around them. Finally I will investigate ONNX and OpenVINO, two tools used in order to speed up the model's inference time.

2.1 Computer Vision

Computer vision is the field of Artificial Intelligence that focuses on trying to replicate the complexity of the human vision system. It aims at making it possible for computers to identify and process objects in images and videos in the same way that humans do. Even though the capacity of computers to "see" and recognize the elements of videos and images is still limited, research in artificial intelligence and innovations in deep learning and neural networks have enabled computers to surpass humans in some tasks related to detecting and labeling objects, in particular in terms of speed.

Computer vision is all about pattern recognition: the standard method to teach a computer how to understand visual data is to feed it images - lots of images - thousands, millions if possible, that have been labeled. These images go through complex algorithms that allow the computer to learn how to recognize patterns in all the elements that relate to those labels. The biggest improvement was provided by Deep learning, that quickly become the most effective method to do computer vision and AI in general. In most cases, creating a good deep learning algorithm comes down to gathering a large amount of labeled training data and tuning the parameters such as the type and number of layers of neural networks and training epochs.

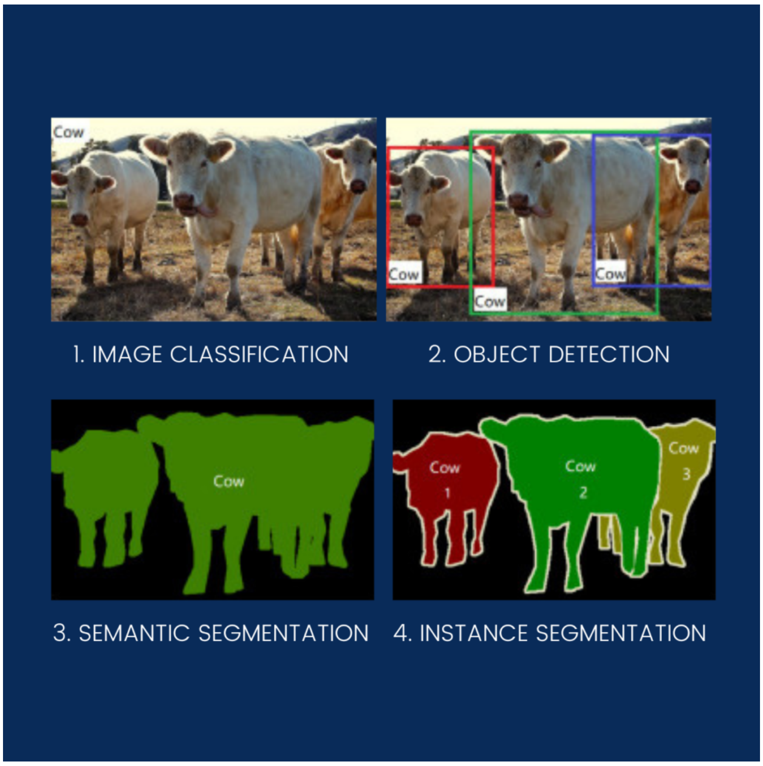


Figure 2.1. Computer Vision main tasks [1].

The main Computer Vision tasks are summarized in Figure 2.1. In particular:

1. **Image classification**: identifying the class to which the main object represented in the picture belongs. In the example in the picture, the deep learning model will determine that the animal detected in the image belongs to the cow class with the highest probability.

2. **Object detection**: it enables the detection of objects in an image and their spatial location. Bounding boxes, that are rectangles, are used to delimit the objects within them.

3. **Semantic segmentation**: identifying similar objects in the image that belong to the same class at the pixel level. The similar objects are colored in the same way to symbolize belonging to the same class.

4. **Instance segmentation**: it recognizes the different instances given in the image with their boundaries at the deep pixel level.

2.1.1 Convolutional Neural Networks

CNNs are deep neural networks that consist of an input layer, an output layer, and a variable number of hidden layers with different purposes.

What differs with respect to standard neural networks is that CNNs have a particular type of layer, called convolutional layer. The input image is represented as a NxMx3 matrix, where the third dimension corresponds to the three color planes: Red, Green, and Blue. The role of Convolutional Network is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction. This is done by the kernel, which is a filter that slithers across the image-matrix and acquires information by making calculations. The objective of the Convolution Operation is to extract the high-level features from the input image. At the end of this process the resulting matrix is passed to the following layer, where other operations will be performed.

To summarize, each network typically contains several convolutional layers of different sizes, each one of them performing a convolution operation that filters its input before passing it to the next layer. The goal of each layer is to reduce or modify the parameters, thus allowing the network to be deeper. An example can be observed in Figure 2.2.

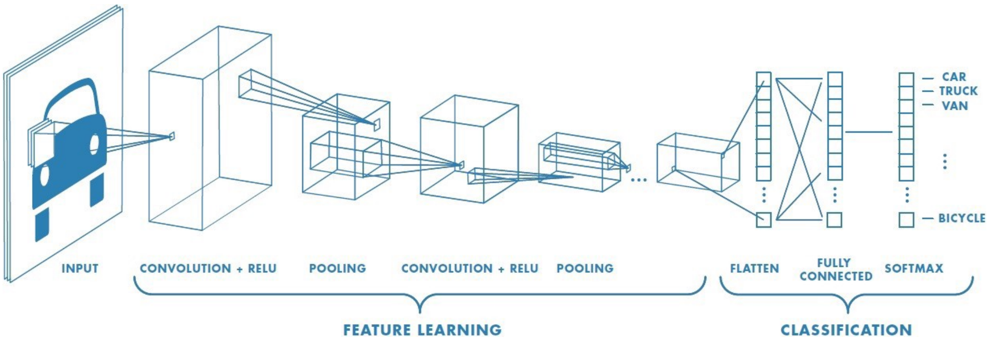


Figure 2.2. Convolutional Neural Network example structure.

CNNs, among others, are mainly used to perform computer vision tasks; in particular they achieve good results in solving object detection problems.

2.2 Object Detection

As one of the primary tasks of computer vision, the ultimate goal of object detection is to give as output the classes and locations of objects. Its success has been amplified by the availability of large-scale open-access datasets, the development of powerful models and the availability of vast computational power. Thus the final aim of object detection is to develop computational models that are able to answer to the question: "What objects are where?" [2].

In the last decade, the fast improvements in the deep learning field have greatly accelerated the pulse of object detection. In fact the complexity of deep learning networks and the computing power of GPUs have made it possible to significantly improve the performance of object detectors, achieving significant breakthroughs in object detection. As a result, numerous real-world applications, such as healthcare monitoring, autonomous driving, video surveillance, anomaly detection, or robot vision, are based on deep learning object detection.

Even though deep learning based techniques are more robust to complex scenes in the images, they usually consist of supervised methods. As a consequence a huge amount of training data is needed. In order to build a suitable training set the process of image annotation is necessary: it may be labor-intensive and expensive, since a great number of images are required. For example, labeling thousands of images to train a custom DL object detection algorithm can be considered a small dataset. However, luckily many benchmark datasets (MS COCO, Caltech, KITTI, PASCAL VOC, V5) provide a great amount of labeled data. Moreover most object detectors are trained for a specific resolution of input. These detectors generally underperform for inputs having different scales or resolutions. In general it is important to pay enough attention on the training data: for example all the object detection algorithms will tend to perform well on larger objects if the model is trained on larger objects and will show poor performance on comparatively smaller sized objects.

Object detectors aim at extracting features from the input image/video frame by solving two main tasks:

1. Find an arbitrary number of objects (possibly even zero);

2. Classify every single object and estimate its size with a bounding box.

We can divide object detectors into two main types: One-stage and Two-stage detectors.

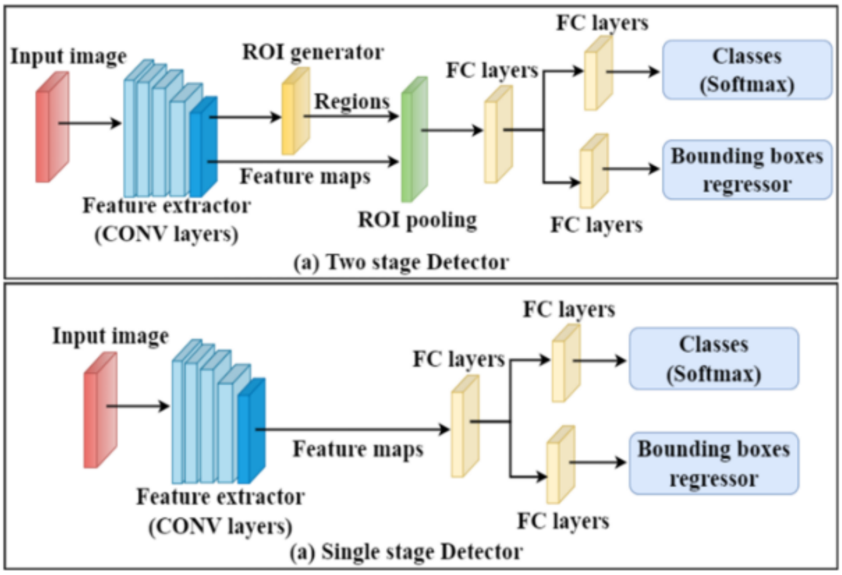


Figure 2.3. Two-staged vs One-stage Detectors Diagram. [3]

You can observe their structure in Figure 2.3.

In a two-staged detector the tasks described above are solved in two separate stages: first several object candidates are proposed, known as regions of interest (RoI), using reference boxes (anchors). In the second phase, the proposals are classified and their localization is refined.

In R-CNN the authors used an external selective search in order to build proposals that are passed to a CNN in order to perform classification and bounding box regression. Later, Faster-RCNN significantly improved this process by proposing a scheme in which features are shared between both stages: a convolutional backbone network, such as VGG or ResNet, outputs global feature maps shared between the Region Proposal Network (RPN) and the detection network. This process reduces the cost of generating proposals externally.

Faster R-CNN became a pioneer and has inspired many follow-up works that have tried to improve detection accuracy with different approaches. This type of architecture achieves a very high accuracy rate but is rather slow, then these models are usually considered unsuitable for real-time applications.

One-stage detectors combine both tasks into one step to achieve higher performance at the cost of accuracy. They predict bounding boxes over the images without the region proposal step, as they contain a single feed-forward fully convolutional network that directly provides the bounding boxes and the object classification. These detectors do not apply a separate network for extracting candidate regions, hence they show better performance in terms of processing time and can be used more easily in real-time applications. The most popular one-stage detectors include YOLO, SSD, and RetinaNet.

2.2.1 Faster R-CNN

As anticipated, Faster R-CNN is a two-stage deep convolutional network used for object detection that can accurately and quickly predict the locations of different objects. In order to understand how Faster R-CNN works, it is necessary to do a quick overview of the networks that it evolved from, namely R-CNN and Fast R-CNN.

R-CNN (Region-based Convolutional Neural Network) [4] is a deep convolutional network that can detect 80 different types of objects in images. It consists of the following 3 main modules that are summarized in Figure 2.4:

- The first module delivers 2,000 region proposals using the Selective Search algorithm.
- The second module extracts a feature vector of length 4,096 from each region proposal.
- The third module uses a pre-trained SVM algorithm to classify the region proposal to either the background or one of the object classes.

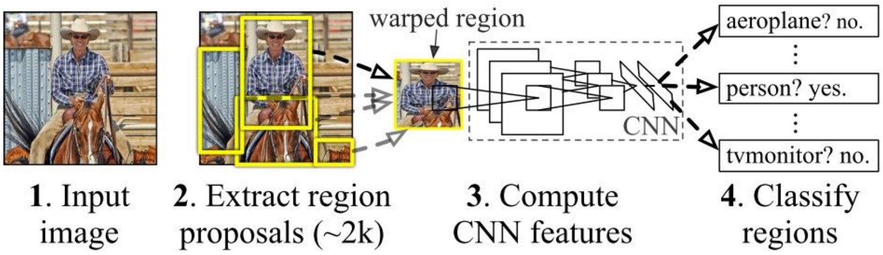


Figure 2.4. R-CNN

To overcome some limitations of R-CNN, the Fast R-CNN [5] model has been proposed as an extension. Its main characteristics are:

- The authors introduced a new layer called ROI Pooling which extracts equal-length feature vectors from all proposals (i.e. ROIs) in the same image.

- The new ROI Pooling layer is also used to share the computations (i.e. convolutional layer calculations) across all proposals (i.e. ROIs), while in R-CNN calculations were done singularly for each proposal. This makes Fast R-CNN significantly faster.
- Fast R-CNN does not cache the extracted features and thus does not need so much disk storage compared to R-CNN.
- Fast R-CNN reaches a greater accuracy with respect to R-CNN.

Faster R-CNN [6] was created as an extension of Fast R-CNN. The main contributions brought by its authors are:

- Region Proposal Network (RPN): a fully convolutional network that generates proposals with various scales and aspect ratios.
- Anchor boxes: reference boxes of a specific scale and aspect ratio. With multiple reference anchor boxes, then multiple scales and aspect ratios exist for the single region. This can be thought of as a pyramid of reference anchor boxes. Each region is then mapped to each reference anchor box, and thus detecting objects at different scales and aspect ratios.
- The convolutional computations are shared across the RPN and the Fast R-CNN. This reduces the computational time.

Summarizing, Faster R-CNN is composed of two modules:

1. Region Proposal Network (RPN): a fully convolutional network that takes an image as input and outputs a set of object proposals, which are regions in the image that are likely to contain objects. The RPN is designed to be computationally efficient and can generate proposals very quickly. In particular it takes an image as input and passes it through a series of convolutional layers to generate a set of feature maps. At each spatial location in the feature maps, the RPN generates a set of anchor boxes, boxes of different sizes and aspect ratios that are used to propose object locations. For each anchor box, the RPN predicts two values: the probability that the anchor box contains an object, and the offset between the anchor box and the ground-truth box. These predictions are made using a set of convolutional layers and a set of fully connected layers. The RPN then applies non-maximum suppression (NMS) to the proposed boxes to remove redundant detections and select the top N

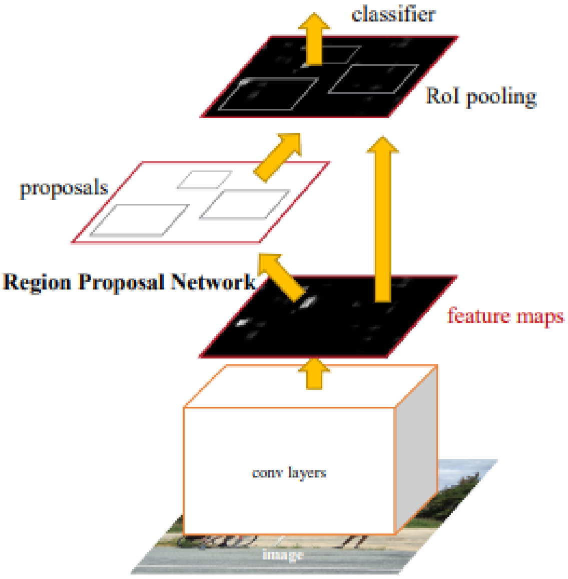


Figure 2.5. Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network.

boxes for further processing. These boxes are then passed to the second component of the Faster R-CNN architecture.

2. Fast R-CNN network: its goal is to classify each proposed box as containing an object or not, and to refine the coordinates of the box so that it more accurately encompasses the object. The Fast R-CNN network takes each proposed box as input and passes it through a series of convolutional and fully connected layers to generate a fixed-length feature vector. This feature vector is then used to classify the box and refine its coordinates using two separate output layers. The classification layer produces a probability distribution over the object classes, and the bounding box regression layer produces four offsets that are used to refine the coordinates of the box. The final output of the Fast R-CNN network is a set of object detections, each with a class label and a refined bounding box.

2.2.2 YOLO

Currently YOLO [7] represents the state of the art of object detection in real-time. As a matter of fact, although the models increased in accuracy each year, object detectors lacked speed that allowed them to perform in real time. That is the goal of YOLO: turning the entire object detection pipeline into a single network that could be optimized end-to-end directly on detection performance.

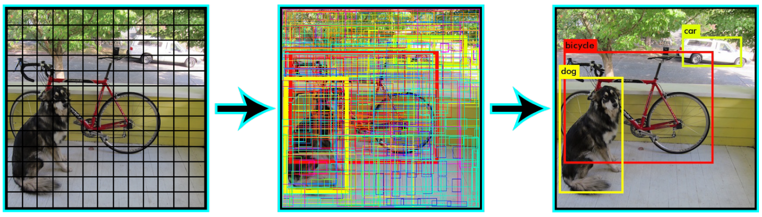


Figure 2.6. YOLO divides the image into an S × S grid and for each grid cell predicts B bounding boxes. Only the bounding boxes with the highest confidence are kept.

This is how YOLO works:

- Divides images into S regular grids;
- Performs detection and localization on these grids, obtaining B bounding boxes. Each bounding box consists of:
    1. the bounding box **coordinates**: usually they are made up of 5 numbers: the x coordinate of the box's center, the y coordinate of the box's center, the width, the height, and the confidence. Note that the width and height are fractions relative to the entire image size;
    2. the object **label**, which is represented by a one-hot vector of length C, the number of classes in the dataset. Thus it will be an all-0 vector, unless for the i-th position, which is a 1, where i is the label index;
    3. the **probability** that the object is in the cell grid; the classification score will be from 0.0 to 1.0, with 0.0 being the lowest confidence level and 1.0 being the highest. These confidence levels capture how much the model is certain that there exists an object in that cell and that the bounding box is accurate.
- Suppresses the bounding boxes that have lower probability scores;

Then the overall prediction of the model after the first two steps is a tensor of shape $S \times S \times (C + B * 5)$.

One possible drawback of this process is that each cell predicts B bounding boxes and multiple bounding boxes may overlap. Therefore the same object could be predicted with multiple slightly different bounding boxes. In order to solve this issue, YOLO exploits a Non-Maximal suppression algorithm: in summary, it divides the image into grids of equal size, performs object detection and classification, then it chooses the highest probability score and suppresses all the bounding boxes having the largest IoU with the current high probability bounding box. Basically if the same object is surrounded by multiple overlapping bounding boxes, only the one with the highest score is kept. It repeats these stages in a loop until the final bounding boxes are obtained.

YOLOv5

At the time when this project has been developed, the latest YOLO version available was YOLOv5 [8], the implementation provided by Ultralytics.

YOLOv5 consists of the same three key components, its architecture can be observed in Figure 2.7:

1. **backbone**: it is made up of convolutional layers and it aims at detecting the key features of an image and processing them.

2. **neck**: it combines information from layers of different depths, uses the features from the convolutional layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates.

3. **head**: it is the final output layer of the network, it is responsible for the predictions and it can be interchanged with other layers with the same input shape in order to transfer learning.

More specifically, in YOLOv5 the three components [11] are built in the following way:

- **CSP-Darknet53 as a backbone**: CSP-Darknet53 is just the convolutional network Darknet53 to which the authors applied the Cross Stage Partial (CSP) network strategy. As a matter of fact YOLO uses residual and dense blocks in order to deal with the vanishing gradient issue. However this leads to the problem of redundant gradients. CSPNet helps tackle this problem by truncating the gradient flow.

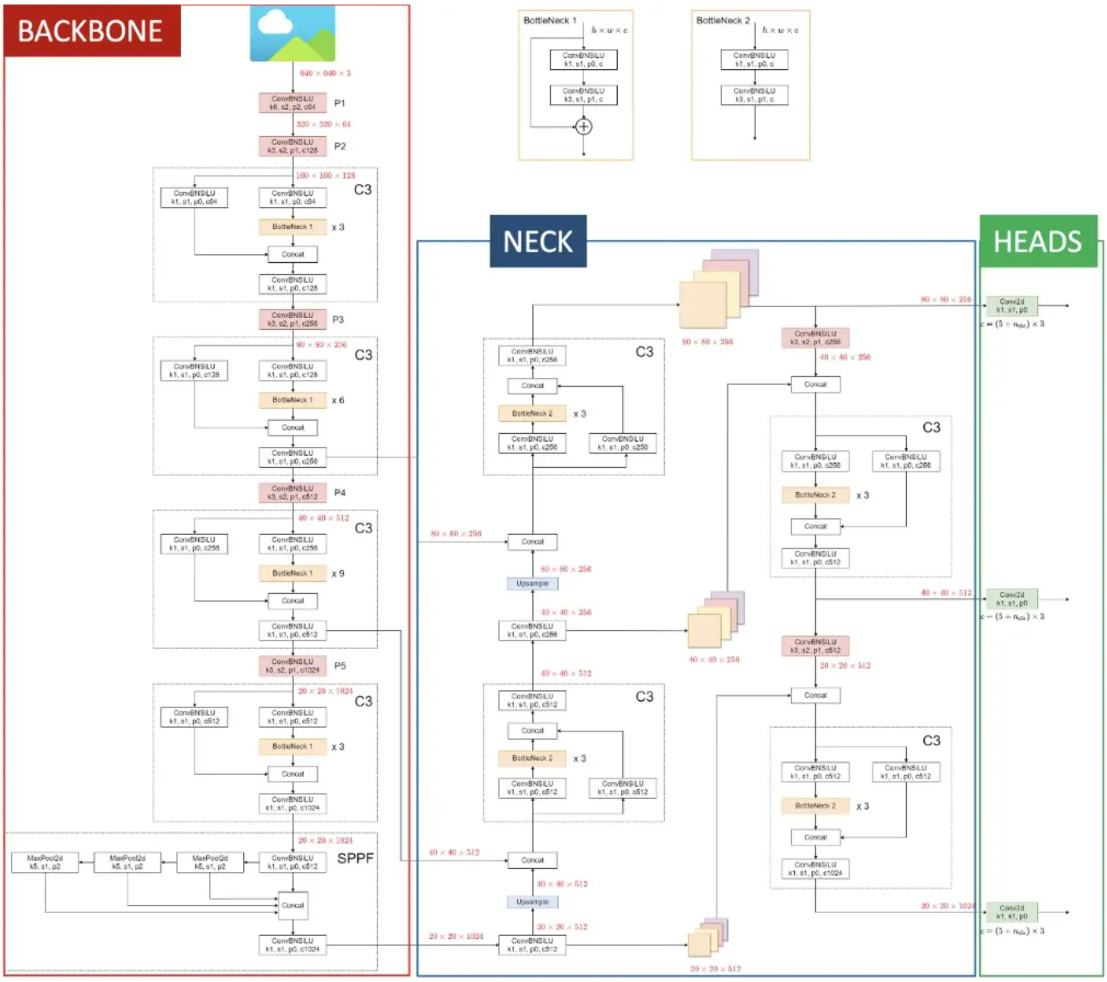
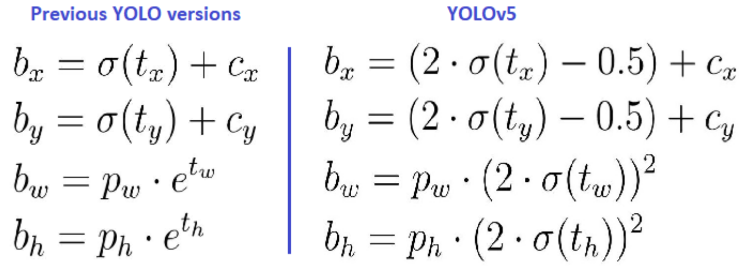

BACKBONE
640 × 640 × 3
ConvBNSiLU k6, s2, p2, c64
P1
320 × 320 × 64
ConvBNSiLU k3, s2, p1, c128
P2
160 × 160 × 128
C3
ConvBNSiLU k1, s1, p0, c64
ConvBNSiLU k1, s1, p0, c64
BottleNeck 1
x 3
Concat
ConvBNSiLU k1, s1, p0, c128
ConvBNSiLU k3, s2, p1, c256
P3
80 × 80 × 256
C3
ConvBNSiLU k1, s1, p0, c128
ConvBNSiLU k1, s1, p0, c128
BottleNeck 1
x 6
Concat
ConvBNSiLU k1, s1, p0, c256
ConvBNSiLU k3, s2, p1, c512
P4
40 × 40 × 512
C3
ConvBNSiLU k1, s1, p0, c256
ConvBNSiLU k1, s1, p0, c256
BottleNeck 1
x 9
Concat
ConvBNSiLU k1, s1, p0, c512
ConvBNSiLU k3, s2, p1, c1024
P5
20 × 20 × 1024
C3
ConvBNSiLU k1, s1, p0, c512
ConvBNSiLU k1, s1, p0, c512
BottleNeck 1
x 3
Concat
ConvBNSiLU k1, s1, p0, c1024
20 × 20 × 1024
SPPF
MaxPool2d k5, s1, p2
MaxPool2d k5, s1, p2
MaxPool2d k5, s1, p2
ConvBNSiLU k1, s1, p0, c512
Concat
ConvBNSiLU k1, s1, p0, c1024
BottleNeck 1
$h \times w \times c$
ConvBNSiLU k1, s1, p0, c
ConvBNSiLU k3, s1, p1, c
BottleNeck 2
$h \times w \times c$
ConvBNSiLU k1, s1, p0, c
ConvBNSiLU k3, s1, p1, c
NECK
HEADS
80 × 80 × 256
C3
ConvBNSiLU k1, s1, p0, c256
Concat
BottleNeck 2
x 3
ConvBNSiLU k1, s1, p0, c256
ConvBNSiLU k1, s1, p0, c256
80 × 80 × 256
Concat
80 × 80 × 256
Upsample
40 × 40 × 256
ConvBNSiLU k1, s1, p0, c256
40 × 40 × 256
40 × 40 × 512
C3
ConvBNSiLU k1, s1, p0, c512
Concat
BottleNeck 2
x 3
ConvBNSiLU k1, s1, p0, c512
ConvBNSiLU k1, s1, p0, c512
40 × 40 × 512
Concat
40 × 40 × 512
Upsample
20 × 20 × 512
20 × 20 × 1024
ConvBNSiLU k1, s1, p0, c512
20 × 20 × 512
80 × 80 × 256
ConvBNSiLU k3, s2, p1, c256
40 × 40 × 256
Concat
C3
ConvBNSiLU k1, s1, p0, c256
ConvBNSiLU k1, s1, p0, c256
BottleNeck 2
x 3
Concat
ConvBNSiLU k1, s1, p0, c512
40 × 40 × 512
ConvBNSiLU k3, s2, p1, c512
20 × 20 × 512
Concat
C3
ConvBNSiLU k1, s1, p0, c512
ConvBNSiLU k1, s1, p0, c512
BottleNeck 2
x 3
Concat
ConvBNSiLU k1, s1, p0, c1024
20 × 20 × 1024
Conv2d k1, s1, p0
$c = (5 + n_{cls}) \times 3$
Conv2d k1, s1, p0
$c = (5 + n_{cls}) \times 3$
Conv2d k1, s1, p0
$c = (5 + n_{cls}) \times 3$

Figure 2.7. YOLOv5 architecture.

The application of this strategy represents a big advantage to YOLOv5, since it helps reduce the number of parameters hence reducing the number of FLOPS needed. This leads to increasing the inference speed, which is a crucial point in real-time object detection models and one of the main strengths of YOLO.

- **SPPF and PANet as a neck**: PANet is a feature pyramid network, it was already used in YOLOv4 and in other previous YOLO variants in order to improve information flow and help in the proper localization of pixels in the task of mask prediction. In YOLOv5 this network has been modified by applying the CSPNet strategy. Also a variant of Spatial Pyramid Pooling (SPP), that is the SPPF, has been used: it is a block that performs an aggregation of the information that receives from the inputs and returns a fixed length output. For this reason it increases the receptive field and isolates the most relevant context features without lowering the speed of the network.

- **Convolution layers as head**: Just like in YOLOv3 and YOLOv4, the head is composed of three convolution layers that predict the location of the bounding boxes (x, y, height, width), the scores and the objects classes. The equation to compute the target coordinates for the bounding boxes has changed from previous versions, the difference is shown in Figure 2.8.



Previous YOLO versions

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w \cdot e^{t_w}$$
$$b_h = p_h \cdot e^{t_h}$$

YOLOv5

$$b_x = (2 \cdot \sigma(t_x) - 0.5) + c_x$$
$$b_y = (2 \cdot \sigma(t_y) - 0.5) + c_y$$
$$b_w = p_w \cdot (2 \cdot \sigma(t_w))^2$$
$$b_h = p_h \cdot (2 \cdot \sigma(t_h))^2$$

Figure 2.8. YOLOv5 equation change.

Other important YOLOv5 features include:

- **Activation function**: first of all the Sigmoid Linear Unit is used with the convolution operations in the hidden layers, while the Sigmoid activation function has been used with the convolution operations in the output layer.

- **Loss function**: as anticipated in the previous sections, YOLOv5 gives three outputs as a result: the classes of the detected objects, their bounding boxes and the objectness scores. The performances over these components are computed by using the Binary Cross Entropy (BCE) to compute the classes loss and the objectness loss, while the Complete Intersection over Union (CIoU) loss to compute the location loss. Then the final loss is computed by combining the three losses in the following way:

$$Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc}$$

2.3 ONNX

ONNX [9] stands for Open Neural Network Exchange. It is used for mainly two different tasks:

- Convert model from any framework to ONNX format, and then from ONNX format to any desired framework
- Faster inference using ONNX model on supported runtimes

Considering that the aim of this project is to build a real-time system capable of identifying and hitting weeds in the shortest time possible, the third task is particularly interesting.

In general if you want to reduce the computing power required to train an AI model you have two options: you can make the AI model smaller, which is risky in terms of performances, or make the model more efficient. This is where ONNX comes in our help: you can use ONNX to make a given model faster, eliminating the need to use a GPU instead of a CPU. As a matter of fact we need to consider that the PC used by the cultivators is a middle-level computer and in general CPUs have a broader availability and are cheaper to use with respect to GPUs.

From a technical point of view, the ONNX format is composed of a protobuf schema, that defines the structure of the model, and a set of binary data files, that report the actual values of the model's parameters. The protobuf schema defines the structure of the model as a graph, where nodes represent operations and edges represent the flow of data between them. Each node in the graph represents a particular operation in the model, such as a convolution or a fully connected layer, and contains information about its inputs, outputs, and parameters.

When you export a model to the ONNX format, the framework converts the model's graph into the ONNX protobuf schema and saves the parameter values as binary data files. When you import the model into another framework, that framework can use the ONNX protobuf schema to reconstruct the graph and load the parameter values from the binary data files. This allows the model to be used in the new framework without having to retrain it from scratch.

In conclusion ONNX supports a wide range of operations and architectures, making it a versatile format for exchanging models between different frameworks, and provides a standardized way to represent deep learning models, making it easier and faster to share and reuse models across different frameworks and platforms.

2.4 OpenVINO

OpenVINO [10] stands for Open Visual Inference and Neural Network Optimization. It is an open-source toolkit used to optimize and deploy AI inference on Intel products. The OpenVINO toolkit covers both computer vision and non-computer vision workloads across Intel hardware. It maximizes performance and accelerates application development.

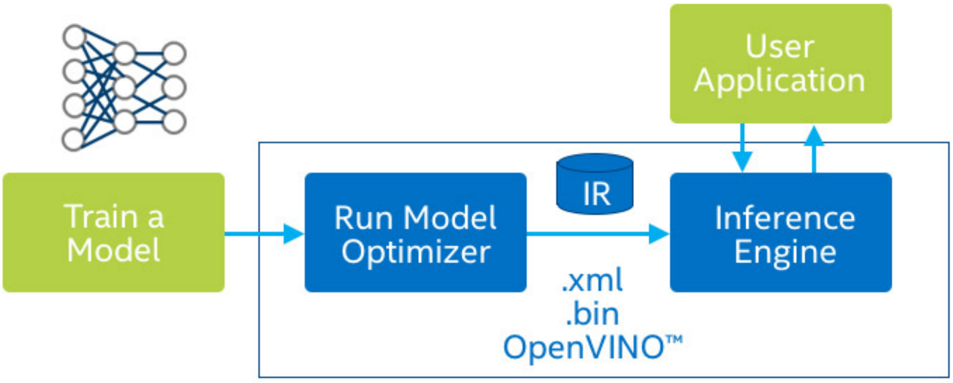


Figure 2.9. The workflow of OpenVINO.

The OpenVINO workflow is summarized in Figure 2.9 and it mainly consists of four main steps:

1. **Train**: A model is trained with code.
2. **Model Optimizer**: The trained model is fed to the Model Optimizer. Its aim is to optimize the model and deliver an Intermediate Representation (.xml + .bin files) of the model. The models are optimized with techniques such as quantization, freezing, fusion, and more. In this step, pre-trained models are configured according to the framework chosen and then converted with a simple, single-line command.
3. **Inference Engine**: The Intermediate Representation is passed to the Inference Engine. The inference engine checks for model compatibility based on the original framework used to train the model as well as the environment used, which is the hardware. Frameworks supported by OpenVINO include, for example, TensorFlow, and ONNX.
4. **Deployment**: The application is deployed to devices.

2.4.1 Model Optimizer

There are many Deep Learning frameworks widely used in the industry such as TensorFlow; on the other hand, Intel provides a wide range of devices and platforms to choose from. Mapping from each of these frameworks to any of these devices can be a complex task. First because the representation of Deep Learning models is totally different from one framework to the other; second, each of these devices has a different architecture, different instruction set and programming model.

Intel supplies one common API that could be used to implement inference across all of these devices and basically abstract the Hardware for you. It works in two phases:

- Convert the model into one unified representation called Intermediate Representation (IR).
- Use the IR file to inference on multiple devices.

The model optimizer is responsible for the first phase: it converts the format, optimizes and converts the weights and biases.

- Convert: no matter how the model is represented by the Deep Learning framework, the Model Optimizer will convert it to IR. The IR is composed of two files: an .xml file which describes the topology, the layers, the connectivity, the parameters; a binary file that holds all the weights and biases.
- Optimize: the MO optimizes the model using techniques like fusion of layers, batch-normalization and more. These are hardware agnostic optimizations which can save a lot of computation and memory.
- Weight format: models today are usually trained with weight format of FloatingPoint32, but sometimes the device used can run faster using FP16. The MO can convert the format of the weights automatically.

However we cannot just squeeze the weights into a smaller format without losing accuracy of the model. We get to a trade-off: the more bits are allocated to represent the data and weights, the wider range they could represent and potentially the better accuracy of the model. But it comes with a price, because the bigger the data, the more space in memory is required to store it and the more computational resources and time it requires to compute.

Chapter 3

Problem statement and Solution

In this chapter I will clarify the main goal of this project, as well as the main obstacles encountered, especially during the data collection process, and the different solutions that have been adopted.

3.1 Context

The main goal of this project is to develop a Computer Vision tool capable of identifying weeds and their roots in cultivated fields. This algorithmic tool could be used by a machine equipped with pistons to automatically detect weeds through a camera and then extract them through the pistons. For this specific weed detection application, it is crucial for the model to accurately detect weed's roots so that the piston can successfully extrapolate them. If the piston hits the weed but not in the root, the weed will later grow again, leading to a decrease in the effectiveness of the weed detection system. By accurately detecting and localizing the weed's roots, the system can effectively remove the entire weed from the soil, preventing future regrowth. Thus, the ability of the model to accurately detect and localize weeds' roots is a critical component of the weed detection system, and should be a key consideration when selecting the appropriate computer vision algorithm.

A Semantic Segmentation approach would be significantly data demanding, as it requires more detailed and comprehensive annotation of each pixel in the image, and a larger amount of labeled data to effectively train the model. Therefore the best option was proceeding in an Object Detection direction: the goal is to make the model able to recognize each weed, border it inside a rectangular box, so that the piston could hit right in the center of the box and extrapolate it. This is a real-time process: the tractor is equipped with a camera that takes the picture; approximately one meter following the camera we have the pistons system that, once the photo has been analyzed and the weeds detected, acts in order to hit them.

In order to train a model to identify the weeds and distinguish them from the good crops, it is necessary to build a suitable dataset. For this reason we needed the cultivators to take pictures of the cultivated fields in the same condition in which the tool will be finally used. Especially at the beginning of the project, the cultivators had not provided us with a big amount of images: collecting the images was a time-consuming process for them, as well as some crops were still too small to be captured. Also they faced some climate-related issues as it did not rain for some weeks. As a consequence the crops have not grown up enough for some weeks longer than usual. Thus in the very first weeks I had to run my first experiments using only 300 images and the very first models were fine-tuned using this little dataset. Luckily after some time the farmers were able to collect some new images that enriched our dataset: we could get 1369 images. Some of the crops already reached a significant dimension, with a length of more than 1 cm. Some other crops were just born, measuring only few millimeters.

3.2 Camera features

Figure 3.1 reports the overall system used to collect the images. The camera is attached to a sliding structure that is pulled by a tractor. The tractor slowly moves across the fields, while the camera automatically takes pictures. They used a Flir Blackfly S, a small and compact machine vision camera. Its main features are summarized in Table 3.2.

Resolution	2048x1536
Megapixels	3.2 MP
Sensor	Sony IMX252, CMOS, 1/1.8"
Pixel size	3.45 micron

Table 3.1. Flir Blackfly S camera main features.

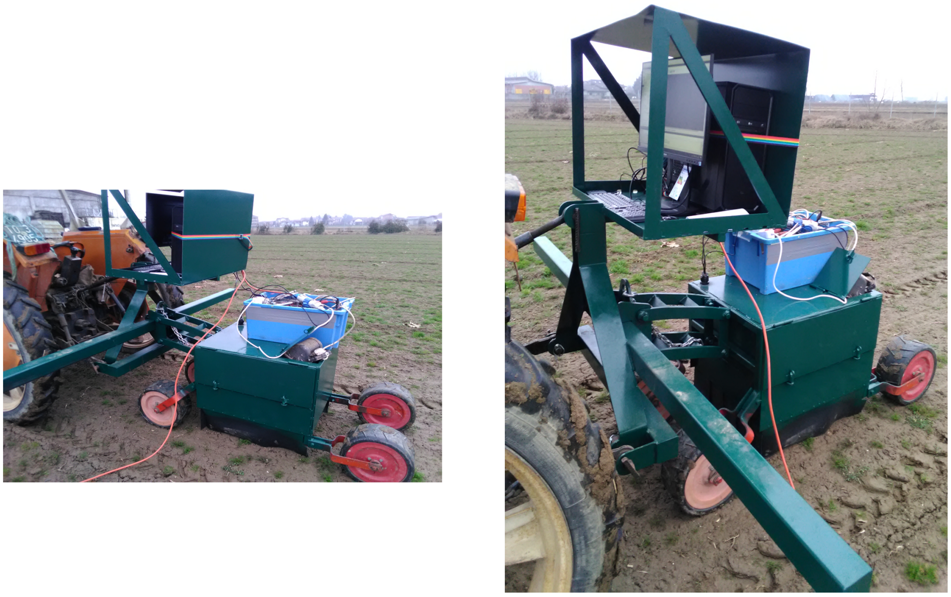


Figure 3.1. Tractor with camera attached.

3.3 Dataset

The dataset was built thanks to the farmers who collected hundreds of images of the crops in the fields. Thus we could create our dataset with 1369 images distributed along 6 different types of crops:

- **Valerian**: 182 images, it is commonly used as a sleep aid and as a sedative for nervous tension, stress, and intestinal cramps;
- **Hypericum**: 61 images, traditionally considered as a medicinal herb to treat a variety of ailments, including depression, anxiety, and nerve pain;
- **Mallow**: 347 images, its properties are exploited to treat sore throat, cough, and other respiratory problems, as well as for culinary purposes;
- **Savory**: 346 images, it is usually used for culinary and medicinal purposes, to help relieve stress and promote relaxation;
- **Eschscholzia**: 304 images, it is particularly used in cosmetics, as it is thought to have anti-inflammatory properties and can help reduce redness and irritation;

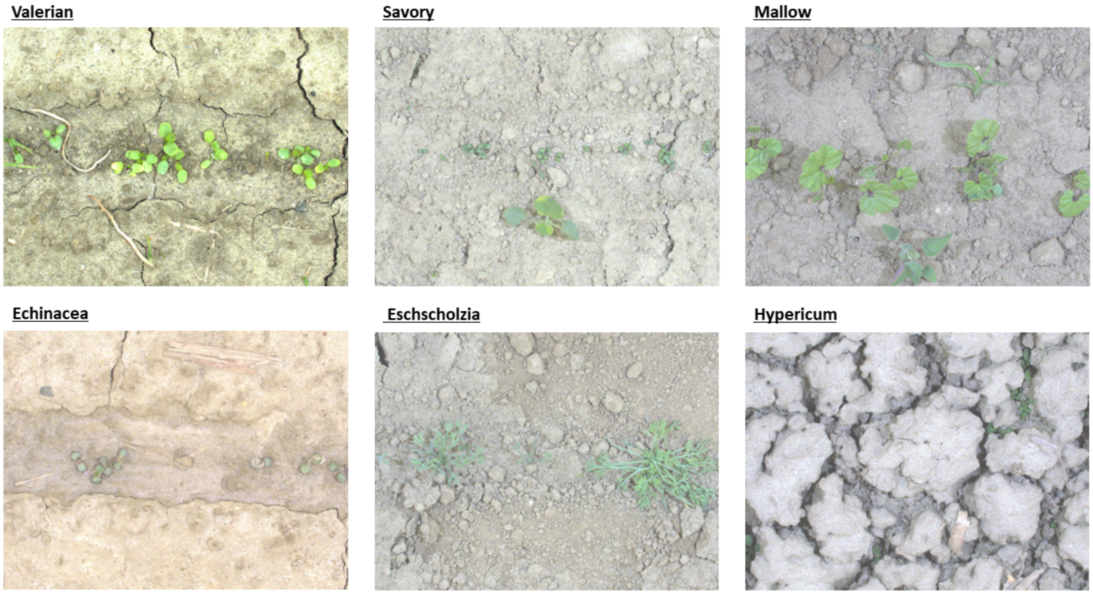


Figure 3.2. Dataset composition in terms of crops.

- **Echinacea**: 129 images, it is widely used as a natural remedy to boost the immune system and help fight off infections as well as in skincare products due to its anti-inflammatory and antioxidant properties.

Their form can be seen in Figure 3.2: note that some of them, like Mallow and Eschscholzia, have a very different aspect, but are both in an advanced growing state. At the same time Echinacea and Hypericum are at the very beginning of their growing process. In the latter case it is even difficult to notice the little crops hidden under the ground.

Regarding the weeds, I summarized the most frequent macro-categories of weeds' types in Figure 3.3. Note that weeds can assume multiple forms and sizes, independently from the crop field in which they grow up. Therefore it is not easy to categorize them, but by analyzing the images it was possible to identify the most frequent and representative typologies.

Weeds can vary a lot in the aspect, thus it will be a significant obstacle for the model in the training phase, where it needs to find common features in order to learn how to recognize them.

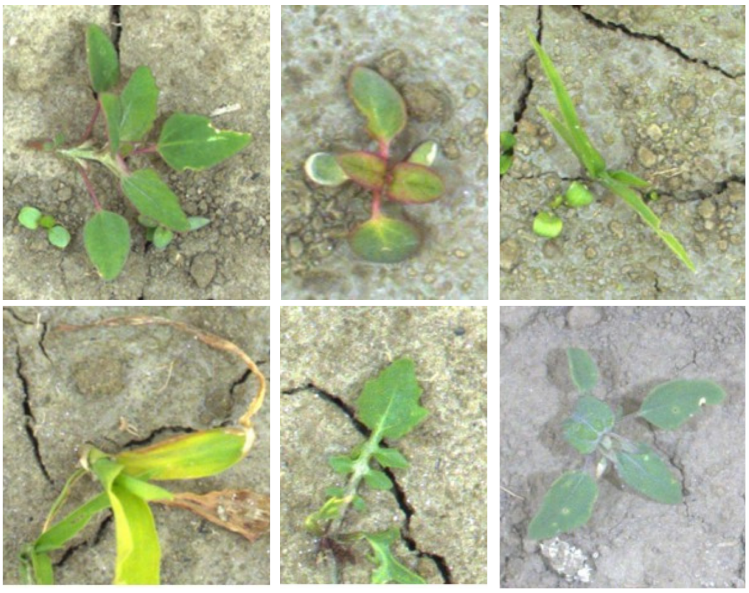


Figure 3.3. Dataset composition in terms of weeds.

3.3.1 Labeling

After collecting the data, in order to build a complete dataset the labeling step was necessary: this stage consists of delimiting in rectangles the elements of interest, that the algorithm aims to recognize, using the LabelImg [12] tool.

The labels for the images need to be reproduced in the YOLO recognized format. For each image, one .txt file is required and it denotes the objects' labels for the given image. The .txt file reports one row for each object: if multiple objects are present in a single image, then we will get multiple rows corresponding to the objects' labels found in the image. Each row has 5 values of the corresponding bounding box in the following sequence:

`class number`, `x-center`, `y-center`, `width`, `height`.

Note that the coordinates for the bounding box should be in the normalized format, so the values should be between 0 and 1. Finally each class corresponds to a number and the numbering starts from 0.

After that I implemented a script to transform each .txt file to a .xml, the PASCAL VOC format that is used by Faster R-CNN. In particular the image data is stored in a folder, with each image file given a unique identifier, and for each image an annotation file is created to describe the objects present in the image. This file is typically stored in XML format and contains information about the class label, bounding box coordinates, and any other relevant information about each object. I then used the annotations in PASCAL VOC format to run my experiments on Faster R-CNN.

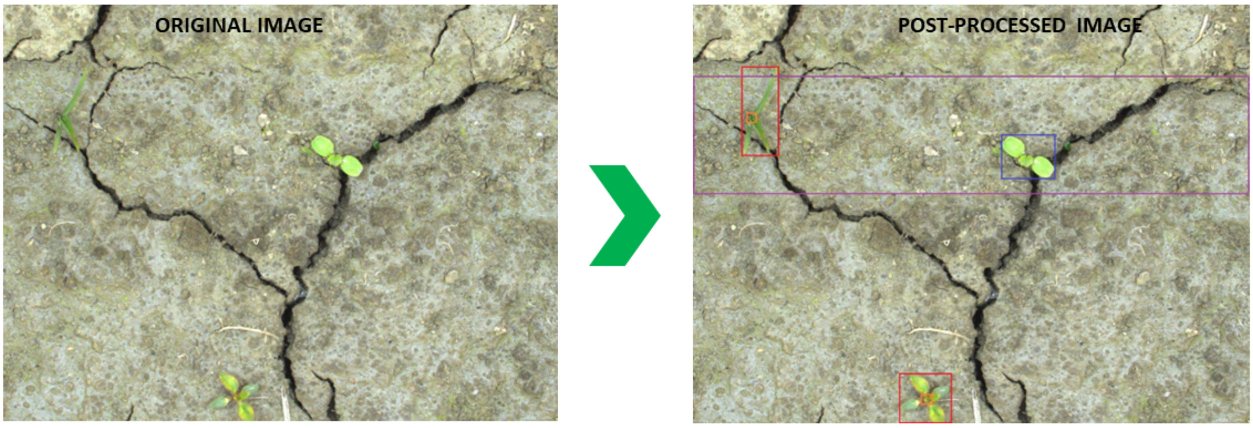


Figure 3.4. Labeled objects: crops in blue, weeds in red, weeds' roots in orange, row of crops in purple.

As represented in Figure 3.4, in our images I labeled:

- **Crops**: The good plants
- **Weeds**: The infesting plants
- **Roots**: The origin of the weeds, where we want the pistons to hit
- **Rows**: The line of crops

Note that all the crops' typologies were generically labeled as "crops", and the same goes for weeds.

As expected this process took a long time since labeling needs to be as much precise as possible. In particular labeling the roots for each weed took a large amount of time and we could not exploit all the images that the client had sent to us, since some of them were empty, or some others were repeated.

Communication and collaboration with cultivators were critical during the labeling process.They possess valuable knowledge about the characteristics of different crops and weeds, which could aid in accurately labeling the training dataset. By working closely with cultivators to understand the differences between different plant species and their growth patterns, we could ensure that the training dataset was properly annotated, which is essential for developing an accurate and effective weed detection system.

Once the labeling process was concluded, the following amount of each element was identified:

- 7201 crops
- 2088 weeds
- 2088 weeds' roots
- 1251 rows of crops

3.4 Object Detection Training

Now the dataset has been split into three sets:

- Training set: 1092 images (80%)
- Validation set: 140 images (10%)
- Test set: 137 images (10%)

The splitting cannot be done randomly because photos were taken one after the other and most of the time a given image shares some percentage of the left or right part with the nearby image. Therefore if a given image is used to train the model, when it is tested on the nearby image, it will surely recognize the already seen elements quite easily, since some of those elements were part of the training set.

For this reason, I had to re-plan the splitting of the dataset, using "blocks" of images for each crop type. For each crop type, the first 80% of images were assigned to the training set, the following 10% to the validation set and the last 10% to the test set. In this way the model is not tested on the same portions of images on which it has been trained on.

For my weed detection application, I selected and evaluated two popular object detection algorithms, Faster R-CNN and YOLOv5, on my dataset. Both algorithms were chosen because of their proven effectiveness in detecting and classifying objects in complex and cluttered environments, which is a key requirement for accurately detecting weeds in fields with crops. For both algorithms, since the training dataset is limited in terms of dimensions, I used a pre-trained model on a significantly larger public dataset and then I fine-tuned it on the context-specific dataset.

3.4.1 Faster R-CNN

The original Faster R-CNN paper used VGG16 as the backbone network. Since then, several other backbone architectures have been used, with ResNet being one of the most popular. Using ResNet can improve the performance of the model while reducing the training time. For this reason I finetune the pretrained Faster R-CNN model with a ResNet-50-FPN backbone provided by torchvision library. In the fine-tuning phase I used as hyperparameters `image size` = 1280, `learning-rate` = 0,001, `batch-size` = 4.

The Faster R-CNN model included in the torchvision package is pre-trained on the Microsoft Common Objects in Context (COCO) dataset. This is a large-scale object detection dataset that contains over 330,000 images and over 2.5 million object instances labeled with 80 different object categories. The COCO dataset is commonly used to pre-train object detection models due to its large size and diverse range of object categories and scenes. By pre-training on this dataset, the Faster R-CNN model included in torchvision is able to learn a general understanding of object detection tasks, which can be fine-tuned on custom datasets for specific applications.

3.4.2 YOLOv5

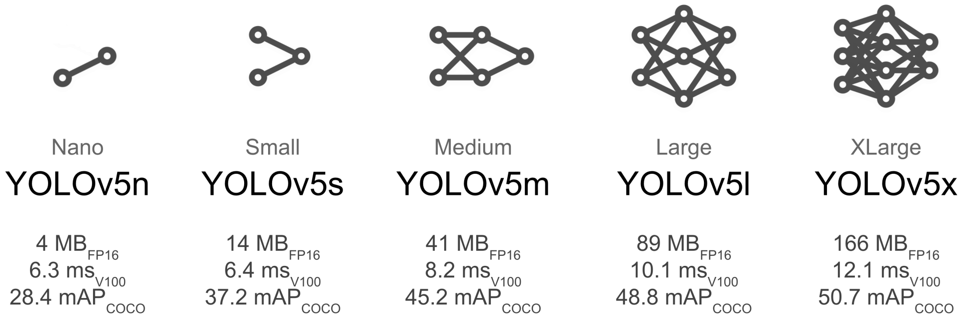


Figure 3.5. YOLOv5 different model sizes, where FP16 stands for the half floating-point precision, V100 is an inference time in milliseconds on the Nvidia V100 GPU, and mAP based on the original COCO dataset.

The YOLOv5 model by Ultralytics is pre-trained on the COCO dataset, just like Faster R-CNN in torchvision. As a matter of fact The COCO dataset is a popular choice for pre-training object detection models because it contains a large and diverse set of images and object categories, which allows the model to learn robust representations of objects in general.

Five different pre-trained models alternatives have been proposed by YOLOv5's authors. There is no difference between the five models in terms of operations used except for the number of layers and parameters. Their structure is summarized in Figure 3.5

In particular, I fine-tuned YOLOv5 using Ultralytics' repository and I choose the three pre-trained models in the middle in Figure 3.5, which are YOLOv5s, YOLOv5m and YOLOv5l, as a basis. YOLOv5s is the smallest and fastest variant of YOLOv5, with fewer parameters and lower accuracy compared to the other variants. It is a good choice for real-time applications or deployment on low-power devices. YOLOv5m is the medium-sized variant, which offers a balance between speed and accuracy. It has more parameters and higher accuracy compared to YOLOv5s, but still maintains a reasonable inference time. Finaly YOLOv5l is the large-sized variant of YOLOv5, which has even more parameters and higher accuracy compared to YOLOv5m. However it is slower and requires more computational resources, but offers better accuracy especially for detecting small objects. I tested these three pre-trained models with default hyperparameters, `image size` = 640, `learning-rate` = 0,01, `batch-size` = 16, and selected the best performing one, then I operated hyperparameters tuning on that.

3.5 Real performances estimation

Once the training phase was concluded, a question arose: "How can we evaluate if the piston will actually hit the root of the weeds, especially when the model is not able to recognize the weed's root?".

For this reason, we elaborated our strategy:

- When the algorithm detects the root, the piston directly hits the root;
- When the algorithm does not detect the root but successfully detects the weed, the piston hits the center of the box surrounding the weed.

However the cultivators explained that a weed can be successfully extrapolated only if it is hit right in the root. If the piston hits the weed in some marginal part, this will result only in a reduction in weed's size. This would not be a successful result since the weed would grow up again in a short time and would represent a problem for the surrounding crops.

Therefore it is fundamental to estimate the percentage of successfully extirpated weeds, both considering the ones that have been detected with their roots and the ones where the algorithm has missed the root. In order to perform this estimation one point is missing: we do not know what is the actual size in centimeters of each weed/crop. This means that, in those cases in which the model detects the weed but does not detect its root, the system would simply hit the center of the weed's box, but we do not know if doing so the piston would actually hit the root.

Our solution was very simple: I asked the cultivators to come in our help, taking some additional photos of the cultivated fields reproducing the same conditions in which the dataset pictures were taken, but putting a meter on the ground, so that I was able to estimate the actual size of weeds and crops in images. An example of these images is reported in Figure 3.6.

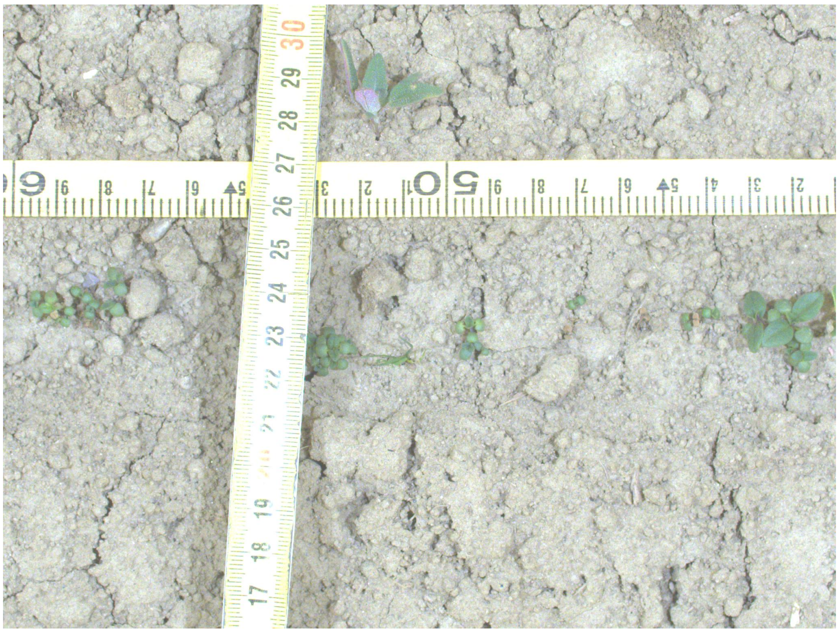


Figure 3.6. Image taken putting a meter on the ground, in order to collect information about crops and weeds' real dimensions.

I passed this image on the labeling tool, LabelImg, where I draw a 1cm x 2cm box, following the meter's values in the image. After that, I considered the label that was now created and compared centimeters to pixels. Following this process, I was able to discover that

$$1\,cm = 110\,pixels.$$

Once the model had been trained and I had chosen the best-performing one, I applied the final model on the test set. For each image, a .txt file has been delivered, reporting all the objects' coordinates and classes that the model was able to detect. In order to evaluate the actual performances of the model I followed these steps:

- When a weed is detected together with its root, we can assume that the piston would successfully hit and extrapolate the given weed;
- When the weed is detected, but not its root, we need to consider that the piston would hit the center of the weed's box. Therefore I measure how far the center of the box is with respect to the actual root of the weed. If the distance falls inside the action range of the piston, then the root would be hit anyway; otherwise, the root is missed by the piston.

3.6 Inference optimization with ONNX and OpenVINO

The final aim of this application is to make our system able to recognize and extrapolate the weeds in a real-time environment, hence speed plays a fundamental role. Considering that models can be very heavy and need a lot of time to be run, there is a significant focus around faster inference for real time use-cases, especially when, just like in our case, models will be run on a not-powerful computer. This is why a few years back Intel released OpenVINO toolkit for optimizing inference of Deep Learning models on Intel's hardware. In our case, the computer owned by the cultivators is provided with a Intel CPU, hence it will be possible to exploit OpenVINO. Note that OpenVINO is not a toolkit for faster training of Deep Learning tasks, but for faster inference of already trained Deep Neural models. Since at the moment it is not possible for me to test OpenVINO directly on the cultivators' computer, I will simulate the whole processing of converting the model and testing it on my computer, which is provided with a Intel CPU as well: 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz.

However it is important to point out that you usually have to sacrifice a bit of accuracy in inference in order to get better speed: this is a trade-off that we need to consider and include in the evaluation of the final result. I will investigate the possibility to improve the speed of the best-performing model in inference and confront its improvement with the actual changes in terms of performance metrics.

The whole process consists of the following steps:

- Take the pre-trained best-performing YOLOv5 model;

- Convert the Pytorch model to ONNX, which enables interoperability between different frameworks;
- The ONNX model is passed to the Model Optimizer, in order to produce an Intermediate Representation composed of an xml and a bin file.
- Test the OpenVINO model on my personal computer, that is provided with a Intel CPU and compare the results with the original model.

Observe that this process will be applied only on the best-performing model, evaluated considering the overall performance metrics. As a matter of fact it would not make sense to repeat the steps above for all the models that we will compare.

Moreover one of the intermediate stages is to convert the original model to the ONNX format, but one of the main ONNX's potential tasks is to speed up the model's inference time. Therefore I test the ONNX-converted model as well and compare its inference time with the original and the OpenVINO ones. I also evaluate how its performances change with respect to the Pytorch format.

As an additional experiment I test OpenVINO also on the GPU, that is a NVIDIA GeForce RTX 3060, in order to demonstrate and check that we get an improvement in terms of inference time only on Intel products.

Chapter 4

Results

In this chapter I will report all the experiments that I have run as well as the metrics used to evaluate and compare them.

Each experiment consists of a training phase conducted on the training dataset, while the performance are computed on the validation set. Once the model parameters have been tuned, the performances are re-computed and verified on the test set. In order to evaluate the performances of each model, multiple metrics are taken into consideration.

4.1 Metrics

Note that the object detection task localizes the objects in the images with a bounding box associated with its corresponding confidence score to report how certain the bounding box of the object class is detected. A given object could be detected by the model, but not completely in its covered area or it could be completely missed. At the same time objects can be wrongly identified by the model even when they are not there. For this reason object detection metrics serve as a measure to assess how well the model performs on an object detection task, not only considering the number of objects correctly detected, but also the precision with which the bounding box is drawn around them.

4.1.1 IoU

One of the main metrics in Object Detection is the Intersection Over Union (IoU) [13]. Before defining how it works, it is necessary to remind the definitions of:

- True Positive (TP): the prediction by the model is correct.
- False Positive (FP): the prediction by the model is incorrect.
- False Negative (FN): an element in the Ground-truth has not been detected by the object detector.
- True Negative (TN): a background region was correctly not detected by the model. Actually this metric is not used in object detection because background regions are not explicitly annotated when preparing the annotations.

IoU evaluates the degree of overlap between the ground truth and prediction. They can be of any possible shape: rectangular box, circle, or even irregular shape.

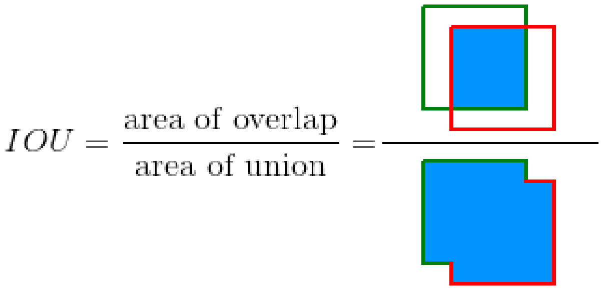

$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$

Figure 4.1. Intersection over Union formula.

As shown in Figure 4.1, IoU is computed as the area of the intersection divided by the area of union between ground-truth and the predicted box. The result will be a numeric value between 0 and 1. If the two areas perfectly overlap, then the IoU will be 1; instead if they do not have any common point, the result will be 0.

But how to define how much overlapping area, and so how much IoU, is enough to make a prediction a "good" enough prediction? For example, if the ground-truth and the predicted box correspond to a $IoU = 0.8$, will the prediction count as a True Positive or as a False Positive? We need to define a threshold.

From the example in Figure 4.2 we can observe that, given a IoU threshold set to 0.7, the first case would be considered as a good prediction (True Positive). In the second case, the amount of overlapping area is not enough, thus the ground-truth is considered as not-detected (False Negative), and the prediction as a false prediction (False Positive). Same for the third case.

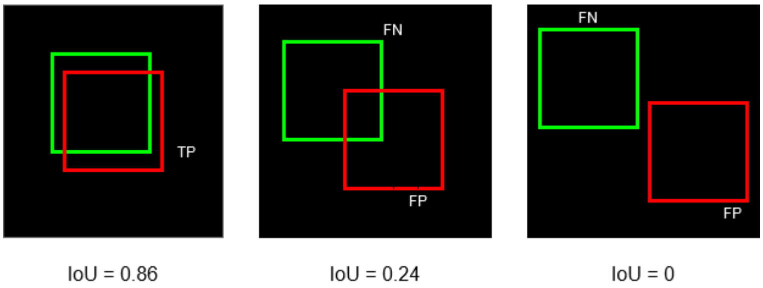


Figure 4.2. Case example of how to define TP, TN, FP, FN, when the IoU threshold is set to 0.7.

Note that, if in the same example the threshold was lowered to 0.2, the second case would become a True Positive. This underlines that when we use IoU as a metric, it is important to evaluate the detections taking into account the IoU fixed threshold.

4.1.2 Precision and Recall

Precision and Recall are commonly used in many Machine Learning fields. They are defined as:

- Precision: it is the degree of exactness of the model in identifying only relevant objects. It is the ratio of correct detections over all detections made by the model. More formally:

$$Precision = \frac{TP}{TP + FP}$$

- Recall: it measures the ability of the model to detect the ground truths. It is the ratio of correct detections over all the ground truths.

$$Recall = \frac{TP}{TP + FN}$$

4.1.3 Mean Average Precision

Mean Average Precision (mAP) is commonly used to analyze the performance of object detection and segmentation systems. The mAP is also used across several benchmark challenges such as Pascal, VOC, COCO, and more.

To better understand how to compute mAP, here is a summary of the steps to calculate the AP:

- Generate the predictions using the model
- Calculate TP, FP, TN, FN
- Calculate the precision and recall metrics
- Calculate the area under the precision-recall curve
- Measure the average precision

The mAP incorporates the trade-off between precision and recall and considers both false positives (FP) and false negatives (FN). This property makes mAP a suitable metric for most detection applications.

mAP metric is dependent on the IoU threshold, so calculating mAP over an IoU threshold range avoids the ambiguity of picking the optimal IoU threshold for evaluating the model's accuracy.

4.2 Faster R-CNN Results

In this paragraph, I report the results that I got utilizing Faster R-CNN to analyze our weed dataset and evaluate its performance in detecting and classifying weeds and crops. The results of the analysis are reported in the Table 4.1 and reveal that, while the model was able to achieve moderate success in identifying and detecting crops and weeds, it struggled to detect very small objects such as weeds' roots. At the same time, it finds it very difficult to detect the rows of crops, probably due to their indefinite dimension. However the model produced discrete results on crops and weeds, that are slightly bigger objects with respect to the roots, highlighting the limitations of the algorithm in accurately detecting smaller objects

Class	Images	Istances	Precision	Recall
all	140	1207	0.216	0.289
Crop	140	670	0.405	0.546
Weed	140	205	0.460	**0.607**
Weed Root	140	205	0.001	**0.003**
Row of crops	140	127	0.000	0.000

Table 4.1. Complete performances of the model evaluated on the validation set, fine-tuning Faster R-CNN with ResNet-50 backbone on 100 epochs.

Given the very low performance of the model in identifying the rows of crops, I tested it by excluding the rows of crops from the dataset, so that the training process will be more centered on learning how to detect crops and weeds. The results are reported in Table 4.2: as expected I get better results in weeds' recall, passing from 0.607 to 0.724, while there is no significant improvement in roots' recognition.

Class	**Images**	**Istances**	**Precision**	**Recall**
all	140	1207	0.345	0.463
Crop	140	670	0.464	0.667
Weed	140	205	0.572	**0.724**
Weed Root	140	205	0.001	**0.003**

Table 4.2. Complete performances of the model evaluated on the validation set, fine-tuning Faster R-CNN with ResNet-50 backbone on 100 epochs, excluding rows of crops from the dataset.

4.3 YOLO Results

First I decided to compare the three "middle" models in terms of size: YOLOv5s, YOLOv5m and YOLOv5l. As reported in Figure 3.5, their main difference is the total number of parameters, which implies a consequent greater or smaller time required to perform inference. As anticipated, the main term of comparison between the experiments is the recall computed for weeds and for weeds roots. As a matter of fact, it is fundamental to reduce the number of weeds as much as possible, even though some crops could be mistaken for weeds and hence extirpated.

Model size	**Weeds' Recall**	**Weeds roots' Recall**
YOLOv5s	0.932	0.678
YOLOv5m	**0.932**	**0.688**
YOLOv5l	0.922	0.678

Table 4.3. Comparison of the performances of YOLOv5s, YOLOv5m and YOLOv5l evaluated in terms of recall of the weeds and of the roots of the weeds. Training processes were run using 100 epochs and the default settings.

Results are reported in Table 4.3: YOLOv5m is the best performing model.

It is capable of detecting the 93.2% of weeds correctly and in the 68.8% of cases, it also detects the weeds' roots successfully.

At this stage I tune the hyperparameters related to the image size and the batch size. The image size is the pixel size at which photos are processed: larger image sizes usually lead to better results, but take longer to process, so it's up to the user to find the right compromise between speed and accuracy. Consider that the various YOLOv5 models were trained on 640 x 640 images. Our images are taken at high quality (2048x1536 in terms of resolution), hence I keep 1280 as a starting image size and try with different batch sizes, maximizing the GPU memory usage on Google Colab. Due to the GPU memory limit, it was not possible to run the experiment using both `batch size` = 32 and `image size` = 640, then I downgrade the `image size` to 640.

The results are summarized in Table 4.4. It is reasonable that by lowering the `image size`, we get a significant drop in terms of weeds roots' recall: their dimension is very small, then it makes sense for the model to struggle in learning the main features related to the roots having a smaller number of pixels to look at.

Image size	**Batch size**	**Weeds' Recall**	**Weeds roots' Recall**
1280	8	**0.932**	**0.688**
1280	16	0.927	0.668
640	32	0.917	0.357

Table 4.4. YOLOv5m results gotten by tuning the image size and the batch size. Experiments are evaluated in terms of recall of the weeds and of the weeds roots. Training processes were run using 100 epochs

In Table 4.5 you can observe the complete performances of the best performing model with `image size`=1280 and `batch size`=8 with all the metrics described previously. Moreover it is possible to better analyze how and when the model is doing mistakes by observing the Figure 4.3.

We can state that the model successfully recognizes weeds and it is widely capable of distinguishing them from the good crops. This is fundamental in order not to get the pistons to hit and kill the crops.

When weeds are mistaken, it is because the model seems to identify weeds in the background, when they are not there. This is comforting since the pistons would simply hit the free ground in most of cases.

Moreover a good number of weeds' roots have been successfully identified.

Class	Images	Istances	Precision	Recall	mAP50	mAP50-95
all	140	1207	0.779	0.788	0.825	0.474
Crop	140	670	0.902	0.936	0.965	0.727
Weed	140	205	0.87	**0.932**	0.956	0.721
Weed Root	140	205	0.63	**0.688**	0.664	0.207
Row of crops	140	127	0.715	0.598	0.716	0.24

Table 4.5. Complete performances of the model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8

Due to their very small dimensions, roots' performances are slightly lower with respect to weeds.

Class	Images	Istances	Precision	Recall	mAP50	mAP50-95
all	140	1080	0.836	0.815	0.845	0.551
Crop	140	670	0.905	0.918	0.963	0.737
Weed	140	205	0.911	**0.902**	0.959	0.738
Weed Root	140	205	0.692	**0.626**	0.613	0.177

Table 4.6. Complete performances of the model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8, without the rows of crops.

I also repeated the experiment with the same parameters, but excluding the rows of crops from the labels. The result can be observed in Table 4.6, where it is possible to note that there is a decrease in performances in terms of recall of weeds and weeds' roots.

After conducting further experiments on my dataset, it was found that YOLOv5 outperformed Faster R-CNN in detecting and classifying both small and larger objects such as weeds, crops, and weeds' roots. YOLOv5 achieved significantly higher accuracy in identifying smaller objects that were previously missed by Faster R-CNN, while also maintaining its performance on larger ones. These findings suggest that YOLOv5 is better suited for this specific weed detection application, where the identification of smaller objects such as weeds' roots is crucial.

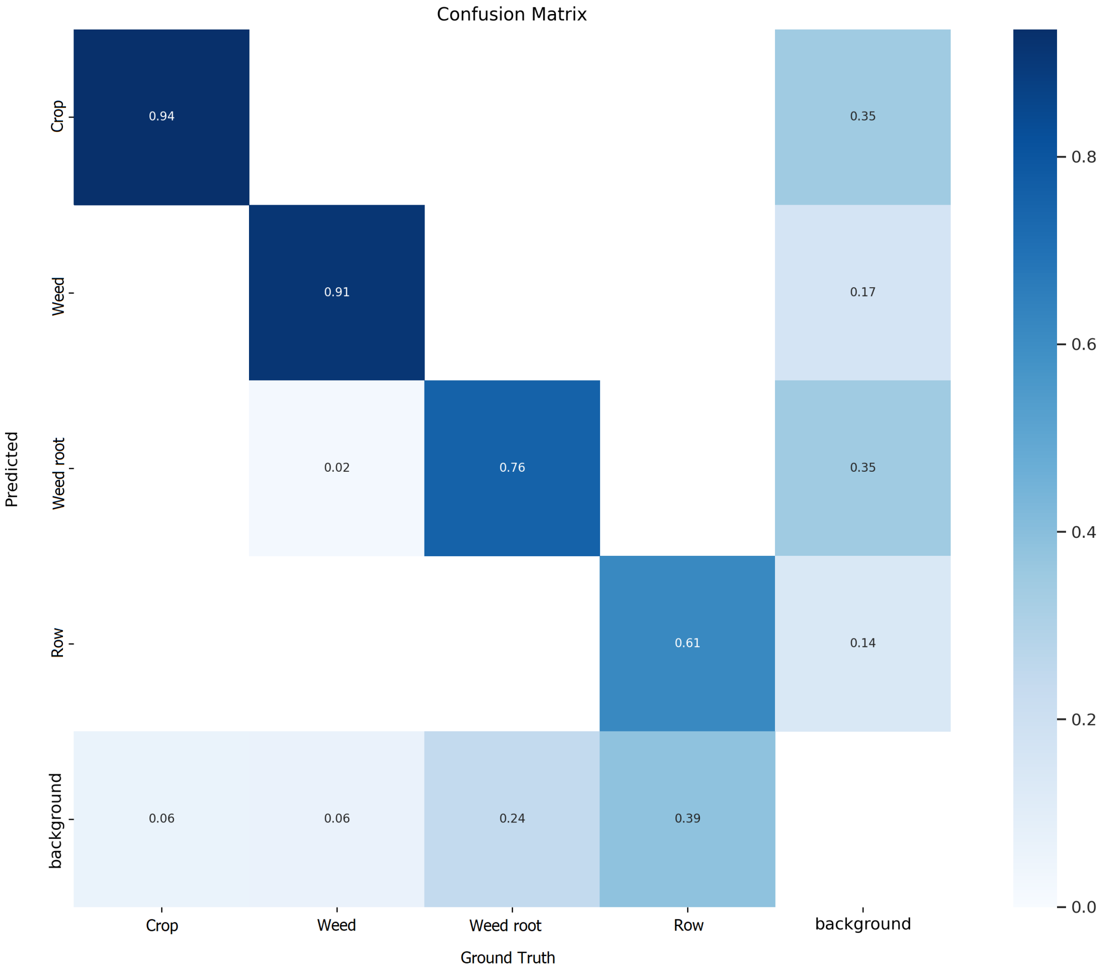


Figure 4.3. Confusion Matrix of the best-performing model. On the horizontal axis there is the ground truth, while the predictions are reported on the vertical axis. In the squares of the matrix we observe the corresponding percentages.

4.4 Real performances estimation

In order to correctly evaluate the model's performance in the real application, it needs to be tested considering the pistons' dimensions as well as the crops and weeds' real size.

As a matter of fact in Section 4.3 we have observed that the model is able to correctly detect the 93,2% of weeds, but only the 68,8% of weeds' roots in the validation set. The final strategy can be summarized as follows:

- hit the weed's root when it is detected inside the weed box;
- hit the center of the weed's box when no root is detected.

Therefore now it is fundamental to understand if the pistons would actually hit the roots, even when no root has been identified by the model.

To finalize this process I considered the result of the prediction of the best-performing model over the validation and test sets. I took as a reference the weeds reported in the labels. First of all, I excluded all the weeds whose root was correctly identified by the model: 297 weeds out of 436. At this point, I took into consideration all the weeds that were detected by the model but not their root. For each of those target weeds, I computed the distance between the center of the weed's box (detected by the model) and the actual position of the root. At first, the distance was calculated in pixels, and then converted to centimeters considering that, thanks to the reference images provided by the cultivators, I was able to observe that 1 cm in real life corresponds to 110 pixels in the image.

The piston that will be used to hit the weeds has a diameter of 1 cm, therefore the weed's root has to be inside this range in order to be successfully extrapolated. Therefore if the distance between the center of the box detected by the model and the actual position of the weed's root is smaller than 0.5 cm (piston's radius) we can consider the weed as extirpated, otherwise the weed survives. The final results can be seen in Table 4.7.

Result	**Number of images**
Weed's root detected by the model	297
Weed's root inside the piston's radius	113
Weed's root not extirpated	26

Table 4.7. Estimation of the actual performances of the model in extirpating the weeds' roots, considering a piston with a 1 cm diameter.

As a result for 410 weeds out of 436, which is the 94% of the total, the root gets within the range of action of the piston; the remaining 26 roots would not be hit by the piston. This successful result is given by the fact that the majority of weeds whose root has not been detected by the model has a very small size, hence even if the model is not able to detect their root, their center is anyway inside the range of the piston.

Finally I was able to observe that in most cases, when the model is not even able to detect the weed itself, it's because the weed is very small, or it blends in with the ground.

4.5 Inference optimization with ONNX and OpenVINO

As a first step I compare the different results obtained by performing inference on the validation set using the same model, that is the best performing one from Section 4.3. From Table 4.8 it is possible to observe the results in terms of inference time. As expected we have an improvement both with ONNX and OpenVINO, with respect to the Pytorch model, which is the baseline. Regarding ONNX, the model is sped up by the 43.9% of the original inference time; while OpenVINO brings an advance of 13,4%.

	Mean inference time (ms)	**Standard deviation (ms)**
Pytorch	221.63	7.95
ONNX	**124.14**	0.85
OpenVINO	192.02	5.65

Table 4.8. Mean inference time for best-performing model in different formats: Pytorch, ONNX, OpenVINO. Statistics are computed on Intel CPU.

In particular for what concerns ONNX, we get a satisfactory result. However, we need to evaluate how the model's performances have changed, and possibly got worsened.

Class	**Images**	**Istances**	**Precision**	**Recall**	**mAP50**	**mAP50-95**
all	140	1207	0.818	0.779	0.806	0.5
Crop	140	670	0.852	0.942	0.956	0.708
Weed	140	205	0.863	**0.917**	0.937	0.68
Weed Root	140	205	0.575	**0.351**	0.355	0.095
Row of crops	140	127	0.983	0.908	0.974	0.517

Table 4.9. Complete performances of the ONNX model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8.

In Table 4.9 we observe the overall performance of the ONNX model on the validation set. We need to compare it with Table 4.5: there is a decrease in weed's recall from 0.932 to 0.917, and a significant drop in weed root's recall from 0.688 to 0.351. This is very meaningful and demonstrates that we cannot just fasten the model without losing accuracy. We get to a trade-off between speed and performance.

Class	**Images**	**Istances**	**Precision**	**Recall**	**mAP50**	**mAP50-95**
all	140	1207	0.848	0.765	0.803	0.507
Crop	140	670	0.9	0.925	0.957	0.716
Weed	140	205	0.881	**0.922**	0.933	0.686
Weed Root	140	205	0.622	**0.322**	0.345	0.094
Row of crops	140	127	0.991	0.891	0.976	0.53

Table 4.10. Complete performances of the OpenVINO model evaluated on the validation set, fine-tuning YOLOv5m and keeping image size=1280 and batch size=8.

We can proceed with the same evaluation for the OpenVINO model and observe the Table 4.10: here we can note a slightly lower decrease in weed's recall from 0.932 to 0.922, but again an even more significant drop in weed root's recall from 0.688 to 0.322. If we balance this result with the small improvement in terms of inference time - that was only 13.4% - the performance worsening helps us move the needle.

As an additional experiment I tested OpenVINO and ONNX also on NVIDIA GeForce RTX 3060 GPU. This is not an Intel product, hence we expect no improvement in terms of inference time. The result can be seen in Table 4.11: while Pytorch performs significantly better on GPU, passing from 221.63 ms as mean inference time to 10.84 ms, ONNX is quite the same with respect to CPU performances, and OpenVINO gets worse, passing from 192.02 ms to 227.46 ms.

	Mean inference time (ms)	**Standard deviation (ms)**
Pytorch	10.84	0.56
ONNX	127.61	2.32
OpenVINO	227.46	8.59

Table 4.11. Mean inference time for the best-performing model in different formats: Pytorch, ONNX, OpenVINO. Statistics are computed on NVIDIA GPU.

Chapter 5

Conclusion

The objective of this thesis was to investigate the use of computer vision techniques for real-time weed recognition in cultivated fields. The idea behind this project is to build a system equipped with a set of pistons, capable of automatically identifying weeds in cultivated fields and consequently extrapolating them through the use of mechanical components. Due to the limited size of the dataset, an object detection approach was preferred over a semantic segmentation one. Two state-of-the-art pre-trained models, Faster R-CNN and YOLOv5, were utilized, and their performance was evaluated after finetuning on a custom dataset.

The results of the experiments showed that both models are capable of detecting weeds and roots in the images. However, YOLOv5 outperformed Faster R-CNN: the evaluation criteria were specifically focused on weed detection, with an emphasis on recall, especially for weed roots. This is because in agricultural applications, it is crucial to detect as many weeds as possible, even if some crops could be mistaken for weeds.

The finetuned YOLOv5 model demonstrated high recall rates for both weed detection and weed root detection, making it a viable solution for weed management in cultivated fields. Hence, YOLOv5 was selected as the preferred model and was further optimized through hyperparameters tuning.

In order to correctly evaluate the model's performance in the real application, it was tested considering the pistons' dimensions as well as the crops and weeds' real size. To address this, I developed a strategy to hit the weed root when it is detected inside the weed box, and hit the center of the weed's box when no root is detected. Based on this analysis, the roots of 94% of the total weeds were within the range of the piston, hence would be successfully extrapolated. The majority of the remaining 6% were not detected by the model since they were very small and/or blended in with the ground.

Since YOLOv5 model is intended to be used in a real-time application, it is crucial to optimize its inference time to ensure that it can keep up with the demands of the system. To accomplish this, I used ONNX and OpenVINO to optimize the model's performance in terms of inference time. Even though this process is necessary to ensure that the model can operate in real time without significant lag or delay, it could result in a decrease of quality metrics.

After carefully assessing the resulting trade-off between speed and recall loss, it emerged that ONNX brings an improvement in terms of inference time, speeding up the model of 44%, at the expense of 0.33 recall points on weeds' roots.

In conclusion, the use of Artificial Intelligence in agriculture has proven to be a promising technology that can significantly improve crop yield, reduce costs and increase efficiency. One area in particular where AI has shown great potential is weed detection and this thesis has demonstrated the feasibility and effectiveness of using object detection techniques, specifically YOLOv5 model, for weed recognition in cultivated fields. The use of a bigger dataset with images from various crop types could potentially improve the model performance. Currently, the model has only been trained on specific crop types, which limits its generalizability. A bigger dataset could provide the model images of new crops and weeds that the model has not encountered before, allowing it to learn better how to distinguish weeds from crops. Moreover, a bigger training dataset can enable the use of semantic segmentation techniques, leading to more accurate and detailed predictions. In general, the use of AI-based weed detection systems can enable farmers to identify and target weeds more accurately and efficiently, allowing for timely intervention and reducing the need for herbicides. This not only saves time and money but also reduces the environmental impact of farming. As the demand for food continues to increase with the growing population, the adoption of AI in agriculture will become more important than ever before.

Bibliography

[1] Computer vision: the ultimate guide on the 4 main tasks; N. Cacciotti; http://www.smart-interaction.com/2022/07/14/computer-vision-the-ultimate-guide-on-the-4-main-tasks/

[2] Object Detection in 2023: The Definitive Guide; G. Boesch; https://viso.ai/deep-learning/object-detection/

[3] Object Detection State of the Art 2022; P. Azevedo; https://medium.com/@pedroazevedo6/object-detection-state-of-the-art-2022-ad750e0f6003

[4] Rich feature hierarchies for accurate object detection and semantic segmentation; Ross Girshick et al.; https://arxiv.org/pdf/1311.2524.pdf

[5] Fast R-CNN; Ross Girshick et al.; https://arxiv.org/pdf/1504.08083.pdf

[6] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks; S. Ren et al.; https://arxiv.org/pdf/1506.01497.pdf

[7] YOLO: Real-Time Object Detection; Redmon et al.; https://pjreddie.com/darknet/yolo/

[8] YOLOv5 by Ultralytics; https://github.com/ultralytics/yolov5

[9] ONNX; https://onnx.ai/

[10] OpenVINO; https://viso.ai/computer-vision/intel-openvino-toolkit-overview/

[11] YOLOv5 model architecture; C. Imane; https://iq.opengenus.org/yolov5/

[12] LabelImg; Tzutalin; https://github.com/heartexlabs/labelImg

[13] Object Detection Metrics With Worked Example; Kiprono Elijah Koech; https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e