POLITECNICO DI TORINO and
UNIVERSIDAD POLITÉCNICA DE MADRID

MASTER's Double Degree in MECHATRONICS ENGINEERING
and INDUSTRIAL ENGINEERING

MASTER's Degree Thesis

Development of a Proprietary AI Model for Road Element Detection: A Comprehensive Approach with Comparative Evaluation

SUPERVISORS

Prof. Roberto GARELLO

CANDIDATE

Ricardo SERRANO SANTA TERESA

CO-SUPERVISOR

Caterina LIA

JULY 2025

Development of a Proprietary AI Model for Road Element Detection: A Comprehensive Approach with Comparative Evaluation

Ricardo Serrano Santa Teresa

ABSTRACT

This thesis presents the design, development, and evaluation of a proprietary object detection model tailored for road surface defect identification, with a particular focus on potholes. Developed in collaboration with the Italian startup LOKI s.r.l. in relation to its project Asfalto Sicuro, the work addresses the limitations of relying on pre-trained third-party models by proposing a fully customized deep learning architecture. Two pre-trained backbones —MobileNetV2 and Darknet-53— were integrated into a YOLO-inspired architecture featuring a split detection head. The model was trained and evaluated on a curated pothole detection dataset using extensive data augmentation techniques, including translation, cropping, mosaic patterns, and perspective distortion. Performance was benchmarked against state-of-the-art models (YOLOv8 and Faster R-CNN).

ACKNOWLEDGMENTS

I would like to first thank professor Garello for giving me this opportunity to work on a subject that aligns so much with my interests.

I am also really grateful for the help received from Caterina, who has guided me along the whole process, giving me very valuable input for this work. Same gratitude goes to Francesco and Giovanni, and to LOKI as a whole. They have provided me with not only the time and resources necessary for the project, but with a positive and welcoming attitude that is much appreciated.

Table of Contents

List of Figures

Chapter 1

Introduction

This thesis aims to explore and evaluate the potential of a proprietary object detection model trained to detect road defects, in comparison with pre-trained model available online. The work was done in collaboration with LOKI s.r.l., a young Italian startup, in relation to its project Asfalto Sicuro, which intends to improve the safety of the roads by automatically detecting and mapping the main road defects, as potholes, cracks, and alligator cracks.

1.1 Motivation of the project

As previously outlined, the Asfalto Sicuro project is dedicated to the detection and geolocation of road surface anomalies, such as potholes and cracks. The primary data source for this task consists of road images acquired through a camera system mounted on a moving vehicle. To enable automatic identification of surface defects from these images, an object detection algorithm is employed. Currently, the detection pipeline leverages pre-trained models developed by third parties.

In the following section, we examine the technical and strategic motivations for pursuing the development of a proprietary object detection model tailored to the specific requirements of the project. Some of the main concerns are shown in the Table 1.1.

Aspect	**Custom Model**	**Pre-trained Model (e.g. YOLOs)**
Ownership & Licensing	Full control, no external dependencies or license restrictions.	AGPL or commercial licenses, usage may require attribution or fees. [1]
Cost	High initial cost (R&D, compute, validation).	Low upfront cost, recurring license fees may apply (e.g. Enterprise license for YoloV8).
Flexibility & Customization	Fully adaptable architecture and feature set.	Limited to provided structure, customization constrained.
Optimization	Tuned for specific data/hardware, supports pruning, quantization, etc.	Pre-optimized for general cases, adaptation needed for custom deployments.
Deployment	Can target edge/mobile efficiently.	Often requires conversion or simplification.
Maintenance	Updates fully managed internally.	Dependent on third-party updates and support.
Vendor Lock-in	None, infrastructure independent.	Risk of dependency on providers or APIs.
Legal Risks	Lower risk, licensing is fully controlled.	Higher risk, misuse of open licenses can have legal implications. [2] [3]

Table 1.1: Pros and cons of developing a custom object detection model versus using a pre-trained model, such as the available YOLO versions.

1.1.1 Control of the training process

First of all, the task of detecting road defects inherently requires a customizable model, one that can be trained on domain-specific data and fine-tuned to achieve the desired performance. This is true for most object detection tasks that aim at detecting a specific element or set of elements which are not included in the generic datasets. The most common technique for this kind of task is to take a network that was trained on a generic dataset and retrain it using the new data that the model is supposed to learn from. This is done by unfreezing either some or all of the layers of the Neural Network, so that the weights can change and adapt to the new data.

Building a proprietary model allows not only to retrain it if needed, but also to change the whole architecture of the model. In this way, it is much easier to implement new features if needed further down the line. Some possible examples of these changes are: adapting the model for a new type of data, adding other kinds of predictions, using the trained parts for other purposes, or adding custom filters in any of the stages. It also has the potential to be fine-tuned with full control of the process.

1.1.2 Ownership and independence

From a company's perspective, full ownership of the model used provides significant strategic advantages. It allows the company to be much more independent from any change in the environment, reducing its reliance on third-party providers. Machine Learning has long been characterized by a strong open-source culture, but there has been a shift towards a more closed approach in recent years. This is mainly due to the presence of more and more complex models, which employ significant resources to develop. One of the consequences is that most advanced detection models are licensed under fees for commercial use. Owning a proprietary model would not only eliminate these costs but also mitigate the risk of any change in said fees. The availability of these models can also change, limiting their use or making it more opaque to the user, which leads to less control of the model.

For instance, Table 1.2 shows the licenses that apply to some of the most common YOLO models. The first models are generally less restrictive in their use compared to the new ones.

YOLO Version	**License**	**Commercial Use Allowed**
YOLOv1–v3	Public Domain	Yes
YOLOv4	GPLv3	No (copyleft, must open-source your code)
YOLOv5	AGPLv3 (code), Ultralytics License (models)	Limited (requires license for SaaS or closed-source use)
YOLOv6	Apache 2.0	Yes
YOLOv7	GPLv3	No (copyleft)
YOLOv8	AGPLv3 (code), Ultralytics License (models)	Limited (same as YOLOv5)

Table 1.2: License types and commercial use allowance for YOLO models

1.1.3 Customization and flexibility

Another significant advantage of developing a custom detection model lies in the increased flexibility and independence from any specific third-party platform. Relying on pre-trained models often entails constraints related to the underlying infrastructure, software compatibility, or licensing restrictions. In contrast, a proprietary model can be adapted and optimized to suit various deployment environments, regardless of their computational characteristics.

For example, if the goal is to perform inference on edge devices —such as embedded systems or mobile hardware with limited processing power— the model architecture can be modified accordingly. Techniques such as model quantization, pruning, and knowledge distillation can be employed to reduce the computational load and memory footprint without substantially compromising performance. This enables efficient real-time processing even in resource-constrained environments.

Considering these advantages, the development of a proprietary object detection model represents a compelling direction that aligns with both technical requirements and long-term system adaptability.

1.2 Goals and outline

The main goal of this project is to develop the custom model and be able to compare it to other related models, such as R-CNN and YOLOv8. Therefore, here are the points that should be achieved:

- Designing and implementing a custom, trainable model architecture.
- Identifying and/or constructing suitable datasets to enable comparative evaluation, specifically for the task of pothole detection.
- Training state-of-the-art models on the curated dataset for benchmarking purposes.
- Training the proposed model using the same dataset to assess its performance.
- Comparing the performance of the developed model against existing approaches.

1.3 Thesis outline

This report will take the following structure:

- **State of the Art:** This section presents an overview of the different technologies involved in this thesis. It mainly addresses methods of road defect analysis and the field of object detection.

- **Materials and methods:** This section describes in detail the characteristics of the models both used and developed, as well as the resources employed for it.
- **Results:** Here all the data obtained will be presented and discussed, focusing on the comparison between models and their suitability.
- **Conclusions:** Some brief conclusions about the whole project and future related work.

Chapter 2

State of the art

2.1 Road defect detection

2.1.1 Fields of interest/applications

Road conditions play a critical role in ensuring traffic safety for all users. Among the various types of surface degradation, potholes are some of the most frequent and hazardous defects. They can cause significant damage to vehicles and pose a heightened risk of accidents, especially for road users with less stability and protection, such as cyclists and motorcyclists. According to Cycling UK, 15% of the cyclists they assist after crash-related injuries were involved in incidents caused by road surface defects [4]. Additionally, potholes are a leading cause of suspension and tire damage in passenger vehicles, often resulting in costly repairs and increased accident risk during evasive maneuvers or braking [5].

However, safety is not the only concern related to road defects. Any road incidence affects traffic flow, resulting in higher delays, bigger road jams, slower traffic, and lower efficiency. Some research suggests that poor road conditions result in a decrease of vehicle speed by 55% and the average exhausted emission increases by 2,49% [6].

These are some of the reasons why authorities and maintenance companies have focused on identifying these defects to later fix them. This maintenance has typically been reparatory, usually needing a report of an accident or strong disturbance of the road to then take note of the incident and repair it. This process is slow and inefficient, which has led the sector to gradually adopt a more preventive approach. However, prevention relies on early assessment of road conditions and timely detection of relevant defects. Performing this task manually through human labor is not feasible, which is why automatic detection technologies have emerged as a valuable tool.

These detection systems are also employed in other fields. For instance, the rise of self-driving vehicles has created a demand for real-time detection capabilities, which has significantly driven the advancement of these technologies. They are valuable not only in fully autonomous vehicles [7], but also in driver-assistance systems. When fast enough, such systems can serve as active safety mechanisms, identifying defects that may pose a risk to the vehicle and enabling appropriate responses.

2.1.2 Current established technologies

After reviewing the importance of defect detection in ensuring road safety, the next step is to examine the technologies and approaches currently available to support this process. This work focuses on the detection of potholes, as they are among the most common pavement defects and have a considerable impact on overall road conditions. Nevertheless, the approaches reviewed also include other types of defects, given the significant overlap in the underlying technologies, many of which are designed for multipurpose detection.

Most methods discussed in the literature adopt an image-based approach, relying on 2D visual data in either RGB or grayscale format. These images are often captured using mobile cameras, which are both widely available and cost-effective. Additionally, the abundance of well-established image detection techniques and pre-trained models makes it possible to adapt existing solutions to this task, providing a strong foundation for further development.

2.1.2.1 Traditional 2D image processing

There is a wide range of methods that can be used to detect defects based on just image data. The review at [8] provides an analysis of some of the traditional image-based methods used for pavement assessment, showcasing their advantages and limitations. As an example, the work done in [9] focuses on three of these techniques and it portrays how they can be used for difficult tasks such as road fine-defect detection.

Traditional image processing usually includes thresholding algorithms, which consist of applying a composition of filters to the image's pixels and then introducing a threshold to determine the most probable areas where the image would show defects. These areas can then be used to perform detection or segmentation in the image. Those filtering functions are mostly hand-crafted or chosen between options that have been proven to be effective. Some of the most common thresholding algorithms are Otsu's thresholding [10], triangle thresholding [11], and adaptive thresholding [12].

In the review [8] there is also a thorough analysis of many classification methods and their performance for pavement defects. The majority of them are supervised machine learning algorithms such as Support Vector Machine (SVM) or simple Linear Regression (LR), but it also includes some unsupervised learning methods such as k-means clustering. It even mentions some use of dynamic neural networks used for threshold selection.

However, most of these methods have one limitation in common: they are very sensitive to foreign objects. Therefore, the images used for these studies rarely include anything other than the pavement. Some works propose a pre-processing solution for this limitation by first isolating the road with a crafted mask, feeding then to the algorithm only the pavement region. An example of this masking is shown in Figure 2.1.

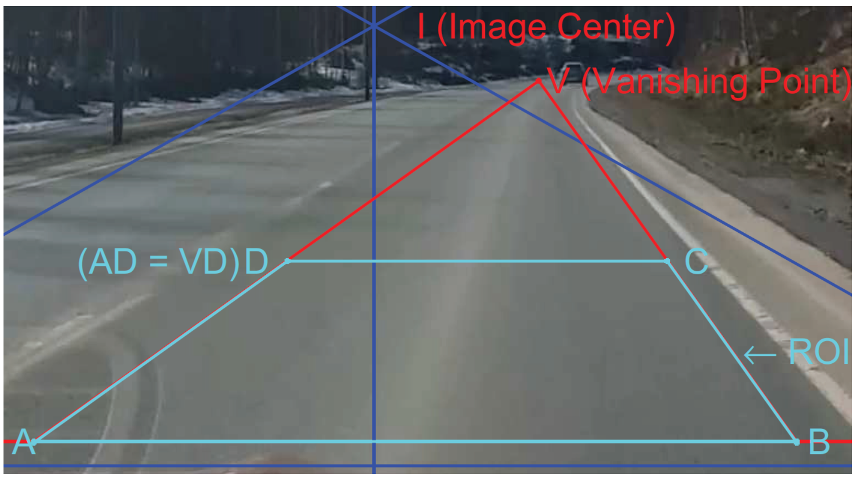


Figure 2.1: Example of masking a Region of Interest by selecting a vanishing point as reference.Taken from [12].

They are also quite sensible to lighting conditions, which adds complexity to the pre-processing stage. When the images come from a moving vehicle in a real traffic scenario, both foreign objects and different lighting conditions are present. Therefore, a more robust method is required to avoid a large pre-processing step to isolate and adapt the pavement part of the image.

2.1.2.2 Deep Learning 2D image processing

Some of the traditional image processing approaches reviewed earlier include machine learning algorithms such as SVM and LR. A subset of machine learning methods that has shown remarkable performance in tasks such as image classification and object detection is known as Deep Learning (DL). DL models are composed of multiple layers of neural networks capable of learning complex, hierarchical patterns from raw input data. For image-based tasks, Convolutional Neural Networks (CNNs) are the most commonly used architecture, as they are specifically designed to handle spatially structured data such as images. CNNs use convolutional layers with learnable filters to automatically extract relevant features from the input while preserving spatial relationships. These filters are typically followed by non-linear activation functions, pooling layers, and normalization steps, progressively increasing the level of abstraction. This enables CNNs to capture low-level features such as edges in early layers, and high-level semantic features such as shapes and objects in deeper layers, making them highly effective for image classification, detection, and segmentation tasks. An example of a simple CNN diagram is shown in Figure 2.2.

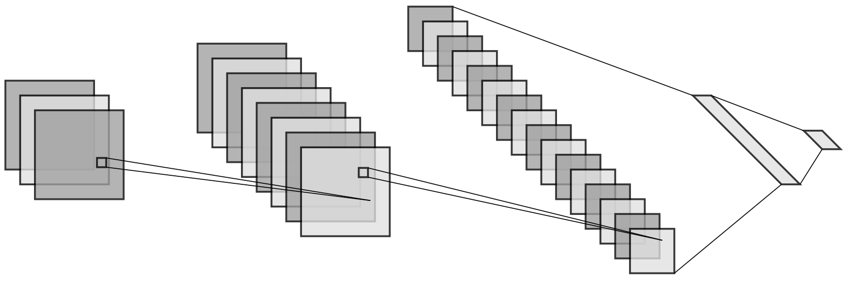


Figure 2.2: Simple CNN architecture. Generated using NN-SVG [13].

The inclusion of these methodologies made substantial progress in pothole detection [14]. For instance, in [15] a mean average precision of 0.75 is obtained for pothole detection with the use of CNN based models. The images used for the training are street shots, which contain foreign objects such as vehicles, street signs and pedestrians. They were even capable of deploying the detection in real-time scenarios, given how efficient these kinds of models can be. One of the key advantages of these models is their versatility. They often learn to identify multiple types of objects, which makes them able to discern irrelevant data present in the images.

Given the huge development of DL object detection in recent years, their use in the field of road defect detection has also grown, adapting some of the latest models to the task. There are many examples of the use of the three main technologies, which are Single Shot Multibox Detectors (SSD) [16] [17], YOLO models [18] [19] [20] [21] [22], and Region based CNNs (R-CNN) [23] [24] [25].

Since this is the technology that will be employed for the project, its details will be covered later, mainly in Section 2.2 and Section 3.

2.1.2.3 Stereo images

Still using an image-based approach, the use of stereo cameras provides better 3D information, resulting in higher robustness in detection and better precision for depth estimation [26] [27].

However, this method has its drawbacks, such as the difficulty in handling reflective or texture-less surfaces. For this technology to be accurate, the images must have some features to use as references for the 3D estimation. The deployment of a stereo camera also often requires manual calibration, which is case-dependent and would need periodic maintenance. Regardless, stereo cameras remain a popular choice for depth estimation due to their affordability, simplicity, and effectiveness within their limitations [28].

2.1.2.4 Depth sensors

In recent years, there has been a growing interest in using depth maps or point clouds, which are usually less sensitive to changes in illumination conditions [28]. These alternatives give a better 3D understanding of the elements detected, which help evaluate how severe a defect can be.

However, these approaches require other kind of hardware, such as laser scanners [29] [30] or LiDARs [31], which can render them impractical for certain projects due to constraints like cost, device complexity, and operational considerations. For instance, LiDAR systems often involve high equipment and maintenance costs, while laser scanners may require precise calibration and stable environmental conditions to function effectively. Additionally, the deployment of such systems may not be feasible in projects with limited budgets, challenging terrain, or tight portability requirements.

2.2 Object detection background

Object detection is a field of Computer Vision (CV) where the goal is to determine instances of objects (road defects in this case), figuring out their location inside an image. It is exploited in many applications such as robot vision, autonomous driving and video surveillance. CNNs have played a vital role in the development of CV, and especially in Object Detection tasks [32].

Its flourishing has greatly benefited from the emergence of parallel computing systems, allowing to perform intense training tasks more efficiently.

The most common use of CNNs for object detection tasks can be grouped in three types of models: R-CNNs, YOLO models, and SSDs.

2.2.1 R-CNN

Region based CNNs are the most common region-based architectures. They are a three-step process:

- Selective Search process is first used to propose a number of regions, potential locations where the image is more probable to have an object to detect.
- After that, a CNN extracts features from each of the proposed regions. The proposed areas must be adapted to the expected size of the CNN to be able to take them as input, usually achieved by warping and resizing each region.
- Finally, SVMs are applied to determine if an object is or not present in each region. In the case of a multiclass model, there would be a SVM for each of the classes to detect. A regressor is used to adjust the coordinates of the detected box in each region.

Once the model has selected which boxes contain an object, some post-processing is used, being Non-Maximum Suppression (NMS) the most common as well as some containing filters.

Some variations of this architecture were later developed to improve either accuracy, efficiency or sensibility. Figure 2.3 shows a diagram of some of the various R-CNN architectures. For instance, Fast R-CNN introduces a region of interest pooling layer before the CNN. This makes it possible to directly feed the image into the CNN, instead of feeding each of the proposed regions.

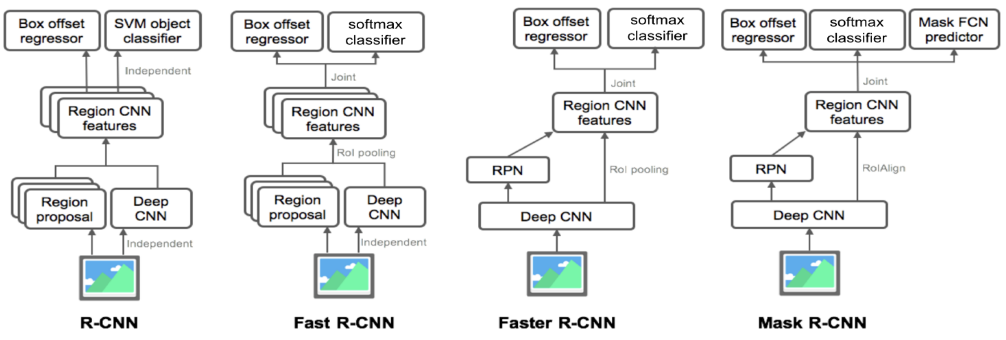


Figure 2.3: Comparison of R-CNN architectures. Taken from [33].

Both the original R-CNN and Fast R-CNN use selective search to propose regions, which acts as a bottleneck for fast computation. This caused the development of Faster R-CNN, which carries the task of proposing regions by using another network: Region Proposal Network, a fully convolutional network.

Mask R-CNN [34] is quite similar to Faster R-CNN, adding a third branch to the output of the network. Instead of having just a class label and a bounding-box offset, it also includes a mask that represents the object, providing more information on the detection.

2.2.2 YOLO models

YOLO (You Only Look Once) takes its name from the fact that the image only passes once through the network, therefore being a single-stage detector. The input image is split into a square grid of equal size cells (commonly 13x13), and it predicts the same number of bounding boxes for each of them. Along with the box coordinates, the model also outputs a confidence score and a class prediction for each of the boxes, as shown in Figure 2.4. A threshold is set to only take the most confident predictions, thus obtaining the detected boxes.

All of these predictions come from the CNNs feature extractor, which is then fed into another network to predict at the same time the confidence and the coordinates of the boxes.

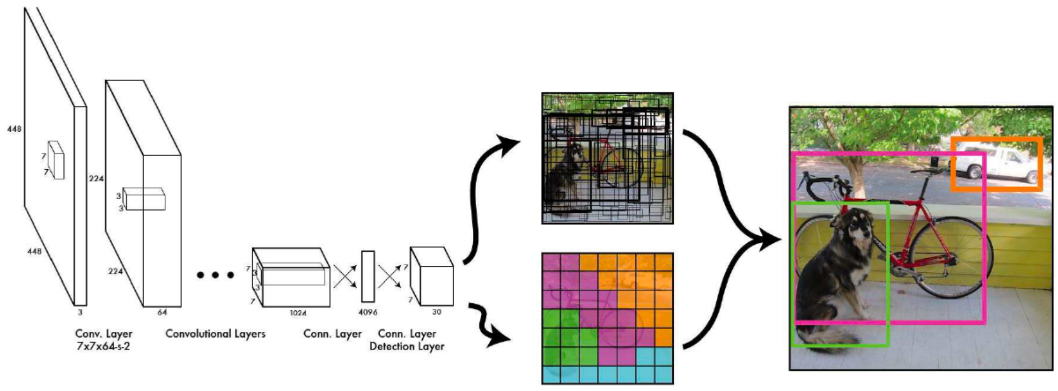


Figure 2.4: Yolov1 architecture diagram. Taken from [35].

One of the main limitations of the original YOLO architecture is its difficulty in accurately detecting small objects. This issue arises from the coarse spatial resolution of the final feature maps used for prediction, which may not capture fine-grained details necessary for identifying small targets. To address this, later versions of YOLO introduced multi-scale prediction strategies. These involve generating outputs from multiple grid sizes, each corresponding to different stages of the feature extraction pipeline. Using intermediate feature maps with higher spatial resolution alongside deeper, more abstract representations, the model improves its ability to detect objects of varying sizes, particularly smaller ones.

2.2.3 SSDs

Single Shot multibox Detectors are also single-stage, but they aim at solving the issue of YOLO in relation to small objects. They make use of multiple feature maps at different resolutions, similar to some of the later versions of YOLO. However, they focus on matching the predictions to predefined anchor boxes at each of the feature maps, instead of letting the model predict boxes in each of the predefined gridcells. They are slightly slower than YOLO, and they add a couple of extra convolutional layers due to the multiple detection. A comparison of both architectures is shown in Figure 2.5, taken from the publication where this technology was first presented [36].

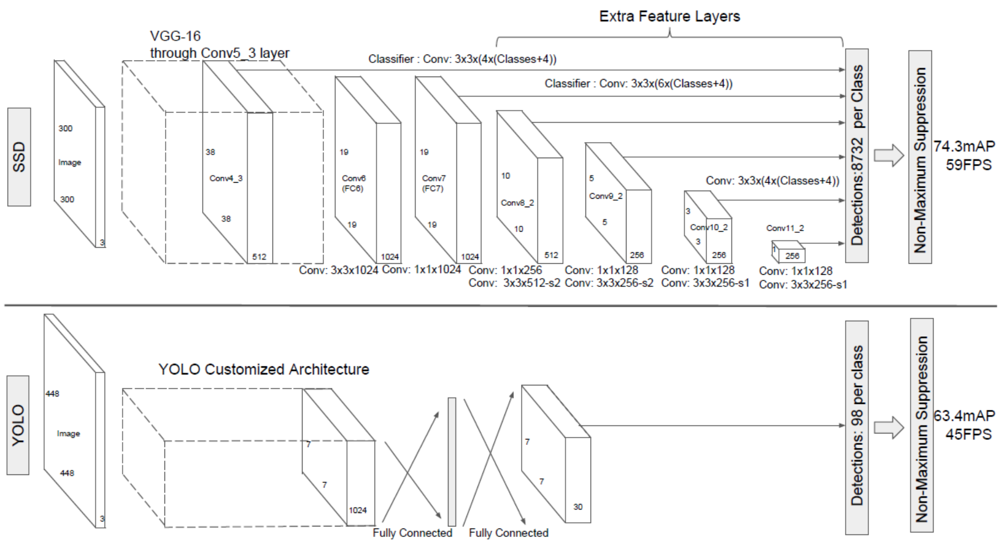

SSD
YOLO
VGG-16 through Conv5_3 layer
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256 Conv: 3x3x512-s2
Conv: 1x1x128 Conv: 3x3x256-s2
Conv: 1x1x128 Conv: 3x3x256-s1
Conv: 1x1x128 Conv: 3x3x256-s1
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS
YOLO Customized Architecture
Fully Connected
Fully Connected
Detections: 98 per class
63.4mAP 45FPS

Figure 2.5: Comparison of YOLO and SSD architectures. Taken from [36].

2.3 Belonging of current project

As previously mentioned, this project relies on RGB images as the primary data source.

To perform pothole detection, a version of the YOLO algorithms will be employed. The goal is to investigate the capabilities of YOLO's architecture in this context, evaluate its performance on the available dataset, and explore strategies for data adaptation to maximize the model's effectiveness.

Chapter 3

Materials and methods

Building a DL model requires defining two main elements: the architecture of the neural network and the data on which it will be trained. Another key aspect is the method used to train the model, which will depend on both the data used and the structure of the system.

- The **architecture** defines how each layer and component —such as convolutions, activation functions, or pooling layers— of the CNN is connected. It is the core of the system, and it will determine the complexity, capabilities, efficiency, and overall suitability for the task.
- The **data** used to train and evaluate the model determines how much information it will be exposed to, being even more important when some part of the network has not been trained before. It is essential to take into account not only the amount, but also the quality of the data that is fed into the model. As it will be later discussed, there are techniques to improve the exposure of the model to this data, so that it can extract as much information as possible.

3.1 Architecture

As previously stated, the model developed in this work was inspired by the architecture of the YOLO family of object detectors. Therefore, it is relevant to briefly examine the evolution and core components of the original YOLO architecture.

YOLOv1, introduced in 2016 [35], proposed a novel approach to object detection by framing it as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation. Its backbone was inspired by GoogLeNet [37], but replaced the inception modules with a combination of 1×1 convolutions followed by 3×3 convolutional layers, as detailed in [38]. The feature extraction pipeline also included max-pooling layers to progressively reduce spatial dimensions while preserving semantic information.

In its original design, YOLOv1 concluded with two fully connected layers, responsible for producing the final predictions. However, later versions abandoned these dense layers in favor of a fully convolutional design, which improved generalization and allowed variable input sizes. This shift not only enhanced efficiency but also aligned with modern deep learning practices favoring convolutional operations throughout the network. Another key characteristic common in the YOLO family is the use of LeakyReLU as an activation function for convolutional layers. The difference with the standard ReLU is that it has a slope in the negative part, in opposition to the standard ReLU which maps any negative input to zero. This feature avoids neurons "dying" and becoming useless when a section of the network goes to negative numbers during training. By introducing a small slope, the neuron remains active improving gradient flow and therefore obtaining a better convergence.

The architecture of YOLO algorithms varies throughout versions, but the core structure and design components have remained the same. As shown in Figure 3.1, it is composed by a backbone, a neck and a head [39]:

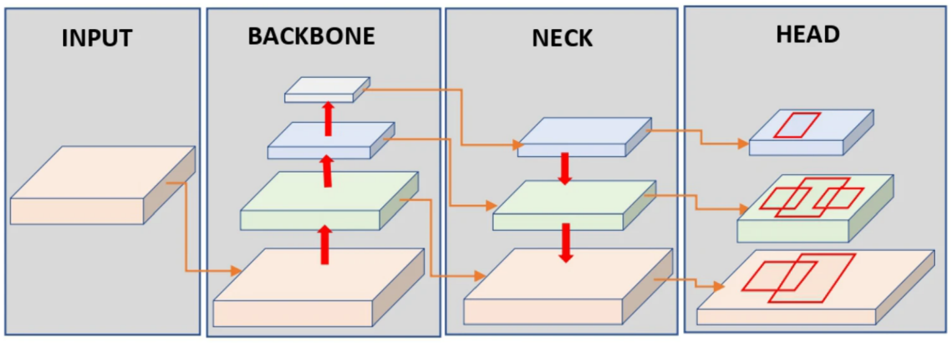


Figure 3.1: Diagram of YOLO's core structure. Taken from [40].

- **Backbone:** It is a CNN that performs the feature extraction, obtaining different scale feature maps from the input images. It heavily uses the 1×1 and 3×3 convolutional block mentioned before. Some of the main backbones used throughout the models are the DarkNet19, first employed in YOLOv2 and Darknet-53, used in YOLOv3 to improve precision.

- **Neck:** A network that fuses and merges the extracted features from the backbone to obtain spatial information, which can later be used for the detection or segmentation task. Depending on the version, it can take input from various stages of the backbone, or just from the last feature map output.

- **Head:** The head is responsible for refining the object detection, taking the feature map information and giving the final shape to the predictions. Depending on the model, there can be multiple detection heads that predict objects at different grid sizes.

3.1.1 Design considerations

Before exploring the selection of each component in the network and properly understanding the structure, the shape of the predictions must be described.

The proposed models follow the YOLO-style bounding box structure, where the output consists of a grid over the input image, and each cell predicts a fixed number of bounding boxes. Each bounding box encodes the spatial coordinates, an objectness score —which reflects the confidence that an object is present—, and a predicted class label for the detected object.

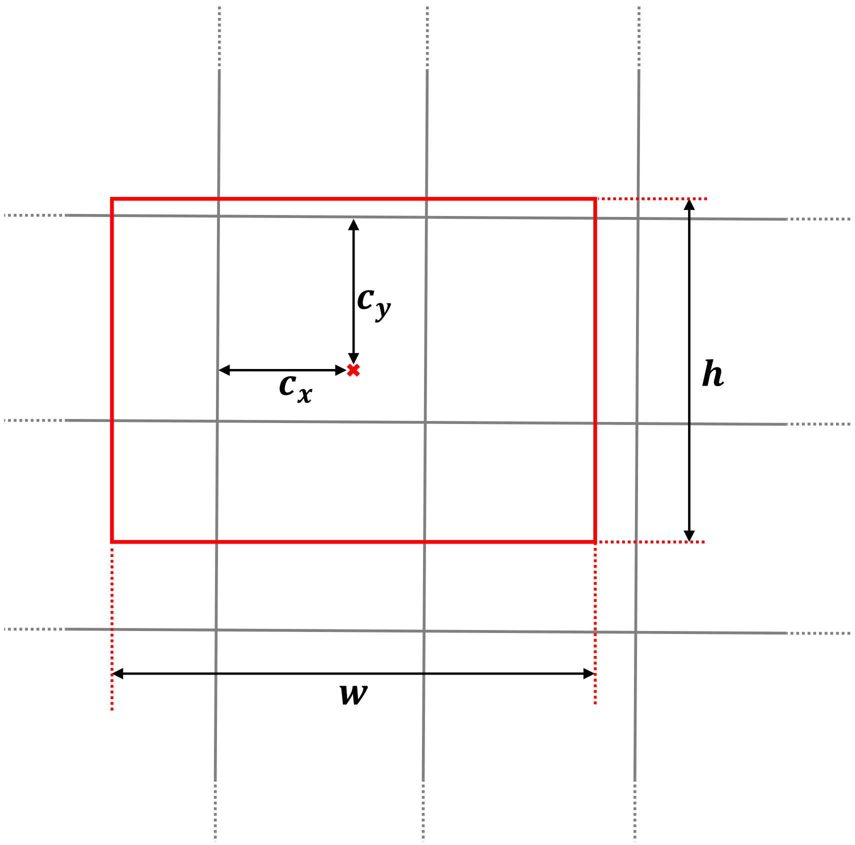


Figure 3.2: Bounding box coordinates, relative to grid cell.

Each bounding box prediction can be therefore seen as an array with the structure:

$$\{\mathbf{c_x}, \mathbf{c_y}, \mathbf{w}, \mathbf{h}, \mathbf{conf}, \mathbf{label}\}$$

- $\mathbf{c_x}$ and $\mathbf{c_y}$ encode the center coordinates, ranging $[0, 1]$. These coordinates are relative to the grid cell, so that the spatial representation of each cell is the same. As shown in Figure 3.2, $(c_x = 0, c_y = 0)$ would mean the box is in the top left of the cell and $(c_x = 1, c_y = 1)$ would be the bottom right.

- **w** and **h** represent the *width* and *height* of the bounding box, where $(w = 1, h = 1)$ refers to a bounding box with the size of the full picture.

- The objectness term —**conf**— also ranges $[0, 1]$, as commonly done in these models.

- The **label** term can be encoded as a one-hot array, where each class has a confidence prediction ranging $[0, 1]$ and the one with the higher activation is assigned to the bounding box.

Once the structure of each predicted bounding box has been defined, it is important to consider other design elements that shape the model's output behavior. Some of those key aspects are discussed below.

- Although the developed model structure allows multiclass detection, the tests have been done using it as a single-class detector, focusing only on potholes.

- The size of the output grid is set at 7×7, just like the first version in YOLOv1. Although later versions make use of larger grid sizes —varying from 13×13 in YOLOv2 up to 80×80 in YOLOv8—, the single-class nature of the model implies that fewer objects are expected per image, reducing the need for widely distributed spatial attention.

- A total number of 2 boxes is predicted in every grid cell. Relying again on the fewer objects expected in each image, the need for a larger number of predicted boxes is not supposed to be a concern.

- Since the experiments have been done using a single class, the feature of the one-hot class encoding remains inactive.

- To define which are the final predicted boxes, a threshold must be chosen.

Any box whose objectness is higher than the threshold will be considered a predicted object. However, the definition of this parameter is done after the training, so it belongs closer to the testing stage.

3.1.2 Backbone selection

For the backbone used in this work's model, there have been two main paths explored: MobileNetV2 and Darknet-53.

3.1.2.1 MobileNetV2

MobileNetV2 was presented in 2018, where the authors discuss how a simple network architecture allowed them to build a family of highly efficient mobile models [41]. Their basic building block —the bottleneck depth-separable convolution with residuals— allows for memory-efficient inference while relying on using standard operations present in most neural frameworks.

Understanding the architecture begins with the depthwise separable convolutions, which decompose a standard convolution into two simpler operations: a depthwise convolution followed by a pointwise convolution.

In a standard convolution, each output channel is computed by applying a $k \times k$ filter across all input channels, resulting in a computational cost of:

$$H \cdot W \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot k^2$$

In contrast, a depthwise separable convolution first applies a $k \times k$ filter to each input channel independently (depthwise convolution), followed by a 1×1 convolution that combines these outputs across channels (pointwise convolution). The combined cost is:

$$H \cdot W \cdot C \cdot (k^2 + C_{\text{out}})$$

This results in a significant computational reduction at only a small accuracy trade-off. The key difference lies in separating spatial filtering (depthwise) from channel mixing (pointwise), making this approach highly efficient for lightweight networks.

The term bottleneck refers to the architectural pattern within this block. Unlike traditional bottleneck layers that first reduce dimensionality, MobileNetV2 blocks initially expand the number of channels, apply depthwise separable convolution, and then project the result back to a lower-dimensional space. This expansion–compression structure preserves model expressiveness while keeping the computational cost low.

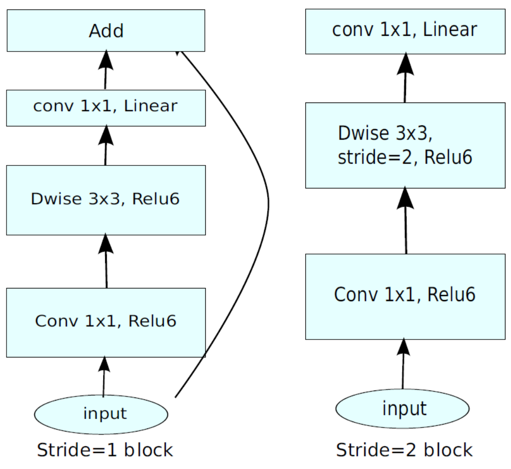


Figure 3.3: Diagram of MobileNetV2's blocks. Taken from [41].

The residual connection is a skip path that directly adds the input of a block to its output. This technique is used to enable better gradient flow during training and to preserve information across layers. For a residual connection to be used, the input and output of the block must have the same spatial and channel dimensions. Therefore, as shown in Figure 3.3, MobileNetV2 only uses the residuals when the stride is $s = 1$.

Putting all of these design concepts together, the structure of the block is the one shown in Table 3.1.

Input	**Operator**	**Output**
$h \times w \times k$	1×1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times (tk)$	3×3 dwise $s = s$, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times (tk)$	linear 1×1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 3.1: Bottleneck residual block, transforming from k to k' channels, with stride s, and expansion factor t. Adapted from [41].

The whole MobileNetV2 architecture is the one described in Table 3.2, where each line describes a sequence of 1 or more identical layers, repeated n times. The first layer is a standard convolution composed of 32 filters. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is applied as described in Table 3.2.

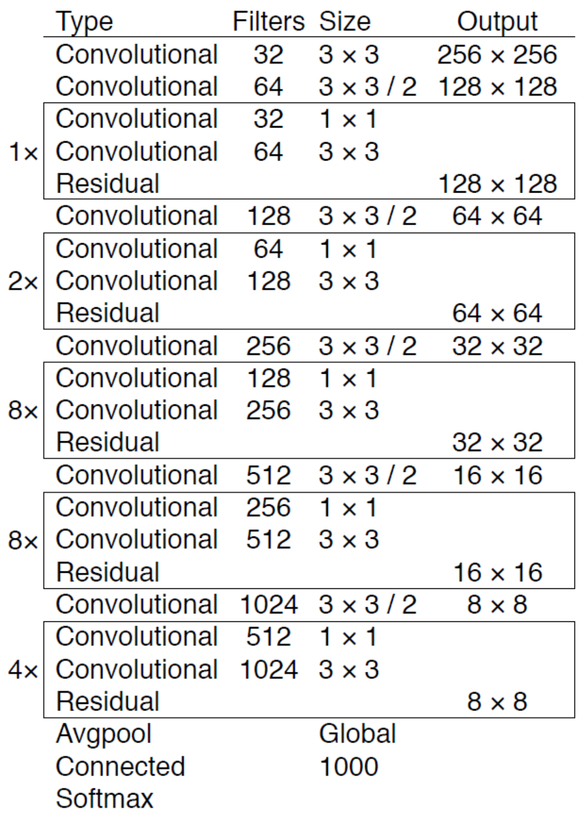
Input	**Operator**	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1
$7^2 \times 1280$	avgpool 7×7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1×1	-	k	-	-

Table 3.2: MobileNetV2 architecture. Adapted from [41].

This network was considered as an option for the custom architecture based on simplicity, efficiency and proven ability to extract relevant features of images to perform object detection. Efficiency would become particularly useful if the model if it is ever used for inference on edge devices.

3.1.2.2 Darknet-53

Darknet-53 was first introduced in 2018 as the backbone for YOLOv3. As explained in the original paper [42], it is a hybrid approach between the Darknet-19 used in YOLOv2 [43] and the new residual network designs that were coming up at that time. It uses the successive 3×3 and 1×1 convolutional layers characteristic of the YOLO family, while adding some shortcut connections. The whole structure is shown in Figure 3.4 and results in a network much larger than the Darknet-19, but also much more powerful at feature extraction. It allowed YOLOv3 to perform on par with the models that were state of the art at that time while better utilizing the GPU structure and, for instance, being 3 times faster than the SSD variants.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.4: Architecture of Darknet-53 as used in YOLOv3. Taken from [42].

Although more recent YOLO versions have shifted toward alternative backbone architectures, Darknet-53 remains a foundational component in the evolution of YOLO models. YOLOv4 further built upon this structure with CSPDarknet-53, incorporating Cross Stage Partial connections to enhance efficiency while preserving representational power.

While later models such as YOLOv5, YOLOv6, YOLOv7, and YOLOv8 adopted new design paradigms, the legacy of Darknet-53 is evident in their shared focus on optimizing both performance and computational cost.

In this project, Darknet-53 was selected as an option for the backbone due to its proven ability to extract multi-scale features effectively, its compatibility with YOLO-style detection heads, and its relative simplicity compared to newer architectures. This makes it a practical and reliable choice for road defect detection, particularly in scenarios where computational resources are limited.

3.1.3 Common architecture blocks

There are some building blocks common the architectures employed that will be later referenced when explaining each option, so they must be first described in detail.

3.1.3.1 Conv layer

Any convolutional layer mentioned in the custom architecture refers to a 2D convolution with n filters and stride $s = 1$ as default. If not stated otherwise, the kernel size is 3×3. They include a weight regularizer ($L2$) and sometimes include a dropout to avoid overfitting, but it will be specified when it is in use. Just like most YOLO models, LeakyRELU is applied as an activation function after every convolutional layer (except for the last one, which will be explained) with a slope of $\alpha = 0.1$ for the negative range.

Normalization is applied before the activation function, although some of the last layers do not include this part. The normalization chosen is the *Layer Normalization* present in the *tensorflow* framework, even though YOLO models commonly use batch normalization for this purpose. The reasoning behind this choice is the computational limitation for the training stage. To fully exploit the computational power of the GPU used, the models are trained in a batch of $n = 1$. This would make a batch normalization useless, so the layer normalization is chosen as its performance is not dependent on batch size.

3.1.3.2 Residual block

This structure is very similar to the one used in Darknet-53, composed by a block of two convolution layers that is repeated N times. The first one has a kernel of 1×1 and $\frac{n}{2}$ filters, while the second one has the standard 3×3 kernel and n filters. The result these convolutions is latter added to the input of the block, which is what gives it the residual character, as shown in Figure 3.5.

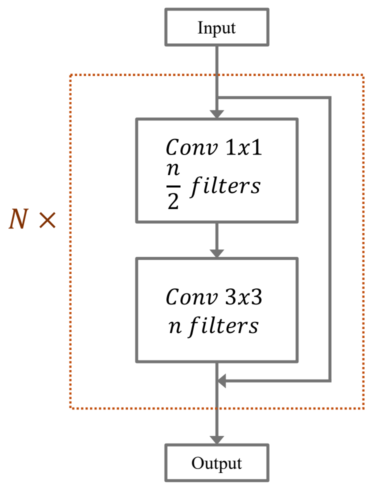


Figure 3.5: Residual block diagram.

3.1.3.3 Bottleneck block

A similar approach is followed in the custom bottleneck block used in the proposed architecture, which aims at changing the dimensionality of the input from c_1 to c_2 in the output while reducing the computation needed. As shown in Figure 3.6, this block also consists of two consecutive convolutional layers: the first one applies a 1×1 kernel to reduce the number of channels from c_1 to a smaller intermediate value $c' = e \cdot c_2$, and the second one uses a 3×3 kernel to project the features back to c_2 channels. If the number of input and output channels match ($c_1 = c_2$), the input is added to the output of the two convolutions, forming a residual connection that facilitates gradient flow during training.

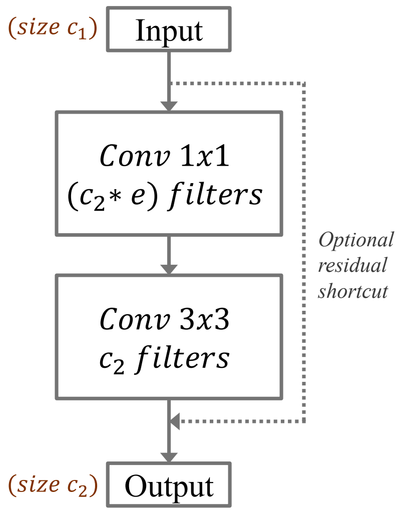


Figure 3.6: Bottleneck block diagram.

3.1.3.4 Activation stage

The last layer of any of the heads is where the final activation functions are placed. It takes the output of the previous convolution layers and gives it the final shape. It splits the final array to perform the following activation functions:

- **Sigmoid** to the (c_1, c_2) center coordinates, so that it favors movement of the box center inside each cell.
- **Exponential** function for the *width* and *height*, so that the scale of change remains the same no matter the size of the box.
- **Sigmoid** to the *objectness* as it is common practice for any value that represents confidence of a prediction (also used for class *labels* if a multiclass

model is employed).

3.1.4 Head selection

The detection head is the last part of the model, where predictions are given their final shape taking as input the extracted feature maps. It takes the feature map already with the final grid size (7×7) and a high depth —measured as number of filters coming from the feature map—, reducing its depth until it has the desired final shape. In case of single-class detection, this last size is:

$$\mathbf{n}_{GridCells} \cdot \mathbf{n}_{BoxesPerCell} \cdot \mathbf{n}_{ParametersPerBox} = \mathbf{(7 \cdot 7) \cdot 2 \cdot 5}$$

Throughout the work, different architectures for the detection head have been considered. Based on the YOLO family, the main consideration is to keep a simple architecture that can be efficient, making use of the building blocks that have been proven great performance in other models.

3.1.4.1 Split head

This architecture first reduces the depth with two Bottleneck blocks and then with a convolution with kernel 3×3, decreasing the number of filters at each stage. It then splits the result to perform different operations to the coordinates and to the confidence, as shown in Figure 3.7.

It uses three convolutional layers for both cases. The first two convolutions employ a kernel size 1×1 for the confidence to favor isolation of prediction in terms of which cell gets contains an object. For the coordinates, these two first convolutions have a kernel size 3×3 and only the last one has kernel 1×1. This aims to retain spatial awareness, supposing that information of the surroundings of the cell will remain relevant to predict the position and size.

In this case, the first four convolution layers use a small dropout. It goes from 0.01 in the first one to 0.005 for the rest. The last two convolutions both for coordinates and for confidence do not include normalization, so that the network has freedom to predict different results depending on the input picture, not forcing the range of predictions.

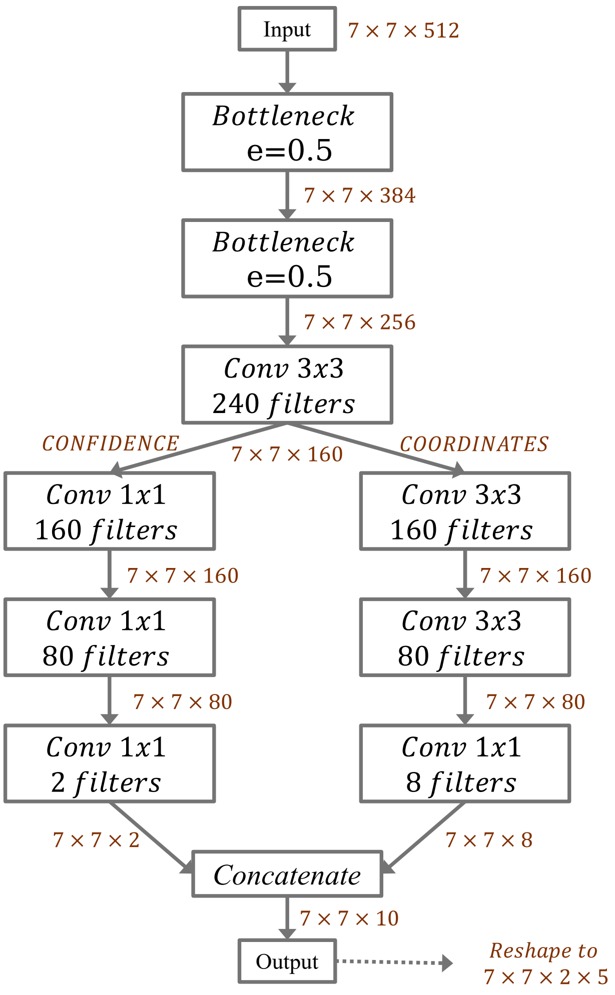


Figure 3.7: Split head structure diagram.

3.1.5 Neck selection

Early versions of YOLO (v1 and v2) used minimal or no explicit neck structure to connect the feature map to the detecting head, relying instead on direct connections between feature maps and the output layers. These architectures performed detection on a single scale, limiting their ability to capture objects of varying sizes. Subsequent versions, beginning with YOLOv3, introduced more sophisticated neck designs, such as Feature Pyramid Networks (FPN) and later Path Aggregation Networks (PANet), enabling multi-scale feature fusion for improved performance, particularly in detecting small objects. Recent versions, such as YOLOv7 and YOLOv8, have further refined neck structures to enhance efficiency and adaptability.

However, in this work, the model architecture follows a simpler approach akin to the original YOLO versions, with only minimal adaptation layers bridging the backbone and detection head. Their main purpose is to adapt the backbone output dimension to the input of the detecting head ($7 \times 7 \times 512$) while avoiding to lose feature information, by using either bottleneck or residual blocks to reduce the depth and some convolutional layer with stride of 2 or an average pooling layer to reduce grid size if needed.

3.1.6 Post-processing

To obtain the final predictions, a final stage processes the network's output. As previously noted, only the bounding boxes with confidence scores above a specified threshold are retained. This threshold can therefore control the sensitivity of the model.

In models such as YOLO, where there are many predictions for each region of the picture, it is common to find boxes that belong to the same object. To avoid redundant predictions, two functions are included to the end of the predicting pipeline: greedy NMS and a containment filter. These functions are not trainable, they both depend on a threshold parameter that set either manually or by evaluating the best performing for each case.

3.1.6.1 Greedy NMS (Non-Maximum Suppression)

Non-Maximum Suppression removes boxes with a high overlap between them, assuming that two boxes with a high IoU (Intersection over Union) are probably reacting to the same instance of an object.

Greedy NMS is the most straight-forward method of this kind, and its logic is performed as follows:

- All selected boxes are sorted in descending order based on their confidence score.
- The first box is kept. The algorithm then computes the IoU between that box and the rest. Any box with a value higher than the threshold is removed, like the one shown in Figure 3.8.
- The same step is then performed until every box is either kept or discarded.

It is common to set the IoU threshold at 0.5, but for this case the value is set at 0.4. Based on the nature of the detection, where two conjoined defects can either be considered as one or two distinct potholes, it would be expected that two boxes with some overlapping are referring to the same defect. That is the reasoning behind using a stronger NMS.

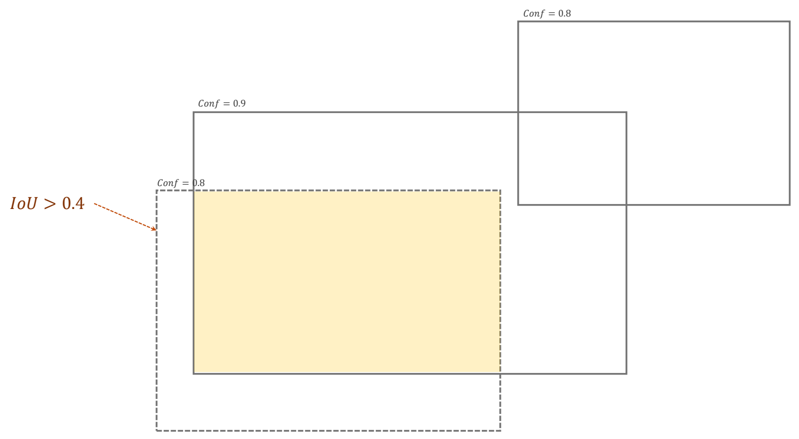


Figure 3.8: Example of box discarded by greedy NMS.

3.1.6.2 Containment filter

NMS is able to discard many redundant predictions, but there are some clear cases that it fails to identify, like the ones shown in Figure 3.9. For instance, there could be a box fully contained in another one with a significant difference in size. If the smaller box is a fraction of the bigger one lower than the NMS threshold, it would remain after applying greedy NMS. A containment filter is applied to filter those kinds of cases.

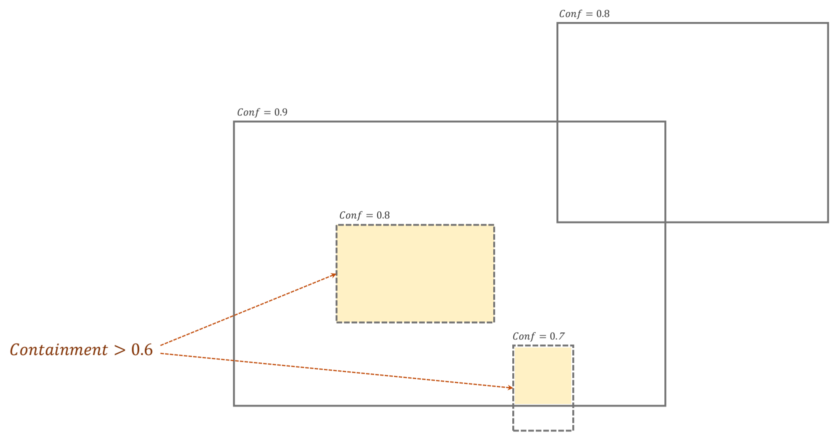


Figure 3.9: Example of boxes discarded based on containment.

Containment refers to the fraction of the inner box's area that is inside an outer box. This filter checks the containment ratio for every pair of boxes. If the fraction is higher than a set threshold, the bounding box with the highest confidence is kept.

Although it is common to use high threshold values for the containment, ranging from 0.8 to 0.85, for this work the threshold was set at 0.6. The stronger filter follows the same reasoning as the one chosen for the NMS.

3.2 Training characteristics

This section provides a detailed overview of the training characteristics and parameters used in the development of the proposed model. Key elements of the training configuration are discussed, including the choice of loss function, learning rate scheduling strategies, optimizer settings, and other hyperparameters critical to the model's convergence and performance. These aspects were selected and adjusted with the aim of promoting robust learning while avoiding overfitting and optimizing generalization capabilities. The criteria behind these choices are outlined in the following subsections.

CNNs are trained like like most neural networks, by exposing the network to labeled data that serves as *ground truth*, consisting of the input data —*the image*— and the expected output the network should produce —*the bounding boxes*—. For every batch of data, the network predicts an output that is compared to the ground truth. A *loss function* then computes a scalar that quantifies how far the prediction is from the ground truth. This loss is used to guide the adjustment of the network's weights through the *gradient descent* algorithm, which updates each weight in proportion to its contribution to the overall error.

The term *hyperparameters* refers to variables that can be varied to tune the training process. For this type of model, the most critical hyperparameters include the *batch size*, the *learning rate*, and the *loss function weights*, which can influence the model's convergence and performance.

3.2.1 Loss function

In object detection tasks, the loss function plays a critical role in guiding the learning process by quantifying the discrepancy between the predicted bounding boxes and the ground truth. Most object detectors, including region-based and single-stage models, employ composite loss functions that combine multiple objectives, such as bounding box regression, objectness scoring, and classification accuracy.

Specifically, YOLO models utilize a multi-part loss function comprising:

- **Localization loss**: Measures the difference between predicted and true bounding box coordinates. Early YOLO versions (e.g., YOLOv1) used mean squared error (MSE), but this was later refined to better handle scale invariance and overlapping boxes.

- **Confidence loss**: Penalizes incorrect objectness predictions.
- **Classification loss**: Quantifies the difference between predicted class probabilities and the true class labels, typically using cross-entropy.

Later YOLO versions introduced anchor boxes and used more robust loss functions like binary cross-entropy (BCE) for confidence and classification, and IoU-based metrics (e.g., GIoU, DIoU, CIoU) for localization. These advanced loss functions help the model better capture the overlap between predicted and ground truth boxes and improve performance on small or overlapping objects.

For this work, the loss is computed by summing the weighted contributions of the following terms: *coordinates loss*, *detected object loss*, *not detected object loss*, and *class loss.* Each of these terms has an associated term λ_i:

$$\mathbf{Loss} = \lambda_{\mathbf{coord}} * \tilde{\mathbf{L}}_{\mathbf{coord}} + \lambda_{\mathbf{obj}} * \tilde{\mathbf{L}}_{\mathbf{obj}} + \lambda_{\mathbf{noobj}} * \tilde{\mathbf{L}}_{\mathbf{noobj}} + \lambda_{\mathbf{class}} * \tilde{\mathbf{L}}_{\mathbf{class}}$$

3.2.1.1 Coordinates loss L_{coord}

This term computes how different the localization of the predicted boxes is from the ground truth. Although the first versions of YOLO models used simple approaches such as MSE, this work focused on IoU-based approaches. The final loss function uses Complete Intersection over Union (CIoU), which is considered one of the best options to promote effectiveness and convergence. CIoU is computed between a predicted box and a true box as follows:

$$\text{CIoU} = 1 - \text{IoU} + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{\text{gt}})}{c^2} + \alpha v$$

where:

- IoU represents the Intersection over Union between the predicted bounding box $\mathbf{b}$ and the ground truth bounding box $\mathbf{b}^{\text{gt}}$.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- $\rho(\mathbf{b}, \mathbf{b}^{\text{gt}})$ denotes the Euclidean distance between the centers of the predicted and ground truth boxes.

$$\rho^2(\mathbf{b}, \mathbf{b}^{\mathrm{gt}}) = (b_x - b_x^{\mathrm{gt}})^2 + (b_y - b_y^{\mathrm{gt}})^2$$

- c is the diagonal length of the smallest enclosing box covering both $\mathbf{b}$ and $\mathbf{b}^{\mathrm{gt}}$.
$$c^2 = (c_w)^2 + (c_h)^2$$

- v measures the similarity of aspect ratios between the two boxes.
$$v = \frac{4}{\pi^2}\left(\arctan\frac{w^{\mathrm{gt}}}{h^{\mathrm{gt}}} - \arctan\frac{w}{h}\right)^2$$

- α is a positive trade-off parameter that balances the aspect ratio term.
$$\alpha = \frac{v}{(1 - \mathrm{IoU}) + v}$$

CIoU combines overlap area, distance between box centers, and aspect ratio similarity providing a more comprehensive localization loss. This makes it more effective than traditional IoU or even generalized IoU (GIoU), particularly in promoting stable and efficient convergence during training.

The coordinates term is only added to the loss at the cells where at least one true box exists, since there are no boxes to compare for the rest of the cells.

To improve the accuracy of the localization penalty, a *box matching* technique is employed. This approach begins by computing the IoU between each predicted box and all the true boxes associated with the same grid cell. Among these, the true box with the highest IoU is selected, and only the CIoU computed for this best-matching box is used in the loss calculation. This strategy effectively compares each predicted box to its most similar true box, ensuring that the model is not penalized for diverse predictions that may occur among different boxes within the same cell.

A term is later added that computes the center distance. That is, the euclidean distance between the center of the predicted box and its assigned true box. This term is weighted by a constant λ_{center}, which is set at a low value compared to the rest of the terms. The aim of this term is to add a constant "pull" towards the correct localization, no matter what the intersection is.

Both the relevant CIoUs and the center terms are then squared and summed, obtaining the coordinates loss L_{coord} as showed in Equation 3.1

$$L_{coord} = \sum_{i \in \mathcal{C}} \left(\text{CIoU}_i^2 + \lambda_{center} \cdot \rho_i^2 \right) \tag{3.1}$$

where:

- $\mathcal{C}$ denotes the set of grid cells that contain at least one ground truth bounding box.
- CIoU_i represents the Complete Intersection over Union between the predicted box and the best-matching ground truth box in cell i.
- ρ_i^2 denotes the squared Euclidean distance between the center of the predicted box and the center of its matched ground truth box in cell i:

$$\rho_i^2 = (b_{x,i} - b_{x,i}^{\text{gt}})^2 + (b_{y,i} - b_{y,i}^{\text{gt}})^2$$

- λ_{center} is a weighting factor for the center distance term, typically set to a small constant value.

3.2.1.2 Detected object loss L_{obj}

The detected object loss, denoted as L_{obj}, penalizes incorrect objectness confidence predictions. It combines two components: one based on assigned predictions, and one as a fallback for unmatched ground-truth slots. Specifically, the loss is computed as:

$$L_{obj} = \sum_{i} \left(m_i (1 - \hat{c}_i)^2 + o_i (1 - r_i)(1 - \hat{c}_i)^2 \right) \tag{3.2}$$

where:

- i indexes grid cells and predicted boxes.
- $\hat{c}_i$ is the predicted objectness confidence for box i.
- m_i is the *assigned mask*, indicating whether box i has been matched to a ground truth box.
- o_i is the *object mask*, indicating whether box i corresponds to a location where a ground truth object exists.

- $r_i = \max_k r_{i,k}$ represents the maximum responsibility score over potential matches for box i, indicating whether it was assigned any responsibility for matching.

The first term, $m_i(1 - \hat{c}_i)^2$, penalizes incorrect confidence for matched boxes (i.e., those with a clear assignment to ground truth). The second term, $o_i(1 - r_i)(1 - \hat{c}_i)^2$, supervises unmatched ground truth boxes, acting as a fallback by penalizing low confidence in locations where ground truth objects exist but no assigned match was found. This hybrid approach ensures robustness by supervising both matched and unmatched predictions, thereby improving the model's objectness prediction capabilities.

3.2.1.3 Not detected object loss L_{noobj}

The loss associated with predictions at locations where no object is present is denoted as L_{noobj}. This term penalizes high objectness confidence scores in grid cells or boxes that have not been assigned to any ground truth object. It is computed as:

$$L_{noobj} = \sum_i \left((1 - m_i)\hat{c}_i^2 \right) \tag{3.3}$$

where:

- i indexes grid cells and predicted boxes.
- m_i is the *assigned mask*, indicating whether box i has been matched to a ground truth object.
- $\hat{c}_i$ is the predicted objectness confidence for box i.

This loss applies to boxes where $m_i = 0$ (i.e., not matched to any ground truth box) and penalizes predictions with high confidence scores in regions where no object exists. This encourages the model to minimize false positives by suppressing incorrect predictions in empty areas of the image.

3.2.1.4 Class loss L_{class}

Although class loss is not used for the experiments of this work given the single-class nature, it is included in the model in case of a multiclass detection is desired. The associated term is noted as L_{class} and, as most classification models, it would use a categorical cross-entropy loss, computed as:

$$L_{class} = \sum_i o_i \cdot \mathcal{L}_{CE}\left(\mathbf{y}_i, \hat{\mathbf{p}}_i\right) \tag{3.4}$$

where:

- i indexes grid cells and predicted boxes.
- o_i is the *object mask*, indicating whether a true object is present in box i.
- $\mathbf{y}_i$ is the one-hot encoded ground truth class vector for box i.
- $\hat{\mathbf{p}}_i$ is the predicted class probability vector for box i.
- $\mathcal{L}_{CE}$ represents the categorical cross-entropy loss:

$$\mathcal{L}_{CE}\left(\mathbf{y}_i, \hat{\mathbf{p}}_i\right) = -\sum_c y_{i,c} \log \hat{p}_{i,c}$$

3.2.1.5 **Total** *Loss*

After computing each individual loss term, they are normalized and combined into the total loss L_{total}, as expressed in Equation 3.5. The terms corresponding to object-containing boxes (e.g., L_{coord}, L_{obj}, L_{class}) are normalized by dividing by the total number of grid cells containing at least one true box, denoted as N_{obj}. Conversely, the term associated with background (i.e., L_{noobj}) is normalized by dividing by the total number of grid cells without any true box, denoted as N_{noobj}. This normalization ensures that each predicted box, whether for an object or background, contributes proportionally to the total loss, regardless of the number of boxes present in a particular image.

The total loss is computed as:

$$\mathbf{Loss} = \lambda_{coord}\frac{L_{coord}}{N_{obj}} + \lambda_{obj}\frac{L_{obj}}{N_{obj}} + \lambda_{noobj}\frac{L_{noobj}}{N_{noobj}} + \lambda_{class}\frac{L_{class}}{N_{obj}} \tag{3.5}$$

where:

- L_{coord} is the coordinate (localization) loss, as defined in Equation 3.1.
- L_{obj} is the object confidence loss, as defined in Equation 3.2.
- L_{noobj} is the background (no-object) confidence loss, as defined in Equation 3.3.

- L_{class} is the classification loss, as defined in Equation 3.4 (set to zero in this case).
- λ_{coord}, λ_{obj}, λ_{noobj}, and λ_{class} are weighting coefficients that balance the contributions of each loss component.
- N_{obj} denotes the total number of grid cells containing at least one ground truth object, ensuring proper normalization.
- N_{noobj} denotes the total number of grid cells without any ground truth object.

This formulation maintains a balance between the contributions of different loss components, ensuring that no single aspect disproportionately influences the total loss due to variations in the number of objects or background cells across images.

3.2.2 Learning rate

Another key parameter that affects the training process is the learning rate. It determines the size of the step taken by the gradient descent algorithm when updating the network's weights. Specifically, the learning rate defines how significantly the weights are adjusted in response to the computed gradient of the loss function for each batch of data.

It is common practice to use a learning rate scheduler, which defines the evolution of the training rate value throughout the epochs (the term "epoch" refers to each cycle where the network is trained using all the data). The error is expected to be high at the beginning and diminish as the model improves. Therefore, a high rate is employed at first and decay throughout the training, letting the model explore the solution space in the beginning and using a more precise tuning at the end.

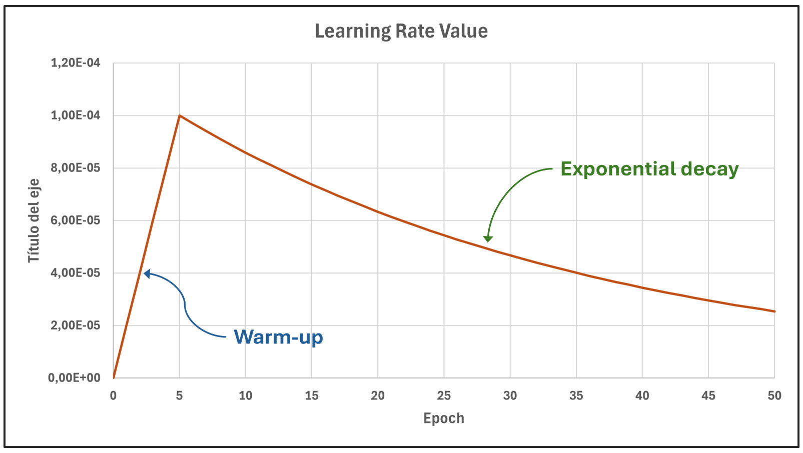


Figure 3.10: Example of learning rate scheduler function with warm-up and decay.

For this training, an exponential decay is used, which is one of the most common functions to use as an scheduler. It updates the learning rate for each batch, decaying in each epoch to 97% of the value of the previous one. Additionally, a warm-up phased is used for the first epochs, which slowly brings the learning rate from 0 to the initial value, using a linear function. This warm-up is particularly useful when a network has not been previously trained preventing the weights from diverging to extreme values, slowly exposing the network to the data. The result scheduler function follows the shape shown in Figure 3.10.

3.2.3 Batch size

Another fundamental parameter in the training process is the batch size, which defines the number of samples processed together in a single forward and backward pass of the model. The choice of batch size affects both the stability and efficiency of learning. Smaller batch sizes can introduce more variability in gradient updates, potentially aiding generalization but slowing convergence. On the contrary, larger batch sizes provide smoother gradient estimates and enable more efficient computation, but they may require careful tuning of other parameters, such as the learning rate, to avoid suboptimal learning dynamics.

Given the limited size of the datasets used, a relatively small batch size was selected, typically ranging from 4 to 16 images per batch. To maximize GPU utilization and maintain computational efficiency, the model was trained using a gradient accumulation strategy. In this approach, the model runs one image at a time, computing and accumulating the gradients for each forward-backward pass. Once the accumulated gradients reach the effective batch size, the optimizer applies the weight update. This technique effectively simulates training with a larger batch size, allowing for full utilization of GPU resources while avoiding memory constraints associated with large physical batches. However, it limits the use of batch normalization for the network layers, as previously mentioned.

3.3 Baseline Models

To evaluate the performance of the proprietary model, some state-of-the-art models will be trained using the same data. When choosing the baseline models the goal is to find established and well-documented technologies that can be trained on custom data, while preserving some similarities to the model evaluated.

3.3.1 YOLOv8

The first baseline model selected is YOLOv8 [44], which was released by Ultralytics on 2023. Although Ultralytics did not publish a formal research paper for it, the model builds upon the advancements of previous YOLO versions, such as YOLOv7 [45]. Some of the key features that it took from previous versions are the following:

- **Multi-scale prediction:** As done in most of the later YOLO versions, the model predicts at different scales, which improves the detection of objects with a wide variety of size. In particular, YOLOv8 predicts at three different scales.

- **C2f blocks:** These blocks build upon the idea of residual connections from Darknet-53 mentioned in Section 3.1.2.2. While residual blocks simply add the transformed and identity paths, C2f splits the input into two parts, processes one with transformations, and then fuses (concatenates) the features. This results in better feature reuse, improved learning efficiency, and a lightweight architecture.

- **Use of SPPF (Spatial Pyramid Pooling-Fast) modules:** This module enhances feature aggregation at multiple scales. As shown in Figure 3.11, it applies a single pooling layer sequentially. It is an attempt at making it lighter than the original SPP module, which was first implemented in YOLOv3 and applied a different pooling layer in parallel for each scale.

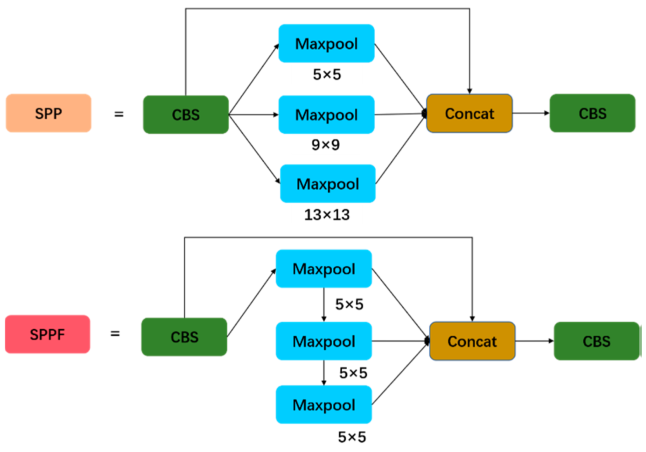


Figure 3.11: Structure of SPP and SPPF. Taken from [46].

- **PAN (Path Aggregation Network) style neck:** This kind of structure was first introduced in 2018 aiming at boosting information flow in proposal-based instance segmentation framework. It uses bottom-up path augmentation, shortening the information path from the lower layers to the top features, as seen in Figure 3.12. [47]

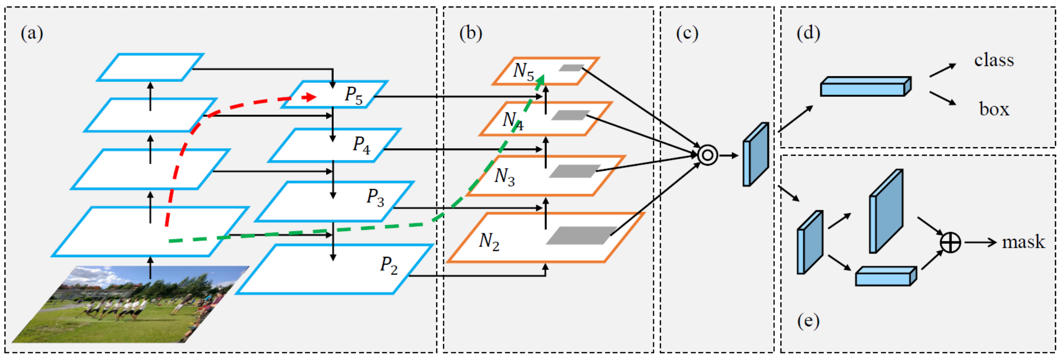


Figure 3.12: Original use of PANet. The bottom-up structure *(b)* shortens the path for the layers coming from FPN backbone *(a)*. Taken from [47].

The strong performance obtained with YOLOv8 has made it a popular option for a wide range of projects and research studies. The framework provided by Ultralytics simplifies the process of training on custom data, which also adds to how widespread it has become. As a result, YOLOv8 has been applied in numerous studies addressing pothole detection, such as [48, 49, 50]. Although these works differ in their datasets and methodologies, they all rely on YOLOv8 as the core detection model.

Both the performance and the wide use of this model on custom data have been the main reasons to adopt YOLOv8 as one of the baseline models for comparison in this work.

3.3.2 Faster R-CNN

As previously mentioned, RCNN have played a huge roll in object detection, and they still are some of the most used architectures. Therefore, adding one of these models as a baseline was considered a good option.

To train it on custom data, the tools provided by Detectron2 were employed. Detectron2 was built by Facebook AI Research (FAIR) to support rapid implementation and evaluation of novel computer vision research, including algorithms such as Mask R-CNN and Faster R-CNN [51]. It simplifies the process of training these models on custom data. For this work, Faster R-CNN was the architecture used.

As mentioned in Section 2.2.1, Faster R-CNN is composed of two models. The first module is a deep fully convolutional network that proposes regions —Region Proposing Network (RPN)—, and the second module is the Fast R-CNN detector [52] that uses these proposed regions. [53]

Some important characteristics of their approach are the following:

- **RPN:** It takes an image as input and outputs a set of rectangular object proposals, with an objectness score. To generate these proposals, they slide a small network over the convolutional feature map. Each sliding window is mapped to a lower-dimenstional feature, which is fed into some fully-connected layers.

- **Anchor boxes:** The proposals are parametrized relative to k anchor boxes, which help with the diversity of aspect ratios. A diagram of this structure is showed in Figure 3.13.

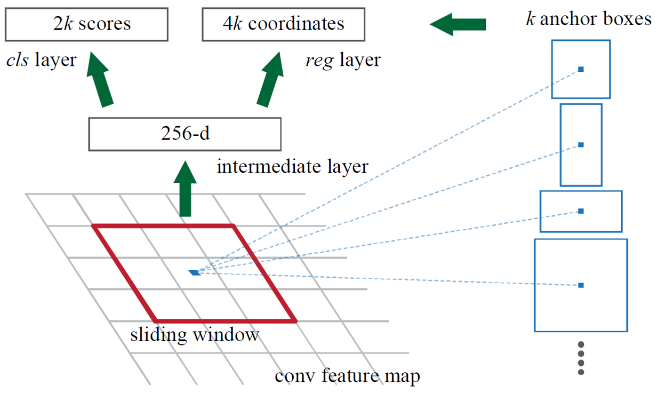


Figure 3.13: Region Proposal Network (RPN) used in Faster R-CNN. Taken from [53].

- **Multi-Scale anchors:** To address the challenge of varying object sizes, R-CNN algorithms had used two main approaches, where either the image was fed into the network at different scales, or the filters were applied with multiple sizes. As shown in Figure 3.14, Faster R-CNN uses a *pyramid of anchors*, using anchors of multiple scales and aspect ratios. Therefore, it only relies on images and feature maps of a single scale and uses filters of a single size.

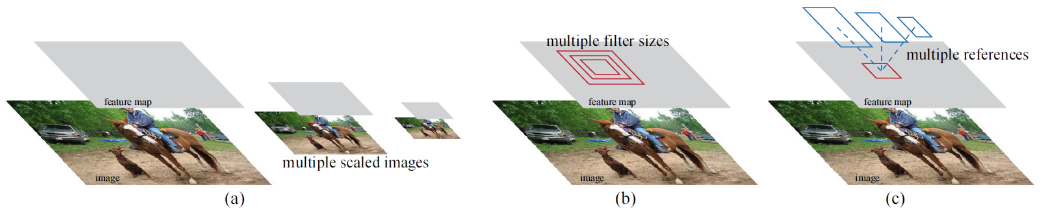


Figure 3.14: Schemes for addressing scales and sizes. (a) Pyramids of images. (b) Pyramids of filters. (c) Pyramids of anchor boxes, approach used for Faster R-CNN. Taken from [53].

The adoption of Faster R-CNN as a baseline model in this work was motivated by its established reputation for accuracy and reliability in object detection tasks. As one of the most influential two-stage detectors, it has served as a reference standard in numerous academic and industrial benchmarks.

3.4 Datasets

There is no benchmark dataset for potholes yet, so the selection of data to use varies from one project to another, depending on the goal and focus of each case.

It is worth mentioning the Road Damage Dataset (RDD2022), which is one of the biggest annotated datasets for road defect detection. It was released as part of the Crowd sensing-based Road Damage Detection Challenge (CRDDC'22), which gathers researchers to propose solutions for automatic road damage detection in multiple countries. [54]

RDD2022 contains four main types of road damage: longitudinal cracks, transverse cracks, alligator cracks and potholes. However, potholes are the least represented of those defects. After filtering all the images from RDD2022 to select just the ones containing potholes in a scenario similar to the desired one, just 1734 images were found suitable to use.

Some of these images that contain potholes are also noisy, including the same reflection for the all the images taken in a country or blocking a considerable part of the view with the car's dashboard. Furthermore, some pothole instances were not labeled as thoroughly as the cracks, which can mislead the model during training. Considering all these reasons, the dataset was not considered suitable for this work.

Although smaller in size, the Potholes Detection Dataset was considered a better option for this project.

3.4.1 Potholes Detection Dataset

Potholes Detection Dataset is publicly available at [55] and consists of 665 annotated potholes. The dataset was created and shared by Atikur Rahman as part of his undergraduate thesis. The original dataset did not contain a validation set, so for this project it was reshuffled into a 70%|15%|10% split (499 images for training, 100 for validation, 66 for testing).

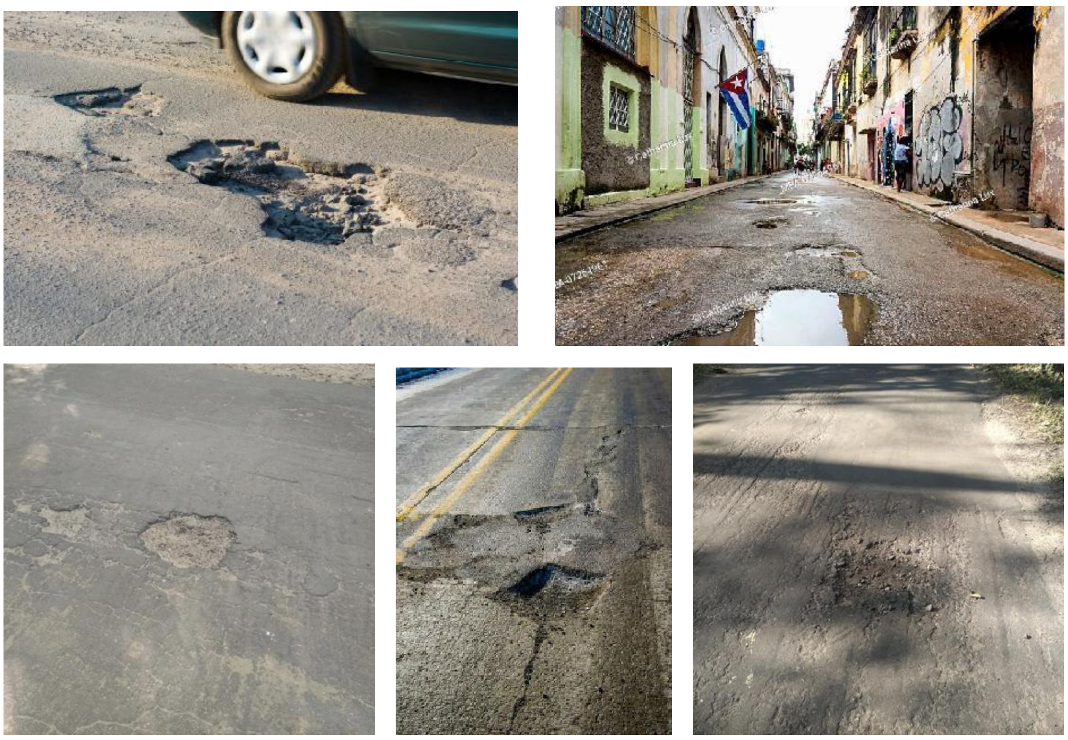


Figure 3.15: Examples of images from Pothole Detection Dataset [55].

This dataset shows a good balance of clear object instances while still appearing in a realistic scenario, some examples are shown in Figure 3.15. It has been used in other similar projects regarding pothole detection, such as [56] which focuses on the use of dilated convolution, or some other works from the same author as the dataset [57].

Given the smaller size of the dataset, it was considered very convenient to apply some data augmentation techniques as part of data pre-processing.

3.4.2 Data augmentation

Data augmentation is a technique used to artificially increase the size of a training dataset, using the original images to create new ones by applying multiple kinds of transformations. It is especially useful to increase the size of small datasets, but that is not the only benefit of using this technique.

By exposing the model to many variations of the original data, it improves the generalization for unseen data. The increased diversity and the bigger input size also help preventing overfitting. In essence, data augmentation allows the model to better utilize the information of the original dataset, getting a more complete feature extraction and a better generalization. This ultimately leads to better performance of the model.

This work has studied the effect of several techniques, whose transformations will now be described. Figure 3.16 showcases an example of each of them.

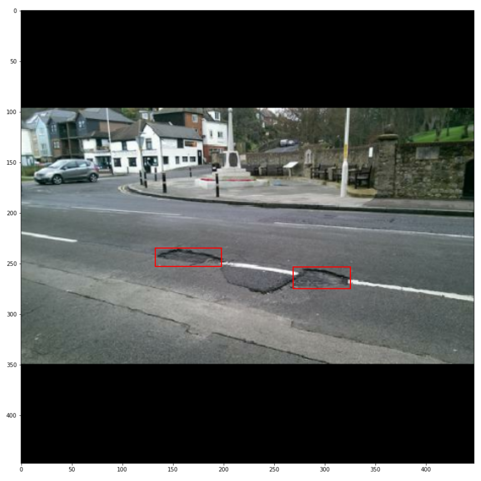
3.4.2.1 Horizontal flipping

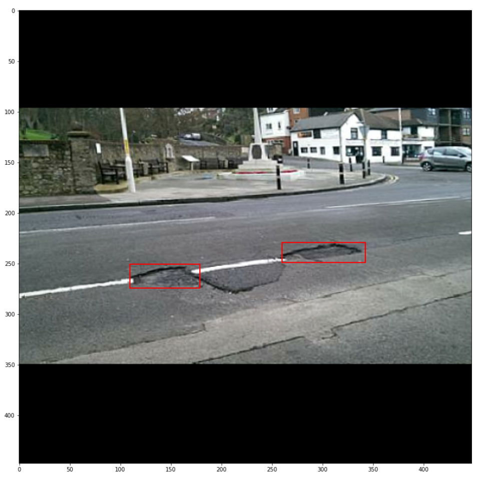
Flipping the image is one of the basic transformations applied for data augmentation. It keeps the training symmetrical over the desired axis. In the case of road images, vertically flipping the image (over a horizontal axis) is not common, since there is some spatial information that must be preserved —e.g.: sky above, road below, potholes facing up—. However, horizontally flipping maintains the proper perspective.

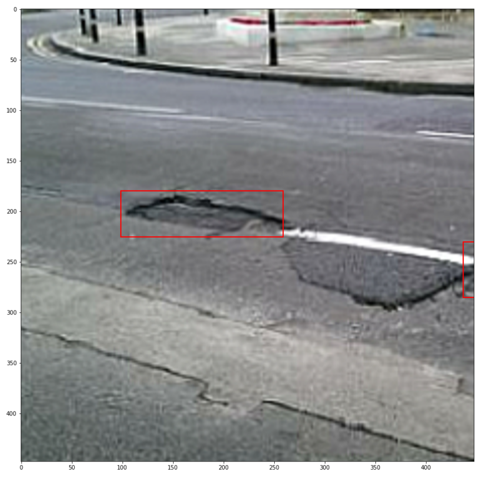
3.4.2.2 Cropping

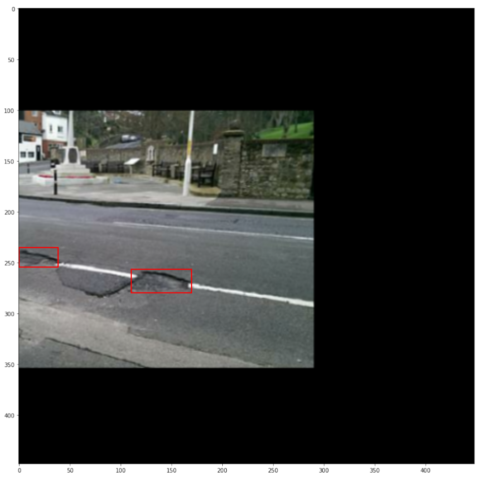
Cropping the image consists on isolating a smaller area of the image. Since the model takes square images as an input, this transformation crops a square section of the original image. This technique serves multiple purposes:

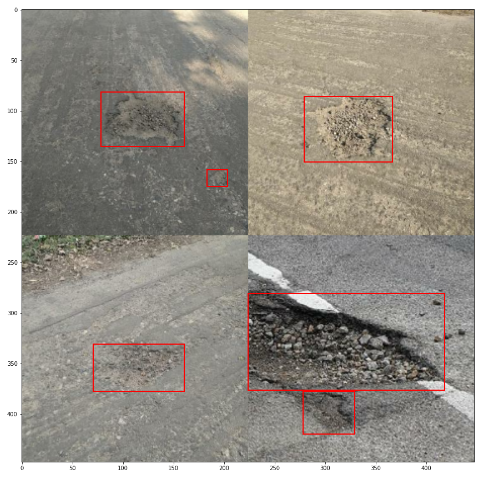
- **Size diversity**: The model is exposed to zoomed smaller sections of the image, which in practice provides an object in a different size.
- **Spatial diversity**: It is common for these kind of datasets to have a bias where the object tends to be towards the center of the image. The cropping can happen in any part of the image, so the object's center can be anywhere in the image. The diversity on these coordinates can help decrease the effect of the bias. It is particularly important for models like the YOLO family, given how each part of the image predicts an object. In this case, the filters might favor one cell over another when a significant bias is present.


(a) Original image


(b) Horizontal flip


(c) Cropping


(d) Translation


(e) 2×2 mosaic

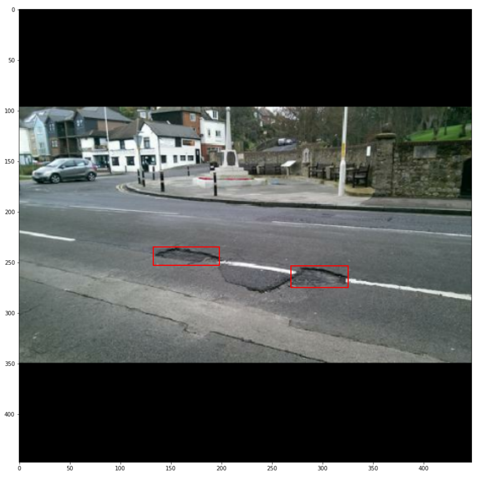

(f) Perspective distortion

Figure 3.16: Examples of transformations used for data augmentation, applied on the same base image (Perspective distortion is hard to notice by sight).

For the tests made, cropping is performed a set number of times for each original image, depending on the influence desired. Each time an image is cropped the coordinates are chosen randomly, either with a uniform function or with a biased function that focuses on the edges.

For this work, the cropping was performed as follows:

- A random zoom is chosen in range $[0.75, 0.9]$, where a value of 1.0 would mean taking the whole image and 0.5 would take a square of half the width or height. If the zoom is not enough to provide a square image without padding, the zoom parameter is set to the minimum necessary for it.

- The coordinates of the cropping are then chosen randomly but biased towards the edges, using a uniform distribution u and computing the offset as $\boldsymbol{\sin^2(\pi u) \cdot max\_val}$. The $\boldsymbol{max\_val}$ parameter is the maximum offset that allows to keep the square cropping in the image.

3.4.2.3 Translation

The translation takes new random center coordinates and shifts the whole image towards them. It uses a uniform distribution with range $[0, 0.4]$ in the horizontal axis and $[0, 0.2]$ in the vertical axis.

The maximum translation in the horizontal axis is greater for two main reasons:

- In a real-life scenario, potholes' position will vary farther horizontally, since it can be from one edge of the road to the other.

- Aspect ratio for this type of images is usually wide, so vertical edges are padded most of the time. It is convenient to expose the model to slight shifts in vertical padding, but it is not as important as exposing the left-most or right-most grid cells to the presence of an object.

Unlike cropping, the image does not change in scale, so after shifting there are empty parts in the resulting image. All these pixels are padded with black.

The benefits of using translation are similar to the ones for cropping. However, the idea here is to force the attention of the model on different grid cells, exposing the same object with same scale to multiple parts of the detection network. When inferring with the model, the non-square images will also be padded with black pixels, so it is convenient to include different paddings in the training stage.

3.4.2.4 Mosaic pattern

Mosaic compositions are very common in the training stage of YOLO models, where multiple images are placed in a square pattern —usually 2×2 or 4×4 patterns—. It addresses the diversity in object sizes, exposing the network to smaller objects throughout the whole space of the image.

The mosaics used in this work have a pattern of 2×2 and were made applying the following steps:

- Four random images are first selected from the training dataset.
- Each image is horizontally flipped as explained above, with a probability of 0.5.
- Each image is then cropped, obtaining a square image with no padding.
- All four images are resized to half the size and placed in the 2×2 pattern.

The resulting image needs no padding, since all images have a full view and are squared. This technique can also help increasing the attention in edge grid cells, as well as preventing the centered bias.

3.4.2.5 Perspective distortion

All images receive a slight random perspective distortion, shifting their corners to a maximum of 12% and adjusting it to the original size. After that, the images are all slightly *sharpened*, enhancing their edges and then blending it with the original image.

This technique aims at preventing overfitting and addressing the different conditions where an image can be taken.

3.4.2.6 Final padding

Images can be taken in any aspect ratio, but the input size of the model is a square image. There are three main options to deal with this mismatch:

- **Cropping** a square portion of the image. It would eliminate part of the image, losing some information. Moreover, cropping is already performed as part of the data augmentation, so its effect is already taken into account.
- **Scaling** all dimensions of the image to fit a square ratio. Although it is common for some object detection models, when the aspect ratio of the

input image is far from square, the distortion of the final image would highly affect the feature extraction.

- **Padding** the edges of the image. It adjusts the longer side to fit the model's input size and pads the remaining space with a constant value —typically 0, i.e. black—. This method is common practice for YOLO models, as it preserves the dimensions of the objects. If the model is exposed to padded images during training, it is able to account for the void space.

Therefore, black padding is performed as described above for any image that is not already squared before feeding it to the model.

Chapter 4

Results

4.1 Overview

In this section, the multiple tests done with each model will be explained, comparing the performance of the proprietary architectures with that of the baseline models. Additionally, the effect of some training parameters in the final performance will be discussed.

4.1.1 Performance metrics

To evaluate the results between different models, there is the need to find a metric that compares the predicted output and the ground truth throughout the whole test dataset. It must be independent from the type of model used, so that any model can be evaluated no matter the architecture employed.

In this case, three standard practices in the field of object detection will be used to evaluate model performance: precision, recall, and mAP@0.5. These metrics provide a balanced assessment of a model's ability to accurately detect and localize objects across the test dataset.

For the baseline models, these are directly obtained via their training framework.

For the custom model, a positive prediction refers to any predicted bounding box with a confidence —*objectness*— higher than a set threshold, which controls how sensitive the model is. It varies throughout the tests, but it always is in range $[0.7, 0.8]$. Those boxes are then processed, as explained in Section 3.1.6, to get the final positive predictions.

True Positives (TP) refer to any bounding box that is predicted where a box is also present in the ground truth. For a predicted box to be matched in these tests, it must share an Intersection over Union (IoU) greater than a threshold with any of the true boxes. The threshold is set to 50% (just like in mAP@0.5). Two boxes are never matched with the same true box for these metrics, as it would lead to faulty results.

Any other predicted box which does not share an IoU greater than the threshold (or any one whose match has already been used for another prediction) is considered a False Positive (FP).

4.1.1.1 Precision

Precision is a metric that measures the accuracy of the positive predictions, computed as a ratio of true positives in relation to all the positives.

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

4.1.1.2 Recall

Recall measures the model's ability to detect all relevant objects in the input images, computed as a ratio of the true positives and all the instances present in the ground truth.

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

False Negatives (FN) refer to any true box that has not been matched to a prediction.

4.1.1.3 mAP@0.5

Precision and recall often work as tradeoffs to each other, where aiming at increasing one often decreases the other. The *mean Average Precision at IoU threshold of* 50% provides a broad assessment of a model's performance, and it takes into account both the precision and recall obtained.

mAP@0.5 summarizes the detector's performance over all predictions by computing the average precision (AP) at a fixed IoU threshold (0.5 in this case), and then averaging it over all classes. Since the current implementation is for a single-class detector, the mAP@0.5 effectively reduces to computing the AP@0.5 for that class.

Average Precision (AP) is computed as the area under the precision-recall curve. To compute it, the 11-point interpolation method is applied. This method samples the precision at 11 fixed recall levels: $\{0.0,\ 0.1,\ 0.2,\ \ldots,\ 1.0\}$. For each recall level r, the maximum precision value observed for any recall greater than or equal to r is selected —$P_{interp}(r)$—. The final $AP@0.5$ is obtained by averaging these 11 interpolated precision values:

$$AP@0.5 = \frac{\sum_{r \in \{0,\ 0.1,\ \ldots,\ 1.0\}} P_{interp}(r)}{11}$$

This approach provides a stable approximation of the area under the curve, smoothing out small fluctuations in the precision-recall plot.

4.2 YOLOv8

This model was trained with the Ultralytics framework, obtaining the following performance metrics:

$$Precision = 0.840 \quad Recall = 0.652 \quad mAP@0.5 = 0.768$$

The architecture used was the YOLOv8s, which comprises around 11.1 million parameters. It was pretrained on the COCO dataset, one of the standard practices for general object detection models. The training for this test was carried out for 50 epochs.

The training pipeline also includes some data augmentation techniques, such as the following:

- Geometric augmentations: horizontal flip, random translation and mosaic augmentation (mosaic deactivated for the last 10 epochs).
- Picture distortions: HSV-based adjustments, blur, grayscale conversion and histogram equalization.

4.3 Faster R-CNN

This model was trained via Detectron2, also for 50 epochs. The Precision-Recall curve is shown in Figure 4.1, obtaining a $mAP@0.5$ of 71.0%.

To have a fair comparison of precision and recall metrics, the precision value

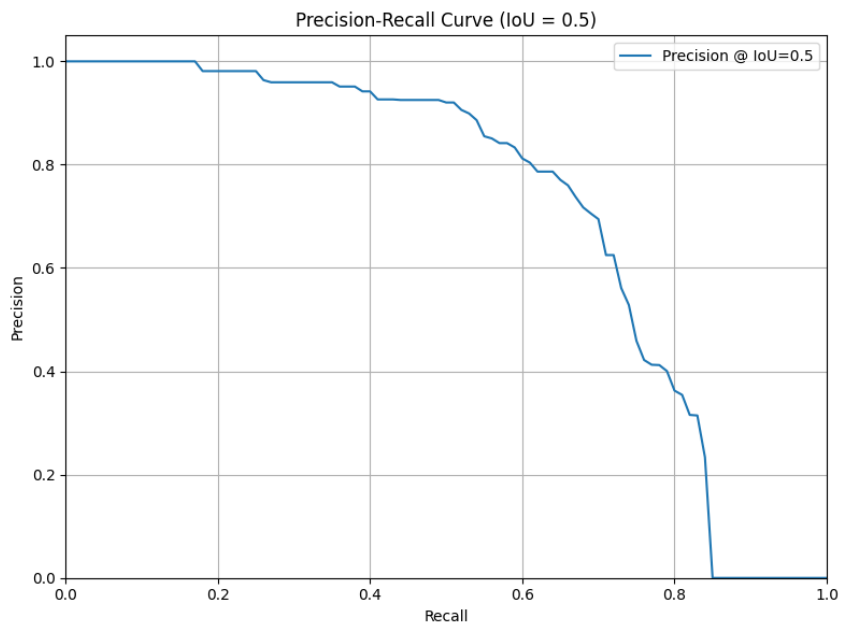


Figure 4.1: Precision-Recall curve obtained for Faster R-CNN.

is taken at the same recall point obtained in YOLOv8's test ($R = 65.2\%$), obtaining the following performance metrics

$$Precision = 0.771 \quad Recall = 0.652 \quad mAP@0.5 = 0.710$$

4.4 Effect of data augmentation

Before carrying out the final trainings of the proprietary model, it is interesting to test the effect of each data augmentation technique, to check which ones are worth adding to the data processing pipeline.

The following training experiments aim at isolating each of these techniques.

All these tests were performed using an input size of $448 \times 448 \times 3$ for the images and the following model architecture:

- MobileNetV2 as a **backbone**, using as output the feature map resulting from the block labeled as `"block_13_expand_relu"`, which is a feature map of output size $28 \times 28 \times 576$. It is common for custom models using MobileNetV2 to extract features from this layer, as it captures abstract semantic features while keeping a moderate spatial resolution.

- **Neck** composed by a residual block —explained in Section 3.1.3.2— followed by a convolutional layer with stride $s = 2$ and an average pooling layer of pool size $p = 2$. This neck aims at reducing the size of the feature grid twice, to achieve the final grid size $7 \times 7 \times f$ that the head can take as input.

- The split architecture explained in Section 3.1.4.1 was used as the **detection head**.

4.4.1 No augmentation

The first test was done with just the original dataset, establishing baseline performance for later comparisons.

$$Precision = 0.269 \quad Recall = 0.215 \quad mAP@0.5 = 0.213$$

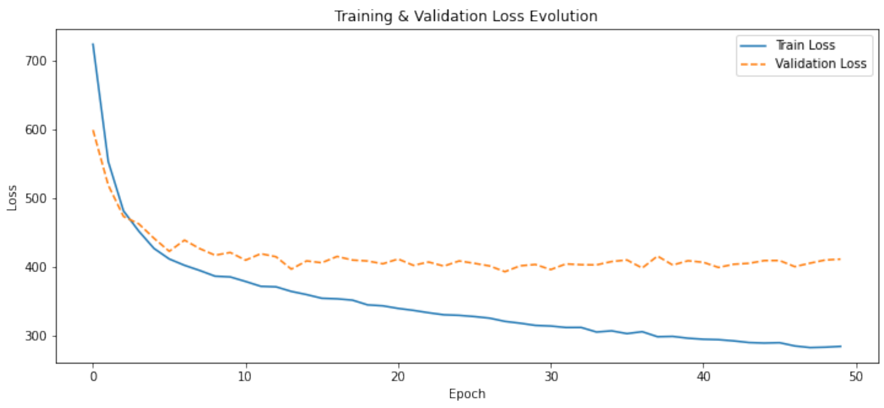


Figure 4.2: Loss evolution when trained with no augmentation techniques.

Other than the performance metrics obtained, the evolution of the loss function throughout the training, shown in Figure 4.2 shows an early overfitting, where the training loss keeps decreasing but the validation loss flattens out. It could be a consequence of the small dataset size.

4.4.2 Horizontal flipping

For this test, every original image was flipped and added to the dataset.

$$Precision = 0.325 \quad Recall = 0.245 \quad mAP@0.5 = 0.247$$

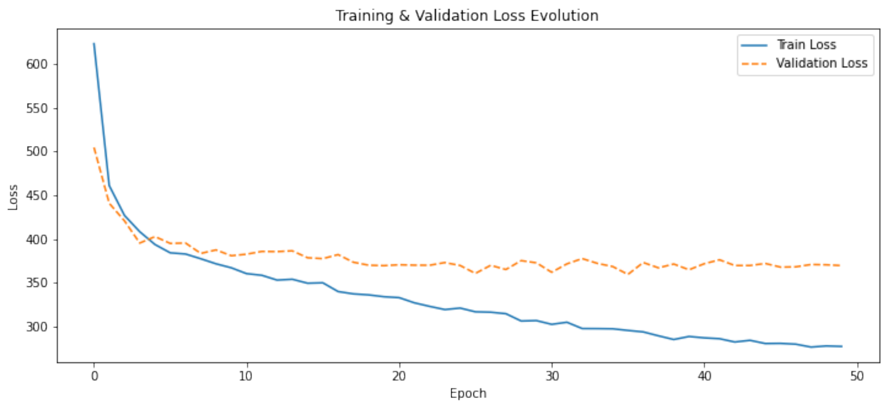


Figure 4.3: Loss evolution when trained just with horizontal flip.

The performance metrics improve, as well as the minimum loss value obtained in the validation set. The validation loss evolution shown in Figure 4.3 also diverges from the training loss a few epochs later than in the previous case.

4.4.3 Cropping

This test was performed adding two randomly cropped sections of each of the original images.

$$Precision = 0.286 \quad Recall = 0.260 \quad mAP@0.5 = 0.249$$

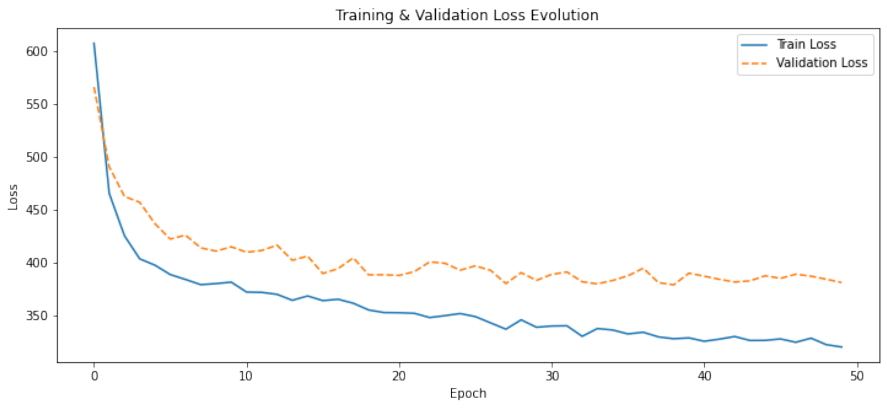


Figure 4.4: Loss evolution when trained just with cropping.

Recall was the metric that improved the most when including cropping. This could be due to the model recognizing potholes in places where it did not before. However, it is worth mentioning that these values are relatively low for the ones obtained by standard object detection models, so any conclusions based on them could be not representative of the reasons behind them. More importantly, the validation loss shown in Figure 4.4 keeps a negative slope further down the training process, which would be a sign of overfitting reduction.

4.4.4 Translation

This test was performed adding two random translations for each of the original images.

$$Precision = 0.394 \quad Recall = 0.298 \quad mAP@0.5 = 0.298$$

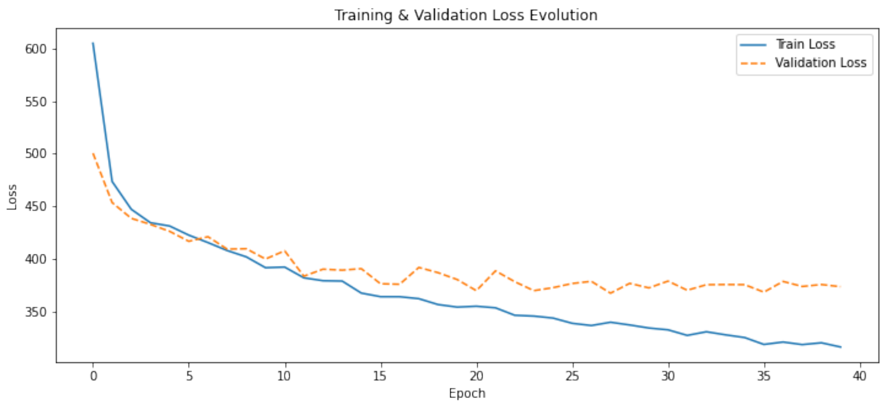


Figure 4.5: Loss evolution when trained just with translation.

All the metrics significantly improved, as well as the loss evolution obtained, which again diverges later in the training process as shown in Figure 4.5.

4.4.5 Mosaic

Mosaic augmentation introduces high variation in the images, creating new patterns with sizes and positions very different from the original dataset. Although the goal of this test is to isolate the effect of the technique, the flipped images were kept as part of the dataset. That way, the amount of images that remain with the same size of the original is larger, so more mosaic images can be introduced without the model overfitting to that pattern.

For every image in the original dataset, four new ones were created with the 2×2 grid as explained in Section 3.4.2.4. The performance obtained is the following:

$$Precision = 0.404 \quad Recall = 0.301 \quad mAP@0.5 = 0.302$$

Similar to the YOLOv8 training pipeline, the mosaic images were removed from the dataset for the last 10 epochs. The goal of this technique is also to prevent the final model from overfitting to the mosaic pattern, which will not appear in real images.

The performance improved significantly, both from the test with just the original dataset and from the one with flipping included.

4.4.6 All augmentations

Once each augmentation technique has proven to achieve better performance by their own, they were all included in the pipeline for the last test. The ratio in which each was used is the following:

- **Horizontal flip:** Each original image was added flipped
- **Cropping:** Same as in the test, every image was randomly cropped twice.
- **Translation:** Also as in the test, every image was randomly translated twice.
- **Mosaic:** To keep a relevant ratio, six mosaic images were created for every original one in the dataset.

$$Precision = 0.422 \quad Recall = 0.299 \quad mAP@0.5 = 0.300$$

These are the metrics obtained. Although significantly higher than the ones obtained with no augmentation, they are not much better than the performance obtained with just one of the two best performing techniques: translation or mosaic. This could be related to the general low performance, which is not comparable to the baseline models, indicating that there are other more relevant aspects of the model that need to be improved to achieve comparable performance.

But more importantly, as shown in Figure 4.6, the loss evolution keeps decreasing for much longer, and shows much less signs of overfitting. Note that the test was kept for 100 epochs, as the curve takes longer to flatten out.

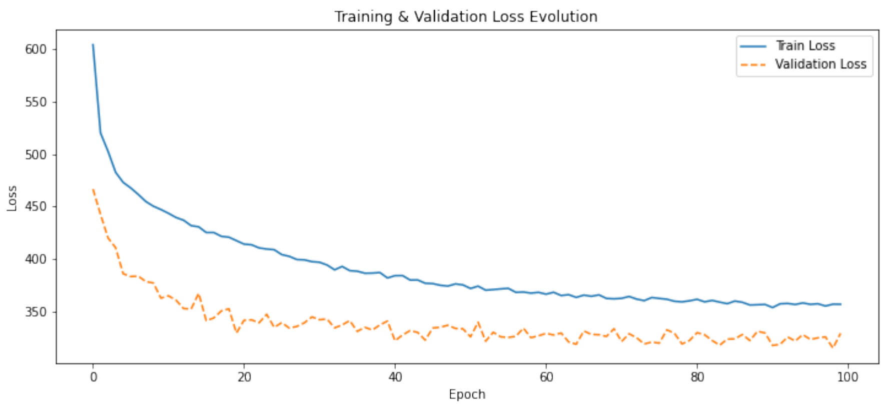


Figure 4.6: Loss evolution using all augmentation techniques.

Finally, Figure 4.7 shows a comparison of the performance metrics obtained in these data augmentation tests.

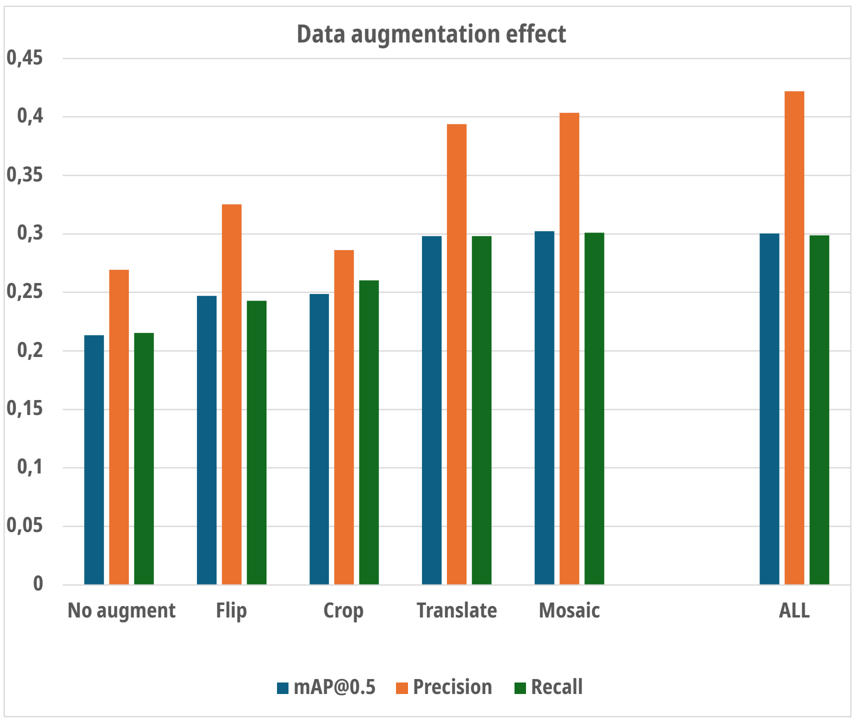


Figure 4.7: Metrics obtained by each augmentation technique.

4.5 Final results with proprietary model

Once a dataset pipeline was established, the different architectures considered could be tested.

4.5.1 MobileNetv2 as backbone

The first model tested was the same described in the previous section, taking MobileNetv2 as a backbone and the split head architecture described in Section 3.1.4.1. It has a total of 9.05 million parameters, with 8.44 million belonging to the custom neck and head and the rest 616 thousand coming from the MobileNetV2 architecture.

A similar model was later tested, changing the architecture of the split head. Instead of the first bottleneck blocks present in the architecture defined in Section 3.1.4.1 to reduce the depth of the filters, those are replaced by three convolutional layers of decreasing number of filters (576, 432 and 288). This change increases the number of parameters up to a total of 11.34 millions.

The metrics obtained when trained with the whole data augmentation scheme are the following:

$$Precision = 0.554 \quad Recall = 0.413 \quad mAP@0.5 = 0.415$$

There was an improvement in performance by using this head structure, probably aided by the increase in complexity. $mAP@0.5$ increased from 30% to 41.5%. Figure 4.8 shows the performance of both models side by side, also including the performance of this last model when trained with no data augmentation.

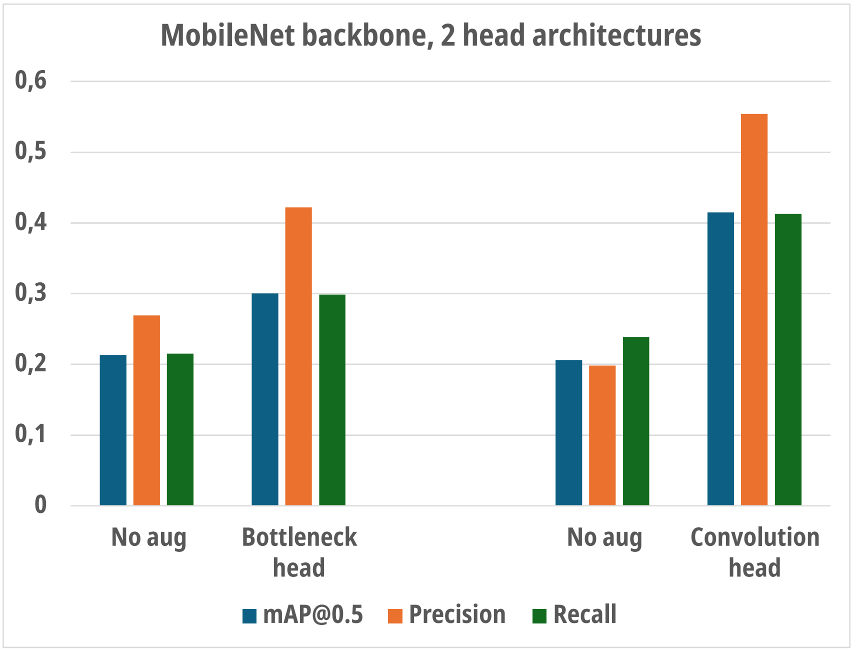


Figure 4.8: Performance from models with different heads. Split head with bottlenecks (left) and split head with convolutions (right).

4.5.2 DarkNet-53 as backbone

The architecture tested that uses DarkNet-53 as a backbone is composed by the following architecture:

- DarkNet-53 as a **backbone**, which outputs a feature map of size $14 \times 14 \times 1024$.
- **Neck** composed by a bottleneck block —explained in Section 3.1.3.3— that reduces the depth from 1024 to 512, followed by a convolutional layer with stride $s = 2$ that reduces the final grid size to 7×7.
- The split architecture explained in Section 3.1.4.1 is used as the **detection head**.

All these tests were performed using an input size of $448 \times 448 \times 3$ for the images.

DarkNet-53 is significantly larger than MobileNetV2, contributing with 40.62 million parameters from the total of 51.18 million that this model employs. This increase in complexity is supposed to be linked with better feature extraction capabilities.

The model was trained using the same augmentation described in the previous section, obtaining the following performance:

$$Precision = 0.642 \quad Recall = 0.447 \quad mAP@0.5 = 0.455$$

(Note: The threshold for the containment filter was lowered to a stricter value of 40% for these tests, as it shows better performance for this model. Therefore, NMS filter does not have any effect here, as any box removed by the NMS filter would also be removed based on containment.)

All the metrics improved when compared with the MobileNetV2 backbone model, being the precision the highest improvement. Another interesting result is the bigger difference between the model trained with and without augmentation, as seen in Figure 4.9.

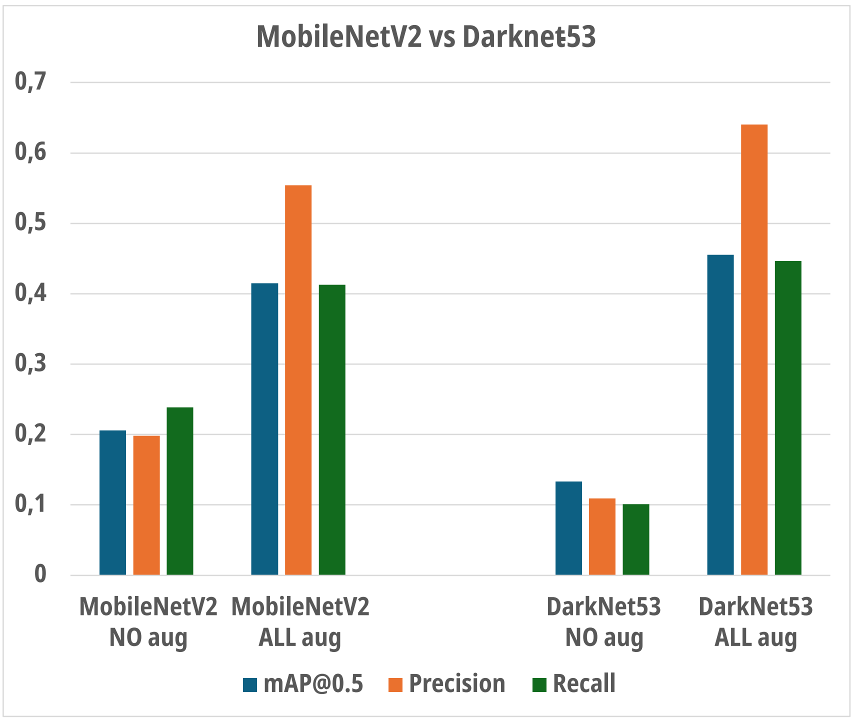


Figure 4.9: Performance from models with different backbones.

Figure 4.10 shows a comparison of the best performing model for each backbone, next to the baseline models' performance. This performance is still significantly lower than the baseline models, with a $mAP@0.5$ difference of around . Some of the potential issues that lead to this difference will be discussed in the following section.

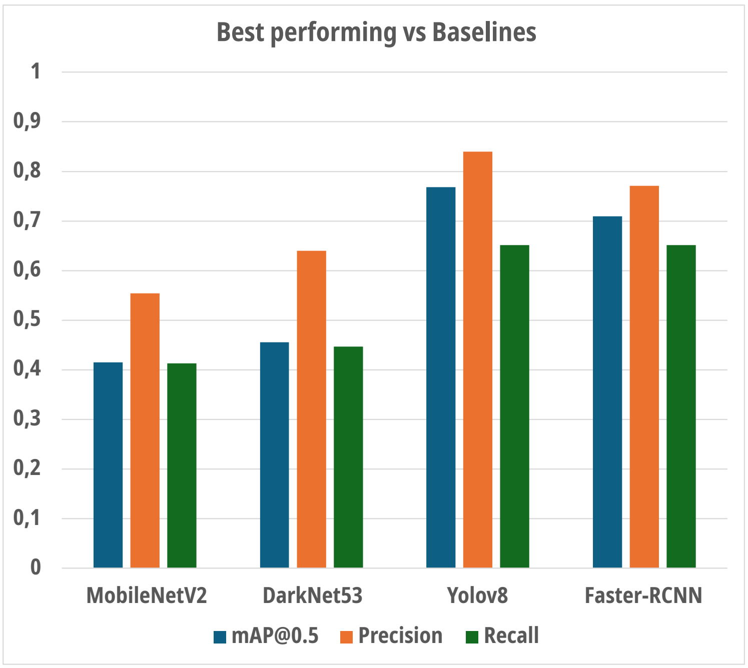


Figure 4.10: Comparison of best performing models against baseline.

4.6 Examples and potential flaws discussion

While quantitative metrics are essential to compare model performance, they may not fully capture all aspects of prediction quality. Factors such as localization precision, false positives, or missed detections can often be better appreciated through visual inspection. For this reason, we include a set of qualitative examples to highlight the typical behavior of the model. These images help illustrate recurring strengths and weaknesses, providing insights that complement the numerical evaluation.

4.6.1 Size and positioning

All the custom models tested seem to better respond to potholes when they are bigger in size and when they are centered in the image. This can be due to the bias already mentioned when explaining the dataset. Although the data augmentation techniques definitely helped reduce this unbalance, a bigger size dataset or a better training scheme could help further improving it.

Another factor to take into account is the number of objects present in the picture. The model seems to become less responsive when there are multiple potholes on sight, also obtaining worse coordinate predictions. In these cases, the model often predicts bounding boxes in the right area (the right YOLO grid cell becomes activated) but the coordinates do not match the object.

Figure 4.11 shows some examples where few potholes are present, leading to accurate predictions.

On the other hand, Figure 4.12 shows some faulty predictions when there are multiple potholes present.

4.6.2 Annotation criteria

Given the irregular nature of the potholes, as well as the lack of a precise way to define that kind of defect, there is a high subjective factor when labeling the data. This leads to some cases where a similar defect can be labeled in some images from the dataset but ignored on others. Additionally, some defects do not have a clear outline, making it difficult to precisely define the coordinates of the corresponding bounding box. For instance, potholes have a much less defined shape in gravel roads.

In some cases, the model tends to group the potholes, predicting a single bounding box where multiple ones are labeled. It can also make a prediction

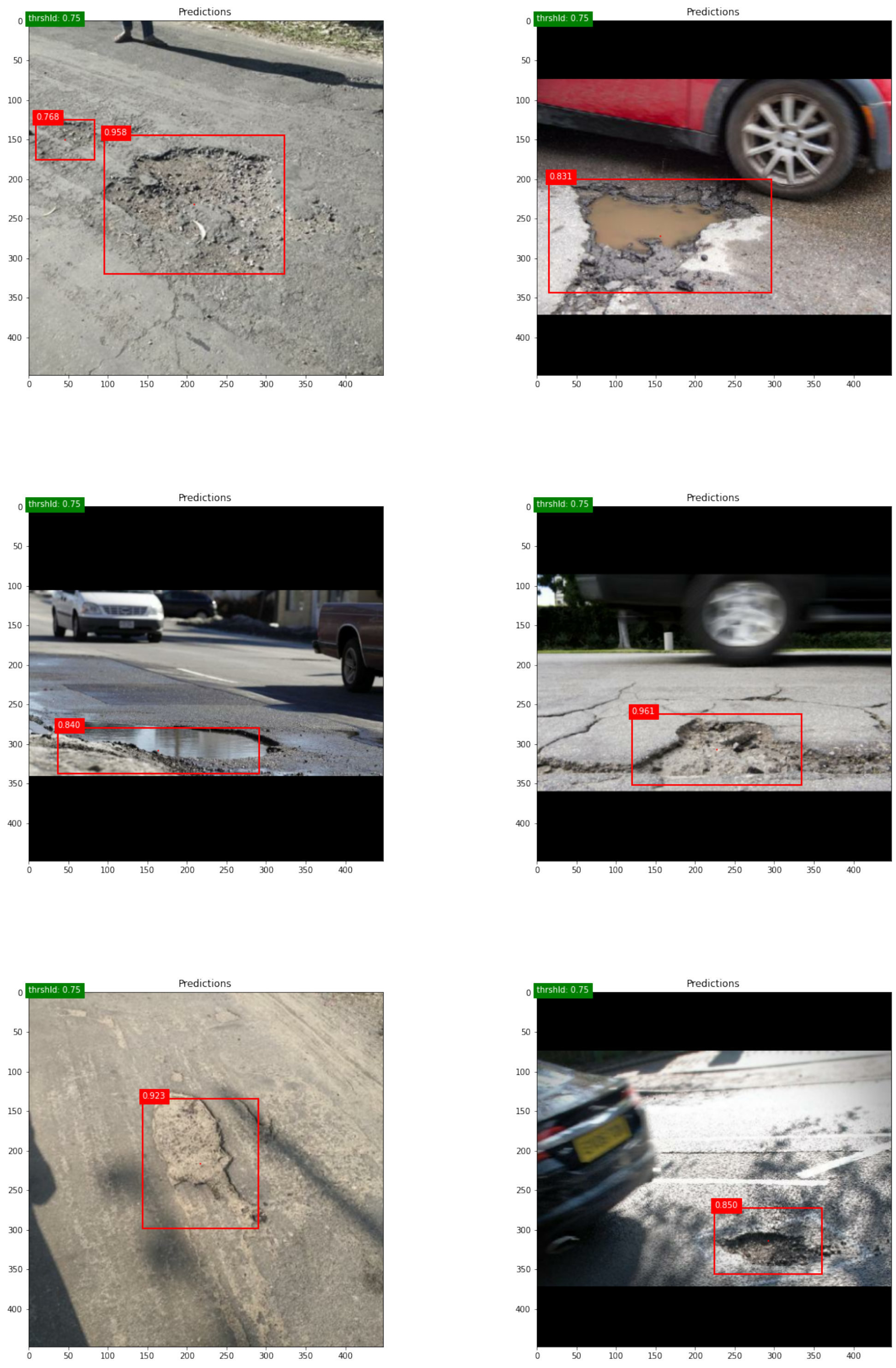


Figure 4.11: Examples of "good" predictions with few potholes.

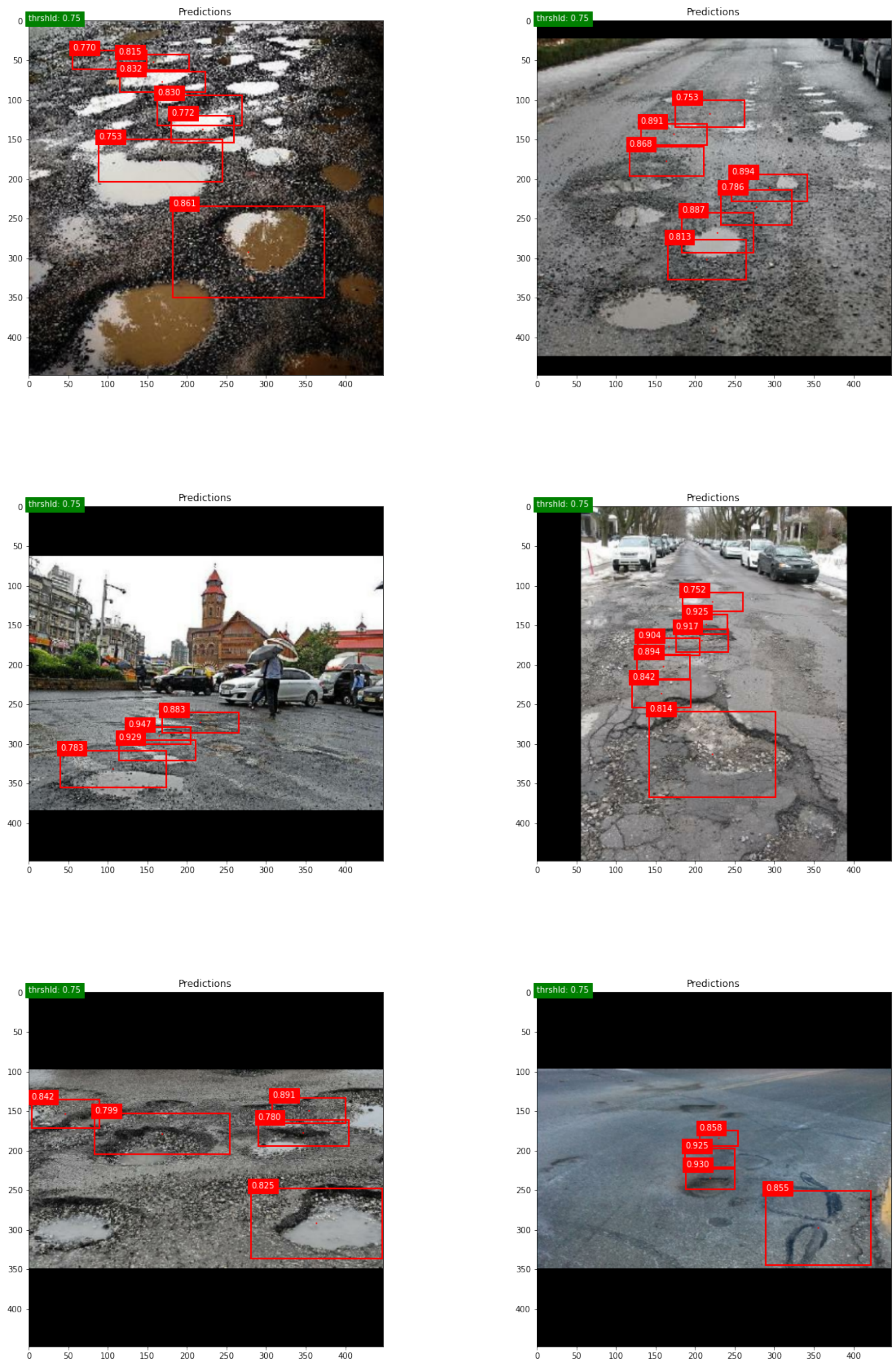


Figure 4.12: Examples of "bad" predictions with multiple potholes.

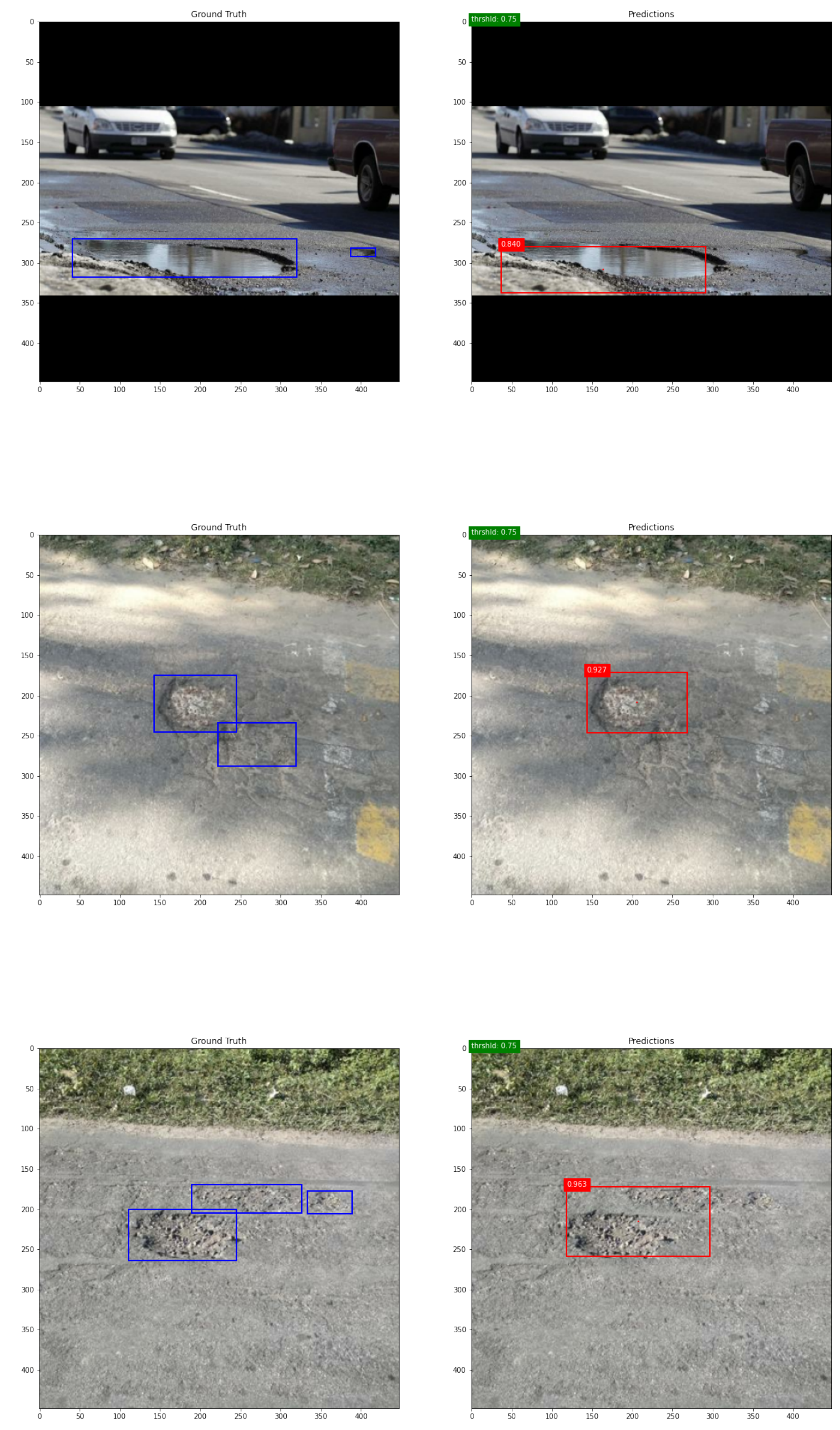


Figure 4.13: Effect of annotation criteria (1). Ground truth on the left, predictions on the right.

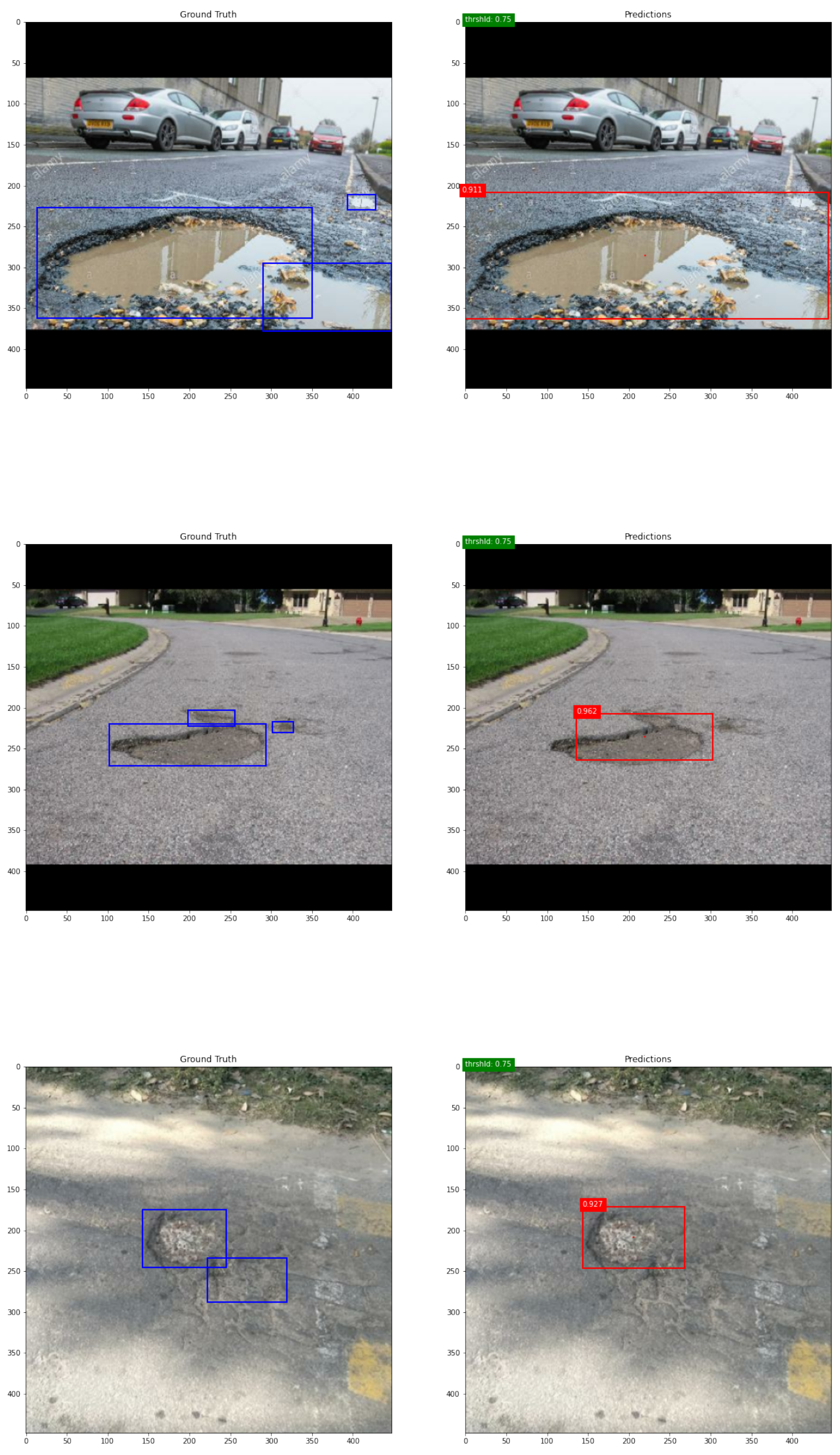


Figure 4.14: Effect of annotation criteria (2). Ground truth on the left, predictions on the right.

in a spot that could be considered a defect but is not labeled as one. In other cases, it fails to predict less obvious potholes, being even more prominent for small objects.

Figure 4.13 and Figure 4.14 show some of those cases, where the annotation criteria would highly impact whether or not a prediction is considered suitable for the given task.

4.6.3 Loss of attention

Related to the size discussion, it often happens that the model loses some sensitivity when there is a clear object in the image. Figure 4.15 illustrates some of those cases, where the presence of a clear big pothole near the middle of the image seems to make it less responsive to the smaller potholes present in its surroundings.

Note: All the predictions for these examples were taken with the DarkNet-53 backbone model explained in Section 4.5.2.

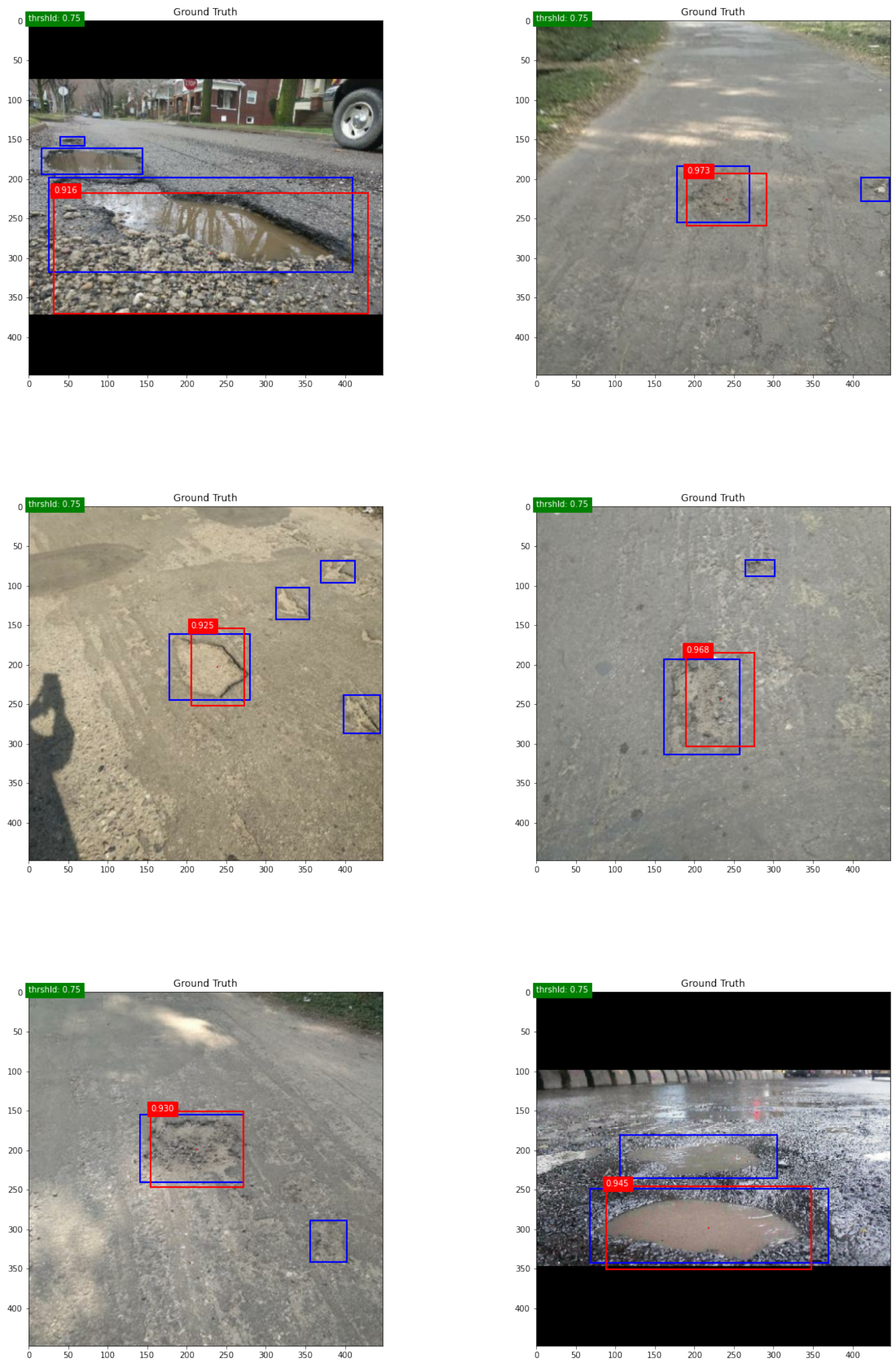


Figure 4.15: Examples of lost attention with a clear pothole present. Ground truth (blue) and predictions (red).

4.7 Clean test dataset

Some of the issues discussed in the previous section could be mitigated by a better selected test dataset, as they are more closely related to the quality of the test samples rather than to the actual performance of the model. For instance, annotation consistency should be maintained more rigorously.

To achieve a better assessment of the model's performance, a new test dataset was created by removing some of these outlier samples that introduced noise. Upon closer inspection of the test dataset, a few images were found as problematic: some contained watermarks and text that the model to make incorrect predictions, while others featured ambiguous annotations or doubtful pothole examples. These images were excluded from the test set. A total of 10 images were removed, keeping approximately 85% of the original test dataset.

With this new dataset, the metrics for both custom models —with MobileNetV2 and DarkNet-53 backbones— were computed again. The results are as follows:

For the MobileNetV2 backbone:

$$Precision = 0.654 \quad Recall = 0.469 \quad mAP@0.5 = 0.477$$

For the Darknet-53 backbone:

$$Precision = 0.749 \quad Recall = 0.518 \quad mAP@0.5 = 0.533$$

The metrics show a clear improvement, with the mAP@0.5 increasing by a 6.1% in case of MobileNetV2 backbone and by 7.9% for DarkNet-53. The most notable change is in precision, which increased by 10.9 percentage points in the latter case, reaching 74.9%. This precision performance is already comparable to that of the Faster-RCNN baseline (77.1%). The updated new results are shown in Figure 4.16, including the baselines for comparison.

Recall values —although also improved with the cleaner dataset— are still low when compared to the values usually obtained by state-of-the-art object detection models. That ultimately leads to a low general metric of mAP@0.5.

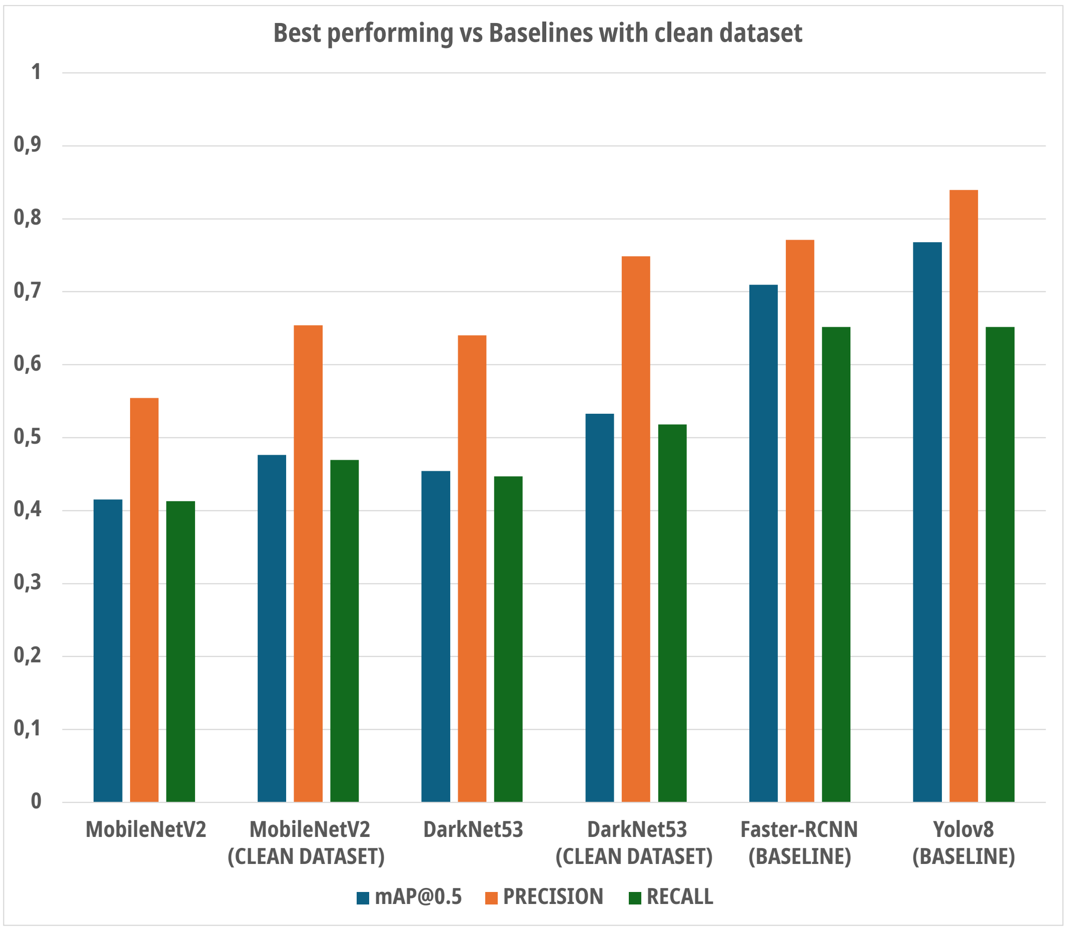


Figure 4.16: Comparison of best performing custom models with the updated cleaner test dataset against baseline performances.

These last results not only show a better assessment of the capabilities of the models tested, but also serve as a confirmation of the biases and flaws mentioned in the previous section.

Chapter 5

Conclusions and future work

The main goal of this thesis was to explore options for a proprietary object detection model specific for potholes, which has been achieved. Experiments were carried out to test different architectures and to get a thorough comparison with some of the state-of-the-art models.

Although the trained models were able to accurately predict potholes under specific conditions, there is a lack in robustness and performance that should be addressed if any of the models are finally used for their intended task. Several potential directions derive from this work to address the observed performance gap, including:

- Training the same models with a different dataset. Size has proven to highly affect the outcome, so a bigger dataset could be helpful. Also including more realistic pictures in the dataset, all taken in the same perspective, could help with generalization for this given task.

- Addressing the poor performance on small objects, for which there are several options. A multi-scale detecting head can be implemented, just like the ones in late YOLO versions. A simpler approach could take feature maps at different stages of the backbone and fusing them in the neck before feeding them to the detection head.

- Including anchor boxes on the model. It was not considered for this work, given the single-class focus and the small size dataset, but it is a feature common to multiple state-of-the-art models.

- Test other architectures for the head, either following the YOLO structure or not.

Although it was not the main focus of the project, some interesting results have been obtained about the effect of data augmentation techniques, especially for small sized datasets. Further elaboration on this topic could be of interest.

One of the advantages for a proprietary model that was mentioned in the work is the potential increase in efficiency and adaptability when inferring on edge devices. Therefore, their performance could be tested on local hardware to compare it with the standard models, also obtaining their Real-Time capabilities.

Other path worth exploring is to optimize the multiclass version of the model, so that it can detect multiple types of road defects (using RDD2022 could be a choice for training). Although the focus for the models proposed in this work was not to be a general-purpose detector, another option to explore would be training the model on a benchmark multiclass dataset (such as COCO or others), to later train it in the specific dataset and test if generalization improves.

Bibliography

[1] Roboflow. *YOLOv8 Model License Guide*. https://roboflow.com/model-licenses/yolov8. Accessed: 2025-02-13. 2023 (cit. on p. 2).

[2] Free Software Foundation. *GNU Affero General Public License v3.0*. https://www.gnu.org/licenses/agpl-3.0.html. Accessed: 2025-05-15. 2007 (cit. on p. 2).

[3] FOSSA. *Open Source Licenses 101: AGPL License*. https://fossa.com/blog/open-source-software-licenses-101-agpl-license/. Accessed: 2025-05-15. 2023 (cit. on p. 2).

[4] BBC News. *Potholes: What are they and why are they dangerous?* https://www.bbc.com/news/uk-england-67958426. BBC News, Accessed: 2025-04-23. 2023 (cit. on p. 6).

[5] Vialytics. *The Dangers of Potholes: A Growing Threat to Public Safety*. https://www.vialytics.com/blog/dangersofpotholes. Accessed: 2025-05-13. 2024 (cit. on p. 6).

[6] Ary Setyawan, Irvan Kusdiantoro, and Syafi'I Syafi'i. "The Effect of Pavement Condition on Vehicle Speeds and Motor Vehicles Emissions". In: *Procedia Engineering* 125 (Dec. 2015), pp. 424–430. DOI: 10.1016/j.proeng.2015.11.111 (cit. on p. 6).

[7] Omar Chaaban, Leen Daher, Yara Ghaddar, Nader Zantout, Daniel Asmar, and Naseem Daher. "An End-to-End Pothole Detection and Avoidance System for Autonomous Ground Vehicles". In: *2025 International Conference on Control, Automation, and Instrumentation (IC2AI)*. 2025, pp. 1–6. DOI: 10.1109/IC2AI62984.2025.10932181 (cit. on p. 7).

[8] Hamzeh Zakeri, Fereidoon Moghadas Nejad, and Ahmad Fahimifar. "Image based techniques for crack detection, classification and quantification in asphalt pavement: a review". In: *Archives of Computational Methods in Engineering* 24 (2017), pp. 935–977 (cit. on pp. 7, 8).

[9] Sylvie Chambon and Jean-Marc Moliard. "Automatic road pavement assessment with image processing: Review and comparison". In: *International Journal of Geophysics* 2011.1 (2011), p. 989354 (cit. on p. 7).

[10] Chen Yu, Chen Dian-ren, Li Yang, and Chen Lei. “Otsu’s thresholding method based on gray level-gradient two-dimensional histogram”. In: *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010)*. Vol. 3. 2010, pp. 282–285. DOI: 10.1109/CAR.2010.5456687 (cit. on p. 8).

[11] Christian Koch and Ioannis Brilakis. “Pothole detection in asphalt pavement images”. In: *Advanced Engineering Informatics* 25.3 (2011). Special Section: Engineering informatics in port operations and logistics, pp. 507–515. ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2011.01.002. URL: https://www.sciencedirect.com/science/article/pii/S1474034611000036 (cit. on p. 8).

[12] Ionut Schiopu, Jukka P. Saarinen, Lauri Kettunen, and Ioan Tabus. “Pothole detection and tracking in car video sequence”. In: *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*. 2016, pp. 701–706. DOI: 10.1109/TSP.2016.7760975 (cit. on p. 8).

[13] Alex Lenail. *NN-SVG: Publication-Ready Neural Network Architecture Schematics*. Accessed: 2025-03-20. 2019. URL: https://alexlenail.me/NN-SVG/ (cit. on p. 9).

[14] Yashar Safyari, Masoud Mahdianpari, and Hodjat Shiri. “A Review of Vision-Based Pothole Detection Methods Using Computer Vision and Machine Learning”. In: *Sensors* 24 (Aug. 2024), p. 5652. DOI: 10.3390/s24175652 (cit. on p. 9).

[15] J. Javier Yebes, David Montero, and Ignacio Arriola. “Learning to Automatically Catch Potholes in Worldwide Road Scene Images”. In: *IEEE Intelligent Transportation Systems Magazine* 13.3 (2021), pp. 192–205. DOI: 10.1109/MITS.2019.2926370 (cit. on p. 9).

[16] Aditi Tithi, Firoj Ali, and Sadman Azrof. “Speed Bump and Pothole Detection with Single Shot MultiBox Detector Algorithm and Speed Control for Autonomous Vehicle”. In: *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)*. 2021, pp. 1–5. DOI: 10.1109/ACMI53878.2021.9528185 (cit. on p. 9).

[17] Anas Al-Shaghouri, Rami Alkhatib, and Samir Berjaoui. *Real-Time Pothole Detection Using Deep Learning*. July 2021. DOI: 10.48550/arXiv.2107.06356 (cit. on p. 9).

[18] Montaser Nalawi, Mohammad Baghdadi, Basel Alyateem, Zaer S Abo-Hammour, and Adham Alsharkawi. “Design of a Real-time Detection System for Potholes and Bumps Using Deep Learning”. In: *2024 22nd International Conference on Research and Education in Mechatronics*

(REM). 2024, pp. 160–165. DOI: 10.1109/REM63063.2024.10735479 (cit. on p. 10).

[19] T C Mahalingesh, Anubhav, Harshit Mishra, R V Arun, and Anshuman Anand. “Pothole Detection and Filling System using Image processing and Machine Learning”. In: *2024 International Conference on Smart Systems for applications in Electrical Sciences (ICSSES)*. 2024, pp. 1–5. DOI: 10.1109/ICSSES62373.2024.10561417 (cit. on p. 10).

[20] Omar Chaaban, Leen Daher, Yara Ghaddar, Nader Zantout, Daniel Asmar, and Naseem Daher. “An End-to-End Pothole Detection and Avoidance System for Autonomous Ground Vehicles”. In: *2025 International Conference on Control, Automation, and Instrumentation (IC2AI)*. 2025, pp. 1–6. DOI: 10.1109/IC2AI62984.2025.10932181 (cit. on p. 10).

[21] Hexiang Bai and Ling Qin. “Research on Road Defect Detection Based on Improved YOLOv5s”. In: *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*. 2024, pp. 2344–2348. DOI: 10.1109/AINIT61980.2024.10581495 (cit. on p. 10).

[22] Pengchun Zhang, Haoran Chen, Jiahui Gao, Liqiang Ma, and Rong He. “Improved YOLOv10 for High-Precision Road Defect Detection”. In: Sept. 2024, pp. 79–83. DOI: 10.1109/CCSB63463.2024.10735633 (cit. on p. 10).

[23] Wenzhe Wang, Bin Wu, Sixiong Yang, and Zhixiang Wang. “Road Damage Detection and Classification with Faster R-CNN”. In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 5220–5223. DOI: 10.1109/BigData.2018.8622354 (cit. on p. 10).

[24] Chuan-Bi Lin and Yikai Liu. “Research on Potholes Detection Based on Improved Mask RCNN Algorithms”. In: *2023 12th International Conference on Awareness Science and Technology (iCAST)*. 2023, pp. 50–54. DOI: 10.1109/iCAST57874.2023.10359275 (cit. on p. 10).

[25] Surekha Arjapure and D.R. Kalbande. “Deep Learning Model for Pothole Detection and Area Computation”. In: *2021 International Conference on Communication information and Computing Technology (ICCICT)*. 2021, pp. 1–6. DOI: 10.1109/ICCICT50803.2021.9510073 (cit. on p. 10).

[26] Zhen Zhang, Xiao Ai, C. K. Chan, and Naim Dahnoun. “An efficient algorithm for pothole detection using stereo vision”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 564–568. DOI: 10.1109/ICASSP.2014.6853659 (cit. on p. 10).

[27] Yaqi Li, Christos Papachristou, and Daniel Weyer. “Road Pothole Detection System Based on Stereo Vision”. In: *NAECON 2018 - IEEE*

National Aerospace and Electronics Conference. 2018, pp. 292–297. DOI: 10.1109/NAECON.2018.8556809 (cit. on p. 10).

[28] Jongmin Yu, Jiaqi Jiang, Sebastiano Fichera, Paolo Paoletti, Lisa Layzell, Devansh Mehta, and Shan Luo. "Road Surface Defect Detection—From Image-Based to Non-Image-Based: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* 25.9 (2024), pp. 10581–10603. DOI: 10.1109/TITS.2024.3382837 (cit. on p. 10).

[29] Xiangyang Xu and Hao Yang. "Intelligent crack extraction and analysis for tunnel structures with terrestrial laser scanning measurement". In: *Advances in Mechanical Engineering* 11 (Sept. 2019), p. 168781401987265. DOI: 10.1177/1687814019872650 (cit. on p. 10).

[30] A. A. Tsesar, S. V. Varshavskiy, Yu. E. Vasiliev, and M. A. Fineeva. "Technology for Collecting and Data Processing of Street-Road Network Monitoring Objects". In: *2023 Systems of Signals Generating and Processing in the Field of on Board Communications*. 2023, pp. 1–6. DOI: 10.1109/IEEECONF56737.2023.10092184 (cit. on p. 10).

[31] Anoop Thomas, Jobin K. Antony, and Sarath Chandran S. "Detection of Road Attributes Using Solid-State LiDAR". In: *2024 11th International Conference on Advances in Computing and Communications (ICACC)*. 2024, pp. 1–5. DOI: 10.1109/ICACC63692.2024.10845297 (cit. on p. 10).

[32] Hassan Muhammad Saddique, Ahsan Raza, Zain Ul Abideen, and Shah Nawaz Khan. "Exploring Deep Learning based Object Detection Architectures: A Review". In: *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. 2020, pp. 255–259. DOI: 10.1109/IBCAST47879.2020.9044558 (cit. on p. 10).

[33] Aydin Ayanzadeh. *A Study Review: Semantic Segmentation with Deep Neural Networks*. Mar. 2018. DOI: 10.13140/RG.2.2.12534.04160/2 (cit. on p. 12).

[34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322 (cit. on p. 12).

[35] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640 (cit. on pp. 13, 15).

[36] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. ISBN: 9783319464480. DOI: 10.1007/978-

3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2 (cit. on pp. 13, 14).

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594 (cit. on p. 16).

[38] Muhammad Hussain. "YOLOv1 to v8: Unveiling Each Variant–A Comprehensive Review of YOLO". In: *IEEE Access* 12 (2024), pp. 42816–42833. DOI: 10.1109/ACCESS.2024.3378568 (cit. on p. 16).

[39] Ghazlane Yasmine, Gmira Maha, and Medromi Hicham. "Overview of single-stage object detection models: from Yolov1 to Yolov7". In: *2023 International Wireless Communications and Mobile Computing (IWCMC).* 2023, pp. 1579–1584. DOI: 10.1109/IWCMC58020.2023.10182423 (cit. on p. 16).

[40] S. Jain, D. Ramesh, E. Damodar Reddy, et al. "A fast high throughput plant phenotyping system using YOLO and Chan-Vese segmentation". In: *Soft Computing* 28 (2024). Published: August 5, 2024; Issue Date: October 2024, pp. 12323–12336. DOI: 10.1007/s00500-024-09946-y (cit. on p. 16).

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474 (cit. on pp. 19–21).

[42] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement.* 2018. arXiv: 1804.02767 [cs.CV]. URL: https://arxiv.org/abs/1804.02767 (cit. on pp. 21, 22).

[43] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger.* 2016. arXiv: 1612.08242 [cs.CV]. URL: https://arxiv.org/abs/1612.08242 (cit. on p. 21).

[44] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLOv8.* Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics (cit. on p. 37).

[45] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.* 2022. arXiv: 2207.02696 [cs.CV]. URL: https://arxiv.org/abs/2207.02696 (cit. on p. 37).

[46] Hao Dong, Mu Yuan, Shu Wang, Long Zhang, Wenxia Bao, Yong Liu, and Qingyuan Hu. "PHAM-YOLO: A Parallel Hybrid Attention Mechanism

Network for Defect Detection of Meter in Substation". In: *Sensors* 23.13 (2023). ISSN: 1424-8220. URL: https://www.mdpi.com/1424-8220/23/13/6052 (cit. on p. 38).

[47] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. *Path Aggregation Network for Instance Segmentation.* 2018. arXiv: 1803.01534 [cs.CV]. URL: https://arxiv.org/abs/1803.01534 (cit. on p. 38).

[48] Saluky, Yoni Marine, Ahmad Zaeni, Ari Yuliati, Onwardono Rit Riyanto, and Nurul Bahiyah. "Pothole Detection on Urban Roads Using YOLOv8". In: *2023 10th International Conference on ICT for Smart Society (ICISS).* 2023, pp. 1–6. DOI: 10.1109/ICISS59129.2023.10291192 (cit. on p. 38).

[49] Riddhi Mirajkar, Anuradha Yenkikar, Shreyash Nawalkar, Rishabh Kaul, Aditya Rokade, and Kedarnath Rothe. "Enhanced Pothole Detection in Road Condition Assessment Using YOLOv8". In: *2024 IEEE International Conference for Women in Innovation, Technology & Entrepreneurship (ICWITE).* 2024, pp. 429–433. DOI: 10.1109/ICWITE59797.2024.10502437 (cit. on p. 38).

[50] Laura Tsanaullailla, Inggit Yeira Budi Agranata, Steven Harun Samba, and Faqih Hamami. "Road Pothole Damage Detector at Night Using YOLOv8". In: *2024 International Conference on Advanced Information Scientific Development (ICAISD).* 2024, pp. 138–143. DOI: 10.1109/ICAISD63055.2024.10894890 (cit. on p. 38).

[51] Meta AI Research. *Detectron2.* Accessed: 2025-05-25. 2019. URL: https://ai.meta.com/tools/detectron2/ (cit. on p. 39).

[52] Ross Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV).* 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169 (cit. on p. 39).

[53] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497 (cit. on pp. 39, 40).

[54] Deeksha Arya, Hiroya Maeda, Sanjay Kumar Ghosh, Durga Toshniwal, and Yoshihide Sekimoto. "RDD2022: A multi-national image dataset for automatic Road Damage Detection". In: *arXiv preprint arXiv:2209.08538* (2022) (cit. on p. 40).

[55] Roboflow. *Pothole Detection Dataset.* Accessed: 2025-06-01. URL: https://public.roboflow.com/object-detection/pothole (cit. on p. 41).

[56] Khaled R. Ahmed. "Smart Pothole Detection Using Deep Learning Based on Dilated Convolution". In: *Sensors* 21.24 (2021). ISSN: 1424-8220. DOI: 10.3390/s21248406. URL: https://www.mdpi.com/1424-8220/21/24/8406 (cit. on p. 41).

[57] Stefan Achirei, Ioana-Ariana Opariuc, Otilia Zvorișteanu, Simona Caraiman, and Vasile Manta. “Pothole Detection for Visually Impaired Assistance”. In: Oct. 2021, pp. 409–415. DOI: 10.1109/ICCP53602.2021.9733610 (cit. on p. 41).

Dedications

This work puts an end to a long journey, one that wouldn't have even started without the encouragement I have always received from my family. It also wouldn't have ended in such a great place if it wasn't for your constant support and your tolerance for my complaints. Gracias a mi madre, mi padre y mi hermano Jorge.

These words aren't just for those who have helped during the thesis, but for everybody that accompanied throughout these years. I feel grateful to have so many to thank that it will be hard to include them all here. First to my friends in Segovia and from my university back at home. Although we are distant right now, it feels great to know that we care for each other. Thanks to those who live in Chicago, who I rarely hear from but are and always will be a big part of me anyways. Same goes for that friend in Germany, the one in Málaga and that other one who I am lucky to hear more often, allowing him to withstand more of my complaining throughout this whole time.

These two last years have been such an experience. Everybody feels lucky for the people they meet when they are abroad, it's probably just a matter of attitude and a change of environment. But I'm truly grateful to have had the chance to surround myself with people I now consider great friends. Thanks to the guy I did my first multipitch with, to that other guy that took me to my first four thousand, to those who have shared so much time with me in the library, and to that person who saw me fall in the best ways possible. Thanks to all those who already left Torino, and to those who will unfortunately leave soon. And thanks to anybody that passed by Nizza, that might be a good way to sum it up.