**POLITECNICO DI TORINO**

**Master's Degree in Data Science and Engineering**

**Master's Degree Thesis**

# Real-Time Human Detection in World Rally Championship for danger recognition system

**Supervisors**

**Prof. Flavio GIOBERGIA**

**Dr. Andrea AVIGNONE**

**Dr. Dennis D'AMORE**

**Dr. Alberto MINNELLA**

**Candidate**

**Francesco MARIGIOLI**

**July 2024**

# Summary

This thesis was conducted in collaboration with Marelli S.p.A. Our objective was to develop a network capable of recognizing humans in real-time, intended for use in an alert system in the World Rally Championship (WRC). The WRC is the premier rally competition globally. Despite the high standards of this sport, the safety marshals responsible for overseeing the season's events face challenges in efficiently monitoring all sections of the circuit to ensure spectator safety. In response, Marelli S.p.A. aims to develop an alert system for these marshals, which will help identify individuals in hazardous positions, allowing for timely intervention and enhanced safety across the entire track. A critical preliminary step in developing this system is real-time human detection. Although object detection has been extensively studied, the unique challenges presented by rally racing necessitate additional research specific to this domain. The main challenge was to work with a single camera installed inside the vehicle, which travels at high speeds and needs to recognize people at considerable distances from the vehicle's position. To address this task, we utilized the YOLO (You Only Look Once) network and experimented with various training combinations to optimize its performance for our specific use case and certain challenging situations. We also examined the impact of image quality on the network's performance. After initial evaluation of existing datasets, we decided to create a custom dataset to enhance the network's performance in our domain and ensure robustness against potential scenario changes, despite the limited number of video samples available. The resulting network meets Marelli's standards for accuracy, speed, and performance, having been thoroughly tested on a hardware configuration very similar to the one that will be installed in the car.

# Acknowledgements

I would like to express my gratitude to Professor Flavio Giobergia and Dr. Andrea Avignone for their support and professionalism shown to me throughout the development of my thesis. Additionally, I would like to thank the entire Marelli Motorsport Team, particularly Dennis D'Amore and Alberto Minnella, for the experience and the opportunity to enter an excellent work environment, which will serve as a cultural asset for my future endeavors.

I would also like to thank my parents for being my first supporters and advisors, allowing me to manage my academic journey independently, letting me make mistakes, learn, and grow. Thanks to my entire family for providing me with an environment rich in love and esteem.

Special thanks to my girlfriend and friends, because you represent who I am in this world: my roots, my passions, and my relationships. And thank you to everyone who is reading or listening to these words, because we all change each other's lives, and our achievements are always a result of what we experience every day.

Finally, I want to thank myself for always viewing life as a challenge and a gift, valuing not a school grade but the number of people to whom I can dedicate these words.

*Francesco Marigioli*

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**AI**

Artificial Intelligence

**AP**

Average Precision

**BN**

Batch Normalization

**Bof**

Bag-of-freebies

**Bos**

Bag-of-specials

**CIoU**

Complete Intersection over Union

**CLI**

Command Line Interface

**CmBN**

Cross mini-Batch Normalization

**CNN**

Convolutional Neural Networks

**COCO**

Common Objects in Context

|  |  |
|---|---|
| **CSP** | Cross-Stage Partial |
| **DFL** | Distribution Focal Loss |
| **ECU** | Engine Control Unit |
| **ELAN** | Efficient Layer Aggregation Network |
| **EMA** | Exponential Moving Average |
| **E-ELAN** | Extended Efficient Layer Aggregation Network |
| **FC** | Fully Connected |
| **GE** | Genetic Evolution |
| **HOG** | Histogram of Oriented Gradients |
| **HSV** | Hue Saturation Value |
| **ILSVRC** | ImageNet Large Scale Visual Recognition Challenge |
| **IoU** | Intersection over Union |
| **KITTI** | Karlsruhe Institute of Technology and Toyota Technological Institute |

| | |
|---|---|
| **mAP** | mean Average Precision |
| **MRI** | Magnetic Resonance Imaging |
| **PAN** | Path Aggregation Network |
| **PIP** | PIP Installs Packages (recursive) |
| **ReLU** | Rectified Linear Unit |
| **RepConv** | Re-Parametrized Convolutions |
| **RoI** | Region of Interest |
| **RPN** | Region Proposal Networks |
| **R-CNN** | Regions-Convolutional Neural Networks |
| **SAM** | Spatial Attention Module |
| **SAT** | Self Adversarial Training |
| **SiLU** | Sigmoid Linear Unit |
| **SOTA** | State Of The Art |
| **SPP** | Spatial Pyramid Pooling |

| | |
|---|---|
| **SPPF** | Spatial Pyramid Pooling Fast |
| **SSD** | Single Shot multibox Detector |
| **SVM** | Support Vector Machine |
| **TOOD** | Task-aligned One-stage Object Detection |
| **VGG** | Visual Geometry Group |
| **VOC** | Visual Object Classes |
| **WRC** | World Rally Championship |
| **YOLO** | You Only Look Once |

# Chapter 1

# Introduction

## 1.1 The impact of Object Detection

Object detection is a significant field within computer vision, a branch of artificial intelligence (AI) that deals with enabling computers to interpret and understand the visual world. Object detection involves identifying and locating objects within images or videos. Unlike image classification, which assigns a label to an entire image, object detection provides more detailed information by identifying the presence of objects, their categories, and their specific locations within the image. The journey of object detection began with basic feature-based methods, which relied on hand-crafted features for tasks such as face detection. However, these methods were limited in accuracy and adaptability. The advent of deep learning and convolutional neural networks (CNNs) marked a significant leap forward. Models like R-CNN and its successors introduced more efficient and accurate object detection and innovations such as YOLO (You Only Look Once) brought real-time object detection capabilities, making it feasible to detect objects in a single evaluation. The development and advancement of object detection technologies have had a profound impact on numerous industries and applications. In autonomous driving, object detection is critical for perceiving the environment, detecting pedestrians, vehicles, traffic signs, and obstacles. It also enhances security through automated surveillance, identifying suspicious activities and unauthorized access. Advanced systems can track multiple individuals, recognize abnormal behaviors, and detect specific objects like weapons, thus improving public safety. In healthcare, object detection is used in medical imaging to identify and localize abnormalities such as tumors in X-rays or MRI scans. Retail applications include inventory management, automated checkout systems, and customer behavior analysis. In agriculture, drones equipped with object detection systems monitor crop health, detect pests, and assess yield. The diverse range of fields where object detection is applied and its proven effectiveness make it one of the most intriguing and extensively studied tasks in deep learning.

## 1.2 Real-Time Human Detection

Real-time human detection is a complex and critical task within the field of object detection and computer vision. One of the primary challenges is ensuring accurate detection in varying and complex environments. Changing lighting conditions, occlusions, and dynamic backgrounds can negatively impact the system's performance. For example, a person may be partially hidden behind an object or be detected in scenes with low or variable lighting, complicating the model's task. Furthermore, detection models must generalize to new environments and scenarios that may not have been present in the training data. Real-time human detection requires models to be computationally efficient, particularly when deployed on devices with limited resources such as cameras, drones, or smartphones. This means the models need to be lightweight and capable of processing data quickly without compromising accuracy. In this Thesis, we investigated the state-of-the-art (SOTA) models for this task, specifically interfacing with the YOLO (You Only Look Once) model, which demonstrates strong performance despite all the aforementioned complications that are inherent of real-time human detection.

## 1.3 The choice of YOLO and the challenges

The advent of YOLO was groundbreaking in this particular task. It is designed to be extremely fast, processing the entire image in a single pass (forward pass). This makes it highly efficient for real-time applications. Despite its speed, YOLO maintains a high level of accuracy. Its ability to balance speed and precision makes it ideal for critical applications. The last main advantage is relatively lightweight and can run on hardware with limited resources, such as mobile devices or embedded cameras. This is crucial for human detection in scenarios where computational resources are constrained.

Despite its excellent performance, YOLO has some limitations that are important to highlight, especially concerning our work. YOLO's architecture is less effective at detecting small objects compared to some other object detection frameworks. This is because the spatial resolution of the feature maps used for detection may be too coarse to capture the fine details of small objects. In addition, it can have difficulty handling occlusions, where parts of the objects are obscured by other objects. In real-world scenarios, objects are often partially occluded, and the ability to accurately detect and classify these partially visible objects is essential. Nonetheless, in scenes with a high density of objects, YOLO may struggle to maintain high detection accuracy. Finally, like other deep learning models, YOLO's performance heavily depends on the quality and diversity of the training data. If the training data is biased or lacks variability, the model may fail to generalize well to new, unseen environments.

## 1.4 WRC further complications

An additional complication arises from our specific use case. The thesis aims to address the task of real-time human detection within the World Rally Championship (WRC) context. Marelli S.p.A. is a leader in producing ECUs and hardware and communication systems for diagnostics in major global motorsport championships, including the WRC. This introduces unique challenges to an already complex task. The practical application of this research would be to develop an alert system for safety marshals overseeing the various races in the championship. This system would help identify individuals in hazardous positions, enabling marshals to intervene and enhance safety along the entire track. Real-time human detection is a crucial preliminary step for developing this system, and this thesis aims to achieve high performance and reliability on this task, providing a foundation for developing a comprehensive system capable of identifying people in danger through domain-specific knowledge. The added difficulties are significant. Firstly, one complication is that we can only obtain data from a camera installed inside the vehicle. This not only limits the field of view but also involves dealing with extremely high speeds, resulting in more challenging video compression, significant frame-by-frame image changes, and rapid scenario shifts. Additionally, this leads to the challenge of recognizing individuals even when they are several tens of meters away, which, in terms of image resolution, may translate to just a few pixels. Furthermore, the variety of environments where rally races occur adds to the complexity. These races can take place at night, in forests, on dirt tracks, in backlit conditions, and under various weather conditions such as snow and rain. Additionally, due to privacy concerns, we obtained a limited number of videos, many with short duration, resulting in a relatively small dataset. Despite this, the dataset offers promise for potential improvements with a larger number of videos.

## 1.5 Problem statement

The thesis aims to utilize YOLO's architecture to develop a network capable of effectively performing real-time human detection, meeting hardware requirements, particularly in the complex context of the WRC. This environment introduces several complexities to the task, compounded by the constraint of a limited dataset. Our research involved a comprehensive analysis of YOLO's performance, including the creation of a dataset tailored to our specific use case for network training. We meticulously selected appropriate hyperparameters for training and explored methods for adapting to various challenging scenarios. Employing Data Augmentation techniques enhanced our flexibility in accommodating potential contextual changes. Lastly, we assessed the impact of image quality, which was notable given that our videos were obtained from a streaming platform provided by the company, resulting in lower compression compared to images directly captured by the vehicle's onboard system.

## 1.6 Structure

1. **Related works** contains a brief explanation of the history of Object detection algorithms with a focus on the most important research in the field of real-time human detection.

2. **Methodology** is where we analyze the structure of our dataset, with all the implementation choices that we made and all the augmentation techniques used.

3. **Experimental results** contains all the experimental details. These include a description of the experimental settings, a description of the evaluation metrics and an explanation of the architecture together with the reasoning behind the choices made. Moreover, in the same chapter we will present the results of the experiments, explaining and interpreting them using relevant tables and plots.

4. **Conclusions** is the chapter where we summarize our works and results. We also propose some ideas for future work on the subject.

5. **Appendix** contains additional material such as additional plots that would have impacted the readability of this report if they were introduced in the Experiments section.

# Chapter 2

# Related Works

Object detection is a fundamental task in computer vision that involves identifying and localizing objects within an image. This task requires not only recognizing what objects are present but also drawing bounding boxes around each detected object. This section explores the significant milestones and current advancements in object detection, converging toward the specialized and highly impactful domain of real-time object detection. It provides an overview of the state-of-the-art models and serves as a foundation for understanding the experiments that will be conducted in this thesis. In the following sections, we will begin with traditional methods that laid the groundwork for this field, then delve into the transformative impact of deep learning. Next, we will explore modern object detection techniques, distinguishing between single-shot detectors and region-based approaches. Finally, we will focus on the specialized domain of real-time object detection, examining the unique challenges and innovations that drive this critical application.

## 2.1 Traditional Approaches to Object Detection

The initial strides in object detection were marked by the use of traditional methods, which predominantly relied on handcrafted features and classical machine learning algorithms. These approaches laid the foundation for modern object detection techniques and provided valuable insights into the complexities of identifying and localizing objects within images. One of the earliest and most influential methods in object detection was the Viola-Jones detector[1], introduced by Paul Viola and Michael Jones in 2001. This approach was groundbreaking due to its real-time face-detection capabilities, which were achieved through the use of Haar-like features and the AdaBoost algorithm. Haar-like features are simple rectangular features that capture the contrast between different regions of an image, while AdaBoost is a machine learning algorithm capable of selecting the most relevant features and combining them into a strong classifier, enhancing detection accuracy. Despite its success in face detection, the Viola-Jones detector faced limitations when applied to more general object detection tasks. Another significant advancement in traditional object detection was the introduction of Histogram of Oriented Gradients (HOG)[2] in 2005. HOG features capture the distribution of gradient orientations in localized portions of an image, making them robust to variations in illumination and small deformations. The combination of HOG features with a Support Vector Machine (SVM)[3] classifier became a popular method for pedestrian detection. Despite their historical significance, these methods had limitations in handling the variability and complexity of real-world object appearances, especially regarding scalability, computational efficiency and the extensive domain knowledge required to handcraft features.

## 2.2 The Deep Learning Revolution

The advent of deep learning marked a significant paradigm shift in object detection, revolutionizing the field with the introduction of convolutional neural networks (CNNs)[4]. This section delves into the key developments and landmark models that have led object detection to new heights, providing more powerful and generalizable feature representations.

### 2.2.1 Regions with CNN features

The R-CNN model[5], introduced by Ross Girshick et al. in 2014, was a pioneering effort that integrated CNNs into the object detection pipeline. It generated region proposals using selective search to identify potential object locations within an image. Features were extracted from each region proposal using a CNN, significantly improving feature quality over handcrafted methods. The extracted features were then classified using SVMs, which separated feature extraction from classification. In [5] the authors demonstrate outperforming results in terms of accuracy for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[6] compared to the classical methods. While R-CNN significantly improved detection accuracy, its major drawback was computational inefficiency, as each region proposal needed to be processed independently by the CNN.

### 2.2.2 Further enhancement of R-CNN

Fast R-CNN[7], also developed by Ross Girshick, was introduced to address the computational inefficiencies of R-CNN. This model improved upon its predecessor by incorporating several key innovations. Fast R-CNN performs a single forward pass over the entire image, which allows for the extraction of feature maps from the image using a CNN. These feature maps are then used to compute features for each region proposal through a process known as Region of Interest (RoI) pooling. This technique allows the network to handle variable-sized input regions efficiently. Fast R-CNN also integrates classification and bounding box regression into a single network, allowing for end-to-end training, which significantly enhances both efficiency and performance. The result is a model that is much faster during both training and testing phases compared to the original R-CNN, making it more suitable for practical applications and reaching a higher accuracy on PASCAL VOC 2012[8].

### 2.2.3 Faster R-CNN

Faster R-CNN[9], proposed by Shaoqing Ren et al., further streamlined the object detection pipeline by introducing Region Proposal Networks (RPNs). This advancement allowed for the generation of region proposals directly from the convolutional feature maps produced by CNN, eliminating the need for external proposal methods like selective search. The RPN component of Faster R-CNN shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. By integrating RPNs with the Fast R-CNN detector, Faster R-CNN creates a unified, end-to-end trainable network that improves both the speed and accuracy of object detection. This innovative approach set new benchmarks for object detection performance, combining high detection accuracy with enhanced computational efficiency, and significantly reducing the processing time required for generating region proposals.

## 2.3 Real-time Object Detection

Even if region-based approaches, particularly Faster R-CNN and its variants, continue to lead in terms of detection precision, real-time constraints necessitate a delicate balance between accuracy and speed, prompting the development of lightweight architectures and optimization techniques. Contrasting with region-based models the Single Shot Multibox Detector (SSD)[10] and You Only Look Once (YOLO)[11] offer a streamlined approach to object detection. Both SSD and YOLO eliminate the need for a separate proposal stage, opting instead for single-pass architectures that achieve high speed and efficiency. These models are particularly suited for real-time applications like surveillance systems, pedestrian safety and autonomous vehicles, enabling responsive human-computer interaction interfaces.

### 2.3.1 Single Shot Multibox Detector

The Single Shot Multibox Detector (SSD)[10], introduced by Wei Liu et al., advanced object detection by merging speed and accuracy. It utilizes a single convolutional network to predict object classes and bounding boxes across multiple scales and aspect ratios without a separate region proposal phase. SSD's use of multi-scale feature maps allows it to effectively detect objects of various sizes, while its default boxes handle different shapes and scales. This results in a real-time capable model that achieves a balance between speed and precision, making it ideal for applications demanding quick and reliable detection.

### 2.3.2 You Only Look Once

YOLO[11], developed by Joseph Redmon et al., transformed object detection by reimagining it as a single regression problem. In the first version, YOLO divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously. This approach enables the entire detection process to occur in a single forward pass, significantly boosting detection speed. YOLO captures contextual information about objects and their locations, which enhances detection accuracy. Its design allows for rapid image processing with high accuracy. Over the years, various researchers have developed multiple versions of YOLO, each introducing changes and improvements to the original model. These enhancements have progressively boosted the network's performance, enabling it to achieve outstanding results in terms of both speed and accuracy.

### 2.3.3 The choice of YOLO

Our use case necessitates evaluating which network to select, given the critical requirement for high accuracy coupled with the need for a very fast system capable of streaming object detection results in real time. This evaluation is further influenced by the constraint of using hardware that is not prohibitively expensive, thus having fairly limited resources. After considering these factors, we leaned towards choosing YOLO, specifically YOLOv8[12], due to its excellent balance of speed, accuracy, and efficiency on resource-constrained hardware. Nonetheless, YOLO is known for its superior speed, which is crucial for real-time applications. YOLO achieves real-time processing capabilities by framing object detection as a single regression problem, simplifying the pipeline and reducing computation time significantly. Unlike SSD[10], which generates multiple feature maps for different scales, YOLO uses a single neural network to predict both the bounding boxes and class probabilities in one evaluation, resulting in faster detection times. As depicted in [13] YOLO tends to be more efficient in terms of processing power. Its architecture, particularly in its later versions like YOLOv3[14] and YOLOv4[15], is designed to balance accuracy and speed effectively. This makes it an excellent choice for environments with limited computational resources where maintaining high frame rates is essential. It was determined in [13] that YOLOv3[14] is the fastest, with SSD following closely behind and Faster R-CNN[9] being the slowest. However, the choice of algorithm depends on the specific use case. For instance, if the dataset is relatively small and real-time results are not required, Faster R-CNN[9] is preferable due to its higher accuracy. Conversely, YOLOv3[14] is the best choice for analyzing live video feeds due to its superior speed. On the other hand, SSD[10] provides a good balance between speed and accuracy. Comparing YOLO directly with SSD[10], research on real-time pill identification systems[16] has shown that YOLO can achieve high detection speeds while maintaining competitive accuracy, making it a preferred choice in time-sensitive applications. All of these analyses were conducted on versions of YOLO before YOLOv5[17]. Consequently, the results from these studies, combined with the improvements found in newer versions of YOLO, led us to choose YOLO as the foundation for our thesis.

## 2.4 History of YOLO

Understanding and analyzing the structure of YOLO is essential for grasping its functionality. At the same time, examining the updates and improvements introduced in each successive version is useful for comprehending the network's evolution step by step, culminating in the final form of the version we are using.

### 2.4.1 YOLOv1

YOLO[11] by Joseph Redmon et al. was published in 2016. It was the first method that proposed a new point of view for object detection, as an end-to-end method approaching real-time. This method differs significantly from earlier techniques. Traditional approaches relied on sliding windows paired with a classifier, necessitating numerous runs per image. More sophisticated strategies divided the process into two stages: initially pinpointing potential object regions (region proposals) and subsequently applying a classifier to these regions. YOLO, short for "You Only Look Once", performs this task in a single pass through the network. It simplifies this by combining these steps into one, making the detection process faster and more efficient.

YOLOv1 treated object detection as a regression problem to spatially separated bounding boxes and class probabilities directly from full images in one evaluation. It divided the input image into an $S \times S$ grid and predicted $B$ bounding boxes and $C$ class probabilities for each grid cell, leading to a unified prediction framework outputting a tensor of size $S \times S \times (B \times 5 + C)$. The number 5 in the dimension of the generated tensor is justified because each bounding box prediction includes five values: $Pr, x, y, h, w$. The variable $Pr$ denotes the confidence score, which reflects both the likelihood of the box containing an object and its accuracy. The coordinates $x$ and $y$ represent the center of the box relative to the grid cell, while $h$ and $w$ denote the height and width of the box relative to the entire image.

The YOLOv1 architecture comprises 24 convolutional layers followed by two fully connected layers. These components serve the crucial function of predicting bounding box coordinates and probabilities. Throughout the network, except for the final layer, leaky rectified linear unit (leaky ReLU[18]) activations are employed. To enhance computational efficiency and minimize parameter redundancy, YOLOv1 strategically incorporates $1 \times 1$ convolutional layers. Table 2.1 outlines the YOLOv1 architecture.

The authors started by pre-training the initial 20 layers of YOLOv1 on the ImageNet dataset[20] at a resolution of $224 \times 224$. Following this, they appended the last four layers with weights initialized randomly. To improve object detection accuracy, the model went under fine-tuning on the PASCAL VOC 2007[19] and VOC 2012[8], utilizing an increased resolution of $448 \times 448$ to capture finer details. To augment the data, random scaling and translations were employed, adjusting up to 20% of the input image size. Additionally, random exposure and saturation were adjusted with a maximum factor of 1.5 in the HSV color space.

YOLOv1 utilizes a loss function composed of multiple sum-squared errors to train the network:

$$\begin{aligned}
&\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(x_i-\hat{x_i})^2+(y_i-\hat{y_i})^2] \\
&+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w_i}})^2+(\sqrt{h_i}-\sqrt{\hat{h_i}})^2] \\
&+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(C_i-\hat{C_i})^2] \\
&+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}[(C_i-\hat{C_i})^2] \\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in classes}[(p_i(c)-\hat{p_i}(c))^2]
\end{aligned} \tag{2.1}$$

where $\lambda_{coord}$ acts as a scaling factor that prioritizes the accuracy of bounding box predictions, while $\lambda_{noobj}$ scales down predictions for boxes without objects. The first two terms in the loss function account for localization loss, assessing the error in predicted bounding box coordinates $(x, y)$ and dimensions $(w, h)$. These terms are computed exclusively for boxes containing objects, as indicated by $\mathbb{1}_{ij}^{obj}$, ensuring penalties are applied only when objects are present in the respective grid cells.

**Table 2.1:** YOLOv1 structure. The sizes and outputs are referred to the original YOLO paper[11]. In this paper, the authors employed the PASCAL VOC[19] dataset comprising 20 classes (C = 20), utilizing a grid size of 7 × 7 (S = 7), and allowing for a maximum of 2 classes per grid element (B = 2). Consequently, the output prediction is structured as a 7 × 7 × 30 tensor.

| | Type | Filters | Size/Stride | Output |
|---|---|---|---|---|
| | Conv | 64 | 7 × 7 / 2 | 224 × 224 |
| | Max Pool | - | 2 × 2 / 2 | 112 × 112 |
| | Conv | 192 | 3 × 3 / 1 | 112 × 112 |
| | Max Pool | - | 2 × 2 / 2 | 56 × 56 |
| 1× | Conv | 128 | 1 × 1 / 1 | 56 × 56 |
| | Conv | 256 | 3 × 3 / 1 | 56 × 56 |
| | Conv | 256 | 1 × 1 / 1 | 56 × 56 |
| | Conv | 512 | 3 × 3 / 1 | 56 × 56 |
| | Max Pool | - | 2 × 2 / 2 | 28 × 28 |
| 4× | Conv | 256 | 1 × 1 / 1 | 28 × 28 |
| | Conv | 512 | 3 × 3 / 1 | 28 × 28 |
| | Conv | 512 | 1 × 1 / 1 | 28 × 28 |
| | Conv | 1024 | 3 × 3 / 1 | 28 × 28 |
| | Max Pool | - | 2 × 2 / 2 | 14 × 14 |
| 2× | Conv | 512 | 1 × 1 / 1 | 14 × 14 |
| | Conv | 1024 | 3 × 3 / 1 | 14 × 14 |
| | Conv | 1024 | 3 × 3 / 1 | 14 × 14 |
| | Conv | 1024 | 3 × 3 / 2 | 7 × 7 |
| | Conv | 1024 | 3 × 3 / 1 | 7 × 7 |
| | Conv | 1024 | 3 × 3 / 1 | 7 × 7 |
| | FC | - | 4096 | 4096 |
| | Dropout 0.5 | - | - | 4096 |
| | FC | - | 7 × 7 × 30 | 7 × 7 × 30 |

The third and fourth terms contribute to confidence loss: the third term measures confidence error when an object is detected, while the fourth term measures error when no object is detected. Given that many boxes remain empty, $\lambda_{noobj}$reduces the weight of this loss. Finally, the classification loss component evaluates the squared error of class conditional probabilities, applied only when an object($\mathbb{1}_i^{obj}$) is detected in the cell.

### 2.4.2 YOLOv2

YOLOv2[21], released in 2017, marked a significant advancement from YOLOv1[11], maintaining speed while enhancing robustness. One key innovation was the integration of anchor boxes, predefined shapes tailored to typical object geometries. Multiple anchor boxes per grid cell were used to predict coordinates and class probabilities, directly influencing the network's output size. To optimize anchor box selection, the authors employed k-means clustering on training data, ultimately adopting five anchor boxes that balanced recall and model complexity effectively.

As it can be seen at Table 2.2, batch normalization was universally applied across all convolutional layers in YOLOv2, facilitating faster convergence and acting as a regularizer to mitigate overfitting. By eliminating fully connected layers, YOLOv2 achieved flexibility in handling inputs of varying sizes. During training, the model's adaptability to diverse input dimensions was reinforced by periodically varying input sizes every ten batches. The backbone architecture, Darknet-19, comprised layers 1 through 23, encompassing 18 convolutional and five max-pooling layers. Notably, the 19th convolutional layer of Darknet-19 functioned as the object detection head, containing 1000 filters, followed by an Average Pooling layer and a Softmax layer in its original configuration.

### 2.4.3 YOLOv3

Here we present the main changes from YOLOv2[21] introduceb by YOLOv3[14], published in 2018 by Joseph Redmon and Ali Farhadi.

In YOLOv3, bounding box prediction is handled similarly to earlier versions, but with a significant change: the confidence score is replaced by an objectness score calculated using logistic regression. This score is set to 1 for the anchor box with the highest overlap with the ground truth object and 0 for all other anchor boxes, ensuring each object is matched to a single anchor box. When an object isn't matched to any anchor box, only the classification loss is applied, which avoids penalties for localization and confidence.

Moreover, YOLOv3 changes its approach to classification. Instead of using a softmax function, it employs binary cross-entropy loss to train individual logistic classifiers, treating classification as a multilabel problem. This change allows a

**Table 2.2:** YOLOv2 structure. The sizes and outputs are referred to the original YOLOv2 paper[21]

| Num | Type | Filters | Size/Stride | Output |
|---|---|---|---|---|
| 1 | Conv/BN | 32 | 3 × 3 / 1 | 224 × 224 |
| 2 | Max Pool | - | 2 × 2 / 2 | 112 × 112 |
| 3 | Conv/BN | 64 | 3 × 3 / 1 | 112 × 112 |
| 4 | Max Pool | - | 2 × 2 / 2 | 56 × 56 |
| 5 | Conv/BN | 128 | 3 × 3 / 1 | 56 × 56 |
| 6 | Conv/BN | 64 | 1 × 1 / 1 | 56 × 56 |
| 7 | Conv/BN | 128 | 3 × 3 / 1 | 56 × 56 |
| 8 | Max Pool | - | 2 × 2 / 2 | 28 × 28 |
| 9 | Conv/BN | 256 | 3 × 3 / 1 | 28 × 28 |
| 10 | Conv/BN | 128 | 1 × 1 / 1 | 28 × 28 |
| 11 | Conv/BN | 256 | 3 × 3 / 1 | 28 × 28 |
| 12 | Max Pool | - | 2 × 2 / 2 | 14 × 14 |
| 13 | Conv/BN | 512 | 3 × 3 / 1 | 14 × 14 |
| 14 | Conv/BN | 256 | 1 × 1 / 1 | 14 × 14 |
| 15 | Conv/BN | 512 | 3 × 3 / 1 | 14 × 14 |
| 16 | Conv/BN | 256 | 1 × 1 / 1 | 14 × 14 |
| 17 | Conv/BN | 512 | 3 × 3 / 1 | 14 × 14 |
| 18 | Max Pool | - | 2 × 2 / 2 | 7 × 7 |
| 19 | Conv/BN | 1024 | 3 × 3 / 1 | 7 × 7 |
| 20 | Conv/BN | 512 | 1 × 1 / 1 | 7 × 7 |
| 21 | Conv/BN | 1024 | 3 × 3 / 1 | 7 × 7 |
| 22 | Conv/BN | 512 | 1 × 1 / 1 | 7 × 7 |
| 23 | Conv/BN | 1024 | 3 × 3 / 1 | 7 × 7 |
| 24 | Conv | 1000 | 1 × 1 / 1 | 7 × 7 |
| 25 | Avgpool | - | Global | 1000 |
| 26 | Softmax | - | - | - |

single box to have multiple labels, adapting to datasets with overlapping categories, such as an object being labeled both as a *Person* and a *Woman*[22].

YOLOv3[14] introduces the Darknet-53 backbone architecture, detailed in Table 2.3. This architecture abandoned max-pooling layers, opting instead for strided convolutions. A notable feature is the inclusion of residual connections, which link the input from the $1 \times 1$ convolutions directly to the output of the $3 \times 3$ convolutions. Darknet-53 is composed of 53 convolutional layers, each equipped with batch normalization, enhancing both the stability and efficiency of the network.

**Table 2.3:** Backbone of YOLOv3 structure. The sizes and outputs are referred to the original YOLO paper[11]. The architecture presented here comprises solely the backbone without including the detection head, which is responsible for making multi-scale predictions.

| | Type | Filters | Size/Stride | Output |
|---|---|---|---|---|
| | Conv | 32 | $3 \times 3$ / 1 | $256 \times 256$ |
| | Conv | 64 | $3 \times 3$ / 2 | $128 \times 128$ |
| 1× | Conv | 32 | $1 \times 1$ | |
| | Conv | 64 | $3 \times 3$ | |
| | Residual | | | $128 \times 128$ |
| | Conv | 128 | $3 \times 3$ / 2 | $64 \times 64$ |
| 2× | Conv | 64 | $1 \times 1$ | |
| | Conv | 128 | $3 \times 3$ | |
| | Residual | | | $64 \times 64$ |
| | Conv | 256 | $3 \times 3$ / 2 | $32 \times 32$ |
| 8× | Conv | 128 | $1 \times 1$ | |
| | Conv | 256 | $3 \times 3$ | |
| | Residual | | | $32 \times 32$ |
| | Conv | 512 | $3 \times 3$ / 2 | $16 \times 16$ |
| 8× | Conv | 256 | $1 \times 1$ | |
| | Conv | 512 | $3 \times 3$ | |
| | Residual | | | $16 \times 16$ |
| | Conv | 1024 | $3 \times 3$ / 2 | $8 \times 8$ |
| 4× | Conv | 512 | $1 \times 1$ | |
| | Conv | 1024 | $3 \times 3$ | |
| | Residual | | | $8 \times 8$ |

A significant enhancement in YOLOv3 is its ability to make multi-scale predictions, effectively addressing the previous versions' struggles with detecting smaller objects. This feature involves generating predictions at different grid sizes, which greatly improves the model's versatility and accuracy across various object scales.

Here's how the multi-scale detection mechanism operates: The initial predictions are made on an $8 \times 8$ grid as the final output of the network. For the second set of predictions, feature maps from the 2 previous layers of Darknet-53 are merged with those from the Res $\times 8$ layer. To permit this, the smaller $8 \times 8$ feature maps are upsampled before concatenation with the larger $16 \times 16$ maps. This technique is performed twice, therefore the final set of predictions involves combining the $16 \times 16$ feature maps with even larger $32 \times 32$ feature maps, again using an upsampling step.

Regarding the establishment of bounding box priors, YOLOv3 continues the practice of using k-means clustering. However, YOLOv3 implements a more sophisticated approach by distributing three anchor boxes per grid cell across three different scales. This change enhances the model's ability to predict bounding boxes more accurately across a wider range of object sizes.

### 2.4.4 YOLOv4

YOLOv4[15] was published in April 2020 by different authors. Its improvements and performances made it considered the official YOLOv4 by the community. YOLOv4 introduced a variety of enhancements categorized as **bag-of-freebies** (Bof) and **bag-of-specials** (Bos), aiming for an optimal balance between training complexity and inference efficiency. The **bag-of-freebies** involves methods that modify the training process, increasing training costs without affecting inference time, such as data augmentation. Conversely, **bag-of-specials** comprises techniques that slightly increase inference costs but significantly improve accuracy.

Within the *bag-of-freebies*, the authors extended beyond standard augmentations like random brightness, contrast, scaling, cropping, flipping, and rotation adjustments. They introduced mosaic augmentation, which combines four images into one. This technique helps detect objects in varied contexts and reduces the need for large mini-batch sizes in batch normalization. Additionally, they integrated CIoU loss[23] and Cross mini-batch normalization (CmBN), which aggregates statistics across the entire batch instead of individual mini-batches. To further strengthen model robustness against perturbations, self-adversarial training (SAT) was implemented. This method tricks the model into thinking the ground truth object is absent, yet the original label leads to accurate detection.

For the *bag-of-specials*, various backbone architectures were explored. The most effective one was a customized version of Darknet-53[14] enhanced with cross-stage partial connections (CSPNet)[24], reducing computational load while maintaining accuracy, along with the Mish activation function[25]. The neck, connecting the backbone to the head, aggregates and refines features, improving spatial and semantic information across different scales. They used a modified spatial pyramid pooling (SPP) that concatenates multiple max pooling outputs with different kernel sizes $k \times k$ where $k = 1, 5, 9, 13$, allowing a larger receptive field without subsampling (stride = 1). Multi-scale predictions, similar to YOLOv3, were employed, but with a modified path aggregation network (PANet)[26] where features are concatenated rather than added. Additionally, a refined spatial attention module (SAM)[27] was used, which slightly increases training calculations but does not affect GPU inference speed and improves accuracy. Anchors were employed in the detection head, following YOLOv3 principles, resulting in the model being named CSPDarknet53-PANet-SPP.

### 2.4.5 YOLOv5

In 2020, Glen Jocher introduced YOLOv5[17] shortly after the release of YOLOv4. This version integrated many of the advancements from YOLOv4 but was developed using PyTorch instead of Darknet. YOLOv5 features an innovative algorithm called AutoAnchor by Ultralytics, which optimizes anchor boxes according to the dataset and training configurations, such as image size. Initially, the k-means algorithm is applied to dataset labels to generate starting conditions for a Genetic Evolution (GE) algorithm, which refines these anchors over 1000 generations using CIoU[23] loss and Best Possible Recall as the fitness function.

YOLOv5's architecture includes a modified CSPDarknet53 backbone. The architecture also features the SPPF (Spatial Pyramid Pooling Fast) layer, which speeds up network computation by pooling features of different scales into a fixed-size feature map and upsample layers that enhance the resolution of feature maps. Each convolution layer is followed by batch normalization and SiLU[28] activation. The neck utilizes SPPF and a modified CSP-PAN, while the head maintains a structure similar to that of YOLOv3.

For data augmentation, YOLOv5 employs several techniques, including Mosaic, HSV augmentation, and random horizontal flips, along with additional augmentations from the albumentations package. These augmentations help improve the model's robustness and performance. Furthermore, YOLOv5 enhances grid sensitivity, making it more stable and less prone to runaway gradients.

YOLOv5 comes in five scaled versions to accommodate various hardware constraints and application needs: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large). These versions vary in the width and depth of their convolution modules, allowing users to select the model that best fits their resource availability and performance requirements. For instance, YOLOv5n and YOLOv5s are lightweight models designed for low-resource devices, while YOLOv5x is optimized for high performance, albeit with a trade-off in speed.

In conclusion, YOLOv5 provides a robust, flexible, and efficient approach to object detection, with its PyTorch implementation facilitating further development and integration. The model's scalability, advanced features like AutoAnchor, and diverse augmentation techniques make it a versatile choice for a wide range of applications, from edge devices to high-performance systems.

### 2.4.6 YOLOv6

Released on ArXiv in September 2022 by Meituan Vision AI Department, YOLOv6[29] brings a series of innovations that enhance its efficiency and performance. At its core, YOLOv6 integrates an advanced backbone featuring blocks such as RepVGG[30] and CSPStackRep, which were initially presented in the CSPNet paper[24] and are typically utilized in larger models. This structure includes a PAN[26] topology for the neck and introduces a decoupled head, which separates the classification and regression tasks. This separation helps address the misalignment issues between classification confidence and localization accuracy[31][32], leading to improvements in precision and faster model convergence.

YOLOv6 also adopts the Task Alignment Learning approach from the TOOD paper[33] for more effective label assignment, which resolves ambiguities in overlapping box classifications[34]. Additionally, it employs enhanced quantization techniques, such as post-training quantization[35] and channel-wise distillation[36], contributing to faster and more accurate detectors. These innovations collectively allow YOLOv6 to surpass the performance of previous state-of-the-art models, including YOLOv5, by improving both accuracy and speed metrics.

### 2.4.7 YOLOv7

Published on ArXiv in July 2022 by the developers behind YOLOv4, YOLOv7[37] introduces significant architectural upgrades and a suite of *bag of freebies* techniques, aiming to enhance accuracy without sacrificing inference speed, though at the expense of extended training times. One of the major architectural innovations is the Extended Efficient Layer Aggregation Network (E-ELAN). Building on the principles of ELAN[38], which focuses on optimizing learning and convergence in deep models by efficiently managing gradient paths, E-ELAN furthers this by employing a strategy of shuffling and merging cardinalities across different groups. This technique enhances the network's learning capabilities while maintaining the integrity of the original gradient path.

YOLOv7 also redefines scaling methods for concatenation-based models. Traditional depth scaling adjusts input-output channel ratios in transition layers to conserve hardware resources. However, YOLOv7 introduces a unique scaling strategy that proportionally scales both the depth and width of blocks, maintaining the optimal structure of the model and achieving scalability across different model sizes.

Several new *bag of freebies* elements stand out in YOLOv7. Drawing inspiration from re-parameterized convolutions (RepConv) used in RepVGG[30] blocks, the researchers discovered that identity connections within RepConv could eliminate the residuals of network blocks. To counter this, they removed the identity connection, resulting in a modified approach named RepConvN. Furthermore, they implemented a balanced label assignment strategy that evenly distributes training responsibilities between the auxiliary head, which aids in the training process, and the lead head, which generates the final output. For the final inference model, the researchers adopted the Exponential Moving Average (EMA), enhancing the model's stability and performance.

In summary, YOLOv7 integrates numerous innovations that significantly improve its performance and scalability, making it a formidable advancement over its predecessors.

### 2.4.8 YOLOv8

YOLOv8[12], introduced by Ultralytics in January 2023 as a successor to YOLOv5[17], offers five different versions catering to various scales: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra large). This iteration maintains a backbone similar to YOLOv5 while integrating substantial enhancements to its architecture and functionality.

One of the notable upgrades in YOLOv8 is the C2f module, originally known as the CSPLayer. This module, featuring cross-stage partial bottlenecks with dual convolutions, plays a pivotal role in integrating contextual information with high-level features, thereby significantly augmenting detection accuracy.

Furthermore, YOLOv8 incorporates a Spatial Pyramid Pooling Fast (SPPF), utilizes the sigmoid function (SiLU[28]) to determine objectness scores, indicating the likelihood of objects within bounding boxes, and employs softmax for class probabilities, assessing the likelihood of objects belonging to different classes.

To refine object detection capabilities, YOLOv8 integrates advanced loss functions such as CIoU[23] and DFL[39] for bounding box accuracy, alongside binary cross-entropy for classification tasks. These methodologies have proven effective in enhancing the model's ability to detect smaller objects with precision.

From a practical standpoint, YOLOv8 supports seamless deployment via command line interface (CLI) execution and PIP installation, facilitating straightforward integration into labeling, training, and deployment workflows. The ease of use we just described surely influenced our decision to adopt the latest version, facilitating the development, training, and testing process in our specific context.

However, although it is not to be underestimated, the reason we chose to use YOLOv8 is essentially because of its performance, both in terms of speed and accuracy and in terms of its adaptability to our use case. As we have just seen, it is well suited for small object recognition, which is still a very complex task, especially for real-time applications, but at the same time improved as much as possible in this version. Thus, considering the need to have to recognize people at large distances from the camera, which therefore results in a negligible number of pixels in the image, this feature prompted us to use this version. The architecture also was able to adapt to the very limited hardware environment in which it was developed, ensuring the desired applicability through the use of its smallest version, the YOLOv8n. The latter can guarantee good accuracy, in a real-time environment that would allow inference on the image and immediate return of the frame with the boxes already drawn in a time under 50 fps with our hardware, the speed at which our camera shoots the scene. Generally speaking, YOLOv8 demonstrates significantly improved performance compared to its predecessors. Generally speaking, YOLOv8 demonstrates significantly improved performance compared to its predecessors. This can be observed in [40], which illustrates that YOLOv8 achieves superior results in terms of both accuracy and speed. Specifically, there is a noticeable enhancement over previous release by Ultralytics, such as YOLOv5. Additionally, YOLOv8 outperforms other versions like YOLOv6 and YOLOv7, neither of which has a suitable version for our specific hardware environment.

This clearly depicts these advancements, highlighting **YOLOv8 as the optimal choice** due to its compatibility with our hardware constraints and its superior performance metrics across the board.

## 2.5 Human Detection Datasets

A key step in the development of our network involved adapting YOLO to our specific use cases, as the initial version produced poor results due to the challenging context of the WRC (World Rally Championship) compared to the typical images used to train the basic version of YOLOv8[12], such as those in the COCO[41], ImageNet[20] or PASCAL VOC[19] datasets. Factors such as the high speed of the camera, the small size of the people to be recognized, and unique lighting conditions necessitated the exploration of additional datasets that better aligned with our domain. Over the years, numerous datasets have been created and refined to provide benchmarks for the human detection task, aiding in the improvement and evaluation of detection models. Each dataset offers unique attributes that face different aspects of the detection task, ensuring comprehensive training and evaluation of detection algorithms.

For example, the KITTI[42] dataset is famous for autonomous driving research. It provides a wide array of sensor data, including high-resolution stereo images and 3D point clouds, collected in diverse driving scenarios directly from a camera installed on the top of a vehicle. The videos include scenarios from cities, highways and rural areas, tackling various driving speeds and densities of people in the collected data. This dataset's extensive annotations for object detection, tracking, and scene understanding have set benchmarks for evaluating the performance of algorithms in real-world driving environments. However, this dataset contained a limited number of images and too few detections of individuals per image to significantly and noticeably improve YOLO's performance, particularly when applied to our use case. Our use case involved samples with a very high density of individuals, such as crowds gathered around strategic points of the circuits. Consequently, our research focused on finding a dataset with a larger number of images and a significantly higher average number of individual detections per image.

CrowdHuman[43], as depicted in Table 2.4, is specifically designed to address the challenges of detecting humans in crowded scenes. It offers detailed annotations for occluded and overlapping persons. This dataset is particularly useful for improving the accuracy and robustness of human detection models in densely populated environments, which is critical for applications like surveillance and public safety. Despite the optimal characteristics of this dataset, the size of the individuals to be detected was significantly smaller compared to that presented in the dataset, and the network struggled to recognize entities of such a small scale.

| | KITTI | COCOPersons | CrowdHuman |
|---|---|---|---|
| # images | 3,712 | **64,115** | 15,000 |
| # persons | 2,322 | 257,252 | **339,565** |
| # ignore regions | 45 | 5,206 | **99,227** |
| # person/image | 0.63 | 4.01 | **22.64** |

**Table 2.4:** Volume, density and diversity across the various human detection datasets. To ensure equitable comparison, we exclusively present the statistics of the training subsets.

TinyPerson[44] focuses on detecting small-scale persons in images, a task often overlooked in standard datasets. This dataset includes images from various sources with annotations for tiny persons, making it an excellent resource for enhancing models' ability to detect small, distant, or partially visible individuals. Unfortunately, in addition to significantly worsening the detection of relatively nearby individuals, which still needed to be maintained at a high level, the images did not lead to improvements in identifying people with reduced sizes. This was likely due to the substantial domain and viewpoint differences between the dataset images and our case. The dataset images were predominantly captured from helicopters or high-mounted cameras along coastal areas for beach security, nothing comparable to the world of rally.

In the end, our choice was to implement an ad-hoc dataset to train and test our network. This led to a significant increase of the performance, facing all the issues that we had just presented. It will be described in the next section.

# Chapter 3

# Methodology

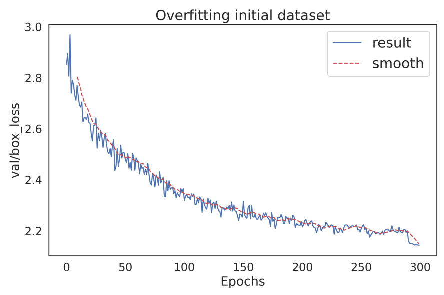
In this chapter, we will thoroughly analyze our choices made to enhance the YOLOv8 performance for our specific use case. We will describe the characteristics of the dataset we created to effectively address our specific domain. Additionally, we will discuss the application of data augmentation techniques, detailing the parameters used to tune them.

## 3.1 Dataset creation

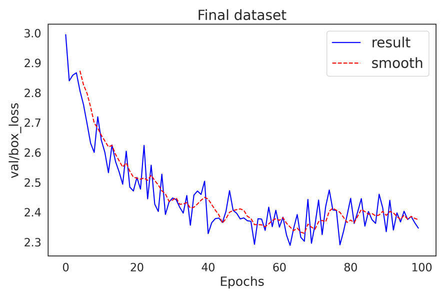
As outlined in Chapter 2, we conducted research to incorporate pre-trained datasets for training our network with images and labels tailored to our specific use case. Unfortunately, none of the datasets we evaluated met our performance criteria, which was crucial for the success and applicability of our project. Initially, due to restrictions regarding permissions and privacy concerns, we were limited to a single video from a special rally race of the previous season, consisting of 7 minutes of footage.

From this video, we created an initial dataset by excluding the stationary segments at the beginning and end and sampling one image every 15 frames, considering the camera's 50 frames per second rate to avoid nearly identical images. This approach yielded a dataset of 1121 images, manually annotated accordingly. This dataset served as our starting point for testing the network.

During the training phase, we observed that the validation loss, as illustrated in Figure 3.1a, continued to decrease even after an extended number of epochs, without reaching a minimum point. Despite this apparent improvement, visual inspection revealed that the network performed worse after 300 epochs compared to when trained for just 100 epochs. This discrepancy led us to conclude that the network was overfitting, likely due to the strong correlation between the training and testing data, which were derived from an 80% and 20% split of the aforementioned 1121 frames, respectively. These images were highly correlated, depicting the same track and race session with almost identical backgrounds.



**(a)** Trained for 300 epochs on the first dataset.



**(b)** Trained for 100 epochs on the final dataset.

**Figure 3.1:** Comparison of the validation losses. (a) shows the training result with 300 epochs. (b) shows the training result with 100 epochs on a final dataset. The first does not reach a minimum, while the second shows a better curve, crucial for selecting optimal weights for YOLOv8 inference.

To address this challenge, we secured access to another video featuring a race on a different circuit. From this new video, we extracted 499 frames, meticulously labeled them, and adopted this as our definitive testing dataset, while retaining the initial 1121 images for the training dataset. This adjustment not only bolstered the model's resilience and performance but also effectively resolved the overfitting issue, as evidenced by a more favorable trend in the validation loss graph (Fig. 3.1b), aligning with our expectations and allowing the architecture to choose the best configurations of weights to make inference, based on the ability of YOLOv8 to keep in memory the best configuration all over the past epochs.

## 3.2 Dataset characteristics

In this section, we aim to analyze the choices made regarding the dataset in terms of labeling and structure.

### 3.2.1 Person and Crowd

One primary objective was to identify humans or rather their presence, even when they appeared very small in the images. A critical issue that needed to be addressed was the presence of **crowds**. Often in WRC races, spectators are not spaced far apart but frequently gather in substantial crowds. From an image perspective, this makes it difficult to recognize the individual contours of people, rendering the resulting bounding boxes challenging to obtain and draw for training purposes. The presence and individuation of these crowds are as important as that of individuals watching the race separately, for whom labeling is considerably simpler.

Initially, we labeled both identifiable individuals and dense crowds of people as part of the same class *Person*. However, since they both fell under the same class despite having very different characteristics starting with the shape of the bounding box (usually a vertical rectangle for standing individuals or a square-like rectangle for seated ones, versus a horizontal rectangle often filled diagonally for crowds), we decided to separate the two classes as *Person* and *Crowd* as depicted in Figure 3.2. As shown in Figure 3.3, this significantly improved all the metrics we used to evaluate the various training choices, outperforming the single-class version. This likely arises from the fact that the network can create ad-hoc features separately for the two classes, thus enabling more effective recognition.

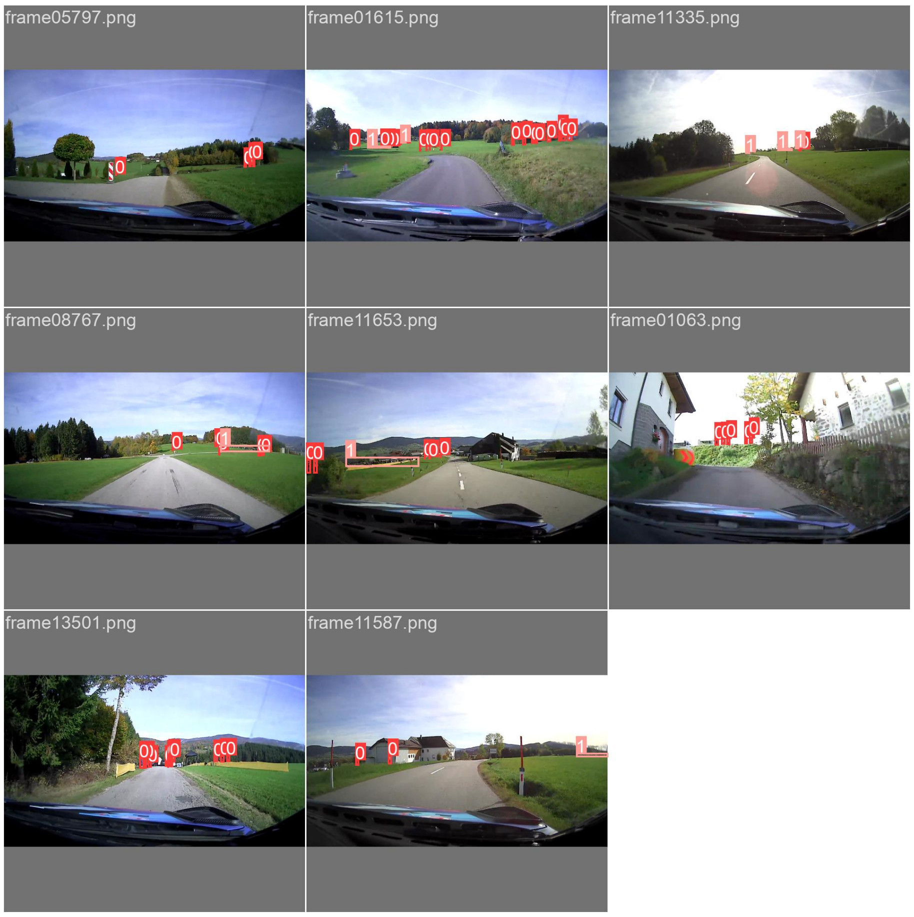


**Figure 3.2:** An example of 8 images batch derived from our initial dataset. The annotation corresponds to 0 for the *Person* class, or 1 for the *Crowd* class.

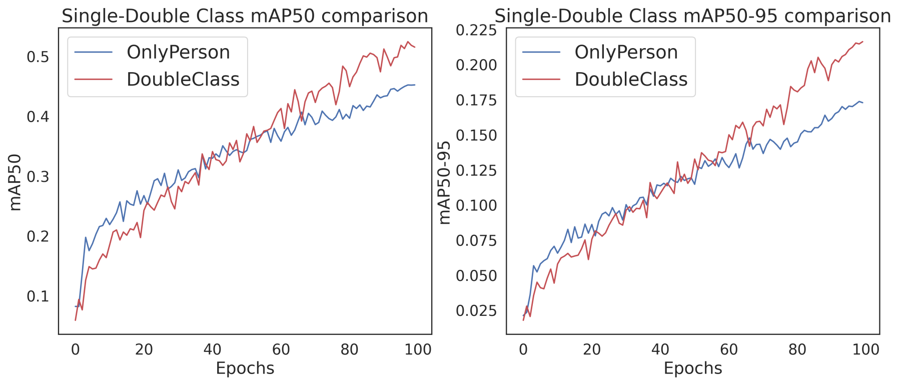


**Figure 3.3:** Comparison of the evaluation metrics between dataset version with only *Person* labels and the Double Class version including *Crowd* labels.

### 3.2.2 Risk recognition

The initial objective of this work was not only to identify the presence of humans but also to distinguish between those at risk and those who are safe. A thorough analysis supported by domain knowledge is crucial for advancing this task, likely by adopting additional technologies. One solution could be to segment and recognize the roadway and calculate the distance between the human and the road to assess the level of danger. Alternatively, more specific information such as speed, trajectory, and the person's position relative to a curve could be considered.

An initial attempt was made by training the network with a dataset designed to include these features. We introduced a distinction between the two main classes and two additional classes: *PersonAtRisk* and *CrowdAtRisk*. Although the results obtained, as shown in Figure 3.4, are somewhat sufficient, the reliability of this approach is compromised by the highly imbalanced number of labels.

Unfortunately, the instances of humans in danger detected by our network were very few and thus not reliable, as shown in Table 3.1. We believe that a postponed classification approach, built with a more precise and efficient ad-hoc pipeline, would be more effective and consistent.

| Class | Number of Instances |
|---|---|
| *Person* | 1999 |
| *Crowd* | 168 |
| *PersonAtRisk* | 18 |
| *CrowdAtRisk* | 3 |

**Table 3.1:** Number of instances of the *AtRisk* dataset for each type of entity in the testing dataset.

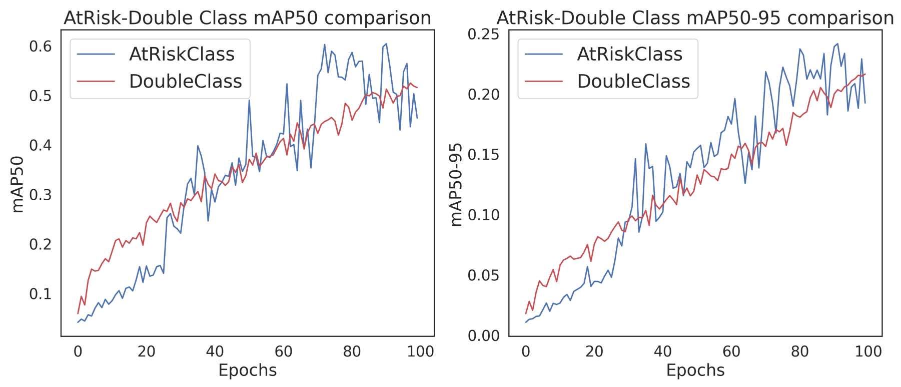


**Figure 3.4:** Comparison of the evaluation metrics between the Double Class version and dataset version with *PersonAtRisk* and *CrowdAtRisk* labels.

## 3.3 Data Augmentation

Data Augmentation techniques artificially increase the diversity of the training data by modifying the initial set of images, thereby enhancing the model's robustness and generalization capabilities. The transformations applied to the images are also applied to the corresponding bounding boxes. Each transformation has a parameter controlling either the probability of its application or the magnitude of the transformation itself. YOLOv8 provides a set of base augmentations and functions that users can easily modify to adjust the parameters influencing these augmentations, for which we present a summary in Table 3.2. An example of the application of the data augmentation techniques can be seen in Fig 3.5.

| Parameters | Probability of application |
|---|---|
| degree | 0 |
| translate | 0.1 |
| scale | 0.5 |
| hgain | 0.015 |
| sgain | 0.4 |
| vgain | 0.4 |
| hflip | 0.5 |
| vflip | 0 |
| Mosaic prob | 1.0 |
| Mosaic n.images | 4 |
| copy-paste | 0.5 |

**Table 3.2:** Parameters settings used for Data Augmentation

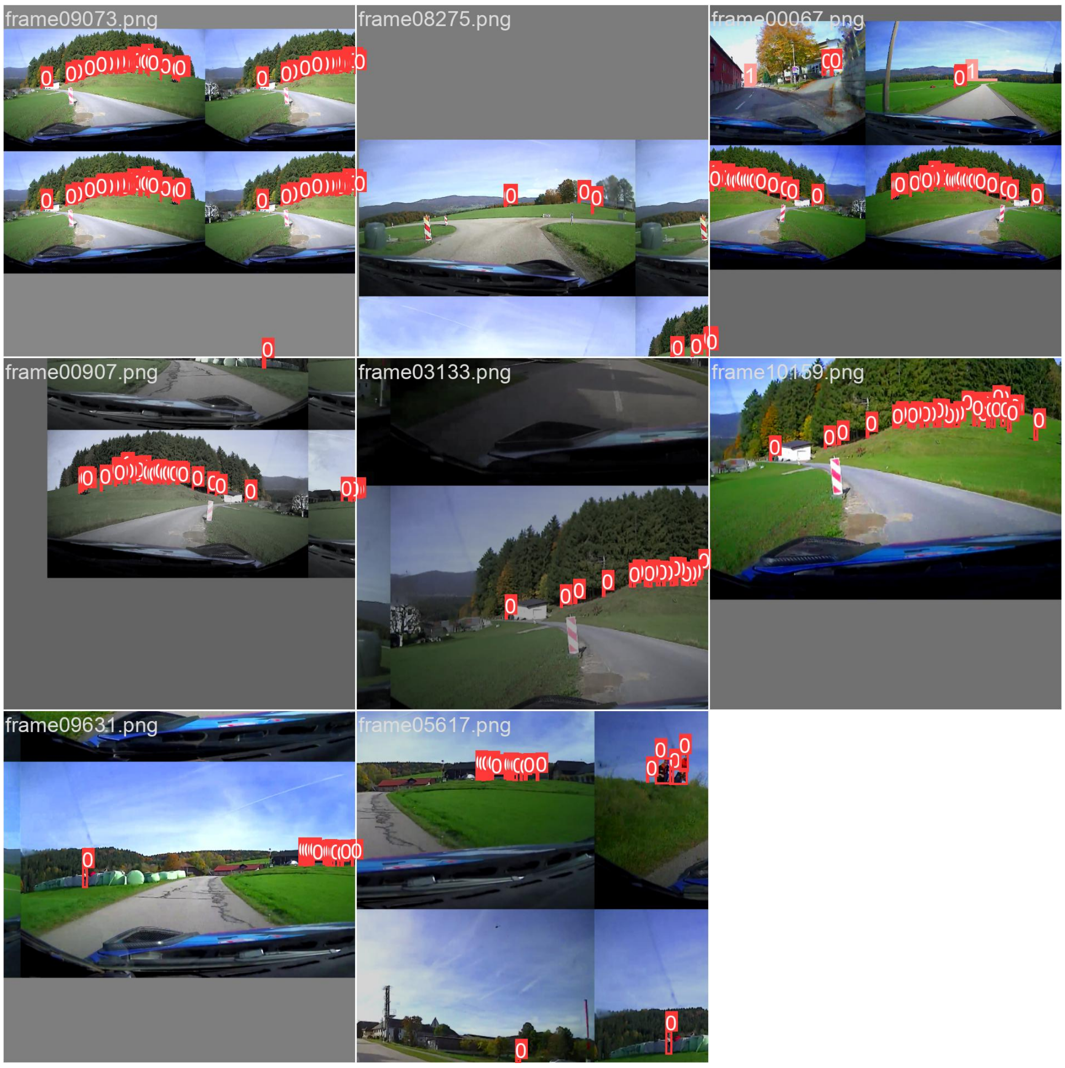


**Figure 3.5:** An example of an augmented batch of 8 images used for network training.

### 3.3.1 RandomPerspective

*RandomPerspective* is a class that includes rotation, translation, and scaling. Typically, the rotation parameter, which controls the degree of rotation, is set to 0. This setting is beneficial when dealing with images that have different perspectives or orientations. In our case, we maintained this policy due to the consistent perspective of our images, all taken from the same camera with the same orientation.

The translation parameter is set to 0.1, indicating the fraction of the total width and height for random translation in four directions. This means the image can be shifted by up to 10% of its dimensions, helping the model recognize objects even when slightly displaced.

The scaling factor allows resizing within a selected range. We used a scaling factor of 0.5, which permits resizing between 50% and 150% of the image's original size. This helps the model handle variations in object size more effectively.

### 3.3.2 Image Composition

In addition to the *RandomPerspective* class, other techniques modify the composition and positioning of elements within an image. One such tool is *RandomFlip*, which flips images horizontally (and more rarely, vertically) to create mirrored versions of the original images. This helps ensure that object orientation does not interfere with the network's ability to recognize them. The probability for horizontal flipping was set to 0.5, while vertical flipping was set to 0.

Another technique used is *Mosaic* augmentation, which combines multiple (four in our case) images into a single mosaic image. This augmentation was applied with a probability of 1. Mosaic augmentation is particularly effective for improving the detection of small objects and crowded scenes, as it allows the model to see various contexts and object arrangements.

Lastly, we employed *copy-paste* augmentation with a probability of 0.5. This technique involves copying objects or regions from one image and pasting them into another. It has been described in [45] and is useful for creating diverse training scenarios.

### 3.3.3 RandomHSV

The *RandomHSV* class modifies the Hue, Saturation, and Value (HSV) channels of an image through random adjustments. Hue (H) determines the type of color and is measured in degrees. Saturation (S) indicates the intensity or purity of the color, while Value (V) represents its brightness. Randomly altering these parameters helps the model adapt to varying color conditions, mimicking different times of day and diverse lighting and environmental settings encountered in real-world scenarios. The adjustments are constrained by default limits set by *hgain*, *sgain*, and *vgain*, each set to 0.5. In our experiments, we used a *hgain* of 0.015 to avoid significant color shifts in our images, while *sgain* and *vgain* were set to 0.4.

# Chapter 4

# Experimental results

This chapter provides a comprehensive overview of the experiments conducted during this Master's thesis. It begins with a detailed explanation of the experimental settings and evaluation metrics employed. Following this, we present the results of our work. Additionally, we explore specific scenarios of interest using a tailored approach. Finally, we include a section dedicated to studying the impact of image quality.

## 4.1 Experimental setup

The experiments were conducted using an NVIDIA T4 GPU with 16 GB of RAM, provided by the Google Colab environment. Training for the main experiments was carried out for 100 epochs with a batch size of 8 images. YOLOv8 includes an early stopping algorithm based on validation loss, simply checking if an improvement appears within 10 epochs from the best obtained so far. For most of the networks trained, we observed that epochs before reaching the early stopping threshold ranged between 90 and 100, with an average of approximately 93 epochs and none below 88 epochs. This led us to enforce training for 100 epochs to uniformly compare results. This was feasible because YOLOv8, even when forced to train for more epochs, can save all specifications and weights of its best version up to that point. The loss function referenced has already been introduced in the paragraph on YOLOv8 architecture in Chapter 2. The network employs the AdamW optimizer with weight decay equal to 0.0005, $\epsilon = 10^{-6}$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$. YOLOv8 typically employs a fixed learning rate of 0.01, but using the AdamW optimizer the architecture automatically selects the learning rate depending on the data structure and adapts the learning rate during the different epochs of the training process. Typically, in our experiments the final learning rate is in the order of $10^{-3}$. This is reached using a cosine annealing technique, applied after a warm-up phase of some epochs. The learning rate is updated using this equation:

$$lr(e) = \begin{cases} lr_0 \cdot \frac{e+1}{w_e} & \text{if } e < w_e \\ lr_f + (lr_0 - lr_f) \cdot \frac{1+cos(\pi \cdot e/n_e)}{2} & \text{else} \end{cases} \tag{4.1}$$

where $lr_0$ and $lr_f$ are the initial and final learning rate, $e$ is the current epoch, $n_e$ and $w_e$ are the total number of epochs and the warm-up epochs, respectively. The final trained version of our architecture was then tested using a JETSON XAVIER UNIT, that contains an NVIDIA XAVIER GPU, with 8 GB RAM to make inferences on videos and verify the applicability of the network to a real-world application. This environment is very similar to the most probable camera hardware setting that will be installed in the WRC cars when this project is completed and applied. These resources were kindly provided by Marelli s.p.a.

## 4.2 Evaluation metrics

To assess the performance of the network we used typical evaluation metrics. These metrics help determine how well the model is able to detect and classify persons and crowds in our images. The most commonly used metrics in object detection are precision, recall and mean average precision (mAP).

### 4.2.1 Precision and recall

Precision is the ratio of true positive detections to the total number of detections made by the model (both true positives and false positives), while recall is defined as the ratio of true positive detections to the total number of actual objects present in the dataset (true positives and false negatives). The first measures how the model identifies objects without mistakenly labeling background or other objects as the target objects. On the other hand, High recall indicates that the model is capable of detecting most of the objects present in the images, even if it also produces some false positives.

### 4.2.2 Mean Average Precsion

The mean average precision (mAP) is a comprehensive metric that encapsulates both precision($p$) and recall($r$) information. It is calculated as the mean of the average precision (AP) across all classes. To compute the AP for a single class, we plot the precision-recall curve and determine the area under this curve, providing a scalar representation of the model's performance for that class as:

$$AP = \int_{r=0}^{1} p(r)dr \tag{4.2}$$

In evaluating YOLOv8, we utilize two methods: mAP50 and mAP50-95. The distinction between these methods lies in the threshold value used to determine whether an object is detected. This threshold is based on the Intersection over Union (IoU), a metric that quantifies the overlap between the predicted bounding box and the ground truth box. The IoU is calculated as the ratio of the area of intersection to the area of union of the two boxes.

The mAP50 refers to the mean average precision for objects with an IoU of at least 50%. It is computed as:

$$mAP50 = \frac{1}{N}\sum_{i=1}^{n} AP_i \tag{4.3}$$

where N is the number of classes.

On the other hand, mAP50-95 is a more stringent and comprehensive metric. It averages the AP across multiple IoU thresholds, ranging from 50% to 95% in increments of 5%. This can be expressed as :

$$mAP50 - 95 = \frac{1}{N}\sum_{i=1}^{n}(\frac{1}{K}\sum_{j=1}^{k} AP_i(IoU_j)) \tag{4.4}$$

where N is the number of classes and K is the number of IoU thresholds, that are (0.5,0.55,0.6, ...,0.95).

This approach offers a more holistic view of the model's performance by considering various IoU thresholds, making it a more robust and reliable evaluation metric for object detection tasks. Analyzing mAP50 is still interesting because this metric is particularly useful for evaluating the network's efficiency in detecting smaller objects, where achieving higher IoU values is challenging due to the limited number of pixels, and minor detection inaccuracies can significantly reduce the IoU.

By employing both mAP50 and mAP50-95, we can gain a nuanced understanding of the model's capabilities, ensuring that it performs well across different object sizes and detection challenges.

## 4.3 Results

In this section, we aim to analyze the results in terms of both accuracy and computational performance of the network. Figures 4.1 and 4.2 present the evaluation metric curves and the two validation loss graphs for classification and bounding box detection.

Numerically, the results are suboptimal, particularly regarding the metrics used. Visually however, as we see in Figure 4.3, this discrepancy is not evident, meeting

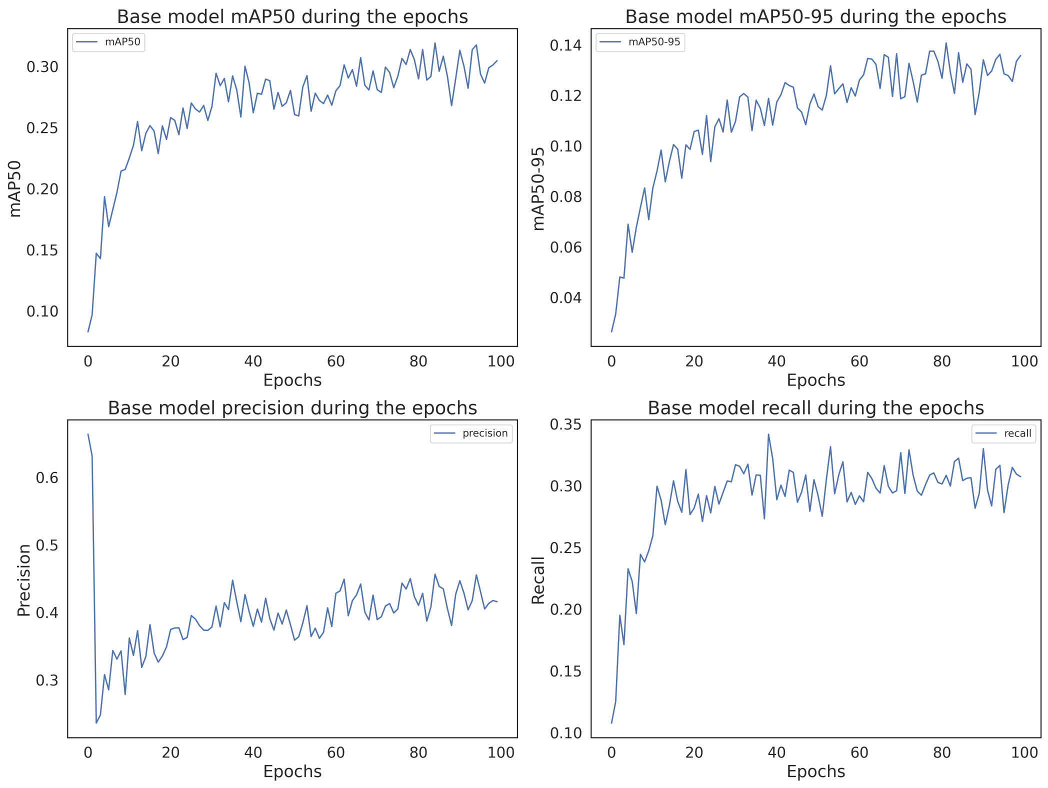


**Figure 4.1:** Plot of the evolution over the epochs of the 4 evaluation metrics for the base version of our model.

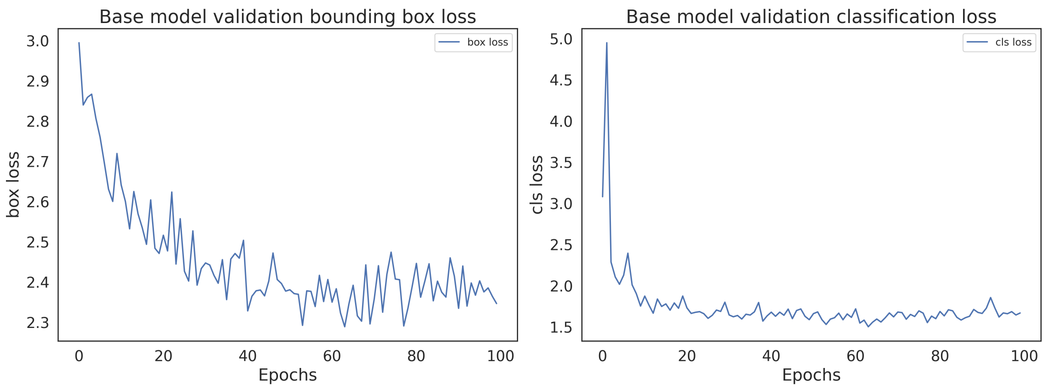


**Figure 4.2:** Plot of the two validation loss graphs. On the left, the figure represents the bounding box detection loss, while on the right it depicts the classification loss.

Marelli's qualitative standards. This prompted us to investigate the underlying cause.

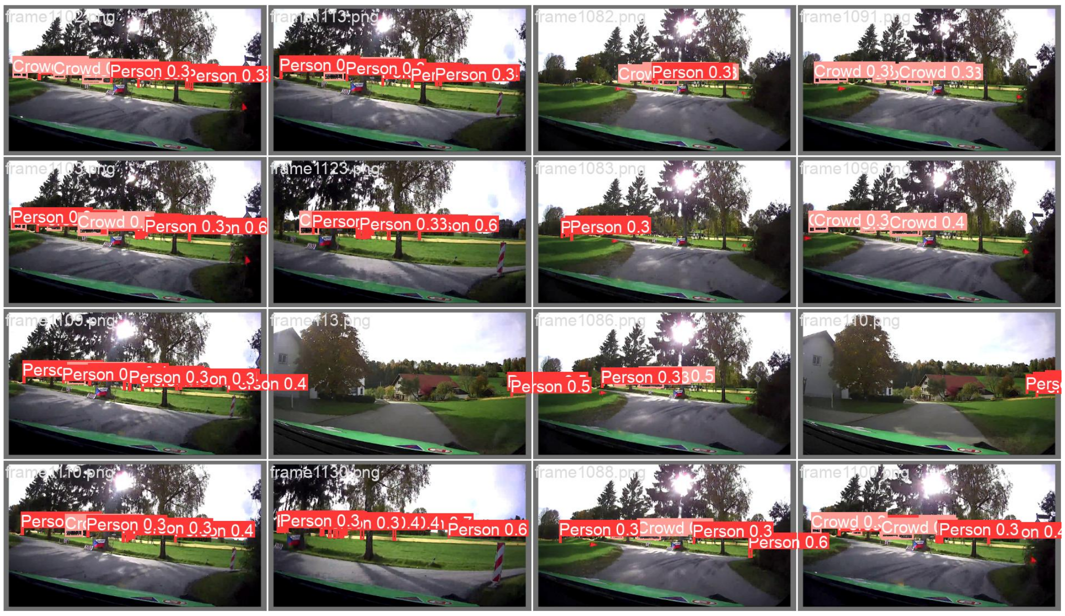


**Figure 4.3:** An example of 16 post-processed images batch, in which the predicted bounding boxes are drawn along with the respective label and a number representing the confidence of the prediction.

Figure 4.4 reveals a significant class imbalance. Consequently, we aimed to obtain differentiated results for the two classes using our best model version.

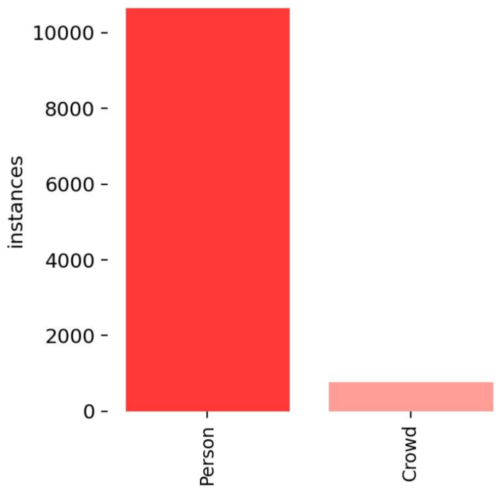


**Figure 4.4:** Histogram representing the number of detections for *Person* and *Crowd* classes considering both training and validation datasets.

The results, shown in Table 4.1, indicate that the performance for the *Person* class is markedly superior to that of the *Crowd* class across all metrics. Notably, the **mAP50** for *Person* recognition is very high, indicating effective detection of small individuals. This is crucial because, as shown in Figure 4.5, the bounding box sizes are relatively small, making high performance on the mAP50 metric optimal.

| Class | Instances | Precision | Recall | mAP50 | mAP50-95 |
|---|---|---|---|---|---|
| all | 5657 | 0.44 | 0.289 | 0.302 | 0.15 |
| *Person* | 5016 | 0.62 | 0.459 | 0.505 | 0.267 |
| *Crowd* | 641 | 0.261 | 0.12 | 0.0989 | 0.0337 |

**Table 4.1:** Evaluation metrics separated for each class considering the 499 validation images using the best weights for the network in the first 100 epochs.

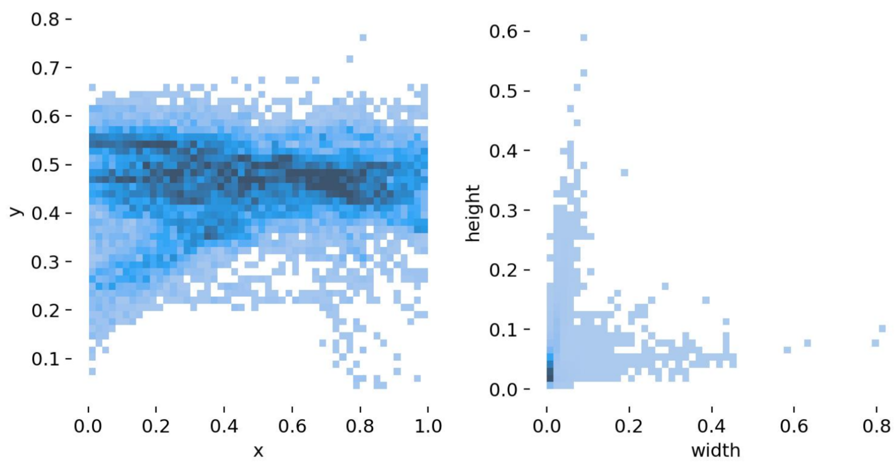


**Figure 4.5:** Distribution of bounding boxes. On the left, the distribution of the coordinates of the center of the bounding boxes. On the right, the distribution of the dimensions.

The network's challenges in recognizing crowds are likely due to class imbalance, with fewer instances to train on, and the ambiguous nature of crowd boundaries. Visually, this problem does not seem too impactful. Our goal was to recognize the presence of humans, whether alone or in groups. From an image processing and visual result perspective, this is achieved. However, the network assigns a single bounding box to each entity, leading to a significant drop in metrics for crowds. Many crowds are recognized as multiple closely packed *Person* entities, effectively breaking down the *Crowd* into many small persons.

While the results are satisfactory from our objective's perspective, numerically they are lacking for this reason. Future research should focus on developing a dedicated metric that highlights the quality of our results, considering both the *Crowd* label and a dense cluster of *Person* detections as equally accurate.

To demonstrate that the improvement achieved in the first 100 epochs is our target, we repeated the experiments, extending the epochs to 400, as shown in Figure 4.6. It is evident that after 100 epochs, both the classification and bounding box validation losses begin to rise. Simultaneously, the evaluation metrics remain constant or, in some cases, decline after the 100th epoch (Appendix A), confirming the quality of our choice.

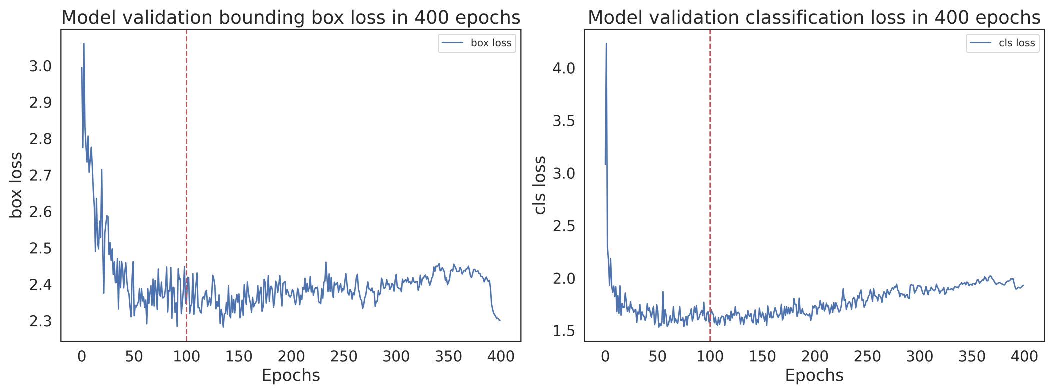


**Figure 4.6:** Validation loss graphs computed for 400 epochs. On the left, the figure represents the bounding box detection loss, while on the right it depicts the classification loss.

To validate the effectiveness of our augmentation techniques, we tested our model with all augmentation parameters turned off. The comparison shown in Figure 4.7 highlights the benefits of our approach, leading to improvements across the entire set of evaluation metrics.

From a hardware performance standpoint, Marelli aimed to develop a real-time system that could deliver processed images at the same speed or faster than the camera's capture rate. The camera used captures images at **50 fps**. This requires the system to capture an image and preprocess it, perform inference to detect entities in it and return the processed image with bounding boxes in under 20 ms.

Achieving this exact target was hazardous, as even a few frames per second could result in unclear image returns. However, by using the nano version of YOLOv8, we successfully met these standards, remaining below the half of our threshold. When testing the network on 499 validation dataset samples, the network averaged 0.2 ms for image preprocessing, 3.5 ms for inference, and 4.5 ms for post-processing. Consequently, the total time from image receipt to the return of the post-processed image averaged **8.2 ms**, perfectly aligning with our goals.

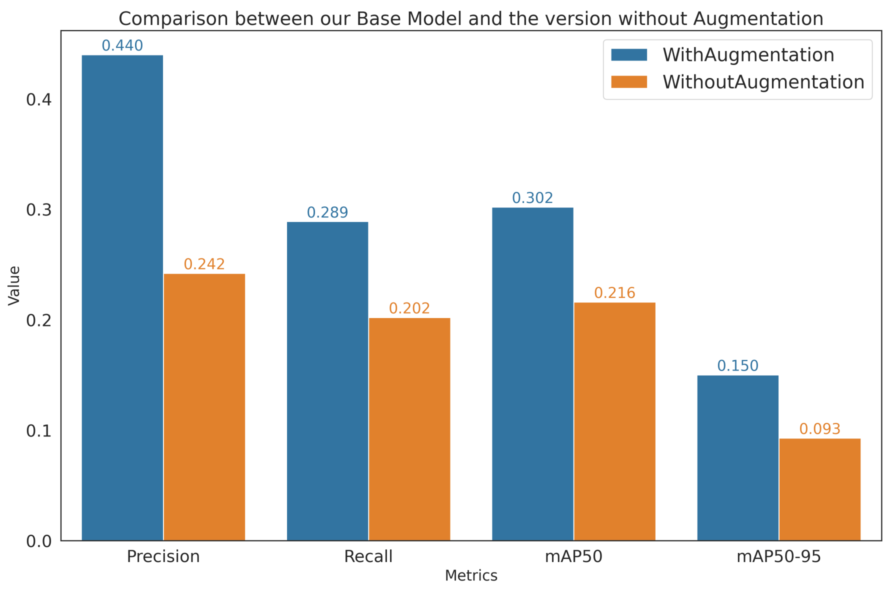


**Figure 4.7:** Comparison on the Evaluation Metrics between our base model with applied augmentation techniques and the version without any data augmentation process.

## 4.4 Special scenarios

The results displayed were obtained using the datasets described in Chapter 3. These datasets, as previously mentioned, are sourced from WRC championship races. While more significant than the original dataset, which only included data from a single race, this dataset uses one race for training and another similar race for testing, both with normal weather conditions and similar lighting and colors. Although this reflects the most common scenario in rally races, about 10-15% of WRC races occur under adverse conditions. This prompted us to investigate how these conditions might impact our network's performance.

To explore this issue, we obtained short videos from races with unusual conditions. Five short videos were provided: one with a snow-covered track under normal conditions, one with snow at night, another night race in a forest, one forest race in the rain, and one on a dirt track nearly in full backlight. The specifics of these new datasets are detailed in Table 4.2.

The limited data, compared to other dataset, produced less precise and smooth curves for both losses and evaluation metrics, but evaluating these datasets was necessary to understand our architecture's performance in these rarer but significant

| Dataset | *Backlight* | *Snow* | *Night* | *Night-snow* | *Rain* |
|---|---|---|---|---|---|
| tot images | 215 | 227 | 262 | 205 | 221 |
| train images | 172 | 182 | 210 | 166 | 177 |
| val images | 43 | 45 | 52 | 41 | 44 |
| *Person* tot instances | 442 | 1357 | 294 | 204 | 1526 |
| *Person* train instances | 337 | 1105 | 243 | 146 | 1202 |
| *Person* val instances | 105 | 252 | 51 | 58 | 324 |
| *Crowd* tot instances | 5 | 182 | 27 | 10 | 98 |
| *Crowd* train instances | 4 | 152 | 23 | 8 | 76 |
| *Crowd* val instances | 1 | 30 | 4 | 2 | 22 |

**Table 4.2:** Dataset statistics for each special scenario dataset.

scenarios.

As shown in Figure, performance using the base model trained on the initial dataset drops significantly when tested on the special scenario validation set. The comparison was made considering the detections of Person entities, given the negligible cardinality of Crowd entities, which would have led to insignificant results.

While part of the decline is due to the dataset's quality and size, the decrease is most notable in conditions that significantly alter image brightness and colors. Notably, the Snow scenario, despite not matching the training domain, performed best among the special scenarios, likely because brightness and colors remained largely unchanged. The Rain scenario also showed minimal issues. Interestingly, the Night dataset performed better than the Night-Snow dataset, despite the latter having higher brightness, probably due to differences in color distribution since in the Night-Snow scenario, the camera rendering focused the brightness and color primarily on the entirely white track. In contrast, for the nighttime forest in the Night scenario the color and brightness were concentrated on the track's sides. Regardless, their poor performance is significantly impacted by the limited number of instances even for the *Person* class, as depicted in Table 4.2.

We conducted various tests to address this problem. Initially, we retrained with datasets specific to each domain, which yielded better results as described in Table 4.3, but was limited to the specific domain.

We then aimed to adapt our initial version to generalize across all special cases (even the not yet considered scenarios), focusing on augmentation parameters. Specifically, we modified HSV channel parameters, increasing *sgain* to 0.5, *vgain* to 0.8, and *hgain* to 0.1 to enhance color and brightness variability. This, shown in Figure 4.9, did not yield significant improvements.

As observed, the only improvement was in the Night dataset, but it remains far

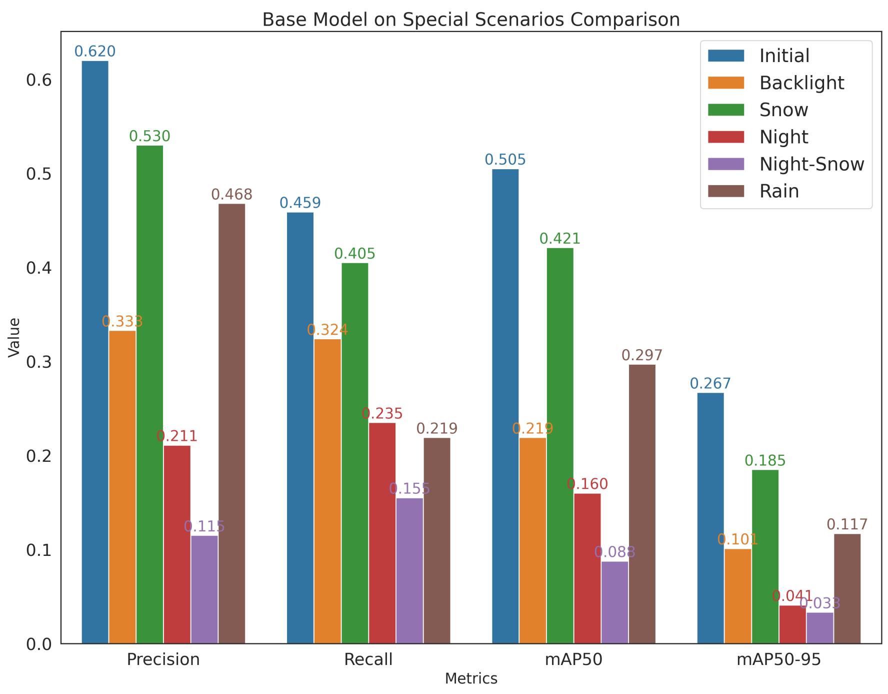


**Figure 4.8:** Comparison of the performance of the base model in the special scenarios. We highlighted also the initial dataset behavior to have a complete overview.

from an acceptable result and is not comparable to the performance achieved with the specific scenario dataset. Additionally, it degraded the performance of datasets like Rain and Snow, which performed better with the base version of the model.

Next, we tested more drastic augmentation, setting all three values to 1 for maximum variability of the network to explore limitless modifications on the three different channels, but again, the results were unsatisfactory as shown in Figure 4.10.

Realizing that nearly unlimited variability was ineffective, we mathematically normalized HSV channels to match the mean and standard deviation of the target domain images. While theoretically ideal for generalization, this method also failed to produce satisfactory results, worsening the performance by obtaining less than half of the basic model results.

Ultimately, we opted for a solution that, while not generalizing to unforeseen domains, ensured optimal performance in the considered special cases. We added domain-specific images to the training dataset alongside the original images. This

| Metric | Dataset | *Backlight* | *Snow* | *Night* | *Night-snow* | *Rain* |
|---|---|---|---|---|---|---|
| precision | Initial | 0.333 | 0.530 | 0.211 | 0.115 | 0.468 |
| | Specific | 0.458 | 0.703 | 0.621 | 0.937 | 0.618 |
| | Delta | +0.125 | +0.173 | +0.410 | +0.822 | +0.150 |
| recall | Initial | 0.324 | 0.405 | 0.235 | 0.155 | 0.219 |
| | Specific | 0.667 | 0.623 | 0.392 | 0.241 | 0.523 |
| | Delta | +0.343 | +0.218 | +0.157 | +0.086 | +0.226 |
| mAP50 | Initial | 0.219 | 0.421 | 0.160 | 0.088 | 0.297 |
| | Specific | 0.619 | 0.681 | 0.467 | 0.450 | 0.581 |
| | Delta | +0.400 | +0.260 | +0.307 | +0.362 | +0.284 |
| mAP50-95 | Initial | 0.101 | 0.185 | 0.041 | 0.033 | 0.117 |
| | Specific | 0.271 | 0.323 | 0.205 | 0.195 | 0.244 |
| | Delta | +0.170 | +0.138 | +0.164 | +0.162 | +0.127 |

**Table 4.3:** Performance of each dataset tested using the base model with initial images or the specific scenario images as the training dataset.

approach maintained effective performance in the new domains without compromising the base dataset's performance under normal conditions, as depicted in Figure 4.11. Our final solution, which combines all domains with the original dataset, performs optimally across all tested datasets, including the initial one.

## 4.5 Image quality impact

The videos we obtained were captured for live streaming of the race, resulting in compression to half the original quality. This led us to consider that higher image quality could potentially enhance our network's performance. In practical applications, the hardware setup that processes the images using our network is mounted directly on the vehicle, allowing access to the original video file without the need for streaming compression.

Lacking access to the original file quality, even with upscaling tools, we initially turned to the literature. Papers such as [46] and [47] demonstrate that increasing image quality generally improves performance, but the benefits diminish beyond 720p. However, these papers specify that this trend does not apply to small objects, which constitute the majority of our detections.

To understand the impact of image quality on our architecture, we estimated the effects by downscaling our dataset and then predicting the impact of higher quality. We conducted downscaling to 50%, 25%, 12.5%, and 6.25% quality, trained on these images and used regression to estimate the effect of image quality enhancement. The results, shown in Figure 4.12, indicate that increasing image quality does not

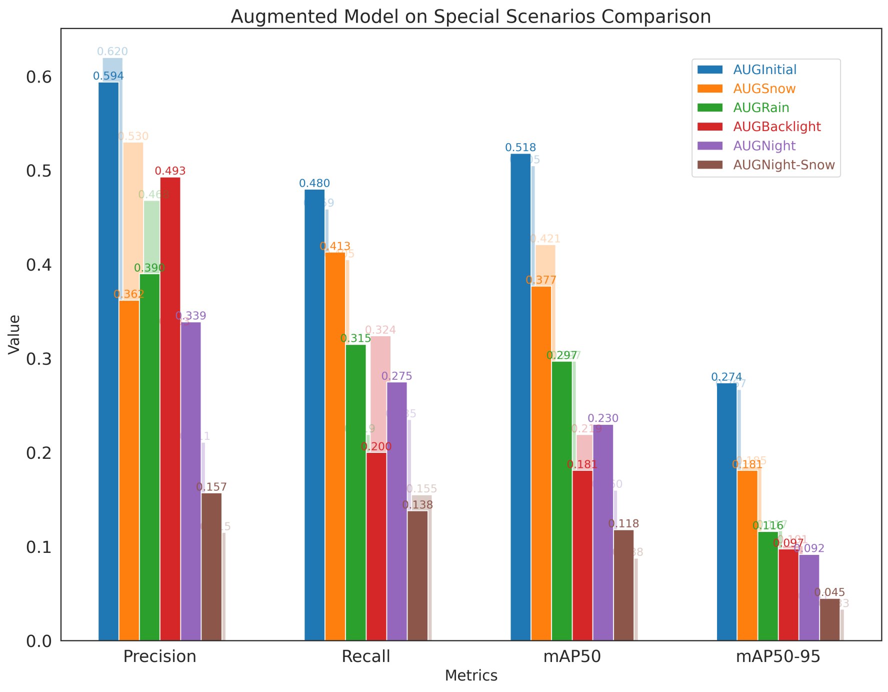


**Figure 4.9:** Comparison of the model's performance with stronger augmentation in special scenarios. The model with basic augmentation is slightly visible in the background, allowing for a comparative analysis.

linearly enhance performance.

However, improvements in detecting small objects are significant, as we considered only the *Person* detections. While image quality increase eventually saturates for object detection tasks, this effect is not seen for small objects. Since the pixel count for these objects is low, we believe that higher definition significantly impacts their detection, even with high-quality images.

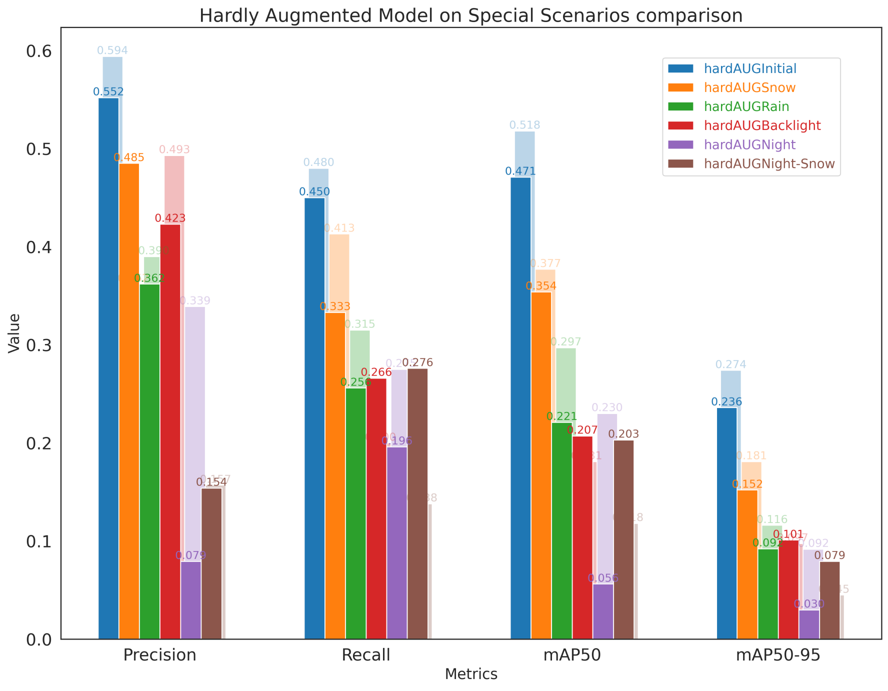


**Figure 4.10:** Comparison of the model's performance with hard augmentation in special scenarios. The model with less strong augmentation is slightly visible in the background, allowing for comparative analysis.

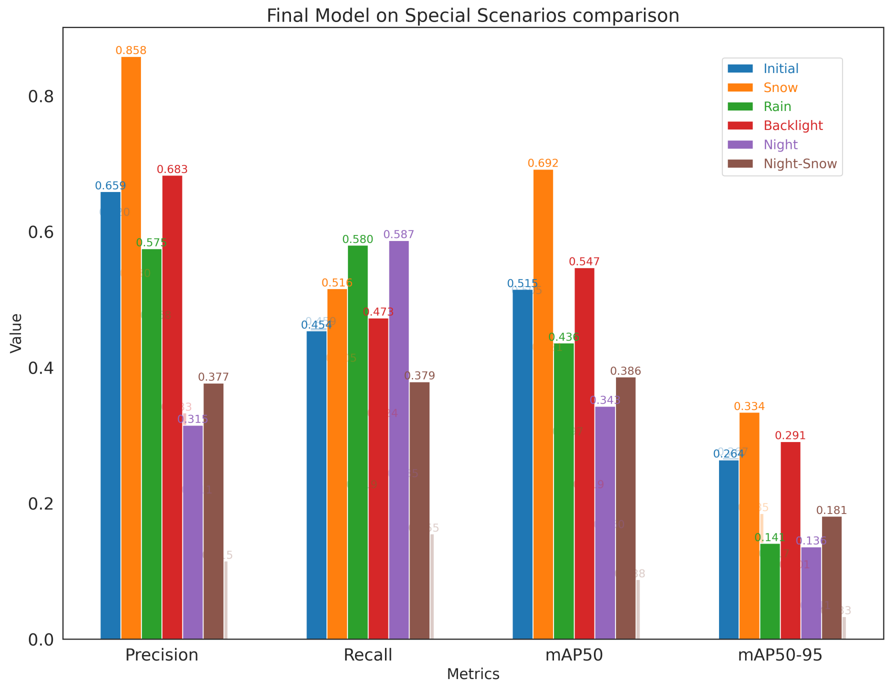


**Figure 4.11:** Comparison of the final model version in special scenarios. The model with basic augmentation is slightly visible in the background, allowing for a comparative analysis.

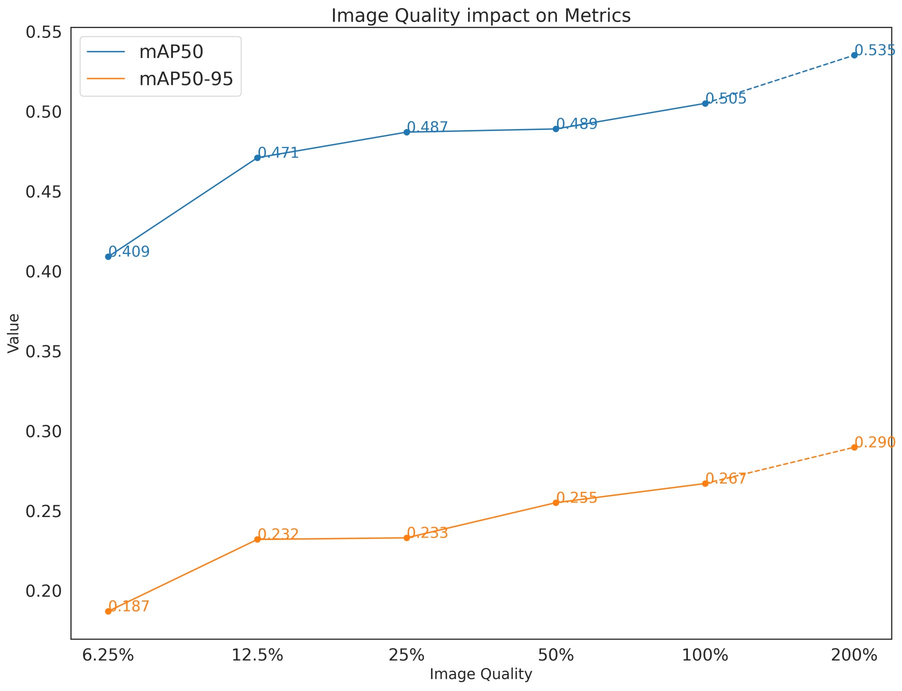


**Figure 4.12:** Mean Average Precisions calculated using reduced quality of images. The last point is calculated using Linear Regression considering the LogScale that we adopted to do the downscaling.

# Chapter 5

# Conclusions

Our work focused on developing a robust alert system for safety in the World Rally Championship (WRC) by identifying humans during rally races. Marelli S.p.A. aimed to achieve real-time detection to enable marshals overseeing the races to quickly intervene in dangerous situations on the circuit. Marelli's standards were met in terms of both accuracy and performance, with precise human detection and image processing that outputs the detection results faster than the camera's capture speed.

We utilized YOLOv8n, which met our qualitative and computational objectives. This involved an in-depth study of the network's capabilities and parameter calibration to enhance performance. A custom-labeled dataset reflecting our domain was crucial, ensuring human detection under adverse conditions, such as distant individuals or crowds, complicated by high rally speeds.

We also considered less common but significant scenarios, like races with unusual weather or lighting. These conditions initially degraded our network's performance. We explored general solutions using image augmentation and mathematical adjustments to color channels to adapt to various conditions independently. However, these general solutions worsened performance metrics across all datasets. Instead, combining data from all domains, including the initial dataset, allowed our network to perform efficiently across all the considered scenarios while maintaining excellent performance in normal conditions.

Additionally, we analyzed the impact of image quality on our results, demonstrating that access to uncompressed original videos could further improve our architecture. Future developments should focus on two areas: refining our architecture as a foundation for the complete system and developing the system itself. Improvements could include new technologies for real-time human detection, leveraging newer YOLO versions, creating a unique metric to recognize both *Crowd* and dense *Person* detections as equally exact, or developing data augmentation methods to generalize across all domains, even the ones that we did not consider.

The next steps involve developing the remaining technology. The ultimate goal is a system that not only detects humans but also distinguishes between those in danger and those safe. This requires domain knowledge to teach the network to recognize trajectories, speeds, and distances. A first step could be track detection through segmentation and monocular distance estimation added to our human detection, maintaining the system's real-time capabilities.

# Appendix A

# Evaluation metrics after 400 epochs

In this section, we present the evaluation metrics results using the base version of the model. As noted earlier, most metrics do not show improvement; in fact, there is a slight decline. This observation, along with the considerations discussed in Chapter 4, confirms the soundness of the decision to limit training to 100 epochs, as the best results were obtained before reaching this threshold. This approach was adopted based on these results, which are crucial in justifying our decision. The slightly constant precision, without any increase, results in no real improvement in detection accuracy. Overfitting is likely indicated by the observed decline in recall, which produces fewer detections compared to the base version. This is probably because the network, having overfitted, has features that are too specific to the training dataset, preventing it from generalizing well to different environments.

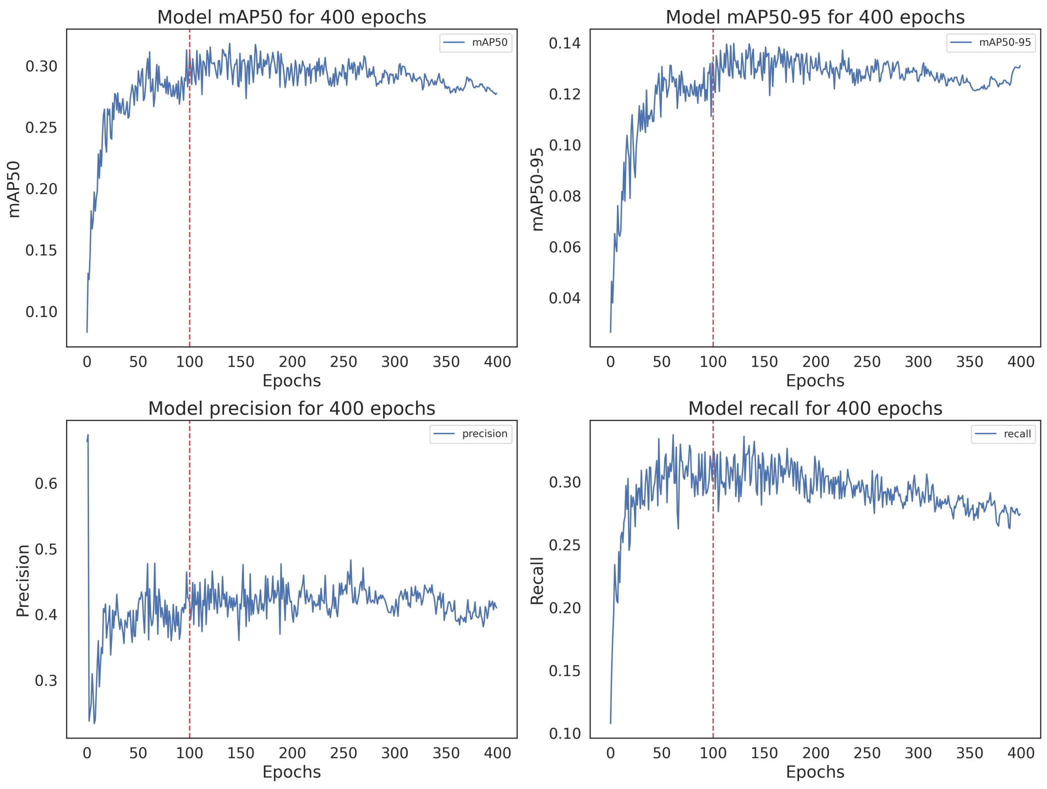


**Figure A.1:** Evaluation metrics computed for 400 epochs.

# Bibliography

[1] P. Viola and M. Jones. «Rapid object detection using a boosted cascade of simple features». In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* Vol. 1. 2001, pp. I–I. DOI: `10.1109/CVPR.2001.990517` (cit. on p. 5).

[2] N. Dalal and B. Triggs. «Histograms of oriented gradients for human detection». In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).* Vol. 1. 2005, 886–893 vol. 1. DOI: `10.1109/CVPR.2005.177` (cit. on p. 6).

[3] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. «Support vector machines». In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28. DOI: `10.1109/5254.708428` (cit. on p. 6).

[4] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks.* 2015. arXiv: `1511.08458` (cit. on p. 6).

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation.* 2014. arXiv: `1311.2524` (cit. on p. 6).

[6] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge.* 2015. arXiv: `1409.0575` (cit. on p. 6).

[7] Ross Girshick. *Fast R-CNN.* 2015. arXiv: `1504.08083` (cit. on p. 6).

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.* URL: `http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html` (cit. on pp. 7, 10).

[9] 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` (cit. on pp. 7, 9).

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. «SSD: Single Shot MultiBox Detector». In: *Lecture Notes in Computer Science*. 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. 7–9).

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 (cit. on pp. 7–9, 11, 12, 14).

[12] G. Jocher, A. Chaurasia, and J. Qiu. *YOLO by Ultralytics*. https://github.com/ultralytics/ultralytics. Accessed: October 30, 2023. 2023 (cit. on pp. 8, 18, 19).

[13] S. Srivastava, A.V. Divekar, C. Anilkumar, et al. «Comparative analysis of deep learning image detection algorithms». In: *Journal of Big Data* 8.66 (2021). DOI: 10.1186/s40537-021-00434-w. URL: https://doi.org/10.1186/s40537-021-00434-w (cit. on pp. 8, 9).

[14] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 (cit. on pp. 8, 9, 12, 14, 15).

[15] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 (cit. on pp. 8, 15).

[16] L. Tan, T. Huangfu, L. Wu, et al. «Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification». In: *BMC Medical Informatics and Decision Making* 21.324 (2021). DOI: 10.1186/s12911-021-01691-8. URL: https://doi.org/10.1186/s12911-021-01691-8 (cit. on p. 9).

[17] G. Jocher. *YOLOv5 by Ultralytics*. https://github.com/ultralytics/yolov5. Accessed: March, 2024. 2020 (cit. on pp. 9, 16, 18).

[18] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. «Rectifier Nonlinearities Improve Neural Network Acoustic Models». In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. 3. 2013. URL: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf (cit. on p. 10).

[19] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. «The PASCAL Visual Object Classes (VOC) Challenge». In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338 (cit. on pp. 10, 11, 19).

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848` (cit. on pp. 10, 19).

[21] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: `1612.08242` (cit. on pp. 12, 13).

[22] Ivan Krasin et al. *OpenImages: A public dataset for large-scale multi-label and multi-class image classification*. 2017. URL: `https://github.com/openimages` (cit. on p. 14).

[23] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. 2019. arXiv: `1911.08287` (cit. on pp. 15, 16, 18).

[24] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. arXiv: `1911.11929` (cit. on pp. 15, 17).

[25] Diganta Misra. *Mish: A Self Regularized Non-Monotonic Activation Function*. 2020. arXiv: `1908.08681` (cit. on p. 16).

[26] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. *Path Aggregation Network for Instance Segmentation*. 2018. arXiv: `1803.01534` (cit. on pp. 16, 17).

[27] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. *CBAM: Convolutional Block Attention Module*. 2018. arXiv: `1807.06521` (cit. on p. 16).

[28] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: `1606.08415` (cit. on pp. 16, 18).

[29] Chuyi Li et al. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. arXiv: `2209.02976` (cit. on p. 17).

[30] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. *RepVGG: Making VGG-style ConvNets Great Again*. 2021. arXiv: `2101.03697` (cit. on pp. 17, 18).

[31] Guanglu Song, Yu Liu, and Xiaogang Wang. *Revisiting the Sibling Head in Object Detector*. 2020. arXiv: `2003.07540` (cit. on p. 17).

[32] Yue Wu, Yinpeng Chen, Lu Yuan, Zicheng Liu, Lijuan Wang, Hongzhi Li, and Yun Fu. *Rethinking Classification and Localization for Object Detection*. 2020. arXiv: `1904.06493` (cit. on p. 17).

[33] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R. Scott, and Weilin Huang. *TOOD: Task-aligned One-stage Object Detection*. 2021. arXiv: `2108.07755` (cit. on p. 17).

[34] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. *OTA: Optimal Transport Assignment for Object Detection.* 2021. arXiv: `2103.14259` (cit. on p. 17).

[35] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi Huang, Jungong Han, and Guiguang Ding. *Re-parameterizing Your Optimizers rather than Architectures.* 2023. arXiv: `2205.15242` (cit. on p. 17).

[36] Changyong Shu, Yifan Liu, Jianfei Gao, Zheng Yan, and Chunhua Shen. *Channel-wise Knowledge Distillation for Dense Prediction.* 2021. arXiv: `2011.13256` (cit. on p. 17).

[37] 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` (cit. on p. 17).

[38] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. *Designing Network Design Strategies Through Gradient Path Analysis.* 2022. arXiv: `2211.04800` (cit. on p. 17).

[39] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. *Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection.* 2020. arXiv: `2006.04388` (cit. on p. 18).

[40] Ultralytics. *YOLOv8—Ultralytics YOLOv8 Documentation.* Accessed: 2024. 2023. URL: `https://docs.ultralytics.com/models/yolov8/` (cit. on p. 19).

[41] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context.* 2015. arXiv: `1405.0312` (cit. on p. 19).

[42] Andreas Geiger, Philip Lenz, and Raquel Urtasun. «Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite». In: *Conference on Computer Vision and Pattern Recognition (CVPR).* 2012 (cit. on p. 20).

[43] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. *CrowdHuman: A Benchmark for Detecting Human in a Crowd.* 2018. arXiv: `1805.00123` (cit. on p. 20).

[44] Xuehui Yu, Yuqi Gong, Nan Jiang, Qixiang Ye, and Zhenjun Han. *Scale Match for Tiny Person Detection.* 2019. arXiv: `1912.10664` (cit. on p. 20).

[45] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D. Cubuk, Quoc V. Le, and Barret Zoph. *Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation.* 2021. arXiv: `2012.07177` (cit. on p. 28).

[46] Miloud Aqqa, Pranav Mantini, and Shishir Shah. «Understanding How Video Quality Affects Object Detection Algorithms». In: Jan. 2019, pp. 96–104. DOI: 10.5220/0007401600960104 (cit. on p. 39).

[47] Yu Hao, Haoyang Pei, Yixuan Lyu, Zhongzheng Yuan, John-Ross Rizzo, Yao Wang, and Yi Fang. *Understanding the Impact of Image Quality and Distance of Objects to Object Detection Performance*. 2022. arXiv: 2209.08237 [cs.CV]. URL: https://arxiv.org/abs/2209.08237 (cit. on p. 39).