NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF
ELECTRICAL AND COMPUTER ENGINEERING

RICARDO MIGUEL PIRES DE SOUSA
BSc in Electrical and Computer Engineering

DETECTION AND TRACKING OF RED TRAFFIC LIGHT INFRINGEMENTS USING DNNS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING
NOVA University Lisbon
September, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF
ELECTRICAL AND COMPUTER ENGINEERING

DETECTION AND TRACKING OF RED TRAFFIC LIGHT INFRINGEMENTS USING DNNS

RICARDO MIGUEL PIRES DE SOUSA

BSc in Electrical and Computer Engineering

Adviser: André Damas Mora
Assistant Professor, NOVA University Lisbon

Examination Committee:

Chair: João Murta Pina
Assistant Professor, FCT-NOVA

Rapporteur: José Manuel Ribeiro da Fonseca
Associate Professor with Habilitation, FCT-NOVA

Adviser: André Damas Mora
Assistant Professor, FCT-NOVA

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2023

Detection and Tracking of Red Traffic Light Infringements Using DNNs

Copyright © Ricardo Miguel Pires de Sousa, NOVA School of Science and Technology, NOVA University Lisbon.

The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created with Microsoft Word text processor and the NOVAthesis Word template [1].

To my father.

ACKNOWLEDGMENTS

My first thanks go to Professor André Mora for, above all, giving me the opportunity to do this thesis, but also for all the attention he has given me over the months, always finding time in his schedule to help me and meet with me, and for the friendliness with which he has always treated me.

I would also like to thank the whole CA3 group and especially Shuai, who shared the same room with me at Uninova, being my greatest companion during the times I was working on this thesis, always offering me ice creams on hot days, and for all the good humor with which he always addressed me.

I would like to thank the university for the years I've spent at it, which, although they've been a great struggle and challenge, have above all been a time of personal development.

I would like to thank my group of friends from university for all the good times we had together and for sharing the same struggles, which made these years a little easier. I would also like to thank my circle of friends outside of university for giving me many moments to unwind from my studies.

I would like to express my greatest gratitude to my parents for all the support they have always given me. To my mum, for all the education and values she has always instilled in me, and to my dad, for all his generosity and for fighting to give me the opportunity to complete this course.

Finally, the most special thanks go to Sofia, for all the love she gave me and for always believing in me so much and encouraging me to complete this stage.

ABSTRACT

The efficient control of traffic signals, with red traffic lights operating as a primary mechanism to prevent accidents at intersections and pedestrian crossings, is one of the key components in ensuring road safety. Red traffic light infringements continue to be a chronic concern, resulting in numerous accidents, injuries, and fatalities each year around the world despite the vast traffic management measures in place.

Traffic management systems are beginning to get exhausted by increased traffic congestion and urbanization growth. Moreover, it is almost impossible for police forces to be present on all roads and crossings. It then becomes an urgent need for creative, effective solutions to improve safety and lower traffic infringements.

A methodology for detecting and tracking red traffic light infringements based on deep neural networks is proposed, using the YOLO algorithm to detect and classify vehicles and traffic lights, as well as the Deep SORT algorithm, capable of tracking the movement of vehicles over time.

The results obtained show a very promising system, capable of detecting red traffic light infringements with plenty of room for progression and improvement, making it possible to dream of implementing it in the real world in the near future and helping to increase road safety.

Keywords: Computer Vision, Red Traffic Light Infringements, Object Detection, Object Tracking, Deep Neural Networks, Traffic Light Recognition, Vehicle Detection, Vehicle Tracking, YOLO, Deep SORT

RESUMO

O controlo eficiente dos sinais de trânsito, com os semáforos vermelhos a funcionarem como mecanismo principal para evitar acidentes em cruzamentos e passagens de peões, é um dos principais componentes para garantir a segurança rodoviária. As infrações aos sinais vermelhos continuam a ser uma preocupação crónica, resultando em numerosos acidentes, ferimentos e mortes todos os anos em todo o mundo, apesar das vastas medidas de gestão do tráfego em vigor.

Os sistemas de gestão do tráfego estão a começar a ficar esgotados devido ao aumento do congestionamento do tráfego e ao crescimento da urbanização. Além disso, é quase impossível para as forças policiais estarem presentes em todas as estradas e cruzamentos. Torna-se então urgente encontrar soluções criativas e eficazes para melhorar a segurança e reduzir as infrações de trânsito.

É proposta uma metodologia de deteção e seguimento de infrações aos semáforos vermelhos baseada em redes neuronais profundas, utilizando o algoritmo YOLO para detetar e classificar veículos e semáforos, bem como o algoritmo Deep SORT, capaz de seguir o movimento dos veículos ao longo do tempo.

Os resultados obtidos mostram um sistema muito promissor, capaz de detetar infrações aos semáforos vermelhos com muito espaço para progressão e melhoria, tornando possível sonhar com a sua implementação no mundo real num futuro próximo e ajudar a aumentar a segurança rodoviária.

Palavras chave: Visão Computacional, Infrações aos Semáforos Vermelhos, Deteção de Objetos, Seguimento de Objetos, Redes Neuronais Profundas, Reconhecimento de Semáforos, Deteção de Veículos, Seguimento de Veículos, YOLO, Deep SORT

CONTENTS

LIST OF FIGURES ..... **XIX**

LIST OF TABLES ..... **XXI**

ACRONYMS ..... **XXII**

1 INTRODUCTION **1**

1.1 Motivation 1

1.2 Problems and Objectives 2

1.3 Approach and Contributions 3

1.4 Document Organization 4

2 STATE OF THE ART **5**

2.1 Differences between Machine Learning and Deep Learning fields 6

2.2 Vehicle Detection Based On Machine Learning 6

2.2.1 Feature Extraction Methods 6

2.2.2 Classifiers 12

2.3 Vehicle Detection Based On Deep Learning 14

2.3.1 Faster R-CNN 15

2.3.2 Single Shot Detector 16

2.3.3 You Only Look Once 17

2.4 Tracking algorithms 19

2.4.1 Kalman Filter 20

2.4.2 Deep SORT 20

2.5 Traffic Light Recognition 22

2.5.1 Challenges 22
2.5.2 Color-based methods 23
2.5.3 Feature-based methods 24
2.5.4 Deep Learning-based Methods 27
3 METHODOLOGY 29
3.1 System Overview 29
3.2 Setup of the Environment 31
3.2.1 Hardware Units 31
3.2.2 Tools and Frameworks 33
3.2.3 Libraries 34
3.3 YOLO Detector 35
3.3.1 Dataset 35
3.3.2 Model Training 38
3.4 Deep SORT Tracker 42
3.5 Red Traffic Light Infringement Control 43
3.5.1 Selection of Regions of Interest 43
3.5.2 Support Functions 44
3.5.3 Infringement Detection Method 47
4 RESULTS AND DISCUSSIONS 49
4.1 Detector's Training Performance 49
4.1.1 Evaluation Metrics 49
4.1.2 Results Obtained 53
4.2 System Performance On Videos 56
4.2.1 Evaluation Criteria 59
4.2.2 Detector's Performance Consistency 61
4.2.3 Tracker's Performance Consistency and Quality 62
4.2.4 Red Traffic Light Infringements Detection Performance 63
4.3 Processing Time 64
5 CONCLUSIONS AND FUTURE WORK 67

5.1 Conclusions 67
5.2 Future Work 68
REFERENCES 69

LIST OF FIGURES

Figure 2.1: Example of the Haar-like features. 7
Figure 2.2: Faster R-CNN network. 15
Figure 2.3: SSD architecture. 17
Figure 2.4: YOLOv1 working approach. 18
Figure 2.5: YOLOv1 architecture. 18
Figure 2.6: Deep SORT working diagram. 21
Figure 2.8: Representation of top-down and bottom-up methods. 26
Figure 2.9: Example of the search for the circle object when its radius is unknown. 27
Figure 3.1: Project's workflow. 29
Figure 3.2: System overview. 30
Figure 3.3: NVIDIA Jetson TX2 Developer Kit. 32
Figure 3.4: Examples of images from the dataset 36
Figure 3.5: Examples of images with annotated objects. 38
Figure 3.6: Class balance. 38
Figure 3.7: Example of a text file with vehicles' label information. 40
Figure 3.8: Example of selected ROIs in a given scene. 44
Figure 3.9: Example of a vehicle's BB corner inside the ROI. 45
Figure 3.10: Example of a case where the vehicle's BB intersects the ROI without having any corner inside it. 46
Figure 3.11: Flowchart of the infringement detection method. 48
Figure 4.1: Illustration of the IoU calculation. 50
Figure 4.2: mAP@0.50 and average loss chart of the YOLO detector training. 53
Figure 4.3: Scenario of each video used to test the system. 57
Figure 4.4: Examples of traffic lights where the light is not clearly discernible due to the brightness or position of the light. 58
Figure 4.5: Examples of traffic lights together on the same pole, where the detector gets confused. 58
Figure 4.6: Examples of yellow traffic lights being mistaken for red traffic lights. 59

LIST OF TABLES

Table 1: MSI computer specifications. 31
Table 2: Jetson TX2 module specifications. 32
Table 3: Software features installed on the Jetson TX2 Developer Kit. 33
Table 4: Tools and frameworks used to develop the project. 33
Table 5: Libraries used to implement the project. 34
Table 6: Values of precision, recall, F1-score, and total of TP, FP, and FN (related to the validation set). 54
Table 7: Values of TP, FP, and AP obtained for each class (related to the validation set). 54
Table 8: Values of precision, recall, F1-score, and total of TP, FP, and FN (related to the testing set). 56
Table 9: Values of TP, FP, and AP obtained for each class (related to the testing set). 56
Table 10: Ground-truth total of objects, duration, number of TP, FP, and FN registered in each video. 57
Table 11: Detection consistency obtained in each video. 61
Table 12: Tracking consistency obtained for each video. 62
Table 13: Tracking quality obtained for each video. 62
Table 14: Number of infringements, TP, FP, and FN obtained for each video. 63
Table 15: Detector's frame rate obtained in each video at three different resolutions using the Jetson TX2 GPU. 64
Table 16: Detector's frame rate obtained in each video at three different resolutions using the Tesla T4 GPU. 65

ACRONYMS

AdaBoost
Adaptive Boosting 12, 14
ADAS
Advanced Driver-Assistance System 22
AI
Artificial Intelligence 31, 33, 42
AP
Average Precision 51, 52, 54, 56
API
Application Programming Interface 33, 34
CCTV
Closed-circuit Television 12
CIoU
Complete Intersection over Union 53
CNN
Convolutional Neural Network 14, 17
CPU
Central Processing Unit 31, 65
DoG
Difference of Gaussians 9
FCN
Fully Convolutional Neural Network 15
FN
False Negatives 50, 54, 55, 56, 57, 58, 63
FP
False Positives 50, 54, 55, 56, 57, 58, 63, 64
FPS
Frames Per Second 64, 65, 67
GPU
Graphics Processing Unit 28, 31, 33, 41, 53, 64, 65, 67, 68
HOG
Histogram of Oriented Gradients 6, 11, 12, 69
HSV
Hue, Saturation, Value 23
ID
Identifier 4, 22, 43, 47, 48, 59, 60, 63, 64, 67
IoU
Intersection over Union 49, 50, 52, 54, 55, 56

ISM
Implicit Shape Model 10
mAP
Mean Average Precision 52, 53, 54, 55, 56, 67
R-CNN
Region-based Convolutional Neural Network 5, 15, 16, 18, 19, 28, 70
RGB
Red Green Blue 23, 24
ROI
Region of Interest 11, 20, 43, 44, 45, 46, 47, 48, 64
ROIs
Regions of Interest 15, 16, 43, 44, 45, 48
RPN
Region Proposal Network 15
SIFT
Scale-Invariant Feature Transform 6, 8, 9, 10, 13, 69
SORT
Simple Online and Realtime Tracking ..... xi, xiii, 1, 3, 4, 19, 20, 21, 22, 31, 34, 42, 43, 71
SSD
Single Shot Detector 5, 16, 17, 18, 70
SURF
Speeded-Up Robust Features 6, 10, 11, 69
SVM
Support Vector Machine 10, 12, 13, 69
TP
True Positives 50, 54, 55, 56, 57, 60, 62, 63
UAV
Unmanned Aerial Vehicle 9
YOLO
You Only Look Once ….. xi, xiii, 1, 3, 5, 17, 18, 19, 21, 22, 28, 31, 33, 34, 35, 39, 40, 42, 53, 67, 70, 71
YOLOv1
You Only Look Once version 1 ..... 17, 18
YOLOv3
You Only Look Once version 3 ..... 19, 70
YOLOv4
You Only Look Once version 4 ..... 3, 4, 35, 39, 40, 42, 52, 55
YOLOv8
You Only Look Once version 8 17

1

INTRODUCTION

1.1 Motivation

The safe and effective circulation of vehicles on our roadways is ensured by traffic management, a crucial part of urban infrastructure. The efficient control of traffic signals, with red traffic lights operating as a primary mechanism to prevent accidents at intersections and pedestrian crossings, is one of the key components in ensuring road safety. Red traffic light infringements continue to be a chronic concern, resulting in numerous accidents, injuries, and fatalities each year around the world despite the vast traffic management measures in place.

The importance of managing red traffic light infringements is made even more clear in light of the growing urbanization and growth of road networks. Traffic congestion in urban areas is still worsening, putting more strain on traffic management systems and highlighting the need for creative, effective solutions to improve safety and lower traffic infringements.

The necessity to create a comprehensive system for the automated detection and tracking of red traffic light infringements using cutting-edge deep neural networks serves as the driving force behind this research project. Deep learning approaches have enabled significant improvements in a variety of computer vision tasks, from object detection to object tracking. Models like YOLO (You Only Look Once) in particular have proven to be exceptionally accurate and effective in object identification and classification tasks, making them suitable for real-time applications like traffic monitoring.

Furthermore, by combining object detection with tracking algorithms like Deep SORT, we are able to not only spot red light infringements but also keep an eye on how the behavior of moving vehicles changes before, during, and after such instances. The dynamics of traffic infringements are better understood thanks to this comprehensive approach, which also helps law enforcement organizations enforce traffic laws and increase road safety.

1.2 Problems and Objectives

The main objective of this work is to create, implement, and assess a deep neural network-based system that can detect vehicles and track their movements in order to capture when they run red traffic lights. In addition, detecting traffic lights and recognizing their state (green, yellow, or red) is also one of the fundamental objectives, because in order to identify a red light infringement, it is not enough to detect only vehicles, but also to identify when they can or cannot move forward.

It is also essential for a system of this type to operate in real-time, as many vehicles travel at high speeds and the state of some traffic lights also changes quickly, being only a matter of seconds, such as the change from yellow to red lights. And a delay of just a few seconds is enough to lose track of an infringement or even an improper record of one.

By accomplishing these goals and by implementing the system in real life, it is hoped that it will help to lower the number of accidents and fatalities brought on by running red lights in the near future. Additionally, this study is consistent with a larger aim of creating intelligent transportation systems that improve traffic flow, lessen congestion, and pave the path for more efficient and sustainable urban mobility.

Of course, there are many challenges to these tasks and there are often many variables to deal with, even unexpected ones. There are numerous possible obstacles on real-world roads. Other objects, such as vehicles, road infrastructures, trees, and pedestrians can completely or partially block traffic lights and vehicles. It is challenging for detection and tracking algorithms to maintain precise and uninterrupted object tracking due to this occlusion, which breaks continuous vision. Depending on how close vehicles and traffic lights are to the camera, they also change in scale and perspective. For accurate identification and tracking over various distances and perspectives, managing these scale and perspective variations is essential. The visibility and identification of objects are substantially impacted by variations in lighting conditions and adverse weather, which is also a big challenge.

Congestion frequently results in vehicles being tightly packed, which creates complex traffic situations, particularly in urban locations or during high-traffic hours. The tracking of individual objects and the recognition of traffic lights may be made more difficult by this density growth. Not all motorists strictly obey the laws of the road. Some of them drive erratically or exhibit unusual habits, like abrupt lane changes or illegal U-turns. These variations can put tracking algorithms to the test.

For the development and assessment of detection and tracking algorithms and to overcome some of these issues, having access to a broad and representative dataset for training and testing is essential. The gathering of such data might be a laborious process, and it takes

a lot of work and subject knowledge to produce correct labels and annotations for training data. Inaccurate labeling or insufficient annotation can result in an inefficient model performance.

Deep neural network training, especially for YOLO, calls for a lot of computational resources. It can be difficult to gain access to high-performance GPUs or cloud-based resources for model training.

Understanding these difficulties is essential to succeed in enhancing traffic management and road safety while striving to develop precise and reliable detection and tracking technologies.

1.3 Approach and Contributions

The approach's core component is the use of the cutting-edge YOLO (You Only Look Once) deep neural network for object detection and classification. The algorithm was specifically customized and trained to excel at recognizing the different states of traffic lights and different types of vehicles.

To train the model, various images of traffic lights and vehicles were collected and integrated into a dataset. All the images were annotated in order to highlight all the objects of interest so that the algorithm could learn their features. This dataset was divided into three parts: training set, validation set, and testing set. The pre-trained weights from YOLOv4-tiny were used to start training the algorithm, so as not to have to start from scratch and thus become more accurate more quickly.

The file with the final trained weights served as input to the Deep SORT tracker, which is also a deep neural network capable of tracking objects by learning their appearance. To integrate the detector with the tracker, the weights file was converted into a Tensorflow model. Some changes have also been made to the tracker's code to track only vehicles and continue to show the detections of the traffic lights, without tracking them.

Finally, code was created to mechanize the detection of red traffic light infringements. Some functions have been developed to identify whether the bounding boxes of the vehicles intersect certain areas of the image. And, depending on the state of the traffic light, the system is able to detect whether a vehicle is committing an infringement or not.

The main contributions to the field of this work are the following:

- Creation of a class-balanced dataset with 1398 annotated images of different vehicles and traffic lights.

- A YOLOv4-tiny detector capable of detecting vehicles and recognizing the state of the traffic lights in real-time.
- Deep SORT Tracker capable of tracking the movement of the detected vehicles over time, assigning them an ID.
- Method developed in Python, which using the information provided by the detector and the tracker, is able to detect in videos red traffic light infringements committed by vehicles.

1.4 Document Organization

In addition to this chapter, this dissertation has four more chapters:

State of the Art. This chapter presents the literature review carried out to study the problem in question. Techniques and algorithms used for object detection and tracking based on machine learning and deep learning are presented. Some examples of works developed by different authors for each technique are also given.

Methodology. The purpose of this chapter is to explain to the reader all the procedures necessary to carry out the work and how they were performed. It begins by presenting an overview of the system created, then all the hardware and software used to develop the project is presented and each of the system's integral parts is explained.

Results and Discussions. This chapter presents the results obtained from all the work performed and explained in the previous chapter. As they are presented, they are also analyzed and some clarifications are made. Firstly, the accuracy metrics used to evaluate the object detector are described, as well as how they can be calculated. Next, the detector's training results are presented. Then, it is moved on to the results relating to the detector, tracker, and infringement detection performance in testing videos.

Conclusions and Future Work. The final chapter contains all the conclusions and future work that could be done to improve the project. The conclusions section summarizes what has been achieved through the work developed, as well as aspects for improvement or that could have been done in other ways. The future work section presents some ideas that could be implemented in the future in order to evolve and make the project more robust.

2

STATE OF THE ART

This chapter presents the literature review on systems capable of detecting, tracking, and classifying vehicles and traffic lights based on computer vision, focusing on the machine learning and deep learning techniques used to serve different challenges while implementing this kind of systems.

This literature review begins by briefly introducing the principal existing differences between machine learning and deep learning fields. Then, a section dedicated to several different machine learning algorithms applied to vehicle detection is presented. These algorithms are divided into two categories: feature extraction and classifier training. This is followed by another section dedicated to the main deep learning algorithms used to detect and classify vehicles, such as Faster R-CNN, SSD, and YOLO. All these algorithms are explained and well supported with existing examples of vehicle detection systems in the literature. After that, is presented a section for the vehicle tracking algorithms.

The last section is dedicated to traffic light detection and recognition. Firstly, the main challenges associated with the problem are identified, then various techniques that seek to solve the problem are presented, supported by examples found in literature. These techniques are divided into three topics: color-based, feature-based and deep learning-based techniques. The objective of this chapter is to provide a good understanding of the existing study on the different approaches used to implement vehicle and traffic light detection and tracking systems.

2.1 Differences between Machine Learning and Deep Learning fields

Both machine learning and deep learning belong to artificial intelligence by dealing with algorithms capable of learning from data, but there are some differences between these two fields.

One of the main differences is the model's architecture. Deep learning uses neural networks and has a more complex architecture with hidden layers and non-linear associations between the inputs and the outputs. Machine learning uses mathematical models, such as linear regression and decision trees. Machine learning approaches may require preprocessing of data, like rescaling, for example. Deep learning approaches can learn from raw images and extract the most important data information. Deep learning can learn the features from the data automatically, without depending on domain-specific knowledge, as machine learning algorithms often do.

Deep learning approaches require more computational resources than machine learning approaches because there are more parameters to optimize while training.

In short, the choice between machine learning and deep learning methods to solve a problem may depend on the resources available, the system's complexity, and the amount of training data available. Deep learning is more powerful in dealing with complexity and is more flexible, but needs more computational resources.

2.2 Vehicle Detection Based On Machine Learning

The traditional techniques of vehicle detection often struggle to deal with the variety of vehicle appearances, real-world particularities, and lighting conditions. Machine learning algorithms have an important role in the vehicle detection field due to their capability to respond to those demands. These algorithms teach the computer how to identify a vehicle by learning vehicle features and patterns from labeled data. This characteristic makes them more robust and capable of recognizing vehicles from different angles, backgrounds, and lighting conditions.

The machine learning methods for vehicle detection consist essentially of feature extraction and classifier training [1]. In the next sub-sections, these two topics are covered in more detail and some examples of their use in the literature are presented.

2.2.1 Feature Extraction Methods

There are many feature extraction methods that are used for vehicle detection, such as Haar-like features, SIFT, SURF, and HOG descriptors. These methods should easily extract and

differentiate the relevant features of a vehicle and ensure that the extracted features stay stable when the position and type of vehicles change [1].

2.2.1.1 Haar-like Features

The Haar-like features method was proposed by Viola and Jones [2] in 2001. This method was first used for face detection, but it can be used for detecting other objects, such as vehicles.

The basic concept consists of using rectangular patterns called Haar-like features to define the intensity differences inside a specific region in the image. The calculation of the feature values is given by the subtraction of the sum of pixel values in the white regions from the sum of pixel values in the grey regions. The Haar-like features can be of three types:

- Two-rectangle feature: This feature consists of two contiguous rectangular regions which one is white and the other is grey. The two regions have the same shape and size and can be vertically or horizontally contiguous. The value of this type of feature is given by the difference between the sum of the pixels of each region [2].
- Three-rectangle feature: This feature consists of three contiguous rectangular regions which one region is grey located between two white regions. The value of this type of feature is given by the subtraction of the sum of the pixel values of the two white regions from the sum of the pixel values of the grey region [2].
- Four-rectangle feature: This feature consists of four contiguous rectangular regions where it is calculated the difference between diagonal duos of rectangles [2].

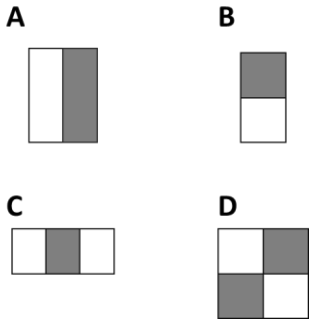

Figure 2.1: Example of the Haar-like features. A and B - two-rectangle feature. C - three-rectangle feature. D - four-rectangle feature. Adapted from [2].

For a faster computation of the rectangle features, the authors introduced the concept of the integral image which is a representation of the original image, where the value of every

pixel is the sum of all pixel values inside the rectangle formed by the top-left corner and the present pixel [2].

Han et al. [3] propose a vehicle detection algorithm using Haar-like features. The algorithm is divided into two procedures: hypothesis generation and hypothesis verification.

In the first procedure, the authors try to find the vehicle candidate regions in the image by searching two types of relevant vehicle features: shadow and vertical edge. The shadow can be found under the vehicles, but sometimes is difficult to extract the region containing the shadow due to the brightness of the image. To overcome the issue, the authors normalize the brightness of the image to find better the shadow region and then, only the region above the shadow point is used to detect the vehicles. The second relevant vehicle feature is the vertical edge which can be normally found in the left and right parts of the vehicles. To get these vertical edges distinctively and clearly, the authors apply a Haar-like feature to obtain the image with the edges of the vehicles. After this step, the vehicle candidate regions are finally found and the authors continue to the second procedure of the algorithm.

The second procedure consists of two verification methods that determine if the vehicle candidates are real vehicles or not. In the first one, the horizontal edge feature of the vehicles is extracted using the horizontal Haar-like feature mask. This method normally removes successfully the candidates that are not vehicles, but sometimes there are some objects having the horizontal edge feature. To overcome this problem, it is used a second verification method which is the calculation of the symmetry of the vertical edge features of the vehicle candidates. If the symmetric edge feature is observed in a candidate, this candidate is considered a vehicle.

The algorithm obtained a short execution time and it can be used for a real-time application, but there is a limitation in the number of detecting vehicles while using a single camera. It was also verified errors due to lighting conditions.

In short, object detection algorithms based on Haar-like features have as advantage points the simplicity and efficiency required to be used in real-time systems, and can capture some minor changes in the appearance of the objects. As negative points, this type of methods has difficulties capturing complex patterns which limits its performance in variable scenes and is very susceptible to variations in scale and rotation, demanding the intervention of other techniques to solve the problem.

2.2.1.2 Scale-Invariant Feature Transform

The well-known computer vision technique called Scale-Invariant Feature Transform (SIFT) [4][5] is used for visual feature extraction, matching, and object recognition. David Lowe created it in 1999, and since then it has been used as the basis for several applications such as

object detection, image stitching, and 3D reconstruction. The robustness of SIFT to changes in rotation, scale, light, and partial occlusion is well established.

Firstly, a scale-space representation of the input image is made by SIFT. It requires repeatedly applying different scales of Gaussian filters to the image. This is done to make the algorithm scale invariant. Key points are recognized by SIFT as local extrema in the Difference of Gaussians (DoG) images after the scale-space pyramid has been constructed. Within the scale-space pyramid, adjacent Gaussian-blurred images are subtracted to produce the DoG images. These key points may be the locations of the image's recognizable features.

A 3D quadratic function is applied to the scale-space extrema surrounding each candidate key point in order to precisely locate key points. By removing key points with low contrast or poorly located extrema, this procedure increases the feature detection process' overall strength. SIFT calculates the local image gradients' main orientation for each key point in the area. The features are more resistant to image rotations because of the rotational invariance provided by this orientation. Then, based on the gradient directions, key points are assigned to one or more orientations.

By gathering information on the distribution of gradient orientations and magnitudes in the vicinity, SIFT creates a descriptor for each key point. The region is broken up into smaller sub-regions, and histograms of oriented gradients are produced. The appearance of the key point is represented by a feature vector created by concatenating these histograms. Distance measures can be used to compare SIFT descriptors that have been retrieved from different images. Potential matches correspond to features with short distances.

A good example in the literature of the usage of SIFT is the paper proposed by Chen and Meng [6]. The authors propose a vehicle detection system using UAVs. The vehicle features must be resistant to changes in scale, rotation, translation, and lighting since the UAV is the one capturing the video. For this reason, it is used the SIFT algorithm which is capable to find the key points resistant to those changes.

It is required to determine the difference between the contiguous scale images in order to identify the key points that are scale-invariant. Each pixel can be compared with its neighbours after the DoG has been generated. Every key point is compared with 26 neighbours in total and a point is classified as key point in the scale's image if it is the minimum or the maximum in the vicinity. A filter is also set to reject points that are poor among the edges and have low contrast because DoG is very susceptible to edges and noise in the image. A direction of the highest gradient direction in the histogram is determined for each key point. A key point's sample region is in the center of its neighbors, and it uses histogram information to account for the gradient orientation of the neighbors' pixels. The peak of the histogram represents the principal gradient direction of the key point's neighbors.

The final phase assigns a descriptor with gradients to each key point. Each descriptor's direction is rotated to align with the principal direction of the image during descriptor generation. The rotational invariance is created by this. Finally, the key point can be identified using the coordination and classify it using the descriptor.

After the integration of the SIFT algorithm with the ISM, in order to improve the detection accuracy, the authors use an SVM classifier to classify the key points. The method achieved a result of 94.13% in accuracy, but it was verified a slow processing time due to the need for every key point to be processed by SIFT.

2.2.1.3 Speeded-Up Robust Features

The feature extraction and matching approach in computer vision called Speeded-Up Robust Features (SURF) was introduced by Bay et al. [7]. SURF is utilized for a variety of applications, including object detection, image stitching, and tracking, much like SIFT, however, it provides speed advantages while keeping high performance.

SURF uses integral images as one of its core concepts. Any rectangular area of an image can have its pixel value sums quickly calculated using integral images. For the effective computation of numerous image features, this is essential. Similar to SIFT, SURF uses subsampling and Gaussian blurring to create a scale-space pyramid. SURF, on the other hand, uses box filters for convolution, which are more easily computed with integral images. SURF uses an approximation of the Hessian matrix to find key points. In order to do this, the determinant of the Hessian matrix must be calculated for various scales and positions. When the determinant hits a maximum or a minimum, key points are found that denote alterations in both scale and position. Similar to how Haar-like features are computed, SURF computes orientation using Haar-wavelet responses. By detecting the predominate direction of gradient responses, it provides rotation invariance.

SURF descriptor measures local intensity patterns around a key point using gradient orientations and magnitudes. With the aid of box filters to approximate Gaussian derivatives, the process is speeded up. Matches are determined using the lowest distance between descriptor vectors, much as SIFT.

Hsieh et al. [8] proposed a method to detect vehicles using a symmetrical SURF descriptor. The SURF descriptor is a very useful feature detector for real-time applications, but there's a gap, which is its inability to detect symmetric objects. With this in mind, the authors introduced a symmetric transformation to improve the SURF's performance. The method was then tested for a vehicle make and model recognition system achieving convincing results.

The symmetric SURF descriptor was later used in the paper of Momin and Kumbhare [9] for a video surveillance system to detect vehicles in real time. The proposed algorithm starts

by applying the symmetric SURF descriptor to each frame of the input video in order to extract the key points. After that, the algorithm tries to find the mirror of each detected key point. If it is found a matching key point in the same row as the detected key point, this matching pair is denoted as a pair of symmetric SURF points. The center of every detected matching pair is then calculated and is the product of a histogram-based analysis in order to identify the central line of the vehicle. The central line is used to detect the vehicle's shadow that can be found underneath this line. The shadow is fundamental to finding the bottom boundary of the vehicle candidate's ROI and can be detected using a Gaussian shadow model. The remaining ROI boundaries (up, left, and right) are calculated, and if some conditions are satisfied, the vehicle candidate is successfully identified as a vehicle with the ROI representing the vehicle's image.

2.2.1.4 Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG) was first mentioned in the 2005 paper of Dalal and Triggs [10], and since then, it has been an extremely used feature descriptor for object detection and recognition. By examining the distribution of gradient orientations across different image regions, HOG is able to capture the local gradient information in an image.

HOG starts by calculating the image's gradients. Using derivative filters like Sobel filters, gradients can be generated to reflect the intensity variations in the image. Gradients are calculated in both the vertical and horizontal orientations. The gradient's magnitude and direction are determined for each individual pixel. While the gradient orientation denotes the change's direction, the gradient magnitude indicates how strong the intensity change is. Normally, orientations are quantized into a number of bins, such as nine bins that range from 0 to 180 degrees.

The image is broken up into tiny cells, usually measuring 8x8 pixels. Gradient orientations are gathered within each cell, and is created a histogram of orientations. The distribution of gradient orientations within the cell is quantized by this histogram. Neighboring cells are organized into blocks in order to adjust for regional changes in illumination and contrast. The blocks move over the image with covering areas. The histograms from the cells within each block are merged to create a bigger feature vector. The normalization process improves the histogram's robustness against variations in lighting. The final feature vector, also called the HOG descriptor, is created by merging the normalized histograms from various blocks.

Cao et al. [11] proposed a vehicle detection system where the videos are recorded from a low-altitude airborne platform. The biggest challenges for a system like this are the lighting variance and the complex scenes. To deal with these problems, the authors use the HOG feature extraction method due to its invariance to lighting changes and good performance in describing object appearance and form. However, HOG features can be very dispendious in time

during training and classification procedures due to their high dimensionality. To reduce the feature vectors' dimensionality, it is proposed a boosting HOG feature method using the AdaBoost classifier. In this way, the execution time of the detection process can be decreased.

As referred previously, HOG divides the image into small cells which are grouped to form blocks to cover a bigger spatial area. A sliding window scans the image to obtain these blocks and the narrow appearance information. The narrow appearance information is not enough to extract the vehicles' large-scale shape features, then a multi-scale method is implemented by designing three types of block sizes. The larger size of the blocks allows the system to obtain the vehicles' global information and thus improve the detection performance. In the end, an SVM classifier is trained using the final feature vector for the vehicle classification.

The final results showed a system capable of detecting vehicles fast, achieving also, an accuracy rate of 90% in the detection process.

2.2.2 Classifiers

The extracted features are then used by classifiers to determine if a specific target is a vehicle or not. The most common classifiers used in machine learning are SVM, decision tree, and AdaBoost.

2.2.2.1 Support Vector Machine

A Support Vector Machine (SVM) is a learning algorithm that can be used for classification and regression purposes. It was first introduced by Vapnik [12] in 1995. It tries to find a hyperplane that divides data into different classes. The best hyperplane for a system is the one that maximizes the margin between the two classes' nearest points, which are called support vectors. SVM tries to find the ideal hyperplane but also tries to classify the training data properly. It is used a cost function to evaluate the hyperplane's quality and by minimizing this cost function, it can be chosen the ideal hyperplane.

In the work of Chen et al. [13] an SVM classifier is used to classify vehicles into four different categories (van, car, motorcycle/bicycle, and bus). The data is provided by a CCTV camera installed on a street. Within a restricted area of the image and at a predetermined distance from a virtual line, vehicles are manually segmented. Using the segmented vehicles' binary silhouettes, a collection of shape and size features are extracted to form the feature vector. SVM uses each feature from this vector to classify the dataset achieving a true positive accuracy rate of 96%.

2.2.2.2 Decision Tree

A standard supervised machine learning algorithm for classification tasks is the decision tree-based classifier. It is an architecture that resembles a tree, whose internal nodes individually make decisions based on a set of binary options. A new branch is created for each option, and each leaf node corresponds to a class label. Decision trees are helpful for both novices and specialists in a variety of areas since they are simple to comprehend and visualize.

When there is a large number of features in the system, sometimes it is used a Random Forest regression/classification method, which is a combination of several decision trees that merges their estimations into a final one. This helps to decrease overfitting and to increase accuracy.

A regression method based on Random Forests for moving and static people-counting on station platforms is proposed by Farhood et al. [14]. The main idea is to create a model that by learning with a regression based on Local Random Forests can predict the crowd density in a video frame. In this way, the authors avoid large consumption of resources, compared to Deep Learning approaches.

Opposing to what is done in other methods that use random feature patches, in this paper, the image is divided into a determined number of fixed sub-windows to save time and avoid the perspective distortion issue.

The system's inputs are some annotated images, the extracted features of the images, and a codebook (package of features of training examples). Instead of using the bounding box annotation, the authors put a dot at the center of each person in each frame, in order to specify the person's position. Using a bundle of training images and taking into account their ground truth densities, each frame is divided into sub-windows and for each sub-window, the SIFT features are extracted in order to project them to a density map by learning a decision tree. A Random Forest is generated using decision trees that rely on the connections of sequential estimations with different inputs. A Random Forest is generated for each sub-window. The combination of counts of sub-windows is used by linear regression to predict the total number of people.

In the paper of Chen et al. [13] (referred in the section of SVM) is also used a random forest classifier to categorize the vehicles into four classes, as described previously. The authors used two classifiers (random forest and SVM) to compare their performance. The random forest classification has as advantages the resistance to overfitting and a quick training process. However, in comparison with the SVM classifier, the authors realised that the SVM outperformed the random forest classifier by achieving a better accuracy rate and lower computational cost as well.

2.2.2.3 Adaptive Boosting

Adaptive Boosting (AdaBoost) was first presented by Freund et al. [15]. AdaBoost is a widely known ensemble learning algorithm that improves weak classifiers into stronger ones by learning from their mistakes. The definition of a weak classifier is one that outperforms random guessing but still does badly at assigning classes to objects.

AdaBoost sets for each training sample some initial weights. All instances are given equal weight at the beginning. The algorithm advances over iterations. A weak classifier is trained on the weighted training data during each iteration. Taking into account the weighted examples, the performance of the weak classifier is assessed, and its error is calculated. Depending on how well it performs, each weak classifier is given a weight. A weak classifier's contribution to the final strong classifier is indicated by assigning a higher weight to the weak classifier with lower error rates. If a weak classifier has a lower error rate, a higher weight is assigned to this classifier contributing this way to the final strong one. Combining the weighted predictions of each weak classifier yields the final strong classifier. A greater importance is given to the weaker classifiers who are more exact while making the final selection.

The final prediction is given to the class that received the majority of votes, weighted by the classifier weights.

A representative application of the AdaBoost algorithm in the literature can be found in the paper of Haselhoff and Kummert [16] where a vehicle detection system is proposed. The features are extracted using the Haar-like features method and the AdaBoost classifier select relevant features based on the lowest weighted classification error. The weak classifiers are combinations of the selected features, which then become a strong classifier. Some strong classifiers are combined in a cascaded arrangement, where just by using some features can reject quickly sub-windows that do not have objects similar to vehicles, thus making the system capable of operating in real time.

2.3 Vehicle Detection Based On Deep Learning

Deep learning is a field that belongs to machine learning which uses neural networks to learn from data and make decisions based on the data information. As previously referred, deep learning algorithms are more robust and make more accurate estimations than machine learning algorithms, by learning high-level data representations automatically and having the ability to be trained with large datasets.

The most common type of deep learning algorithms used to detect objects are convolutional neural networks (CNN).

This section presents some of the approaches found in the literature using different types of deep learning algorithms to accurately detect vehicles.

2.3.1 Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Network) [17] is a two-stage object detection algorithm. The first stage is the region proposal network (RPN), where a set of regions of interest that are probable to contain objects are proposed. Then, these ROIs are sent to the second stage (detection network) which will use them to classify the objects and adjust their locations.

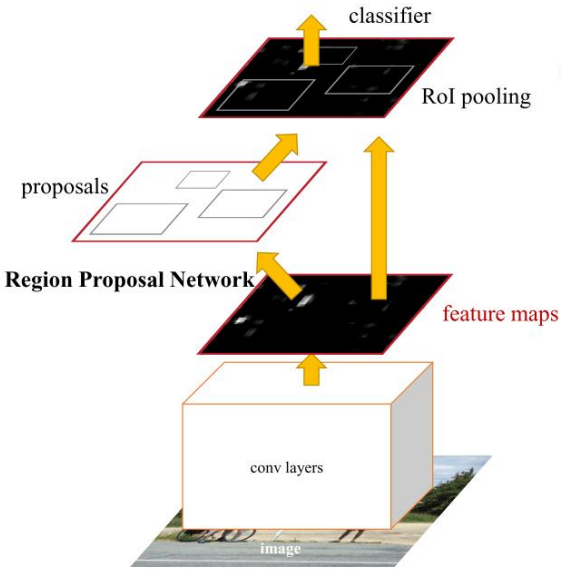

Figure 2.2: Faster R-CNN network. Obtained from [17].

RPN applies anchor boxes and sliding windows to the image in order to estimate object proposals. These anchor boxes have a fixed size and can be used for different aspect ratios and scales. After RPN applying the anchor boxes to the image, the ROIs are generated based on their scores.

The detection network, which is an FCN (Fully Convolutional Neural Network), has as input the ROIs generated by RPN and generates the object class scores and the objects' bounding boxes. It also adjusts the objects' locations by using the ROIs' ground-truth labels.

Faster R-CNN is a very accurate and fast algorithm for object detection purposes and is widely used in various studies.

Fan et al. [18] conducted an analysis of the Faster R-CNN's performance for a vehicle detection system. In this paper, the authors changed some parameters such as training and test scales, and the number of proposals using the KITTI dataset to see how these parameters can affect the behavior of Faster R-CNN.

Both training and test scales are set as 1000 pixels by default. With this value, the training did not perform very well, therefore Faster R-CNN was trained for a range going from 800 to 1800 pixels. The authors concluded that the higher the training scale is, the better the performance is. For the test scale, it was tested ranging from 800 to 2500 pixels. The results showed that the performance is dependent on the test scale used, which can suggest that scale invariance is data-dependent. It was also observed that if the test scale is smaller than the training scale, the performance is poorer.

In order to know the number of proposals needed, the authors observed that with few proposals, Faster R-CNN could achieve good performance as well as using a larger number, and sometimes even better.

In the work of Espinosa et al. [19], a comparative study is done between Faster R-CNN and AlexNet models for moving vehicle detection. A video of a real urban scene with good weather conditions was used for the experiments. The authors state that the Faster R-CNN outperformed the AlexNet model by having a better correct detection rate (fewer false detections), agreeing to the F1-score metric, and achieving an almost real-time performance unlike AlexNet's performance, which was further from that.

2.3.2 Single Shot Detector

Single Shot Detector (SSD) [20] is an object detection algorithm that only needs one convolutional neural network to generate object proposals and classification. It is known as a one-stage detector due to not having a separate ROIs proposal stage like the Faster R-CNN algorithm, which is a two-stage detector.

The algorithm consists of several pooling and convolutional layers extracting features from the image and then, the following prediction layers predict the bounding boxes' locations and the object class scores for each anchor box. The last detections are generated by merging these estimations and eliminating duplicates.

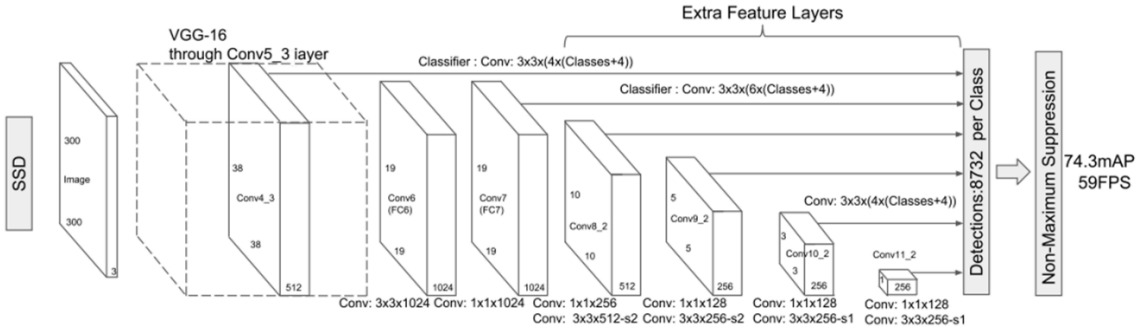

Figure 2.3: SSD architecture. Obtained from [20].

The SSD algorithm's one-stage detection is very appropriate for real-time object detection and has good performance and high accuracy.

The paper of Chen et al. [21] proposes a combination of SSD and MobileNet algorithms for fast vehicle detection. MobileNet is used in this work instead of VGG16 as the backbone feature extraction network for the SSD algorithm to improve its performance.

MobileNet [22] is a deep convolutional neural network architecture aimed at mobile and embedded devices. It is composed of depthwise separable convolution layers that are inserted with pointwise convolution layers. It is applied an individual filter by the depthwise separable convolution layers to every input channel in order to decrease the number of parameters and the network's computational complexity. The results of the depthwise separable convolution layers are combined with the pointwise convolution layers, which generate the final feature maps.

The usage of smaller filters, step values in the initial layers, and global average pooling instead of fully connected layers allows MobileNet to have a significant reduction of the network's computational complexity and a smaller number of parameters when compared to habitual CNN architectures. MobileNet is commonly used in several computer vision studies and when computational resources are scarce.

The authors in [21], show that using the MobileNet network can improve the detection accuracy and reduce the inference time of the SSD algorithm in comparison with the original backbone network.

2.3.3 You Only Look Once

You Only Look Once (YOLO) [23] is a well-known object detection algorithm that only uses one convolutional neural network and it is a one-stage detector, as the SSD algorithm. The YOLO family has been evolving from year to year, with new versions appearing on a regular basis. So far, the YOLO family ranges from YOLOv1 to YOLOv8 (presented in 2023). Each version offers improvements over the previous ones.

The original YOLO (YOLOv1) divides the input image into a grid of cells. Every cell is in charge of estimating the existence and location of objects that are contained within. The locations of the bounding boxes and the object class scores are estimated by the network. The last detections are generated by merging these estimations and eliminating duplicates.

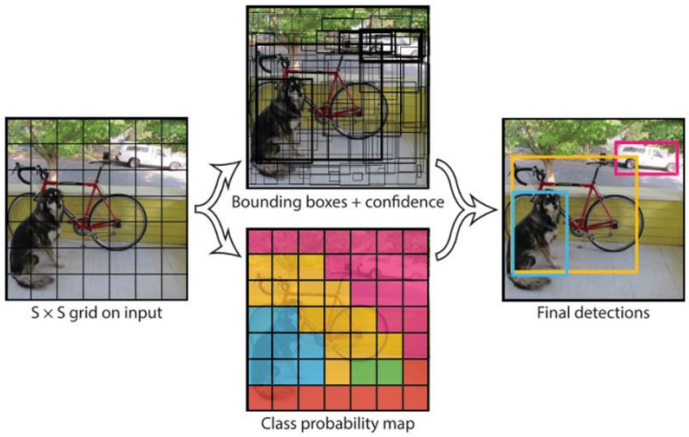

Figure 2.4: YOLOv1 working approach. Obtained from [23].

The YOLOv1 architecture consists of 24 convolutional layers, 4 max-pooling layers, and 2 fully-connected layers, as can be seen in Figure 2.5.

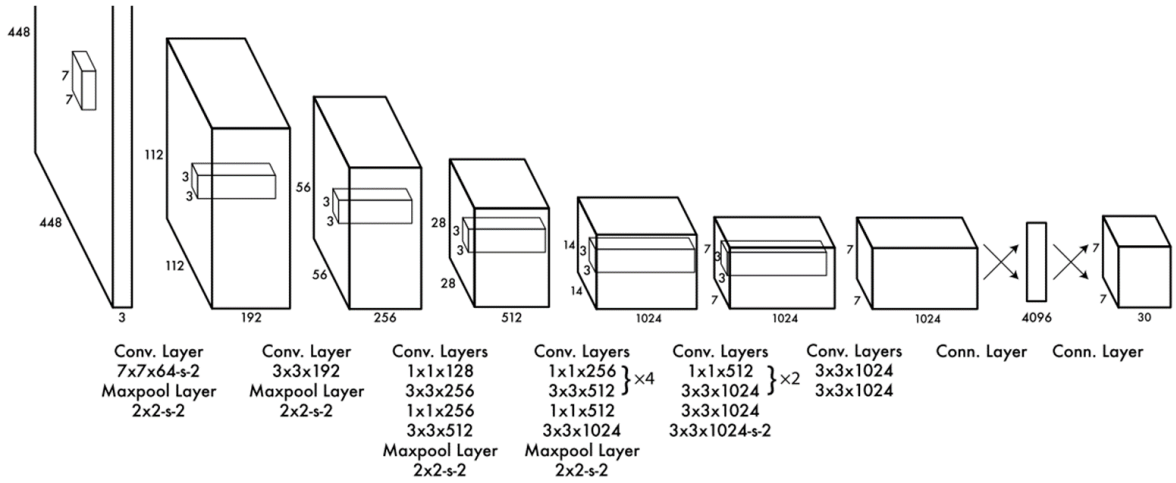

Figure 2.5: YOLOv1 architecture. Obtained from [23].

YOLO predicts several objects in a single pass and for that reason, it is considered a real-time object detector. YOLO uses anchor boxes, which have different sizes and shapes, to increase the accuracy of the predictions, like Faster R-CNN and SSD algorithms.

The YOLO family reaches very high performance and accuracy and it is a popular selection for real-time computer vision applications.

Benjdira et al. [24] conducted a comparison study between YOLOv3 and Faster R-CNN performances for car detection using aerial images captured by UAVs. The authors use the same training and test datasets for both models. With the results obtained, it can be observed that the precision is very similar for both models, being high, which indicates that both models are quite good at detecting true cars. YOLOv3 outperforms Faster R-CNN in sensitivity which indicates that Faster R-CNN misses more car instances than YOLOv3. YOLOv3 is very accurate in extracting all the car instances in the image. Taking into account the F1-score, YOLOv3 outperforms Faster R-CNN as well, indicating that YOLOv3 is an algorithm more robust. Considering the processing time measure, it was observed that YOLOv3 spends less time detecting cars, having an inferior time average rate per image.

In the work of Xu et al. [25], vehicle detection is done using a YOLOv3 with some modifications. The authors state that object detectors as YOLO are not well prepared to detect vehicles via aerial image, because in this type of image, the vehicles have smaller sizes in comparison with general data, and sometimes the vehicles' roofs are misclassified as house's roofs. In order to improve the performance of YOLO while detecting vehicles via aerial images to a desirable level, the authors set the convolutional layers' number to 75, thus increasing the network's depth. Taking into account that the top-level features can provide more information about small-sized objects, the authors change the links between down-sampling and up-sampling layers to emphasize these features.

2.4 Tracking algorithms

The principal tracking algorithms used in the literature for tracking objects are the Kalman filter and the Deep SORT. Apart from these algorithms, there are some other methods less used, which is the case of the work of Rahman et al. [26] where is used a centroid tracking algorithm to track vehicles. The authors propose a system able to detect and track wrong-way vehicles using image information from surveillance cameras.

The vehicles are detected using the YOLO algorithm and their bounding boxes serve as input for the centroid tracking algorithm, where the central point of each bounding box is calculated to know the vehicle's position. This tracker is based on the presumption that there is an extremely small variation in a vehicle's position across consecutive frames of the video.

If the centroid of the vehicle's bounding box is inside of a particular region of interest defined initially, an identification number is assigned to this vehicle and it is entered in a tracking list. This vehicle is only removed from the tracking list if its bounding box centroid leaves the region of interest.

In order to know the direction of a vehicle, the authors use the centroid's height. When a vehicle enters in the region of interest, its centroid's height is calculated, and as long as the vehicle moves inside the region of interest the centroid's height continues to be calculated and updated. If the initial height calculated when the vehicle entered the ROI is smaller than the consequent heights, the algorithm assumes that the vehicle is moving in the right direction (towards the camera), and when this does not occur, it is assumed that the vehicle is going in a wrong direction.

2.4.1 Kalman Filter

The Kalman filter [27] is a mathematical method used to do estimations about the state of a system over time. It is composed of two procedures: estimation and correction.

In the estimation procedure, it is used a system's model and the current state prediction to estimate the system's state at the following time step. In the correction procedure, it is updated the state prediction based on the new measurements.

The Kalman filter is commonly used in computer vision applications due to its efficiency in predicting the system's state over time. It can handle imperfect measurements and it has the capacity to incorporate the system's model, which makes it a highly requested algorithm.

Teoh and Bräunl [28] introduce a method for vehicle tracking based on the integration of a Kalman filter and a reliability point system. The system is used to track the movement of vehicles in consecutive frames of a given video.

The authors use the Kalman filter to have the best prediction of a detected vehicle's location in the subsequent frames, helping to reduce the search region for re-detecting vehicles, which saves a lot of time. It also reduces false detections because areas that do not contain vehicles are left out of the search. The Kalman filter's smoothing impact helps the tracking results from irregularities related to the measurement error.

The reliability point system is used to evaluate the quality of tracking. This system consists of assigning two reliability points to each vehicle when it is first detected, and then based on how consistently this vehicle is re-detected, its number of reliability points can decrease or increase. If the vehicle has more than two points, it appears on the tracking output. If the vehicle does not have any points, it is removed from the tracking list.

2.4.2 Deep SORT

Simple Online and Realtime Tracking (SORT) [29] is a real-time multiple-object tracking algorithm.

The algorithm begins to detect objects frame by frame using an object detection algorithm and then, using a data association algorithm (Hungarian method), associates the detections with the existing tracks in order to know which detections belong to which tracks.

It uses a Kalman filter to estimate each track's location in the following frame and to update it based on the new detections.

In order to improve the real-time performance, avoid identity changes, and be able to keep tracking objects through higher periods of occlusion in the image, the SORT algorithm was improved to the Deep SORT [30] algorithm, which uses a deep association metric that combines appearance and motion descriptors.

Through appearance-based features, Deep SORT can learn the object's appearance and use it to track it over time. Deep metric learning, by learning a distance metric (Mahalanobis distance), compares the similarity between the detections and the tracks. These deep learning techniques make it no longer necessary to use a data association algorithm such as the Hungarian method to associate the detections with the tracks as SORT does.

Deep SORT also has a re-identification mechanism to re-identify and continue to track the object when it is occluded in the image for a longer period of time.

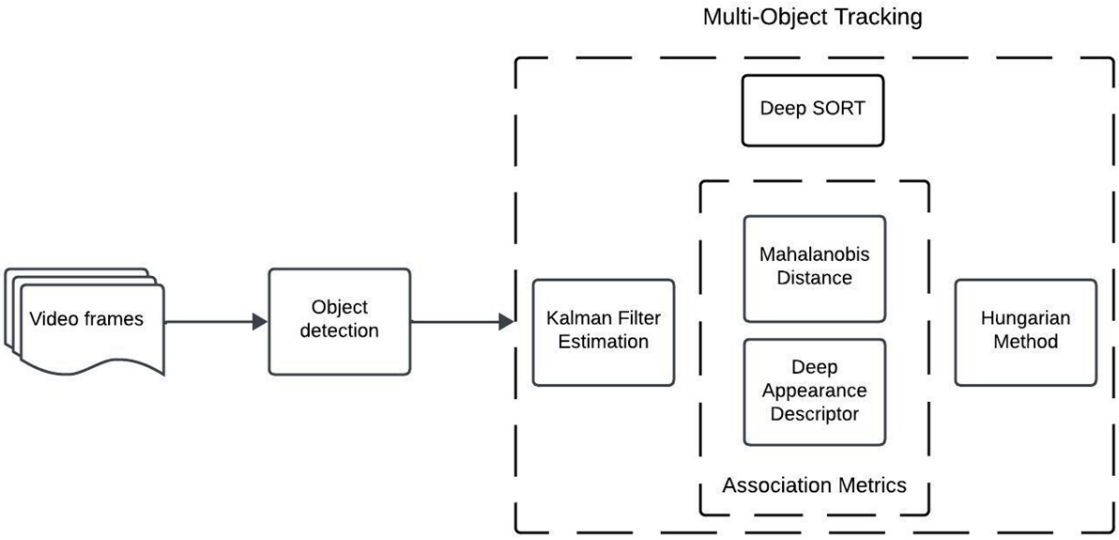

Figure 2.6: Deep SORT working diagram.

Doan and Truong [31] propose a real-time system using YOLO and Deep SORT algorithms to detect and count vehicles. The Kalman filter of the Deep SORT algorithm can help to track the objects by providing the information of their previous states and predict the current ones. By defining a threshold in the first frame, the Kalman filter also avoids vehicles' duplicate tracking.

In this work, the counting procedure starts by providing the outputs of YOLO as inputs to the Deep SORT algorithm. The Deep SORT assigns an ID to each detected vehicle in a region of interest previously set and the counter relative to the vehicle's class is incremented by one.

In the paper of Hou et al. [32] is implemented a low confidence track filter to be integrated with the Deep SORT algorithm. The objective of the implementation of this filter is to reduce the impact of inaccurate detections on vehicle tracking, i.e. reduce false positive tracks. The tracks with a low detection confidence are removed in their initial frames. This is done in order to reduce the detection's confidence value and to prevent losing detections.

2.5 Traffic Light Recognition

Traffic light recognition is fundamental for controlling traffic red-light infringements since it allows locating the exact spot of a traffic light on a given image/frame and having information about its specific color emitted.

Commonly, in the literature, traffic light recognition is used by autonomous vehicles and ADAS to assist drivers and to provide more security to the roads. There are some challenges associated with this problem, as well as different types of methods to solve them. After a state-of-the-art review, it is noticeable that the proposed methods can be synthesized by three topics such as color space, feature extraction, and classification/validation.

2.5.1 Challenges

Detecting and classifying traffic lights can be a hard task to handle because there are multiple factors involved that can compromise the success of traffic light recognition.

Usually, traffic lights have different appearances, such as different shapes, sizes, colors, and designs. Traffic lights' designs also differ from country to country and their appearance can change over time due to weather conditions that can degrade them. Sometimes, traffic lights are occluded by other objects (vehicles, buildings, trees, etc.) which makes it difficult to detect them. In the case of a deep learning-based system, data augmentation can be used to help increase artificially the training examples, which allows the dataset to be better represented by traffic lights with different appearances and can also overcome the occlusion problem by creating examples of traffic lights occluded.

Lighting conditions are also one of the most important factors to take into account. Illumination variations provoke changes in the perception of the traffic light's colors. For example, on bright days, sometimes, the green color becomes almost imperceptible, appearing to be off. If the traffic light is in the shade, the yellow light can look darker becoming almost red which

can create some confusion while predicting the traffic light's state. Color normalization techniques such as white balancing are often used to decrease the impact of illumination variations. The use of adaptive color thresholding techniques instead of using a fixed color threshold can help fine-tune the threshold values based on the image characteristics, reducing the impact of intensity variations in lighting.

A good real-time performance of traffic light recognition is also an important aspect for decision-making. Both for autonomous vehicle systems and for traffic control systems it is necessary to know the state of the traffic light immediately because if there's a delay it may be enough to occur an accident or to not capture a red-light infringement. The real-time requirements implicate more usage of computational resources and a limitation on the complexity of algorithms. GPUs are often used by systems that need to perform applications in real time due to their high computing power.

2.5.2 Color-based methods

A traffic light emits three different colors (red, yellow/amber, and green) and the authors propose some methods using this information to detect and distinguish traffic lights from the background. These methods are very sensitive to lighting conditions and researchers often combine them with other methods to have a better performance of the traffic light recognition.

2.5.2.1 Color Space Conversion

The original images containing a scene with traffic lights to be detected are often in RGB color space. The RGB components can be affected by lighting conditions, then many authors prefer to convert this color space to another, for example, to HSV. HSV is well-known to be invariant to illumination changes.

A color space is a mathematical model that contains components each representing different color aspects. There are several color spaces that are destined for different purposes and allow the manipulation and analysis of colors differently. The two most commonly used color spaces for traffic light recognition are RGB and HSV. RGB color space is most used for digital imaging and can directly represent how screens emit or cameras capture colors. RGB represents colors by a combination of three colors: red, green, and blue. HSV is used to select and manipulate colors and is often considered to define the color similarly to human vision. It has three attributes: hue, saturation, and value. Hue represents the type of color, saturation represents the color's intensity and the value represents the brightness.

2.5.2.2 Color Segmentation

After the color space conversion, color segmentation is applied by thresholding the image to separate regions of interest with colors identical to the traffic light colors. The thresholding is done by establishing lower and upper bounds for the color space components.

In the work done by Omachi and Omachi [34], color segmentation is used to find candidate regions of traffic lights. The procedure starts by converting the RGB input image into a normalized RGB image in order to not suffer from color changes due to lighting conditions. The normalization is done by dividing every red, green and blue component of each pixel by the sum of the three components of the pixel.

Candidate regions of interest for the traffic lights are extracted from the normalized RGB image. A pixel belongs to a candidate region if it satisfies some conditions. If the normalized components (R, G, and B) from the pixel are lower or higher than a determined value, the pixel belongs to the candidate region. The selected pixels form a connected region which is a candidate region of the traffic light. Some authors apply morphological operations after the color segmentation to improve the quality of the candidate regions. For example, erosion can be used to remove undesired noise and small regions, and dilation can be used to connect some fragmented regions.

2.5.3 Feature-based methods

This type of methods relies on recognizing geometric features or characteristics of traffic lights to detect and classify them. Template matching and Hough Transform are some of the feature-based methods most used in the literature. These methods can be very effective in detecting traffic lights when they have a consistent appearance and are captured without lighting changes. This is not often possible and researchers integrate these methods with other techniques to have a better performance.

2.5.3.1 Template Matching

Instead of using color segmentation to detect regions of interest for the traffic lights, there is another technique called template matching that can be applied to the image and locate the traffic lights.

Template matching consists of having a template image (smaller image/pattern) of the object to be detected and comparing it to regions of the larger image to find the best match. It is important to choose as a template a cropped image that best represents the object containing the distinguishable features that are expected to be found in the larger image. The template is often moved pixel by pixel in the larger image and compared with each fragment of the same size as the template's size. The comparison is done using a similarity confidence score which

the higher it is, the better the match. The position in the larger image with the highest score corresponds to the location of the object. Like all the other techniques, template matching has its own limitations which include high sensitivity to light, orientation, and scale changes. It can be computationally demanding if applied to the entire image and might not perform well in complicated scenes where there's chaos or similar patterns. It is recommended to use this technique combined with other ones like machine learning in order to improve the quality of the detection performance.

De Charette and Nashashibi [35] use the template matching technique to recognize the traffic lights present in a scene. The template matching is only applied after the detection of spots lights in a grayscale image. The candidates for traffic lights are created uniquely where the spots are detected in order to make the system faster. These candidates are evaluated by a similarity confidence score which will determine if the candidate is a traffic light or not. With the purpose of developing a generic system capable of detecting different types of traffic lights, the authors use adaptive templates which allow adding new element types and operators to the system.

The adaptive templates are a combination of the two-dimensional shape representations of the three-dimensional elements that form the traffic light. Each element can have algorithmic operators linked to it and both of them can have distinct weights and similarity confidence thresholds. They can also be listed as "nondiscriminatory", which means that they are not relevant for the evaluation. The similarity confidence score of an element is given by the weight and confidence scores of each descendant element or associated operator. The authors use a top-down method to specify the geometry of the template, as shown in Figure 2.7. The templates consist of elements describing the visual look of a traffic light and the elements can be of several types such as Square, Circle, Rectangle, etc.

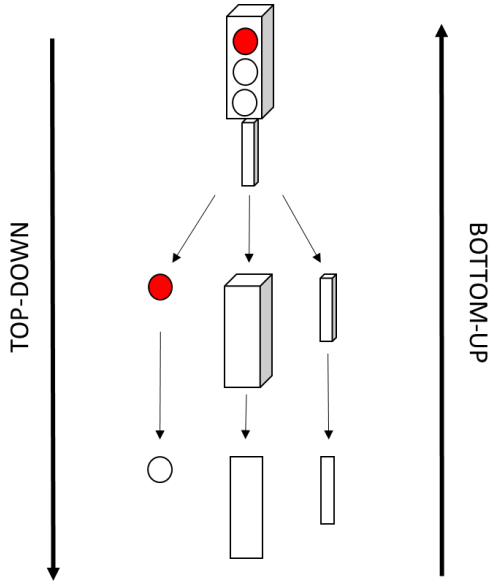

Figure 2.7: Representation of top-down and bottom-up methods. Adapted from [35].

The operators associated with the elements take the elements as inputs and output values according to their internal behavior. For example, there is an operator that receives an element as input and outputs its position in the three-dimensional space using the element's metric properties and the camera calibration data.

The template matching evaluation is done using template candidates. These templates, as referred previously, contain elements and operators. All these elements and operators associated are evaluated hierarchically. Confidence thresholds are set for every operator and element to perform the evaluation. If a confidence score of an operator or element does not reach the desired confidence threshold, the template candidate is rejected. When a candidate is accepted as a traffic light, the authors define other three templates that represent the possible traffic light's states. These templates are evaluated similarly as described before and the accepted one indicates the traffic light's state.

2.5.3.2 Hough Transform

This technique is used to localize shapes inside an image using the mathematical equations that parameterize them. In the literature, the Hough Transform is mostly used to detect circles or rectangles, but it can be extended to other shapes too. In the case of traffic light recognition, the researchers often use this to detect circles in the image that can represent the lights of the traffic lights since the emitted lights have a circular shape.

The Hough Transform aims to represent a shape in a transformed/parameterized space, where every point in the parameterized space relates to a possible occurrence of the shape in the original image. The collection of points in the parameterized space can help identify the shape's parameters.

The Widyantoro and Saputra [36] approach can help understand this technique. The authors use a circle-shape detection based on Hough Transform to detect and recognize traffic lights.

The parameterization of a circle contains three parameters: a, b, and r, where a and b are the coordinates of the circle's center and r is the circle's radius. The objective is to find the values for these parameters that fit the circle. This is done by crossing the edge pixels of an object. When the radius is known, circles are drawn in the parameter space with each edge point corresponding to a circle's center. When the radius is not known, circles are drawn iteratively from the minimum to the maximum values of the radius, as Figure 2.8 shows. In this case, the radius' minimum and maximum values and the increment of the values are the algorithms' parameters. A counter is set to count the number of circles that are passing across a specific location. The object is recognized as a circle if the highest value of the counter surpasses a determined threshold and the location of the counter corresponds to the circle's center. Otherwise, the object is rejected as a circle. The threshold is set as the smaller limit to define a circle, which the higher it is, the more accurate the circle's detection is.

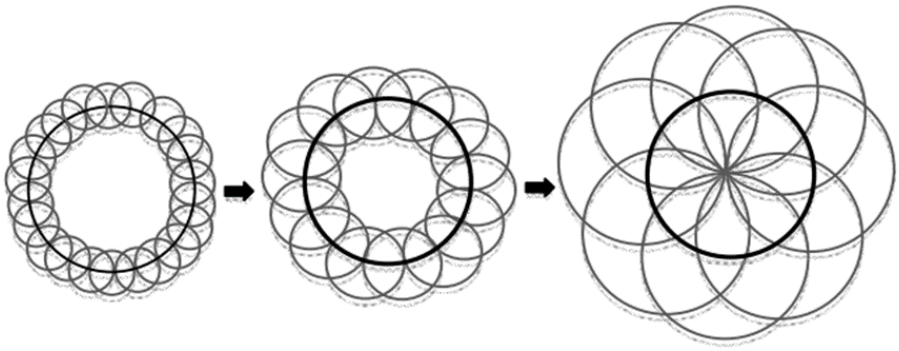

Figure 2.8: Example of the search for the circle object when its radius is unknown. Obtained from [36].

The Hough Transform can be a very computationally intensive technique when applied to an entire image. To reduce the computation time, the authors only apply the technique to the upper part of the images, because normally it is where the traffic lights are, ignoring the images' lower part.

2.5.4 Deep Learning-based Methods

In recent years, the demand for traffic light recognition systems capable of operating in real-time with high precision has grown. The use of deep learning algorithms to detect and classify objects has also greatly helped this area.

In the paper of Behrendt et al. [37], is proposed a system based on deep learning to detect, track, and classify traffic lights for automated driving. The traffic lights' detection is done

using a trained neural network. The authors adapted the network to detect small traffic lights in the image. For the tracking, it is used stereo vision, and odometry information to predict the movement of traffic lights relative to the vehicle's movement. It is also used another neural network to correct the location prediction obtained in the tracking process.

First it was used the YOLO model to detect the traffic lights, but the authors observed a low performance on their dataset when using the classification part of the YOLO architecture. The dataset contained small-sized traffic lights, then it was required to adapt the YOLO model in order to detect small traffic lights. By removing the YOLO's classification part, and only using the model to detect objects, the results improved. The authors created a small classification network to distinguish the traffic lights' states and also to remove false positives.

An object tracker was implemented also to fill in some of the gaps left by the detector. The tracker starts by receiving the bounding boxes provided by the detector. Every time a new detection is found, the average distance between the four corners of the nearest bounding box that was previously tracked is calculated. The tracked detection is updated to the new one if the calculated distance is less than a threshold defined as twice the width of the tracked bounding box. Otherwise, a new object is initialized to be tracked. As the system is used for autonomous vehicles, as the vehicle moves, the position of the traffic lights also moves in the image. To estimate the position of the traffic light in each frame, the vehicle's stereo cameras are used to provide odometry and triangulation information.

Another example of the use of deep learning in the field of traffic light recognition is the work of Kulkarni et al. [38], where it is used a Faster R-CNN model to detect and recognize traffic lights in self-driving vehicles. The deep neural network model was trained on a dataset containing only Indian traffic lights. For the classification of the traffic lights, it was created five classes such as left, right, straight, yellow, and red. For the training, it was used a GPU with the Tensorflow framework installed. The system achieved decent results when classifying images containing traffic lights on Indian streets, but it had yet to be tested on videos and in real time.

3

METHODOLOGY

In this chapter is presented all the work developed. Firstly, a system overview is presented to explain how the system works in general and how each module is interconnected with the other. All the setup required and the software used for the system's implementation are then presented in the following section. After that, a section is dedicated to each constituent module of the system to explain in more detail how it was designed and developed. Figure 3.1 shows the project workflow with all the steps taken to reach the final result.

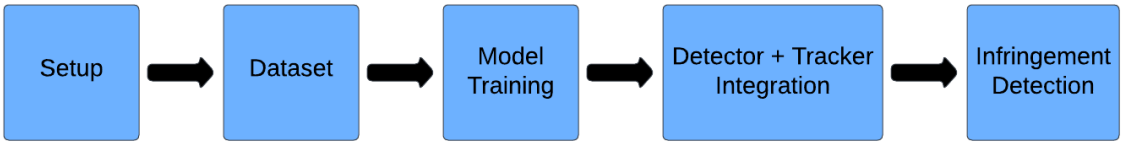

Figure 3.1: Project's workflow.

3.1 System Overview

A system capable of detecting red traffic light infringements must first be able to identify what vehicles are and distinguish them from other objects that may appear on the scene. The same applies to traffic lights: the system must be able to identify the traffic light in the scene and then be able to recognize the color it emits so that it can tell when vehicles should or should not pass the light. For this purpose, an object detector is required.

It's not enough just to detect vehicles and traffic lights. For the system to know if vehicles have run a red light, it must know how to follow their individual movement. It is essential to know in which direction vehicles are traveling over time so that it can know if a vehicle has run the light when it shouldn't. For this to happen, an object tracker must be integrated with the object detector.

It is also necessary for the user to tell the system which areas in the image correspond to the area before the traffic light (where vehicles are coming from), the area after the traffic light (where vehicles should not go if there is a red light), and also the traffic light of interest, because sometimes there are several traffic lights in a scene and the system must know which traffic light has to be monitored.

By combining the information provided by the user and the one collected by the detector and the tracker, the system is able to identify red light infringements on its own and also record the vehicles that commit them. This system allows vehicles that commit infringements to be fined, thus helping the security forces to maintain order and safety on the roads. An overview of the system can be seen in Figure 3.2.

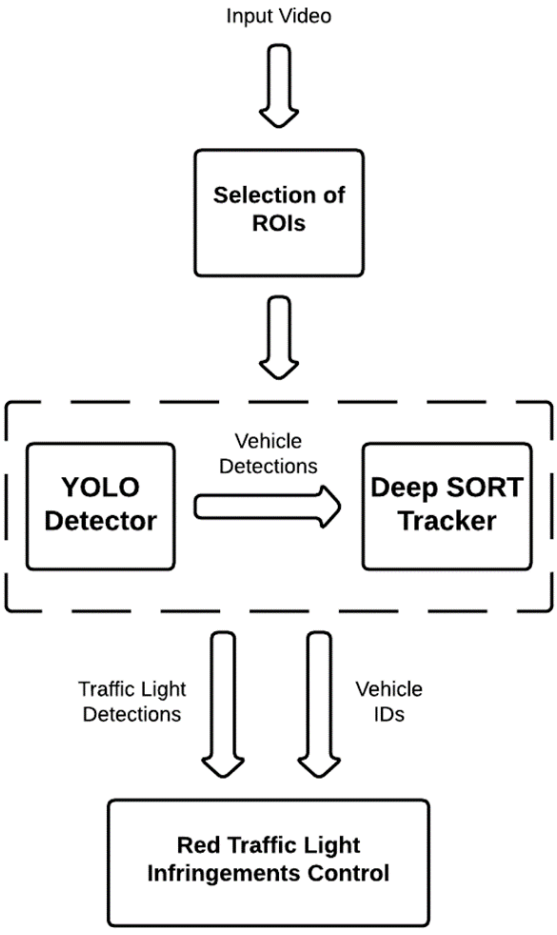

Figure 3.2: System overview.

3.2 Setup of the Environment

This section contains all the information on the hardware, tools, frameworks, and libraries used to develop the project.

3.2.1 Hardware Units

During the development of this work, it was used three hardware units in total.

To implement the project and run the final product, it was used an MSI GL62M 7RD computer. The specifications of this computer can be seen in Table 1.

Table 1: MSI computer specifications.

Modules	Specifications
CPU	Intel Core i7-7700HQ 2.80GHz
RAM	8 GB
Graphics	GeForce GTX 1050 2GB
Operating System	Windows 10

The Tesla T4 GPU provided online by Google Colab was also used to train the detector. To train deep neural network models such as YOLO, which are computationally demanding, it is recommended to use GPUs due to their great computational performance, as they can process information in parallel, i.e. perform tasks and make multiple calculations at the same time, which makes the process faster compared to CPUs that perform tasks and calculations sequentially.

One of the initial objectives of this project was that the entire system developed (YOLO detector + Deep SORT tracker + infringements detection) could be run by the NVIDIA GPU Jetson TX2 from the Jetson TX2 Developer Kit, to later be tested in a real-world and real-time scenario. However, due to dependency problems in the installation of the Tensorflow framework, it was not possible to run the Deep SORT tracker on the board. Therefore, it was only possible to run the YOLO detector on this GPU.

3.2.1.1 NVIDIA Jetson TX2 Developer Kit

The NVIDIA Jetson TX2 Developer Kit gives access to develop and test software and hardware for a Jetson TX2 module. The Jetson TX2 module is an AI supercomputer very useful for products demanding high computational performance. The specifications of the Jetson TX2 module can be seen in Table 2.

Table 2: Jetson TX2 module specifications.

Modules	Specifications
GPU	256-core NVIDIA Pascal
RAM	8GB
Operating System	Ubuntu Linux 18.04.6

The developer kit consists of a carrier board with the Jetson TX2 module attached to it and some hardware interfaces for adaptable development and prototyping. Figure 3.3 shows the developer kit.

Figure 3.3: NVIDIA Jetson TX2 Developer Kit.

For this kit to work properly, it was required to install the NVIDIA Jetpack SDK. For this installation, it was necessary to connect the kit to a Linux host computer with the NVIDIA SDK Manager installed, which enables the software installation to the kit's module. The most relevant software features installed on the board and their versions can be seen in Table 3.

Table 3: Software features installed on the Jetson TX2 Developer Kit.

	Version	Description
JetPack	4.6.3	Package containing all the software tools needed to develop AI applications on the Jetson module.
CUDA	10.2.300	API that enables GPU parallel computing.
cuDNN	8.2.1.32	GPU-accelerated primitives library for deep neural networks.
OpenCV	4.1.1	Computer vision and machine learning open-source library.
Python	3.6.9	High-level programming language.

As previously mentioned, the entire system was developed on the MSI computer, thus the tools, frameworks, and libraries presented in the following sections are those installed and used on this computer.

3.2.2 Tools and Frameworks

Table 4 shows the required tools to develop the work, as well as the frameworks used.

Table 4: Tools and frameworks used to develop the project.

Tools	Frameworks
Roboflow	Darknet
Google Colab	Tensorflow
Google Drive	
Microsoft Visual Studio Code	

All the work of building the dataset and annotations was done using the Roboflow online platform [39]. To train the YOLO detector it was necessary to use Google Colab, which, by providing access to a GPU, made it possible to speed up the process of training the algorithm.

The files needed for running the training were saved on Google Drive, which provides a good amount of memory for data storage, which allowed to not overload the computer. It also allowed the storage of the training results directly in the cloud, which avoided losing these important data, as each session opened on Google Colab has a limited run time and when it ends, all files are lost.

For the integration of the YOLO detector and the Deep SORT tracker, as well as for all the writing of the developed program, the Microsoft Visual Studio Code editor was used.

The Darknet framework was used, which is an open-source framework to train neural networks written in C and which is the basis of the YOLO algorithm. To integrate the Deep SORT tracker with YOLO, it was necessary to convert the Darknet weights file generated at the end of the detector training to a Tensorflow model. Tensorflow is another open-source framework used for deep learning applications, which uses the Python language as a front-end API to develop applications.

3.2.3 Libraries

Table 5 shows all the libraries used to implement the project in Microsoft Visual Studio Code, as well as the versions used. The code was written using the Python language (version 3.10.11).

Table 5: Libraries used to implement the project.

Libraries	Version	Description
OpenCV2	4.7.0	Computer vision and machine learning open source library.
Numpy	1.23.5	Open-source Python library for scientific computing.
Matplotlib	3.7.1	Python library for graph plotting and data visualization.
Pillow	10.0.0	Python library that provides image processing tools.
Absl-py	1.4.0	Python library code collection for developing Python applications.
Easydict	1.10	Python package that allows accessing dictionary values as attributes.
Tqdm	4.65.0	Python library used to visualize the progress of a loop by adding progress bars.
Lxml	4.9.2	Python library that allows handling with HTML and XML files.

3.3 YOLO Detector

In order to detect vehicles and traffic lights, an object detection algorithm had to be used. The detector should be robust, accurate, and quick to process the information. To this end, the deep neural network YOLO was used.

It was chosen the version 4 of the YOLO algorithm (YOLOv4) because of the extensive information on it and the large support community, but mainly because it is the latest possible version compatible with Python's version of the Jetson TX2. The Python version of Jetson TX2 is 3.6.9 and the YOLO versions newer than the fourth require Python version 3.7.

3.3.1 Dataset

For a deep learning model like YOLO to be able to identify an object, it needs to know the most relevant features of that object, and, to do this, it needs to be provided with the necessary information about the object that it is aimed to be recognized.

The way to provide YOLO with this information is by creating a dataset. A dataset consists of a bank of images containing the objects we want to identify, which must be correctly labeled so the algorithm can know which object is of interest in the image. A good dataset is one that is as varied as possible, containing images of objects from various angles, sizes, and perspectives. The aim is for the set of images in a dataset to best represent how the objects will be found in a real-world scenario, which can sometimes be quite complex.

The larger the dataset, the better prepared and more accurate the detector becomes. One of the great advantages of deep learning models is that they can be trained with large datasets quickly. However, it's not just about quantity, quality is essential. The model will learn everything that is placed in the dataset, so it is very important that it is not provided with low-resolution images, as this will only confuse the algorithm more and lead to poor detection. Care must be taken not to have duplicate images, as this can lead to the algorithm being biased or overfitting, which produces poor results. Overfitting is when the model predicts very well on the training data, but the same does not occur on the validation set. This happens because the model memorizes the training data patterns, instead of learning them, which leads to not being able to generalize to unseen data.

Next, more detailed information on how the dataset was created and how the images were annotated is provided.

3.3.1.1 Data Collection

As this work serves as a starting point for the implementation of a system capable of detecting red light offenses to be placed and operated in the real world, it was necessary to make some decisions and start from somewhere.

It was therefore decided to collect images of various types of vehicles (cars, vans, buses, and trucks) and Portuguese traffic lights. Only images of basic three-color circular traffic lights were collected for this study, leaving out bicycle and bus traffic lights. The focus was on collecting images only in daylight, with good visibility (no fog or rain) and good lighting conditions. Images with occlusion were also included. As an initial study, it is important to start with the easiest cases and prove that this is a promising study, and only then extend it to more difficult cases. It was with this in mind that these decisions were made.

The images were taken from: the streets of Almada and Lisbon; Google Maps; Open Images Dataset v7 [40]; Roboflow public projects; and various public Internet sources. In total, the dataset contains 1398 images.

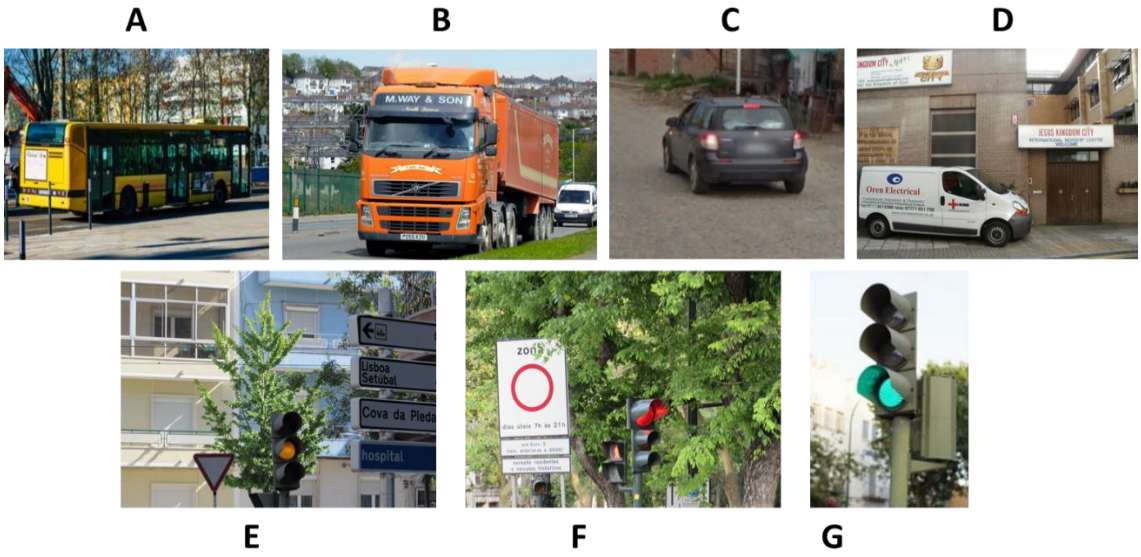

Figure 3.4: Examples of images from the dataset. A - bus; B - truck; C - car; D - van; E - yellow traffic light; F - red traffic light; G - green traffic light.

3.3.1.2 Data Organization

For the model to distinguish the various objects of interest, it is necessary to assign a class to each of them.

Initially, it was assigned seven different classes. Four were assigned to the different types of vehicles (car, van, bus, and truck), and the other 3 were allocated to the 3 possible states of a traffic light (green traffic light, yellow traffic light, and red traffic light). However, there were too many classes, which would only make the training process more time-consuming, and since there were not endless resources in Google Colab, it was not favorable for the training to take so many hours. Moreover, it was not necessary to differentiate between vehicles, because in a real-world scenario, the fine and the infringement are viewed in the same way for all vehicles, i.e. if a truck driver and a car driver pass on the red signal, the penalty is equal for both, and the infringement committed and the severity are the same for both cases. Therefore, the same class was assigned to all vehicles, thus becoming a dataset with 4 different classes (vehicle, green_light, yellow_light, and red_light).

3.3.1.3 Data Annotation

The algorithm is trained to learn the features of the objects of interest that are found in the provided images. However, it is not enough to show images of vehicles and traffic lights to the algorithm and expect it to know which object we are interested in to identify.

Before showing the images to the algorithm during the training, it is necessary to labeling the objects of interest in all the images, in this way, the algorithm will know which objects to identify and which ones to ignore. It is very important to label all occurrences of the objects of interest in the image because if any object is left unmarked, the algorithm will be confused and the detection's accuracy will be affected. This task was significantly time-consuming and tedious, as it was necessary to annotate objects manually from image to image.

An annotation consists of drawing a bounding box around the object. It is important that it is drawn as close as possible to the object, delineating its boundaries. Roboflow was used for this task. In Figure 3.5 can be seen examples of images with annotated objects.

Figure 3.5: Examples of images with annotated objects.

It is important to mention that the classes must be balanced. This means that the total number of occurrences of objects must be the same in all classes. If this balance does not exist, the algorithm may become biased for a given class, which leads it to detect very well one that has more occurrences and to not detect as well another that has fewer. In the 1398 images of the dataset, 2000 annotations were made in total, and these were divided by 4 classes equally, i.e. 500 annotations were made for each class, as can be seen in Figure 3.6.

Figure 3.6: Class balance.

3.3.2 Model Training

After having the entire dataset properly annotated, it was necessary to divide it into three sets: training set, validation set, and testing set.

A training set is used to train the model, which means that it has the function of teaching the model to recognize the features and patterns of objects. The validation set is used to evaluate the model during its training. At the end of each training iteration, the model is submitted to the validation set, to evaluate how it behaves with unseen data, and to check what needs to be improved in it and whether it needs to continue to be trained. The validation set also helps

to optimize the model's hyperparameters and is a way to prevent the occurrence of overfitting. The testing set is used to evaluate the model after the training is completed. This is the last evaluation, which helps to figure out whether the model is fit or not to be used in real scenarios. With this testing set, it can be verified how well the model detects objects in unseen data.

It is recommended that the division be done as follows: 70% of the total number of images in the dataset go to the training set, 20% go to the validation set and 10% to the testing set. Following this recommendation, out of the 1398 images in the dataset, 985 went to the training set, 303 went to the validation set, and 110 went to the testing set.

3.3.2.1 Conversion of the Labels to the YOLOv4 Format

YOLO's vision is different from human vision. Where the human eye sees bounding boxes drawn around the objects in the images, YOLO only sees the coordinates of these. Therefore, it was necessary to convert the labels that had been made to the proper format in order for the algorithm to interpret the information.

For each image file, a text file was created with the same name and in the same directory as the image files. This text file contains the information of all objects of interest present in the image. Each line corresponds to the information of a single object. The text must be in the following order:

$$< object\ class > < x_{center} >< y_{center} >< width >< height >$$

Where:

- $< object\ class >$ corresponds to the integer number assigned to the object class (this number can be between 0 and $(classes - 1)$);
- $< x_{center} >$ and $< y_{center} >$ correspond to the coordinates of the bounding box center point. These are relative values, which means that $< y_{center} > = \frac{absolute_y}{image_height}$ and can be between 0.0 and 1.0;
- $< width >$ and $< height >$ correspond to the width and the height of the image, respectively. These are relative values, meaning that, for example, $< width > = \frac{absolute_width}{image_width}$, and can be between 0.0 and 1.0.

An example of a text file with this information for five labeled vehicles can be seen in Figure 3.7.

Figure 3.7: Example of a text file with vehicles' label information.

Roboflow generates these files automatically, helping in the dataset preparation.

3.3.2.2 YOLOv4-tiny Weights

There are weights for the network convolution layers that have been previously trained. These weights help to speed up the training process, as the algorithm no longer starts from scratch and begins to be trained with some prior knowledge. In addition to saving time during training, these weights contribute to the algorithm becoming more accurate in the detection.

Instead of using the custom YOLOv4 weights file, it was chosen to use the YOLOv4-tiny weights file. The difference between the custom YOLOv4 and the YOLOv4-tiny is that the YOLOv4-tiny is a smaller version of the custom YOLOv4. The YOLOv4-tiny has a more simplified and reduced structure, also suffering a reduction in the number of parameters. It was developed with the aim of being more easily used in mobile and embedded devices. The use of YOLOv4-tiny allows training and detection to be much faster. This reduced model has only two YOLO layers instead of the three of the larger model and 29 convolution layers are pre-trained, instead of 137 as in the custom model. YOLOv4-tiny is much faster at detecting objects than the normal YOLOv4 but is less accurate. YOLOv4-tiny is the most recommended for real-time object detectors, as fast inference time turns out to be more important than accuracy.

It was with this in mind that it was chosen to use this model instead of the custom one, as it is crucial that a red signal infringement control system operates in real-time. The other reason for this choice was due to the shorter training execution time. As in Google Colab (where the model training was carried out) there were no infinite resources and there was a limited execution time, it was necessary that the training did not take too many hours, as would be the case with the custom YOLOv4 model.

3.3.2.3 Configuration File

Before starting the training of the algorithm, it was necessary to prepare a configuration file. In this file are defined the values of the parameters that will influence the performance of the training. These values are always chosen before training starts and are useful for model optimization.

The following are the values selected for the parameters with a brief explanation for each:

- Batch size = 64
- Subdivisions = 64
- Width = 608 and Height = 608
- Max. Batches = 8000
- Steps = 6400, 7200
- Classes = 4
- Filters = 27

The batch size corresponds to the images' number per iteration. The authors of the algorithm recommend setting a batch size of 64, because the higher this value is, the faster the algorithm is, and it prevents it from changing its weights all the time.

The subdivisions represent the number of segments the batch is divided into for GPU memory. The smaller the value of the subdivisions, the faster the training is. For this parameter, it is recommended to set a value of 16, but the lower this number is, the more memory is required from the GPU and therefore, the value of 64 was chosen to be compatible with the memory of the used GPU.

The width and the height represent the image size. The size must be a multiple of 32 and the standard size is 416x416. The algorithm can produce better results with the increase in image size, but it will affect the duration of the training, which will take more time. With this in mind, it was decided to set the image size to 608x608 to obtain more accurate results than those produced with the standard size but to not make the training too time-consuming.

The maximum batches are the number of iterations. It is recommended to have 2000 iterations per class, and this value must be at least 6000 and not smaller than the training images' total number. Therefore, the maximum batches were set to 8000 due to the training being for 4 classes (4*2000 = 8000).

The steps' values must be 80% and 90% of the maximum batches' value, respectively. These values are checkpoints where the multiplication between scales (coefficients) and the learning rate occurs. With the increase of iterations, the steps will determine how the learning

rate is changed during training. The learning rate is used to manage how slowly or quickly the model learns.

The classes represent the number of the different objects of interest. The training was set for 4 classes (vehicle, green traffic light, yellow traffic light, and red traffic light).

The filters represent the number of kernels in the convolutional layer that precedes each YOLO layer. The value of filters was defined through the following equation:

$$filters = (classes + number_coordinates + 1) \times 3 \quad (3.1)$$

Where:

- $classes$ correspond to the number of classes;
- $number_coordinates$ correspond to the total number of coordinates in the YOLOv4 format (width, height, x, and y).

Multiplication by 3 occurs due to YOLOv4 detecting three bounding boxes per grid cell.

3.4 Deep SORT Tracker

As mentioned earlier at the beginning of this chapter, to track the movement of vehicles and know where they were heading over time it was necessary to integrate a tracker into the object detector. For this purpose, it was chosen to use the Deep SORT algorithm.

Deep SORT receives frame-by-frame the YOLOv4 output with vehicle detections and uses this information to track them. For the integration of YOLOv4 detections with Deep SORT, a script found on Git Hub written by The AI Guy [41] was used. This integration uses the Tensorflow framework, consequently, it was necessary to run the script *save_model.py* to convert the Darknet weights file generated after training the detector into a Tensorflow model.

After that, by running the *object_tracker.py* script over a particular video, it was possible to track the objects detected by YOLOv4. However, it was necessary to make small changes in the code of this script to only track vehicles, because there was a problem when tracking the traffic lights. For example, when a red light changed to green, the tracker continued to indicate that it was the red light from the previous frame. In addition to this problem, there was no need to track the traffic lights, as they are static objects and remain in the same position throughout the video.

Before the changes were made, the script allowed to filter from the list of classes those that were supposed to be tracked. However, by excluding the classes related to the traffic

lights, the bounding boxes of these stopped appearing during the tracking. This occurred because the detections of the classes that were out of the list of classes to be tracked were destroyed. As the objective was to track only the vehicles, but continue to detect the traffic lights as well, a few changes were made in the Deep SORT code in order to obtain information on their state. These changes consisted of crossing the list of detections created for the current frame and checking whether these were of the vehicle class. If they were not vehicles, the respective bounding boxes of the detections were drawn in the image and their indexes were stored in an array. Then a new list was created with only the detections with indexes different from those found in the array. This new list containing only the saved vehicle detections was then sent to the tracker. In this way, the information from the traffic lights' detections was not lost and only the vehicles were tracked.

3.5 Red Traffic Light Infringement Control

The last step is the control of infringements in the red traffic light.

The aim here is for the algorithm to realize when a vehicle can or cannot overtake a traffic light. Using the information provided by the detector, the algorithm knows the traffic light state in each frame, and using the information from the tracker, which associates an ID for each detected vehicle, is able to follow each vehicle's trajectory over time. Therefore, if the vehicle does not respect the red traffic light, it will be identified by the algorithm.

3.5.1 Selection of Regions of Interest

The algorithm can detect vehicles coming towards the traffic light and also others going in the opposite direction. In both cases, the vehicles are overtaking the traffic light, but those coming in the opposite direction are not subject to the state of the traffic light facing the camera. It is, therefore, necessary to provide the algorithm with information on which traffic light it needs to monitor, thus ignoring the state of other traffic lights in the image, and also which areas in the image are associated with it. In order to do this, in the first frame, the algorithm asks the user to select the following three regions of interest (ROIs) from the image:

- *Preceding ROI*: This area corresponds to the area before the traffic light, i.e. the area where vehicles leave from.
- *Following ROI*: This area corresponds to the area after the traffic light, i.e., the area where vehicles are forbidden to move if the traffic light is red.

- *Traffic Light ROI*: This area corresponds to the region of the traffic light of interest, i.e. the region where the traffic light that the algorithm is responsible for monitoring is located.

An example of each of these areas in a given scene can be seen in Figure 3.8.

Figure 3.8: Example of selected ROIs in a given scene.

3.5.2 Support Functions

For the algorithm to be able to control whether vehicles are in the preceding ROI or the following ROI, two support functions had to be implemented to check whether the vehicles' bounding boxes intersect or are inside the ROIs. Another support function has been implemented to check whether the center of a particular detected traffic light is within the ROI selected as the traffic light of interest.

3.5.2.1 Corner Verification Function

This function called *isInside* checks whether there is any corner of the vehicle's bounding box within a certain ROI. The function takes as input parameters the coordinates of the vehicle's bounding box and the coordinates of the ROI.

A list is created with the 4 corners of the bounding box of the vehicle, in which each corner has a coordinate (x,y). This list is analyzed through a loop, where it is checked if any corner is within the boundaries of the ROI given as an input parameter. For this, two Boolean variables (*insideXAxis* and *insideYAxis*) are used, each of which serves to verify whether the x and y coordinates of the corner are within a range, respectively.

At the end, if both variables *insideXAxis* and *insideYAxis* have the value *True*, it means that this corner is inside the ROI and the function returns *True*, and there is no need to check any other corner because it is enough to have a corner within the ROI to conclude that the vehicle is inside that area. If any of the variables have the value *False*, it means that the corner is not within the ROI. If no corner is within the ROI, the function returns the *False* value, indicating that the vehicle is not within that area.

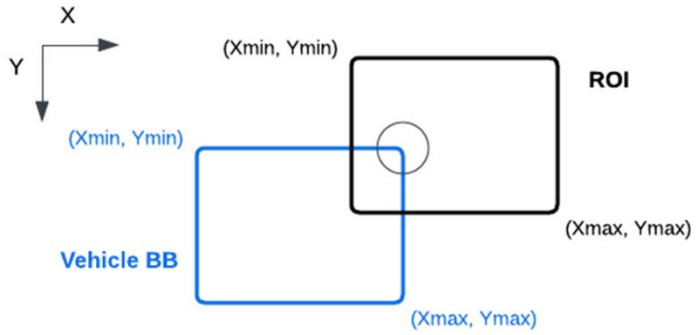

Figure 3.9: Example of a vehicle's BB corner inside the ROI.

3.5.2.2 Intersection Verification Function

In some cases, the bounding box of a vehicle may intersect one of the ROIs, but without having any of the corners inside them. Therefore, the intersects function was created to check if the bounding box crosses the ROI. This function takes as input parameters the coordinates of the vehicle's bounding box and the coordinates of the ROI.

It starts by checking the y-axis. It is checked whether the y_{min} and y_{max} of the ROI are within the range between the y_{min} and y_{max} of the bounding box. If they are, it is verified that at least one of the x_{min} and x_{max} points of the bounding box is within the range between the x_{min} and x_{max} points of the ROI. If this happens, the function returns the *True* value, indicating that the vehicle is in that area.

If the above conditions for the y-axis are not verified, a verification for the x-axis is performed. It is checked if the x_{min} and x_{max} of the ROI are within the range between the x_{min} and x_{max} of the bounding box. If they are, it is checked whether at least one of the y_{min} and y_{max} points of the bounding box is within the range between the y_{min} and y_{max} of the ROI. If this occurs, the function returns the *True* value, indicating that the vehicle is in that area.

If the above conditions are also not verified for the x-axis, the function returns the value *False*, which indicates that the vehicle is not in that area.

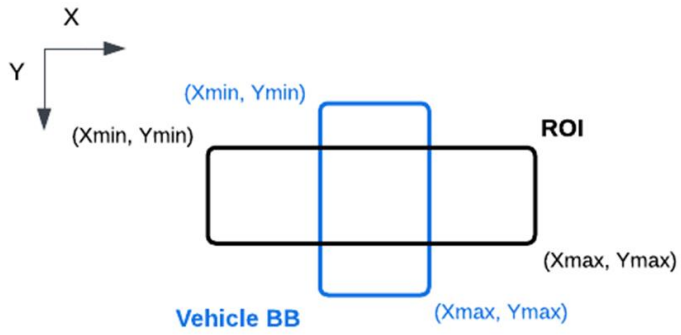

Figure 3.10: Example of a case where the vehicle's BB intersects the ROI without having any corner inside it.

3.5.2.3 Traffic Light Center Verification Function

This function, called *centerIsInside*, is used to check whether a particular traffic light detected in the image is the one that the user has indicated as the traffic light of interest.

The function will check whether the central point of the traffic light is within the ROI indicated by the user as the region where the traffic light to be monitored is located. The function receives as input parameters the coordinates of the bounding box of the detected traffic light and the coordinates of the ROI.

It starts by calculating the coordinates ($x_{central}$, $y_{central}$) of the traffic light's bounding box central point, using the following equations:

$$x_{central} = \frac{x_{min} + x_{max}}{2} \tag{3.2}$$

$$y_{central} = \frac{y_{min} + y_{max}}{2} \tag{3.3}$$

Where:

- $x_{central}$ e $y_{central}$ are the coordinates of the central point of the bounding box;
- x_{min} e x_{max} are the minimum and maximum points of the bounding box on the x-axis, respectively;
- y_{min} e y_{max} are the minimum and maximum points of the bounding box on the y-axis, respectively.

Two Boolean variables (*checkX* and *checkY*) are then created to check whether each coordinate of the bounding box's central point is within the range created by the ROI's minimum and maximum points.

At the end, the function returns these two variables, where if they both have the value *True*, it means that the traffic light detected is the traffic light of interest that the system is going to monitor. If one of the variables has the value *False*, or even both, it means that the traffic light detected is not the one to be monitored, as it is not within the selected area, then its state will be ignored.

3.5.3 Infringement Detection Method

Before developing the method for detecting vehicle infringements at red lights, it was necessary to understand under what circumstances a vehicle should be considered to have committed the infringement. It is considered that there is a red light infringement when the vehicle has moved from the preceding ROI to the following ROI or is in both at the same time when the traffic light is red.

Knowing that the tracker assigns an ID to each vehicle detected, two arrays were created to store the vehicle IDs. These arrays are as follows:

- *red_ids*: This array is used to store all the IDs of the vehicles that appear in the preceding ROI when the traffic light is red.
- *avoid_ids*: This array is used to store all the vehicle IDs that should be ignored. For example, it contains the IDs of those vehicles that have committed an infringement or have stopped between the preceding ROI and the following ROI and are also committing an infringement. By storing the IDs of these cases, the algorithm will ignore these IDs in the following frames and it will avoid identifying the same infringement more than once, as it only needs to be identified once.

Figure 3.11 shows the diagram that can summarize the operation of the infringement detection method developed.

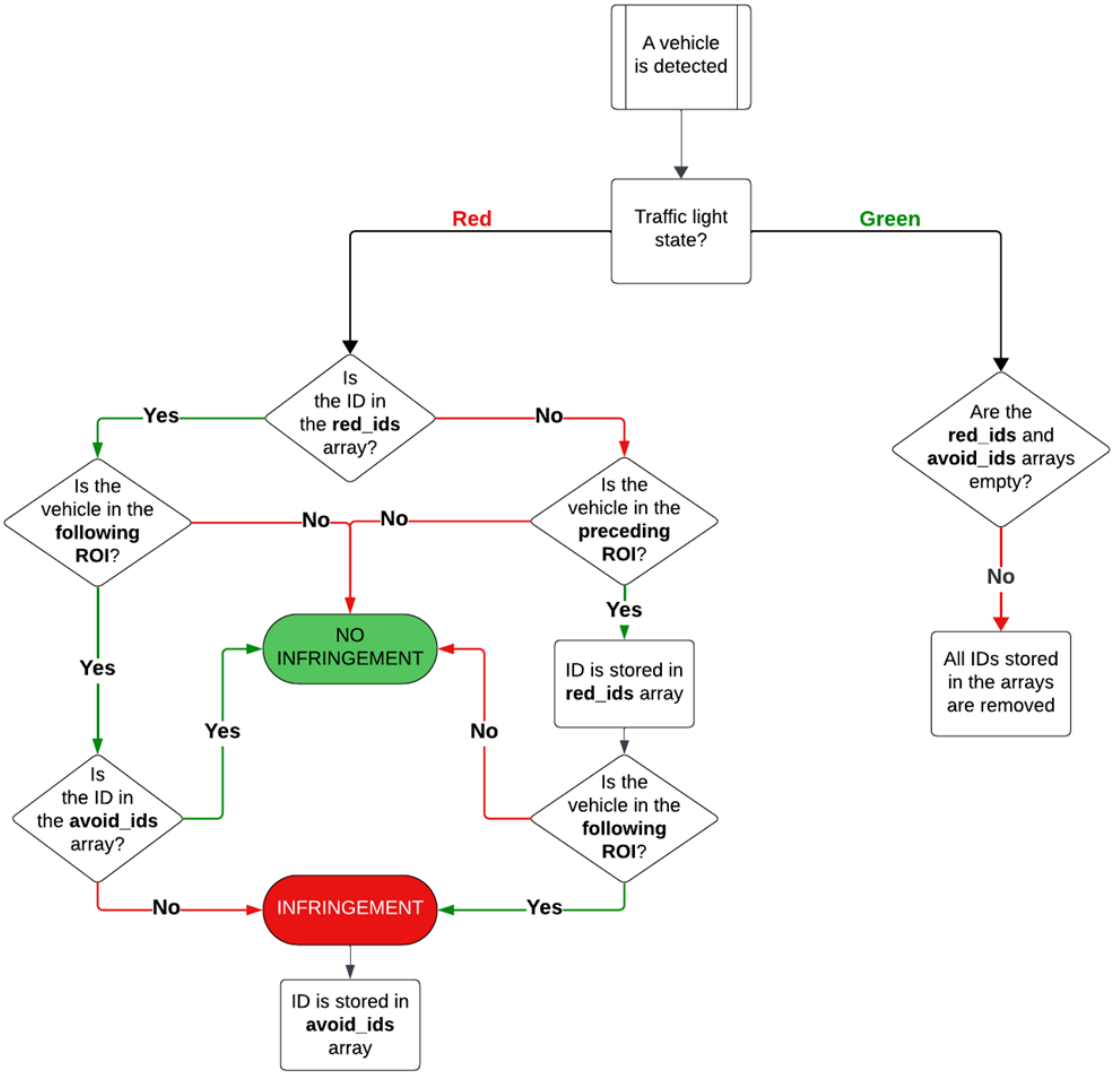

Figure 3.11: Flowchart of the infringement detection method.

All the IDs of the vehicles that are detected within the preceding ROI when the traffic light is red are stored in the *red_ids* array. This array is cleared whenever the traffic light state turns green.

During the time the traffic light is red, every time a vehicle is detected, it is checked that its ID is stored in the *red_ids* array. If it is and the vehicle is in the following ROI, it is checked whether its ID is stored in the *avoid_ids* array. If not, an infringement is detected and the ID of this vehicle is stored in the *avoid_ids* array.

If the vehicle ID is not stored in the *red_ids* array and the vehicle is in both preceding and following ROIs, an infringement is detected and the ID is stored in the *red_ids* and *avoid_ids* arrays.

4

RESULTS AND DISCUSSIONS

This chapter presents the results obtained from the previously explained work and all the discussions about these results.

4.1 Detector's Training Performance

Before presenting the results obtained, it is important to mention which metrics were used to evaluate the detector's training, how they are calculated, and what they mean.

4.1.1 Evaluation Metrics

Evaluation metrics are very useful when comparing different deep learning models and testing their performance in the real world. The selection of the evaluation metrics depends on the specific environment in which the algorithm is embedded and their suitability. These metrics can help understand how well the model is being trained and can serve as a guide for optimizing the algorithm's performance by adjusting the training parameters or even modifying the model's architecture.

4.1.1.1 Intersection over Union

Intersection over Union (IoU) measures how much the ground-truth bounding boxes and those predicted by the model overlap [42]. A ground-truth bounding box represents the correct place where an object is located in the image. IoU can help to understand how much a bounding box predicted by the model for a certain object is close to the correct one for that object. The values of IoU range between 0 and 1, where the value 1 means that there is a perfect match between the ground truth and predicted bounding boxes and the value 0 means that there is no intersection between them at all [42].

The IoU is calculated using the following equation:

$$IoU = \frac{area\ of\ overlap}{area\ of\ union} = \frac{BB_p \cap BB_{gt}}{BB_p \cup BB_{gt}} \tag{4.1}$$

Where BB_p is the predicted bounding box and BB_{gt} is the ground-truth bounding box. Figure 4.1 can help understand this better.

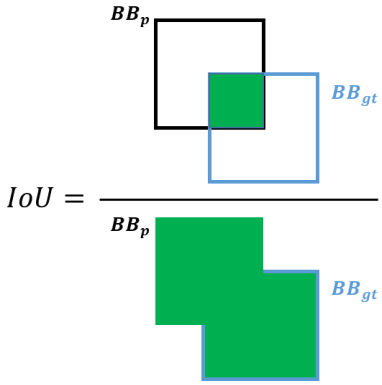

Figure 4.1: Illustration of the IoU calculation. Adapted from [42].

A threshold can be assigned to the IoU to make the other evaluation metrics more restrictive or less restrictive, i.e. they can be more restrictive when considering a correct detection. The closer the threshold is to 1, the more perfect the detection has to be to be considered a correct detection. The closer the threshold is to 0, the lower the requirement, i.e. the overlapping area resulting from the ground-truth and detected bounding boxes can be very small and the detection can be considered correct [42]. The IoU threshold commonly is expressed in a percentage, and normally it is used the threshold of 50%, but it can be used other thresholds, such as 75% [42].

4.1.1.2 Precision and Recall

Precision is an evaluation metric that represents the model's capability to correctly identify only pertinent objects [42]. Recall represents the model's capability of finding all the pertinent cases, i.e. ground-truth bounding boxes [42]. These two metrics can range between 0 and 1, and to calculate them, is required to classify the detected bounding boxes as:

- *True Positives* (TP): Ground-truth bounding boxes correctly detected.
- *False Positives* (FP): The existing objects that were inappropriately detected or the non-existing objects that were incorrectly detected.
- *False Negatives* (FN): Ground-truth bounding boxes that were not detected.

Then, precision and recall can be calculated using the following equations:

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

Normally, these two metrics are used together to evaluate a model by forming a plot called the precision-recall curve. The precision is on the y-axis and the recall is on the x-axis. The model is considered a good model if, as the recall increases, the precision remains high.

4.1.1.3 F1 Score

The F1 score is an evaluation metric that is closely related to precision and recall, representing the harmonic mean of these two metrics [42]. It can range between 0 and 1 and can be calculated using the following equation:

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

4.1.1.4 Average Precision

The three outputs of an object detector are the object class, the object's bounding box, and a confidence level. This confidence level can range between 0 and 1 and indicates how confident the model is with its prediction [42].

A confidence threshold can be defined and taken into account when calculating precision and recall. Only detections with a confidence level higher than the threshold are considered positive. A model is considered a good model when, for smaller confidence thresholds, the precision stays high as the recall increases [42].

The average precision (AP) is related to the precision-recall curve by representing the area under the curve. This metric represents how well the model identifies pertinent objects and ignores irrelevant ones throughout all the different confidence levels of the model's predictions [42].

Before calculating the AP, it is necessary to calculate the precision and recall for different confidence thresholds.

AP can then be defined by the following equation:

$$AP = \int_{r=0}^{1} p(r)dr = \sum_{k=0}^{K} (r(k) - r(k+1)) \times p(k) \tag{4.5}$$

Where $p(r)$ is the precision-recall curve, K is the number of thresholds, $(r(k) - r(k+1))$ is the difference in recall between the threshold values of k and $k+1$, and $p(k)$ is the precision at the threshold value of k.

4.1.1.5 Mean Average Precision

Mean average precision (mAP) is a metric based on the AP metric, which represents the overall detection performance for all classes of the dataset. AP is calculated for each class, and mAP is the average value of all the APs calculated for each class [42]. mAP can be calculated using the following equation:

$$mAP = \frac{1}{N} \sum_{j=1}^{N} AP_j \tag{4.6}$$

Where N is the total number of classes and AP_j is the AP value of the j-th class [42].

When calculating the mAP, an IoU threshold is assigned. This threshold can be a single value or a range. When only one value is assigned, the one usually assigned is 0.5 (mAP@0.5). mAP@0.5 is the mean average precision at 0.5 IoU threshold. When a range is assigned, the one most commonly assigned is 0.5 to 0.95 in increments of 0.05 (mAP@0.5:0.95).

mAP@0.5:0.95 is a more complete way of evaluating the object detector, because unlike mAP@0.5 which evaluates the detector for just one IoU threshold value, mAP@0.5:0.95 makes the evaluation more global considering the accuracy for several different IoU thresholds. The higher this value, the better the model's detection performance.

4.1.1.6 Loss Function

The purpose of using the loss function is to minimize the error between the predicted output and the ground-truth output. Calculating this error helps to adjust the weights of the neural network during the backpropagation procedure. It is important to achieve a low error loss value in the training data so that the model performs more accurately on unseen data.

In YOLOv4, the loss function is a combination of three losses:

- *Classification loss*: Associated with the object's classification into one of the predefined classes regarding the probabilities of that class.
- *Confidence loss*: Associated with the prediction of the object's existence or not in a grid cell.
- *Complete Intersection over Union (CIoU) loss*: Associated with how the predicted bounding box overlaps the ground-truth bounding box.

The final loss is the sum of these three losses.

4.1.2 Results Obtained

As previously mentioned, the YOLO detector was trained using Google Colab's Tesla T4 GPU. The training was carried out using the parameters with the values defined in Section 3.3.2.3 and took 4 hours and 21 minutes for the 8000 iterations to be concluded.

4.1.2.1 Training and Validation Sets Results

Figure 4.2 shows the graph obtained after training has been completed, with the values of mAP@0.5 (red) and average loss (blue). The mAP values are related to the validation data, and the average loss is related to the training data.

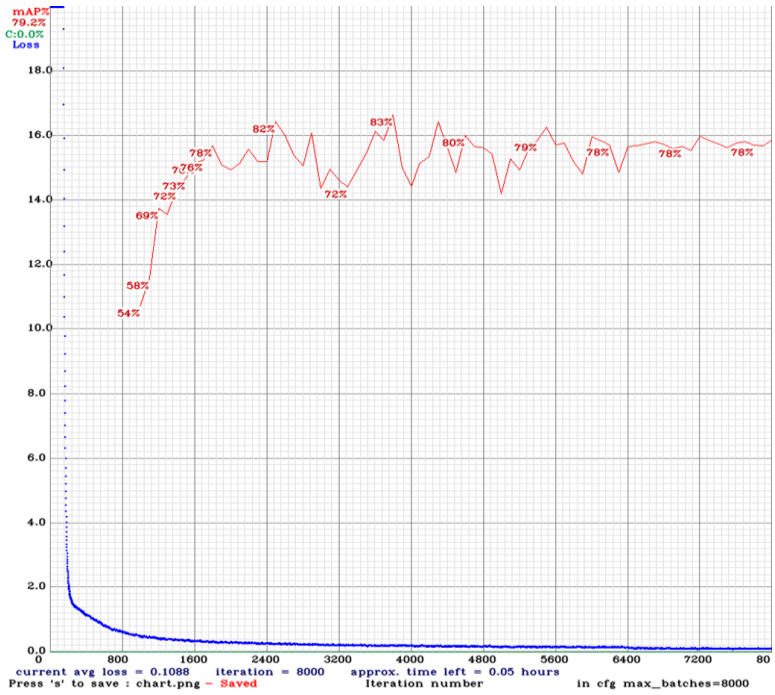

Figure 4.2: mAP@0.50 and average loss chart of the YOLO detector training.

It can be observed that at the end of the training session, the average loss was approximately 0.1, which is a good value, given that the recommended range for the final value is

between 0.05 and 0.3. The loss function decreased during training, getting closer and closer to 0, which is the expected behavior for a well-performed training session.

From the thousandth iteration onwards, mAP@0.5 began to be calculated. Its calculation was repeated every 100 iterations until the end of the training.

Calculating the mAP@0.5 values during training was important in deciding which trained weights to use for object detection. The weights with the highest mAP@0.5 value during training should be used. It can be seen from the graph that the highest mAP@0.5 value achieved was 83%, so the weights chosen for object detection were the ones that achieved this result.

The total number of objects present in the validation dataset was 421, which corresponds to the ground-truth value. For a confidence threshold of 0.25 and an IoU threshold of 0.5, it can be seen the values of precision, recall, F1-score, and the total numbers of true positives (TP), false positives (FP), and false negatives (FN) obtained in Table 6. The values of true positives, false positives, and the average precision obtained for each class can be seen in Table 7.

Table 6: Values of precision, recall, F1-score, and total of TP, FP, and FN (related to the validation set).

Precision	Recall	F1-score	Total TP	Total FP	Total FN
79%	76%	77%	318	84	103

Table 7: Values of TP, FP, and AP obtained for each class (related to the validation set).

Class (ID)	True Positives (TP)	False Positives (FP)	Average Precision (AP)
Green Light (0)	81	14	87.53%
Red Light (1)	81	22	88.11%
Vehicle (2)	69	32	67.37%
Yellow Light (3)	87	16	88.99%

Table 6 shows that the values obtained for precision and recall are quite identical and can be considered high. From the precision value, it can be seen that the detector has a convincing percentage of success in detecting only correct cases, indicating that it doesn't get too confused by identifying objects of no interest as if they were of interest. With regard to recall, it can be observed that the detector also has a convincing percentage of success in detecting all relevant cases, which indicates that it doesn't leave many objects of interest undetected.

The F1 score value can also be considered high and indicates a good balance between precision and recall.

In terms of the total number of true positives, false positives and false negatives counted, these can be considered positive results, as the number of TP is greater than the FP and FN, and the number of FN is greater than the FP. It's better to have a higher number of FN than FP than the other way around, because for a red light infringement detection system that serves to fine drivers who commit infringements, it's preferable for it to miss a few vehicles that have committed the offense, than identifying vehicles that haven't committed an infringement, as if they had committed it, and then being unfairly fined for it.

Table 7 shows that the results were generally quite good. The three classes related to the traffic light states obtained very similar results and achieved high APs. The vehicle class, on the other hand, achieved lower results than the others, but this is due to the fact that this class contains several different types of vehicles, with different shapes and sizes. This is not the case with the traffic lights' classes, where the traffic lights are very similar to each other, varying more in their pose, color, and level of obstruction. In short, it's easier for the algorithm to learn the traffic light pattern than the vehicle pattern, as this class contains many different vehicles and there are more patterns to learn.

For an IoU threshold of 0.5, the results were very positive, but to have a better understanding of the overall detector performance, it is important to obtain the results for other IoU thresholds. Then, it was calculated the mAP@0.5:0.95 which represents the average of mAPs obtained for ten different thresholds, ranging from 0.5 to 0.95 with increments of 0.05. The value obtained for mAP@0.5:0.95 was 42.6%.

It is normal for this value to decrease significantly compared to mAP@0.5, as it involves more demanding IoU thresholds, where the overlap level between the predicted and the ground-truth bounding boxes must be very high and almost perfect, in order for detections to be considered correct. The value achieved was not bad, but it was also not excellent. The main reason for this value to be lower is due to YOLOv4-tiny being used, which has a smaller architecture than the YOLOv4, reflecting a less accurate but faster performance. The two ways to raise this value would be to increase the dataset and re-train the model with it, or re-train it with the YOLOv4 pre-trained weights, instead of the YOLOv4-tiny ones, but this would require more resources to be done.

4.1.2.2 Testing Set Results

Having a testing set is not mandatory, but it is a good way to test the object detector with unseen data, which allows a better understanding of how the detector will behave in a real-world scenario. The testing set used contained 110 images, with 158 objects of interest in total (ground-truth value). A mAP@0.5 of 85% and a mAP@0.5:0.95 of 45% were obtained.

In Table 8 are the values obtained for the precision, recall, F1-score, and total numbers of TP, FP, and FN, and in Table 9 are the values of TP, FP, and AP for each class. All these values were obtained with a confidence threshold of 0.25 and an IoU threshold of 0.5.

Table 8: Values of precision, recall, F1-score, and total of TP, FP, and FN (related to the testing set).

Precision	Recall	F1-score	Total TP	Total FP	Total FN
78%	78%	78%	123	35	35

Table 9: Values of TP, FP, and AP obtained for each class (related to the testing set).

Class (ID)	True Positives (TP)	False Positives (FP)	Average Precision (AP)
Green Light (0)	30	7	93.19%
Red Light (1)	32	7	86.72%
Vehicle (2)	29	12	69.75%
Yellow Light (3)	32	9	89.62%

When comparing these results with those obtained for the validation set, it can be seen that they are very similar, however the results of the testing set are slightly better. A higher F1-score was obtained, indicating a higher balance between the precision and the recall, which were equal due to the same number of total FP and total FN.

The mAP@0.5 and mAP@0.5:0.95 were also higher than those obtained for the validation set, which is quite positive as it demonstrates that the detector has even had a slightly better performance with data never seen before, indicating that it has an acceptable performance in a real-world scenario. The reasons for these values not to be even higher and the ways to increase them are the same that were indicated for the validation set earlier.

4.2 System Performance On Videos

In order to find out how the detector and the tracker would perform in a real-world scenario, they had to be subjected to videos containing scenes of vehicles traveling through intersections with traffic lights. It's not an easy task to record videos of roads and intersections with all the traffic situations we wanted to test this work with because although there are many infringements committed, it takes luck to find one when recording a video outside. Nevertheless, we managed to film a few.

It was used 4 videos recorded by a drone to test this project. The video's resolution is 1920x1080. Each video scenario can be seen in Figure 4.3.

Figure 4.3: Scenario of each video used to test the system.

For each video, the real total number of objects was counted, which represents the ground-truth value. This value includes all the vehicles and the different traffic light states that appear during the entire video. The number of TP, FP and FN was also counted. Each video, although only a few seconds long, contains hundreds of frames and counting these numbers for each frame would be an endless job, so these values correspond to the number of objects in general in the video and not the total of all the frames, i.e. if a green traffic light is detected in all the frames, it is only counted once and not every time it appears in a frame. All these values can be found in Table 10.

Table 10: Ground-truth total of objects, duration, number of TP, FP, and FN registered in each video.

Videos	Duration	Total Objects	TP	FP	FN
1	11 s	13	9	0	4
2	20 s	27	17	4	10
3	30 s	19	17	0	2
4	30 s	26	21	3	5

It can be seen that the number of TP in each video was always higher than the number of FP and FN, which is a very positive result, meaning that the detector succeeded in detecting more objects of interest than the ones it missed or confused.

Most of the FN are vehicles that are very far from the camera and the traffic light of interest, meaning that the detector has difficulties detecting small objects. Regarding the traffic lights counted as FN, some of them are traffic lights that, due to their position, the light they emit is not well observed, which means that the detector does not realize their state, as can be seen in the examples in Figure 4.4. Others are traffic lights that are close together, two on the same pole. The detector often does not detect one of them or assumes that the two traffic lights are just one, as can be seen in the examples in Figure 4.5.

Figure 4.4: Examples of traffic lights where the light is not clearly discernible due to the brightness or position of the light.

Figure 4.5: Examples of traffic lights together on the same pole, where the detector gets confused.

As for the number of FP, fortunately, there were not many, most of which were confusions that the detector made with the color of the traffic lights. Sometimes it indicates that yellow traffic lights are red traffic lights or assigns both states simultaneously, as can be seen in the examples in Figure 4.6. This is because sometimes the yellow color appears darker or the red appears lighter due to the brightness of the street.

Figure 4.6: Examples of yellow traffic lights being mistaken for red traffic lights.

4.2.1 Evaluation Criteria

In order to assess the performances of the detector and the tracker during the entire 4 videos, evaluation criteria were assigned based on consistency and quality.

The detector and the tracker are evaluated based on their consistency, i.e. how well they detect and track the objects throughout all the frames in which they appear. The quality of the tracker is also assessed, i.e. whether it assigns the same ID to a particular vehicle over time, or whether it changes it. For consistency and quality, four levels were established for each.

In each video, the objects are analyzed one by one, and depending on the detector and tracker's performances in relation to the object, a level is assigned to that object. Once a level has been assigned to all the objects, the average level of each type of object is calculated, two types in total (vehicle and traffic light), in order to better understand how the detector and tracker perform with each of the different objects. In the case of the detection evaluation, the average value (mean average level) of the two average levels obtained is also calculated, in order to understand how the detector behaves in general, encompassing all objects. In the case of the tracker, this is not calculated, as only one type of object (vehicles) is tracked.

Each of the levels for the detection consistency, the tracking consistency and quality, and the calculation of the average levels and mean average levels are described below.

4.2.1.1 Levels for the Detection Consistency

The four levels for the detection consistency are the following:

- *Level 1 (Weak)*: The object is not detected.
- *Level 2 (Medium)*: The object detection oscillates throughout the video. Its bounding box makes a flashing effect.
- *Level 3 (High)*: If the object is a vehicle, it is detected in all frames, but is no longer detected when it moves away from the traffic light, or the other way round, if it is initially far from the traffic light and is not being detected, but as soon as it

approaches the traffic light it is detected in all frames. If the object is a traffic light, it is detected in practically every frame of the video, failing in very few, and the failure is almost imperceptible to the real-time view.

- *Level 4 (Very High)*: The object is detected in every frame.

4.2.1.2 Levels for the Tracking Consistency

The four levels assigned to the tracking consistency are:

- *Level 1 (Weak)*: The vehicle is not tracked.
- *Level 2 (Medium)*: The vehicle's tracking oscillates during the video. The bounding box flashes.
- *Level 3 (High)*: The vehicle is tracked in every frame but is no longer tracked when it moves away from the traffic light, or the other way round, if it is initially far from the traffic light and is not being tracked, but as soon as it approaches the traffic light it is tracked in every frame.
- *Level 4 (Very High)*: The vehicle is tracked in every frame.

4.2.1.3 Levels for the Tracking Quality

The four levels attributed to tracking quality are as follows:

- *Level 1 (Weak)*: The tracker changes the vehicle ID 3 times or more, or does not assign any ID to the vehicle during the entire video.
- *Level 2 (Medium)*: The tracker changes the vehicle ID twice.
- *Level 3 (High)*: The tracker changes the vehicle ID once.
- *Level 4 (Very High)*: The tracker assigns a unique ID to the vehicle throughout the video.

4.2.1.4 Average Level and Mean Average Level

The average level per type of object can be calculated using the following equation:

$$AL = \frac{1}{N}\sum_{i=1}^{4} i \times Q_i \tag{4.7}$$

Where N is the ground-truth total number of objects or, in relation to tracking quality, represents the total number of TP, i is the level number and Q_i is the number of objects obtained for that level.

The mean average level can be calculated using the following equation:

$$mAL = \frac{1}{M}\sum_{j=1}^{M} AL_j \tag{4.8}$$

Where M is the number of types of object and AL_j is the average level calculated for each type of object.

4.2.2 Detector's Performance Consistency

All results obtained for each video regarding the detection consistency, the average level of each type of object, and the mean average level, can be consulted in Table 11.

Table 11: Detection consistency obtained in each video.

		Levels					
Video	**Number of Objects**	**1**	**2**	**3**	**4**	**AL**	**mAL**
1	9 vehicles	4	4	1	0	1.7	2.3
	4 traffic lights	0	2	1	1	2.8	
2	17 vehicles	8	6	2	1	1.8	2.2
	10 traffic lights	2	4	0	4	2.6	
3	14 vehicles	2	3	2	7	3	2.7
	5 traffic lights	0	4	0	1	2.4	
4	14 vehicles	1	5	7	1	2.6	2.5
	12 traffic lights	4	4	0	4	2.3	

From the values obtained for the average level, it can be seen that in videos 1 and 2, the detector was more consistent in detecting traffic lights than vehicles, reaching a high average level for traffic lights and medium for vehicles. However, the results for videos 3 and 4 were the opposite, reaching a high average level for vehicles and medium for traffic lights. The reason for this is that videos 3 and 4 were recorded from a lower height than the other two videos. The detector detects vehicles better when the video is recorded closer to the ground and catches vehicles more from the side. This is due to the dataset being made up of more photographs of vehicles taken closer to the ground, containing very few images taken from above, leading the detector to not learn the characteristics of vehicles seen from above as well.

For these reasons, the values obtained for the mean average level in videos 3 and 4 were also better, reaching a high average level, and for videos 1 and 2, a medium level.

4.2.3 Tracker's Performance Consistency and Quality

The results obtained for each video regarding tracking consistency and tracking quality, as well as average levels, can be observed in Table 12 and Table 13, respectively.

Table 12: Tracking consistency obtained for each video.

		Levels				
Video	**Number of Vehicles**	**1**	**2**	**3**	**4**	**AL**
1	9 vehicles	5	3	1	0	1.6
2	17 vehicles	8	6	2	1	1.8
3	14 vehicles	2	5	3	4	2.6
4	14 vehicles	1	6	6	1	2.5

Table 13: Tracking quality obtained for each video.

		Levels				
Video	**TP**	**1**	**2**	**3**	**4**	**AL**
1	5 vehicles	2	1	2	0	2
2	9 vehicles	0	0	6	3	3.3
3	12 vehicles	2	1	6	3	2.8
4	13 vehicles	0	3	5	5	3.2

The results in Table 12 show the same situation as in Table 11: the average level for the vehicle tracking consistency in videos 3 and 4 was high and the average level for videos 1 and 2 was medium. It's normal for these results to be similar to those obtained for detection consistency, as the tracker receives the detections from the detector as input in order to track them. If the detections fluctuate a lot, their tracking will also fluctuate. The similarity between these results and the previous ones once again reinforces what was referred earlier about how the detector detects vehicles better.

The results in Table 13 show that the tracker achieved a high average level of tracking quality in videos 2, 3 and 4, and a medium average level in video 1. Video 1 achieved a lower value than the other videos because it was the video recorded at the highest height by the drone. As previously mentioned, the results are not as good when the vehicles are recorded from above, so in this case the tracker assigned several IDs to the same vehicles, as it assumed that they would change due to the large oscillations in the detections.

Another aspect that can be seen with these results is that, in the videos 2 and 4, which obtained the best results and are very similar, there are not many vehicles changing direction, and there are more going in the same one, while in videos 1 and 3, there are more vehicles changing direction.

When changing direction, the appearance of the vehicle changes, and as the tracker uses its appearance to track it, if the appearance changes too much over time, the tracker will have more difficulty in assigning the same ID to the vehicle, because it assumes that it is a different one. As a consequence, it assigns different IDs to the same vehicle.

To be absolutely sure of this, it would be necessary to submit the tracker to more tests, but these results may be an indication that this tracker does not work so well for cases where vehicles have to change direction, i.e. it may not be so recommended for scenarios in which is monitoring infringements and the traffic light of interest is relative to a change of direction.

4.2.4 Red Traffic Light Infringements Detection Performance

Regarding the detection of red traffic light infringements, only videos 1 and 2 contained infringements. Video 1 contained one and video 2 contained two, in total. Videos 3 and 4 did not contain any infringements. The number of infringements, and the number of TP, FP, and FN obtained for each video, can be seen in Table 14.

Table 14: Number of infringements, TP, FP, and FN obtained for each video.

Videos	Number of Infringements	TP	FP	FN
1	1	1	0	0
2	2	1	0	1
3	0	-	0	0
4	0	-	0	0

The results show that the system did not get any FP, which is a very positive result, because it's better not to detect a false infringement and let a real one through than the other way round. Of course, even better than this result would have been no FN.

As can be seen from Table 14, the system successfully detected the infringement in video 1 and only detected one of the two infringements in video 2, missing the other. This undetected infringement was due to fluctuations in the detection of the vehicle that committed it, and the tracker assigned several different IDs to the vehicle, so the system did not realize that the vehicle that ran the red light was the same one coming from the area before the light. As soon as

the vehicle committed the infringement, passing the traffic light, the tracker assigned it a different ID, and as this ID was not part of the IDs that came from the preceding ROI, the system did not count the infringement.

The four videos used to test the system only represent some of the real-life situations, and in order to test the system with more different situations, it would be necessary to subject it to more videos. However, through the problems encountered with detection and tracking previously discussed, the system's behavior can be predicted for some other situations. For example, in a case where the yellow traffic light is mistaken as red with a vehicle passing through the traffic light at that moment, it will generate a FP, because the system will consider that the vehicle has passed through the red traffic light, when in reality it was yellow. It is therefore very important to resolve this situation in a future work, as it will reduce the number of FP.

4.3 Processing Time

In order to understand whether the system is capable of operating in real time, it is necessary to evaluate its processing time. To do this, it is used a metric called FPS, which indicates how fast the model takes to process an image/frame. The FPS can be calculated as follows:

$$FPS = \frac{Total\ Frames}{Inference\ Time} = \frac{1}{T} \tag{4.9}$$

Where T represents the inference time in seconds of a unique frame.

The detector's processing time was tested using the four videos at three different resolutions: 480p, 720p, and 1080p. In addition, two different GPUs were used for this test: the NVIDIA Jetson TX2 and the Google Colab's Tesla T4. Table 15 and Table 16 show the frame rates obtained using the Jetson TX2 and Tesla T4 GPUs, respectively.

Table 15: Detector's frame rate obtained in each video at three different resolutions using the Jetson TX2 GPU.

	Resolutions			
Videos	**480p**	**720p**	**1080p**	
1	21 FPS	7.5 FPS	7.3 FPS	**Frame Rate**
2	21.5 FPS	18.1 FPS	7.9 FPS	
3	21.4 FPS	19.4 FPS	8.4 FPS	
4	21.6 FPS	9.2 FPS	9.1 FPS	

Table 16: Detector's frame rate obtained in each video at three different resolutions using the Tesla T4 GPU.

	Resolutions			
Videos	**480p**	**720p**	**1080p**	
1	76.8 FPS	15.3 FPS	13.7 FPS	**Frame Rate**
2	78.5 FPS	35 FPS	15.6 FPS	
3	76.7 FPS	38.2 FPS	18.3 FPS	
4	59 FPS	19.6 FPS	11.2 FPS	

The results obtained show that the detector is able to operate in real-time using these GPUs. It can also be seen that the lower the video resolution, the less time it takes to detect objects in the image.

From the results in Table 15, it can be seen that to get the best out of the Jetson TX2 for real-time applications, it is recommended to use it more for low or medium image resolutions, having achieved very good results for 480p and 720p resolutions, although it fluctuated a little for the 720p resolution when comparing the values of the four videos. For the 480p resolution, the maximum recorded was 21.6 FPS and for the 720p resolution, it was 19.4 FPS. In the highest resolution, 1080p, the maximum achieved was 9.1 FPS in video 4, which is a good value given this is considered a high-quality resolution. However, for situations in which vehicles are traveling at high speeds, it may not be enough for more accurate detection.

Table 16 shows better results than those obtained with the Jetson TX2 because the Tesla T4 GPU is more powerful. This indicates that it is possible to use the detector in real-time at higher resolutions when combined with a powerful GPU. The maximum frame rate achieved with 1080p resolution was 18.3 FPS, which is a very positive result.

Regarding the tracker, it was not possible to obtain its frame rate on the GPUs, due to problems related to TensorFlow installation dependencies that have not been possible to solve so far, as mentioned in the previous chapter. However, this was tested using the CPU of the MSI computer, having achieved a frame rate of 6 FPS for each video. This indicates that the CPU is not sufficient for real-time tracking.

5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Concerning the system developed, it is worth highlighting an architecture based exclusively on deep neural networks, integrating a detector with a tracker. The detector can recognize and classify vehicles and traffic lights in real-time, with a high level of accuracy (mAP@0.5 = 83%). The tracker can track the movement of vehicles over time and properly associate them with an ID. The mechanism for detecting red light infringements is based exclusively on computer vision. It does not require human intervention, showing great promise for implementation in the real world and helping security forces to ensure road safety.

During this work, it was realized that the dataset provided to the model to learn the features of the objects has a major impact on the final results. The more different images representing different real-world situations it contains, the more accurate the model's detection and, consequently, its tracking becomes. Of course, it takes a lot of work, time, and patience to create a good dataset, because it requires capturing lots of images of a wide variety of situations, and it is also required to annotate all the objects of interest in each image to train a model like YOLO. For these reasons, it is important to highlight the dataset created for this project, organized with 1398 annotated images of vehicles and traffic lights, ready to be used for further training and future studies.

By testing the detector using the GPUs, it was understood that, for real-time applications, it is recommended to use the Jetson TX2 GPU with lower resolutions, having achieved good and better frame rates (21-22 FPS).

In short, very positive results were obtained, proving this can be a promising study. The system still has a lot of room for progression, and by adding some improvements in future work, it could certainly be implemented in the real world.

5.2 Future Work

In future work, to solve some of the problems encountered and to reduce the number of false positives and false negatives, the dataset can be expanded by adding more images of small-sized vehicles (photographed at a bigger distance) and more images of vehicles taken from several different angles (mainly seen from above). It would also be important to add more images of yellow and red traffic lights to better distinguish between the two and add more images containing cases of traffic lights together on the same pole. Data augmentations can also be used to increase the dataset and make it more robust.

The detection can be extended to scenarios with different light and weather conditions (night, rain, fog, etc.), and adding the motorcycle class to the dataset would allow the detection of infringements committed by motorcycles.

It would be interesting for the system to also detect other kinds of traffic signs and lanes. It could be detected when a vehicle steps on a continuous lane or when it goes into the bus lane improperly. The system could also detect if a vehicle does not stop on a crosswalk with people waiting to cross.

Finally, it would be a great achievement to deploy the system on the Jetson TX2 or another better GPU and connect a camera to the board, to test the system live and in real-time. Whenever an infringement is detected, it could be sent to a server.

REFERENCES

[1] Z. Wang, J. Zhan, C. Duan, X. Guan, P. Lu, and K. Yang, "A Review of Vehicle Detection Techniques for Intelligent Vehicles," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 34, no. 8, pp. 3811–3831, 2022, doi: 10.1109/TNNLS.2021.3128968.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, 2001, doi: 10.1109/cvpr.2001.990517.

[3] S. Han, Y. Han, and H. Hahn, "Vehicle detection method using Haar-like feature on real time system," *World Acad. Sci. Eng. Technol.*, vol. 59, no. 11, pp. 455–459, 2009.

[4] D. G. Lowe, "Object recognition from local scale-invariant features," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2, pp. 1150–1157, 1999, doi: 10.1109/iccv.1999.790410.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[6] X. Chen and Q. Meng, "Vehicle detection from UAVs by using SIFT with implicit shape model," *Proc. - 2013 IEEE Int. Conf. Syst. Man, Cybern. SMC 2013*, pp. 3139–3144, 2013, doi: 10.1109/SMC.2013.535.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, 2008, doi: 10.1016/j.cviu.2007.09.014.

[8] J. W. Hsieh, L. C. Chen, and D. Y. Chen, "Symmetrical SURF and Its applications to vehicle detection and vehicle make and model recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 1, pp. 6–20, 2014, doi: 10.1109/TITS.2013.2294646.

[9] B. F. Momin and S. M. Kumbhare, "Vehicle detection in video surveillance system using Symmetrical SURF," *Proc. 2015 IEEE Int. Conf. Electr. Comput. Commun. Technol. ICECCT 2015*, pp. 1–4, 2015, doi: 10.1109/ICECCT.2015.7226055.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Proc. - 2005 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition, CVPR 2005*, vol. I, pp. 886–893, 2005, doi: 10.1109/CVPR.2005.177.

[11] X. Cao, C. Wu, P. Yan, and X. Li, "Linear SVM classification using boosting HOG features for vehicle detection in low-altitude airborne videos," *Proc. - Int. Conf. Image Process. ICIP*, pp. 2421–2424, 2011, doi: 10.1109/ICIP.2011.6116132.

[12] V. Vapnik, *The Nature of Statistical Learning Theory*, vol. 6, no. August. Springer New York, 1995. doi: https://doi.org/10.1007/978-1-4757-2440-0.

[13] Z. Chen, T. Ellis, and S. A. Velastin, "Vehicle type categorization: A comparison of classification schemes," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 74–79, 2011, doi: 10.1109/ITSC.2011.6083075.

[14] H. Farhood, X. He, W. Jia, M. Blumenstein, and H. Li, "Counting People Based on Linear, Weighted, and Local Random Forests," *DICTA 2017 - 2017 Int. Conf. Digit. Image*

Comput. Tech. Appl., vol. 2017-Decem, pp. 1–7, 2017, doi: 10.1109/DICTA.2017.8227421.
[15] Y. Freund, R. E. Schapire, and M. Hill, "Experiments with a New Boosting Algorithm Rooms f 2B-428 , 2A-424 g," 1996.
[16] A. Haselhoff and A. Kummert, "A vehicle detection system based on haar and triangle features," *IEEE Intell. Veh. Symp. Proc.*, pp. 261–266, 2009, doi: 10.1109/IVS.2009.5164288.
[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.
[18] Q. Fan, L. Brown, and J. Smith, "A closer look at Faster R-CNN for vehicle detection," *IEEE Intell. Veh. Symp. Proc.*, vol. 2016-Augus, no. Iv, pp. 124–129, 2016, doi: 10.1109/IVS.2016.7535375.
[19] J. E. Espinosa, S. A. Velastin, and J. W. Branch, "Vehicle detection using alex net and faster R-CNN deep learning models: A comparative study," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10645 LNCS, no. Dl, pp. 3–15, 2017, doi: 10.1007/978-3-319-70010-6_1.
[20] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.
[21] Z. Chen *et al.*, "Fast vehicle detection algorithm in traffic scene based on improved SSD," *Meas. J. Int. Meas. Confed.*, vol. 201, no. December 2021, p. 111655, 2022, doi: 10.1016/j.measurement.2022.111655.
[22] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017, [Online]. Available: http://arxiv.org/abs/1704.04861
[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.
[24] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni, "Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3," *2019 1st Int. Conf. Unmanned Veh. Syst. UVS 2019*, pp. 1–6, 2019, doi: 10.1109/UVS.2019.8658300.
[25] B. Xu, B. Wang, and Y. Gu, "Vehicle Detection in Aerial Images Using Modified YOLO," *Int. Conf. Commun. Technol. Proceedings, ICCT*, pp. 1669–1672, 2019, doi: 10.1109/ICCT46805.2019.8947049.
[26] Z. Rahman, A. M. Ami, and M. A. Ullah, "A Real-Time Wrong-Way Vehicle Detection Based on YOLO and Centroid Tracking," *2020 IEEE Reg. 10 Symp. TENSYMP 2020*, no. June, pp. 916–920, 2020, doi: 10.1109/TENSYMP50017.2020.9230463.
[27] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *In Pract.*, vol. 7, no. 1, pp. 1–16, 2006, doi: 10.1.1.117.6808.
[28] S. S. Teoh and T. Bräunl, "A Reliability Point and Kalman Filter-based Vehicle Tracking Technique," *Int. Conf. Intell. Syst. (ICIS 2012)*, vol. 1, no. 3, pp. 134–138, 2012, [Online]. Available: http://therevproject.com/publications/uwa/C2012-ICIS-A Reliability Point and Kalman Filter based Vehicle Tracking Technique-Teoh Braunl.pdf
[29] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2016-Augus, pp. 3464–3468, 2016, doi: 10.1109/ICIP.2016.7533003.
[30] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2017-Septe, pp. 3645–3649, 2018, doi: 10.1109/ICIP.2017.8296962.

[31] T. N. Doan and M. T. Truong, "Real-time vehicle detection and counting based on YOLO and DeepSORT," *Proc. - 2020 12th Int. Conf. Knowl. Syst. Eng. KSE 2020*, pp. 67–72, 2020, doi: 10.1109/KSE50997.2020.9287483.

[32] X. Hou, Y. Wang, and L. P. Chau, "Vehicle tracking using deep SORT with low confidence track filtering," *2019 16th IEEE Int. Conf. Adv. Video Signal Based Surveillance, AVSS 2019*, pp. 1–6, 2019, doi: 10.1109/AVSS.2019.8909903.

[33] Z. Ozcelik, C. Tastimur, M. Karakose, and E. Akin, "A vision based traffic light detection and recognition approach for intelligent vehicles," *2nd Int. Conf. Comput. Sci. Eng. UBMK 2017*, pp. 424–429, 2017, doi: 10.1109/UBMK.2017.8093430.

[34] M. Omachi and S. Omachi, "Traffic Light Detection with Color and Edge Information," *Proc. - 2009 2nd IEEE Int. Conf. Comput. Sci. Inf. Technol. ICCSIT 2009*, pp. 284–287, 2009, doi: 10.1109/ICCSIT.2009.5234518.

[35] R. De Charette and F. Nashashibi, "Traffic Light Recognition using Image Processing Compared to Learning Processes," *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, pp. 333–338, 2009, doi: 10.1109/IROS.2009.5353941.

[36] D. H. Widyantoro and K. I. Saputra, "Traffic lights detection and recognition based on color segmentation and circle hough transform," *Proc. 2015 Int. Conf. Data Softw. Eng. ICODSE 2015*, pp. 237–240, 2015, doi: 10.1109/ICODSE.2015.7437004.

[37] K. Behrendt, L. Novak, and R. Botros, "A Deep Learning Approach to Traffic Lights: Detection, Tracking, and Classification," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1370–1377, 2017, doi: 10.1109/ICRA.2017.7989163.

[38] R. Kulkarni, S. Dhavalikar, and S. Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning," *Proc. - 2018 4th Int. Conf. Comput. Commun. Control Autom. ICCUBEA 2018*, pp. 1–4, 2018, doi: 10.1109/ICCUBEA.2018.8697819.

[39] "Roboflow platform." https://roboflow.com/

[40] "Open Images Dataset v7." https://storage.googleapis.com/openimages/web/index.html

[41] T. A. Guy, "yolov4-deepsort." https://github.com/theAIGuysCode/yolov4-deepsort

[42] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. Da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electron.*, vol. 10, no. 3, pp. 1–28, 2021, doi: 10.3390/electronics10030279.