FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Environment Detection and Classification for Safety-Critical Railway Systems

Vasco Macedo da Costa

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Daniel Gouveia Costa

July 31, 2023

Resumo

Dada a busca por um aumento no uso dos transportes públicos na mobilidade urbana e a busca pela automação em várias tarefas realizadas por humanos, é certo que isso afetará todos os aspectos desse campo. Ferrovias e elétricos podem beneficiar da introdução desses aspectos de maneira a melhorar a confiabilidade e a segurança. Considerando que a automação e os sistemas de assistência a motorista nestes campos ainda podem ser aplicados à maioria das linhas ferroviárias e de elétricos existentes, e que esses sistemas dependem de diferentes tipos de dados para cumprir seu propósito, nesta tese clarificamos o possível uso de visão baseada em câmera como fonte de dados para a melhoria da perceção do seu ambiente, por parte dos sistemas de eléctricos. Concentramo-nos em conceitos como detecção, segmentação, classificação, semáforos de trânsito de elétricos/carros e transferencia de conhecimento. Devido à pesquisa limitada sobre implementações de elétrico deste tipo, esta tese funciona como uma prova de conceito para possíveis implementações destas tecnologias. Ao aplicar um modelo de *deep learning* amplamente utilizado para tarefa de detecção de semáforos de trânsito de elétricos, medimos a viabilidade desse tipo de aplicação e chegamos à conclusão de que definitivamente vale a pena aprofundar a pesquisa nestes tipos de implementações. Portanto, esta tese fornece ao leitor conhecimentos sobre: a criação de um *dataset* a partir de informações disponíveis publicamente; a aplicabilidade e viabilidade de *transfer learning* entre implementações diferentes; a afinação desses modelos com *transfer learning*.

Abstract

Given the increased demand for public transportation in urban mobility and the pursuit of automation across most human-performed tasks, such changes are sure to influence every aspect of this field. Both railways and trams can benefit from these developments to improve reliability and safety. Given that automation and driver assist systems in these fields can still be applied to the majority of existing railway and tram lines, and that these systems rely on different types of data to achieve their purpose, this dissertation offers insights into a possible usage of camera-based vision as a source of data for the improvement of tramway systems' awareness. This research focuses on concepts such as detection, segmentation, classification, tram/car traffic lights and transfer learning. Due to the limited research on tramway implementations of this type, this dissertation serves as a proof of concept for possible applications of these technologies. By applying a widely used deep learning convolutional model to the task of tram traffic light detection, this work gauges the feasibility of these applications, and reaches the conclusion that further research into these kinds of implementations is worthwhile. Thus, this dissertation provides the reader with knowledge on: the creation of a dataset from publicly available information; the applicability and feasibility of transfer learning between different implementations; and the tuning of these transfer learning models.

Keywords: Trams, Railway, Tram Traffic Lights, Classification, segmentation, Detection, Convolutional Neural Networks, Deep Learning, Neural Networks, Transfer Learning, Computer Vision, ADAS.

Acknowledgements

I would like to express my gratitude to the exceptional team at Continental for their warm reception and for creating an outstanding work environment. In particular, I would like to extend my sincere appreciation to António Pereira, Leonor Santos, and António Carreiro who accompanied me and whose support and presence throughout this project were instrumental in my professional growth, as they not only provided valuable guidance but also created a friendly and collaborative atmosphere. Their willingness to assist and offer constructive feedback was greatly appreciated.

I would also like to acknowledge the invaluable contribution of Professor Daniel Costa, who served as the supervisor for my thesis at FEUP (Faculty of Engineering, University of Porto). His expertise and guidance were invaluable in shaping the direction of this project. Professor Costa's availability and willingness to engage in discussions and provide valuable insights were immensely valued.

Furthermore, I would like to extend my heartfelt thanks to my fellow colleagues at "Toca do Menino" and "Grupo de Fãs" for their invaluable input on much-needed tomfoolery in my downtime. The same can be said about my close family, which was instrumental in my being able to finish this project whilst remaining sane.

Thank you all.

Vasco Costa

'(...)for example, a kid asks mommy why is the grass green and very often you get an answer, 'Don't ask dumb questions' or 'who knows.' (...) How much better would it be to say to the child 'that's a good question, I don't know the answer, maybe we can look it up'"

Carl Sagan

Contents

1 Introduction **1**
- 1.1 Problem definition . . . 1
  - 1.1.1 Objectives . . . 2
  - 1.1.2 Keywords and research questions . . . 2
  - 1.1.3 Document overview . . . 2

2 Theoretical concepts **4**
- 2.1 Computer vision, its' domains and approaches . . . 4
- 2.2 Traditional computer vision techniques . . . 5
  - 2.2.1 *Hough* transform . . . 5
  - 2.2.2 Scale-Invariant feature transform . . . 6
  - 2.2.3 *Haar* cascade . . . 6
  - 2.2.4 Histogram of oriented gradients . . . 6
  - 2.2.5 Traditional techniques overview . . . 7
- 2.3 Neural networks: components and working principles . . . 8
  - 2.3.1 Basic understanding . . . 8
  - 2.3.2 Components . . . 10
  - 2.3.3 Losses in neural networks . . . 11
  - 2.3.4 Back-propagation . . . 12
  - 2.3.5 Hyper-parameters and model tuning . . . 13
  - 2.3.6 Key performance indicators . . . 16
- 2.4 Convolutional neural networks . . . 19
  - 2.4.1 Architecture . . . 19
  - 2.4.2 Transfer learning in CNNs . . . 21
  - 2.4.3 Models for object detection . . . 22
- 2.5 Datasets . . . 26
  - 2.5.1 Data augmentation . . . 27

3 Bibliographical Review **29**
- 3.1 Car-centric implementations and their feasibility . . . 29
  - 3.1.1 Traffic sign and light detection . . . 29
- 3.2 Existing research on railways and public Transport . . . 31
  - 3.2.1 Rail maintenance and obstacle detection . . . 31
  - 3.2.2 Traffic sign and light classification and segmentation in Railroads . . . 33

4 YOLOv3 Based Detection and Classification **35**
- 4.1 The dataset . . . 35
  - 4.1.1 Choosing the dataset . . . 35

4.1.2 The environments in the dataset and German tram traffic lights 36
4.2 The model, transfer learning and pipelines 40
4.2.1 The original implementation: dataset and transfer learning 40
4.2.2 Traffic light recognition pipeline 43
4.3 Training and reasoning 48
4.3.1 Tunable hyper-parameters 49
4.3.2 Training iterations 51
4.4 *C++* package 58

5 Results and Discussion **60**
5.1 Discussion of results 60
5.1.1 Iteration 1: baseline. 60
5.1.2 Iteration 2: spatial augmentation. 64
5.1.3 Iteration 3: colour augmentation. 65
5.1.4 Iteration 4: the impact of rotation. 67
5.1.5 Iteration 5: changing learning rate. 68
5.1.6 Iteration 6: applying clustering. 68
5.1.7 Iteration 7: changing the learning rate. 70
5.1.8 Iteration 8: resolution increase. 70
5.2 Overall conclusions from training results. 71

6 Conclusion **73**
6.1 Future works 74

References **75**

A Extra Results **83**
A.1 Iteration 1 83
A.1.1 Training for 25 epochs 83
A.1.2 Training for 55 epochs 85
A.2 Iteration 2 87
A.3 Iteration 3 88
A.3.1 Training with HSV=[0.2;0.2;0.2] 88
A.3.2 Training with HSV=[0.3;0.3;0.3] 90
A.3.3 Training with HSV=[0.4;0.4;0.4] 93
A.3.4 Training with HSV=[0.1;0.678;0.6] 95
A.4 Iteration 4 97
A.5 Iteration 5 99
A.6 Iteration 6 101
A.6.1 Parameters 101
A.6.2 KPIs 102
A.7 Iteration 7 103
A.8 Iteration 8 105

List of Figures

2.1 Example of over-fitting on a given dataset . . . 9
2.2 Example of a fully connected network . . . 9
2.3 Analogy of a biological and artificial neuron and its components . . . 10
2.4 Representation of Intersection over Union . . . 12
2.5 Figure representing a local and global minimum . . . 13
2.6 Figure exemplifying impact of learning rate . . . 13
2.7 Image depicting loss values decreasing throughout the epochs of training . . . 15
2.8 Example of a confusion matrix, used for the calculation of KPIs . . . 16
2.9 Example of a fully convolutional neural network . . . 19
2.10 Representation of a CNNs' convolution . . . 20
2.11 Example of a max pooling layer with a size of 2 . . . 20
2.12 Image depicting different types of detectors . . . 22
2.13 Depiction of how YOLO works . . . 23
2.14 Depiction of how YOLO performs in the *COCO* . . . 24
2.15 Example of Non Max Suppression . . . 24
2.16 Depictions of the workings of R-CNN . . . 25
2.17 Training time and testing time of R-CNN and Fast R-CNN . . . 26
2.18 Image depicting data augmentation . . . 27

4.1 Tram traffic light signs according to German law . . . 37
4.2 Images depicting an urban, and a suburban environment of the dataset . . . 37
4.3 Different of labels used when creating the dataset . . . 38
4.4 Examples of the lack of detail in an image present in the dataset. . . . 39
4.5 Similarities between car and tram-designed traffic lights. . . . 41
4.6 Different weather scenarios of the LISA . . . 42
4.7 Overview of the main pipeline relevant to this project. . . . 44
4.8 The nomenclature of the images and labels created in this dataset. . . . 45
4.9 The format in which bounding boxes are saved as they are saved in a label file. . . . 45
4.10 Example of a *.json* file containing the details of a label from LabelMe . . . 46
4.11 The translated format that is accepted by the input pipeline and is saved in a text file. 46
4.12 The original format that is accepted by YOLO and is saved in a text file. . . . 48
4.13 Example of a command that runs the training script, with the necessary arguments. 49
4.14 Values of the learning rate throughout training. . . . 52
4.15 Values of the learning rate throughout transfer learning training. . . . 53
4.16 Examples of images fed to the network with spatial augmentations. . . . 54
4.17 Examples of images augmented in the HSV colour space. . . . 54
4.18 Comparison of the label on an image with rotation. . . . 55
4.19 Learning rate of this specific iteration of training. . . . 56

4.20 Clusters used to modify the configuration file of the YOLOv3 model. 57
4.21 Figure depicting values of exponentially decreasing learning rate throughout training. 57

5.1 Example of low and high interactivity scenarios 61
5.2 KPIs retrieved when training from scratch (iteration 1). 62
5.3 KPIs from training with transfer learning for 55 epochs (iteration 1). 63
5.4 KPIs resulting from enabling space data augmentation (iteration 2). 64
5.5 KPIs resulting comparing different values of colour augmentation (iteration 3). . 66
5.6 KPIs from training with rotation disabled in the hyper-parameters (iteration 4). . 67
5.7 KPI from training with a new learning rate, seen in Figure 4.19 (iteration 5). . . . 68
5.8 KPIs from training with model clustering modifications of our labels (iteration 6). 69
5.9 KPIs from training the model with learning rate modifications, seen in Figure 4.21 (iteration 7). 70
5.10 Depiction of KPIs during training with an increase in resolution (iteration 8). . . 71
5.11 Examples of edge cases present in the dataset. 72

List of Tables

2.1 Summary of traditional feature extraction techniques 8
2.2 Summary of hyper-parameters and their impact on the network. 15
2.3 Summary of Evaluation Metrics for CNNs 18

4.1 Table with values of the learning rate throughout training. 50
4.2 Table comparing all the training iterations done to the model, their changes and the reasoning behind them. 58

5.1 Table comparing the KPIs of transfer learning for 55 epochs with training the network from scratch (iteration 1). 63
5.2 Table comparing the KPIs of transfer learning for 55 epochs with the Key Performance Indicators (KPI)s from the impact of enabling spatial augmentations (iteration 2). 65
5.3 Comparison of the KPIs for the different colour augmentation parameters values (iteration 3). 66
5.4 Table comparing the best KPIs of iteration 3, with the ones resulting from disabling rotation in space data augmentation (iteration 4). 68
5.5 Table comparing the best KPIs from iteration 4, with new ones from training with the new learning rate of Figure 4.19 (iteration 5). 69
5.6 Table comparing the KPIs of the impact of modifying the model with *kmeans* with the iteration 5 (iteration 6). 69
5.7 Table comparing the KPIs of the impact of modifying learning rate from Figure 4.19 to Figure 4.21 (iteration 7). 70
5.8 Table comparing the KPIs of iteration 5, with those of training with a resolution of 1024p (iteration 8). 71

Abbreviations and Symbols

ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
AUC-ROC	Area Under The Receiver Operating Characteristics Curve
TSR	Traffic Sign Recognition
TLS	Traffic Light Segmentation
TLC	Traffic Light Classification
AP	Average Precision
GPS	Global Positioning System
CNN	Convolution Neural Networks
GIoU	Generalized Intersection over Union
IoU	Intersection over Union
ITS	Intelligent Transportation Systems
KPI	Key Performance Indicators
HOG	Histogram of Oriented Gradients
mAP	Mean Average Precision
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Networks
R-CNN	Region Based Convolutional Neural Network
RoI	Region of Interest
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SSD	Single-Shot Detector
SVM	Support Vector Machine

TLR	Traffic Light Recognition
TSD	Traffic Sign Detection
YOLO	You Only Look Once
FPS	Frames Per Second
RGB	Red;Green;Blue
BGR	Blue;Green;Red
HSV	Hue;Saturation;Value
LAB	Lightness;Green-Magenta;Blue-Yellow
YUV	Brightness;Blue Projection; Red Projection
GTSRB	German Traffic Sign Recognition Benchmark
GB	Gigabytes
LIDAR	Laser Imaging, Detection, and Ranging
ONNX	Open Neural Network Exchange
FE-SSD	Feature Enhanced Single Shot Detector

Chapter 1

Introduction

1.1 Problem definition

The introduction of public transportation in cities has been increasing steadily since the introduction of the European Union [1], as it strives for a low-carbon economy and lifestyle for its inhabitants. This has meant that even though car usage and ownership have increased since 2013 [2], providing users with options for transport has been at the core of the urban planning push inside the European Union [3].

As the emphasis shifts towards reducing and potentially eliminating emissions [3], the transportation sector (particularly personal transportation) emerges as ripe for a paradigm change in how society approaches both short and long-distance travel. This drive towards low-carbon mobility has positioned countries such as Austria, Singapore, and Japan [4] as leaders of urban mobility. The core of these public transportation systems comprises walking, cycling, and the use of subway/metro and rail networks, providing efficient and environmentally conscious ways to navigate cities.

When striving for more transportation options, **safety** must remain at the forefront of any implementation. Advanced Driver Assistance Systems (ADAS) are at the heart of these safety concerns and have been the focus of **extensive research** with the objective of ensuring safety throughout the path to achieving complete automation. Regarding safety in rail transport, the number of accidents in railways and trams has been dropping [2], as observed with most modes of transportation. Nevertheless, a considerable quantity still remains, particularly with fatal accidents, of which two-thirds involve rolling stock in motion [2].

One example of rolling stock subject to safety concerns is **trams**. Trams usually operate either in dedicated lanes or sharing lanes with other **active traffic participants**, such as bicycles, motor vehicles and people. Because of this, improving trams' or the drivers' awareness of their environments would be beneficial in reducing accidents. These accidents often result from the lack of due procedure in accordance with traffic laws, as one-third of fatal ones happen at crossways [2]. Given that traffic lights are one of the main ways of physically representing traffic laws, a vision-based system that detects these lights could help diminish the number of accidents if integrated

into decision-making systems. Despite regional railways and trams being a primary mode of land transportation [2], the intensity of research and development efforts devoted to obstacle detection for these systems needs to catch up to that of road transport [2]. Nonetheless, advances in car-centric algorithms have the potential to be transferable through additional improvements.

In conclusion, as our society moves towards a sustainable future [3], public transport will inevitably become widespread, with maintaining safety in their implementations being imperative.

1.1.1 Objectives

This dissertation aims at **developing a model for detecting tram traffic lights**, focusing specifically on five types of traffic lights outlined in German legislation [5]. By successfully detecting and classifying these tram traffic lights, valuable insights can be gained into developing a strategy that, in the future, could reduce the risks associated with tram public transportation.

It is important to highlight that this work was conducted within the framework of an internship at Continental Engineering Services in Porto, Portugal. Therefore, the development and implementation of this project adhered to the standards and practices set by the company. The collaboration with Continental Engineering Services provided an invaluable opportunity to leverage their expertise in the field.

1.1.2 Keywords and research questions

Within the context defined **above** and the **current state** of the usage of models to detect tram traffic lights, the main **research keywords** were **tram traffic light detection**, **classification**, **segmentation** and **deep learning**. With the main research questions derived from the aforementioned keywords, it is possible to raise the following questions:

- Is using an artificial intelligence model to detect tram traffic lights possible?
- Is there any available public model in the context of trams in urban areas?
- What are the main drawbacks when implementing such a model for the selected algorithms in the defined scenario?

1.1.3 Document overview

The document starts with Chapter 2, in which a comprehensive overview of the essential concepts necessary to grasp the work presented in this dissertation is provided. It delves into classical methods and continues on to explore Neural Networks (NN)s, Convolution Neural Networks (CNN)s and their parameters. This chapter also discusses the tuning of these parameters and highlights real-world implementations, providing insights into their functionality and overall effectiveness.

Sequentially, Chapter 3 offers a survey of the current state-of-the-art techniques and implementations in car-focused and rail-focused scenarios. It encompasses various applications, from railway object detection to sign recognition and rail maintenance.

In Chapter 4, the core of this dissertation is discussed, detailing the work undertaken throughout this project. This chapter encompasses a comprehensive account of the entire process, beginning with the task of dataset collection and labelling, supplying images and labels into a model and the training of said model with various parameter variations.

Chapter 5 focuses on the results obtained from the employed techniques and developed strategies, following the same sequence of events outlined in Chapter 4. Results and metrics are examined and conclusions are drawn to evaluate the extent to which the objectives were accomplished.

Lastly, Chapter 6 serves as the conclusion of this dissertation. In this chapter, the entire work is revisited, analyzing the results in the context of our initial objectives. Additionally, this chapter identifies potential areas of improvement and suggests ways for future exploration and improvement in specific aspects of this project.

Chapter 2

Theoretical concepts

In this chapter, the focus will be on the main concepts needed to fully understand the work being done in this thesis. Before delving into computer vision techniques, this chapter clarifies the main domains and distinguishes classical from Artificial Intelligence (AI)-based techniques. Then, it presents the concept of NN, its components and working principles, and optimization strategies. All of this can then be focused on CNNs, which will be fully understood as well, as these are a sub-category of NN, which usually focus on image analysis. Additional focus will be given to datasets and data augmentation, as they are crucial to any model using NN.

2.1 Computer vision, its' domains and approaches

Computer vision tries to derive information from visual data so as to interpret and understand said data. This can be done for multiple objectives, such as classifying and segmenting entities in images and videos. Using AI for object detection is a sub-field of **computer vision** that allows computers to perceive, analyze and comprehend visual data, with the ultimate goal of emulating human vision and detecting objects inside images [6], this supported by the authors work in demonstrating our ability to generalize enormous amounts of classification problems, and our robustness to interference. This technology is enabled by training on vast amounts of data and deriving correlations between said data points with greater dimensionality [6]. Sub-domains within computer vision include:

- **Object classification**, which is the task of assigning a label or class to an object in an image, focusing on identifying the presence of specific objects or categories within the image.
- **Object segmentation**, which involves localizing objects within an image or video.
- **Object detection** joins both segmentation and classification with the goal of giving both the position and class of an object.

Over time, advancements in camera technology, processing power, and model architectures have led to the development of different techniques for real-time object detection, with the emphasis driving the research in this topic being into **automating various human-based systems** [7].

The task of **object detection** can be done with techniques that are mainly divided into two broad categories[8]: classical and deep learning techniques. Those considered to be classical computer vision have had less attention in recent years, but perform well in lower variability and adaptability scenarios [9].

2.2 Traditional computer vision techniques

Here, techniques such as the *Hough* transform, Scale-Invariant Feature Transform (SIFT), Haar cascade, and Histogram of Oriented Gradients (HOG) will be explained briefly, as these techniques have been widely used for **object detection tasks**. Keeping in mind that these techniques exist among many others, those were chosen due to their prevalence in the research corpus and their comparison with deep learning, as in [8]. Their working principles will be discussed, as well as their capabilities, drawbacks and how they might be able to contribute and be applicable to more complex scenarios.

2.2.1 *Hough* transform

The *Hough* transform is a feature extraction technique used for **detecting simple shapes**, such as lines, circles, and ellipses in an image. It transforms the image into a parameter space where the parameters represent the properties of the shape to be detected. In the case of line detection, the parameter space consists of two dimensions: the line's slope and intercept. Each point in the image that belongs to an edge contributes to the parameter space by voting for all the lines that pass through it. The lines with the most votes are considered the detected lines.

Although the *Hough* transform has been widely used for detecting straight lines in images, it may not be effective for recognizing more complex objects, such as traffic lights and the lights inside them, as these, as a set, are more complex than the capabilities of said model. Still, it seems promising that, for the traffic lights for trams explored in this dissertation, it could be used to classify just the lights, as the ones for trams are simple lines on black backgrounds, in conjunction with a NN that narrows the scope by setting the given traffic light area. In [10], the authors manage to use a modified version of *Hough* Transform where the number of voting pixels is reduced in order to better the performance in curved lines. This is done with the goal of detecting lane markings, with their method matching or outperforming other state-of-the-art ones in detection rate but also outperforming "most of the state-of-the-art lane detection methods in computation time". Further research, like the one in [11] also applies *Hough* transform to detect lines in varied scenarios, from city skylines to shelves in supermarkets, going as far as to create a new dataset to test their technique. This technique consists of a using CNNs as a basis for a "deep *Hough* transform (DHT) that converts the deep representations from the spatial domain to the parametric domain", detecting those lines, and reversing the transformation, achieving detection on the spatial domain.

2.2.2 Scale-Invariant feature transform

The SIFT [12] is an algorithm that extracts and describes local features in images. SIFT operates in four main steps: first, scale-space *extrema* detection is performed by constructing a scale-space representation of the image using a series of blurred versions at different scales. Then, at each scale, the algorithm identifies local extrema as potential keypoints, which correspond to areas of high contrast and distinctive features; Third, object keypoints are refined by comparing to a threshold; Orientation assignment is performed consecutively to achieve invariance to image rotation, by assigning an orientation to each keypoint; Finally, a keypoint descriptor is created, by getting a histogram of the neighbours of a keypoint, where steps are taken to improve performance against changes in illumination. This technique is particularly **robust** to changes in scale, rotation, and illumination, making it suitable for matching and recognizing objects across different images. Despite its success in various object recognition tasks, due to its computational complexity, SIFT might not be the best choice for detecting traffic lights in **real-time applications**, such as tram Traffic Light Recognition (TLR). Furthermore, SIFT may not perform well in the presence of occlusions or when the object's appearance changes significantly, as it relies on matching local features. An implementation of such techniques is present in [13].

2.2.3 *Haar* cascade

Haar Cascade [14] is a machine learning-based object detection algorithm that focuses on detecting objects in images or video streams. Its name derives from the use of *Haar*-like features, which consist of rectangular features used to classify image regions. This algorithm works by training a cascade classifier with positive and negative samples of the target object. Positive samples represent instances of the object, while negative samples are regions without the object. During training, the algorithm learns to distinguish between positive and negative samples by evaluating different *Haar*-like features and their respective weights. The object detection process consists of sliding a detection window over the image and evaluating the cascade of classifiers at each position. This technique excels in speed, both in training and detection, and high robustness to changes in lighting, position and clutter [15]. While *Haar* cascade has been successfully applied to detect faces and other simple objects, it may not be suitable for detecting objects like traffic lights and the lights inside them, as it would be challenging to avoid edge cases that have similar features to traffic lights, detailed in Chapter 4. Additionally, it may not perform well in situations with highly varying illumination or occlusion due to the lower adaptability of the features learned in this algorithm. This technique is demonstrated in [15], where TLR is achieved in real-time and on 1080p images by using a similar technique that sequentially applies filters, creating key-regions, of which some turn out to be traffic lights that are then classified by a Support Vector Machine (SVM).

2.2.4 Histogram of oriented gradients

The HOG [16] is a feature descriptor used for object detection in computer vision. It involves computing the gradient magnitude and orientation at each pixel in the image and then dividing

the image into small cells. A histogram of gradient orientations is computed for each cell, and the histograms are concatenated to form the final descriptor. Detection using HOG might excel in detecting traffic light outer boxes and content that appears straight ahead with significant speed but, due to distortions of the image or variations in traffic light content, it may struggle with variations in position, such as in curves, and situations where the traffic lights differ in the details inside them. HOG has been widely used for detecting objects, such as pedestrians and vehicles, and could be adapted to recognize tram traffic lights. However, its performance may be affected by the presence of occlusions, changing lighting conditions, or significant changes in the object's appearance. In [17], HOG principles are applied by taking regions of interest and detecting the traffic signals inside them, followed by classification using an SVM. Results were promising, achieving high precision and recall but limited in adaptability to new data, as the dataset used was limited in variability in camera hardware and weather conditions.

2.2.5 Traditional techniques overview

Techniques like the ones just mentioned can be used **independently** or **merged** with others to increase the accuracy of object detection in a way that progressively filters bad results, this being visible in Table 2.1. Despite this, they all share similar disadvantages, all of which make them unfeasible to use without any combination with the techniques that will be explored in this chapter.

Technique	Advantages	Disadvantages	References
Hough Transform	- Effective for detecting simple shapes like lines, circles, and ellipses.	- Limited effectiveness in recognizing complex objects.	[10], [11]
SIFT	- Robust to changes in scale, rotation, and illumination.	- Computational complexity for real-time applications. - Performance degradation in the presence of occlusions and major appearance changes.	[13]
Haar Cascade	- Fast training and detection. - Robust to changes in lighting, position, and clutter.	- Difficulty in distinguishing edge cases similar to the target object. - Performance degradation in varying illumination or occlusion.	[15]
HOG	- Effective for detecting objects with significant speed and straight-ahead appearance.	- Difficulty in handling variations in position and detailed differences in the target object. - Performance degradation in the presence of occlusions, changing lighting conditions, or appearance changes.	[17]

Table 2.1: Summary of traditional feature extraction techniques

2.3 Neural networks: components and working principles

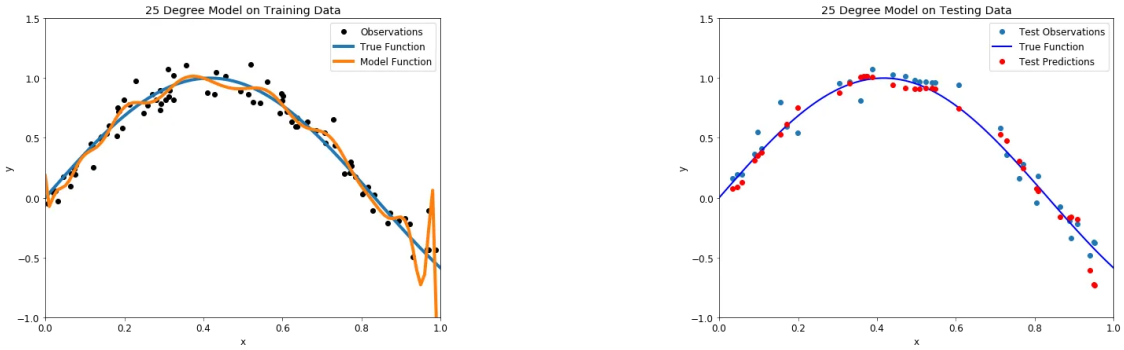
Having explored traditional techniques and knowing their advantages, disadvantages and use cases, deep learning is now explored in this section. NNs fall under the modern techniques category and, as such will be explored due to their prevalence in outperforming classical techniques[8].

2.3.1 Basic understanding

As a subset of machine learning algorithms, NN are used in many different applications, such as image recognition, natural language processing and predictive analytics, as they offer great analytical capacity. They have become widely used as their capabilities grew, in parallel with architectural improvements and computing advances. Artificial neurons are their base component and

these connect themselves in multiple interconnected layers to form NN, each serving a particular purpose in the network's learning process.

NN are highly dependent on data in terms of size, congruence with the desired situation and consistent labelling. This means that a model can become handicapped if the amount of data is insufficient, has inadequate labelling or if it has been trained in a biased way. Also, the opposite can also occur due to the same causes. The model may start **over-fitting** the data, as seen in Figure 2.1, making it so it finds patterns or relationships that exist but are not relevant or simply don't work to achieve our intended output, making it incapable of reaching said output in new situations.



(a) Over-fitting behaviour on train data

(b) Over-fitting behaviour on test data

Figure 2.1: Example of over-fitting, on a given dataset (adapted from [18]).

These NN are generally comprised of input, middle and output layers, depicted in Figure 2.2. The input layer receives the data that is to be analyzed as input, which is then processed by the hidden layers, where the gross of the computation occurs, **deriving features and relationships** between data points from altering features of these layers. It is usually said that the middle layers and the specific relationships they acquire when active are a "black-box", as we humans find it difficult comprehend the relationship between so many parameters and neurons, their dimensionality and also their non-linearity. This does not mean that there is no way to comprehend and manipulate the network, as several key performance indicators and hyper-parameters exist to let us understand its behaviour on a larger scale and push it in the right direction.

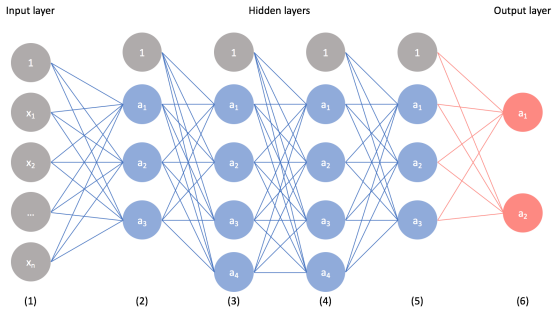


Figure 2.2: Example of a fully connected network (adapted from [19]).

2.3.2 Components

NN, at their core, rely on the concept of **neurons**, which are inspired by the biological neurons in the human brain. These artificial neurons play a crucial role in the learning processes of NN. A neuron receives inputs from other artificial neurons or input data. It then computes a weighted sum of these inputs, considering the strength of the connections between neurons. Additionally, each neuron has a bias value, which introduces an offset and impacts when the neuron activates. The combination of the weighted sum and the bias value forms the neuron's output. After undergoing an activation function, the output of a neuron is then passed on to the next layer of neurons in the network. The **activation function** introduces non-linearity into the network, enabling it to learn a broader range of patterns from the input data. Common activation functions include the *sigmoid* function and the *ReLU* [20] function or variations thereof.

In more advanced NN architectures, additional components are often considered, enabling NNs to effectively process and learn from complex datasets. Two such components are dropout and weight decay. **Dropout** refers to the probability of a neuron being disabled during the training process of the network. This technique has been shown to improve network performance, as demonstrated in studies such as [21] and [22]. **Weight decay**, on the other hand, involves the introduction of additional factors that influence the weights of the connections between neurons. This regularization technique helps prevent over-fitting and can contribute to better network performance [23], in which the authors "empirically demonstrate the effectiveness of AdaDecay", an adaptive weight decay technique, "in comparison to the state-of-the-art optimization algorithms using popular benchmark datasets: MNIST, Fashion-MNIST, and CIFAR-10". Most of these aspects can be seen and are represented in Figure 2.3.

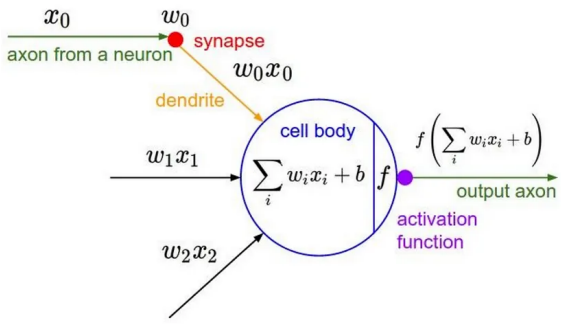


Figure 2.3: Analogy of a biological and artificial neuron and its components (adapted from [24]).

The neurons that are in Figure 2.3 combine with other neurons to form **input**, **hidden** and **output** layers. Depending on the task at hand, the output form can appear in many forms, from simple classes (as confidence scores of each class) to bounding boxes for objects or value prediction. In this way representing relationships between data points, managing to accurately **classify**, **describe** and **identify** objects and patterns (as will be seen in Chapter 3) and apply them to new situations. This is done by the forward propagation process (also known as **inference**), which can be followed by back-propagation, which will be explained in the following sections.

2.3.3 Losses in neural networks

Loss functions play a crucial role in training NN by quantifying the discrepancy between predicted outputs and the ground truth. This section will delve into the concept of losses, exploring what they are, how they function, and their purpose in the training process. Additionally, the focus will be on the usage of class, object, and generalized Intersection over Union (IoU) for losses, which are commonly employed in object detection tasks.

At its core, a loss function measures the **dissimilarity** between the predicted output of a NN and the ground truth or desired output. It assigns a numerical value, referred to as the loss, based on how well the network's predictions align with the ground truth. By minimizing this, the network aims to improve its predictive capabilities and optimize itself by changing its' parameters. In the context of object detection, where the goal is to identify and localize objects within an image, different types of losses are employed. These losses capture various aspects of the detection task and guide the network to learn appropriate representations classification and segmentation areas. To achieve this, class loss, object loss, and Generalized Intersection over Union (GIoU) loss are commonly utilized.

Class loss deals with the accurate classification of objects by penalizing incorrect class predictions. It is typically computed using a classification loss function like categorical cross-entropy [25]. This loss measures the discrepancy between the predicted class probabilities and the true class labels. By optimizing the class loss, the network learns to assign the correct class labels to detected objects, enhancing its ability to recognize different object categories.

Object loss deals with the localization accuracy of objects within the image. It quantifies the disparity between the predicted bounding box coordinates (such as the coordinates of the top-left and bottom-right corners) and the ground truth box coordinates. Regression-based loss function, like Mean Squared Error (MSE), are commly employed to calculate the object loss. The network aims to improve its ability to precisely locate objects by minimizing this loss.

GIoU loss is a popular choice for evaluating the localization performance of object detection models. It utilizes the concept of IoU (Equation Equation 1), which measures the overlap between predicted and ground truth bounding boxes, depicted in Figure 2.4. The GIoU loss builds upon IoU by incorporating additional terms that consider the enclosing regions of the boxes. This loss function encourages tight bounding box predictions that better align with the ground truth and penalizes imprecise or incomplete localization.

$$IoU = \frac{Area\_of\_Intersection}{Area\_of\_Union} \qquad \text{(Equation 1)}$$

In summary, losses are fundamental components of NN that quantify the discrepancy between predicted outputs and desired targets. They guide the training process by measuring how well the network is performing and allowing for optimization through Stochastic Gradient Descent (SGD) [27] or other techniques. By utilizing these different types of losses in an object detection framework, NN can effectively learn to classify and segment objects in images. The combination of

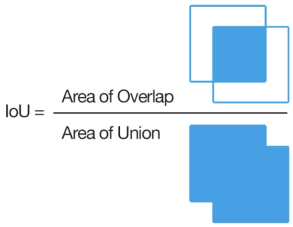


Figure 2.4: Representation of Intersection over Union (adapted from [26]).

class loss, object loss, and GIoU loss enables the network to simultaneously optimize for accurate **classification**, precise object **segmentation**, and improved bounding box **predictions**. By minimizing these losses, the network learns to make more accurate and reliable detections, contributing to the overall success of the object detection system.

2.3.4 Back-propagation

Regardless of the type of layer, activation function or architecture, a NN will always use back-propagation in its learning process. This resembles a **negative feedback loop**, in which after the model tries to output the answer to our problem, the training algorithm goes back and, according to the losses calculated, **adjusts the weights and biases**, sequentially in a backwards order through the layers. Thus, the goal is to reduce the error in the output and converge with the combination of all the model parameters that give us a solution, that is, the best performance.

Usually, this tuning happens in the form of computing the gradient of the loss function throughout the different layers from the last to the first one. The most common forms of computing this gradient are based in SGD, with derivatives adding memory optimization (mini-batch) or momentum. These can have a positive impact [28], where the assumption of "smaller batch sizes yield lower loss" is backed by the results. All of their basic working principles are represented in Figure 2.5, in which back-propagation enables us to use an iterative process to slowly overcome the local minimum of optimization and arrive at the so-called **global minimum**. This calculation is done by the chain rule principle, which allows the algorithm to continuously derive the contributions of each neuron to the error at the end. These methods have their drawbacks, such as getting stuck at local minimums and high memory and compute cost, and can be replaced by others, depending on our use-case. A comparison between algorithms [29] was done where multiple ones are applied to different datasets and tasks (CNNs and vision being one of them), and it is found that ADAM [30] seems to be the best and more versatile.

Another critical factor affecting the back-propagation process is the **learning rate**. This determines the step size taken during gradient descent. Controlling it determines how quickly or slowly a model converges towards the optimal weights, that is, the global minimum of optimization, as represented in Figure 2.6. A lower learning rate may require more iterations to reach the optimum solution but may offer **greater precision**. Contrarily, a higher learning rate can speed up the process but risks **overshooting** the optimal solution, scaling the weight updates.

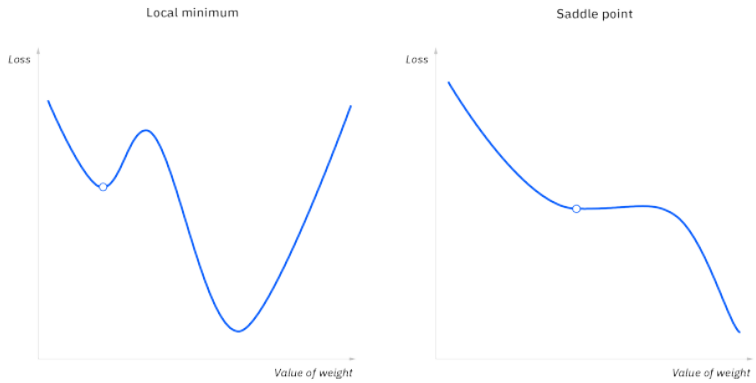


Figure 2.5: Figure representing a local (left) and global (right) (adapted from [31]).

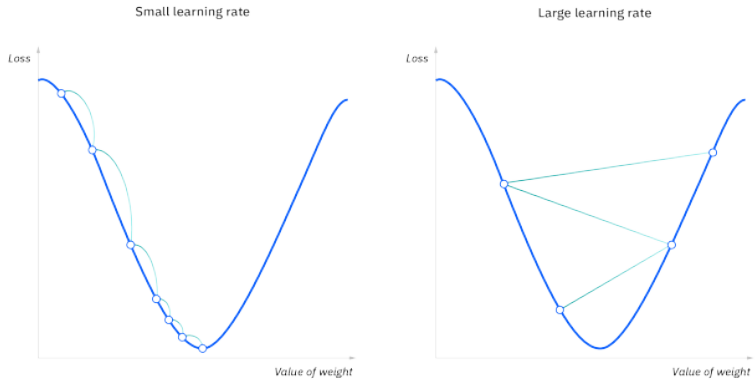


Figure 2.6: Figure exemplifying impact of learning rate. Smaller learning rate on the left and higher learning rate on the right, adapted from [31].

The whole process of back-propagation happens after the calculation of the loss values, upon which the weights update in accordance with our chosen method, *e.g.* SGD, with the ultimate goal of **minimizing** said loss value.

2.3.5 Hyper-parameters and model tuning

Hyper-parameters are parameters that have an **impact in training** of NN. As they are set before training and, although there are best practices regarding their values in certain models, they always require some fine-tuning by changing their values, retraining the network and comparing performance indicators. When fine-tuning these parameters, one has to analyze the loss function, the error and the KPIs, as well as their progression through training and alter their values. Here, the focus will be on **learning rate**, **batch-size**, **epochs**, **regularization** and **network architecture**.

The **learning rate**, as seen in 2.5, determines how heavily the model adjusts its weights in response to errors during training. This can be adjusted before or in the middle of training. By tuning the learning rate, a balance between training speed and accuracy can be achieved. Changing

the learning rate throughout the training can also be done, as this allows the network to first generalize the more important features more broadly, such as the shape of traffic lights or the contrast with the background, and find a set of features that correspond to a global minimum. Sequentially, a lower learning rate stops it from losing the previously gained knowledge.

Batch-size also impacts performance, both in training and in final performance. This parameter determines the **number of samples** that are processed in each training iteration. Using a smaller batch size (also called mini-batch) may lead to faster convergence due to more frequent weight updates, but it also introduces noise in the gradient estimates. This noise can sometimes be beneficial, as it helps the model escape local *minima*. On the other hand, a larger batch size provides more accurate gradient estimates by averaging over a larger number of samples, which can lead to more stable and smooth convergence. Using a mini-batch is more dispersed in the research landscape, as it tends to fit better into classification scenarios, as seen in [32]. Available memory is also a significant consideration when choosing a batch size, as they usually have to fit entirely inside the available memory.

An epoch is a complete pass of the entire training dataset through the network. So, adjusting the **number of epochs** will have an impact on the NN's performance. Increasing the number of epochs may allow the model to learn more complex relationships in the data, but it may also lead to over-fitting.

Regularization coefficients are hyper-parameters that are used to prevent over-fitting. It involves adding an additional term to the loss function during training, typically in the form of a **penalty** on the model's weights or parameters. L1 regularization (which encourages **sparsity** in the weights) and L2 regularization (which encourages **smaller** weights overall) are two common techniques that can be used to penalize large weights in the network, encouraging the model to find simpler solutions that generalize better. Dropout, which was also already mentioned, can positively affect the network, preventing over-fitting, by setting nodes to zero at each iterating of training, as well as the chosen optimizer, wich will also affect the performance of the network, as it defines how the weights of the network are calculated and changed.

Finally, the **network architecture** can be tuned to improve the performance of a NN. This includes the number of layers, the number of neurons per layer, and the type of activation function used, as well as other variations, like skip connections from [33], which tend to improve performance and are especially important for object detection purposes. A more complex architecture may be able to learn more complex relationships in the data. The architecture of a network also has to be adjusted to the specific application as, taking as an example this dissertation's objective, one should not choose an architecture which cannot detect in real-time. This means changing the number of parameters, layers, neurons and even other techniques like introducing different convolutional layers that reduce computational cost without much loss in performance, as in [34], or compression techniques like in [35] and [36].

An overview of these hyper-parameters and how they affect the network is found at Table 2.2. All the parameters contribute, in their own way, to network's performance. This density of parameters and combinations must be accompanied by informed reasoning about the impact of

each one to ensure that the model has not just found a local minimum of optimization and not the global one.

Parameter	Definition	Impact
Learning Rate	Determines the magnitude of change of each parameter at each training iteration.	Requires balancing high and low values throughout training.
Batch Size	Number of samples processed in each training iteration.	Requires a balance between speed, hardware capability, and training time.
Number of Epochs	Number of times the dataset is completely used for training.	Requires determining the correct number to avoid over-fitting the dataset.
Regularization	Additional changes to how weights are updated.	L1, L2 and dropout might prevent over-fitting.

Table 2.2: Summary of hyper-parameters and their impact on the network.

Finding the exact combination of hyper-parameters to get the maximum performance from the network is virtually impossible due to finite resources, but this doesn't mean users are completely blindsided when finding a close enough combination. Metrics, like the ones in 2.3.6, can be extracted from the network that point us in the general direction of optimization, as well as previously explored changes by other researchers. Both need caution to ensure they are interpreted correctly. This is demonstrated in both [37] and [38], with the former focusing more on architectural features (filter size, input image size and the number of neurons and layers) and the latter on learning rate, batch normalization and initialization methods. Optimizing these parameters will lead to better performance and lower loss values in scenarios like the one in Figure 2.7.

These optimizations can be achieved by either manual tuning, which can become overwhelming depending on the number of possible combinations of hyper-parameters, or automatically, where one programs an algorithm to push the network (by tuning hyper-parameters) to better performance, as is demonstrated in [39], where auto-tuning agents were used to adjust parameters (one per hyper-parameter, independently of each other) to improve accuracy in the *MNIST*[40] dataset.

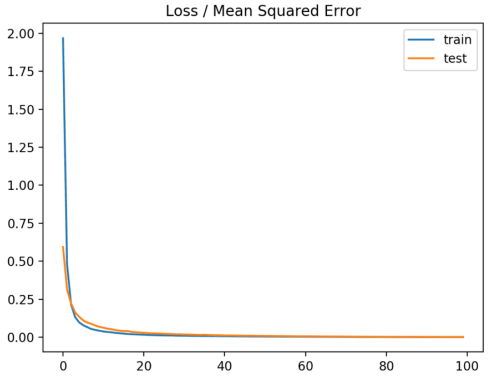


Figure 2.7: Image depicting loss values decreasing throughout the epochs of training (adapted from [41]).

2.3.6 Key performance indicators

Given the background knowledge given on NNs, ranging from architecture to components and optimization, it is crucial to discuss the KPIs used to evaluate these networks. These KPIs help to measure the **performance** of NNs in various tasks. Some of the most commonly used metrics for evaluating NNs are precision, recall, F1 score, and Area Under The Receiver Operating Characteristics Curve (AUC-ROC). Each of these offering an unique perspective on model performance, it is essential to consider them collectively to gain a comprehensive understanding of a model's capabilities, being careful not to rely solely on just the most common ones, which can turn our analysis of the network into a reductive one. Most of these KPIs interact with the confusion matrix, represented in Figure 2.8.

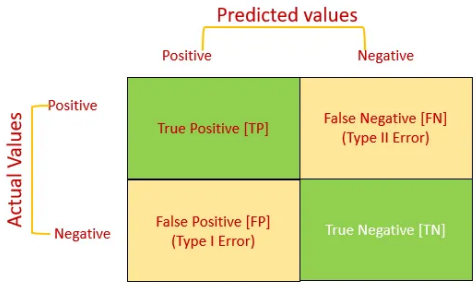


Figure 2.8: Example of a confusion matrix, used for the calculation of KPIs (adapted from [42]).

To perfectly understand the following KPIs and Figure 2.8 these concepts must be defined:

- **True positive** is an instance that was predicted as positive and whose ground truth is positive.
- **False positive** is an instance that was predicted as positive but is actually negative.
- **True negative** is an instance that was predicted as negative and whose ground truth is actually negative.
- **False negative** is an instance that was predicted as positive and whose ground truth is positive.

Precision represents the proportion of true positive instances among those predicted as positive. It quantifies the rate of positive predictions, indicating how reliable the model is when it identifies a sample as belonging to a certain class. Calculating precision happens per the equation in Equation 2. A **higher precision** indicates **fewer false positives**, which means the model is less likely to **misclassify negative instances** as positive.

$$Precision = \frac{True\_Positives}{True\_Positives + FalsePositives} \qquad \text{(Equation 2)}$$

On the other hand, **recall**, calculated according to Equation 3, measures the proportion of true positive instances among the actual positive instances in the dataset. It measures the ability to identify the ground-truth instances. A higher recall indicates fewer false negatives, implying that the model is less likely to miss existent ground truths.

$$Recall = \frac{True\_Positives}{True\_Positives + FalseNegatives} \quad \text{(Equation 3)}$$

F1 score, an aggregate mean of precision and recall, provides a single metric that balances the trade-off between these two above-mentioned KPIs. The F1 score is particularly useful when both false positives and false negatives are equally important, and it, as with the previous two, ranges from 0 to 1, with higher values indicating better performance. Though the F1 score offers a more balanced evaluation, it still assumes equal importance for precision and recall, which may only sometimes be the case in specific applications.

These last three KPIs are widely used but vary in importance depending on our objective when deploying a model for classification and detection. The example usually given is that of the medical field, in which false positives and false negatives can have significant repercussions, the former in expensive treatments and the latter in late detection of diseases. Still, to give an example more in tune with the objective of this thesis, such as if we are deploying a model for ADAS. Then the detection of a traffic light can be seen as more important than a highly correct performance in classification, leading to acceptable lower precision and recall. When this happens, if the driver is alerted sooner than he would see the traffic light, he can then classify it and take appropriate precautions. However, if deploying a fully automated system, feeding the network bad data may lead to accidents. So the model must make sure it only provides accurate data to the autonomous system. Other situations may arise when analyzing these KPIs, e.g. when precision is high, and recall is low, this may mean that the model is performing well in specific scenarios but may not have enough confidence to try to predict something in situations the user would have wanted it to.

Mean Average Precision (mAP) is a crucial metric that combines precision and recall across multiple class labels. To calculate mAP, precision and recall are computed for each class individually, and then the average precision is obtained by calculating the area under the precision-recall curve for that class. The mAP is the mean of these average precision values across all classes. Hence, the mAP metric provides a comprehensive assessment of the model's performance, taking into account both the accuracy of object detection and the ability to correctly classify different object categories.

The AUC-ROC measures the performance of a NN across all possible classification thresholds. It plots the true positive rate (recall) against the false positive rate and computes the area under the curve. A model with perfect classification would have an AUC-ROC of 1, while a random classifier would have an AUC-ROC of 0.5. The AUC-ROC is particularly useful for evaluating models with **imbalanced datasets**, as it is less sensitive to class imbalance than accuracy.

Other additional metrics include top-1 and top-5 scores, which check if the target (ground

truth) equals our top prediction or if it is in the top 5, respectively. Top-1 and top-5 scores have decreased in relevance in narrower scenarios, where few classes mean the desired class is frequently in the top 1 or top 5, like in [43]. Still, they can be used is wider multi-class implementations [44] and [45]. All these metrics are available at a glance in Table 2.3.

KPI	**Calculation**	**Usefulness**
Precision	$\frac{\text{True Positives}}{\text{True Positives + False Positives}}$	Measures the reliability of the model is when it identifies a sample as belonging to a certain class.
Recall	$\frac{\text{True Positives}}{\text{True Positives + False Negatives}}$	Measures the ability to correctly identify positive instances, particularly important when minimizing false negatives.
F1 Score	Harmonic mean of Precision and Recall	Provides a balanced metric that considers the trade-off between Precision and Recall, useful when both false positives and false negatives are equally important.
Mean Average Precision	Average Precision across multiple classes	Evaluates the overall performance of the model in object detection tasks, combining precision and recall across different object categories.
AUC-ROC	Area under the ROC curve	Measures the overall performance of a model across all possible classification thresholds, useful for evaluating models with imbalanced datasets.
Top-1 Score	Check if the target is the top prediction	Evaluates if the ground truth matches the model's top prediction, indicating the model's accuracy in its top choice.
Top-5 Score	Check if the target is within the top 5 predictions	Evaluates if the ground truth is among the top 5 predictions of the model, indicating the model's performance within its top choices.

Table 2.3: Summary of Evaluation Metrics for CNNs

2.4 Convolutional neural networks

CNNs are a subset of deep learning NN that are created specifically to process data that has a **grid-like topology**, specializing in a range of image and video processing tasks, such as object classification and segmentation. They vary in their depth and strategies, but their core consists of layers, each manipulating the input data in a particular way. The convolutional layer, pooling layer, and fully connected layer, depicted in Figure 2.9, are CNNs' three main building blocks [46]. They have evolved from small parameters, small models with low performance, to high performance, real-time or not, capable and adaptable models. They take advantage of matrix multiplication to extract features in a manner **more suited to visual data**. Although matrix multiplication is computationally demanding, it is well suited to the **paralellization** of work that is a signature of graphics processing units.

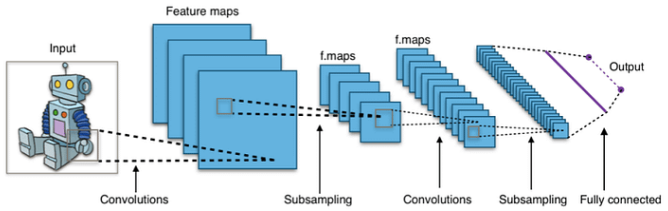


Figure 2.9: Example of a fully convolutional neural network (adapted from [47]).

2.4.1 Architecture

CNNs are specifically designed to process and analyze data with a grid-like structure, such as images. CNNs employ convolutional layers that utilize **filters** (also known as a **kernel**), as a learnable parameter to extract relevant features, like edges, lines, and shapes, from the input data. The key operation in this process is the convolution operation, which is represented in Figure 2.10. It involves taking a filter and sliding it across the input image, performing element-wise multiplication between the filter and the corresponding image pixels, while generating a feature map, representing the source image's highlights and specific patterns. In order to capture all of the features in the input image, **multiple filters** can be used to create various feature maps. The size of the filters is one of the main ways in which one can increase or decrease the size of the model, as well as determine which kind of information (contextual or specific) one wants to recover in a specific layer. These operations when done in close succession, lead to the formation of a hierarchy of features, from simpler and more generalized ones to more complex ones, which all culminate in a detection.

In the overall architecture of a CNN pooling layers might coexist with convolutional layers. This shrinks the size of the feature maps that the convolutional layer produces, making it easier to analyze, by reducing dimensionality and the compute necessary to perform forward and back passes. This is accomplished by performing operations like max or average pooling on small windows of the feature maps, overlapping a kernel with the image and extracting only the maximum or average values. The operation done by these layers is also called **sub-sampling** or

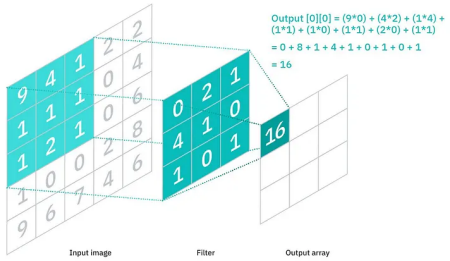


Figure 2.10: Representation of a CNNs' convolution (adapted from [48]).

down-sampling. Throughout the years the research into the impact of the type of pooling led to the spread of max pooling, as seen in Figure 2.11 , in most architectures, as it tends to perform better. This is demonstrated in [49], where the authors found that max pooling had a spread in performance that was comparable to much more complex techniques, and was slightly better than average pooling, when tested on the *MNIST*[40] and *CIFAR*[50] datasets.

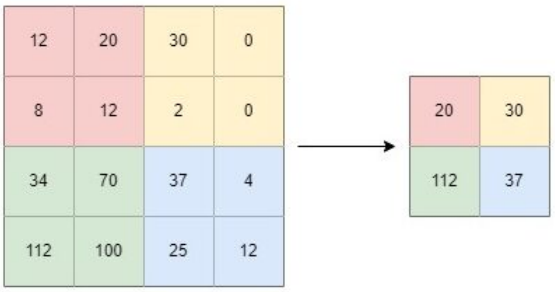


Figure 2.11: Example of a max pooling layer with a size of 2 (adapted from [51]).

For most layers, a stride greater than 1 can also be specified, increasing the down-sampling rate by "skipping" pixels when overlapping the kernel with the image. This reduces the data's **dimensionality** and helps avoid over-fitting. The size of the filter can, when in conjunction with the stride, also reduce dimensionality, by skipping parts of the input where the kernel does not fit. These previous two techniques may lead to losing information on the border of the input, so padding may be added, as zero padding, to increase the size of the input and maintain the dimensions of the output.

The network's **final output** is produced by the fully connected layer. A prediction or classification is made for the input image using the output from the preceding layers. Typically, a softmax function that creates a probability distribution over the potential classes comes after this layer.

Another important consideration, especially in one of the models that will be explored later (Section 2.4.3.1) is **batch normalization**. This consists of a technique designed to improve the training of deep NN by addressing the issue of internal covariate shift, as is discussed in [52], which occurs when the distribution of inputs to a given layer changes during training due to updates in the previous layers' parameters. This used to be dealt with by lower learning rates and more careful parameter initialization (which led to longer training). This happens by normalizing the inputs to each layer, by subtracting the mean of the current batch from each activation, followed

by dividing by the standard deviation. The research done in [52] results in the acceleration of the training process. The normalization of inputs reduces the internal covariate shift, resulting in a more **stable** network during training. This allows for the utilization of higher learning rates without the risk of gradients exploding or vanishing, leading to faster convergence. Another significant advantage of batch normalization is the enhancement of a network's generalization performance. Normalizing the inputs introduces a form of regularization, helping to prevent over-fitting. During the training process, normalization parameters are estimated using the statistics of the current mini-batch, adding some noise to the process. These noise functions act as a **regularizers**, making the model more robust to small perturbations in the input data. Consequently, the CNN becomes less sensitive to specific patterns in the training data, improving its ability to generalize to new, unseen data.

2.4.2 Transfer learning in CNNs

Having analyzed the architecture of a CNN and before continuing to specific models, it is important to establish how important the knowledge of another implementation can be. For a given task using the same architecture, the previously acquired knowledge will help reduce the computational cost and required dataset size. This concept is the core of transfer learning approaches, as was the case in [53]. This survey shows transfer learning being a means of improving performance when the given dataset is small.

In transfer learning, higher-level features (like the object's shape and size), can be transferred to the new task, while the more specific features can be extrapolated to the new classes, by adjusting the weights of the said NN. For example, the task of car TLR shares **similar characteristics** with tram TLR, namely light casing and light position on the streets. So the main segmentation and signal position can be directly transferred, while the light signals' meaning can be adapted.

Furthermore, transfer learning helps overcome the problem of over-fitting, especially when the **dataset is small**. The pre-trained models act as powerful feature extractors that generalize well to unseen data. By fine-tuning the model on our specific task, the learned features can be adjusted to better align with our targets. This is demonstrated in [54], where an analysis on impact of using transfer learning on different sizes of datasets is done, finding that deploying transfer learning greatly reduced the need for more data, as the comparison model (one trained from scratch on the same dataset) quickly over-fitted and had worse results.

2.4.3 Models for object detection

Identifying an object not only comprises understanding of the overall image but also estimate the objects and their locations contained within [55]. The best methods for not only this dissertation's specific application but also for most that need generalized feature extraction, are using CNNs and deep learning. Up until this point the focus has been on the fundamental underlying building blocks to a successful object detection model. However, putting it all together and managing to actually **train**, **validate** and **deploy** a model is a complex task. It requires an understanding of the overall architecture of actual models, such as the input, the backbone, architecture idiosyncrasies, output layers and parameters. In this section the focus will be on describing actual models that have been proven to work so that real world knowledge can accompany the background knowledge that was mentioned in the previous sections of this chapter. These models will focus on two main model categories, just as it can be seen in [56] and in Figure 2.12: single-stage detectors and two staged detectors.

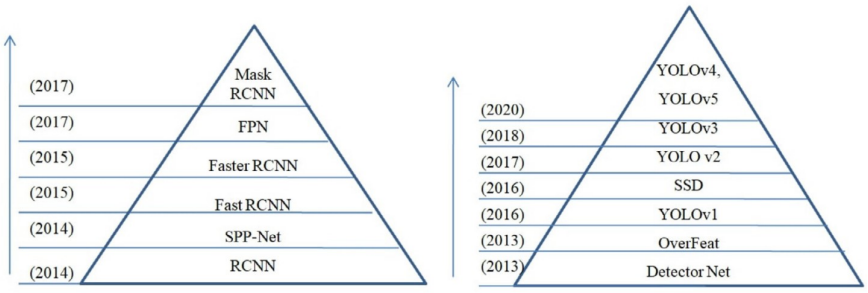


Figure 2.12: Image depicting different types of detectors (adapted from [56]).

Single-stage detectors, as You Only Look Once (YOLO), in [57], operate by directly predicting the class labels and bounding box coordinates in a **single forward pass** of the network. These detectors are known for their high computational efficiency and real-time performance, making them suitable for applications that require low-latency processing, but also making them less accurate.

Two-stage detectors, such as the Region Based Convolutional Neural Network (R-CNN) of [58] and its variants, work in a hierarchical manner. Initially, they generate a set of region proposals that potentially contain objects, often using techniques like selective search [59] or objectness [60] . After this, these proposals are passed through a classification and bounding box regression network to produce the final object detection. These detectors generally exhibit **better accuracy** and are **more robust** when dealing with varying object scales and occlusions. However, they tend to lose to single-stage detectors in speed. In [56], a well-detailed analysis of different backbones is shown, which tend to be more densely connected and deeper in two-staged detectors, as well as their performance relative to each other.

Now, a detailed explanation will be given on models of each of these different categories, providing a profound look into not only how they function but also how they perform in standardized tests. Later, their performance will be shown in more individualized and specialized cases.

2.4.3.1 YOLO

Going into our first example of the architecture of a CNN-based object detection model (single-shot), YOLO is a family of models that prioritize **speed** and **real-time performance**, introduced in [57]. YOLO, seen in Figure 2.13, divides the input image into a grid and assigns a number of bounding boxes, confidence scores for those boxes, and class probabilities to each grid cell, predicting object locations and classes in a single forward pass. The first version of YOLO, YOLOv1, achieved real-time detection with a mAP of 63.4% on the PASCAL VOC 2007 [61] dataset, at a speed of 45 Frames Per Second (FPS), which although was only on par with other state-of-the-art techniques' speed, it managed to achieve this while at least doubling the speed of detection for all models comparable in mAP.

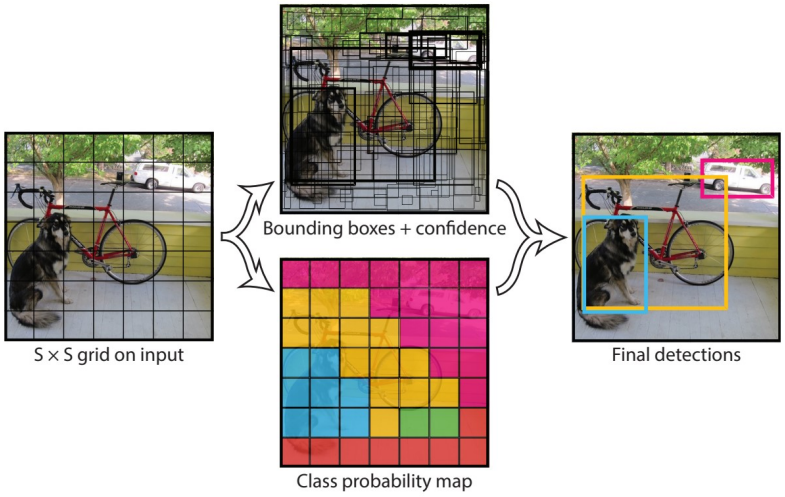


Figure 2.13: Depiction of how YOLO works. Adapted from [57].

After YOLO had been proven to be a very good detector for real-time applications, as seen in [57], numerous iterations on its model came to be in the following years, trying to improve accuracy and speed in standardized challenges and datasets. So, the focus will now be on the main thread of iterations on YOLO, but many more have been done for specialized applications, altering its' architecture, backbone, parameters, etc.

YOLOv2 [62] introduced several improvements over YOLOv1, such as the adoption of anchor boxes, more specifically the ones resulting from a clustering algorithm in the **MS COCO**[63] and **PASCAL VOC**[61] datasets, for better bounding box prediction. A higher resolution input image was also used and batch normalization was also incorporated. These enhancements resulted in **better performance**, achieving a mAP of 76.8% on **PASCAL VOC**[61] and a speed of 67 FPS.

As for YOLOv3 [33], it further refined the architecture by employing a deeper and wider network with **shortcut connection**, inspired by the Darknet-53 [64] backbone. Additionally, YOLOv3 adopted a multi-scale detection approach, predicting objects at three different scales to

better detect objects of varying sizes. This version reached a mAP of 57.9% on the **MS COCO**[63] dataset while maintaining **real-time processing capabilities**, as seen in Figure 2.14.

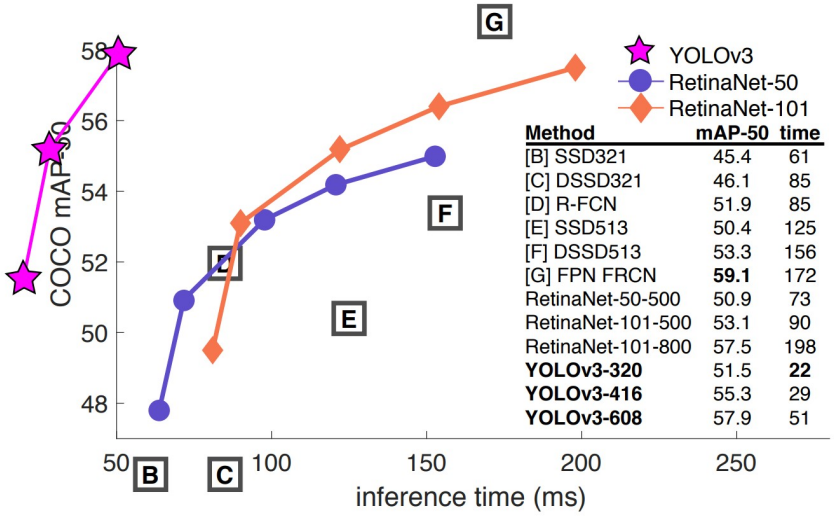


Figure 2.14: Depiction of how YOLO performs in the *COCO* [63]. Adapted from [57].

When dealing with the output of YOLOv3 models, the output takes the form a large number of predictions with different values, from the confidence score of each class, to the coordinates of the bounding boxes. These outputs commonly come in the form of large lists containing this information and, therefore, have to be filtered based on at least two parameters: confidence threshold and IoU threshold. The confidence threshold works by eliminating the variables with confidence score below a certain pre-defined value. The IoU threshold is used in conjunction with non-max suppression to combine bounding boxes that do not meet a certain threshold. A visual representation of this process can be observed in the image provided in the Figure 2.15. The outputs after filtering can then be displayed (in a test scenario), used for loss calculation (in training) or for KPI calculation.

Figure 2.15: Example of Non Max Suppression (adapted from [65]).

YOLOv4 [66], incorporated a wide range of advancements in object detection, such as the incorporation of other architectures architectures into itself, as well as data augmentation techniques like the mosaic technique, which joins 4 training images so that "4 different contexts are mixed". YOLOv4 also made use of the Bag of Freebies and Bag of Specials concepts, which contributed

to its improved performance. With these optimizations, YOLOv4 achieved a mAP of 65.7% on the **MS COCO**[63] dataset while maintaining real-time processing at 62 FPS.

Throughout its various versions, the YOLO family of models has consistently demonstrated the ability to balance high performance with real-time processing capabilities, making it a popular choice for many object detection tasks.

2.4.3.2 R-CNNs

The initial proposal for R-CNN family of models is proposed in [58], introducing the R-CNN network, seen in Figure 2.16. This network solves a previous implementation of Region of Interest (RoI) that used simple methods to identify regions on which to run a CNN and perform object detection, and thus try to **reduce computation complexity**. This same previous method simply could not do this efficiently and suggested too many RoIs, with tremendous computational complexity. As, such, in [58], there are only up to 2000 regions on which said object detection are applied.

In the feature extraction component, each region proposal is wrapped to a fixed size and passed through a pre-trained CNN, typically AlexNet, to extract a fixed-length feature vector. The CNNs final fully connected layer is removed, and the output from the second-to-last layer is used as the feature representation for each region proposal. This feature extraction step helps capture the discriminative information necessary for accurate object recognition.

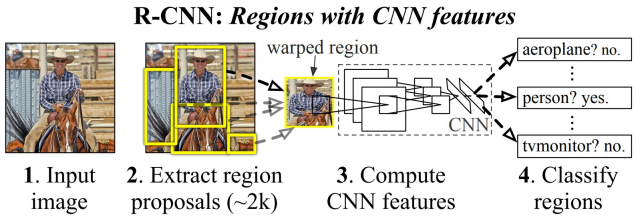


Figure 2.16: Depictions of the workings of R-CNN (adapted from [58]).

The final component of R-CNN involves training a set of linear SVMs for classification. For each object class, a one-versus-all linear SVM is trained using the extracted feature vectors. The SVMs are responsible for determining the class labels for the region proposals, as well as refining the bounding box coordinates using bounding box regression. The final object detections are obtained by combining the class labels and refined bounding box coordinates.

R-CNNs, at the time of its' introduction, achieved performance of 53.7 % mAP on the PASCAL VOC 2010 [61] dataset, outperforming the previous state-of-the-art methods by a significant margin. However, R-CNNs main drawback is its computational complexity, primarily due to the need to process thousands of region proposals for each image, making it **unsuitable** for real-time applications.

Subsequent developments in the R-CNN family, such as Fast R-CNN [67] and Faster R-CNN [68], have sought to address these efficiency concerns while maintaining or improving the high detection accuracy achieved by the original R-CNN. After this first implementation [67] improved on the previous proposal mentioned above by modifying the feature extraction process. Instead of processing each region proposal individually, Fast CNN applies the CNN to the entire input image, generating a feature map from which region proposal features are extracted using a technique called RoI pooling. This change reduces the number of CNN computations required and significantly improves **processing speed**, exemplified in Figure 2.17. Furthermore, Fast R-CNN replaces the linear SVMs used in R-CNN with a softmax classifier and incorporates bounding box regression directly into the network, streamlining the training process.

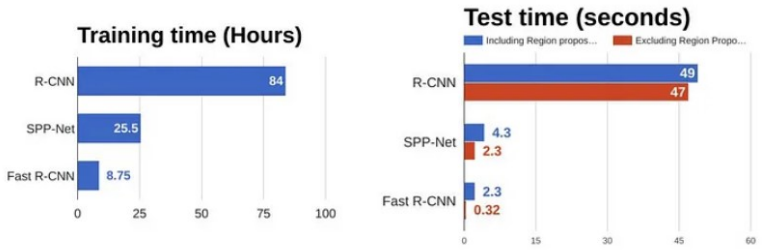


Figure 2.17: Training time and testing time of R-CNN and Fast R-CNN (adapted from [69]).

2.5 Datasets

As a critical factor contributing to the success of object detection using CNNs the availability of large, diverse datasets, as these are key when training a model. These datasets enable CNNs to learn the **complex**, **high-dimensional** patterns and variations present in real-world diverse scenarios, resulting in improved performance and generalization capabilities. However, acquiring large and varied datasets can be challenging, particularly for specific applications where data availability may be limited or difficult to obtain, such as TLR in trams. It is important to note that, depending on the application, or even if transfer learning is being done, the size of the dataset and their variability may not need to be as profound, still, it needs to be adequate for the task and it is safer to use a larger one.

In the case of tram TLR (this serves as an example of creating a dataset for a visual task) using a CNN, gathering a sufficiently diverse dataset poses unique challenges. Firstly, tram systems across different cities or countries may exhibit variations in their traffic light designs, colors, and configurations, requiring a dataset that captures these differences. Additionally environmental factors, such as lighting conditions, weather, and occlusions, must be considered to ensure that the CNN can perform well under various circumstances. Assembling a dataset that covers all these aspects is **time-consuming** and **expensive**, as it may involve collecting images from multiple tram systems and multiple weather conditions, therefore spanning multiple days or times of the year.

This variability impacts the time to create a dataset, with a great contributing factor for this being the labelling process.

As examples of datasets that fit into the rules and take into account precautions to mitigate the above mentioned problems, LISA [70], MS COCO[63] and CIFAR-10 [50] stand out as widely used.

2.5.1 Data augmentation

Data augmentation (exemplified in Figure 2.18) may become an essential technique for the training process of CNNs, particularly if the available dataset is limited in size and or diversity. **Data augmentation** involves applying transformations to the original images in the dataset, generating new samples that capture different perspectives or variations of the original data. The transformations can be applied in such a way as to enhance the dataset in meaningful ways for the context of the model and objective at hand, this could, to give an example, materialize itself in changing the color values of an image to simulate different weather situations. This, as with the underlying structure and choices made for the non-augmented dataset, must be done so as to be relatable to the deployment situation. These transformations include rotation, scaling, flipping, translation, shearing, and adjusting brightness, contrast, or color.

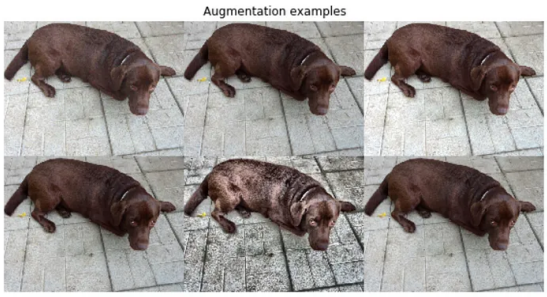


Figure 2.18: Image depicting data augmentation (adapted from [71])

This may help with the dependency of CNNs large datasets, helping to avoid over-fitting. As an example, when data augmentation is applied to TLR, it can help simulate **various environmental conditions** and situations that may, or may not, be present in the original dataset. For example, applying transformations that simulate different lighting conditions, camera angles, and occlusions can better prepare the CNN to handle real-world variations in camera hardware, weather conditions or location. Applying this correctly can mitigate, at the root of the problem, cases of over-fitting and lack of data to extract features, just like the authors of [72] found.

Some types of data augmentation include:

- **Color Augmentation**: In this the value channels of an image are manipulated so as to simulate change in weather, lighting conditions or color variants.

- **Spatial Augmentation**: This augmentation applies transformations like rotation, scaling, translation, flipping and distortion to an image, with the objective of simulating situations like different camera positions or point-of-view changes.
- **Noise**: This augmentation introduces noise to the image, reducing the quality of its' data.
- **Image occlusion**: Patches in the image are occluded, simulating occlusion situations.

Chapter 3

Bibliographical Review

After delving into the theoretical concepts needed to have an understanding of this project, the attention can now be shifted towards practical applications within the railway domain, or those that have relevant features to the objective of this dissertation.

The main focus in this chapter will be on the state-of-the-art solutions on different topics. Initially, **car-centric implementations** will be analyzed. Particularly those that have similar problems and/or objectives will be considered, since they are very comparable to the environment in which railway applications operate. Lastly, this chapter will explore the existing **railway and public transport applications** relevant for this dissertation and their contributions will be weighed and explained.

3.1 Car-centric implementations and their feasibility

Having explored less specific and more variable models in Section 2.4.3, the focus shifts to the state-of-the-art implementations in car-centric situations. This exploration results from the commonalities that these implementations share with tram traffic light recognition tasks due to the inherent similarities between the two types of traffic lights and situations. Car-centric implementations relate to railway applications as both serve the same fundamental objective: controlling and guiding vehicular movement through similar environments and conditions (active and passive participators, varying weather, different lighting conditions, hardware variability, etc). As such, and having established that implementations for car scenarios would be relevant for possible.

3.1.1 Traffic sign and light detection

Firstly, it is important to introduce some of the datasets where implementations of different models can be tested and trained. Datasets like the German Traffic Sign Recognition Benchmark (GTSRB) [73], on which comparisons of the performance of many models, and even against humans themselves, can be found at [74]. This dataset consists of more than 50000 images with 43 classes of varying traffic signs. Aside from this, the *Tsinghua-Tencent.100K*[75] dataset, with over 100000 images and 117 classes, and the LISA traffic light dataset[70] seem to be the more specialized

ones. However, it is worth noting that larger datasets like MS COCO[63] also contain images of traffic signs and lights, but they are not recommended for highly specialized implementations of Traffic Sign Detection (TSD) as their focus is not exclusively on these elements.

In [9] classical Machine Learning (ML) techniques can achieve interesting results in TSD and Traffic Sign Recognition (TSR), with the authors going from simple to advanced techniques and finding a sequence of steps that are usually found in most applications: pre-processing, followed by feature extraction, detection of the objects and the post-processing. Regarding the **pre-processing** step, which usually consists of hardware-specific geometric corrections, colour corrections or image enhancements (e.g. sharpness, contrast or noise), [17] stands out as an example of lateral thinking in their approach to this step, when the authors use the Lightness;Green-Magenta;Blue-Yellow (LAB) colour space, in which the second channel corresponds to a spectrum of green to red values of a pixel, to aid in their detection and classification step. Although the authors achieve lacklustre results when compared to more advanced and deep learning techniques, they still find good performance in their specific use case, in spite of having some dataset drawbacks and lacking applicability to new scenarios. Still on classic ML models for TSD, [76] for SVMs, [13] for SIFT, [15] for *Haar* cascades and [77] for principal component analysis are all still relevant, managing to either fulfil their roles as classifiers or by supporting other models, as add-ons, to improve their performance. Moving on to deep learning, in [9], a clear distinction was found in accuracy and speed when comparing deep learning methods (with or without classical ML ones attached) to solely classical ML ones. The former had a distinct advantage, achieving much better performance, due to their ability to generalize and better feature extraction. Continuing, in [78], the authors once again corroborate the advantages of NNs, as they affirm that they are superior in TSD, when compared to traditional methods, like HOG or SIFT.

In [79], a survey directed at TLR is conducted, in which the colour spaces used by authors in TLR tasks are explored, from Red;Green;Blue (RGB) to Hue;Saturation;Value (HSV), LAB and Brightness;Blue Projection; Red Projection (YUV), and it is found that normalized RGB overcomes illumination variations more easily. Following this, the analysis progresses to the physical details of the light lamps. Here it is detailed that properties like shape, aspect ratio, texture, positioning in space (above the street), orientation (mostly vertical) and size can be used to apply rules that exclude false detections. With this knowledge the authors highlight the implementations that managed to succeed in this task, acknowledging traditional techniques while also emphasizing the integration of *a priori* knowledge, as in [80], where Global Positioning System (GPS) information was used to improve RoI choice, when the distance is below 100 meters. This resulted in a performance increase when compared to the baseline RoI algorithm, with a more significant improvement in dusk scenarios than in afternoon ones.

Shifting the focus to CNN, but remaining in TSD, in [81] the authors choose to use a CNN for their detection model, while using the GTSRB [73] benchmark. This is a broad classification problem, spanning 43 classes of images with resolutions ranging from *15x15* to *250x250* pixels. The authors' model managed to be comparable to other already evaluated models, reaching above 99% top-1 accuracy. Importantly, they were also able to demonstrate that their network is resilient

to the degradation of the test dataset due to fading and rotation, with a performance drop of less than 2% even for the most intensive deteriorations.

Furthermore, [66] uses an implementation of YOLOv4, more specifically in[82], where the cross-layer connections were enhanced by focusing more on high and low-level features throughout the convolutional layers. This focus led to improved feature extraction capabilities, yielding a 1% increase in mAP on the *Tsinghua-Tencent.100K*[75] dataset compared to the original YOLOv4. This was achieved at 31.25 FPS, a frame rate that the authors deemed capable of real-time detection.

In [83], multiple CNN based architectures are compared, focusing on LENET-5[84], IDSIA[85] model, URV[86] model, a CNN with asymmetric kernels and a CNN with 8 layers as the focus of the classification problem. The dataset, designed to complement the GTSRB[73], contains rural and urban environments, at daytime and sunset, and was collected across several European countries. An extensive data analysis was conducted to determine the types of signals present, their categories, and whether they were shared between countries or country-specific. Performance was measured on both datasets, and the KPI indicated not only above 80 %accuracy for all models, but also that they all ran in real-time (on the scale of milliseconds).

When advancing to more complex models, implementations such as those described in [87] become noteworthy. In this study, an average accuracy of over 90% was achieved, a statistic made even more impressive by the fact that this average was calculated based on the model's ability to detect signs at speeds ranging from 80 to 120 km/h.

Thus, it can be concluded that effective models exist for the detection of car traffic objects. Even though these models do not specifically target the objects of interest in our study, similar ones are examined. These implementations, therefore, reinforce the viability of using deep learning models for our task of tram traffic light detection.

3.2 Existing research on railways and public Transport

Research on rail infrastructure represents a significant and relevant part of state-of-the-art techniques relevant for future work. These techniques involve similar scenarios, including environment characteristics, object and obstacle detection, as well as active participant detection. Particularly relevant are sign and light detection, which more directly align with our current objectives.

3.2.1 Rail maintenance and obstacle detection

Rail maintenance represents an important part of state-of-the-art techniques, specifically in the realm of object and obstacle detection. These implementations are mainly within maintenance scenarios and, as such, include, as an example, implementations limited to only nighttime.

The authors of [88] used RailSEM19[89], a public dataset for train management comprising 8500 unique images taken from the perspective of rail vehicles, such as trains and trams. To create the dataset, sensors were integrated into housings mounted on the front profile of a locomotive,

with vibration isolation mechanisms minimizing image quality disruption due to vibrations. The test length spanned 120 km on the Serbian part of the Pan-European Corridor X, with an average speed of 34 km/h and a total runtime of 3.5 hours. Additional runs were conducted to augment the dataset. For AI-based obstacle detection, the image processing incorporated object detection using YOLOv3[33] and distance estimation. This model was trained on the COCO[63] dataset which contains 3500 images of trains but lacks many distant objects. The authors then applied transfer learning to categorize objects relevant to the RailSEM19[89] dataset and enhance the algorithm's ability to detect distant objects. In the evaluation phase, the algorithm reliably identified objects. However, errors in bounding box prediction led to an increase in **distance estimation errors**, as smaller variances in input reliability and accuracy become more pronounced in the output of the distance estimation algorithm.

In [90], the authors choose a thermal imaging framework to detect, locate and classify anomalies in railways during nighttime scenarios. To achieve this goal, a dataset was created using nighttime footage and attaching external lights to aid the camera. The paper also mentions the usage of high-speed frame acquisition to avoid motion blur, which could result in loss of detail, and ensured high resolution to capture detail even in small objects. The detection model was designed in two stages. Firstly, a frame is classified as abnormal. Then, the anomaly location module outputs a probability map indicating the location of the obstacle. In this way, the model is able to perform anomaly detection, even of objects it had not seen in training while operating at around 100 FPS.

Additionally, the survey by [91] provides a comprehensive review of state-of-the-art techniques for on-board detection of obstacles on and near rail tracks, aiming to improve railway safety. Despite the importance of railways as a primary mode of land transport, research and development in this field have been less intense compared to road transport. The paper analyzes three essential aspects of vision-based onboard estimation: rail track extraction, detection of obstacles on rail tracks, and estimation of distances between onboard cameras and detected obstacles. Evaluation tests are examined and grouped into two categories: evaluations based on images/videos available on the internet and evaluations on real-world images/videos.

Moreover, [43] uses an implementation of a R-CNN model, with a focus on improving reliability by performing a fusion of data from multiple sensors: cameras (with long and short focus), Laser Imaging, Detection, and Ranging (LIDAR) and millimetre wave radar. The model aims to detect 11 different classes: person, rail, box, sign, billboard, power distribution box, schoolbag, paperboard, signal, platform, and helmet, having structural information (common length, width and height values) for each. With this, the authors manage to outperform implementations such as those mentioned in subsection 2.4.3.2 or 2.4.3.1 in mAP, while being outperformed in inference time.

In their work, [92] uses an improved version of the YOLOv4[66] algorithm, improving obstacle and pedestrian detection compared to previous methods. One-stage detectors were chosen for this study because of their **faster processing**, despite being less accurate than two-stage detectors. The algorithm divides each image into *SxS* grids, where a grid detects the target whose centre falls

into the cell. Each grid cell outputs B boundary boxes, the area of the object, and the probability of it belonging to a class. The backbone architecture used is DarkNet-53[64] and cosine annealing decay is used for learning rate adjustment. K-means clustering is used to enhance predictions, making it easier to identify lights. **Data augmentation** techniques including flipping, rotation, translation, scaling, contrast change, pooling, and altering brightness in the RGB channels have also been applied to artificially enlarge the dataset. To prevent over-fitting due to limited data, transfer learning from the COCO[63] and KITTI[93] datasets was utilized. The first stage of training involved freezing pre-training parameters, which were then unfrozen in the second stage. This algorithm improved mAP by 2.33% (with other increases across all KPIs of 2.3.6) while adding only less than 10% in inference time. Still, where the model differentiated itself from the other implementations such as YOLO [33] and models from the R-CNN [58] family was when noise was introduced to the test. In this situation, the lead in performance in mAP doubled.

In [94], the authors discuss how Intelligent Transportation Systems (ITS)s have experienced rapid development, with computer-vision perception units serving as their key components. They also affirm that detecting various objects enables train drivers to identify obstacles in time to **prevent collisions**. While the Single-Shot Detector (SSD) can detect objects of different sizes using multi-scale feature maps in real-time, its feature map lacks high-level semantic information, limiting its ability to detect small objects and making it unsuitable for multi-scale railway object detection. To achieve high accuracy and real-time railway object detection, the authors propose a receptive Feature Enhanced Single Shot Detector (FE-SSD). Researchers have explored various approaches for detecting obstacles in railways, and the FE-SSD aims to balance accuracy and real-time performance. By incorporating features like enhancing convolutional layers with former feature maps, associating the scales of anchors with the map location (and then filtering them) bettering the remaining anchors. This model demonstrates high real-time performance and the highest detection mAP among the five detectors tested. Although FB-Net exhibited the fastest real-time performance, its accuracy was lower than the FE-SSD.

Another notable implementation is [95] where challenges such as high computational costs and difficulty in detecting small obstacles are addressed by implementing a CNN-based algorithm with a residual module for improved accuracy and simplified training, as well as a feature extraction network capable of detecting both large and small obstacles. The proposed method consists of two parts: **feature extraction** and **feature fusion**. Using a NN-based extraction architecture with a residual block, the network becomes easier to train as layers deepen. Then, convolution layers are added, to obtain high-level feature maps. Subsequently, the researchers choose six different scales of feature maps for feature fusion to enhance the precision of obstacle detection.

3.2.2 Traffic sign and light classification and segmentation in Railroads

Regarding the state-of-the-art for Traffic Light Segmentation (TLS) and Traffic Light Classification (TLC), in [96], a YOLOv5 implementation is trained to detect railway traffic lights, though these are colour coded and represent green and red lights (meaning *go* and *stop*). The images used for this training were collected from two cameras, one at the end and one at the head of the train.

These were filtered to cut down the size of the footage to around 3000 images, of which 1600 were labelled and used to verify the ability of YOLOv5 to effectively detect railway traffic lights. This resulted in precision and recall values above 94% in all scenarios at 100 FPS.

In [97], an older iteration of YOLO, YOLOv4, is used to try to detect a variety of signals (blue, white and red) as well as pedestrians. This resulted in 93.5% mAP and an improvement of about 6% when compared to an YOLOv3 implementation overall and almost 18% for pedestrians. It is worth noting, particularly in the context of this review, the achievement of 97.38% Average Precision (AP) with a YOLOv3 implementation in the detection of white lights, although these are just a subset of the entire range of signals detected in the study.

In [98], the focus is narrowed to pedestrians, specifically, safe passage in crossings. The dataset for this study is collected from cameras at fixed locations in crossings, where the RoIs are determined *a priori* to reduce computational demand. Results from detection are compiled into a state that has the capability to determine whether or not a pedestrian passage is safe. This model achieved results in the 90th percentile in precision and recall for both safe and unsafe passage states.

Last but not least, for sign recognition [99] deals with Australian rail scenarios, in which small signs are detected with close to 20% less accuracy than the large signs, with medium and large signs being detected with above 79% accuracy by the Faster-R-CNN model. The authors also specify the resolution of the images as one of the main drawbacks of their implementations.

To conclude, the research that is most relevant to our work supports the usage of deep learning methods to detect tram traffic lights.

Chapter 4

YOLOv3 Based Detection and Classification

In Chapter 2 and Chapter 3, the focus was on the background knowledge needed for this work, and the state of the art of said knowledge, that is, the available implementations. Now, in this chapter, we start to explore the procedures, techniques and difficulties, as well as their implementation in applying an open-source YOLOv3 model to the task of detecting and classifying tram traffic lights. The dataset, the model implementation, KPIs and the training done on said model will be explored, with a focus on how the model was improved and what had to be changed to apply it to the use-case of this project. The aim of this chapter, as explained in Chapter 1, is to optimize as many of the variables mentioned above, so as to end up with an implementation capable of segmenting and classifying tram traffic lights.

4.1 The dataset

When compared to other areas of research, the lack of publicly available datasets for lights in railway scenarios is clear (Chapter 3). Although railway datasets exist, they don't seem appropriate for this project, as they fall under the detection of signs and not lights. Therefore, there was a need to create a tailored dataset.

4.1.1 Choosing the dataset

This section aims to explain how the main problems mentioned before were overcome, by focusing on the following objectives:

- Finding a repository with available data suitable for the needs of this project.
- Choosing from the physical locations of the available data (*e.g.* Berlin, Munich and Krakow), that which had length, diversity and quality.
- Determining the guidelines for how data would be labelled.

- Analyzing the created dataset for edge cases so as to make sure it could be easily readjusted for other implementations.

Having outlined these objectives, it was determined that the dataset had to be created from internet data, posted by a third party, and labelled by us. Still, interest in urban planning and public transit is rather small when compared with car infrastructure. This, therefore, meant that options were limited in choosing a group of videos that was not just a one-off recording of a single tram line in a small city, but one that not only had a lot of lines and footage, with a mix of environments and variables that could ensure the dataset would be usable even if small. Note that small repositories of smaller cities do have merit and usefulness, as they can be used to supplement the main dataset or test the performance of the model to ensure that it does not over-fit to a specific camera, environment, weather, etc if they share the same tram traffic lights that the chosen dataset.

When searching the internet for footage with these constraints, while there is a considerable amount of regional train footage from the driver's perspective, options for tram are limited. Still, a good candidate was a series of videos from a YouTube channel [100] that had multiple lines in multiple cities in Germany, with the initial dataset being composed of 6 lines from Berlin. As such, the data was labelled from these videos, focusing on mainly labelling the tram traffic lights individually and as a group, which will be explained in the following section.

4.1.2 The environments in the dataset and German tram traffic lights

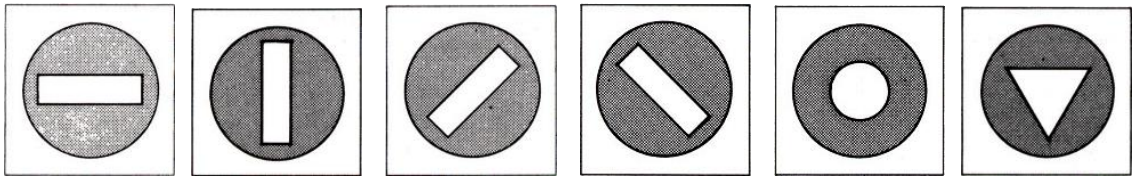
This section will look closer at the dataset used to train the YOLOv3 model, exploring their specific German traffic lights. By doing, this the drawbacks and the advantages of this dataset can be detailed and explained.

Firstly, the focus should be on the types of tram light signals encompassed in the German legislation [5], so that the following dataset can be correctly interpreted. So, after consulting the available lights, it was found that those that pertain to tram movement, and the ones that will be the focus of the detection model, are the *F0*, *F1*, *F2*, *F3*, *F4* and *F5*, as seen in Figure 4.1.

These lights take the following meanings:

- *F0* should be interpreted as the equivalent of a red traffic light in cars, as such, the tram should stop before the light and at a safe distance.
- *F1* is the equivalent of a green light for a car, meaning the conductor can proceed.
- *F2* is interpreted as being allowed to proceed to the right.
- *F3* lights should be interpreted as ride released only to the left.
- *F4* means the driver should stop, the equivalent of the yellow car traffic light.
- *F5* means the driver should proceed carefully.

After searching for available data on the aforementioned traffic lights and deciding to use the YouTube playlist [100], it is possible to draw from the videos the following key points:



(a) F0 (b) F1 (c) F2 (d) F3 (e) F4 (f) F5

Figure 4.1: Tram traffic light signs according to German law [5].

- This playlist included instances of all the signs (except *F5*, which was very rarely present in the tram routes).
- *F0* and *F1* were the more prevalent signs, followed by *F2*, *F3*, *F4* and *F5*, in this order.
- Different types of environments present in the playlist, those being suburban, urban, dedicated tram lanes and coexistence in car traffic lanes and intersections (seen in Figure 4.2).

(a) (b)

Figure 4.2: Images depicting an urban, and a suburban environment of the dataset, *a)* and *b)*, respectively.

This ultimately converged in a dataset of about 8000 images, of which around 2000 were labelled, which amounts to a 20% ratio of labelled to unlabelled images. In terms of dataset content, being Berlin a major city with dense yet sprawl signals, traffic light recurrence does not match other smaller cities. This can be a source of bias to the model.

Going into more detail on the instances of the various tram traffic lights, it is possible to see that *F0* appears the most (1473 labels), with *F1* having 36.85% of this value. The lower *F1* frequency relates to the tram being in movement whenever a *F1* light is encountered, being thus captured for less time. The opposite occurs for *F0*, which has significantly more instances where the tram is stopped. Additionally, these two lights together accounted for 96.64% of all the labels. This means that for the rest of the lights, only 24, 23 and 24 images were labelled, respectively. Note that *F5* had no appearances in this dataset. All these ratios would be roughly maintained even if more data was added to the dataset (unless specific efforts were made otherwise), as the other lines in the playlist share similar characteristics.

The frames were labelled at intervals of 1 second, that is every 25 FPS for these specific videos, due to the time constraints. These chosen frames were always saved and added to the dataset, independently of if they had a label or not. The time interval of 1 second was chosen based on the specific implementation, in which there is no tracking or temporal cohesion, and the segmentation and classification are done on a frame-by-frame basis. As such, it is not expected big impact on the model's performance when deployed, even in higher FPS scenarios (*e.g.* a camera as the source). Therefore, this specific dataset seems to be representative of real-world scenarios, at least in Berlin where the lines do not intersect or diverge regularly. Thus, diversifying the dataset scenarios is one suggestion for future work improvements

It should be mentioned that two instances of the same dataset were done, one in which, due to the aim of compatibility with Continental's tools, the whole casing of the tram traffic light was labelled, and another where only the light was labelled. An example of this can be seen in Figure 4.3. The former labelling process was used for training because it was closer to other resources inside Continental.

(a)

(b)

Figure 4.3: Different of labels used when creating the dataset, with *a)* represents the labelling of only the lights and *b)* the labelling of the whole casing.

Having collected an apt dataset for the work of this dissertation, some existing properties of the said dataset were still lacking: the amount of information within the images was not the best. Firstly, images were downloaded from a YouTube playlist, meaning they were already compressed, making it hard to interpret parts of the image. Then, 720p was the maximum resolution available. Nonetheless, the recordings were still relatively clear, even if there were still some room for improvement.

The limitations manifested themselves in terms of **dynamic range** and **resolution**. More specifically, these limitations meant that areas under shadow and at a distance lacked information. Overcoming these limitations was a significant concern, as it would enhance the system's reliability in testing and improve its ability to extract information from parts of the image with sub-par information or to perform when encountering new situations. Some limitations can even contribute to new edge cases. Therefore, the dataset had to be enhanced in terms of **zooming**, **translation**, **rotation**, **scaling**, and **colour representation** (hue, saturation, and value), to mitigate the limitations. Through testing, various transformation values were explored, and the most optimal ones were identified during training.

Therefore, the images need to be of sufficient quality for our vision system to detect the traffic light signs at a long or adequate distance. Such a dataset would also be acceptable for training a NN. This brings us to another important consideration: as we did not use any dataset from Continental, different camera properties that could be adapted to better fit our needs were not modifiable. For example, more resolution, a wider field of view or a greater dynamic range. Some images of our dataset, as in Figure 4.4, lacked detail, especially in areas of high or low brightness and distant traffic lights. This resulted in detections that would be easier for a human, due to all the sub-context we can derive from the scene, but challenging for a NN. Nonetheless, instances where this happened, were labelled so as to give the network a chance to learn these scenarios.

(a)

(b)

Figure 4.4: Examples of the lack of detail in an image present in the dataset.

Lastly, another limitation encountered was the lack of lower visibility and diverse weather conditions in the dataset. The playlist used for training mainly consisted of recordings captured in similar weather conditions, with good weather conditions being the main focus. Although nighttime videos were considered, they had to be obtained from a location in Poland, which could

have disrupted the dataset and affected the results, due to the fact that only the *F0* and *F1* signals are shared between the two countries, and they might even be used in different circumstances. Considering the project's goal, which primarily aimed to aid in tram driving, focusing on good weather conditions was sufficient as proof of concept for Continental. This would enable them to deploy the model with Continental's camera and a dataset that provides a wider range of variability, ensuring accurate performance in various situations.

4.2 The model, transfer learning and pipelines

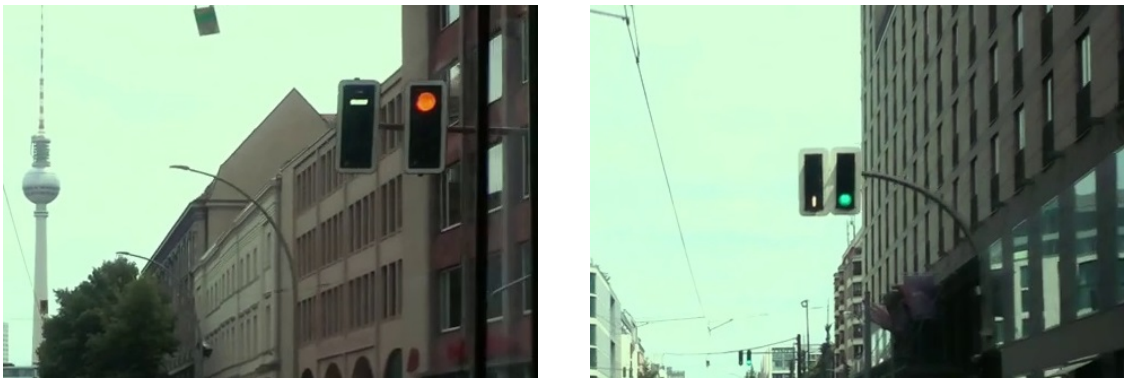
The focus of this section will be the usability and implementation aspects of the YOLOv3 model that was chosen to be deployed. It is crucial to explore the factors that influence its performance, including inputs, outputs, and their treatment, as well as their impact on the model's functionality. Additionally, we will discuss the significance of hyper-parameters, transfer learning, and the model's suitability for the specific use case of detecting tram traffic lights using a CNN. We will specifically outline the KPIs used to assess the model's performance, which will be further discussed in the subsequent chapter in order to evaluate the model's usability in real-world applications. Throughout this analysis, we will relate these factors to the model's architecture, explaining how and why they influence the model's behaviour, both in theory and based on our empirical observations.

4.2.1 The original implementation: dataset and transfer learning

Before delving into the performance, architecture, and pipeline of our developed model, let's establish a baseline to gauge our desired performance. To achieve this, there was the need to consider the previous state of the model used in this project and its implementation, which can be accessed at Github [101]. Therefore, understanding the original implementation, its' advantages and disadvantages allows us to justify the utilization of transfer learning in our specific implementation. It is exactly this that will be now explored.

As discussed in Chapter 3, most CNN implementations in driving scenarios focus on car-centric situations. Hence, due to the similarities in environments and conditions, it is reasonable to assume that such implementations would serve as a good starting point for transfer learning. Namely, the environments are similar, especially when trams do not have a dedicated pathway and share the street with cars and active traffic participants. Strengthening this point is the signalling type, using lights, which brings with it obvious similarities, with the exception of colour, in casing size and shape, as well as with the meaning of the lights. The main ones are *stop* and *go*, which translate easily into the positioning in the light casing and the meaning of said lights. This is apparent in Figure 4.5.

This means that, given the same model, trying to train a network from scratch, when compared to training on top of all the specialization and features of an already existing model, is an option that should, and was, explored. Exploring this meant that the computational requirements, both in terms of time (by the number of epochs) and model size were reduced, as the training needed to



(a) (b)

Figure 4.5: Similarities between car and tram-designed traffic lights. In *a)* a "stop" sign is represented and in *b)* a "go" sign is represented.

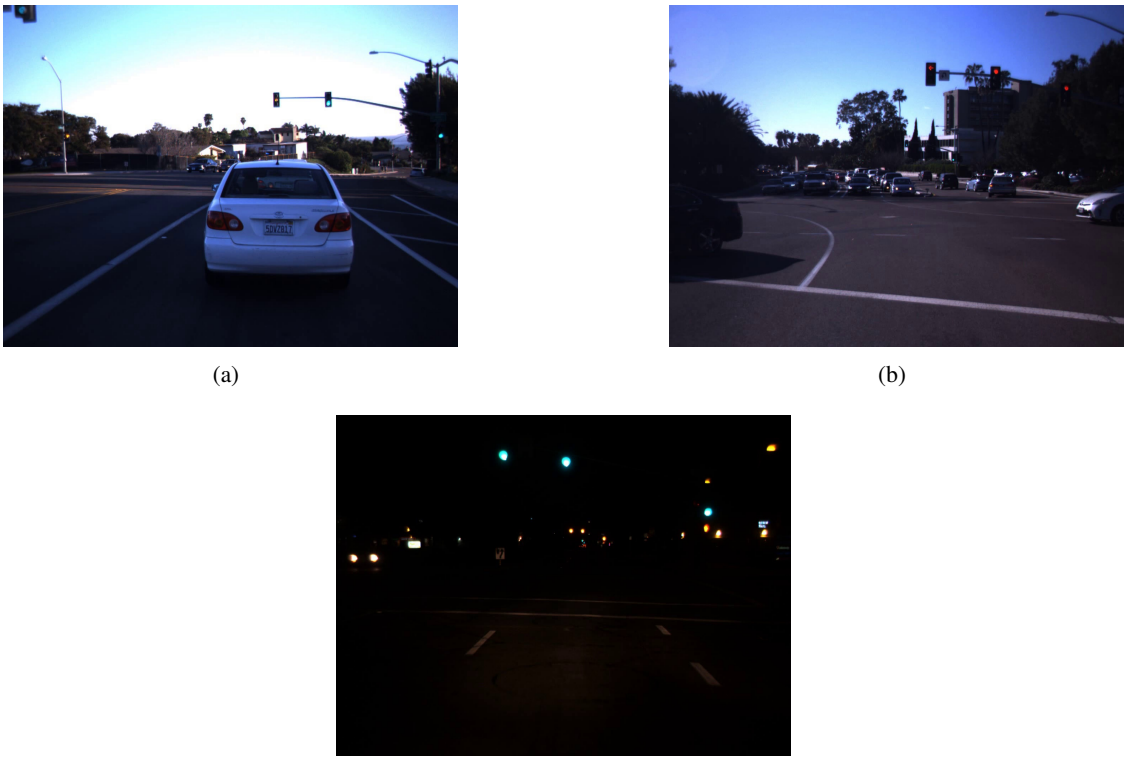
reach certain KPIs thresholds was lower. Another advantage given by this approach is the lowering of the dataset requirements, as training the model to adapt to our situations and environment is lessened, mitigating some of the drawbacks of our dataset. So, knowing that car-centric implementations could be used as a starting point to "jump-start" training and reduce the computational and dataset requirements to achieve our objective, searching for an open-source implementation that had this very same purpose started. On searching for said implementation, and after reviewing a number of state-of-the-art implementations in Chapter 3, algorithms like YOLO [33] were the most abundant, clearly due to its widespread reach not only in terms of performance (applicable to a wide range of detection problems) but also in terms of its availability. The latter means that given that, as YOLOs' source code and backbone are widely known and open-source, they have been the base of many projects, some of which might have fitted our needs. One specific implementation was chosen, the reason being its' well-structured code, information available on GitHub [101], application (car traffic lights) and performance, in inference speed, and KPIs.

Now that the advantages of transfer learning have been established, by detailing the kinds of models used and the reasoning behind using them, we proceed to define the actual baseline performance, which refers to the performance of the previous model and its implementation. The creation of the previous dataset will be briefly discussed, similarly to how our own was presented, and how it compares to ours. Additionally, we will explore the performance and its potential for transfer learning to our own implementation. Specific quantitative information regarding the performance of transfer learning will be reserved for the next chapter.

Beginning with the dataset used in the car-centric implementation, called LISA Traffic Light Dataset [70], in spite of some variations, such as the colour and geometric shape of the traffic lights when compared to tram scenarios, common idiosyncrasies and edge cases, as discussed in the previous sub-chapter, still exist. These will be enumerated in Chapter 6. Although the dataset, location, weather, and camera variables differ, the implementation of a similar detection problem for traffic lights would benefit from leveraging the knowledge gained from car-centric scenarios.

It is also worth noting that the original implementation for car traffic lights was significantly bigger than the current implementation in this project, spanning around 18000 images, all of them labelled, and that seems to have had an impact on our performance, as will be discussed in Chapter 5. Alongside the fact that the dataset used in the previous model contained a larger number of images and annotations, we also note that it had higher quality in terms of camera clarity, resolution, and dynamic range.

Another noteworthy aspect of the previous dataset that could prove beneficial in future implementations, depending on the real-world context and deployment, is the inclusion of diverse weather scenarios and patterns, as demonstrated in Figure 4.6. The previous dataset contains not only footage captured in optimal lighting conditions but also footage captured during the afternoon when visibility is significantly reduced, posing challenges related to dynamic range. Additionally, their dataset includes examples of bright light coming from behind or near the traffic light signs, further enhancing its advantages, as well as a significant portion of nighttime images and labels.



(a) (b)

(c)

Figure 4.6: Different weather scenarios of the LISA [70] dataset. In *a)* an example of bright light shining behind the car traffic light. In *b)* an example of normal lighting conditions. In *c)* an example of nighttime conditions.

Leveraging this dataset, the previous model was trained for 65 epochs using a specific combination of parameters. The achieved performance was commendable, in terms of precision, accuracy, and *F1* score. One notable improvement was the absence of high-frequency noise detections that appear for very brief periods and in a limited number of frames, lacking **temporal coherence**

across frames. These noise detections do not align with our desired targets and were one of the major hurdles that had to be overcome with the process of transfer learning. Usually, this noise also lines up with edge cases of Figure 5.11. All signs of this performance on only 65 epochs point to this being due to the vastness of their dataset.

These findings culminated in a final validation evaluation of 76% precision, 91% recall and 89% mAP when deploying the model, as the average of the last 10 epochs. However, it is crucial to acknowledge that these results were measured in the validation splits of the training dataset and may not be fully representative of real-world testing environments. This criticism is valid, as the validation images share similar characteristics with the training images used, as is usual with training and validation splits.

Regarding other important aspects of this open-source model implementation such as data augmentation, hyper-parameter tuning, image scaling and resizing. These were present in this implementation and contributed to the choice of this specific algorithm for training, as crucial and valuable time was saved by not having to develop these tools but only refine them to achieve the objectives proposed.

All that was detailed in this section points to this being a good starting point for transfer learning, as the model accurately detects traffic lights that share similar features with the targets determined for this project as well as avoiding edge cases that were not only evident from the initial analysis of the dataset, as explained in Chapter 3, but also in avoiding edge cases that were discovered when doing transfer learning. As such we proceeded with training from the last epoch of the original implementation, that is, the weights provided by the best-performing model were used as the initial weights in our training.

4.2.2 Traffic light recognition pipeline

With the decision to do transfer learning and the dataset source ready, work began on the construction of the pipeline, with the aim of labelling, transforming, augmenting and feeding the images to the training process, receiving them on the output, treating the images, detections and analysing and manipulating the data so as be able to derive conclusions and meaningful information and improving the models performance on this new task, that of recognizing tram traffic lights.

The implementation chosen as a base for this project had already the structure and ability not only to train, but also to detect targets in images and, although it did not have all the statistics, data analysis and features that were deemed necessary for this project, working on top of it enabled a narrower focus on the training so as to achieve the best performance. Additionally, it also included the ability to modify the output data to fit our needs and draw our own conclusions. As shown in Figure 4.7, the pipeline already had the foundation to be able to do the work of this project, but to do it effectively with our dataset, some parts needed to be added and modifications needed to be made to best inform the user on how to proceed with the next training. For this, the implementation of open-source libraries like Pytorch [102] and OpenCV [103], in Python, that enable not only the

usage of accelerated computing (on an *RTX 3060* with 12Gigabytes (GB) of memory)[1]) when training and detecting, bringing down the estimate of training time from close to 8 hours per epoch to 4 to 8 minutes per epoch depending on the resolution chosen for the training, but also facilitating the treatment and handling of data. Still, and even if there was a learning curve due to not having worked in Python before, its' structure and the vastness of resources and libraries served as an accelerator not only in modifying the required parts but also in understanding the model and its' inner workings.

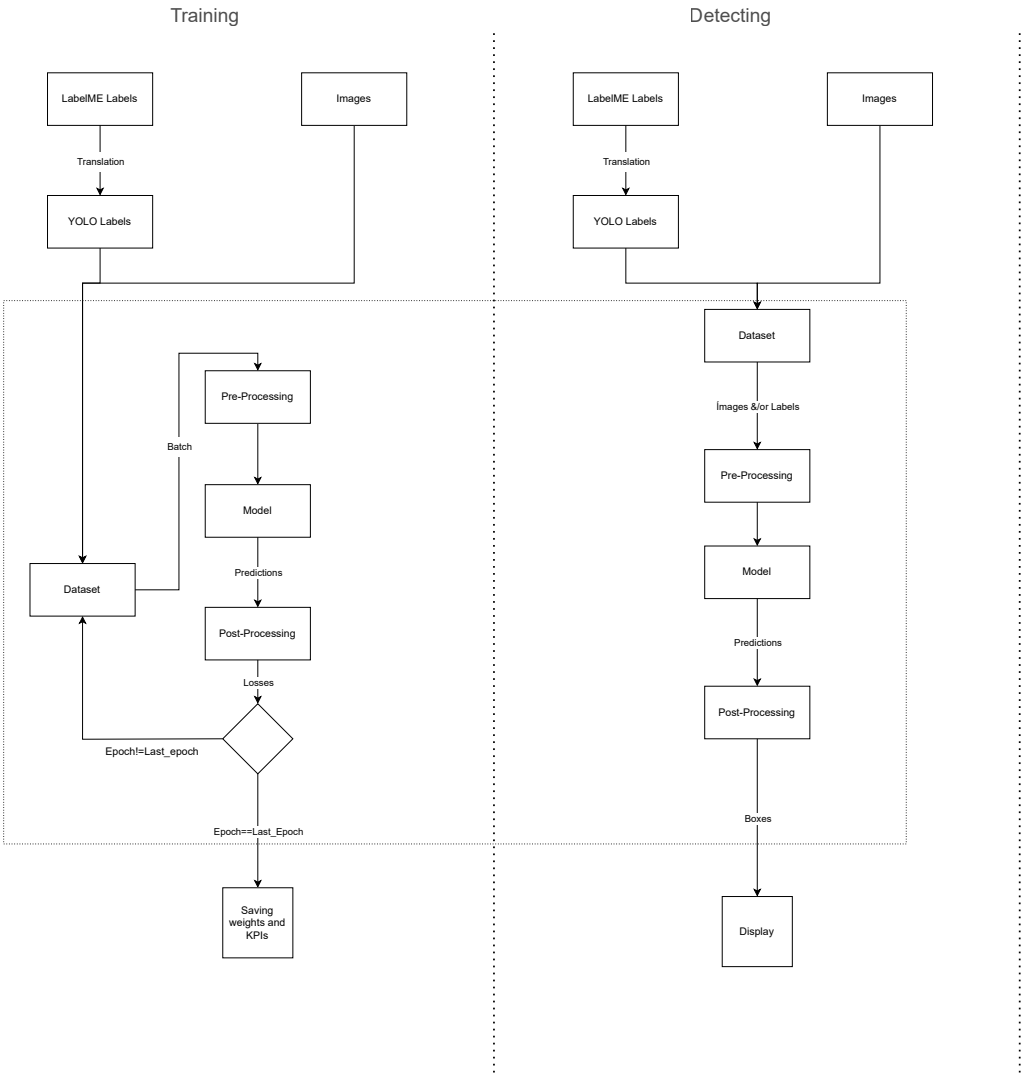


Figure 4.7: Overview of the main pipeline relevant to this project.

4.2.2.1 Labelling and dataset manipulation

Starting at the beginning of the pipeline, this section of the overall pipeline is in accordance with Figure 4.7 and deals with the labelling used, the tool, guidelines for it as well as the transformations

[1] https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3060-3060ti/

done to the data and its' input for any training or detection that is to be done.

The starting point for the whole pipeline is, therefore, some source of video data, which is in this case, as explained in the dataset section 4.1, videos from YouTube that record the cockpit/driver view on Berlin tram lines in Germany. With the data present in our local machine, a small script was developed to separate the videos into frames, frames which all followed the nomenclature present in Figure 4.8.

$$Filename\_frame\_number.extension$$

Figure 4.8: The nomenclature of the images and labels created in this dataset.

This was done so as to avoid conflicts and to be able to easily follow specific frames that contain important information, i.e. edge cases, or even just "simple" frames, when, as it would inevitably happen, some part of the pipeline was not yet understood.

The frames mentioned above serve as the basis for labelling, using the LabelMe [104] tool, which in this case was specifically installed using the python package manager pip so as to facilitate the usage of the tool. This tool is quite well-known and used in other projects, as it allows the user to label quickly and easily, even though it doesn't have more advanced features, such as automatic labelling targets. This tool was used due to time constraints on images that were collected every 25 frames, or every second when dealing with these specific YouTube videos of 25 FPS, resulting in the aforementioned 1800 labelled images, of a total of 8000 for training and validation.

This specific tool, LabelMe [104], outputs only *.json* files, with the same name as the image where the annotation was done, and a very specific and consistent structure. This structure consists of several keys inside a dictionary, the most important of which is the "shapes" key. This "shapes" key has within itself the information of all the bounding boxes that were annotated in the image inside "points" in the format seen in Figure 4.9.

$$[X\_top\_left \quad Y\_top\_left] \quad [X\_bottom\_right \quad Y\_bottom\_right]$$

Figure 4.9: The format in which bounding boxes are saved as they are saved in a label file.

Alongside this information exist the "label" key that indicates the specific label associated with that bounding box. An example of the information inside a specific *.json* file can be seen in Figure 4.10.

When using this tool to label all the images, specific guidelines were followed, these included:

- Drawing the bounding box around the perimeter of the casing of the tram traffic light, so as to not include the background too much which could have negatively impacted the performance.

```
{
  "version": "5.1.1",
  "flags": {},
  "shapes": [
    {
      "label": "f1",
      "points": [
        [
          325.1226993865031,
          185.88957055214723
        ],
        [
          358.2515337423313,
          286.196319018404 9
        ]
      ],
      "group_id": null,
      "shape_type": "rectangle",
      "flags": {}
    }
  ],
  "imagePath": "..\\raw_frames\\strassenbahn_berlin_2020_linie_60__001025.jpg",
  "imageData": "...",
  "imageHeight": 720,
  "imageWidth": 1280
}
```


Figure 4.10: Example of a *.json* file containing the details of a label from LabelMe [104].

- Labelling the signal from the moment it was clear to the labeller that an object was a tram traffic light, even if this meant small labels of faraway traffic lights. This helped further the goal of this project by detecting lights from farther away, which could result in an earlier warning for the driver, even if this might have resulted in less confidence in some detections that included smaller bounding boxes and objects.

Having the data labelled and when being in possession of all the label files and corresponding images, in order to advance to the input pipeline of the model, the LabelMe labels must first be converted into a format that can be received by the training or detecting algorithm. This means not only prepping the images and labels by moving them to the required directory, but also converting the bounding box and label name format of the LabelMe [104], that is, as previously mentioned, the coordinates of the top left and bottom right corner, into the specific format that removes all unnecessary information and leaves only the label name converted to a number, such as transforming *F0* into 0, *F1* into 1, and so on; and the coordinates of the centre of the bounding box, followed by the respective width and height of said bounding box. All these values, except the label, are expressed as relative to the width and height of the image, so using a bounding box with centre exactly in the centre of the image and label zero results in these contents of the text file appearing as in Figure 4.11.

$$0 \quad 0.5 \quad 0.5 \quad \frac{box\_width}{image\_width} \quad \frac{box\_height}{image\_height}$$

Figure 4.11: The translated format that is accepted by the input pipeline and is saved in a text file.

This information is converted and saved with one label and bounding box per line in the file. As with the *.json* files and frames (images), the naming convention mentioned in the beginning of this section is maintained, but this time the lines that contain the bounding boxes and labels translated from the LabelMe [104] annotation are stored in a text file in a specific directory that includes folders both from images and for labels.

The previous steps enable us to feed the data into the model, and this might be enough if the goal is to detect and compare detections with the ground truth (the labels that were translated in the previous step), but this in itself is not enough if training is the objective. To train this model, as is usually done in deep learning, a validation/train split must be done so that data can be extracted of how the model performed throughout the training process, using a certain number of images from the dataset to validate the performance at the end of each epoch. With this specific model not only does one have the ability to choose the percentage of images for training and validation, but the ability to choose what percentage of images for training and testing are unlabelled was also added. This introduces the ability to reduce the training time, without completely ignoring unlabelled images, ensuring, even if to a lower degree, that the model knows and learns what it shouldn't detect, by being exposed to scenarios where no labels exist. When looking in detail at the script that does the aforementioned functions, we learn that the script actually creates and modifies files that can be manipulated and have an impact on the model. The script creates "train.txt" and "val.txt" files, which contain the paths to the images to be used in training and validation, respectively. Note that to reach the corresponding label files to each image, it is as easy as changing the last directory from "images" to "labels" and the extension from that of an image to that of a text file since the images and labels share the same name/nomenclature. Additional files are also used, such as files that have the paths to the previously mentioned ones, corresponding to files with a ".data" extension, as well as files (with the extension ".names") containing the names that are to be used and that need to be compatible with the order chosen to translate the LabelMe [104] annotations into the model labels, as previously mentioned.

For most of the training, although other configurations were explored, this resulted in the usage of 15% of all the images being used for validation, with the rest used for training. In addition to this, a completely different tram line, that is, a different video, was used to test the models not only after finishing training but also at the end of each epoch, ensuring that the progress of the model can be measured and accompanied throughout training and giving even more data points for consideration when changing hyper-parameters.

4.2.2.2 Model input pipeline

Advancing now to the input of images and labels to the model, the process by which images are read, augmented, resized, scaled and fed into the model will be explored, with the focus on providing a holistic understanding of not only the "life" of a single image, and corresponding label, when being loaded for detection or training but also how the dataset and data loader are created and used to train and detect.

The first step in this input pipeline, more specifically in the "pre-processing" stage of the figure, is loading the images into the dataloader. This is achieved by first creating the dataset object, which is an instance of the *LoadImagesAndLabels* class. This class has all the necessary functions and inherent variables that allow for all the processes we have talked about before. When calling the constructor of this class the arguments given are: the path that comes from the ".data" files, mentioned before as containing the paths for the images for training and validation

and the location of the *.names* file, as well as batch size and some other less relevant parameters, we can image by image and label by label, or in batches if the batch size is higher than 1, build the dataset, by storing the training image paths and load the labels it into memory (the option to cache images can be activated as an argument when running the script). Secondly, we create the dataloader object, from the torch library [102], through the dataset object created previously, this dataloader is what will allow the algorithm to iterate through batches, getting the images of each batch and applying the necessary transformations required by the pre-processing, before sending them through the rest of the pipeline. So as to collect more information during training, leading to more informed decision-making, the functionality of repeating the aforementioned dataset and dataloader creation step for a testing dataset was introduced. This led to the ability to analyze the data of how the model performs as it trains and improves its' detection and classification capabilities.

Now that these steps were given that these objects are created, then, at each batch in each epoch, the algorithm can get the images present in the specific batch start by loading and caching the image, as is common with the OpenCV library, as a Blue;Green;Red (BGR) image, and then resizing it according to the input size we choose when running the script, this might mean down-sampling or up-sampling, although down-sampling is more common due to the increase in compute cost with higher images. Right after this, then another conversion on the labels is done, re-scaling them and transforming them into the format of Figure 4.12.

$$X\_top\_left \quad Y\_top\_left \quad X\_bottom\_right \quad Y\_bottom\_right$$

Figure 4.12: The original format that is accepted by YOLO and is saved in a text file.

Lastly, all the colour and space transformations are done, with these being controlled in intensity by the parameters that are present and modifiable in the source code. The image is converted to RGB and its' values are scaled so that their values are converted from [0;255] into [0;1], this being a common technique and transformation when dealing with CNNs, as detailed in Chapter 2. Before entering the model the image object is also transformed into a tensor, so that the model can correctly use it.

4.3 Training and reasoning

This section will delve into the various modifications implemented throughout the model's progression. Beginning by training from scratch, that is without any pre-trained weights, which set a baseline for performance by training from scratch. We then advanced to exploring involved transfer learning. From there, the sequence of parameter research iteration leading to the final version of the model will be explored. Note that specific performance metrics will not be provided in this chapter, as their purpose is to shed light on the decision-making process concerning hyper-parameters and other adjustments.

It is worth highlighting that additional features were incorporated to facilitate informed decision-making. Despite the model already possessing fundamental capabilities for extracting loss values and key performance indicators (such as precision, recall, mean average precision and F1 score), the tracking of these metrics throughout training, specifically with the testing of the model at the end of each epoch, was enhanced. Moreover, post-training analysis encompassing graphical resources was introduced to aid in comprehending and interpreting the model's behaviour.

4.3.1 Tunable hyper-parameters

Before discussing how the model was trained throughout the duration of this project, first, we will outline the various modifiable hyper-parameters and their impact on the model. These hyper-parameters can be adjusted to fine-tune the training process and optimize the model's performance.

When executing the training script or a detection script, the model can receive several parameters as arguments. Although there are more arguments available, we will focus on the ones that were specifically altered during training: the configuration file, epochs, and image size. An example command showcasing these arguments can be observed in Figure 4.13. Additionally, there is

$$python \quad train.py \quad --data \quad data/traffic.data \quad --batch \quad 2 \quad --cfg$$
$$<config\_folder>/<configuration\_file>.cfg \quad --epochs \quad 70 \quad --img-size \quad 32$$
$$--resume \quad --weights \quad <weights\_folder>/<weights>.pt \quad -- \quad <name>$$

Figure 4.13: Example of a command that runs the training script, with the necessary arguments.

a dictionary within the code containing several hyper-parameters that can be modified. The Table 4.1 illustrates these parameters, which include initial learning rates, final learning rate, and the intensity of augmentation in colour space and spatial augmentation.

Parameter	**Value**	**Description**
GIoU	3.54	Giou loss gain
Cls	2.805	Class loss gain
Cls_pw	1	Class BCEloss
Obj	64.3	Object loss gain
IoU_t	0.3	iou threshold
Lr0	0.003	Initial learning rate
Lrf	0.00005	Final learning rate
Momentum	0.937	SGD momentum
Weight decay	0.0005	Weight decay
Fl_gamma	0	Focal loss gamma
HSV_H	0.1	Hue augmentation
HSV_S	0.678	Saturation augmentation
HSV_V	0.4	Value augmentation
Degrees	0	Rotational augmentation
Translate	0.05	Translation augmentation
Scale	0.05	Scaling augmentation
Shear	0.641	Shear augmentation

Table 4.1: Table with values of the learning rate throughout training.

Let's now explore each of these hyper-parameters and their relationship with the model:

- The configuration file encompasses crucial model settings, such as the number and types of layers and their interactions. For our training purposes, an essential aspect of this file is the definition of anchors used by the YOLOv3 Model for prediction.
- The number of epochs, as explained in Chapter 2, refers to the number of times the model iterates over the entire dataset during training.
- Image size corresponds to the dimensions of the input image provided to the model. The dataset images may have different resolutions, but they are resized to fit the chosen resolution specified by this argument.
- The initial and final learning rates determine the starting and ending values of the learning rate during training. The behaviour of the learning rate throughout training plays a significant role and must be set by the user.
- The intensity of colour space augmentations controls the degree to which pixel values in the image can be altered positively or negatively before entering the model. A higher intensity value implies a larger range of saturation changes, as an example within the saturation channel.
- Spatial augmentations are similar but pertain to spatial transformations. The rotation parameter sets the degree range for image rotation, the translate parameter determines the interval for shifting the image horizontally or vertically, and the scale parameter controls zooming and cropping of the image. The shear parameter can distort the images based on the specified values during training.

  By adjusting these hyper-parameters, influence on the behaviour and performance of the model is obtained, enabling us to achieve the desired results in training.

4.3.2 Training iterations

Firstly, and as a baseline for what the training could achieve by itself with a dataset the likes of ours, the network was trained from scratch, that it without training with the original implementation's weights as a starting point. Note that this and the following 3 implementations employed the recommended parameters by the original implementation, with the learning rate taking the values of Figure 4.14. This shape maintained itself in the following training, being adapted only to accommodate the number of additional epochs.

Immediately after the previous training, the decision to employ transfer learning was solidified, making it possible to assess the potential benefits it could bring to the model. Initially, the aim was to determine the extent to which the original parameters could guide us without making any modifications. In other words, we sought to investigate how far the most basic parameter, the

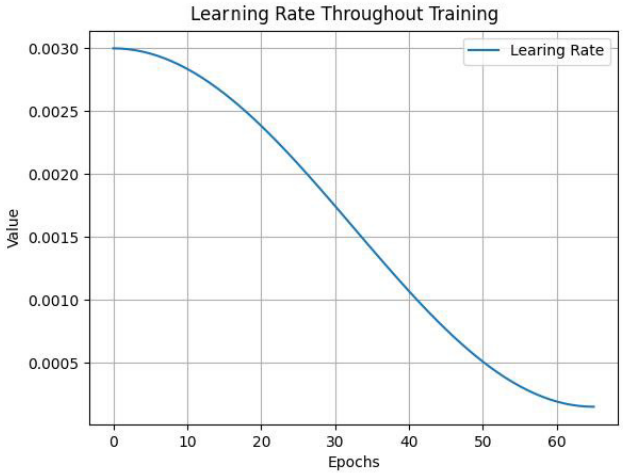


Figure 4.14: Values of the learning rate throughout training.

number of epochs, could influence performance. Our objective was to evaluate the model's capacity to acquire the distinctive features of tram traffic lights instead of car traffic lights and to assess the point at which knowledge transfer would reach a plateau, indicating diminishing returns.

To accomplish this, training started by using the original model's parameters employed in the previous implementation, maintaining the learning rates, that is, starting and finishing on the same value used during the training of the previous weights. As such, the first training was performed for 25 epochs, as it represented half of the duration of the original implementation' training. This duration was considered a reasonable initial estimate, as excessively prolonged training might lead to over-fitting.

After the training for 25 epochs with the original parameters was performed, it was thought that the network needed more time to acquire the capabilities necessary to start detecting tram traffic lights. Therefore, it seemed reasonable to proceed to training for longer epochs, in this case, 30 additional epochs. Note that on this last training of 55 epochs, the learning rate took the values on Figure 4.15, keeping them for all other iterations of training, except if otherwise stated.

Admittedly, this approach might not be the most efficient, due to only having explored 2 lengths of training, but it served as our baseline to establish a starting point and allowed us to reach a point where training was optimised, but did not take so long as to hamper the number of parameters being researched. Additionally, all training iterations after this one used this learning rate, except if otherwise stated.

After having a baseline established, and considering the needs of our dataset, changes to hyper-parameters would hone in on a crucial aspect of data augmentation, particularly in terms of spatial transformations.

With this in mind, all the available space transformations were enabled, including **rotation**, **translation**, **scaling**, and **shear**. Despite having suggested values, with those specific spatial augmentation values being multiplied by zero to nullify their impact in the original implementation,

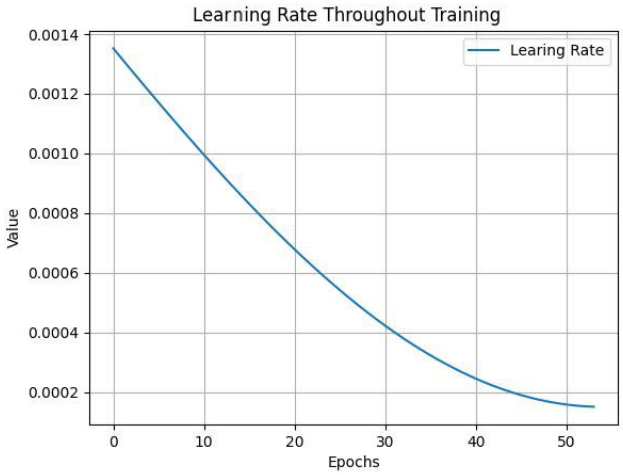


Figure 4.15: Values of the learning rate throughout transfer learning training.

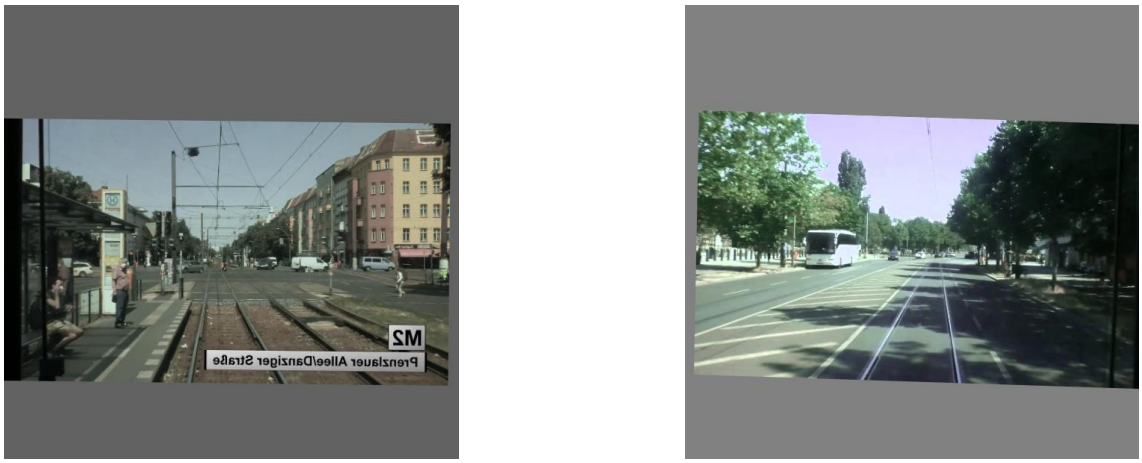
the original author chose not to suggest using these. Still, it appeared to be beneficial to incorporate these transformations within the training process.

The underlying rationale behind this training approach was to simulate a broader range of scenarios for the network to encounter during the detection of traffic lights, by feeding the network training images like those in Figure 4.16. By exposing the model to varied situations, such as curves or unconventional positions, and even detection at longer distances, by simulating situations like ones in which the tram is in an inclined plane, or when curving and encountering a tram traffic light at an angle.

Specific augmentation research into each of the 4 parameters activated in this training was not pursued, as other parameters appeared more promising. This was due to the fact that our tram traffic lights appear in similar physical positions to ours, and the already existing parameters should already have been studied as the best for the previous implementation.

Moving forward, our exploration delved into colour space augmentation, which involved manipulating the intensities of various channels within the HSV colour space utilized to augment the images. Although the original author had this specific augmentation technique enabled, optimizing the values of these augmentations for our specific scenario could be beneficial.

The purpose behind this augmentation was to equip the network with enhanced capabilities in detecting edge cases. For instance, by altering the hue, saturation, and value values, we aimed to address challenges like reflections, which often posed difficulties for accurate detection. Through modifications in hue, the network could potentially perform better in different lighting scenarios, adapting to varying times of day and weather conditions. Similarly, adjustments in value could tackle issues related to dark or crushed areas, as in the edge case linked to tree branches. It might also help mitigate the prominence or intensity of reflections. Finally, adjustments in saturation might help the network decouple from the detection of car traffic lights in at least two ways: the first involves green traffic lights, which sometimes can appear whitish at longer distances, where enhancing saturation might help; the second one is the lowering of saturation, that might help the

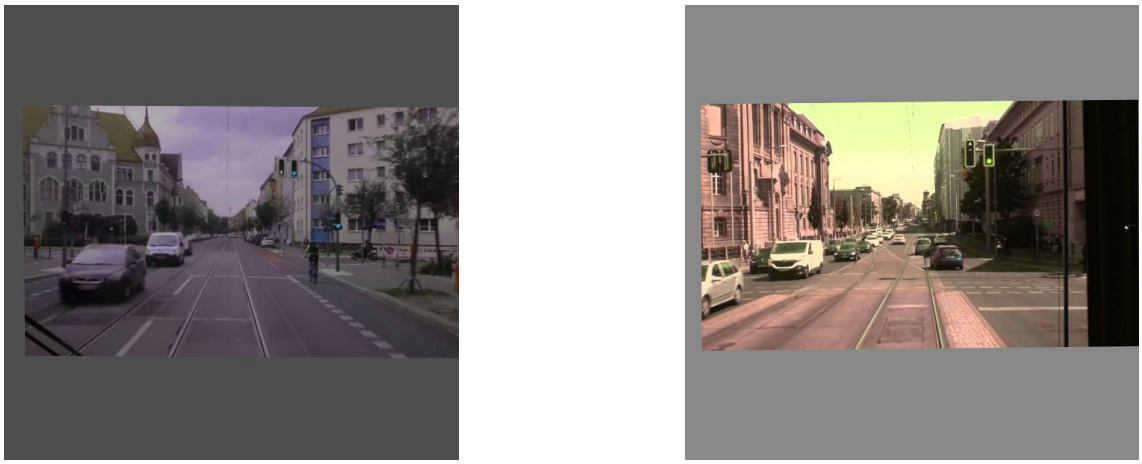


(a) (b)

Figure 4.16: Examples of images fed to the network with spatial augmentations. Both *a)* and *b)* have had augmentations applied.

network focus more on contrasting and extracting knowledge from the shapes of the car traffic lights, as they would appear grey/white just like tram traffic lights.

While doing this augmentation, it is worth noting that increasing the value of, *e.g.*, the saturation parameter does not mean that all images will be more saturated, it means that the interval of intensity transformation of the saturation of an image widens. Examples of the application of colour augmentation can be seen in Figure 4.17. To determine the optimal levels of intensity



(a) (b)

Figure 4.17: Examples of images augmented in the HSV colour space.

variation, we explored different values across all the parameters. Specifically, we experimented with values such as 0.2, 0.3 and 0.4, simultaneously, across all three parameters. By assessing the performance under each variation, we sought to identify the intensity level that yielded the most favourable results. Another possibility was also tested, where the original parameters were tested,

specifically: 0.678 in *saturation*, 0.1 in *hue* and 0.4 in *value*.

Continuing with our exploration of data set augmentation, an interesting observation was made regarding the original implementation's augmentations in the domain of rotation. It appeared that the bounding box failed to accurately encompass the traffic lights, contradicting our established labelling guidelines, by encompassing too much of the background. Take a look at Figure 4.18 to get a visual understanding of this behaviour. Specifically, this discrepancy was noticed when rotation was applied.

(a) (b)

Figure 4.18: Comparison of the label on an image with rotation, in *a)*, and one without, in *b)*.

Given this issue, the next study would focus on the potential impact of excluding rotation in our training process, as it had been activated alongside other spatial transformations thus far. This test allowed us to assess whether rotation played a negative or positive role in the model's performance.

Entering now the later stages of model training, we once again study a crucial aspect of training: the learning rate. Upon closer examination, it was suspected that the values being utilized might not be ideal. Some investigation led to the conclusion that these values, even though they were the recommended ones might be hindering the model. As such, keeping the learning rate progression close to the original was thought to be a good trade-off between the maintenance of the original knowledge and the need to acquire new knowledge, that is context specific features. With this being established, the decision was made to start the training with a higher learning rate and start decreasing it only slightly in the first 10 epochs (following more of a *cosine* shape). In Figure 4.19 a provide a visual representation of the learning rate's trajectory throughout the training is provided, allowing for a meaningful comparison between the values employed in both the original implementation, in Figure 4.15, and our training, knowing that in the latter the best values for hyper-parameter found until now have been employed.

After going through changes in the training hyper-parameters, now something linked to Chapter 2 would be explored, more specifically, the YOLO model configuration. So, clustering algo-

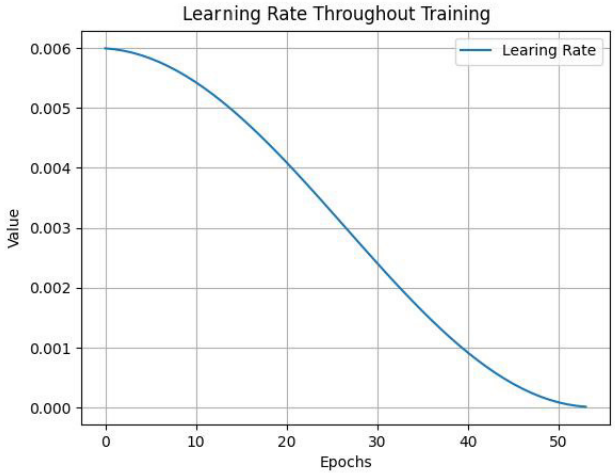


Figure 4.19: Learning rate of this specific iteration of training.

rithms were thought to be a good fit for our objective, having been used by multiple researchers when dealing with YOLO [62], we employed a *K-means* algorithm on our dataset's labels. The goal being to extract the top nine bounding box clusters centroids that capture the idiosyncrasies of our traffic light scenarios. These extracted sizes, available in Figure 4.20, were then integrated into the model's configuration, effectively equipping the model with a solid reference point for its predictions.

This change had the potential to have a meaningful impact on performance, by grounding the model's predictions in sizes that truly represent the real-world situations the cameras of our dataset encountered. The expectation was that this change would manifest itself in several key areas: enhanced detection capabilities at longer distances, a reduced incidence of flickering edge cases, and greater precision in sizing the detected traffic lights, which would better the filtering by IoU. With this, this iteration aimed to not only elevate performance but also fortify our detection in accordance with our labelling guidelines, and, as such, hoped to see a surge in the quality and reliability of our detection outputs.

Yet again, additional changes to better adaptability to our targets were made, this time, another alteration regarding the learning rate. In this new strategy, the learning rates decreased exponentially like in Figure 4.21, starting from a lower initial value compared to the previous training. The rationale behind this strategy was, once again, to guide the optimization process in a way that would enable the model to escape the local minimum (for car traffic lights) quickly in the beginning. Sequentially, rapidly decreasing it to focus on the optimization to tram traffic lights, by letting the model train for more time at a low-value learning rate.

By following this adjusted learning rate, the ultimate goal was that the model would gradually depart from its initial optimization landscape, avoiding stagnation, and progress towards a more favourable region of the parameter space that aligned with the characteristics of tram traffic lights. This approach, if successful, would enable the model to escape the limitations imposed by the previous optimization landscape and converge towards a global minimum or a highly promising

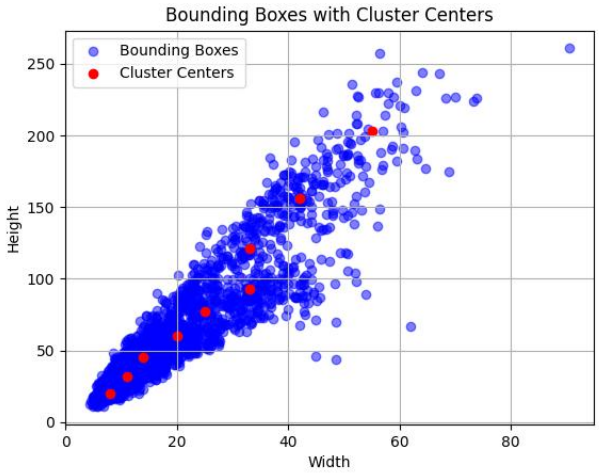


Figure 4.20: Clusters used to modify the configuration file of the YOLOv3 model.

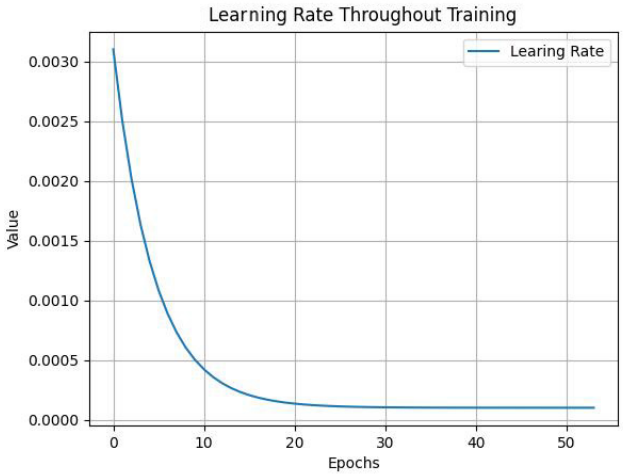


Figure 4.21: Figure depicting values of exponentially decreasing learning rate throughout training.

solution for detecting tram traffic lights.

In this final iteration of training, the best parameters discovered until now were combined. Using these optimised parameters meant going back to the clustering iteration's parameters, now adding to it one more thing: **resolution**. This conclusion was reached after going through multiple rounds of training and noticing a persistent weakness in the model's performance when it came to detecting tram traffic lights that were slightly distorted due to speed or ones present at great distances. For instance, it could aid in distinguishing between F0 signs and F1 signs, which often appear indistinguishable in lower-resolution (680p) footage, particularly when they are located at a distance. As such, the new increased resolution took the value of 1024p.

Beyond addressing the previously mentioned cases, the overall expectation was that increasing the resolution would lead to improvements across the board. By providing the model with a higher level of detail and overall more information. It was easy to anticipate that the model's

overall detection capabilities would also increase. Note that increasing resolution has a significant training time and computational weight, leading to longer training times, with *1024p* being a good trade-off between higher resolution and not as high a training time increase.

Lastly, in order to provide a comprehensive overview of the training iterations conducted, a table summarizing the various parameter modifications made throughout the process was compiled. Table 4.2, serves as a valuable resource for understanding the changes implemented and the reasoning behind them.

Nr.	Changes	Parameters	Reasoning
1	No changes to original parameters	Original Parameters	Test how much the model would achieve without changes.
2	Activation of spatial transformations	degrees=1.98; translate=0.01; scale=0.01; shear=0.641	Better generalization to other scenarios in which lights can appear (crooked, curves, etc)
3	colour space augmentations intensity	hue=[0.2;0.4]; saturation=[0.2;0.678]; hsv_v=[0.2;0.4];	Improving the ability to distinguish between car and tram traffic lights (saturation), improving the handling of reflections (value) and more time-of-day scenarios (hue).
4	Rotation deactivation	degrees=0	See if the bounding box not rotating with the image had a negative impact.
5	Learning rate increase	From Figure 4.15 to Figure 4.19	Applying the previous hypothesis of higher learning rates in the beginning, this leading to better convergence, to the shape of the original's implementation learning rate.
6	Apply K-means to dataset's labels	Model's configuration file altered.	Improve detection at a distance and overall having detections more adequate to our situations.
7	Changing learning rate	From Figure 4.21	Allowing the model to learn more features specific to tram traffic lights
8	Increasing resolution to 1024p	img_size=1024p	Increasing Resolution might help the model extract better features from more information across all situations.

Table 4.2: Table comparing all the training iterations done to the model, their changes and the reasoning behind them.

4.4 *C++* package

After training the model, a section of the work done in this thesis focused on creating a package in *C++* which could run the model in Open Neural Network Exchange (ONNX)-runtime [105]. ONNX is an open-source platform for artificial intelligence implementations with partnerships spanning many companies (*e.g.* NVIDIA [106] and AMD [107]). The core of ONNX operates by allowing developers to move between frameworks at will. The frameworks encompassed include Pytorch [102], TensorFlow [108] and many others. Allowing developers to switch between them benefits development due to some being more applicable in certain situations than others. Models can be exported from these frameworks into to a *.onnx* format and inference can be done using the

onnx-runtime environment. Such an environment where a model can run was created in order to allow our model to run in real-time. This meant creating an executable with *CMAKE* that included all dependencies necessary to ease the use of said executable. This process involved recreating the inference input pipeline, from loading images, to resizing them and changing them from RGB to BGR. Then, after the images are processed, they are fed to the model and the output is filtered according to a predefined confidence and IoU threshold, mimicking the process of the output pipeline of the PyTorch [102] implementation. This process was done in such a way as to ease integration into Continental's tools, although this integration fell out of the scope of this project.

Chapter 5

Results and Discussion

This chapter delves into a thorough analysis and discussion of the results obtained from the implementation of Chapter 4, which focused on the dataset, the training iterations and the reasoning behind them, with the examination also encompassing numerical KPIs. By combining these approaches, the aim is to provide a complete understanding of the implications and veracity of our reasoning, as well as the model's performance and its ability to accurately detect and classify tram traffic light signals in diverse scenarios.

5.1 Discussion of results

Firstly, it is important to mention that all tests conducted utilized the same set of parameters. These parameters refer to the confidence threshold, which filters the model's output based on the confidence level of each prediction, and the IoU threshold.

Additionally, it's worth noting that the visual analysis and KPI testing were performed using the same video clip. This clip was selected from the same playlist that generated the dataset, specifically from line *M5* of the Berlin tram system. This line encompasses both suburban and urban environments, featuring sections with both low and high interaction with active participants, as seen in Figure 5.1, in *a)* and *b)*, respectively. Moreover, it includes both dedicated tram lanes and sections where trams share the road with other vehicles. This selection allowed for a comprehensive evaluation of the model's performance across various scenarios and settings. A final important note before analyzing the data is that, when graphs regarding the overall KPIs are shown, these take into account all the classes, averaging their results. Classes that, due to their low volume in the dataset and in real scenarios, have performance that is close to zero, are also included. In these cases, the results are the byproduct of the amount of available data not being sufficient for the model to be able to generalize for these lights.

5.1.1 Iteration 1: baseline.

When looking at the first training iteration (the first row of Table 4.2), with the original parameters of the model, it was afterwards clear that simply learning the weights from scratch would not be

(a)

(b)

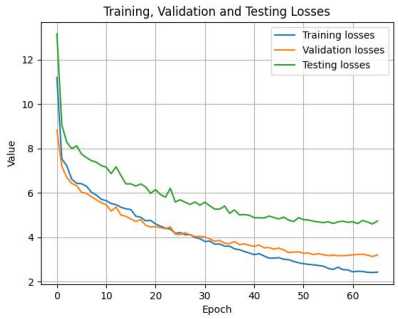
Figure 5.1: Example of low and high interactivity scenarios, *a)* and *b)*, respectively. Both images are from the *M5* Berlin line.

enough to provide acceptable results, furthering the cause of resorting to transfer learning. The numerical KPI results were significantly lower than the ones of the original work, as the algorithm did not perform adequately. Though, this might change if a larger and more robust dataset is used.

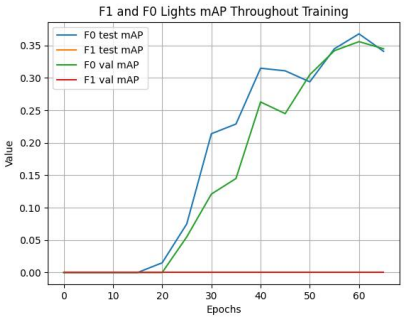
When analyzing Figure 5.2, it is evident that the loss values observed in the training, validation and testing processes did not reach low enough levels, despite following the expected pattern of descending order from high to low values. This observation is further supported by the fact that the most represented signs in the dataset, namely F1 and F0, only achieved positive results for F0 (the light with most instances in the dataset) in the testing and validation stages, reaching an average precision of approximately 35%.

However, when considering the overall mean average precision, as depicted in *c)* of Figure 5.2, it is in accordance with our reasoning when, even after 65 epochs, results point to a value of 7%. This performance is considerably low and does not meet the desired objectives, meaning that this specific iteration of the model cannot properly detect tram traffic lights. Consequently, it is reasonable to conclude that continuing in this path, training for more epochs with our dataset and the original parameters, would have a large computational expense and lead to lacklustre results, thus limiting the ability for generalization. The former, when juxtaposed with the advantages of transfer learning, led to the pursuit of the latter.

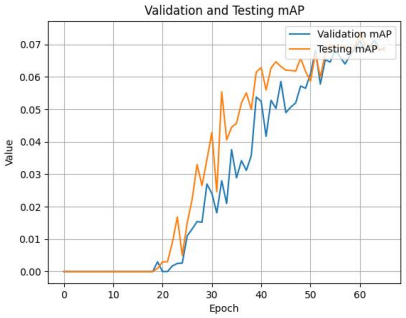
With the understanding that the current parameter configuration did not yield satisfactory results, it was decided to delve into transfer learning, as detailed in Chapter 4. Consequently, it was



(a)



(b)



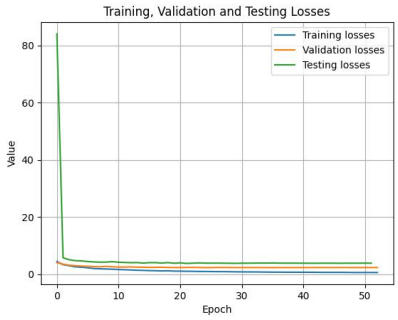
(c)

Figure 5.2: KPIs retrieved when training from scratch (iteration 1).

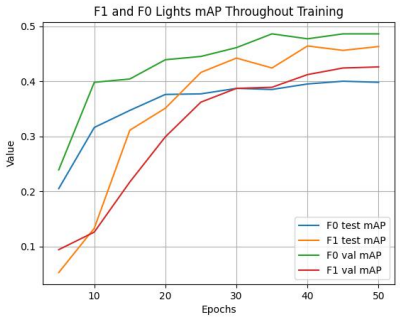
opted to utilize the same parameters as the original implementation, now extending the training process, adding epochs on top of the existing trained models' weights.

As such, this attempt at doing transfer learning employed 25 epochs for training, using the original implementation's weights as the initialized weights, achieving the results seen in Appendix A.1.1. Consequentially, as these results showed potential to improve and training was not too long (at around 1.5 hours), it was decided to prolong training for 30 additional epochs. The outcome is visible in Figure 5.3.

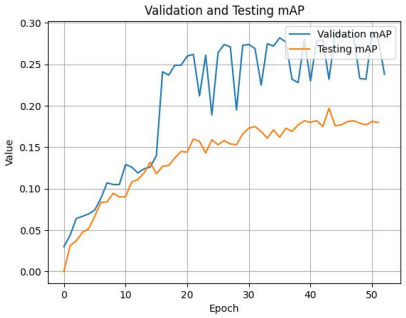
In this iteration, a more optimistic output was noticed, as some KPIs continued to increase. This resulted in a total of 55 epochs of training, that led to better levels in mAP. Additional KPIs can be analyzed in Table 5.1.



(a)



(b)



(c)

Figure 5.3: KPIs from training with transfer learning for 55 epochs (iteration 1).

		Precision (%)	Recall(%)	mAP (%)	F1 (%)
Validation	**Training from Scratch**	12.789	7.6080	6.8270	9.5220
	Transfer Learning	**27.820**	**32.340**	**26.409**	**29.720**
Testing	**Training from Scratch**	9.0780	8.1380	6.9490	8.5560
	Transfer Learning	**21.590**	**23.540**	**17.990**	**21.910**

Table 5.1: Table comparing the KPIs of transfer learning for 55 epochs with training the network from scratch (iteration 1).

When analyzing Table 5.1, a significant improvement can be found in all KPIs, with the most comprehensive one, mAP, achieving an increase of 11 percentage points, finishing with 17.99% mAP with a standard deviation of 0.00594. This enabled the conclusion that transfer learning should be pursued throughout the remaining iterations, however not discarding the need for other techniques to be applied, as the level of performance was still expected to increase and clear candidates for parameter tuning were still existent (potentially improving the model).

5.1.2 Iteration 2: spatial augmentation.

After determining that a new way of updating the learning rate did not bring any improvement, the focus was redirected to the other hyper-parameters available. As stated in Table 4.2, spatial transformations, concerning image augmentation, stood out as something that, when activated, could improve the generalization performance. As such, these were employed with the suggested values, while still keeping the other parameters equal, which until now meant the original parameters plus training for 55 epochs with the previous implementation's weights as a base.

This resulted in achieving an uptake across all KPIs, as seen in Table 5.2 and Figure 5.4.

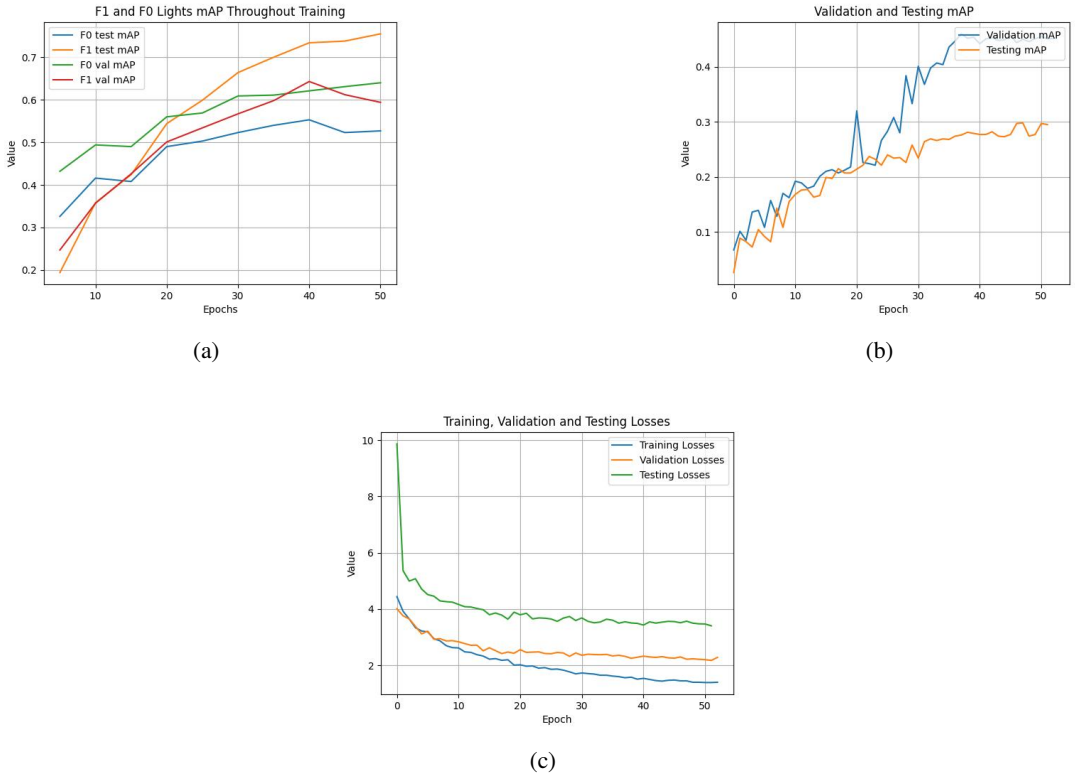


Figure 5.4: KPIs resulting from enabling space data augmentation (iteration 2).

When analyzing the graphs in Figure 5.4 it can be deduced that this change in results is mostly due to the improved generalization capacity that is introduced when using images with much more variability as training, achieving a significant rise in results. Still, when looking at the overall mAP, in *b)*, compared with the validation metric for mAP, the results start to diverge between epochs number 20 and 30. This is due to the drawbacks of our dataset, specified in Chapter 4, more precisely the lack of representation on *F2* through to *F5*. This is the case since the metric regarding overall mAP is the average of the mAP of all tram traffic light classes. Note that similar discrepancies will appear in the future, as this drawback manifests itself throughout training. Regarding *b)*, one can observe an increase of 12.49 and 29.2 percentage points in *F0* and *F1*, respectively. All other classes' mAP was zero or below 10 percentage points, which remains

in every iteration, except if otherwise stated. The previous considerations reinforce the analysis made for the dataset.

		Precision (%)	**Recall(%)**	**mAP (%)**	**F1 (%)**
Validation	**Transfer Learning**	11.383	7.768	0.05315	0.08992
	Current Iteration	**32.080**	**47.799**	**45.030**	**38.020**
Testing	**Transfer Learning**	21.590	23.540	17.990	21.910
	Current Iteration	**40.820**	**32.180**	**28.590**	**29.430**

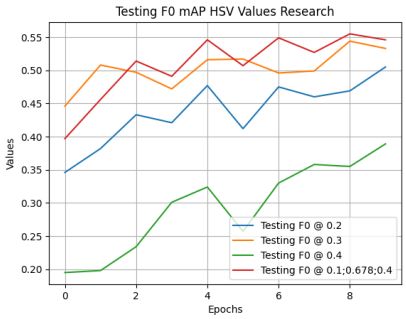
Table 5.2: Table comparing the KPIs of transfer learning for 55 epochs with the KPIs from the impact of enabling spatial augmentations (iteration 2).

Focusing in Table 5.2, a major increase in performance is noticeable, *e.g.* when looking at mAP, of 10.6 percentage points, with the mAP result having 0.013 standard deviation. From the last iteration to this one, this is, on average, a 2.65 point increase per parameter activated. Nevertheless, this is only an average and, due to other areas being thought to be more promising, more detailed training into the impact of each one did not materialize.

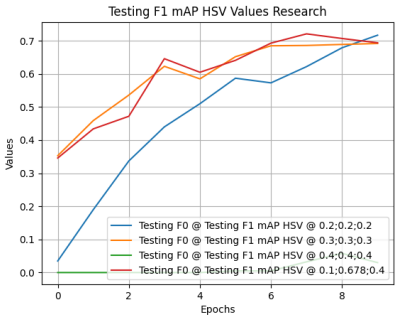
5.1.3 Iteration 3: colour augmentation.

Continuing with the rationale of Chapter 4, the next iteration focused on the realm of colour augmentation. When analyzing this, the development of the KPIs will be approached, specifically when they were all set to the same value and when the values were set to those of the previous implementation, comparing the graphs of the worst and best result, as well as the progression of the final results of each training.

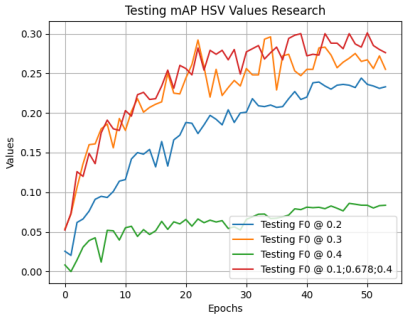
Instead of the graphs usually displayed in the previous iterations, in Figure 5.5 the progression between overall mAP and the AP of *F0* and *F1* lights throughout the different values of colour parameters explored is displayed. For each set of HSV values explored, the remaining graphs are present in A.3.



(a)



(b)



(c)

Figure 5.5: KPIs resulting comparing different values of colour augmentation (iteration 3).

The first iteration, which turned out to be the training's second to worst, was when all colour parameters were set to 0.4. This iteration was followed by one in the same terms but with a value at 0.2, with 0.3 and the original colour parameters following sequentially. This statement pertains to the average mAP, however, when looking at *F0* and *F1* lights, a deterioration of 1.4 percentage points was visible in the former and an improvement of 6.1 percentage points in the latter.

	Parameters	**Precision (%)**	**Recall(%)**	**mAP (%)**	**F1 (%)**
Validation	**HSV @ 0.2;0.2;0.2**	23.850	42.580	33.860	28.370
	HSV @ 0.3;0.3;0.3	47.289	62.310	57.179	71.870
	HSV @ 0.4;0.4;0.4	14.210	11.010	8.5900	10.804
	HSV @ 0.1;0.678;0.4	**42.160**	**61.700**	**52.240**	**58.449**
Testing	**HSV @ 0.2;0.2;0.2**	21.170	27.570	23.460	23.840
	HSV @ 0.3;0.3;0.3	26.639	32.960	26.530	28.999
	HSV @ 0.4;0.4;0.4	12.640	11.140	8.2400	9.7680
	HSV @ 0.1;0.678;0.4	**32.600**	**35.039**	**28.690**	**31.980**

Table 5.3: Comparison of the KPIs for the different colour augmentation parameters values (iteration 3).

When analysing Table 5.3, the last combination of HSV values (0.1; 0.687;0.4) achieved the best results from the values tested, improving 1 percentage point when compared with the previous iteration. Therefore, it was decided to continue with these values in the next iterations. The fact

that the best performance was achieved using default HSV values supports the hypothesis that these values are transferable from the car traffic light implementation to tram traffic light detection.

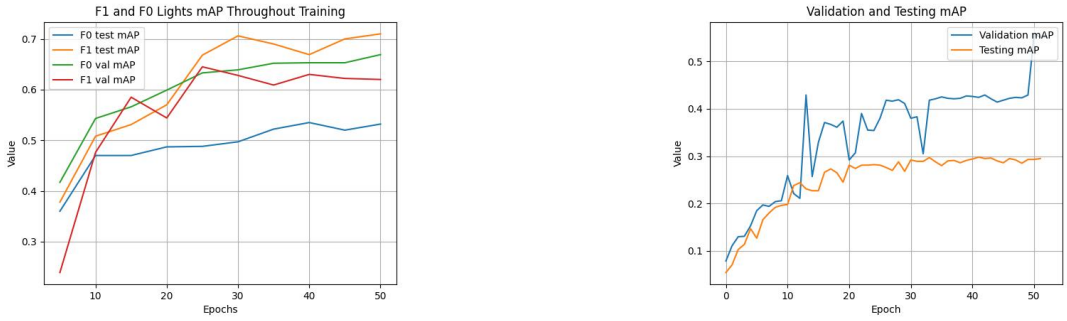
Furthermore, some erratic behaviour was observed when the parameter was set to 0.4, which is probably due to the high value of *hue* being too destructive to the ability to extract information from the image. At the same time, an increasing trend seemed to appear when testing the parameter values in the range of 0.2 to 0.3. These findings validated the hypothesis that higher values were required to accurately simulate a wider range of scenarios, just not across all channels of the HSV colour space.

Overall, as the training approach that yielded the best testing results incorporated a combination of parameter values used in the previous implementation, it is likely that these values had already undergone research and refinement, particularly in the context of car traffic light recognition, proving to be transferable to our scenario.

5.1.4 Iteration 4: the impact of rotation.

Having examined the impact of colour augmentation on our performance, the influence of rotation and its rationale is now considered, as previously discussed in Chapter 4. By analyzing the performance graphs in Figure 5.6, a similar trend appears regarding mean average precision along the 55 epochs, i.e., it diverges around epochs 20 through 30. From *a)*, a further 1.4 percentage points deterioration can be derived in *F0*, contrasted with a further improvement of 1.6 percentage points in *F0*.

When analyzing Table 5.4, a 0.479 percentage points increase is achieved in mAP (with 0.0036 standard deviation for mAP). It is challenging to determine conclusively whether this improvement can be attributed to our hypothesis regarding the non-rotation of bounding boxes with the image, thus adhering to the guidelines previously used for labelling. By carefully assessing the results and considering the practical implications, it was deemed appropriate to update the rotation parameters to those seen in this iteration. This approach ensures consistency and alignment with the established guidelines, while still resulting in slight refinements in performance.



(a) (b)

Figure 5.6: KPIs from training with rotation disabled in the hyper-parameters (iteration 4).

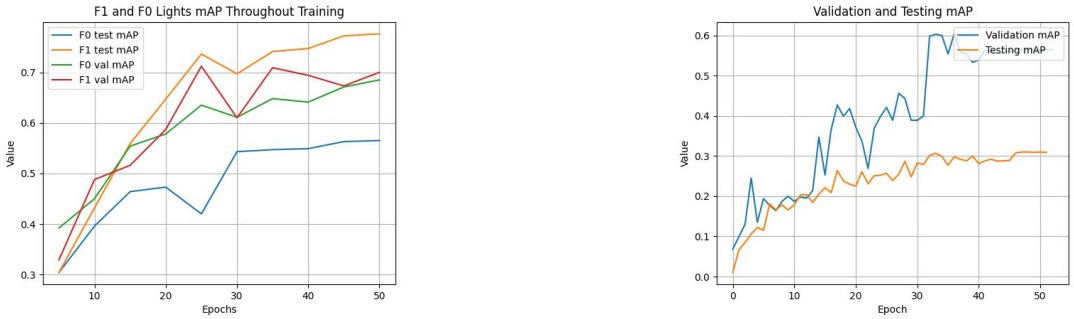
	Iteration	Precision (%)	Recall(%)	mAP (%)	F1 (%)
Validation	**Iteration 3**	42.160	61.700	52.240	58.449
	This Iteration	**32.799**	**52.940**	**46.109**	**38.309**
Testing	**iteration 3**	32.600	35.039	28.690	31.980
	This Iteration	**30.219**	**35.30**	**29.169**	**31.470**

Table 5.4: Table comparing the best KPIs of iteration 3, with the ones resulting from disabling rotation in space data augmentation (iteration 4).

5.1.5 Iteration 5: changing learning rate.

Continuing the exploration, the next step concerned the comparison between the previous scenario and the application of a different learning rate, with the learning rate of the former being showcased in Figure 4.15 and the new one taking the shape of Figure 4.19. This comparative analysis not only yielded an improvement in overall mAP, from 29.169% to 30.5% (with 0.010 standard deviation), but also demonstrated positive effects on other key performance indicators, as evident in the Table 5.5 and Figure 5.7. In *a)*, the best improvement was seen in the testing of the last 3 iterations, that is 3.3 and 6.61 percentage points in *F0* and *F1*, respectively.

This outcome further strengthens the outlined hypothesis of employing a slightly higher learning rate during the initial stages of training, followed by a consistent decrease throughout the remainder of the process. By adhering to this strategy, the targeted superior balance was achieved between rapid initial learning of new tram traffic light features and fine-tuning for optimal performance.



(a) (b)

Figure 5.7: KPIs from training with a new learning rate, seen in Figure 4.19 (iteration 5).

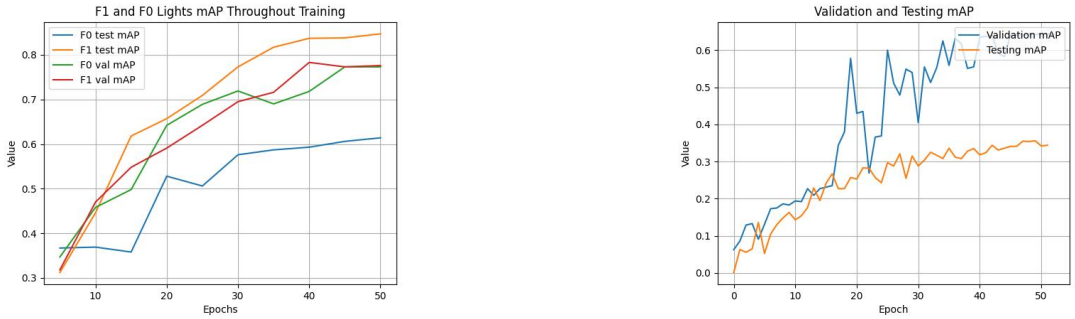
5.1.6 Iteration 6: applying clustering.

In this iteration, the model configuration was changed to use the bounding box sizes resulting from K-means as anchors (seen in Figure 4.20). This modification was in line with the reasoning in Chapter 4: a 4.1 percentage points increase in performance (with 0.0036 standard deviation in mAP results), as demonstrated in Table 5.6 and Figure 5.8. This is one of the most significant

	Iteration	Precision (%)	Recall(%)	mAP (%)	F1 (%)
Validation	**Iteration 4**	32.799	52.940	46.109	38.309
	This Iteration	**40.180**	**63.869**	**56.239**	**47.569**
Testing	**Iteration 4**	30.219	35.300	29.169	31.470
	This Iteration	**43.810**	**34.720**	**30.500**	**32.780**

Table 5.5: Table comparing the best KPIs from iteration 4, with new ones from training with the new learning rate of Figure 4.19 (iteration 5).

increases seen so far. Additionally, the third overall biggest improvement happens in the *F0* and *F1* categories, totalling 4.9 percentage points for *F0* and 7.1 percentage points for *F1*.



(a) (b)

Figure 5.8: KPIs from training with model clustering modifications of our labels (iteration 6).

	Iteration	Precision (%)	Recall (%)	mAP (%)	F1 (%)
Validation	**Iteration 5**	40.180	63.869	56.239	47.569
	This Iteration	**49.520**	**66.110**	**63.130**	**55.099**
Testing	**Iteration 5**	43.810	34.720	30.500	32.780
	This Iteration	**52.380**	**39.500**	**34.609**	**40.510**

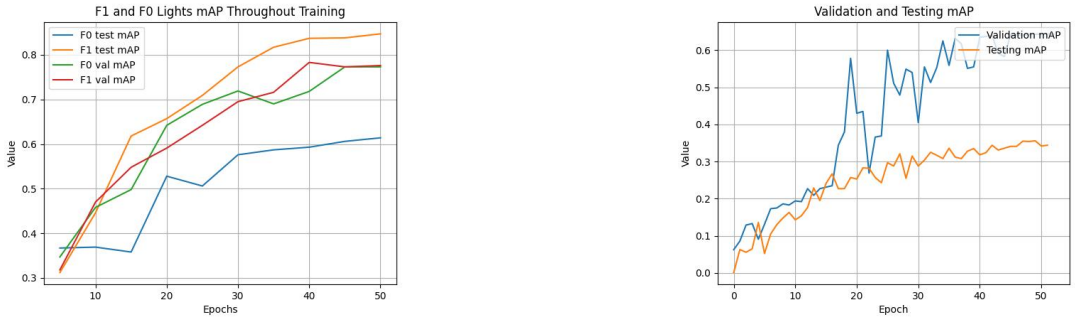
Table 5.6: Table comparing the KPIs of the impact of modifying the model with *kmeans* with the iteration 5 (iteration 6).

When looking further into the increase of mAP seen in Table 5.6, the improvement is due to not only the previously mentioned increase in testing *F0* and *F1*, but also from the significant impact clustering had in *F2* and *F3* tram traffic lights, which for the first time passed 10% mAP, achieving 11.3% and 20.5%, respectively.

We can conclude that applying this step had the biggest impact on performance since the activation of the 4 spatial augmentations, as was expected and explained in Chapter 4.

5.1.7 Iteration 7: changing the learning rate.

The results from implementing the adjusted learning rate strategy indicate that it did not yield the desired benefits. The decision to start with a higher initial learning rate and have it take such a sharp decrease did not effectively preserve the knowledge from the previous model nor facilitate the integration of tram traffic light features. Instead, this approach led to the destruction of some of the previously acquired knowledge, rendering it ineffective in achieving the intended improvements. This loss materialized itself as a 7.1 percentage point decrease in testing *F0* mAP and a major 18.4 point loss in *F1*, seen in Figure 5.9 *a)*.



(a) (b)

Figure 5.9: KPIs from training the model with learning rate modifications, seen in Figure 4.21 (iteration 7).

In Table 5.7, the mAP of all classes also saw a sharp drop of 29.150 points in precision, 11.800 points in recall, 15.371 points in *F1* and 10.859 points in mAP. Note that the value for standard deviation in the mAP result was 0.00618.

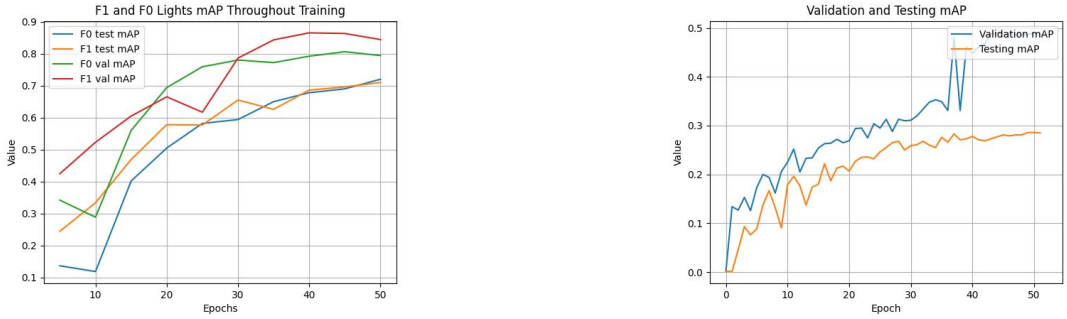
	Iteration	Precision (%)	Recall (%)	mAP (%)	F1 (%)
Validation	**Iteration 6**	49.520	66.110	63.130	55.099
	This Iteration	**30.010**	**39.980**	**36.760**	**33.430**
Testing	**Iteration 6**	52.380	39.500	34.609	40.510
	This Iteration	**23.230**	**27.700**	**23.750**	**25.139**

Table 5.7: Table comparing the KPIs of the impact of modifying learning rate from Figure 4.19 to Figure 4.21 (iteration 7).

5.1.8 Iteration 8: resolution increase.

When it comes to increasing resolution, we expected to see a significant increase in performance due to the higher degree and abundance of detail from which features can be extracted. Though this did not materialize. Note that, as the last iteration did not improve results, the KPIs of this iteration will be compared with the results from iteration 6.

In Figure 5.10 we can see how the KPIs evolved throughout training and, when comparing with the ones from Section 5.1.6, a significant head start starts to materialize itself in the initial epochs of the iterations of Section 5.1.6, which is carried through until the last epochs. In testing the *F0* and *F1* tram traffic lights an increased 10.6 percentage points for the former and decreased 13.7 for the latter is seen.



(a) (b)

Figure 5.10: Depiction of KPIs during training with an increase in resolution (iteration 8).

Contrasting with the results in Table 5.8, we can see that overall mAP results decreased from 34.609% to 28.220%, mainly due to the results from all other traffic lights decreasing back to 0 (with 0.0059 standard deviation for mAP results). This is somewhat in accordance with what we expected, as, at least in *F0* tram traffic lights, there was a noticeable increase which might be due to these lights appearing in greater number and from greater distances.

	Iteration	Precision (%)	Recall (%)	mAP (%)	F1 (%)
Validation	**Iteration 6**	49.520	66.110	63.130	55.099
	This Iteration	**32.380**	**60.210**	**48.769**	**41.590**
Testing	**Iteration 6**	52.380	39.500	34.609	40.510
	This Iteration	**16.570**	**32.960**	**28.220**	**21.939**

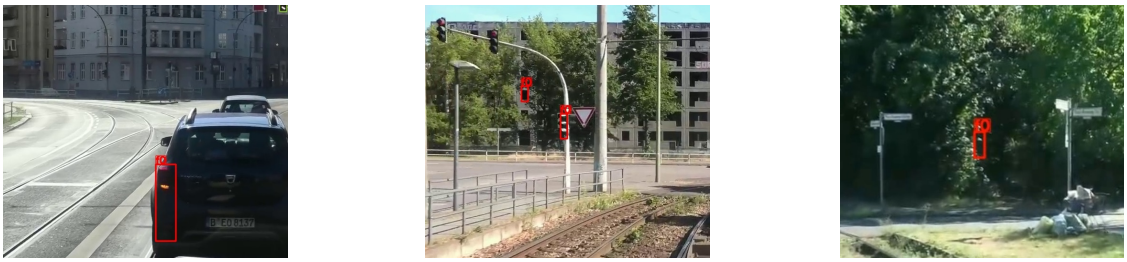
Table 5.8: Table comparing the KPIs of iteration 5, with those of training with a resolution of 1024p (iteration 8).

5.2 Overall conclusions from training results.

Having explored all the results of training iterations from Table 4.2, we conclude that the biggest increase in performance was due to applying transfer learning. Other notable improvements include iterations 2, 6 and 3, with an over 4% improvement. Lastly, the model achieving the best performance was that which resulted from iteration 6, having not only the best performance in *F0* and *F1*, but also in all other lights (except *F5* which remained at 0 in all iterations). This model combined the parameters including altered learning rates, clustering and colour and spatial transformations (a full list can be seen in A.6). If a model were to be used or improved upon, it

would be this one. Still, the results of iteration 8's model also had performance near iterations 6's, although only in *F0* and *F1*. Also, note that, when doing detection on the Python pipeline, the best model (and all that ran at a resolution of 608p) took approximately 16 milliseconds to perform inference in an image.

Additionally, edge cases were not totally eliminated, although their detection ratios improved substantially from the worst to the best parameter combinations found in this thesis. Still, when further exploring edge cases, during the analysis of the labelling process several **edge cases** were identified in the dataset. These edge cases posed remained after training, resulting in problematic detections, as shown in Figure 5.11. One common example was small pockets of light passing through tree branches (that are often black crushed areas), which the model often confused with the vertical or horizontal light of a traffic light for trams. Another identified edge case was the reflection of lights on the metal housings of the traffic light signals, which created the appearance of an *F0* sign. Some of these edge cases could be influenced by the process of transfer learning, *e.g.* white light shinning though trees. This could be explained by car traffic light detection algorithm benefiting from the lights for cars being coloured ones, which in most situations is captured differently from the mostly white light that shines through tree leaves and branches.



(a) (b) (c)

Figure 5.11: Examples of edge cases present in the dataset.

Overall it can be concluded that dataset limitations of tram traffic lights *F2*, *F3*, *F4* and *F5* were the main limitations, as all results point to them potentially being able to have better scores, provided that the dataset possesses more instances.

Chapter 6

Conclusion

With the increasing automation in car-centric scenarios and the evolving situations they encounter, it is crucial to extend the scope of research and development to encompass other modes of transportation and their respective environments. By leveraging existing knowledge and transferring it to new areas, advancements can be made more rapidly, enabling a safer and more comprehensive driving experience.

In this dissertation, the primary objective was to demonstrate the detection of tram lights, which ultimately was done through using a CNN based model. This served as proof of work, showcasing the feasibility of applying existing techniques to a new and specific scenario. The trained model could potentially be integrated into a decision-making system or a driver assistance system, empowering Continental to determine whether further pursuit is warranted.

To facilitate this proof of concept, a dataset was developed, considering the scarcity of publicly available data specifically tailored to tram traffic lights. This dataset added value to Continental as it represents a valuable resource for training and testing in other tram-specific scenarios.

This dissertation achieved the goal that was determined in Chapter 1 and demonstrated the potential of the developed model for detecting tram traffic lights, achieving results in the order of 52% precision, 39.5% recall and 34% mAP, though results from *F0* and *F1* were 61% and 85% mAP, respectively. This points us to the possible improvement opportunities in the dataset, model architecture and parameter combinations. These future enhancements would better the model's performance and robustness, enabling it to potentially be deployed in real-world scenarios with even greater efficacy.

Given these results it is possible to confidently say that the initially posed questions have had their answers found, that is: it is viable to use AI to detect tram traffic lights, although no publicly available models already in use in this target situation (ensuring the need for transfer learning). Regarding the drawbacks of said models, it is possible to point out the need for large and robust datasets, as well as the models susceptibility to edge cases.

In conclusion, this dissertation exemplified the application of an existing technique to the detection of tram lights, showcasing its viability and laying the groundwork for further advancements. The created dataset and the trained model serve as valuable contributions to the field,

supporting Continental's research on enhanced safety and efficiency in tram transportation. By continually improving upon this work, additional contributions could be done to the advancement of a less developed area of transportation .

6.1 Future works

Upon reflection of the work conducted and the results achieved, it becomes evident that there are areas for improvement in aligning our objectives with the outcomes. Specifically, our ability to detect tram light signals labelled as *F2*, *F3*, *F4*, and *F5* showed noticeably inferior performance compared to our detection of *F1* and *F0* signals, with the worst one having fewer instances, *F4* and *F5* lights. To address these shortcomings and enhance the detection of these signs, it is crucial to expand the dataset significantly, as discussed in detail in Chapter 4, gathering more instances of all lights mentioned. This could be done by increasing our labelling pipeline, or even adjusting it to fit into programs such as *Roboflow* [109], which enhance the labelling process by reducing labeling time (introducing this program in training and deployment could also be beneficial). In the future, the utilization of simulated digital twins and environments may significantly reduce the reliance on manual collection of datasets for labelling and testing. These digital replicas can accurately model real-world scenarios and generate diverse, high-quality data for training CNNs. As a result, the creation of large, diverse, and well-annotated datasets becomes more efficient, enabling CNNs to achieve improved performance and generalization capabilities across various applications, including specialized domains like tram TLR.

Continuing with training iterations and exploring the impact of hyper-parameters on model performance could also lead to further improvement. It would be worthwhile to investigate more combinations, such as finding optimal values for color space augmentation and spatial augmentation. Additionally, parameters like learning rates, with variations throughout the training process (*e.g.* halving the learning rate after a certain number of epochs), could also be explored for potential enhancement.

Another avenue for future research involves incorporating different models into the detection pipeline. By utilizing more recent models with improved feature extraction capabilities and architectures, like newer versions of YOLO or even different NN architectures (e.g. transformers), we may achieve better results in our detection tasks. Also, separating the classification and segmentation tasks within the model could allow the model to focus and perform better against some edge cases.

Finally, integrating our model into Continental's pipeline for possible utilization as input in an ADAS decision-making system. Exploring this integration would not only provide insights into the detection time and performance with their specific input data but also open up opportunities for further investigation.

Overall, focusing on addressing the aforementioned limitations, optimizing hyper-parameters, exploring newer models, and integrating the developed model into Continental's pipeline for ADAS decision-making represent important directions for future research.

References

[1] UITP. WORLD METRO FIGURES 2021. https://cms.uitp.org/wp/wp-content/uploads/2022/05/Statistics-Brief-Metro-Figures-2021-web.pdf, 2021. [Online; accessed 26-June-2023].

[2] European Commission, Eurostat, N Jere, L Corselli-Nordblad, E Ford-Alexandraki, and G Xenellis. *Key figures on European transport : 2022 edition*. Publications Office of the European Union, 2023.

[3] EUROPEAN COMMISSION. COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT, THE COUNCIL, THE EUROPEAN ECONOMIC AND SOCIAL COMMITTEE AND THE COMMITTEE OF THE REGIONS Sustainable and Smart Mobility Strategy – putting European transport on track for the future. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52020DC0789#document2, 2020. [Online; accessed 26-June-2023].

[4] UITP. STATISTICS BRIEF 1 URBAN PUBLIC TRANSPORT IN THE 21ST CENTURY. https://cms.uitp.org/wp/wp-content/uploads/2020/08/UITP_Statistic-Brief_national-PT-stats.pdf, 2021. [Online; accessed 26-June-2023].

[5] German Government. Ordinance on the Construction and Operation of Trams. https://www.gesetze-im-internet.de/strabbo_1987/anlage_4.html, 1987. [Online; accessed 26-June-2023].

[6] Robert Geirhos, David H. J. Janssen, Heiko H. Schütt, Jonas Rauber, Matthias Bethge, and Felix A. Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker, 2018.

[7] Vipin Kumar Kukkala, Jordan Tunnell, Sudeep Pasricha, and Thomas Bradley. Advanced driver-assistance systems: A path toward autonomous vehicles. *IEEE Consumer Electronics Magazine*, 7:18–25, 08 2018.

[8] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In Kohei Arai and Supriya Kapoor, editors, *Advances in Computer Vision*, pages 128–144, Cham, 2020. Springer International Publishing.

[9] Banhi Sanyal, Ramesh Kumar Mohapatra, and Ratnakar Dash. Traffic sign recognition: A survey. *2020 International Conference on Artificial Intelligence and Signal Processing, AISP 2020*, 1 2020.

[10] Mehrez Marzougui, Areej Alasiry, Yassin Kortli, and Jamel Baili. A lane tracking method based on progressive probabilistic hough transform. *IEEE Access*, 8:84893–84905, 2020.

[11] Kai Zhao, Qi Han, Chang-Bin Zhang, Jun Xu, and Ming-Ming Cheng. Deep hough transform for semantic line detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):4793–4806, 2022.

[12] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–, 11 2004.

[13] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 11 2004.

[14] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[15] Sang Hyuk Lee, Jung Hawn Kim, Yong Jin Lim, and Joonhong Lim. Traffic light detection and recognition based on haar-like features. *International Conference on Electronics, Information and Communication, ICEIC 2018*, 2018-January:1–4, 4 2018.

[16] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[17] Yang Ji, Ming Yang, Zhengchen Lu, and Chunxiang Wang. Integrating visual selective attention model with hog features for traffic light detection and recognition. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2015-August:280–285, 8 2015.

[18] Towards Data Science. Overfitting vs. Underfitting: A Complete Example. https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765, 2018. [Online; accessed 26-June-2023].

[19] Jeremy Jordan. Convolutional neural networks. https://www.jeremyjordan.me/convolutional-neural-networks/, 2017. [Online; accessed 26-June-2023].

[20] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

[21] Shaofeng Cai, Yao Shu, Gang Chen, Beng Chin Ooi, Wei Wang, and Meihui Zhang. Effective and efficient dropout for deep convolutional neural networks. 4 2019.

[22] Zeke Xie, Issei Sato, and Masashi Sugiyama. Understanding and scheduling weight decay. 11 2020.

[23] Kensuke Nakamura and Byung Woo Hong. Adaptive weight decay for deep neural networks. *IEEE Access*, 7:118857–118865, 7 2019.

[24] Towards Data Science. Introduction to Neural Networks. https://towardsdatascience.com/simple-introduction-to-neural-networks-ac1d7c3d7a2c, 2019. [Online; accessed 26-June-2023].

[25] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018.

[26] Hasty. Intersectiion Over Union. https://hasty.ai/docs/mp-wiki/metrics/iou-intersection-over-union, 2023. [Online; accessed 26-June-2023].

[27] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[28] Xin Qian and Diego Klabjan. The impact of the mini-batch size on the variance of gradients in stochastic gradient descent, 2020.

[29] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[31] IBM. What is gradient descent? https://www.ibm.com/topics/gradient-descent. [Online; accessed 26-June-2023].

[32] Ibrahem Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020.

[33] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[34] Leanconvnets: Low-cost yet effective convolutional neural networks. *IEEE Journal on Selected Topics in Signal Processing*, 14:894–904, 10 2019.

[35] Muhammad Tayyab, Fahad Ahmad Khan, and Abhijit Mahalanobis. Compressing deep cnns using basis representation and spectral fine-tuning. *Proceedings - International Conference on Image Processing, ICIP*, 2021-September:3537–3541, 5 2021.

[36] Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact cnns via collaborative compression. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6434–6443, 5 2021.

[37] Toshi Sinha, Brijesh Verma, and Ali Haidar. Optimization of convolutional neural network parameters for image classification. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, 2017.

[38] Huibin Zhang, Liping Feng, Xiaohua Zhang, Yuchi Yang, and Jing Li. Necessary conditions for convergence of cnns and initialization of convolution kernels. *Digital Signal Processing*, 123:103397, 2022.

[39] Patrick Neary. Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning. In *2018 IEEE International Conference on Cognitive Computing (ICCC)*, pages 73–77, 2018.

[40] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. THE MNIST DATABASE of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998. [Online; accessed 26-June-2023].

[41] Machine Learning Mastery. How to Choose Loss Functions When Training Deep Learning Neural Networks . https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/, 20231. [Online; accessed 25-June-2023].

[42] Towards Data Science. Confusion Matrix — Clearly Explained. https://towardsdatascience.com/confusion-matrix-clearly-explained-fee63614dc7, 2020. [Online; accessed 26-June-2023].

[43] Improved mask r-cnn for obstacle detection of rail transit. *Measurement*, 190:110728, 2 2022.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[45] Yu Liu, Baocai Yin, Jun Yu, and Zengfu Wang. Image classification based on convolutional neural networks with cross-level strategy. *Multimedia Tools and Applications*, 76, 04 2017.

[46] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, et al. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[47] Towards Data Science. Understanding CNN (Convolutional Neural Network). https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4, 2019. [Online; accessed 26-June-2023].

[48] Medium. Convolution Operation in CNN: . https://medium.com/analytics-vidhya/convolution-operation-in-cnn-a3352f21613, 2021. [Online; accessed 26-June-2023].

[49] Florentin Bieder, Robin Sandkühler, and Philippe C. Cattin. Comparison of methods generalizing max- and average-pooling, 2021.

[50] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[51] Baeldung. Neural Networks: Pooling Layers. https://www.baeldung.com/cs/neural-networks-pooling-layers#:~:text=Pooling%20layers%20play%20a%20critical%20role%20in%20the,and%20are%20mainly%20used%20for%20downsampling%20the%20output, 20231. [Online; accessed 26-June-2023].

[52] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2 2015.

[53] Shuteng Niu, Yongxin Liu, Jian Wang, and Houbing Song. A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence*, 1(2):151–166, 2020.

[54] Deepak Soekhoe, Peter Putten, and Aske Plaat. On the impact of data set size in transfer learning using deep neural networks. pages 50–60, 10 2016.

[55] Zhong Qiu Zhao, Peng Zheng, Shou Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30:3212–3232, 7 2018.

[56] A comprehensive review of object detection with deep learning. *Digital Signal Processing*, 132:103812, 12 2022.

[57] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:779–788, 6 2015.

[58] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 11 2013.

[59] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 2013.

[60] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2189–2202, 2012.

[61] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[62] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:6517–6525, 11 2017.

[63] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS:740–755, 5 2014.

[64] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[65] Medium. Non Max Suppression (NMS) . https://medium.com/analytics-vidhya/non-max-suppression-nms-6623e6572536, 2021. [Online; accessed 27-June-2023].

[66] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[67] Ross Girshick. Fast r-cnn, 2015.

[68] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 6 2015.

[69] Towards Data Science. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365 2020. [Online; accessed 20-June-2023].

[70] Mark Philip. LISA Traffic Light Dataset. https://www.kaggle.com/datasets/mbornoe/lisa-traffic-light-dataset, 2018. [Online; accessed 26-June-2023].

[71] Towards Data Science. Data Augmentation for Deep Learning . https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9, 2020. [Online; accessed 20-June-2023].

[72] Harriet L. Dawson, Olivier Dubrule, and Cédric M. John. Impact of dataset size and convolutional neural network architecture on transfer learning for carbonate rock classification. *Computers Geosciences*, 171:105284, 2 2023.

[73] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. Selected Papers from IJCNN 2011.

[74] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 8 2012.

[75] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. Traffic-sign detection and classification in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[76] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 9 2020.

[77] Jonathon Shlens. A tutorial on principal component analysis. 4 2014.

[78] Yanzhao Zhu and Wei Qi Yan. Traffic sign recognition based on deep learning. *Multimedia Tools and Applications*, 81:17779–17791, 5 2022.

[79] Moises Diaz, Pietro Cerri, Giuseppe Pirlo, Miguel Ferrer, and Donato Impedovo. A survey on traffic light detection. volume 9281, 09 2015.

[80] V. John, K. Yoneda, B. Qi, Z. Liu, and S. Mita. Traffic light recognition in varying illumination using deep learning and saliency map. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 2286–2291, 2014.

[81] Omid Nejati Manzari and Shahriar Shokouhi. A robust network for embedded traffic sign recognition. pages 447–451, 10 2021.

[82] Zhang Gan, Li Wenju, Chu Wanghui, and Su Pan. Traffic sign recognition based on improved yolov4. In *2021 6th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 6, pages 51–54, 2021.

[83] Citlalli Gamez Serna and Yassine Ruichek. Classification of traffic signs: The european dataset. *IEEE Access*, 6:78136–78148, 2018.

[84] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[85] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012.

[86] Hamed Habibi Aghdam, Elnaz Jahani Heravi, and Domenec Puig. A practical and highly optimized convolutional neural network for classifying traffic signs in real-time. *International Journal of Computer Vision*, 122:246–269, 4 2017.

[87] Nur Nabilah Abu Mangshor, Nor Syahirah Saharuddin, Shafaf Ibrahim, Ahmad Firdaus Ahmad Fadzil, and Khyrina Airin Fariza Abu Samah. A real-time speed limit sign recognition system for autonomous vehicle using ssd algorithm. *Proceedings - 2021 11th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2021*, pages 126–130, 8 2021.

[88] Artificial intelligence for obstacle detection in railways: Project smart and beyond. *Communications in Computer and Information Science*, 1279 CCIS:44–55, 2020.

[89] Oliver Zendel, Markus Murschitz, Marcel Zeilinger, Daniel Steininger, Sara Abbasi, and Csaba Beleznai. Railsem19: A dataset for semantic rail scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[90] Riccardo Gasparini, Andrea D'Eusanio, Guido Borghi, Stefano Pini, Giuseppe Scaglione, Simone Calderara, Eugenio Fedeli, and Rita Cucchiara. Anomaly detection, localization and classification for railway inspection. *Proceedings - International Conference on Pattern Recognition*, pages 3419–3426, 2020.

[91] Danijela Ristić-Durrant, Marten Franke, and Kai Michels. A review of vision-based on-board obstacle detection and distance estimation in railways. *Sensors*, 21(10):3452, May 2021.

[92] Obstacle detection of rail transit based on deep learning. *Measurement*, 176:109241, 5 2021.

[93] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32:1231 – 1237, 2013.

[94] Tao Ye, Zhihao Zhang, Xi Zhang, and Fuqiang Zhou. Autonomous railway traffic object detection using feature-enhanced single-shot detector. *IEEE Access*, 8:145182–145193, 2020.

[95] Yuchuan Xu, Chunhai Gao, Lei Yuan, Simon Tang, and Guodong Wei. Real-time obstacle detection over rails using deep convolutional neural network. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pages 1007–1012, 10 2019.

[96] Wentao Liu, Zhangyu Wang, Bin Zhou, Songyue Yang, and Ziren Gong. Real-time signal light detection based on yolov5 for railway. *IOP Conference Series: Earth and Environmental Science*, 769:042069, 05 2021.

[97] Huanmin Wang, Huayan Pei, and Jianbai Zhang. Detection of locomotive signal lights and pedestrians on railway tracks using improved yolov4. *IEEE Access*, 10:15495–15505, 2022.

[98] Teresa Pamuła and Wiesław Pamuła. Detection of safe passage for trains at rail level crossings using deep learning. *Sensors*, 21:6281, 09 2021.

[99] Georgios Karagiannis, Søren Olsen, and Kim Pedersen. *Deep Learning for Detection of Railway Signs and Signals*, pages 1–15. 01 2020.

[100] esbek2. Straßenbahn Berlin. https://www.youtube.com/playlist?list=PLt0E6NtaD-P64W8xMNjLwwhipvX3ko0i7, 2018. [Online; accessed 6-March-2023].

[101] sovit 123. Real Time Traffic Light Detection using Deep Learning (YOLOv3). https://github.com/sovit-123/Traffic-Light-Detection-Using-YOLOv3, 2020. [Online; accessed 6-March-2023].

[102] Pytorch. Pytorch. https://pytorch.org/. [Online; accessed 12-June-2023].

[103] OpenCV. OpenCV. https://opencv.org/. [Online; accessed 12-June-2023].

[104] LabelME. LabelME. https://github.com/wkentaro/labelme. [Online; accessed 12-June-2023].

[105] ONNX. Open Neural Network Exchange. https://onnx.ai/, 2017. [Online; accessed 26-June-2023].

[106] NVIDIA. NVIDIA. https://www.nvidia.com/. [Online; accessed 12-June-2023].

[107] AMD. AMD. https://www.amd.com/en.html. [Online; accessed 12-June-2023].

[108] TensorFlow. TensorFlow. https://www.tensorflow.org/. [Online; accessed 12-June-2023].

[109] Robolflow. Robolflow. "https://roboflow.com/, 2020. [Online; accessed 26-july-2023].

Appendix A

Extra Results

A.1 Iteration 1

A.1.1 Training for 25 epochs

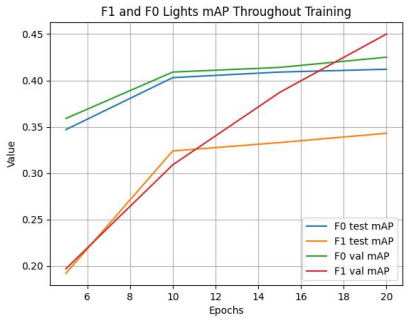
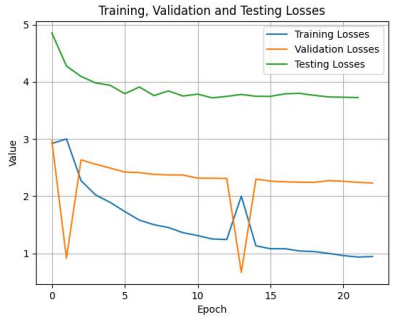
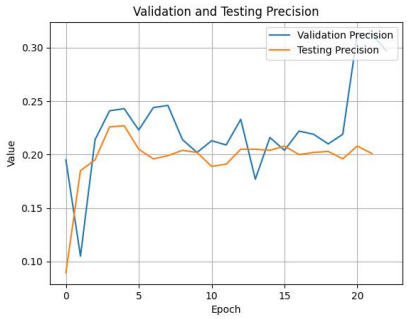


Figure with *F0* and *F1* throughout training and validation for iteration 1 (25 epochs).

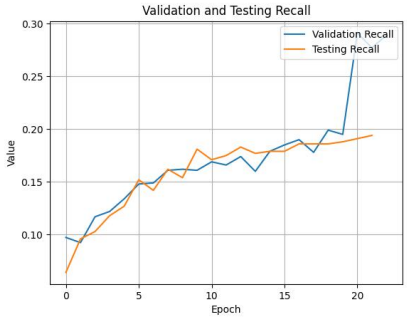
Graph depicting the evolution of learning rate throughout training for iteration 1 (25 epochs).



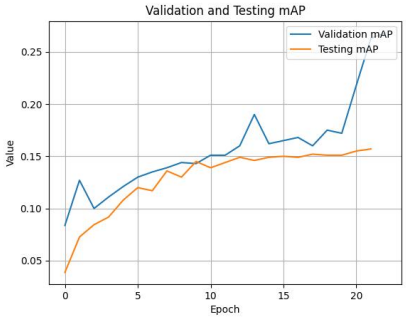
Graph depicting the evolution of training, validation and testing losses for iteration 1 (25 epochs).



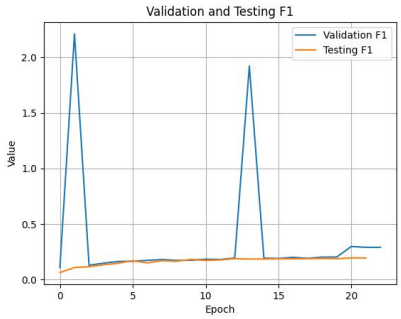
Validation and testing precision throughout training for iteration 1 (25 epochs).



Validation and testing recall throughout training for iteration 1 (25 epochs).

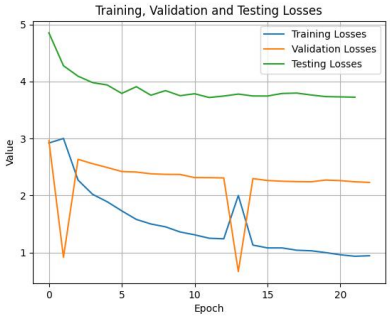


Validation and testing mAP throughout training for iteration 1 (25 epochs).

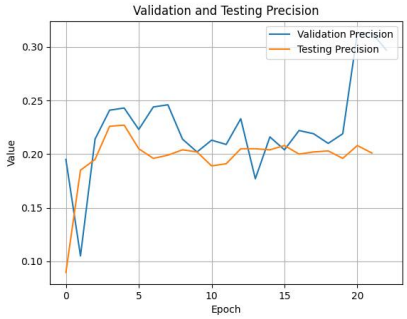


Validation and testing F1 throughout training for iteration 1 (25 epochs).

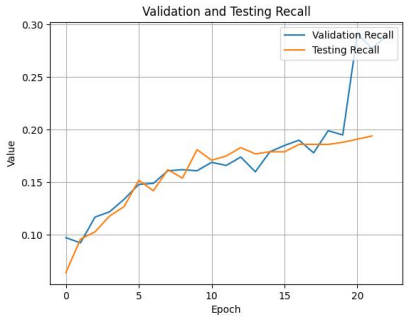
A.1.2 Training for 55 epochs



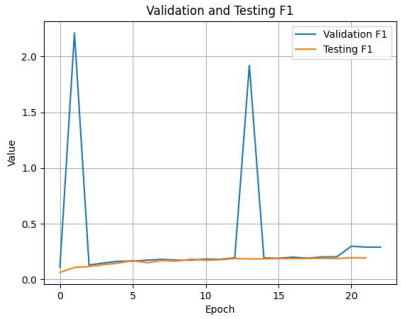
Graph depicting the evolution of training, validation and testing losses (55 epochs).



Validation and testing precision throughout training for iteration 1 (55 epochs).

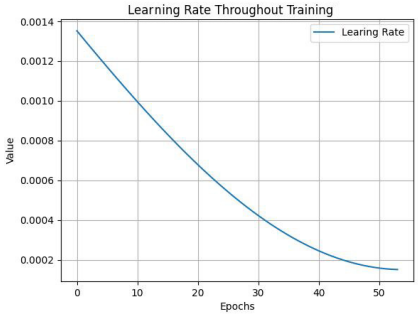


Validation and testing recall throughout training for iteration 1 (55 epochs).

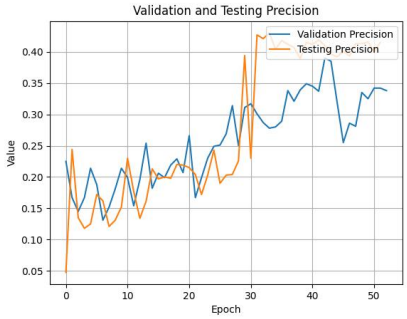


Validation and testing F1 throughout training for iteration 1 (55 epochs).

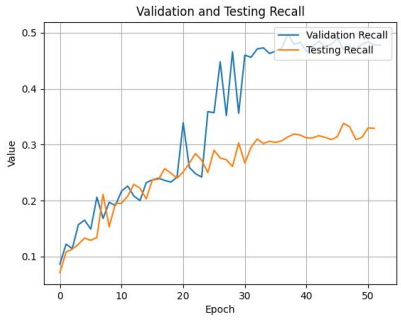
A.2 Iteration 2



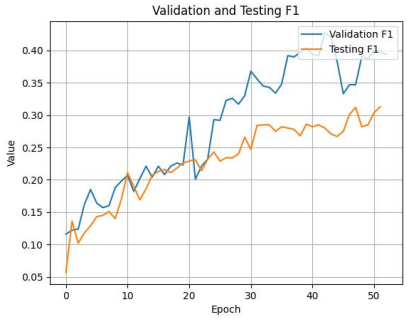
Graph depicting the evolution of learning rate throughout training for iteration 2.



Validation and testing precision throughout training for iteration 2.



Validation and testing recall throughout training for iteration 2.



Validation and testing F1 throughout training for iteration 2.

A.3 Iteration 3

A.3.1 Training with HSV=[0.2;0.2;0.2]

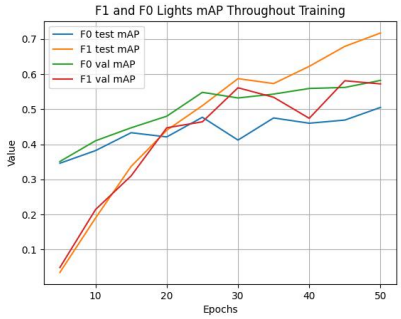
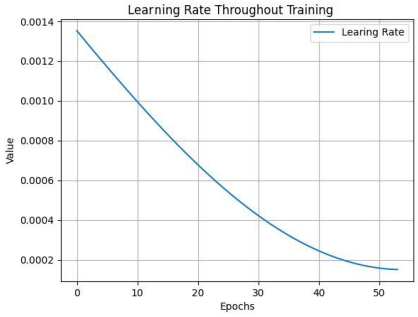
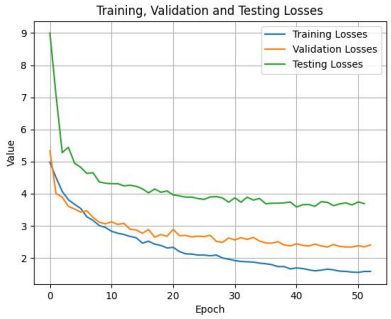


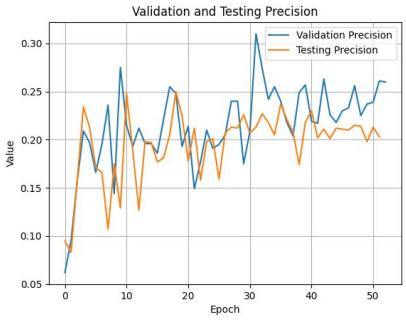
Figure with *F0* and *F1* throughout training and validation for iteration 3 (HSV @ 0.2;0.2;0.2).



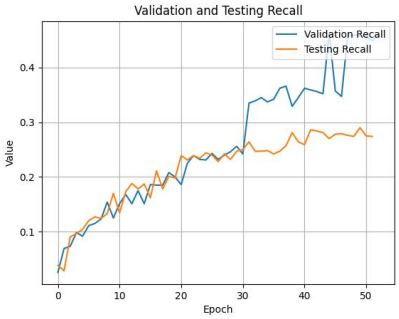
Graph depicting the evolution of learning rate throughout training for iteration 3 (HSV @ 0.2;0.2;0.2).



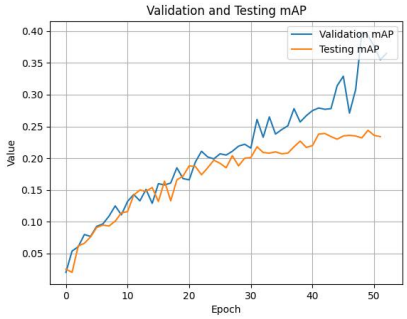
Graph depicting the evolution of training, validation and testing losses for iteration 3 (HSV @ 0.2;0.2;0.2).



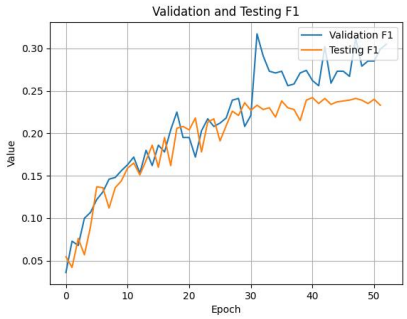
Validation and testing precision throughout training for iteration 3 (HSV @ 0.2;0.2;0.2).



Validation and testing recall throughout training for iteration 3 (HSV @ 0.2;0.2;0.2).



Validation and testing mAP throughout training for iteration 3 (HSV @ 0.2;0.2;0.2).



Validation and testing F1 throughout training for iteration 3 (HSV @ 0.2;0.2;0.2).

A.3.2 Training with HSV=[0.3;0.3;0.3]

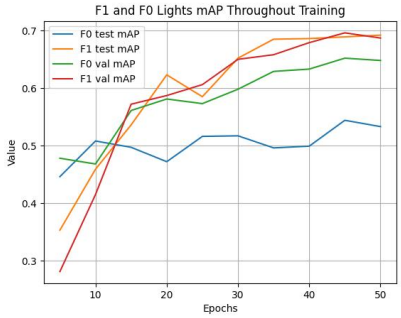
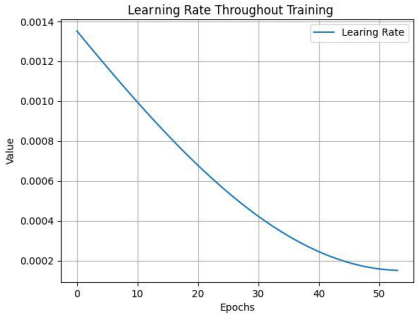
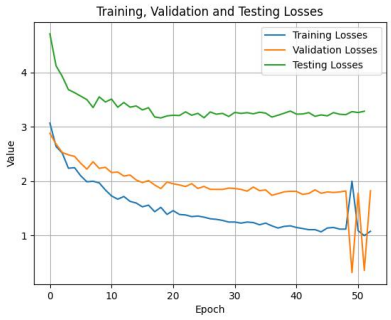


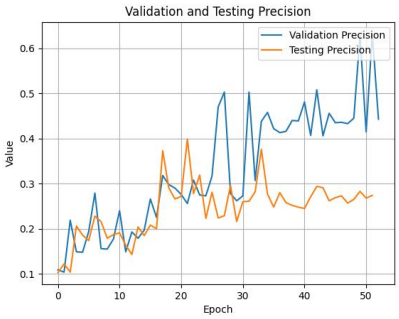
Figure with *F0* and *F1* throughout training and validation for iteration 3 (HSV @ 0.3;0.3;0.3).



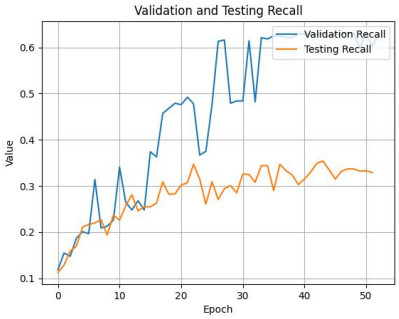
Graph depicting the evolution of learning rate throughout training for iteration 3 (HSV @ 0.3;0.3;0.3).



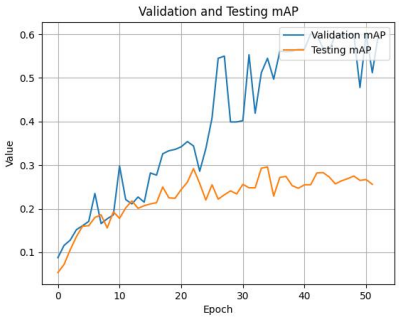
Graph depicting the evolution of training, validation and testing losses for iteration 3 (HSV @ 0.3;0.3;0.3).



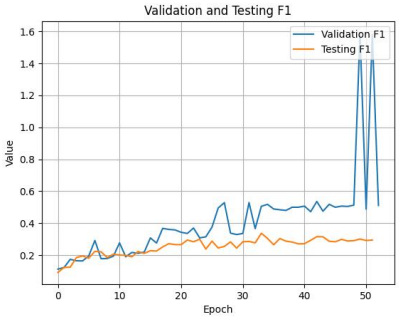
Validation and testing precision throughout training for iteration 3 (HSV @ 0.3;0.3;0.3).



Validation and testing recall throughout training for iteration 3 (HSV @ 0.3;0.3;0.3).



Validation and testing mAP throughout training for iteration 3 (HSV @ 0.3;0.3;0.3).



Validation and testing F1 throughout training for iteration 3 (HSV @ 0.3;0.3;0.3).

A.3.3 Training with HSV=[0.4;0.4;0.4]

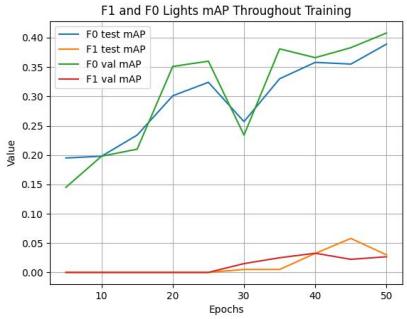
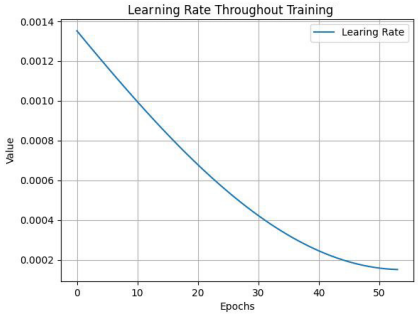
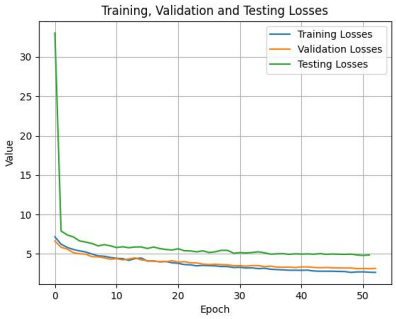


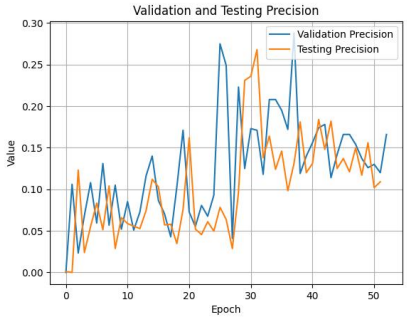
Figure with *F0* and *F1* throughout training and validation for iteration 3 (HSV @ 0.4;0.4;0.4).



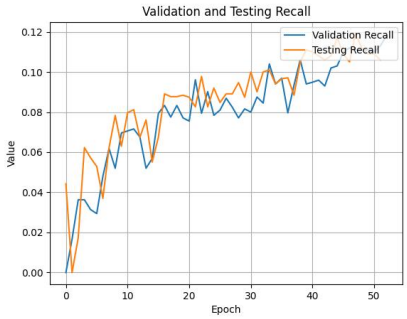
Graph depicting the evolution of learning rate throughout training for iteration 3 (HSV @ 0.4;0.4;0.4).



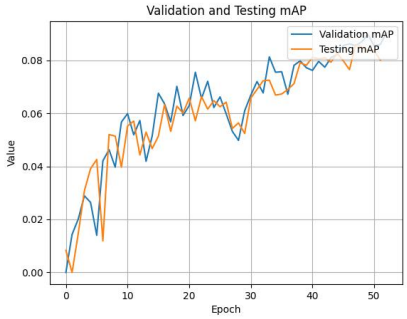
Graph depicting the evolution of training, validation and testing losses for iteration 3 (HSV @ 0.4;0.4;0.4).



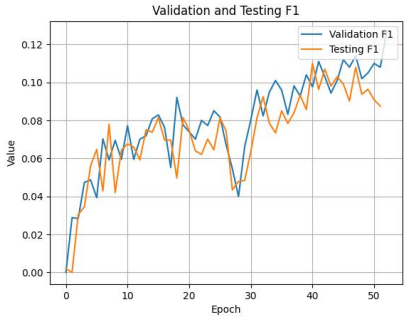
Validation and testing precision throughout training for iteration 3 (HSV @ 0.4;0.4;0.4).



Validation and testing recall throughout training for iteration 3 (HSV @ 0.4;0.4;0.4).



Validation and testing mAP throughout training for iteration 3 (HSV @ 0.4;0.4;0.4).



Validation and testing F1 throughout training for iteration 3 (HSV @ 0.4;0.4;0.4).

A.3.4 Training with HSV=[0.1;0.678;0.6]

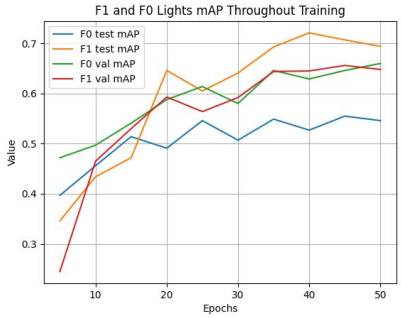
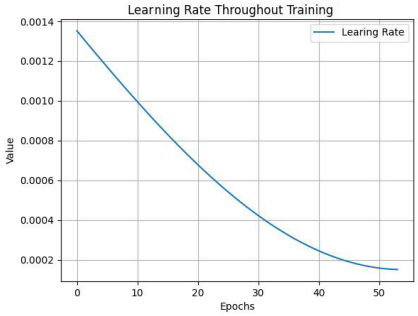
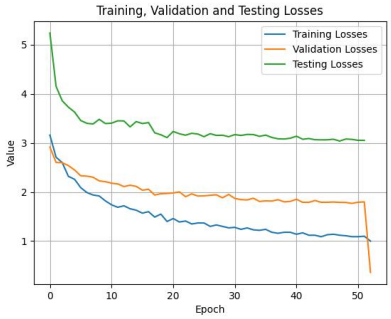


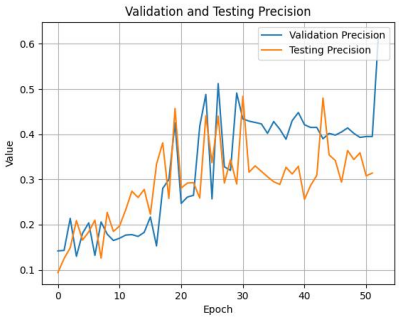
Figure with *F0* and *F1* throughout training and validation for iteration 3 (HSV @ 0.1;0.678;0.4).



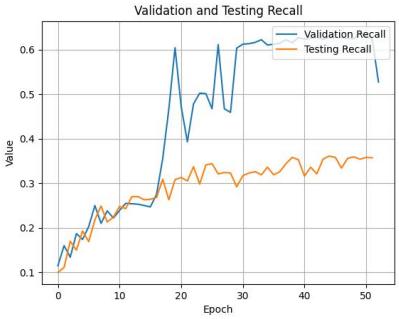
Graph depicting the evolution of learning rate throughout training for iteration 3 (HSV @ 0.1;0.678;0.4).



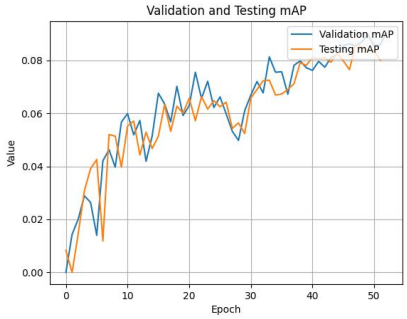
Graph depicting the evolution of training, validation and testing losses for iteration 3 (HSV @ 0.1;0.678;0.4).



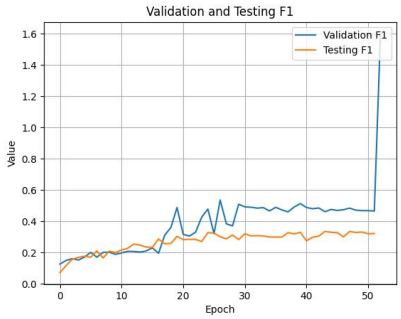
Validation and testing precision throughout training for iteration 3 (HSV @ 0.1;0.678;0.4).



Validation and testing recall throughout training for iteration 3 (HSV @ 0.1;0.678;0.4).



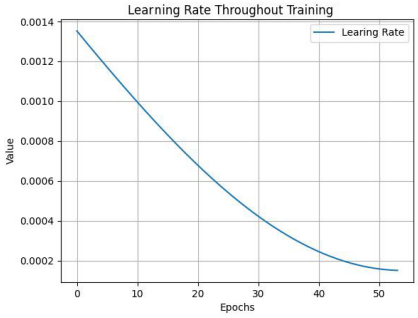
Validation and testing mAP throughout training for iteration 3 (HSV @ 0.1;0.678;0.4).



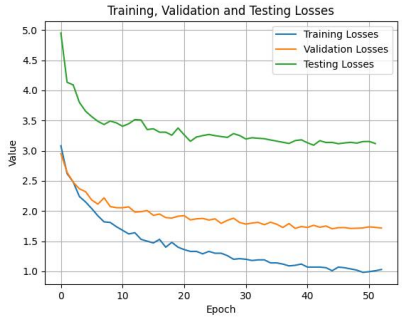
Validation and testing F1 throughout training for iteration 3 (HSV @ 0.1;0.678;0.4).

hfill

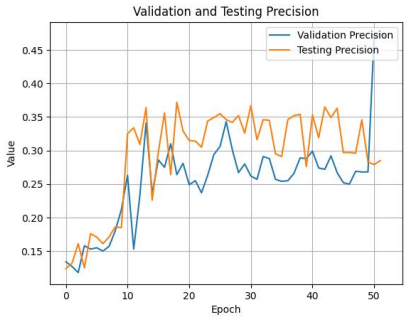
A.4 Iteration 4



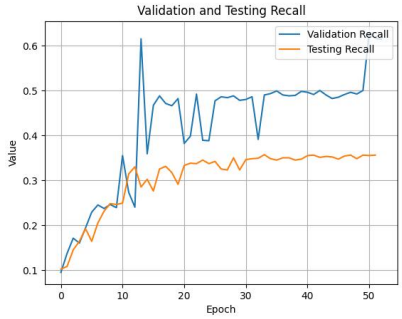
Graph depicting the evolution of learning rate throughout training for iteration 4.



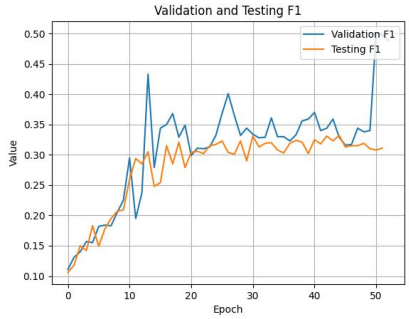
Graph depicting the evolution of training, validation and testing losses for iteration 4.



Validation and testing precision throughout training for iteration 4.

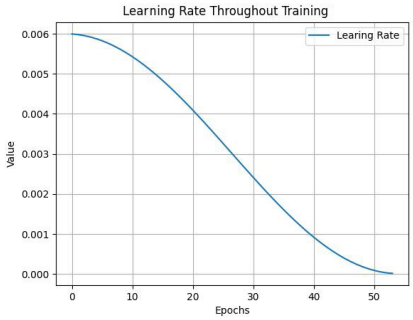


Validation and testing recall throughout training for iteration 4.



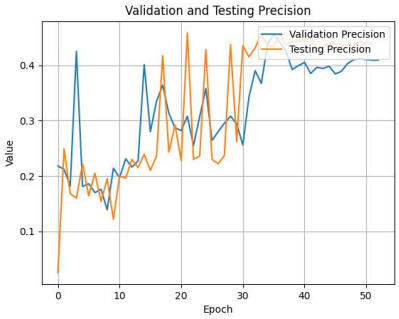
Validation and testing F1 throughout training for iteration 4.

A.5 Iteration 5

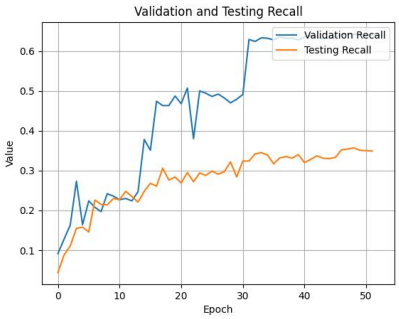


Graph depicting the evolution of learning rate throughout training for iteration 5.

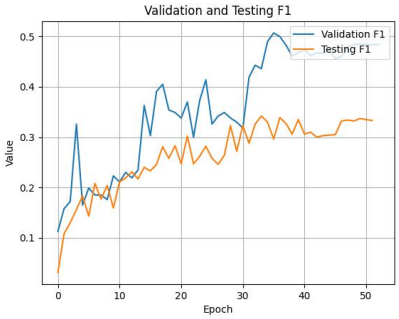
Graph depicting the evolution of training, validation and testing losses for iteration 5.



Validation and testing precision throughout training for iteration 5.



Validation and testing recall throughout training for iteration 5.

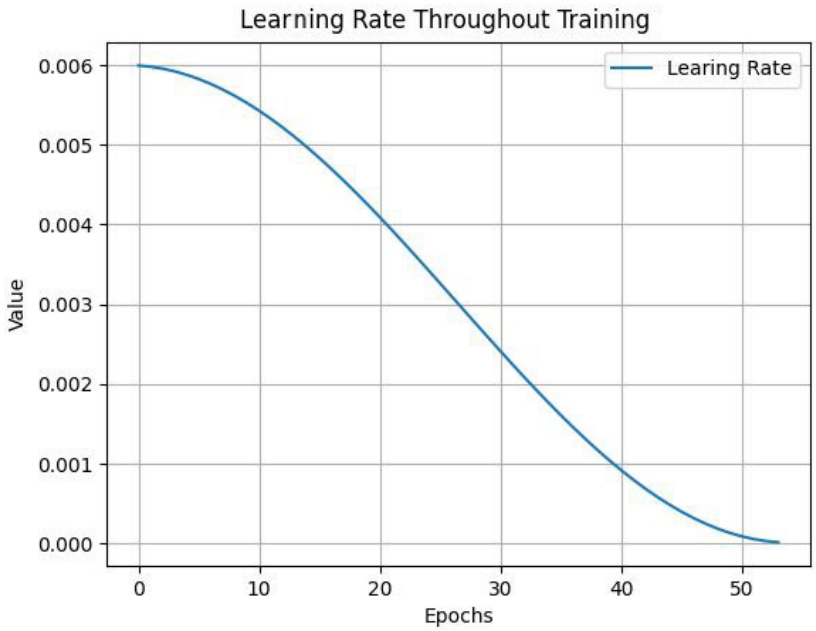


Validation and testing F1 throughout training for iteration 5.

A.6 Iteration 6

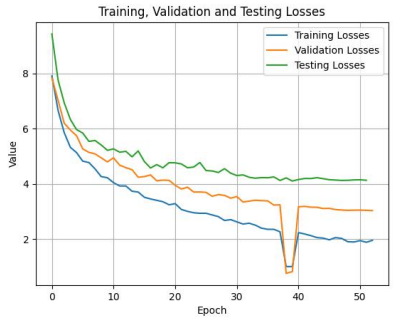
A.6.1 Parameters

Parameters used in the training of our best model. Note that clustering was also used, and is available in Figure 4.20. Also, the learning rate is depicted in the figure below.

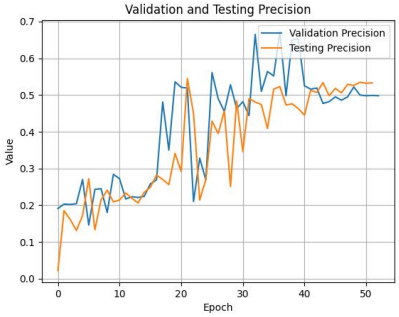


Graph depicting the evolution of learning rate throughout the training of the best model, other parameters available in Figure above.

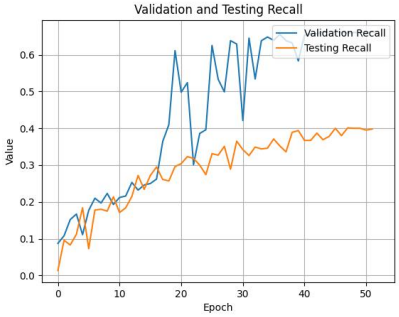
A.6.2 KPIs



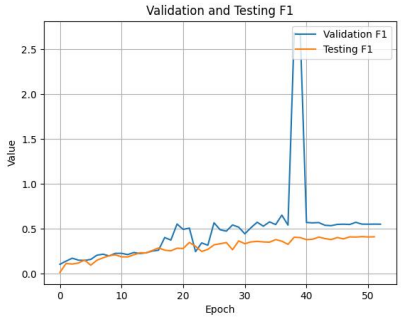
Graph depicting the evolution of training, validation and testing losses for iteration 6.



Validation and testing precision throughout training for iteration 6.

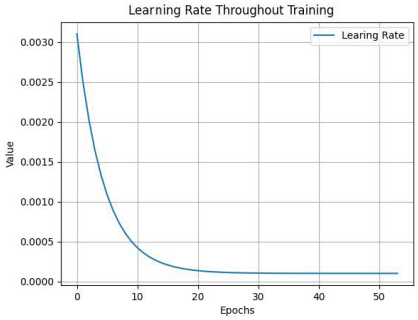


Validation and testing recall throughout training for iteration 6.

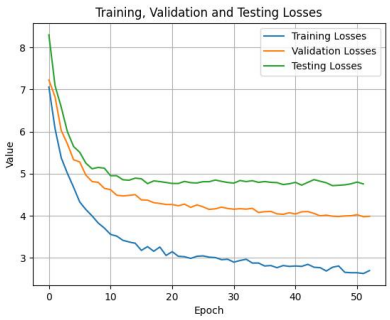


Validation and testing F1 throughout training for iteration 6.

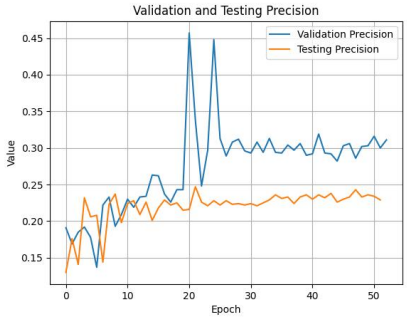
A.7 Iteration 7



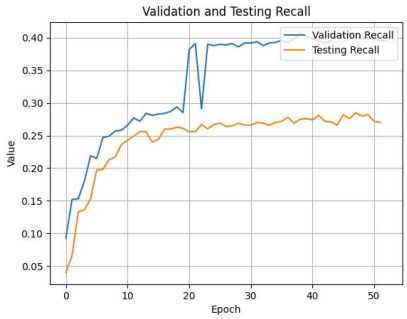
Graph depicting the evolution of learning rate throughout training for iteration 7.



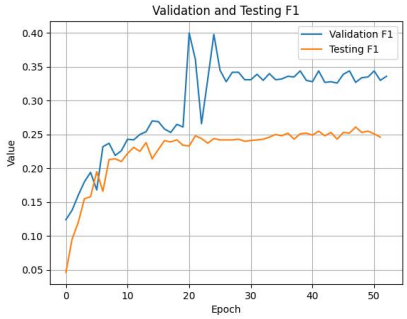
Graph depicting the evolution of training, validation and testing losses for iteration 7.



Validation and testing precision throughout training for iteration 7.

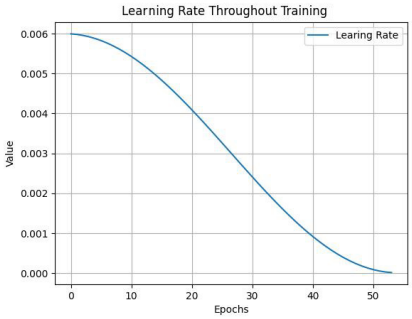


Validation and testing recall throughout training for iteration 7.

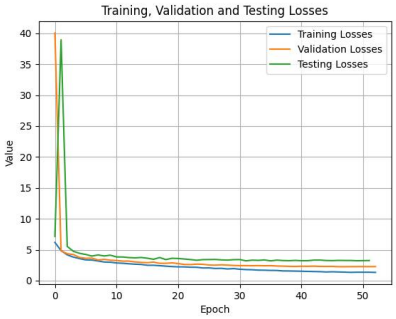


Validation and testing F1 throughout training for iteration 7.

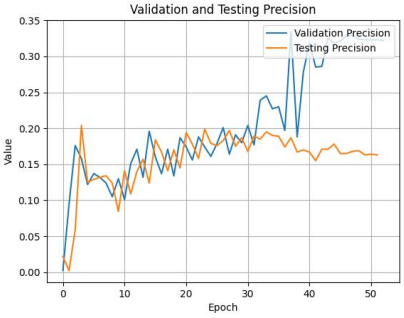
A.8 Iteration 8



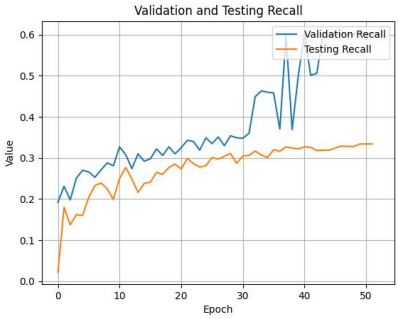
Graph depicting the evolution of learning rate throughout training for iteration 8.



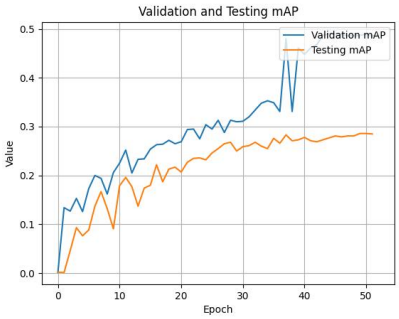
Graph depicting the evolution of training, validation and testing losses for iteration 8.



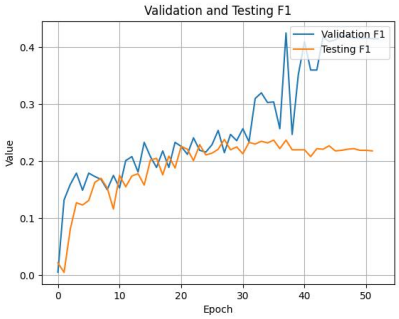
Validation and testing precision throughout training for iteration 8.



Validation and testing recall throughout training for iteration 8.



Validation and testing mAP throughout training for iteration 8.



Validation and testing F1 throughout training for iteration 8.