



1 2 9 0

UNIVERSIDADE D
COIMBRA

Carlos Tiago Neto

**TRAFFIC LIGHT DETECTION WITH NEURAL NETWORK
MODELS**

Dissertation in context of the Integrated Master's on Engineering Physics
Supervised by Dr. António Santos and Dr. Francisco Neves
Presented to the Department of Physics of the Faculty of Science and
Technology

SEPTEMBER OF 2022





UNIVERSIDADE D
COIMBRA

Traffic light detection with neural network models

Supervisor:

Dr. António Santos

Supervisor:

Dr. Francisco Neves

Dissertation submitted in partial fulfillment for the degree of Master of Science in Engineering
Physics.

Coimbra, October 2022

Acknowledgments

A chegada a esta etapa é devida em grande parte àquelas pessoas que me acompanharam desde início neste processo duro que é a vida, pois tal como uma planta, esta deve ser tratada e cuidada para chegar ao seu maior esplendor. A minha mãe e a minha irmã foram estas pessoas, pelo que estarei para sempre em dívida para com elas.

Gostaria de agradecer aos meus orientadores o Dr. António Santos e o Dr. Francisco Neves por me terem acompanhado neste último ano na realização deste projeto, por se terem certificado que nada falhava, seja em termos de equipamento, disponibilidade ou em motivação.

Como é óbvio, a passagem pelo ensino universitário não teria sido a mesma coisa sem as grandes amizades que de lá nasceram e que me acompanharam nesta jornada. Um agradecimento especial ao Micael, ao Eng. Carreira, ao Mário, ao João, ao Ponte, à Sara e ao Pedro. Também agradeço a todas as outras pessoas com quem tive o prazer de me relacionar durante este período e que também fizeram uma diferença positiva na minha vida.

Por último, mas com tanto ou mais significado, quero dedicar este trabalho ao meu avô Abel. Pelo grande homem que sempre foi, pelos grandes valores transmitidos e pela marca que me deixou. Embora fisicamente não presente, os seus valores perduraram naqueles que com ele privaram.

Resumo

A inteligência artificial é um campo em crescimento para aplicações em veículos autónomos. Quando um veículo tem de tomar decisões de forma autónoma, a inteligência artificial é um instrumento adequado para realizar essas escolhas. Os veículos devem, por si só e sem intervenção humana, ser capazes de localizar objectos no seu trajeto e de identificar o tipo de objetos que estão presentes. Este é um problema de deteção de objetos.

A aprendizagem automática é um subcampo da inteligência artificial que inclui o *deep learning*, no qual é possível encontrar algoritmos baseados em redes neuronais convolucionais que foram desenvolvidos especificamente para resolver problemas de deteção de objetos. Tais algoritmos incluem: R-CNN, Fast R-CNN, Faster R-CNN, Single Shot Detector (SSD) e You Only Look Once (YOLO).

O trabalho apresentado nesta dissertação consiste em aplicar um destes algoritmos para detetar semáforos em tempo real e detetar a cor correspondente: vermelho, amarelo ou verde. O trabalho foi realizado no contexto do que a empresa Active Space Technologies está atualmente a desenvolver, que é um veículo autónomo guiado automaticamente (AGV). A infraestrutura e modelo escolhidos são o TensorFlow Object Detection API e o SSD MobileNet V2 320×320 do TensorFlow 2 Detection Model Zoo. São utilizados dois conjuntos de dados diferentes; um feito a partir de imagens do conjunto de dados de semáforos LISA traffic lights e o outro é um conjunto de dados personalizado criado utilizando imagens de Coimbra, Portugal. Para facilitar o processo de etiquetagem deste último, o modelo pré-treinado SSD MobileNet V2 320×320 é utilizado para implementar um algoritmo capaz de etiquetar automaticamente as imagens recolhidas. Uma vez rotulado o conjunto de dados, são realizados diferentes treinos utilizando diferentes hiperparâmetros para encontrar os que se traduzem em melhores resultados num conjunto de dados de teste. Para cada hiperparâmetro alterado, é feita uma discussão sobre as possíveis razões pelas quais os resultados melhoraram ou pioraram. Além disso, é feita também uma discussão sobre o impacto que o meio envolvente aos semáforos tem nos resultados obtidos.

Palavras-chave: Inteligência Artificial, Aprendizagem Automática, Deep Learning, Rede

Neuronal Convolutacional, Detecção de Objetos.

Abstract

Artificial intelligence is a growing field for applications in autonomous vehicles. When a vehicle has to take decisions autonomously, artificial intelligence is a suitable tool to carry out those choices. Vehicles must, on their own and without human intervention, be able to locate objects on their path and to identify the kind of objects that are present. This is a problem of object detection.

Machine learning is a subfield of artificial intelligence. Further diving into machine learning's subset of deep learning, it is possible to find algorithms based on Convolutional Neural Networks (CNNs) that were developed specifically to solve object detection problems. Such algorithms include: R-CNN, Fast R-CNN, Faster R-CNN, Single Shot Detector (SSD) and You Only Look Once (YOLO).

The work presented in this thesis consists of applying one of those algorithms to detect traffic lights in real time as well as detect the correspondent color: red, yellow or green. The work was carried out in context of what the company Active Space Technologies is currently developing, which is an Automated Guided Vehicle (AGV). The chosen framework and model is TensorFlow Object Detection API and the SSD MobileNet V2 320×320 from TensorFlow 2 Detection Model Zoo. Two different datasets are used; one made from images of the LISA traffic lights dataset and the other is a customized dataset created using images from Coimbra, Portugal. To facilitate the labelling process of the latest, the pre-trained SSD MobileNet V2 320×320 model is used to implement an algorithm capable of automatically labelling the dataset. Once the dataset is labelled, different trainings are performed using different hyperparameters to find those that translate to better results in a test dataset. For every changed hyperparameter a discussion is made on the possible reasons why the results got better or worse. Moreover, a discussion is also presented on the way the surroundings of the traffic lights can impact the results obtained.

Keywords: Artificial Intelligence, Machine Learning, Deep Learning, Convolutional Neural Network, Object Detection.

*"Imagination is more important than knowledge. Knowledge is limited.
Imagination encircles the world"*

Albert Einstein

Contents

Acknowledgements	ii
Resumo	iii
Abstract	v
List of Acronyms	xii
List of Figures	xiv
List of Tables	xvi
1 Introduction	2
1.1 Context and motivation	3
1.2 Objectives	3
1.3 Outline of this dissertation	3
2 State of the Art	5
2.1 Deep Learning	5
2.1.1 Deep learning approaches	5
2.2 Neural Networks and object detection	6
2.3 Convolutional Neural Network Structure	7
2.3.1 Convolutional layer	7
2.3.2 Pooling layer	9
2.3.3 Non-linearity layer	9
2.3.4 Fully connected layer	10
2.4 Convolutional neural networks and object detection	11
2.4.1 Region proposal based method	11
2.4.2 SPP-Net	12

2.4.3	Fast R-CNN	12
2.4.4	Faster R-CNN	12
2.4.5	Regression/Classification based method	13
2.5	Transfer learning	13
3	Methods	14
3.1	Hardware and software	14
3.2	Deep learning framework	14
3.3	Choosing a pre-trained object detection model	14
3.4	Training the model	15
3.4.1	Dataset labelling	15
3.4.2	Dataset splitting	16
3.4.3	Creating a label map	17
3.4.4	Creating TensorFlow Records	17
3.4.5	Configuring the training pipeline	17
3.4.6	Initialize the training	18
3.5	Evaluation metrics of the model	18
3.5.1	Loss	19
3.5.2	Precision	19
3.5.3	Recall	19
3.5.4	Relation between precision and recall	19
3.5.5	Average Precision	20
3.5.6	Mean Average Precision	20
3.5.7	Average Recall	21
3.5.8	Mean Average Recall	21
3.5.9	Intersection over Union (IoU)	21
3.5.10	COCO detection metrics	21
4	Implementation and discussion	23
4.1	First Training	23
4.1.1	LISA Traffic lights dataset	23
4.1.2	Labelling	24
4.1.3	Training with the Dataset A	24
4.1.4	Training process	24
4.2	Gathering a second dataset	26

4.2.1	Automatic labelling of the Dataset B	26
4.3	Further training with the new data	33
4.3.1	Training 0	34
4.3.2	Training 1	35
4.3.3	Training 2	36
4.3.4	Training 3	37
4.3.5	Training 4	38
4.3.6	Training 5	40
4.3.7	Training 6	42
4.3.8	Training 7	44
4.3.9	Training 8	48
5	Conclusion and future work	56
	Bibliography	58
A	Appendix	62

List of Acronyms

CNN	Convolutional Neural Network
NMS	Non Maximum Supression
NN	Neural Network
RGB	Red, Green, Blue
BGR	Blue, Green, Red
TFOD API	TensorFlow Object Detection API
API	Application Programming Interface
COCO	Common Objects in Context
AGV	Automated Guided Vehicle
RNN	Recurrent Neural Network
RvNN	Recursive Neural Network
VOC	Visual Object Classes
TL	Traffic Light
HSV	Hue, Saturation, Value
PC	Personal Computer
2D	Two Dimension
AP	Average Precision
AR	Average Recall
mAP	mean Average Precision

mAR	mean Average Recall
SVM	Support Vector Machine

List of Figures

2.1	Convolution process [1].	8
2.2	Filter moving with stride 1 [2].	8
2.3	Zero-padding [2].	9
2.4	Max-pooling with 2×2 filter [2].	9
2.5	Activation functions: a) rectified linear unit (ReLU), b) Sigmoid, c) hyperbolic tangent (tanh) [1].	10
3.1	SSD Mobilenet architecture ¹	15
3.2	Example of labelled objects ²	16
3.3	Dataset division ³	17
3.4	Intersection over Union [3].	21
4.1	Samples of the images used from the LISA traffic lights dataset [4].	23
4.2	Train loss plots at every 100 steps and validation loss plots at the last 7000 thousand training steps, on Dataset A.	25
4.3	Two images samples used to test the training in Dataset A ⁴	26
4.4	Effect of Non Maximum Supression ⁵	28
4.5	HSV color map ⁶	30
4.6	Sample image of traffic lights with bounding boxes generated by the pre-trained model, accordingly to step 2 ⁷	30
4.7	Cropped detection from Figure 4.6, accordingly to step 3.	30
4.8	The three masks for the cropped detection of Figure 4.7a.	31
4.9	The three masks for the cropped detection of Figure 4.7b.	31
4.10	The three masks for the cropped detection of Figure 4.7c.	31
4.11	Situations of bad detections by the pre-trained model on MS COCO.	33
4.12	Train and validation loss plots for the different training iterations of Training 1 for Dataset B.	36

4.13	Train and validation loss plots for the different training iterations of Training 2 for Dataset B.	37
4.14	Train and validation loss plots for the different training iterations of Training 3 for Dataset B.	38
4.15	Detection of a training sample from Dataset B, with the original brightness level and with a different one.	39
4.16	Train and validation loss plots for the different training iterations of Training 4 for Dataset B1.	40
4.17	Train and validation loss plots for the different training iterations of Training 5 for Dataset B.	41
4.18	Train and validation loss plots for the different training iterations of Training 6 for Dataset B.	42
4.19	Train and validation loss plots for the different training iterations of Training 7 for Dataset C.	45
4.20	Train and validation loss plots for the different training iterations (with more 80 000 steps) of Training 7 for Dataset C.	47
4.21	Train and validation loss plots for the different training iterations of Training 8 for Dataset C.	49
4.22	Evolution of the average precision of the detection boxes under the different criteria of COCO detection metrics for the validation set of Dataset C during the different training iterations.	50
4.23	Evolution of the average recall of the detection boxes under the different criteria of COCO detection metrics for the validation set of Dataset C during the different training iterations.	51
4.24	Green traffic light detection samples on the model from Training 8.	53
4.25	Yellow traffic light detection samples on the model from Training 8.	54
4.26	Red traffic light detection samples on the model from Training 8.	55

List of Tables

3.1	CPU and GPU hardware for the different PC	14
4.1	Average precision and average recall COCO detection metrics for Training 5. . .	41
4.2	Average precision and average recall COCO detection metrics for Training 6. . .	43
4.3	Average precision and average recall COCO detection metrics for Training 7. . .	46
4.4	Average precision and average recall COCO detection metrics for Training 7 with more 80 000 steps of training.	47
4.5	Percentage of correct detection for Training 7.	48
4.6	Average precision and average recall COCO detection metrics for Training 8. . .	49
4.7	Percentage of correct detection in the last two training.	52
4.8	Information about the different datasets used.	55

1 Introduction

Human are not flawless; actually they are far from being perfect. Sometimes, this intrinsic characteristic of the humans can lead to dangerous outcomes, such as accidents, specially when operating machines, contributing to equipment damage or even wasted lives. A typical example happens in traffic environment, where there are a lot of variables acting simultaneously. Consider for instance, a person who crosses the road in an area where there is no cross-walk: during the night in poor illumination conditions. In such a situation it can be difficult for a person driving a car in that road to see the subject crossing it, possibly leading to a run over. Another example is when a person driving a car gets distracted and misses a red traffic light hitting another car in an intersection, again, possibly leading to casualties and damaged property. In that sense, over time, humans have been devoting time and effort in developing technology that is able to perform without or with less need of human intervention, decreasing the probability of making errors. The world had started moving towards automation; autonomous vehicles are an example.

In recent years, artificial intelligence has been proving itself to be a very import resource in the development of autonomous vehicles. For this kind of vehicle to be able to freely drive itself, it needs to be aware of all the static and dynamic variables in a road environment such as other cars, people, road limitations, crosswalk paths, traffic signs and traffic lights. The object detection field, in particular, is essential in this context, as it is focused in locating an object in an image with a bounding box as well as classifying it. Object detection can be achieved via deep learning's convolutional neural networks (CNNs) with architectures suited for that purpose. Region based (R) R-CNN, Fast R-CNN, Faster R-CNN, Single Shot Detector (SSD) and You Only Look Once (YOLO) are some of these architectures, mentioned in Section 2.4. In order for any of these architectures to be able to detect and classify the desired objects, they must be trained with a dataset containing samples of them. Moreover, some of these architectures are known to perform well in real time situations, which is fundamental for autonomous vehicles, so their employment to such case is very useful and will be explored in this work.

1.1 Context and motivation

This work is done in collaboration with the company Active Space Technologies (AST). The context is related to their current line of work, which is the development of an automated guided vehicle (AGV). The difference of an AGV from an autonomous vehicle is that the first follows a fixed path along wires or magnetic tape installed in the ground, while the other can move freely within a map. The use of neural networks has been proposed by the company in order to be able to detect objects according to their shape, morphology or color. Upon detection of the objects, more options are opened for further decisions by the AGV, such as contour objects in its path or by controlling its velocity. This technology is critical to ensure safe navigation on complex, crowded environments.

As for motivation, at a personal level, was due to discover about the current importance of neural networks in fields such as astronomy and healthcare. Besides this, the fact that artificial intelligence is a growing field, with its application reaching more areas, it would be enriching to get some knowledge regarding its use for a specific application and to then be able to apply it to others.

1.2 Objectives

The main goal of this dissertation is to train an artificial neural network, more specifically a (CNN) model, with the purpose of detecting traffic lights and their respective colors. For the training process we will be using an already pre-trained model on a large dataset of images, by performing transfer learning, explained in Section 2.5. A dataset containing traffic lights with red, yellow and green colors must then be gathered to train the model for this purpose. The network shall be able to localize traffic lights and identify their state when receiving real time video as input. To this work, an algorithm was designed and developed to facilitate the time consuming process of labelling a dataset.

1.3 Outline of this dissertation

Chapter 2 (State of the Art): This chapter gives some background on deep learning and neural networks (NNs) while focusing more on CNN. A revision of the literature is made on the object detection field with NNs.

Chapter 3 (Methods): Our framework is presented, as well as the chosen pre-trained model. The process of working with the framework is explained and also the output metrics it

provides.

Chapter 4 (Implementation and discussion): In this chapter it is presented the details of the implemented work. The multiple training phases are divided in subsections where it is described the chosen parameters, and a discussion of the obtained results.

Chapter 5 (Conclusion and future work): The conclusion of the work with a summary of what has been obtained during the implementation phase as well as considerations on future work are presented here.

2 State of the Art

2.1 Deep Learning

Deep learning is a subfield of machine learning, which usually deals with enormous amounts of data and built upon NN with high depths, i.e., NN having a large number of hidden layers. The performance of deep learning algorithms improve as the data available increases, in the case where diversity is ensured. This subfield of machine learning is nowadays one of the most important research trends [5]. The high depth neural networks have the ability to learn data representations with various levels of abstraction which, in turn, made outstanding improvements to the state of the art in areas such as object detection, speech recognition, visual object recognition, and more [6].

2.1.1 Deep learning approaches

There are three main categories in which deep learning is classified: unsupervised, partially supervised and supervised learning [5]. There is also another category called deep reinforcement learning, which falls in the semi-supervised and sometimes unsupervised learning methods [5]. Our work fits the last of the main categories.

Supervised learning

Supervised learning works with labeled data, i.e., a group of inputs and the respective outputs are known in advance [5]. For a system to be able to classify images containing, for example, a house or a car, first it is needed to collect a large dataset of images containing those elements and then label them accordingly to their categories [6]. Then, the neural network proceeds to be trained with that dataset. This means that its parameters are updated to predict the categories of elements defined for the dataset in a very effective way. The drawback of this method is the fact that decision boundary can be overstretched when the training set does not contain representatives of a class, nevertheless this technique is more straightforward [5].

Unsupervised learning

Unsupervised learning, contrarily to the supervised method, is able to apply the learning process without labeled data. In this case, the network learns the important features necessary to find unknown structures or relationships in the input data. Methods of dimensionality reduction or clustering are some examples that fit the unsupervised learning class [5]. Animal and human ways of learning are mainly unsupervised [6]. Unfortunately, this approach is unable to offer accurate information regarding data sorting, which is related to sorting the input data in accordance with similarities and differences in the features. It is also computationally complex [5].

Deep semi-supervised learning

This approach is essentially the combination of supervised and unsupervised learning, in the sense that now we are dealing with semi-labeled data [5]. It has the convenience of being able to minimize the amount of necessary labeled data. However, one drawback of this method is that unrelated input features in the training data could lead to wrong decisions [5].

2.2 Neural Networks and object detection

A deep learning neural network consists of connected neurons (usually arranged in layers), which forms its basic structure. Similarly to a neuron in the human brain, when receiving an input, the neuron processes that input and creates an output, which can then be sent to other neurons in the network that are connected to that neuron for additional processing or it can be the final output [7].

There are three main deep learning network architectures: recursive neural networks(RvNNs), recurrent neural networks (RNN) and convolutional neural networks (CNN) [5]:

- The RvNN makes it possible to achieve predictions in a hierarchical structure and also to classify the outputs using compositional vectors [5].
- The RNN is used mainly when we are dealing with sequential data. In this case, the previous output of the neuron is fed back to the input in order to be used to predict the next one. In this network, the neurons make it possible to store information about, for example, previous words for a certain amount of time in order to predict what the next word can be [5][6].

- The CNN is the most known and used neural network in the deep learning field, and its principal benefit is the ability to recognize important features in the data. It has been extensively applied in various fields such as computer vision, namely face recognition [5]. The principal benefit of its use is the weight sharing feature, which reduces the number of parameters to be trained in the network and contributes to improve generalization and also to avoid overfitting of the network [5]. An example of the reduction of the number of trainable parameters in a CNN is given in [2]: if we have an input (red, green, blue) RGB image of size $32 \times 32 \times 3$ and we connect it to a 32×32 neuron layer, we will have a full connection of 3,145,728 weights, but by applying a convolution filter to the input, those connections can be reduced to just 75. Another important aspect of weight sharing is that it brings invariance translations to the model, this allows the convolution filter to learn a feature independently of the spatial location [2]. Also, simultaneously learning the classification and feature extraction, turns the output highly organized and dependent on the features extracted [5]. Besides, CNN makes large-scale network implementation more straightforward when compared to other deep learning networks [5].

All this characteristics make the CNN a more suitable network for object detection. Therefore, this type of neural network (NN) is used as the driving force of this work.

2.3 Convolutional Neural Network Structure

The CNN architecture is a multi-layer neural network, like referred in Section 2.1, in the context of deep learning. This NN has four types of layers: the convolutional layer, pooling layer, non-linearity layer and the fully connected layer.

2.3.1 Convolutional layer

This layer is the most important one in a CNN [8] and it is where most of the computation work is produced [9]. The convolution is a linear operation that has the purpose of extracting specific features using an array of numbers designated by filters or kernels. The values of the kernel, the weights, are first randomly attributed, and then updated during the training process to be able to extract the features of interest [5]. This filter is moved all over the image with a given stride, see Section 2.3.1. The kernel multiplies weights with the values of the input that are superimposed with it and then adds the results producing a single output value that corresponds to a given stride position in the resulting output map; this is called a feature map (also called activation map) [1]. Figure 2.1 represents the aforementioned convolution process. This strategy

can be applied with different filters to create various feature maps emphasising different features. There are two key parameters that establish the convolution operation: number of filters and respective size. Normally, the filters have size of 3×3 , 5×5 , 7×7 . The number of filters will increase the depth of the feature maps; the more features extracted, the deeper will be the output feature maps.

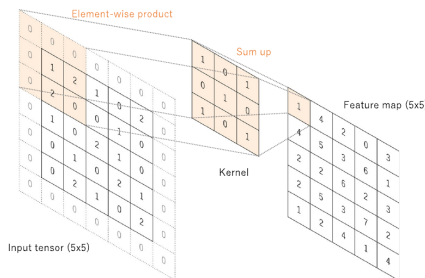


Figure 2.1: Convolution process [1].

Stride

The process of convolution reduces the parameters of the network, and there is a possibility to even decrease them more, using stride, which is basically the step at which the filter moves through the input. This reduction can, however, result in feature loss when stride is big. Increasing the stride, the smaller the overlap between filters would be and the size of the output would be reduced. For an 7×7 input image with stride 1 with a 3×3 filter we will get a 5×5 output. In the same conditions but with stride 2 we will get a 3×3 output [2]. Figure 2.2 illustrates stride.

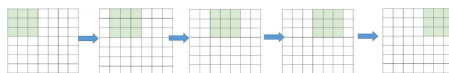


Figure 2.2: Filter moving with stride 1 [2].

Padding

The convolution process shrinks the output feature map in relation to the input [1]. We can recur to zero padding (Figure 2.3) to fix this problem, this adds rows and columns of zeros on both sides of the input. Without recurring to zero padding, the consecutive feature maps would be reduced continuously. Moreover, the convolution process can also induce loss of information that subsists in the image boundary, because only when the filter slides they have opportunity to be detected [2].

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Figure 2.3: Zero-padding [2].

2.3.2 Pooling layer

The goal of this layer is to down-sampling with the intent of decreasing the complexity for the upcoming layers [2]. It reduces the spatial dimension of the feature maps, with no loss of information [9]. Max-pooling is one of the most applied pooling approaches; it consists of dividing the map in rectangular sub-regions of the size of a filter, and returning a map containing only the maximum values of those regions. Figure 2.4 shows an example of max-pooling using a max-pooling window of 2×2 and a stride of 2 (stride 1 is not common because it avoids down-sampling).

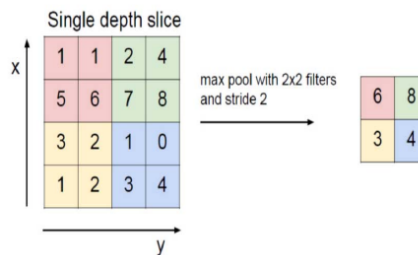


Figure 2.4: Max-pooling with 2×2 filter [2].

2.3.3 Non-linearity layer

Non-linearity layers are applied after all the learnable layers, which are those that have learnable parameters: the weights [5]. These linear parameters are then non-linearly transformed by applying activation functions. The most used activation functions are: sigmoid, tanh, ReLU and Leaky ReLU [9].

Sigmoid function

The sigmoid function is useful when we need to observe slight changes in the output values, because it generates a smoother range of values between 0 and 1. This function, however, has two disadvantages: It saturates and has the vanishing gradient problem [9]; this function is not

zero-centered, which causes the gradients to fluctuate between negative and positive values,

$$f(x) = \frac{1}{1 - e^{-x}}, \quad (2.1)$$

Tanh function

Similar to the sigmoid function but with a different range of values, from -1 to 1. Still, this function also has the vanishing gradient problem [9], but the outputs are zero centered.

ReLU

This function has a constant derivative value for all inputs greater than 0. This characteristic improves speed in the network training, which means the convergence of gradient descent is quicken up.

Leaky ReLU

This function consists of a smaller modification of the prior,

$$f(x) = \begin{cases} x, & x \leq 0 \\ 0.01x, & \text{otherwise} \end{cases}. \quad (2.2)$$

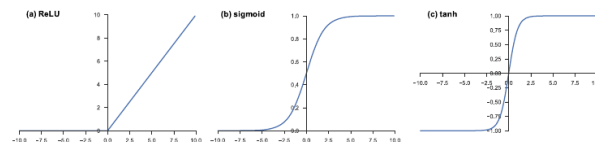


Figure 2.5: Activation functions: a) rectified linear unit (ReLU), b) Sigmoid, c) hyperbolic tangent (tanh) [1].

2.3.4 Fully connected layer

In this layer all the neurons are connected to all the neurons in the next and previous layers, similarly to a regular ANN (artificial neural network) [2]. It has the purpose of being the CNN classifier [5], this means it serves the goal of computing the loss that is the penalty for the difference between the intended output and the actual one [9]. It is the last output of this neural network. The input comes in the shape of a vector created from the activation maps, which comes from the last convolutional layer or pooling layer.

2.4 Convolutional neural networks and object detection

Object detection methods with CNN includes two categories: the region proposal based method with CNN features and the regression/classification based method [10] [11].

2.4.1 Region proposal based method

Traditional object detection is based on sliding windows on the input, but this method is time expensive and has a lot of redundant windows, ending up being not adequate [11]. To solve this issue, the region proposal method emerged. It first finds the possible locations of the objects in the image, that is, regions of interest (RoI), by scanning the image. This guarantees a somewhat higher precision in these conditions of less windows, because the quality of region proposals is higher. This reduces a lot the computation time of further actions. The region proposal algorithm mainly embraces selective Search and edge Boxes [11][10].

R-CNN

R-CNN has been proposed to solve the issues listed above. It was proved a success and launched the object detection based on CNN [11]. R-CNN obtained a (mean average precision) mAP of 53.3% with over 30% improvement regarding the previous best result (DPM histograms of sparse codes) in PASCAL VOC 2012 [10] [11]. R-CNN is divided in three stages [10] [11]:

- **1) Region Proposal Generation:** R-CNN embraces selective search in order to create around 2000 proposals (PASCAL VOC 2012) for each image. The selective search method depends on bottom-up grouping and saliency cues to create more accurate boxes of random sizes and to reduce space to search.
- **2) CNN-Based Deep Features Extraction:** All the proposed regions are warped or cropped into a fixed resolution, and then the CNN module extracts a feature vector from each of the RoI for the final representation. Due to the capabilities of the CNN such as learning capability and expressive power (which is related to the capability of extracting different features), a strong feature representation for all the regions can be obtained.
- **3) Classification and localization:** The object is separated from the background in the regions by specific linear Support Vector Machines (SVMs). The proposal areas are then scored as positive regions and the background as negative.

R-CNN disadvantages

R-CNN has some disadvantages such as the process of generation of region proposals by selective search, even with high recalls, the proposal regions can remain useless. Moreover, R-CNN are time expensive as it takes about 2s to extract 2000 suggestion regions [10]. Another problem is due to having fully connected (FC) layers, the CNN needs a fixed input size. This way, in order for the R-CNN to fit the input image to itself, it must deform or cut the image, causing loss of information, for instance aspect ratio and scale. Another problem is the training requiring significant resources, both in memory allocation and processing time [11] [10].

2.4.2 SPP-Net

This CNN architecture fixes the R-CNN problem of fixed input size, since the input is flexible. With this fix, there is no more information loss from cutting or deforming the image. This problem was solved by placing a Spatial Pyramid Pooling (SPP layer) before the last convolutional layer. The SPP layer concentrates information at a more advanced level of the network, by pooling the features and generating a fixed-length feature output vector representation for each region proposal. This resulted in a better mAP compared to R-CNN [11] [10].

2.4.3 Fast R-CNN

The Fast R-CNN appeared to improve several aspects of the SPP-Net. This architecture introduced the region of interest (RoI) pooling layer influenced by the SPP-layer, while maintaining the ability to process any kind of image size [11]. It displays the following advantages towards its predecessor: unlike the SPP training, which is a multi pipeline structure (feature extraction, SVM classifier, bounding box regression), the Fast R-CNN training proves it self as a all in one, because it operates with multi-task loss on all the labeled regions of interest to combine the classification and bounding-box regression. With Fast R-CNN it is possible to update all the convolutional layers during training, when with SPP-net it is not, preserving more localized information [11] [10].

2.4.4 Faster R-CNN

The improvement on accuracy of Fast R-CNN comes with the cost of speed, making it hard to have object detection in real-time [11]. It was proposed a network to make it close to real time known as Region Proposal Network (RPN) [10]. The intent of the RPN is to make use of the CNN to create a region proposal with a high capability of testing correctly and instantaneously, increasing detection speed, and still providing the possibility of training without a multi-stage

pipeline the same way as in Fast R-CNN. The RPN is obtained via a Fully Connected (FC) network, it creates boxes called anchor boxes with three scales, to predict the likelihood of being background or foreground [8]. This method achieved a 73.2% mAP on VOC2007 and is able to get to a detection speed during training of an average of 5 images per second [11] [10].

2.4.5 Regression/Classification based method

Despite the breakthrough of the last architectures, Faster R-CNN is still unable to match real-time specifications [11]. This situation gave rise to regression methods, which consist of splitting the input image into multiple cells, with each cell predicting boxes and the probability of a corresponding class. YOLO (You Only Look Once) had emerged. According to [11], it increases the speed of detection substantially as it is able to process 45 images per second. YOLO still has a problem, though. On VOC2007 its mAP is low, 63.4%.

Single Shot Detector (SSD) is another regression method. This method mixes the YOLO regression concept with the anchor method from the Faster R-CNN in order to do predictions from activation maps with various scales. Considering this approach, it is possible to obtain similar high accuracy to that obtained with Faster R-CNN and the same real time speed as YOLO; in VOC2007 it is obtained 72.1% .

2.5 Transfer learning

In order to have a good performance, the CNN models need a huge amount of data, like mentioned in Section 2.1, which sometimes is not available [5] [1]. This problem of insufficient data can be solved with a transfer learning method, which makes it possible to train a neural network under the condition of a low amount of data. Transfer learning works by making possible to use a pretrained network model on a very large dataset and then re-utilizing, fine tuning it, which means making adjustments to that prior model, in order to train the network on a small dataset. The use of transfer learning is useful in image classification problems such as object detection [12]. Some pre-trained CNN models include: AlexNet, GoogleNet and ResNet. All these models were pre-trained on huge datasets [5].

3 Methods

3.1 Hardware and software

For the implementation of this work, the chosen programming language was python, version 3.8.10. Three computers were used. They are designated by PC1, PC2 and PC3, for easier reference. Table 3.1 shows the CPU and GPU hardware for the different computers. All computers are operating on Ubuntu, PC1 and PC3 with Ubuntu 20.04.4 LTS and PC2 with Ubuntu 21.10. The respective attributed tasks of each device are clarified in the Implementation chapter.

Table 3.1: CPU and GPU hardware for the different PC

	CPU	GPU
PC1	Intel(R) Core(TM) i7-4720HQ @ 2.60GHz	Nvidia GeForce GTX 950 M 2 GB
PC2	Intel(R) Core(TM) i7-6700 @ 3.40 GHz	Nvidia GeForce GTX 970 4 GB GPU
PC3	AMD Ryzen 9 5950X 16-Core	Nvidia GeForce GT 730 64 GB

3.2 Deep learning framework

Regarding the process of training a neural network model the work was done using TensorFlow [13], which is an open source python library for machine learning and deep learning created by Google. The TensorFlow Object Detection API (TFOD API) [14] is a framework that includes various pre-trained neural network models for object detection, called Model Zoo. This framework is good to build, train and export object detection models. The version of Tensorflow used in this work was 2.8.0.

3.3 Choosing a pre-trained object detection model

To choose a pre-trained neural network model we look into the existing models of the TensorFlow 2 Detection Model Zoo public github repository [15]. These models have been pre-trained

on the Microsoft Common Objects in Context (MS COCO) 2017 dataset, which includes 90 different classes of objects. Because we are going to perform transfer learning, the weights obtained in the pre-training are only used to initialize the network [16]. To choose a model, we need to look at the performance parameters: the speed and COCO mAP metrics values, as well as the type of output. We can see that, normally, the slower models are those with better COCO mAP [15].

Because we are interested in real time object detection where we require a faster network to process each video frame within a suitable time, we opted by the faster model inspite of its worst COCO mAP. Thus, we choose a Single Shot Detector (SSD) version of MobileNet, the SSD MobileNet V2 320×320 model. This model has an inference speed of 19 ms [15], a Common Objects in Context mean average precision (COCO mAP) of 20.2% and an output of boxes surrounding the detected object. The model receives images as input and resizes them to 300×300 . The output comprises the coordinates for the bounding box of the detection as well as the associated confidence score and the predicted class label. Figure 3.1 shows the architecture of this model.

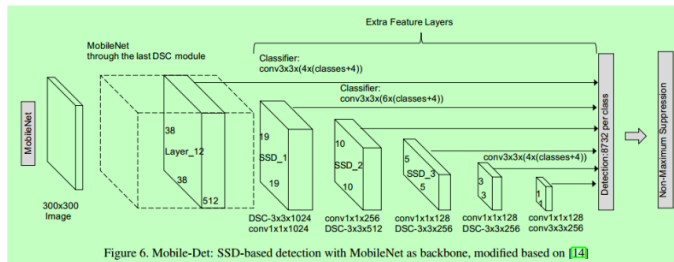


Figure 3.1: SSD Mobilenet architecture ¹.

3.4 Training the model

To be able to perform the training of the model through the TFOD API, a set of steps needs to be followed. These include: dataset labelling; dataset splitting; the creation of a label map; the creation of TensorFlow records and the configuration of the training pipeline.

3.4.1 Dataset labelling

The labelling of the dataset consists of attributing a particular label to an object or various objects present in the dataset images and to provide coordinates for the position of those objects.

¹<https://medium.com/@techmayank2000/object-detection-using-ssd-mobilenetv2-using-tensorflow-api-can-detect-any-single-class-from-31a31bbd0691>

The object coordinates are normalized in relation to the image size. The labelling was done using LabelImg [17], a software written in python that allows to manually create a bounding box around the desired object in the image and then attribute a tag to it. The information is then saved as an XML file in the PASCAL Visual Object Classes (VOC) format, which is the labelling tool’s default. An illustration of labelled objects is presented in Figure 3.2.

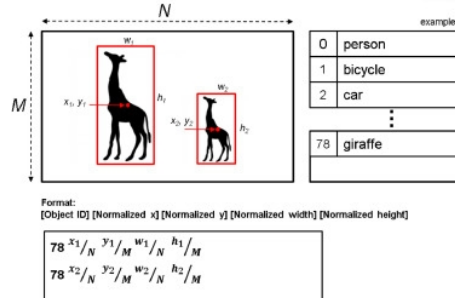


Figure 3.2: Example of labelled objects ².

3.4.2 Dataset splitting

In machine learning it is common practice to divide the dataset into three sets: training, validation and testing. The training set is used to train the model. The validation set finds its use during training to evaluate the progress of the model’s performance. The test set serves the purpose of providing an estimation of the model’s performance after the training process is finished. This last set contains data never previously seen by the model during the training, allowing to check whether the model generalizes well or not [18][19]. Figure 3.3 shows the typical dataset division.

In this work, we want to perform multiple training of the same model, while tuning the models parameters in order to improve its performance. We will be using the same test dataset for all the different training, described in Section 4.3. In this way we can compare directly the evaluation metrics and see if the model is improving or not.

To proceed to the training process we are dividing the dataset into two datasets, one for the training process and the other for validation. For this, a python script from github [20], *test_train_split.py*, is used, a certain percentage of the total dataset goes for training and the remaining for validation. Then, these two datasets are placed inside the TFOD API installation directory. This splitting does not account for the test dataset due to the fact that in this work the test dataset is kept constant as previously mentioned, regardless of the dataset used for training.

²A dataset of labelled objects on raw video sequences - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-the-object-annotation-format_fig2__348000819

The test dataset was chosen from images taken in Coimbra, Portugal. Video frames of various places of the city containing traffic lights in all colors were used. This way, different backgrounds with different lighting are present to see how robust the model is in different conditions.

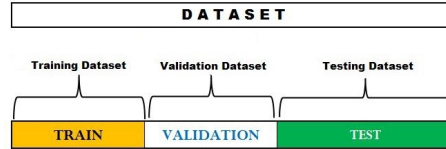


Figure 3.3: Dataset division ³.

3.4.3 Creating a label map

The API requires a file that maps our labels into integer numbers is needed for both the training and detection process. This is called a label map and it is an ptxt file. We are considering three labels: Red light, Yellow light and Green Light, the values assigned to each one are 1, 2 and 3, respectively.

3.4.4 Creating TensorFlow Records

Once we have the three dataset sets: train, validation and test, it is needed to convert the XML file generated by the LabelImg tool in the labelling process to a TFRecord file. We will get three TFRecord files, one for training, other for validation and another for testing. To achieve this we are using a script from [21].

3.4.5 Configuring the training pipeline

In this step, we need to access the downloaded files from our chosen pre-trained model. These contain the pipeline configuration and the last checkpoint from the prior training with the COCO dataset. In order to train our model, it is need to access the pipeline configuration. This configuration is divided into five main groups [22]:

- **model:** This is where the meta-architecture and the feature extractor of the model are defined through a set of parameters. We can find parameters such as the number of classes, which initially are set to 90 (then changed to 3; the API deals automatically with the necessary structure changes to the model), because of the pre-training on the COCO dataset; the resolution at which the input images are resized; anchor box parameters such as scale and aspect ratio.

³<https://vitalflux.com/hold-out-method-for-training-machine-learning-model/>

- **train_config:** In this group we find the parameters to train the model such as: batch size, application of data augmentation types and the optimizer parameters such as the learning rate.
- **train_input_reader:** Definition of the TFRecord file path that corresponds to the training dataset.
- **eval_config:** This is where it is specified the metrics to be used in the evaluation process. In our case it is the COCO detection metrics, see Section 3.5.
- **eval_input_reader:** Definition of the TFRecord file path for the validation dataset or the test dataset.

3.4.6 Initialize the training

The training of the model is initialized by running a script provided by TFOD API, called model *model_main_tf2.py*. At every hundred steps, the script gives information about the training loss. The loss is explained in Section 3.5.1.

3.5 Evaluation metrics of the model

The evaluation metrics used in this work are: Loss, Precision and Recall. These are based on the COCO detection evaluation metrics. For this, we need to install COCO evaluation metrics [23]. The process of evaluation is done by comparing the ground truth boxes of the dataset images, which are created in the labelling process, with the bounding boxes generated by the model when predicting an output for the same images.

During the training process, we can retrieve the evaluation metrics of the trained model at different stages, producing checkpoint files at various training steps. These checkpoints present the status of the model at that iteration, so we can see how well the model is behaving. We will use these checkpoints with our validation dataset to get values for the validation loss so we can make a plot of loss versus steps. That plot also includes the training loss that is generated automatically during training. Once the training is finished, we will test the model on the last checkpoint generated with test dataset to get the metrics for later comparison.

3.5.1 Loss

There are four loss types: localization loss, classification loss, regularization loss and total loss. The localization loss calculates the difference between the ground truth and predicted bounding boxes. The classification loss tells the difference between the predicted and the ground truth class. The regularization loss is derived from the network’s regularization function and serves the purpose of improving the model generalization, in order to obtain better results in the test set [24]. In this API, the regularization loss is computed by the *l2_regularizer*. And finally, the total loss is the result of the sum of all the previous losses.

3.5.2 Precision

Precision measures how good our trained model is in identifying a true positive.

Precision is given by:

$$Precision = \frac{TP}{TP + FP} \quad (3.1)$$

Where *TP* and *FP* refer to the number of true positives and the number of false positives, respectively. The former corresponds to a correct detection of the ground truth box of the object whereas the latter identifies an incorrect detection of an object that does not belong to the particular class or classes of interest or there is not an object at all. The higher *FP* is, the lower the precision will be, meaning the model is less reliable. The contrary situation translates to a more reliable model.

3.5.3 Recall

The recall metric tells how well the model identifies a positive class.

Recall is given by:

$$Recall = \frac{TP}{TP + FN} \quad (3.2)$$

The *TP* has the same meaning as in precision whereas *FN* refers to the number of false negatives and is taken in to account whenever a ground truth bounding box goes undetected. If *FN* is high relative to *TP* in the images, the recall tends to be low as a considerable number of positive objects were not identified.

3.5.4 Relation between precision and recall

The confidence level of the bounding box generated by the detector can be taken into account in the calculations of precision and recall, by considering as a positive detection only those that

have a confidence score superior than a certain confidence threshold [3] [25]. This confidence level is the confidence score, which is a percentage value, the model generates for each detection, and shows how confidence that model is for a specific detection [3]. The confidence threshold, is a confidence score that is established to be the minimum to be considered. The FP decreases as the confidence threshold increases since less detections will be considered as positive, resulting in a higher precision, but can also mean that there are many FN, thus contributing to a lower recall. On the other hand, as the confidence threshold decreases, there will be more detections considered as positive, which may mean a lower FN and also a higher FP, resulting in a higher recall and in a lower precision. In the case of traffic light detection, a higher precision may be more important than a higher recall. This is because most of the times, a traffic light mast contains usually two traffic lights presenting the same color. This means that, in this situation, it is enough to have just one detected. If correctly detected (i.e. correctly classified and located), this illustrates a situation of higher precision than recall.

3.5.5 Average Precision

Although precision and recall seem to have an inverse relationship, for a good performing model it is expected that the precision stays high while its recall increases, that means if the confidence threshold, Section 3.5.4, changes, both precision and recall shall remain high. In this case, the area under the curve (AUC) [3] of the precision-recall plot should be high.

The average precision (AP) metric is obtained by calculating the AUC of a precision-recall plot and represents the precision-recall balance for different confidence thresholds. However, practically, this plot often portrays a zig-zag behavior instead of a monotonic curve one, which makes it hard to get an accurate value for the AUC. Interpolation methods are used to achieve a monotonic precision-recall curve [3] [25].

3.5.6 Mean Average Precision

While AP is acquired individually for each class, when we are dealing with datasets of multiple classes the mean average precision is used for all classes,

$$mAP = \frac{1}{C} \sum_{i=1}^C AP_i, \quad (3.3)$$

Where C corresponds to the number of classes, and AP_i is the AP obtained for class i .

3.5.7 Average Recall

This is another evaluation metric used in object detection and it corresponds to averaging all the recall values obtained for the Intersection over Union (IoU), defined in Section 3.5.9, interval that goes from 50% to 95% for a specific class. In AR calculation, confidence thresholds are not considered [25].

3.5.8 Mean Average Recall

Mean Average Recall (mAR) is important in a multi-class context. Similarly to mAP, it is an average of AR over all classes,

$$mAR = \frac{1}{C} \sum_{i=1}^C AR_i, \quad (3.4)$$

Where C corresponds to the number of classes, and AR_i is the AR obtained for class i .

3.5.9 Intersection over Union (IoU)

To help differentiate a correct detection from a wrong one, in object detection, the IoU is used. The IoU computes the area of the intersection between the predicted bounding box and the ground truth divided by the union area of both. Then, with a given IoU threshold, a detection is classified as being correct or not. When $IoU \geq threshold$ the detection is perceived as correct, otherwise incorrect [3] [25]. Figure 3.4 illustrates IoU.

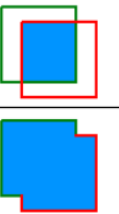
$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{img}}{\text{img}}$$


Figure 3.4: Intersection over Union [3].

3.5.10 COCO detection metrics

In COCO detection metrics AP represents the same as mAP, because it corresponds to the average AP over all different classes. The same happens with AR and mAR [26].

The COCO detection metrics are divided into four groups:

1) AP evaluated with different IoU intervals

- In a range of 50% to 95% by steps of 5%, resulting in 10 IoU thresholds;
- A single IoU value of 50%;
- A single IoU value of 75%.

2) AP across scales:

The AP is calculated for objects with three different sizes:

- Small: objects with area smaller than 32×32 pixels;
- Medium: objects with area between 32×32 and 96×96 pixels;
- Large: objects with area larger than 96×96 pixels.

In the detection of traffic lights it is of interest to have this criteria. These objects can appear at a long distance from our frame of reference, or at medium or close distances, meeting these three criteria.

3) Average Recall (AR):

AR is given in three situations:

- AR for just one detection per image;
- AR for 10 detections per image;
- AR for 100 detections per image.

4) AR across scales:

The AR is calculated for objects with the same sizes as in the AP across scales case.

4 Implementation and discussion

4.1 First Training

To start the implementation of this work, we begin by gathering a first dataset to initiate training with the TFOD API, as detailed in Section 3.2. After the labelling of the dataset we proceed to the training process. In the following sections the hyperparameters used are explained and the training and validation loss plots are presented. A small discussion is then made about the behavior of the plots.

4.1.1 LISA Traffic lights dataset

The first step is to gather pictures of traffic lights. In this work there is a need of images of traffic lights corresponding to three different situations: With red light (stop movement); with yellow light (continuing the movement with caution or stop); green light (continuing the movement). We start by choosing images from a dataset available online, called LISA Traffic lights dataset [4], that contains around 5 GB worth of images (43 007 images) with 1280×960 resolution, taken in the USA in urban environments. These images were taken from a car while driving, which means that in general the traffic lights will be small with respect to the image size and, therefore, some details such as protrusions may be missed. From this dataset, a group of 366 images is picked in daylight conditions. In this set, we have a total of 395 red traffic lights, 84 yellow traffic lights and 554 green traffic lights. Figure 4.1.1 shows three sample images of this group. This group will be referred as Dataset A.



Figure 4.1: Samples of the images used from the LISA traffic lights dataset [4].

4.1.2 Labelling

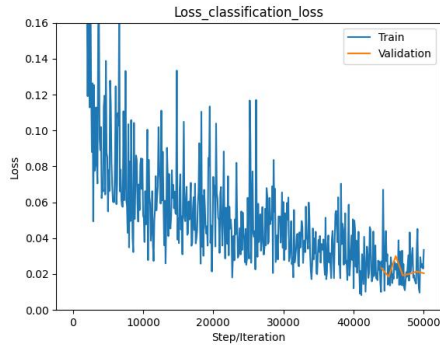
Although LISA TL dataset already comes with annotations (stop, caution and go) for the traffic lights presented in each image, we opted to relabel them with our own chosen tags, which are: Red light; Yellow light, Green light.

4.1.3 Training with the Dataset A

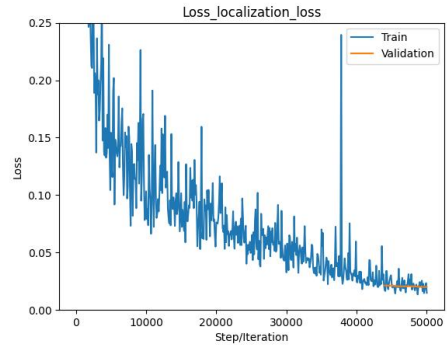
At this stage, the steps listed in Section 3.4 from the previous chapter must be followed. A split of 90% and 10% of the group images is made into train and validation, respectively, resulting in 330 images for training and 36 for validation. The pipeline configuration file is changed in the *num_class* parameter, from 90 to 3, which represents the number of classes we will use in this work: Red light, Yellow light, Green light. Besides that, we are setting the batch size parameter to 32. The training is done on PC2, referred in Table 3.1. Under these conditions, the training takes about 2 days for 50 000 steps. In the TFOD API, a step/iteration corresponds to the operation of updating the weights using a single batch as input. In this example, an epoch occurs after 11 steps, this means that in 50 000 steps 4849 epochs occur.

4.1.4 Training process

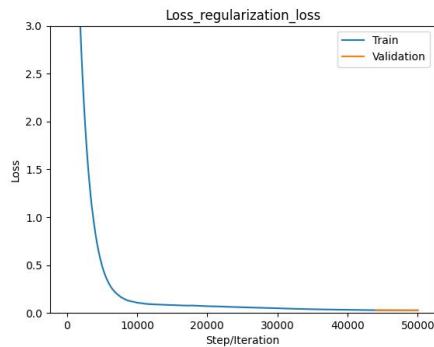
Figure 4.2 shows the evolution of the training and validation metrics (e.g. Classification loss, Localization loss) throughout the iterations. The training loss is calculated at every step, but only registered at every hundred steps for plotting, while the validation loss is only calculated at every thousand steps. The latter is because every checkpoint, referred in Section 3.5, is generated at every thousand steps.



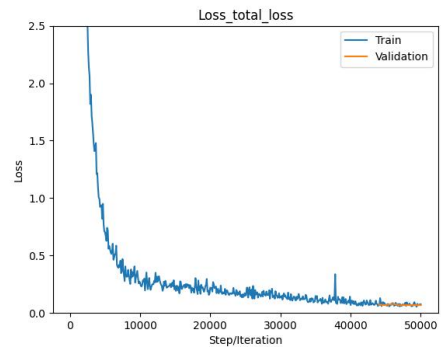
(a) Classification loss



(b) Localization loss



(c) Regularization loss



(d) Total loss

Figure 4.2: Train loss plots at every 100 steps and validation loss plots at the last 7000 thousand training steps, on Dataset A.

Only the last seven checkpoints were able to be retained, due to the fact that in the default configuration of the API, during training, only the last seven checkpoints were to be saved. This represents validation on the last 7000 steps, so it was not possible to make a full validation loss plot ¹. Nevertheless, it is visible that, at the end of the training, the validation loss converges similarly to the training loss indicating a good generalization.

This training was considered a preliminary one, with the intent to see how things work on the TFOD API, that is why only a small subset was chosen from LISA TL dataset. In this case, the performance of the model trained with this dataset was only tested on some internet images. These images contained traffic lights in similar conditions to those of the Dataset A, in terms of distance of the traffic light from the camera and also in other situations where traffic lights are closer to the camera. We could see that the model had more ease at detecting images in similar conditions than in different ones. Figure 4.3 shows two examples. In Figure 4.3a, out of three traffic lights the upper gets a detection, the background and lack of contrast in the lower ones

¹At this preliminary stage and for the sake of time, it was not considered relevant to repeat the training just to register the 50 checkpoints to make the full validation loss plot.

can be the cause. In Figure 4.3b the model is unable to identify any of the traffic lights present, the cause can be attributed to the fact that in Dataset A, there are not images where traffic lights occupy a significant portion of the image, similarly to this case. Thus, some details may be missed, as mentioned in Section 4.1.1.



(a) Sample 1 detection



(b) Sample 2 failed detection

Figure 4.3: Two images samples used to test the training in Dataset A ².

4.2 Gathering a second dataset

At this step, the goal is to further train our model by collecting more images containing traffic lights in the three different states. In order to achieve this, we will build a custom made dataset, that focuses more on diversity. A bigger dataset with images that have more proximity to the objects themselves and with different angles, in order to obtain a better, more diverse representation possible of the traffic lights. This is important to help the model to better learn the features of these objects so that performance can be enhanced due to improvement on generalization.

To create our dataset, we drove around Coimbra, Portugal taking videos of traffic lights using two smartphones. One with video resolution of 1280×720 with 30 frames per second (FPS) and another with resolution of 1980×1080 also with 30 FPS.

After filming these videos, they are splitted in frames with a python script to build our image dataset. This dataset will be designated as Dataset B.

4.2.1 Automatic labelling of the Dataset B

Before making the split into train and validation one must label all the images contained in the Dataset B. The same goes for the test dataset. Unfortunately, when we are dealing with big

²<https://press.siemens.com/global/en/feature/100-years-traffic-light>; <https://www.publituris.pt/2021/06/11/wttc-pede-abandono-do-falido-e-prejudicial-sistema-de-semaforos-britanico>

datasets, this can become a tedious and time-consuming process to do manually. Fortunately, in the present case, we can take advantage of the detector capabilities of the chosen pre-trained model. All models in the TensorFlow 2 Model Zoo were pre-trained with 90 different classes, and one of those classes is precisely the traffic light class, meaning that they can perform detection on them. This way it was possible to develop an algorithm, written in python, that is able to label our traffic light dataset automatically. However, these models can only detect traffic lights and classify them as so, which means that they cannot classify traffic lights accordingly to their color. Therefore, the only thing we can retrieve from the model are the coordinates for the bounding boxes of the traffic lights detected in each image. After this, all the detection must undergo an image processing procedure detailed below to find the color in each traffic light. The algorithm labels the dataset in the format generated by the LabelImg tool, which is an XML file. This is very useful because it allows not only to load back the labeled dataset into LabelImg to check if the labelling was done correctly but also to manually fix the labelling in case the automatic label has errors, whether it is in the class label or in the coordinates of the bounding boxes. This algorithm was written and tested on PC1, referred to in Table 3.1. We concluded that upon a correct detection of a traffic light, the color detection is correct over 95% of the times. Still, the model fails most of the times in detecting the exact position of the traffic light, because the coordinates do not cover the traffic light completely, possible reasons are explained later in this section. This means that over 50% of the labeled images needed to be adjusted in the position. Nevertheless, their classification label is correct most of the time, which makes this algorithm still useful. In situations where the image sequences had little variation between them, a single one was chosen.

The developed algorithm to label the dataset automatically a dataset can be divided in the following 5 steps:

- 1.** Neglect all the classes that are not traffic lights: This way the algorithm does not take into account the detection of other objects that may be presented in the image.

- 2.** From all the obtained detections, only those with a confidence score starting at 50% and a IoU smaller or equal than 40% are considered. Besides that, the algorithm will only consider a maximum of 50 detections for each image. These thresholds are defined with a function from TensorFlow called *tf.image.non_max_suppression*; the effects are illustrated on Figure 4.4.



Figure 4.4: Effect of Non Maximum Suppression ³.

3. Crop the image in the coordinates of the detections normalized to the image dimensions and save them in an array as well as the coordinate values.

4. Analyse the color of each traffic light detected using the OpenCV library [27] to identify the color presented and then attribute a label according to it (see details below). Although OpenCV reads images in BGR (red, green, blue), in this task is used the HSV (hue, saturation, value) color space representation. If we consider a situation where we have a plane with a blue color that has a shadow on top of it, the HSV's hue value, which indicates the color, is less prone to changes due to that lighting variation, whereas the BGR values have more variation, possibly leading to a wrong color detection [4]. This makes HSV color space more suitable for this task than BGR. Another disadvantage of BGR, more specifically to the present case, is that for the detection of yellow color, the red and green channels must be mixed, because, unlike red, green and blue, yellow is not a primary color in the RGB representation.

5. Write an output XML file: An XML file with the obtained information is written with the help of the xml.etree.ElementTree library [28]. Important information about the detected objects and respective image are stored in this function. The image data includes its name, folder and path, and size (width, height, and depth); the object information includes the classification label, bounding box coordinates (xmin, ymin, xmax, ymax). This information is written in the exact same way as LabelImg expects it.

Color detection algorithm

Steps 1 to 3 are achieved through the pre-trained model's capabilities. To illustrate these steps, we will take Figure 4.6 as an image containing traffic lights to be labeled. This figure already shows the output of step 2, that is, the detection obtained under those circumstances. Then, step 3 is illustrated with Figure 4.7.

³<https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c>

For step 4, we start by converting each cropped image from BGR to HSV color space. Then, the detection range of the HSV values for the different colors of the traffic lights must be defined. With the help of a HSV color map, shown in Figure 4.5, and a color picker python script from ⁴ [29], *colorpicker.py*, we are able to define those intervals. This script allows to load an image and adjust the HSV range interactively, so we could see how varying those intervals would isolate better the respective color. By using several sample images of traffic lights we can test various ranges and choose those that would give a better generalization for the different samples, because in the different samples the colors change slightly, for example, a traffic light presenting yellow can have a hue value in the orange range, and another presenting green can have a hue value in the blue range. The chosen ranges are: (172,95,186) to (179, 255,255) for red; (17,97,255) to (35,255,255) for yellow; (60,84,57) to (90,255,255) for green. After this, we create a mask using OpenCV for each color under these ranges that transforms the values of the pixels of the traffic lights detection into black and white pixels, where 0 and 255 correspond to black and white, respectively. If there are pixels in the defined range they will be replaced with white pixels otherwise with black ones. This mask is a 2D array with the size of the cropped image. Figures 4.8, 4.9 and 4.10 shows the three masks for each cropped detection of Figure 4.7. Now, since this array is made of 0's and 255's values, of the three obtain masks, the one with more white pixels is the dominant color corresponding to one of red, yellow or green filters. This way we can identify the color displayed by the traffic light at that moment. Moreover, for the case where only a small amount of each color is detected or none, which can mean that the traffic light is off, the detection is disregarded. This is done by summing each array inside of each of the three 2D array masks and then counting the zeros present in each of the resultant 1D arrays (with size of the width of the cropped detection). The more zeros present mean that the correspondent color in the range specified for the mask is less present. Then the average of the counted zeros for each resultant 1D array is computed and is subtracted to the size of one the resultant arrays (they are all of the same size). If the masks detected a very low quantity of color, their resultant 1D arrays will all have the number of zeros almost equal to that of their array size. Therefore, we considered that if the computed subtraction is less than 1, the traffic light is off.

⁴A comment from user nathancy on a Stackoverflow post.

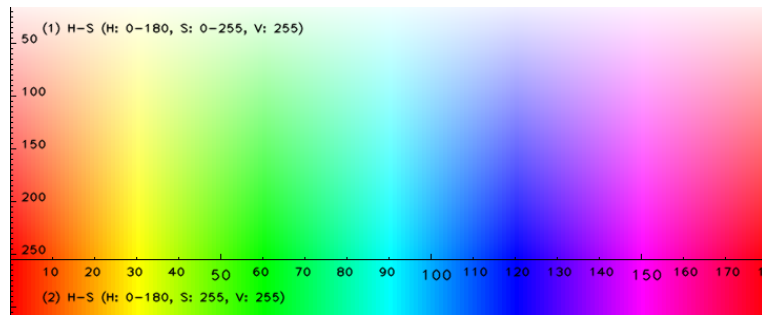


Figure 4.5: HSV color map ⁵.

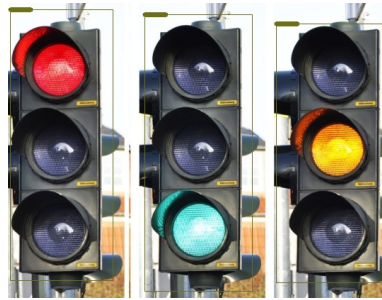
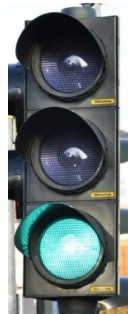


Figure 4.6: Sample image of traffic lights with bounding boxes generated by the pre-trained model, accordingly to step 2 ⁶.



(a)



(b)

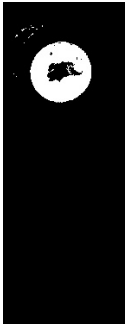


(c)

Figure 4.7: Cropped detection from Figure 4.6, accordingly to step 3.

⁵<https://github.com/AdityaPai2398/Colour-Segmentation-in-OpenCV>

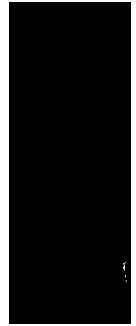
⁶<https://www.pngwing.com/>



(a) Red

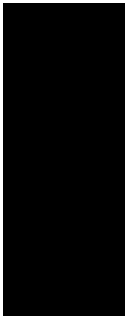


(b) Green



(c) Yellow

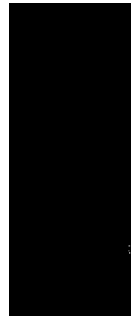
Figure 4.8: The three masks for the cropped detection of Figure 4.7a.



(a) Red



(b) Green



(c) Yellow

Figure 4.9: The three masks for the cropped detection of Figure 4.7b.



(a) Red



(b) Green



(c) Yellow

Figure 4.10: The three masks for the cropped detection of Figure 4.7c.

The algorithm has concluded that Figure 4.7a is a red traffic light, Figure 4.7b is a green traffic light and Figure 4.7c a yellow one. Finally the algorithm writes the XML file with all the necessary information. The script for the labelling algorithm and the XML file created for the previous example are presented in Appendix A. In the beginning of the labelling script the function *readClasses*, *loadModel* and some part of the function *createBoundingBox* were taken from [30].

Discussion

This algorithm is more useful when using images where traffic lights occupy greater portions of the image, that is, when it is closer to the camera, similarly to Figure 4.6 where each traffic light occupies around 20% of the total image. In Dataset B, images where traffic lights occupy around 2% of the images still got a good detection. However, in images where the traffic light occupies less than that, the detection gets less accurate, which may require manual adjustments. In cases like these, traffic lights may fit the category of small objects. The SSD algorithm is known to be less effective in detecting small objects [31]. Besides this, another factor to be taken into account in the detection is the contrast between the traffic light and the background, because greater contrast seems to yield a better detection. In Section 4.3.8 and 4.3.9 there is a statistical study of the possible effects of background and image contrast.

In Figure 4.11a we can see that out of three traffic lights two have been missed. Only the left one got some part detected, because the bounding box does not cover it completely. This can also be possibly attributed to the fact that beneath the traffic light there is a traffic sign. Also, the low contrast of the traffic light from the right relatively to the background can be a reason for that failed detection. The upper traffic light failed detection can possibly be attributed to being a small object relatively to the image.

In Figure 4.11b it is possible to see that the upper traffic light got a better detection. There is only one bounding box. The lower traffic light has buildings in the background and, in addition, is hold on a mast that goes along its height. These features can lead the model to "think" there is more than one traffic light in that spot, degrading its efficiency.



(a) Small traffic lights when compared to the image size.



(b) Contrast situation.

Figure 4.11: Situations of bad detections by the pre-trained model on MS COCO.

4.3 Further training with the new data

At this stage the purpose was to further train our model with our new labelled dataset. The goal here is to start training from the last checkpoint created during the training from Section 4.1. This means the training will start at step 50 000. With this approach it is expected that the previously acquired knowledge is retained. From now on, all the training is performed on PC3, referred in Table 3.1.

Whenever the training process shows converging behavior (i.e. when the decreasing of the losses reaches a point where there is not a significant change in their values for the last few thousand steps) in the validation loss, a test dataset containing 75 images will be used to test the model and get the COCO detection metrics detailed in Section 3.5.10. This dataset consists of 62 red traffic lights, 12 yellow traffic lights and 64 green traffic lights. This dataset will remain

constant throughout the work. Table 4.8 shows information about the different datasets used throughout this work.

4.3.1 Training 0

Dataset B contains a total of 762 images in which a 85% and 15% split is done, for training and validation, respectively. The training set contains 297 red traffic lights, 90 yellow traffic lights and 405 green traffic lights. The validation set contains 77 red traffic lights, 27 yellow traffic lights and 108 green traffic lights.

Concerning the improvement of the training performance, it was decided to increase the batch size from 32 to 64, to see if it would improve the training speed, and contribute to a faster convergence. In order to prevent overfitting of the training and promote a better generalization [32], a dropout rate of 20% was also applied to the model during the training stage. Dropout works by randomly forgetting some neurons of the neural network that could cause overfitting due to the possibility of them trying to compensate errors of other neurons, during the training process. This can lead to learning unwanted features of the training data, such as noise, that are not present on unseen data. By dropping randomly some of the neurons, this situation is unlikely to happen. There were two default data augmentation options applied for the training data in the default pipeline configuration file which are: SSD random crop and random horizontal flip. The first one crops an image randomly, which is explained in greater detail in Section 4.3.7, and the second one has a mirror effect on the training images and happens 50% of the time. Although in the training from Section 4.1, with Dataset A, it apparently did not cause problems, the SSD random crop was removed from this train. The reason to do so, concerns the suspicious it would cut the ground truth bounding boxes of the training data in the color areas of the traffic lights that are crucial for the model to learn to distinguish between them.

Two additional data augmentation techniques were added: random brightness adjustment and random image scalling, the first randomly changes the brightness by up to a defined maximum value and the second expands or shrinks the image also randomly.

The first training with Dataset B was set to run for 110 000 steps, under these new conditions, on PC3, referred in Table 3.1. The training ran for over a week, but the training loss metrics could not converge at all, presenting huge loss values. The high default *learning_rate_base* hyperparameter of the model’s optimizer, which was set 0.8, was suspected to be the cause. The model uses an optimizer, called Momentum, a variant of Stochastic Gradient Descend (SGD) [33], to update its weights after the calculation of the loss of a prediction for the training samples via a loss function in order for the output of the prediction to get as close as possible to the

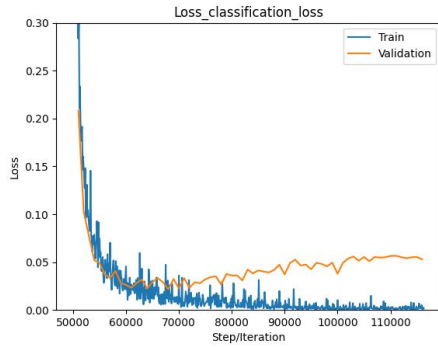
actual value. This process of updating the weights is known as backpropagation and is done from the output layer to the input. When the learning rate is too high, the updating of the weights can become too large and the training loss can increase as well [34].

The relevant learning rate parameters that can be changed to try to overcome this issue are the *learning_rate_base* and *warmup_learning_rate*, respectively, the highest learning rate value reached by the model before decreasing and the initial training learning rate of the model.

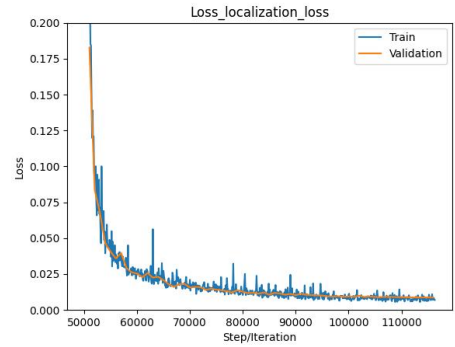
4.3.2 Training 1

We start this training with the same hyperparameters as in the previous subsection except now we are changing the *learning_rate_base* to 0.08 and the *warmup_learning_rate* to 0.0133. Besides this, for post processing, a minimum confidence score threshold of 0.2 is set to filter proposals (i.e. coordinates for objects and their respective classification, output by the model) that are likely to be incorrect, contributing to a more stable training [35]. By taking in consideration what was mentioned in Section 4.3, the training must start from step 50 000 and finish at step 170 000. The training was unable to reach the step 170 000, as the software crashed, reaching only the step 120 000.

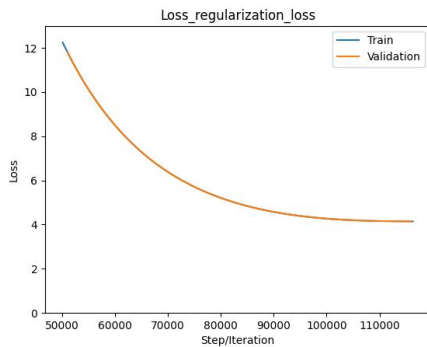
As shown in Figure 4.12, in spite of the crash, the training losses have now converged. The regularization loss, though, remains quite high as it did not converge to losses close to those of the training classification or the localization training loss ones. This contributed to a high total loss. The consequences of the high total loss can be seen in the validation loss of Figure 4.12a as it starts diverging after step 60 000, showing a typical sign of overfitting to the training data.



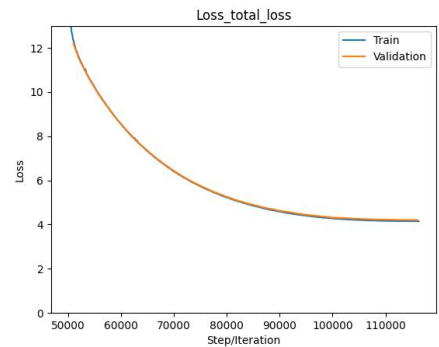
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



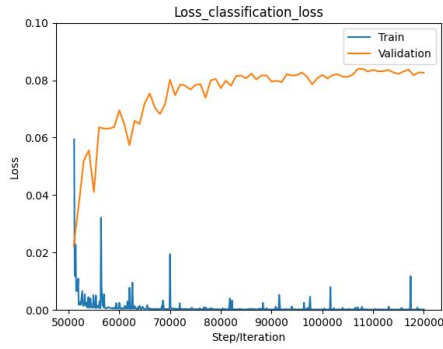
(d) Total loss.

Figure 4.12: Train and validation loss plots for the different training iterations of Training 1 for Dataset B.

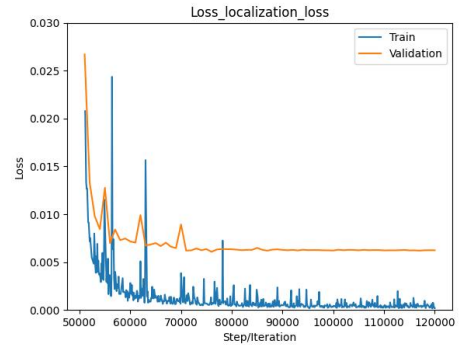
4.3.3 Training 2

To try to solve the issues from Training 1 (Section 4.3.2), a lower *learning_rate_base* was now used with a value of 0.02, while keeping the same values earlier defined for the hyperparameters. Moreover, we have now dismissed all the previously applied data augmentation techniques.

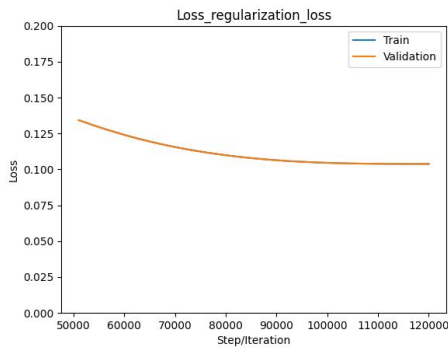
This time, the regularization loss converges to much lower values as can be seen in Figure 4.13c. On the other hand, the validation loss shown in Figure 4.13a seems to diverge more and at an earlier step than the one from Figure 4.12a. The validation loss in Figure 4.13b converges, but is higher than shown in Figure 4.12b. Although all training losses seem to converge now, the total loss in the validation dataset, shown in Figure 4.13d, still indicates a model with poor generalization.



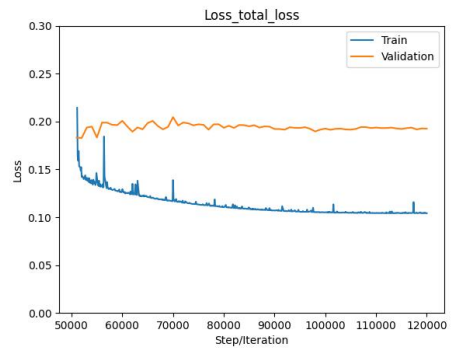
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



(d) Total loss.

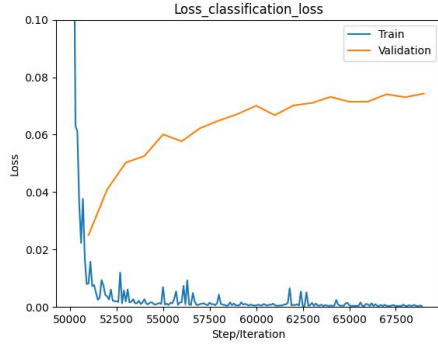
Figure 4.13: Train and validation loss plots for the different training iterations of Training 2 for Dataset B.

4.3.4 Training 3

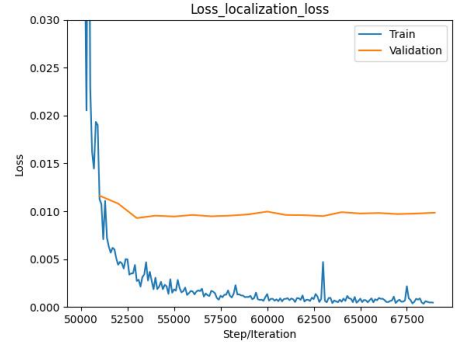
From now on, it was decided that training will only run for around 20 000 steps. This is because it is observable from Figure 4.12a and Figure 4.13a that the validation loss starts to diverge before 20 000 steps. The benefit of this approach is to not waste a lot of a time and select only promising trains.

For this training the two learning rate hyperparameters are further reduced; *learning_rate_base* is now 0.01 and the *warmup_learning_rate* to 0.001.

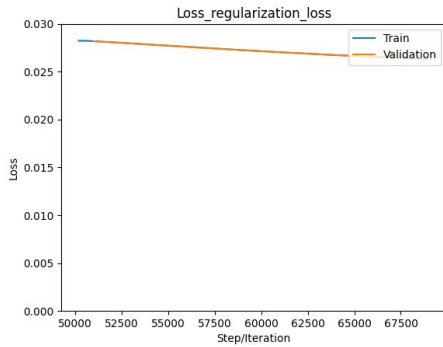
From Figure 4.14 it can be seen that no significant improvement was observed when compared to Figure 4.13.



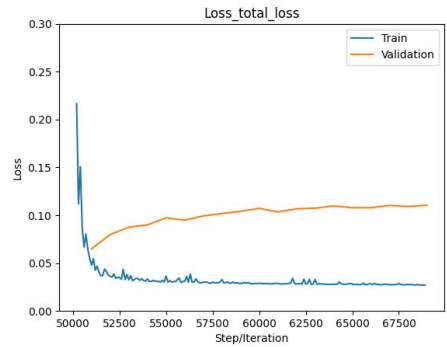
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



(d) Total loss.

Figure 4.14: Train and validation loss plots for the different training iterations of Training 3 for Dataset B.

4.3.5 Training 4

From the last three trainings, it is possible to see that the classification loss of the validation dataset maintained a diverging behavior. This metric describes the ability of the model to correctly identify the label for each detection. By observing the number of traffic lights of each class in the training set, mentioned in Section 4.3.1, one can conclude that the set is not well balanced as each class has a disparate number of elements. In face of these conditions, the model may be more prone to classification errors. In images containing a traffic light of a class that has a lower representativity than the others, it may get its classification as one of the more representative classes. Such situation can be an obstacle to a good generalization and can possibly be the culprit for the problem. The localization loss does not seem to be affected by this issue probably due to the fact that regardless of the color presented, traffic lights have always the same shape and size proportions.

In order to find out whether the lack of a balanced dataset is the source of the problem or not, a subset of Dataset B can be made. To approach this, we decided to get some sample images from

the training set and test them on the exported model from Section 4.3.3. We choose the model over the one presented in Section 4.3.4 because the former was trained for more steps. Then, those samples are subjected to image processing involving a random change in brightness, with the constraint that the change is equal for all of them. After that, the processed samples were also tested. As expected, the training samples got a perfect detection with a 100 % confidence score, an example of such can be seen in Figure 4.15a. The same did not happen with the processed ones, which was also expected. So, in order to balance the dataset, we removed similar images in terms of background and placement of the traffic lights to the samples that were processed and got no detection at all, Figure 4.15b shows an example of that case. We were able to only balance red traffic lights and green traffic lights, because yellow traffic lights correspond to a much smaller percentage of the Dataset B since they are more difficult to gather. This subset of Dataset B will be called Dataset B1. Dataset B1 is composed by 218 red traffic lights, 66 yellow traffic lights and 222 green traffic lights from a total of 330 images. A split of 90% and 10% is then performed on the dataset for training and validation, respectively. No hyperparameter was changed.



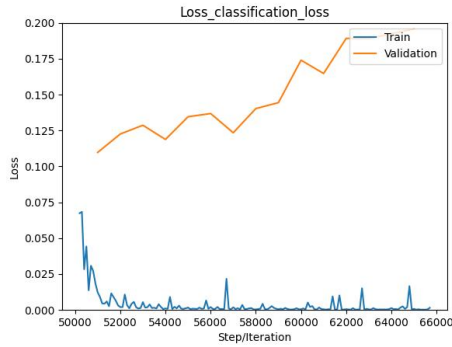
(a) Detection performed on the original training sample.



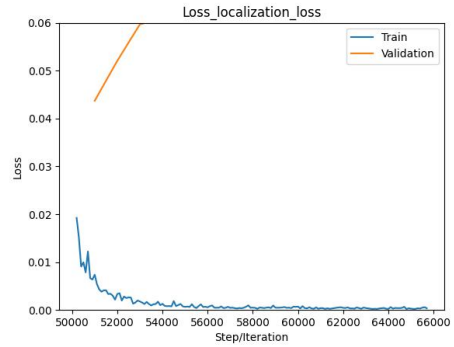
(b) Failed detection on the processed training sample.

Figure 4.15: Detection of a training sample from Dataset B, with the original brightness level and with a different one.

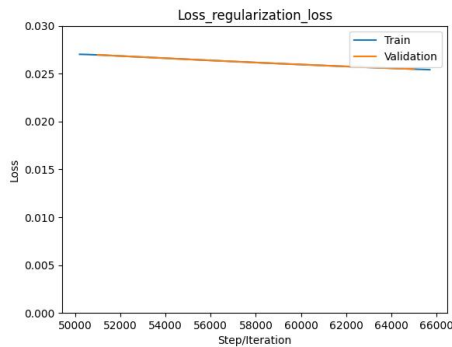
Figure 4.16 shows that this approach is not satisfactory. Now, besides the classification loss in the validation dataset, the localization loss of the validation dataset is also diverging, which can happen due to the decreasing of the training images.



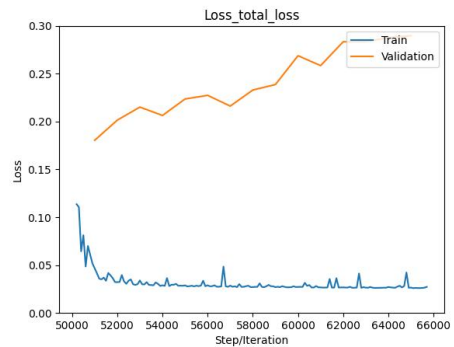
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



(d) Total loss.

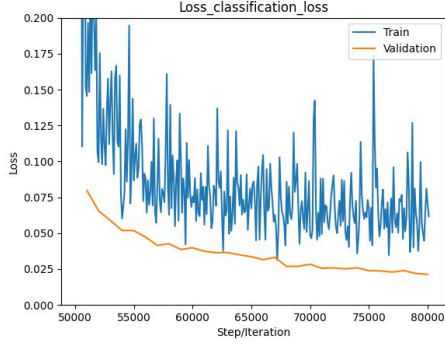
Figure 4.16: Train and validation loss plots for the different training iterations of Training 4 for Dataset B1.

4.3.6 Training 5

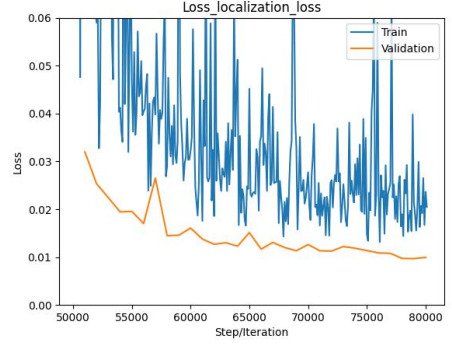
At this stage, data augmentation techniques were again implemented: the SSD random crop, as it did not harm training from Section 4.1, as well as random horizontal flip. Furthermore, hyperparameters such as batch size and learning rate are changed. The batch size is lowered to 32. Dropout use is no longer implemented. The learning rate parameters: *learning_rate_base* and *warmup_learning_rate* are set to 0.02 and 0.01, respectively. Now, we return to Dataset B.

Finally, from Figure 4.17d, it is possible to see a convergent behavior in all the plots. Maybe, SSD random crop has actually a positive effect. An interesting situation can be seen: all validation loss plots, except the one from Figure 4.17c, converge to even lower values than the training loss.

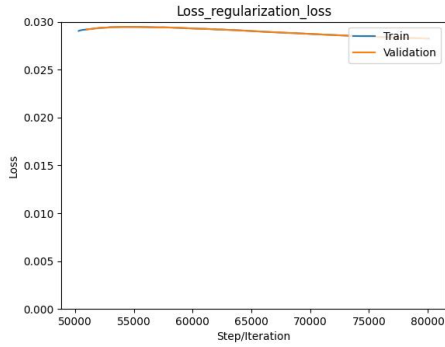
Table 4.1 shows the COCO detection metrics, Section 3.5.10, for the test dataset.



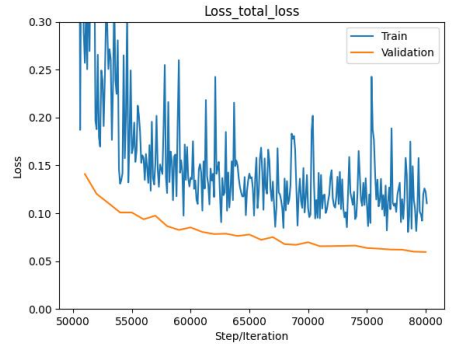
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



(d) Total loss.

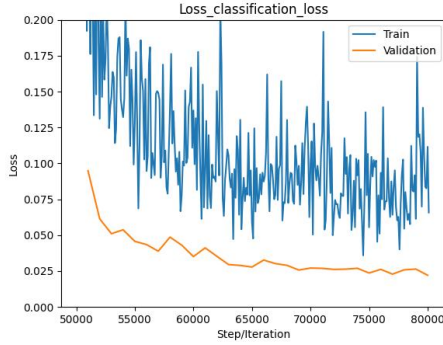
Figure 4.17: Train and validation loss plots for the different training iterations of Training 5 for Dataset B.

Table 4.1: Average precision and average recall COCO detection metrics for Training 5.

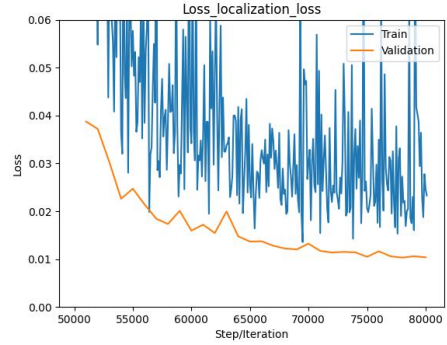
• Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]	0.150
• Average Precision (AP) @[IoU=0.50 area=all maxDets=100]	0.296
• Average Precision (AP) @[IoU=0.75 area=all maxDets=100]	0.151
• Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]	0.012
• Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]	0.290
• Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]	0.323
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 1]	0.205
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 10]	0.314
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 100]	0.333
• Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets= 100]	0.015
• Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets= 100]	0.396
• Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets= 100]	0.558

4.3.7 Training 6

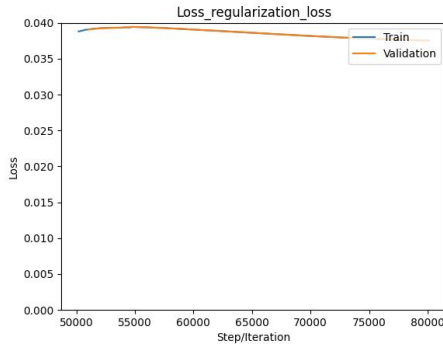
For this training, the dropout was again enabled to check if it produces any improvement. Analyzing the plots from Figure 4.18 it is possible to see that there is a convergence similar to those of the plots shown Figure 4.17.



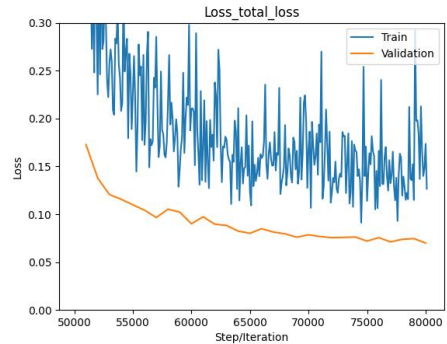
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



(d) Total loss.

Figure 4.18: Train and validation loss plots for the different training iterations of Training 6 for Dataset B.

However, by observing the metrics from the test on Table 4.2, the average precision values improved over the ones from Table 4.1. Yet, the average recall values got slightly worse in some parameters, but an improvement is obtained on small objects. This hints that the use of dropout appears to be useful.

From the behavior of the validation loss plots of both Figure 4.17 and Figure 4.18, we arrive to the conclusion that the data augmentation technique SSD random crop actually helps in the convergence of this metric, contrarily to preliminary assumptions.

Since traffic lights sometimes meet the category of small objects in our dataset, their detection on SSD models can be weak, as mentioned in Section 4.2.1. This SSD model applies image down-sampling on the training images before training, they are resized to 300×300 . If the training

images have high resolution compared to the resized ones and the objects of interest are of small size, during this process those objects can become indistinct or even disappear. One way to tackle this issue is to crop the training images into smaller ones with the help of overlapping tiles. This method results in multiple images that get their ground truth boxes arranged according to each image. In this case, the full frame and the cropped images are used as input training data [36]. The SSD random crop does a somewhat similar work as it also consists in taking smaller regions of the original image. The image is first expanded and then is cropped in such a way that the crop must overlap with no less than a ground truth box, while the centroid of at least a ground truth box must be also present there. The latter is combined with data augmentation horizontal flip, helping to improve the performance a lot, namely on small objects [37].

Overall, data augmentation techniques are very useful to reduce overfitting and promote better generalization, since it artificially increases the dataset by creating new images through transformations to the default training images. The drawback is the consequent increasing in the necessary training time [38]. Still, not every data augmentation available is useful for our problem. An example is the random RGB to gray one, also available on the TFOD API. This type of augmentation is not useful in our case since it would transform the traffic lights colors to gray in the images that suffer the transformation.

Table 4.2: Average precision and average recall COCO detection metrics for Training 6.

• Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]	0.202
• Average Precision (AP) @[IoU=0.50 area=all maxDets=100]	0.490
• Average Precision (AP) @[IoU=0.75 area=all maxDets=100]	0.137
• Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]	0.108
• Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]	0.293
• Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]	0.313
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 1]	0.196
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 10]	0.286
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 100]	0.296
• Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets= 100]	0.115
• Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets= 100]	0.404
• Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets= 100]	0.442

4.3.8 Training 7

For this training, we decided to test an image from Dataset A, mentioned in Section 4.1.1, to verify if the model is not forgetting the knowledge acquired from the training with that dataset. The testing was made with the trained model from Subsection Training 6. There was no detection at all, which indicates that the model, indeed, forgot that knowledge. Thus, assumption made in Section 4.3 does not hold.

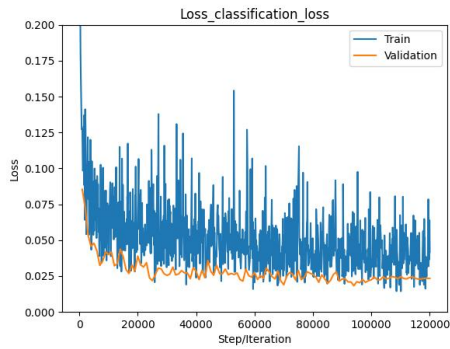
Now, Dataset A will be merged with Dataset B as well as an additional set of images from Coimbra to build a final dataset of 1324 images, which we will call Dataset C. The new images also went through the process presented in Section 4.2.1. A split of around 90% and 10% is made for training and validation, respectively, resulting in 1192 images for training and 132 for validation. The training set contains 789 red traffic lights, 312 yellow traffic lights and 1050 green traffic lights. The validation set contains 91 red traffic lights, 38 yellow traffic lights and 104 green traffic lights.

This training is set to start from step 0, that is, from the last checkpoint of the pre-trained model with the COCO dataset. The *learning_rate_base* is set to 0.01 and the *warmup_learning_rate* is set to 0.001. The rest of the hyperparameters are the same as in training from Section 4.3.7 and the train is set to run for 120 000 steps.

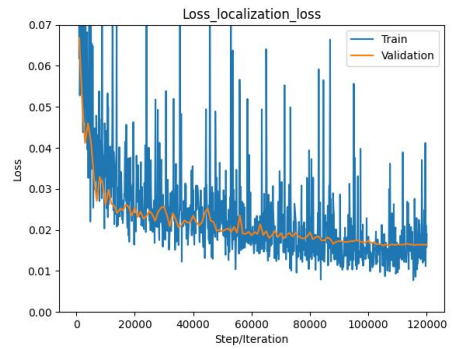
Figure 4.19 shows the different loss plots for training and validation datasets. Table 4.3 shows the metrics of the test dataset.

From the analysis of the detections on our test dataset, with a confidence threshold of 50% within the NMS function, referred in Section 4.2.1, it is possible to see that 56.5% of the red traffic lights, 50.0% of yellow traffic lights and 68.8% green traffic lights have been correctly detected. The test dataset contains traffic lights with a single traffic light similar to images from Figure 4.15 and also traffic lights containing two traffic lights one at the top of the bar and other at the bottom, similar to Figure 4.11b. Further analysis showed that the single traffic lights were detected with a probability of 57.7%, the top traffic lights with a probability of 84.3% and bottom traffic lights with a probability of 58.8%. Although six of the upper traffic lights were not correctly classified (these were classified as yellow traffic lights where they are red traffic lights), the capability of the model to detect these is much higher than in the other situations. By observing each image of the dataset, it is noticeable that the background is quite smooth for the upper traffic lights (e.g. sky with fuzzy clouds). In such a situation, there is a high contrast between the traffic light and the sky. Therefore, traffic light features are more prominent, allowing an easier detection for the model. For the remaining traffic lights, as they are in a lower position most of the times, their background comprises, very frequently, buildings

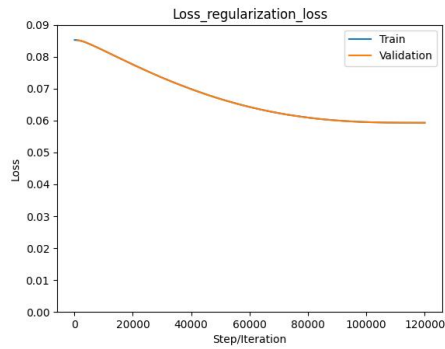
or vegetation, resulting in a lower contrast and, consequently, lacking feature distinction, leading to poor detection.



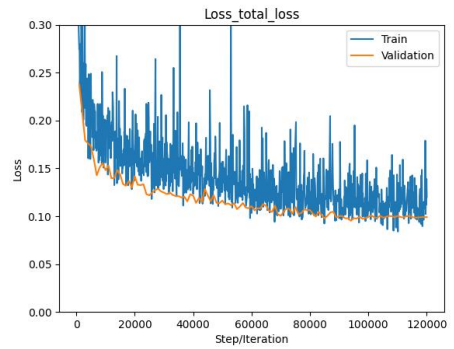
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



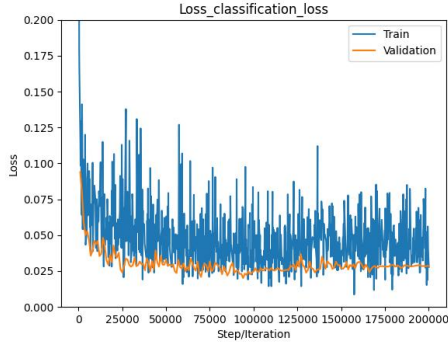
(d) Total loss.

Figure 4.19: Train and validation loss plots for the different training iterations of Training 7 for Dataset C.

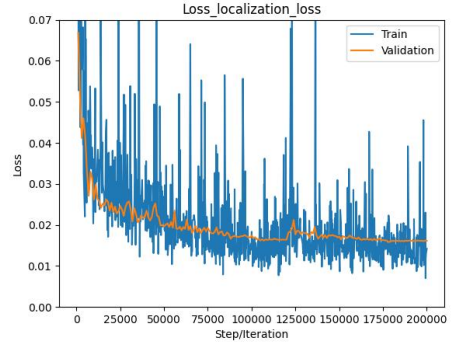
Table 4.3: Average precision and average recall COCO detection metrics for Training 7.

• Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]	0.497
• Average Precision (AP) @[IoU=0.50 area=all maxDets=100]	0.793
• Average Precision (AP) @[IoU=0.75 area=all maxDets=100]	0.527
• Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]	0.233
• Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]	0.649
• Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]	0.776
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 1]	0.487
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 10]	0.671
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 100]	0.671
• Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets= 100]	0.263
• Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets= 100]	0.730
• Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets= 100]	0.792

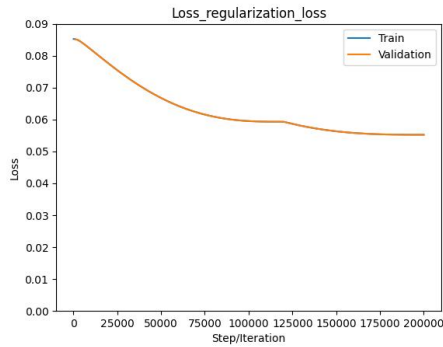
To further improve the detection of the model, we set the model to train for more 80 000 steps. In order to try to fix the classification errors, we set the *classification_weight* from 1 to 1.1 in order for the loss function to prioritize more the classification weights. Figure 4.20 shows the plots of the training losses. Table 4.4 shows the metrics on the test dataset.



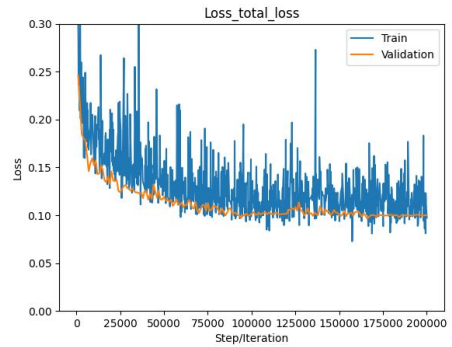
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.



(d) Total loss.

Figure 4.20: Train and validation loss plots for the different training iterations (with more 80 000 steps) of Training 7 for Dataset C.

Table 4.4: Average precision and average recall COCO detection metrics for Training 7 with more 80 000 steps of training.

• Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]	0.508
• Average Precision (AP) @[IoU=0.50 area=all maxDets=100]	0.808
• Average Precision (AP) @[IoU=0.75 area=all maxDets=100]	0.563
• Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]	0.210
• Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]	0.663
• Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]	0.766
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 1]	0.482
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 10]	0.663
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 100]	0.664
• Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets= 100]	0.237
• Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets= 100]	0.739
• Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets= 100]	0.783

Table 4.4 shows a slight improvement in the overall average precision. The average recall values decrease slightly in some cases. However, there is not a significant improvement. At step 120 000 the model seems to have stabilize.

The analysis of the detections on the test dataset shows a 61.29% of correct red light detection, a 62.5 % of correct green traffic light detections and a 50% of correct yellow light detections. Upper traffic lights are detected 76.5% of the time, single traffic lights are detected 61.5%, and bottom traffic lights 58.8%. The wrongly classified traffic lights still get their classification wrong. Table 4.5 shows no improvement in the average of the correct detections.

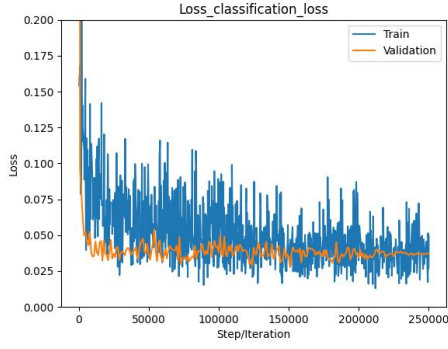
Table 4.5: Percentage of correct detection for Training 7.

	Red traffic light	Yellow traffic light	Green traffic light	Average detections
Training 7 120000 steps	56.5%	50.0%	68.8%	58.4%
Training 7 200000 steps	61.3%	50.0%	62.5%	57.9%

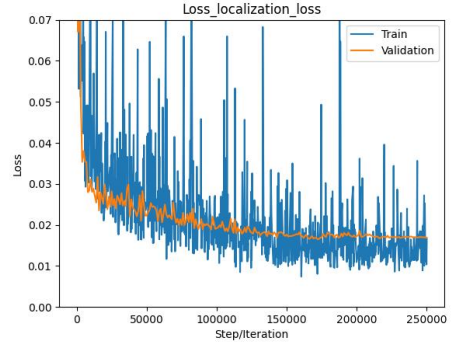
4.3.9 Training 8

As a final attempt to improve the model, the training was set to start from step 0 and to run for 250 000 steps. The configurations for the hyperparameters are the same used in Section 4.3.8, expect now data augmentation random brightness adjustment is added. Figure 4.21 shows the loss for the training and validation datasets and Table 4.6 shows the metrics for the test dataset.

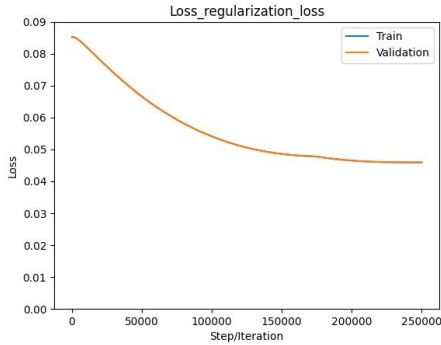
Overall, Table 4.6 shows some improvement on the Average Precision (AP) and Average Recall (AR) when compared to results shown in Table 4.4. However, the average precision and average recall on small objects decreased when compared to both Table 4.3 and Table 4.4. Figure 4.22 and Figure 4.23 show the evolution of the detection on the validation set of Dataset C for each COCO detection metric for AP and AR, respectively. An indication of this problem can be seen in Figure 4.22d and Figure 4.23d. The AP and AR for small objects on the validation set seem to decrease after step 175 000.



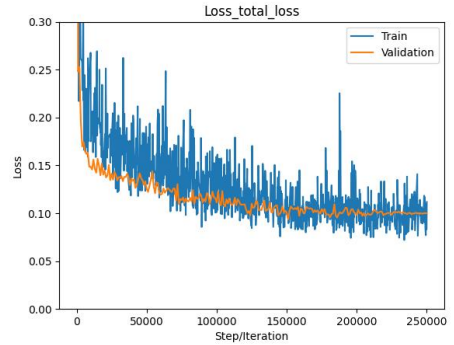
(a) Classification loss.



(b) Localization loss.



(c) Regularization loss.

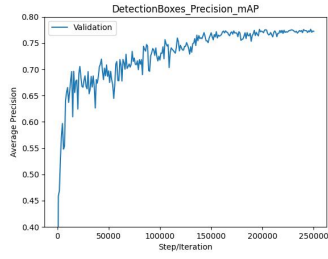


(d) Total loss.

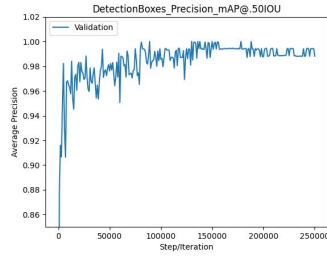
Figure 4.21: Train and validation loss plots for the different training iterations of Training 8 for Dataset C.

Table 4.6: Average precision and average recall COCO detection metrics for Training 8.

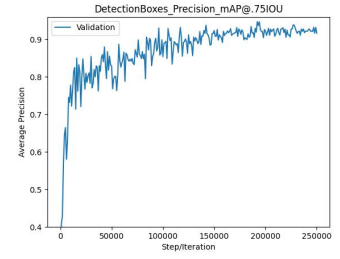
• Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]	0.583
• Average Precision (AP) @[IoU=0.50 area=all maxDets=100]	0.830
• Average Precision (AP) @[IoU=0.75 area=all maxDets=100]	0.670
• Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]	0.118
• Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]	0.609
• Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]	0.777
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 1]	0.520
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 10]	0.645
• Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets= 100]	0.646
• Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets= 100]	0.122
• Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets= 100]	0.672
• Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets= 100]	0.786



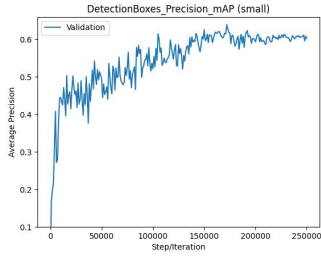
(a) AP IoU=0.50:0.95 all.



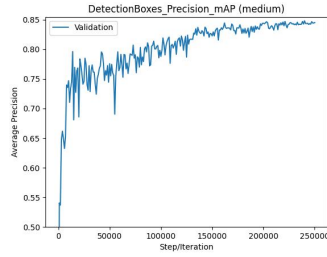
(b) AP IoU=0.50 all.



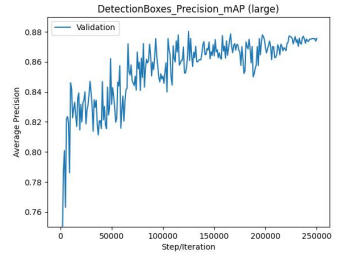
(c) AP IoU=0.75 all.



(d) AP IoU=0.50:0.95 small.

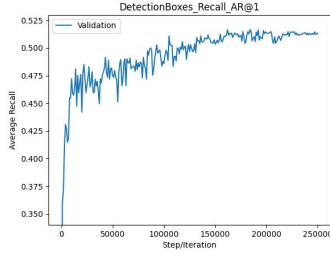


(e) AP IoU=0.50:0.95 medium.

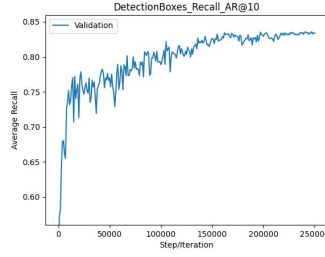


(f) AP IoU=0.50:0.95 large.

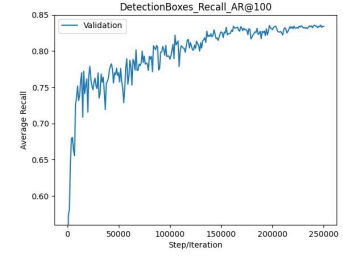
Figure 4.22: Evolution of the average precision of the detection boxes under the different criteria of COCO detection metrics for the validation set of Dataset C during the different training iterations.



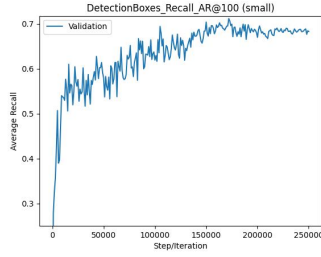
(a) AR IoU=0.50:0.95 all maxDets=1.



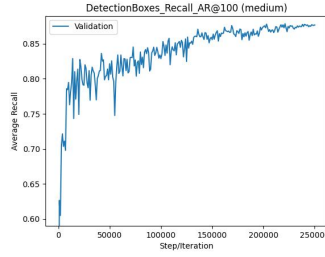
(b) AR IoU=0.50:0.95 all maxDets=10.



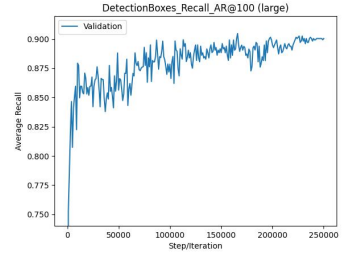
(c) AP IoU=0.75 all maxDets=100.



(d) AP IoU=0.50:0.95 small maxDets=100.



(e) AP IoU=0.50:0.95 medium maxDets=100.



(f) AP IoU=0.50:0.95 large maxDets=100.

Figure 4.23: Evolution of the average recall of the detection boxes under the different criteria of COCO detection metrics for the validation set of Dataset C during the different training iterations.

Regarding the analysis on the test dataset, the red traffic lights are now correctly detected 58.1% of the time, yellow traffic lights 83.3% and green traffic lights 70.3%. The failed classifications from Section 4.3.8 have been reduced from six to just one, showing the positive effect of adding random brightness adjustment data augmentation. Single traffic lights are detected correctly 61.53% of the time, top traffic lights 80.4% of the time and bottom traffic lights 47.1%. To check whether lowering the confidence score threshold (CST) of the NMS function would help to get more, but also, correct detection, it was lowered from 50% to 35%. By repeating the same analysis on the test dataset, we now obtain 67.7% of the red traffic lights correctly detected, 100% of yellow traffic lights correctly detected and 75% of the green traffic lights correctly detected. Single traffic lights were now detected correctly 65.4% of the time, top traffic lights 84.3% of the time and bottom traffic lights 68.6%. This analysis show that the detection has improved without the appearance of redundant or wrong detection due to a lowered confidence threshold. An improved recall can be seen, as more traffic lights were detected. This agrees with what was mentioned about a lower confidence threshold in Section 3.5.4. Table 4.7 shows the improvement on correct detection over the ones shown in Section 4.3.8. The lower percentage of correct detection in the bottom traffic lights and single traffic lights can be again attributed

to a lack of contrast against the background, due to them being placed closer to the ground; an example is shown on the left traffic light of Figure 4.24b.

The test dataset is mainly constituted by groups of sequences of frames of traffic lights in a certain state, in order to better represent a vehicle in a road situation. In this case, there are several frames where traffic lights are in regions of low contrast, thus influencing negatively their detection. Still, all of the 75 images on the test dataset got at least one correct detection of the location of a traffic light, which gives a 100.0% probability of locating a traffic light for each image. In terms of the classification accuracy, out of the 92 traffic lights correctly located only one got wrongly classified, which gives a 98.9% probability of correctly classifying a traffic light detected.

This means that even if there is only one traffic light detected out of three in an image, an automated vehicle can still receive the correct information about the traffic lights present and their state. This is true in our situation, where we do not consider traffic lights controlling changing in traffic direction to the left or right.

Table 4.7: Percentage of correct detection in the last two training.

	Red traffic light	Yellow Traffic light	Green Traffic light	Average detection
Training 7 120000 steps	56.5%	50.0%	68.8%	58.4%
Training 7 200000 steps	61.3%	50.0%	62.5%	57.9%
Training 8	58.1%	83.3%	70.3%	70.6%
Training 8 35% CST	67.7%	100.0%	75.0%	80.9%

Figures 4.24, 4.25 and 4.26 shows some sample detection on the test dataset.



(a) Green traffic light detection sample 1.



(b) Green traffic light detection sample 2.

Figure 4.24: Green traffic light detection samples on the model from Training 8.



(a) Yellow traffic light detection sample 1.



(b) Yellow traffic light detection sample 2.

Figure 4.25: Yellow traffic light detection samples on the model from Training 8.



(a) Red traffic light detection sample 1.



(b) Red traffic light detection sample 2.

Figure 4.26: Red traffic light detection samples on the model from Training 8.

Table 4.8: Information about the different datasets used.

	Red traffic lights	Yellow traffic lights	Green traffic lights	Number of images
Dataset A	395	84	554	366
Dataset B	374	117	513	762
Dataset B1	218	66	222	330
Dataset C	880	350	1154	1324
Test dataset	62	12	64	75

5 Conclusion and future work

The purpose of this work was to train a neural network object detection model to be able to localize in an image a traffic light and then classify it accordingly to its color. The framework TensorFlow Object Detection API in conjunction with a model from TensorFlow 2 Detection Zoo model, proved to be a very efficient way, overall, to achieve the purposed goals. With the use of two datasets from different sources, one available online, the LISA traffic light dataset, and a dataset taken in Coimbra, we were able to create a traffic light detector with a 100.0% probability of location of a traffic light per image on the test dataset and a probability of 98.9% of correctly classifying each traffic light detected. The creation of an automatic labelling algorithm helped to decrease considerably the time expensive task of manually labelling a dataset.

During the implementation phase, it was concluded that high learning rates can be detrimental to the training process, as the updating of the weights can become too large leading to a high training loss and problems to converge. Moreover, although an unbalanced dataset is not ideal, it was found that it was not the cause for the observed diverging effect on the validation loss plots. Furthermore, SSD's poor performance to detect small objects was found to be the major cause of the lack of convergence on the validation loss plots. Usually, SSD random crop is used to help with such situation and by applying it, together with random horizontal flip in our model, the validation loss plot behavior started to converge. The usage of a dropout rate of 20% showed benefits at improving the overall average precision and some of the average recall criteria on the test dataset. When expanding the training dataset size, the result metrics on the test dataset increased significantly. Also, by adding another relevant data augmentation technique to the traffic light problem, the random adjust brightness, further improvement could be seen on the test dataset. Adding more data augmentation techniques required more steps/time for the training to converge. Furthermore, from the analysis on the test dataset we arrived to the conclusion that upper traffic lights are more easily detected due to a better contrast between them and their background. As a consequence, their features are prominent and the model is less prone to mistakes. Lastly, by lowering the confidence threshold from 50% to 35%, an increase

in the correct detection can be seen without the appearance of wrong or redundant detection. Lower layers should have been frozen to perform transfer learning. Also, class weights should have been used in order to rebalance the dataset. In Chapter 4, systematic hyperparameter tuning should have been helpful for a more systematic study of the experiments performed.

As a suggestion for future work, we would increase the size of the dataset in an evenly way in order to get equal representation for all the classes as well as increasing the diversity of the background. This way, it is expected the probability of detection for each class to be more even. It would be interesting to change other hyperparameters such as the *depth_multiplier*, which is related to the feature extractor of the model, the mobile net. This parameter serves the purpose of reducing the size of the model, which can affect the inference time of the model. Adding traffic lights with right and left directions to the dataset would also be of particular interest as these kind of traffic lights are also present in the road environment. Finally, trying a YOLO model would also be a good approach to continue this work as it is also a fast one and good for object detection. For the moment YOLO models are not present in the TensorFlow 2 Model Zoo, but that might change in the future.

Bibliography

- [1] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.
- [2] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, IEEE, 2017.
- [3] R. Padilla, S. L. Netto, and E. A. Da Silva, “A survey on performance metrics for object-detection algorithms,” in *2020 international conference on systems, signals and image processing (IWSSIP)*, pp. 237–242, IEEE, 2020.
- [4] M. P. Philipsen, M. B. Jensen, A. Møgelmoose, T. B. Moeslund, and M. M. Trivedi, “Traffic light detection: A learning algorithm and evaluations on challenging dataset,” in *intelligent transportation systems (ITSC), 2015 IEEE 18th international conference on*, pp. 2341–2345, IEEE, 2015.
- [5] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, “Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions,” *Journal of big Data*, vol. 8, no. 1, pp. 1–74, 2021.
- [6] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [7] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, “Deep learning in spiking neural networks,” *Neural Networks*, vol. 111, pp. 47–63, 2019.
- [8] A. Dhillon and G. K. Verma, “Convolutional neural network: a review of models, methodologies and applications to object detection,” *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

- [9] N. Aloysius and M. Geetha, “A review on deep convolutional neural networks,” in *2017 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0588–0592, IEEE, 2017.
- [10] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, “Object detection with deep learning: A review,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.
- [11] W. Zhiqiang and L. Jun, “A review of object detection based on convolutional neural network,” in *2017 36th Chinese Control Conference (CCC)*, pp. 11104–11109, IEEE, 2017.
- [12] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li, “Development of convolutional neural network and its application in image classification: a survey,” *Optical Engineering*, vol. 58, no. 4, p. 040901, 2019.
- [13] T. developers, “Tensorflow.” <https://zenodo.org/record/6574269#.Yzr1DUrMK-w>, accessed October 2022.
- [14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7310–7311, 2017.
- [15] TensorFlow, “Tensorflow 2 model zoo.” https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md/, accessed March 2022.
- [16] H. Zanini, “Blog tensorflow custom object detection.” <https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html/>, accessed May 2022.
- [17] tzutalin, “Labelimg.” <https://github.com/tzutalin/labelImg/>, accessed March 2022.
- [18] S. Raschka, “Model evaluation, model selection, and algorithm selection in machine learning,” *arXiv preprint arXiv:1811.12808*, 2018.
- [19] Z. Reitermanova *et al.*, “Data splitting,” in *WDS*, vol. 10, pp. 31–36, Matfyzpress Prague, 2010.
- [20] malgo1311, “split.train.validation.” https://github.com/malgo1311/Object-Detection-Train-Test-Split/blob/master/test_train_split.py/, accessed April 2022.

- [21] tutorialTFODAPI, “generatetfrecord.” <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#evaluation-sec/>, accessed April 2022.
- [22] tensorflow, “pipelineconfig.” https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md/, accessed July 2022.
- [23] philferriere, “cocoapi.” <https://github.com/philferriere/cocoapi/>, accessed April 2022.
- [24] J. Kukačka, V. Golkov, and D. Cremers, “Regularization for deep learning: A taxonomy,” *arXiv preprint arXiv:1710.10686*, 2017.
- [25] R. Padilla, W. L. Passos, T. L. Dias, S. L. Netto, and E. A. Da Silva, “A comparative analysis of object detection metrics with a companion open-source toolkit,” *Electronics*, vol. 10, no. 3, p. 279, 2021.
- [26] cocometrics, “coco detection metrics.” <https://cocodataset.org/#detection-eval/>, accessed March 2022.
- [27] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [28] Python, “Xml api.” <https://docs.python.org/3/library/xml.etree.elementtree.html>, accessed may 2022.
- [29] nathancy, “color picker script.” <https://stackoverflow.com/questions/10948589/choosing-the-correct-upper-and-lower-hsv-boundaries-for-color-detection-withcv/>, accessed April 2022.
- [30] TheCodingBug, “Code functions.” https://www.youtube.com/watch?v=2yQqg_mXuPQ&t=1284s, accessed May 2022.
- [31] S. Zhai, D. Shang, S. Wang, and S. Dong, “Df-ssd: An improved ssd object detection algorithm based on densenet and feature fusion,” *IEEE access*, vol. 8, pp. 24344–24357, 2020.
- [32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [33] H. Yu, R. Jin, and S. Yang, “On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization,” in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.),

vol. 97 of *Proceedings of Machine Learning Research*, pp. 7184–7193, PMLR, 09–15 Jun 2019.

- [34] J. Brownlee, “neptune.” <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/>, accessed august 2022.
- [35] A. Morgunovy, “neptune.” <https://neptune.ai/blog/tensorflow-object-detection-api-best-practices-to-training-evaluation-deployment>, accessed May 2022.
- [36] F. Ozge Unel, B. O. Ozkalayci, and C. Cigla, “The power of tiling for small object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.
- [37] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*, pp. 21–37, Springer, 2016.
- [38] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, “Deep learning for generic object detection: A survey,” *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

Appendix A

Appendix

Listing A.1: Dataset labelling algorithm

```
1 from email.mime import image
2 from fileinput import filename
3 from sre_constants import SUCCESS
4 import cv2, time, os, tensorflow as tf
5 import numpy as np
6 from tensorflow.python.keras.utils.data_utils import get_file
7 import matplotlib.pyplot as plt
8 from cgitb import text
9 import xml.etree.ElementTree as ET
10 os.environ['CUDA_VISIBLE_DEVICES']='-1'
11
12 np.random.seed(123)
13
14 class Detector:
15
16     def __init__(self):
17         pass
18
19     def readClasses(self, classesFilePath):
20         with open(classesFilePath, 'r') as f:
21             self.classList = f.read().splitlines()
22             self.colorList=np.random.uniform(low=0, high=255, size=(len(self.
23                 classList), 3 ))
24
25     def loadModel(self):
26         print("Loading Model " + self.modelName)
27         tf.keras.backend.clear_session()
```



```

26     self.model=tf.saved_model.load(os.path.join(self.cacheDir, "
27         checkpoints", self.modelName, "saved_model"))
28 def createBoundingBox(self, image, threshold=0.5):
29     inputTensor=cv2.cvtColor(image.copy(), cv2.COLOR_BGR2RGB)
30     inputTensor=tf.convert_to_tensor(inputTensor, dtype=tf.uint8)
31     inputTensor=inputTensor[tf.newaxis,...]
32
33     detections=self.model(inputTensor)
34
35     bboxes=detections['detection_boxes'][0].numpy()
36     classIndexes=detections['detection_classes'][0].numpy().astype(np.
37         int32)
38     classScores=detections['detection_scores'][0].numpy()
39     imH, imW, imC = image.shape
40
41     bboxIdx=tf.image.non_max_suppression(bboxes, classScores,
42         max_output_size=50, iou_threshold=0.4, score_threshold=0.5)
43
44     allCropped=[]
45     allBBs=[]
46     print(bboxIdx)
47     if len(bboxIdx)==0:
48         imageConfidence=0
49         confidenceList=[]
50         count=0
51     if len(bboxIdx)!=0:
52         for i in bboxIdx:
53             bbox=tuple(bboxes[i].tolist())
54             classConfidence=None
55             classConfidence = round(100*classScores[i])
56             classIndex=classIndexes[i]
57
58             print(classConfidence, classIndex)
59
60             if classIndex==10 and classConfidence >=50:
61                 confidenceList.append(classConfidence)
62
63                 count=count+classConfidence
64
65                 classLabelText=self.classList[classIndex]
66                 classColor=self.colorList[classIndex]

```

```

65
66         displayText='{}: {}%'.format(classLabelText ,
67                                     classConfidence)
68
69         ymin, xmin, ymax, xmax= bbox
70         ymin, xmin, ymax, xmax= (ymin*imH, xmin*imW, ymax*imH,
71                                 xmax*imW)
72         ymin, xmin, ymax, xmax= int(ymin), int(xmin), int(ymax),
73                                 int(xmax)
74
75         allBBs.append(bbox)
76
77         croppedImage=image[ymin+4:ymax-4, xmin+2:xmax-2]
78         allCropped.append(croppedImage)
79
80         cv2.rectangle(image, (xmin, ymin), (xmax, ymax), color=
81                             classColor, thickness=1)
82
83         lineWidth =min(int((xmax-xmin)*0.2), int((ymax-ymin)*0.2))
84         cv2.line(image, (xmin, ymin), (xmin+lineWidth, ymin),
85                 classColor, thickness=5)
86
87     else:
88         classConfidence=0
89
90     if len(confidenceList)==0:
91         confidenceList.append(0)
92         imageConfidence=count/len(confidenceList)
93
94     return image, imageConfidence, allCropped, allBBs
95
96 def findTrafficLightColorHSV(self, croppedImage):
97     hsv_bbox=cv2.cvtColor(croppedImage, cv2.COLOR_BGR2HSV)
98
99     low_red=np.array([172,95,186])
100    high_red=np.array([179, 255, 255])
101    red_mask=cv2.inRange(hsv_bbox, low_red, high_red)
102
103    listRED=[]
104    listRED=sum(red_mask)
105    listREDsum=sum(listRED)

```

```

102
103     low_yellow=np.array([17,97,255])
104     high_yellow=np.array([35,255,255])
105     yellow_mask=cv2.inRange(hsv_bbox, low_yellow, high_yellow)
106
107     listYELLOW=[]
108     listYELLOW=sum(yellow_mask)
109     listYELLOWsum=sum(listYELLOW)
110
111     low_green=np.array([60, 84, 57])
112     high_green=np.array([90,255,255])
113     green_mask=cv2.inRange(hsv_bbox, low_green, high_green)
114
115     listGREEN=[]
116     listGREEN=sum(green_mask)
117     listGREENsum=sum(listGREEN)
118
119     countRED=0
120     for i in range(0, len(listRED)):
121         if(listRED[i]==0):
122             countRED=countRED+1
123     countYELLOW=0
124     for i in range(0, len(listYELLOW)):
125         if(listYELLOW[i]==0):
126             countYELLOW=countYELLOW+1
127     countGREEN=0
128     for i in range(0, len(listGREEN)):
129         if(listGREEN[i]==0):
130             countGREEN=countGREEN+1
131
132     arraySize=len(listGREEN)
133     countSum=countGREEN+countRED+countYELLOW
134     countSumByThree=countSum/3
135     dif=arraySize-countSumByThree
136
137
138     if(listREDsum!=0 and listREDsum>listGREENsum and listREDsum>
139         listYELLOWsum and dif>=1):
140         traffiColor='Red light '
141     elif(listGREENsum!=0 and listGREENsum>listREDsum and listGREENsum>
142         listYELLOWsum and dif>=1):
143         traffiColor='Green light '

```

```

142     elif(listYELLOWsum!=0 and listYELLOWsum>listREDsum and listYELLOWsum>
143           listGREENsum and dif>=1):
144         traffiColor='Yellow light '
145     elif(dif<1):
146         traffiColor='None '
147     return traffiColor
148
149 def creatingDatasetXMLfiles(self , colorTraffic , BBs, imageSize , frameName ,
150                             newpath):
151     colors=[]
152     coordinates=[]
153     size=[]
154     colors=colorTraffic
155     coordinates=BBs
156     size=imageSize
157     if(len(colors)!=len(coordinates)):
158         print('length of the lists dont match')
159         return
160
161     root=ET.Element("annotations")
162
163     doc=ET.SubElement(root , "folder ")
164     doc.text="TRAFFIC_LIGHT_DATASET_1_OFICIAL"
165     doc2=ET.SubElement(root , "filename ")
166     doc2.text=frameName + ".jpg"
167     doc3=ET.SubElement(root , "path ")
168     doc3.text="/home/carlostiago/Desktop/
169              TRAFFIC_LIGHT_DATASET_1_OFICIAL/"+ frameName + ".jpg"
170     doc3=ET.SubElement(root , "source ")
171     doc31=ET.SubElement(doc3 , "database ")
172     doc31.text="Unknown"
173     doc4=ET.SubElement(root , "size ")
174     doc41=ET.SubElement(doc4 , "width ")
175     doc41.text=str(size [0])
176     doc42=ET.SubElement(doc4 , "height ")
177     doc42.text=str(size [1])
178     doc43=ET.SubElement(doc4 , "depth ")
179     doc43.text=str(size [2])
180     doc5=ET.SubElement(root , "segmented ")
181     doc5.text="0"
182
183     for i in range(0,len(colors)):

```

```

181
182     ymin, xmin, ymax, xmax=coordinates[i]
183     ymin, xmin, ymax, xmax= (ymin*size[0], xmin*size[1], ymax*size
184                               [0], xmax*size[1])
185
186     doc6=ET.SubElement(root, "object")
187     doc61=ET.SubElement(doc6, "name")
188     doc61.text=str(colors[i])
189     doc62=ET.SubElement(doc6, "pose")
190     doc62.text="Unspecified"
191     doc63=ET.SubElement(doc6, "truncated")
192     doc63.text="0"
193     doc64=ET.SubElement(doc6, "difficult")
194     doc64.text="0"
195     doc65=ET.SubElement(doc6, "bndbox")
196     doc651=ET.SubElement(doc65, "xmin")
197     doc651.text=str(xmin)
198     doc652=ET.SubElement(doc65, "ymin")
199     doc652.text=str(ymin)
200     doc653=ET.SubElement(doc65, "xmax")
201     doc653.text=str(xmax)
202     doc654=ET.SubElement(doc65, "ymax")
203     doc654.text=str(ymax)
204
205
206     tree = ET.ElementTree(root)
207     with open (os.path.join(newpath , frameName + ".xml"), "wb") as
208           files :
209         tree.write(files)
210
211 def trafficlightdatasetCreationFromFolder(self, folder, threshold=0.5):
212     newpath = "/home/carlostiago/Desktop/AutomaticDataset2_TESTFINAL2/"
213     if not os.path.exists(newpath):
214         os.makedirs(newpath)
215     newpath2 = "ImagenesTestadas"
216     if not os.path.exists(newpath2):
217         os.makedirs(newpath2)
218
219     images = []
220     files=os.listdir(folder)

```

```

221     for i in range(0, len(files)):
222         x=files[i]
223         x=x[5:-4]
224         files[i]=int(x)
225
226     sorted_files=sorted(files)
227     for i in range(0, len(sorted_files)):
228         x=sorted_files[i]
229         x="Frame"+str(x)+".jpg"
230         sorted_files[i]=x
231
232     print(sorted_files)
233     for j in range(0, len(sorted_files)):
234         print("index: ",j)
235         img = cv2.imread(os.path.join(folder, sorted_files[j]))
236         if img is not None:
237             images.append(img)
238
239         frameName=j+6000
240         frameName=str(frameName)
241         cv2.imwrite(os.path.join(newpath, frameName+'.jpg'), img)
242
243         imH, imW, imC = img.shape
244         imageSize=[]
245         imageSize.append(imH)
246         imageSize.append(imW)
247         imageSize.append(imC)
248         bboxImage, imageConfidence, allCropped, allBBs =self.
                createBoundingBox(img, threshold)
249
250         croppedBoxes=[]
251         croppedBoxes=allCropped
252         BBs=[]
253         BBs=allBBs
254
255         cv2.imwrite(self.modelName + "_ImagemTeste_" + ".jpg", bboxImage)
256
257         R="Red light"
258         Y="Yellow light"
259         G="Green light"
260         colorTraffic=[]
261         BBsfinal=[]

```

```

262         for i in range(0, len(croppedBoxes)):
263
264             trafficLightColor=self.findTrafficLightColorHSV(croppedBoxes[i
265                 ])
266             if(trafficLightColor==R or trafficLightColor==Y or
267                 trafficLightColor==G):
268                 colorTraffic.append(trafficLightColor)
269                 BBsfinal.append(BBs[i])
270
271         if(len(BBsfinal)!=0):
272             self.creatingDatasetXMLfiles(colorTraffic, BBsfinal, imageSize
273                 , frameName, newpath)
274
275     cv2.waitKey(5000)
276     cv2.destroyAllWindows()

```

Listing A.2: Created XML file

```

1 <annotations>
2 <folder>TRAFFIC_LIGHT_DATASET_1_OFICIAL</folder>
3 <filename>TesteXML.jpg</filename>
4 <path>
5 /home/carlostiago/Desktop/TRAFFIC_LIGHT_DATASET_1_OFICIAL/TesteXML.jpg
6 </path>
7 <source>
8 <database>Unknown</database>
9 </source>
10 <size>
11 <width>416</width>
12 <height>536</height>
13 <depth>3</depth>
14 </size>
15 <segmented>0</segmented>
16 <object>
17 <name>Green light</name>
18 <pose>Unspecified</pose>
19 <truncated>0</truncated>
20 <difficult>0</difficult>
21 <bndbox>
22 <xmin>193.998544216156</xmin>
23 <ymin>16.426692962646484</ymin>

```

```
24 <xmax>345.47389364242554</xmax>
25 <ymin>406.6780700683594</ymin>
26 </bndbox>
27 </object>
28 <object>
29 <name>Red light</name>
30 <pose>Unspecified</pose>
31 <truncated>0</truncated>
32 <difficult>0</difficult>
33 <bndbox>
34 <xmin>3.654749631881714</xmin>
35 <ymin>9.26746654510498</ymin>
36 <xmax>148.3363058567047</xmax>
37 <ymin>391.1670265197754</ymin>
38 </bndbox>
39 </object>
40 <object>
41 <name>Yellow light</name>
42 <pose>Unspecified</pose>
43 <truncated>0</truncated>
44 <difficult>0</difficult>
45 <bndbox>
46 <xmin>379.3922219276428</xmin>
47 <ymin>29.44076442718506</ymin>
48 <xmax>529.5613417625427</xmax>
49 <ymin>394.8198547363281</ymin>
50 </bndbox>
51 </object>
52 </annotations>
```