

Universidade do Minho

Escola de Engenharia

Tiago Alexandre Barbosa Pinto

**Object detection with artificial vision and
neural networks for service robots**



Universidade do Minho

Escola de Engenharia

Tiago Alexandre Barbosa Pinto

**Object detection with artificial vision and
neural networks for service robots**

Dissertação de Mestrado em Engenharia Eletrónica
Industrial e Computadores

Trabalho efectuado sob a orientação do

Professor Doutor Agostinho Gil Teixeira Lopes

outubro de 2018

ACKNOWLEDGEMENTS

This dissertation ends my cycle of studies concerning the master's degree and is the culmination of five years of knowledge that was obtained by the academic journey.

This journey was filled with very good colleagues who without them this journey would be very different and teachers who promoted the interest for the subjects.

At first, I would like to thank my adviser, Professor Gil Lopes, for the opportunity of working in a theme that I have always had a lot of interest and a very deep curiosity of working with and for the great provided support and freedom for the development of the dissertation, which it gave me a constant desire to carry it out.

Moreover, I would like to thank my parents who have been relentless in making this possible. Without them I would not be at this turning point. They were the ones who always backed me up in the most difficult times by just being there when I needed the most, always promoting my values and my assets.

I am thankful for those who directly or indirectly had contributed to this dissertation.

ABSTRACT

This dissertation arises from a major project that consists on developing a domestic service robot, named CHARMIE (Collaborative Home Assistant Robot by Minho Industrial Electronics), to cooperate and help on domestic tasks. In general, the project aims to implement artificial intelligence in the whole robot.

The main contribution of this dissertation is the development of the vision system, with artificial intelligence, to classify and detect, in real time, the objects represented on the environment that the robot is placed.

This dissertation is within two broad areas that revolutionized the robotics industry, namely the artificial vision and artificial intelligence. Knowing that most of the existent information is presented on the vision and with the evolution of robotics, there was a need to introduce the capacity to acquire and process this kind of information. So, the artificial vision algorithms allowed them to acquire information of the environment, namely patterns, objects, formats, through vision sensors (cameras). Although implementing artificial vision can be very complex if it is intended to detect objects, due to image complexity.

The introduction of artificial intelligence, more precisely, deep learning, brought the capability of implementing systems that can learn from provided data, without the need of hard coding it, reducing slightly the complexity and the time consumption of implementing complex problems. For artificial vision problems, like this project, there is a deep neural network that is specialized in learning from three dimensional vectors, namely images, named Convolutional Neural Network (CNN). This network uses image data to learn patterns, edges, formats, and many more, that represents a certain object.

This type of network is used to classify and detect the objects presented in the image provided by the camera and is implemented with the *Tensorflow* library. All the image acquisition from the camera is performed by the *OpenCv* library.

At the end of the dissertation, a model that allows real-time detection of objects from camera images is provided.

Keywords

Deep learning, Computer Vision, Convolutional Neural Networks, Object Detection, Service Robot, TensorFlow

CONTENTS

- 1. Introduction 17
 - 1.1 General motivations..... 18
 - 1.2 Objectives 19
 - 1.3 Structure of the Dissertation 19
- 2. Literature Review 21
 - 2.1 Convolutional neural networks 22
 - 2.1.1 Convolutional Layer 23
 - 2.1.2 Pooling layer..... 26
 - 2.1.3 Fully-connected layer 27
 - 2.2 Theoretical foundations 28
 - 2.2.1 Supervised and unsupervised learning 28
 - 2.2.2 Classification vs regression 29
 - 2.2.4 Intersection over Union 30
 - 2.2.5 Average precision 31
 - 2.2.6 Convolutional neural network architectures 34
 - 2.3 State of the art 38
 - 2.3.1 Region based Convolutional Neural networks (R-CNN) 39
 - 2.3.2 You Only Look Once (YOLO) 45
 - 2.3.3 Single Shot Multibox Detector (SSD) 48
 - 2.3.4 Related work..... 51
- 3. Methods and Methodologies..... 53
 - 3.1 Model Framework 53
 - 3.2 Acquisition 55
 - 3.2.1 Segmentation methods..... 57
 - 3.2.2 Final method 67

3.2.3	Problems.....	70
3.3	Training	72
3.3.1	Process of the Dataset.....	72
3.3.2	Train	76
3.3.3	Export	79
3.4	Test	81
4.	Results	84
4.1	Frame rate	84
4.2	Data and parameter selection.....	85
4.3	Final prototype	109
5.	Conclusions.....	114
6.	Future work	116
	References	117
	Appendix A.....	123
	Appendix B.....	124
	Appendix B1	124
	Appendix B2	126
	Appendix B3.....	128

ILLUSTRATION INDEX

Figure 1 – ANN and DNN standard architectures [11]	21
Figure 2 – neural network node [13]	21
Figure 3 - Local receptive field [11]	22
Figure 4 - Applying a filter of 3x3x3 on the input layer	24
Figure 5 - Stride of 1	25
Figure 6 - Zero-padding	25
Figure 7 - Visualization of activations in subsequent layers [18]	26
Figure 8 - Max-pooling, with filter size of 2x2, and stride 1	27
Figure 9 - Average pooling, with filter size of 2x2, and stride 1	27
Figure 10 - Intersection and Union [30]	31
Figure 11 - Precision and recall graph	32
Figure 12 - precision interpolation	33
Figure 13 - Lenet 5 architecture [8]	34
Figure 14 - AlexNet architecture [20]	35
Figure 15 - VGG configurations [43]	36
Figure 16 - Inception Module [42]	37
Figure 17 - Residual Block [9]	38
Figure 18 - R-CNN architecture [38]	39
Figure 19 - SPP layer representation [49]	40
Figure 20 - Fast R-CNN network [39]	41
Figure 21 - Faster R-CNN [40]	42
Figure 22 - RPN [40]	43
Figure 23 - Mask R-CNN tail [50]	44
Figure 24 - Mask R-CNN architecture [50]	45
Figure 25 - YOLO model [24]	46
Figure 26 - YOLO CNN architecture [51]	47
Figure 27 - SSD architecture [53]	49
Figure 28 - Multiple scale feature maps and default boundary boxes [53]	49
Figure 29 - Model framework	54

Figure 30 - Acquisition of the dataset where (a) is the raw image with the bounding box and (b) is the corresponding ground truth information, namely, the box coordinates (xmin, ymin, xmax, ymax) and the respective class. Note that the coordinates presented in (b) is just an example, as it is not the corresponding box presented in (a)	54
Figure 31 - Training	55
Figure 32 - Testing the trained model.....	55
Figure 33 - Acquisition framework.....	56
Figure 34 - Running the acquisition script.....	57
Figure 35 - Classes definition	57
Figure 36 - Acquisition model with color segmentation.....	58
Figure 37 - Conversion of BGR to HSV. Where (a) is the source frame and (b) is the HSV frame	58
Figure 38 - BGR and HSV values of the clicked position	59
Figure 39 - Color Segmentation.....	60
Figure 40 - Result of the acquisition model using color segmentation.....	60
Figure 41 - Background (a) and foreground (b) frames	61
Figure 42 - Acquisition model with background subtractor	62
Figure 43 - Acquisition model with MOG background subtractor.....	63
Figure 44 - Segmentation with MOG background subtractor, where (a) is the background frame and the respective segmentation and (b) represents the frame with the object, foreground, and the correspondent segmentation. (a) and (b) are successive frames.	63
Figure 45 - MOG background subtractor segmentation, where (a) is the background frame and (b) is a consecutive frame and the respective segmentation.	64
Figure 46 - Acquisition with the arm.....	65
Figure 47 - Segmented frame	65
Figure 48 - Application of the bitwise NOT operation on the MOG mask (a) and on the color segmentation mask (b), resulting on the final mask (c)	66
Figure 49 - Final Acquisition framework.....	67
Figure 50 - Initial windows of the acquisition algorithm, where (a) is the main frame and (b) is the MOG mask.....	67
Figure 51 - Color segmentation and MOG segmentation with the arm and the desired object. (a) is the main frame, (b) corresponds to the MOG mask and (c) is the color segmentation mask	68
Figure 52 - Main frame with the bounding box.....	69

Figure 53 - Saved image frames and the respective files with the ground truth information.....	70
Figure 54 - Bad arm segmentation.....	71
Figure 55 - Bad MOG segmentation	71
Figure 56 - Training AP precision curve on Tensorboard	79
Figure 57 - Validation AP precision curve on Tensorboard.....	79
Figure 58 - Train the network	80
Figure 59 - Test frame with no NMS.....	81
Figure 60 - Test frame using NMS to suppress overlaps that are over 0.75	82
Figure 61 - Test frame using NMS to suppress overlaps that are over 0.5	82
Figure 62 - Test frame using NMS to suppress overlaps that are over 0.1	83
Figure 63 - Frame rate.....	84
Figure 64 – Types of Bottles in the dataset	85
Figure 65 - Data1 training precision curves	86
Figure 66 - Data1 validation precision curves	87
Figure 67 - Data2 training precision curves	88
Figure 68 - Data2 validation precision curves	89
Figure 69 - Data2 training precision curves, with 150x150 as the input size.....	90
Figure 70 - Data2 validation precision curves, with 150x150 as the input size.....	91
Figure 71 - Data2 validation precision curves, with different learning rates, with 20 epochs and 150x150 as the input size	92
Figure 72 - Data3 training precision curves	94
Figure 73 - Data3 validation precision curves	95
Figure 74 – Detection when the bottle is being grabbed	96
Figure 75- - Detection of various bottles in other perspective.....	96
Figure 76 – Bad bottle detection	97
Figure 77 – Types of cans in the dataset.....	98
Figure 78 - Training AP curves of the bottle class	99
Figure 79 - Training AP curves of the can class	99
Figure 80 - Validation AP curves of the bottle class	100
Figure 81 - Validation AP curves of the can class	100
Figure 82 – Training AP of Data 4 and Data5 curves of the bottle class	102
Figure 83 – Training AP of Data4 and Data5 curves of the can class	102

Figure 84 – Validation AP of Data4 and Data5 curves of the bottle class	103
Figure 85 – Validation AP of Data4 and Data5 curves of the can class	103
Figure 86 – Detections of a single object in a frame	104
Figure 87 – Detections of both classes.....	105
Figure 88 - Types of bag of chips in the dataset.....	105
Figure 89 – Training curves of each class	106
Figure 90 - Validation curves of each class	107
Figure 91 - Training curves of each class using 200x200 as the input size	108
Figure 92 - Validation curves of each class using 200x200 as the input size	109
Figure 93 - Bottle detections	110
Figure 94 - Can detections.....	110
Figure 95 - Bag of chips detections	111
Figure 96 – Network detections in more than one class.....	111
Figure 97 –Bad recognition.....	112
Figure 98 – Nonexistent detection.....	113
Figure 99 – Activation functions [79].....	123

TABLE INDEX

Table 1 - Precision and recall values	32
Table 2 - Comparison of results between Fast R-CNN and R-CNN (Data source [39])	41
Table 3 - Timing (ms), for each image until it reaches the end of the network.(Data source: pag.8 [40]).	43
Table 4 - Difference on performance between two methods, both using Fast R-CNN object detector with VGG16 (Data Source: [40])	44
Table 5 - Performance and speed of the object detectors, using the same dataset (Data source: [51])	48
Table 6 - Results of the current leading object detection network using the same dataset (Data source: [53])	48
Table 7 - Hue values and the respective color.....	59
Table 8 - Size of the SSD feature maps and the total number of anchors according to the preset size of the input layer	74
Table 9 – Input size of the feature maps	75
Table 10 - Results on Data1, with different number of epochs	86
Table 11 - Results on Data2, with different number of epochs	88
Table 12 - Results on Data2, with different number of epochs, with 150x150 as the input size.....	90
Table 13 - Results on Data2, with different learning rates, with 20 epochs and 150x150 as the input size	92
Table 14 – Results on Data3, with different number of epochs	94
Table 15 - Results on Data4, with different number of epochs	98
Table 16 – Final AP values of the network with 50 epochs, using the Data5	101
Table 17 – Training AP values of the network using Data6.....	106
Table 18 – Validation AP values of the network using Data6	106
Table 19 - Training AP values of the network using Data6 and input of 200x200.....	107
Table 20 – Validation AP values of the network using Data6 and inputs of 200x200	108

NOMENCLATURE

2D – Two Dimensions

3D – Three Dimensions

API – Application Programming Interface

ANN – Artificial Neural Network

AP – Average Precision

BGR – Blue Green Red

CNN – Convolutional Neural Network

CPU – Central Processing Unit

DNN – Deep Neural Network

FN – False Negatives

FP – False Positives

FPS – Frame Per Second

FC – Fully Connected

GPU - Graphical Processing Unit

HOG – Histogram of Oriented Gradients

HSV – Hue Saturation Value

ILSVRC – ImageNet Large Scale Video Recognition Competition

IoU – Intersection over Union

mAP – mean Average Precision

MOG – Mixture of Gaussian

NMS – Non-Maximum Suppression

RGB – Red Green Blue

RCNN – Region Convolutional Neural Network

RoI – Region of Interest

RPN – Region Proposal Network

SS – Selective Search

SSD – Single Shot Detector

SVM – Support Vector Machine

TN – True Negative

TP – True Positive

YOLO – You Only Look Once

1. INTRODUCTION

The visual information, even as Human beings, has a high relevance in the accomplishment of tasks as well as influences our behaviour in certain decision making. It is so important and complex that the Human brain has a part solely for visual processing, the visual cortex. It is where there is a vast amount of information that is obtainable and is why we strongly rely on our visual sense.

Therefore, artificial vision plays an important role in robotics since it has caused a great change in the paradigm of robotics, one of which is the capability of the robot to visually perceive the environment and interact with it to perform tasks such as, obstacle avoidance, object recognition and manipulation and Human interaction [1].

This dissertation is within the development of a service robot for domestic tasks with the goal of implementing a vision system that can detect objects presented in the environment to allow the robot their manipulation, and thus it addresses a computer vision problem on a robotic vision system.

According to [1], a computer vision problem aims for the understanding of a scenario, for example, detect objects, however, they are used for specific applications and for individual problems. On the other hand, a robotic vision problem treats the vision as one of the several sensory systems that the robot possesses which together are used to fulfil tasks.

Computer vision problems such as object classification, object detection and recognition and semantic segmentation requires a very deep understanding of image processing algorithms and thus they were very complex to implement due to image diversity and complexity as well as some factors that needed to be considered that increases the complexity of the vision problem, such as:

- The variations in viewpoint, that is, the same object can have different positions and angles in an image;
- The difference in illumination, where different images can have different light conditions;
- The object could be partially hidden;
- The background clutter, that is, some objects might blend into the background.

Even with the introduction of machine learning algorithms, which brought rise to the development of systems that can learn from provided data, like the Human, according to Yann Lecun [2], the conventional machine learning techniques do not have the ability to process natural data in their raw form.

So, computer vision problems, before deep learning, were usually manually performed by applying a hand-crafted feature extractor to transform the raw data into a suitable relationship that,

afterwards, could be used by a learning subsystem (classifier) that could detect or classify the patterns in the input, for example “HOG” (Histogram of Oriented Gradients) features [3] and feature matching [4]–[6].

The deep learning has facilitated the resolution of these problems that previously required a high level of engineering. Indeed, deep learning methods can learn the representation of the raw input data and transform it successively to a more abstract representation to afterwards be detected or classified, and thus, according to [2], the key aspect of deep learning is that the features are learned from the data and not designed by Humans, and so they require “very little engineering by hand”.

1.1 General motivations

The deep learning systems, according to [7], has promoted a deep change in the pattern recognition, machine learning and computer vision communities since that most or all previous techniques are now based on deep learning methods. In fact, it has provided the possibility of implementing systems without “hard coding” them.

Yet, according to Yann Lecun [8], the conventional deep learning systems are unable to extract all the variants of computer vision problems, and so he devised a deep network for computer vision tasks named CNN (Convolutional Neural Network).

The CNNs have been in an uprising until nowadays, since that most computer and robotic vision problems now adopt this type of network as the main processing method, proving the added value of its use. Indeed, these networks have outperformed the previous deep learning algorithms in computer and robotic vision tasks, being that, with their constant evolution, they have also surpassed the Human in the accomplishment of classification tasks, like the ResNet architecture [9].

However, since these networks work with images, they have, in most architectures, millions of parameters, and therefore the usage of the CNNs require a high computation power, such as GPUs (Graphical Processing Unit), so most of the currently CNN based methods and architectures do not consider the usage of lower computation power such as CPUs (Computer Processing Units), which are way less expensive than the GPUs.

In addition, these networks, to precisely learn the properties of the data, need to be trained with a high quantity of images. Although there are online, datasets of images that are already prepared to train some networks to classify and detect a set of objects, these datasets are specific for computer vision problems, since they are composed of static images.

Moreover, these datasets not only remove the freedom of choosing the inputs and the outputs of the network, but also may have more outputs than necessary, or may not have the necessary outputs for the solution.

Thus, this dissertation will try to prove the viability of the usage of CNN even when using low end resources, such as CPUs, reinforcing the usage of CNNs in every machine vision problem. It will also prove that is possible to achieve good precision rates, even when using own made Datasets.

1.2 Objectives

The main goal of this dissertation is the development of an intelligent robotic vision system that can detect a set of objects presented in the environment in real-time and, therefore, it will be implemented a CNN-based vision system for object classification and detection using an own made dataset.

To achieve such goal the problem was divided by the following tasks:

- Study of the several CNN architectures developed as well as the state of the art CNN object detectors, to find the most suitable network to use in this solution;
- The development of the model framework, of which it will be divided in three main parts;
 - i. The acquisition part, where the dataset will be created;
 - ii. The part where the network will be trained with the previously created dataset;
 - iii. The final part, named test, where the network, after being trained, is tested for the final application.
- The selection of the best dataset as well as the best parameters that represents the desired results, that is, the detection with a good precision rate in real time;

At last, after choosing the suitable dataset as well as the training parameters, it is intended to have a model that can detect the desired objects regardless of its position and perspective.

1.3 Structure of the Dissertation

According to the main objectives presented above, this dissertation is divided in four main parts. The first part or chapter 2, named literature review, is composed of the theoretical part of this dissertation in which, first, an introduction to the CNN is exhibited, where it is briefly presented why the conventional DNNs (Deep Neural Networks) were not appropriate to extract the feature on images. Afterwards, in the subsection 2.1 it is presented the main characteristics of a CNN which brought rise to this kind of

network. This will allow a better understanding behind the CNN and what are the differences in comparison the other neural networks.

Then, still on chapter 2, is shown the theoretical foundations, in subsection 2.2, where the concepts for this dissertation are demonstrated along with the chronological evolution of the CNN architectures and the respective performance. The subsection 2.3 is composed of the main state of the art object detectors networks developed as well as some applications on robotic vision problems in which it helps to verify which network is the best to use.

In chapter 3 the methods and methodologies used for the development of the prototype are demonstrated, since the acquisition of the dataset until the network testing. Firstly, it is revealed what are the materials (software and hardware) used for the development of the prototype. Afterwards, in subsection 3.1, the model framework is displayed to present the entire model, that is, the parts are briefly demonstrated and its supposed output. In the subsequent subsections (3.2, 3.3, 3.4), the parts are explained in detail along with the methods used to implemented them.

After the implementation of the model framework, the dataset and the parameters need to be tuned to choose the best combination that results on the best performance. Hence, the result chapter corresponding to chapter 4, is composed of the data and parameters selection of the model prototype displaying the results of each model with the different dataset and/or parameters along with the discussion of the results obtained.

In chapter 5 it is presented the conclusions of the realization of the prototype and of the obtained results, followed by the future work in chapter 6.

2. LITERATURE REVIEW

The DNNs are an upgrade of the conventional neural networks, called ANN (Artificial Neural Network) that was first emerged as a learning algorithm in 1948 by Donald Hebb [10]. The main difference between the ANNs and the DNNs is the number of hidden layers in the network hence the name deep (figure 1).

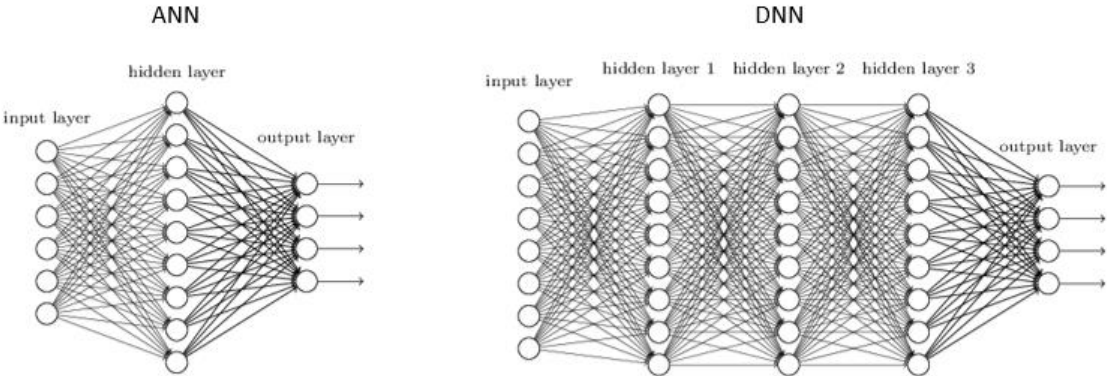


Figure 1 – ANN and DNN standard architectures [11]

Each circle displayed in figure 1 is a neuron (the nuclear element of a neural network), also called node (figure 2). Each of them establishes a weighted connection with all the subsequent nodes [12]. The output value of any node is performed by an activation function or non-linearity, such as: The Sigmoid function; the Tanh activation; the ReLU (See also appendix A pag. 123).

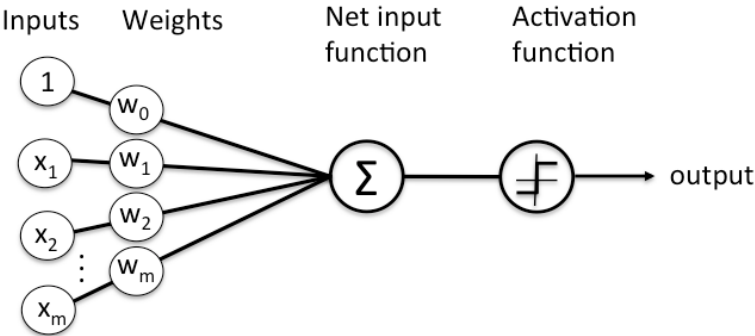


Figure 2 – neural network node [13]

Yann lecun [8] refers that the traditional multi-layer network, as the DNN presented in figure 1 where all nodes in one layer are connected to all nodes on the next layer, are able to recognize raw images (usually size-normalized and centered) but they have some flaws.

For computers, an image is a multiple-dimension array of numbers and each number represents a pixel value, if it is intended to train an DNN to recognize something on images, then each input node should be a pixel. So, for example, if the dataset has a set of 30x30 images, then there should be 900 input nodes. If it is a colored image, i.e. an RGB (Red Green Blue) image the number of input nodes are increased by x3.

If the input is a high-resolution image the number of input nodes grows drastically. Hence, the number of totals parameters in a DNN grows sharply, increasing the computational resources needed to implement and train the network.

Still, the number of parameters may not stand a problem on some modern computers. According to Yann Lecun [8] the topology represented in figure 1 have no built-in invariance that uncovers the variations on the input images, namely, translations, scales and geometric distortions. Moreover, it completely ignores the strong 2D (Two Dimension) local correlation that images have, i.e. the pixels that are nearby are highly correlated.

2.1 Convolutional neural networks

The CNNs were introduced to uncover the drawbacks of the conventional DNNs being more robust and automatic in extracting features on images. The main characteristic of the CNN is that instead of connecting each neuron to all the previous ones, as in figure 1, it connects to a set of neurons located in a specific location in the previous layer, known as local receptive field [8] as shown in figure 3.

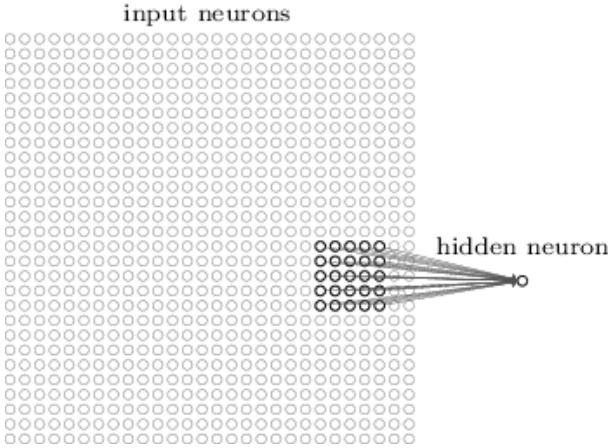


Figure 3 - Local receptive field [11]

The CNN was mainly inspired by the research of Hubel and Wiesel in, 1959 and 1968 [14], [15], who studied how the mammalian visual cortex processes the visual information, they first researched on a cat [15], and then they performed the tests on a monkey [14]. They checked the presence of some

neurons, on the test subject, that responds to different situations: the “simple cells” that are sensitive to orientations and the “complex cells” to movements, that is, to differences in positions. They concluded that the neurons in the visual system have local connectivity, hence the presence of local receptive fields, also the neurons on a layer perform the same kind of processing and the size of the receptive fields increases with depth [14], [15].

In conclusion, the CNN covers three architectural ideas which are, the local connections, the shared weights and spatial sub-sampling or pooling [8].

The local connections allow the extraction of features in an area of a feature map, i.e. they take advantage of the local correlation that the nearby pixels have, since they can form local features that are easily detected. On the other hand, Yann Lecun et al [2] states that the features of an image are invariant to location, that is, the feature can appear in any part of the image, hence, the idea of sharing weights in the same feature map allows the detection of the same pattern in any part of the image.

Therefore, according to [2], the neurons or units in a convolutional layer are disposed in feature maps where each unit is the result of the local weighted connection of a set of units presented in the previous feature map. A set of units is the result of convolving a filter in a certain position of the previous map. The result of the convolution is passed through a non-linearity layer, usually the ReLu (see appendix A pag 123). All the units in the same feature map are the result of the application of the same filter in the previous map, hence they share the same set of weights. Nevertheless, different feature maps use different set of weights.

While the convolutional layer is responsible to detect the local features from the previous layers the pooling layer merges semantically similar features into one, by increasing the size of the receptive field. The convolutional layer and the pooling layer form the first stage of the CNN known as the feature extractor, while the last stage of the CNN is formed by the fully-connected layer [7].

2.1.1 Convolutional Layer

As the name says, the convolutional layer is a layer that performs convolutions on the input. The convolution is fulfilled by applying a filter or kernel that slides across the entire layer (figure 4).

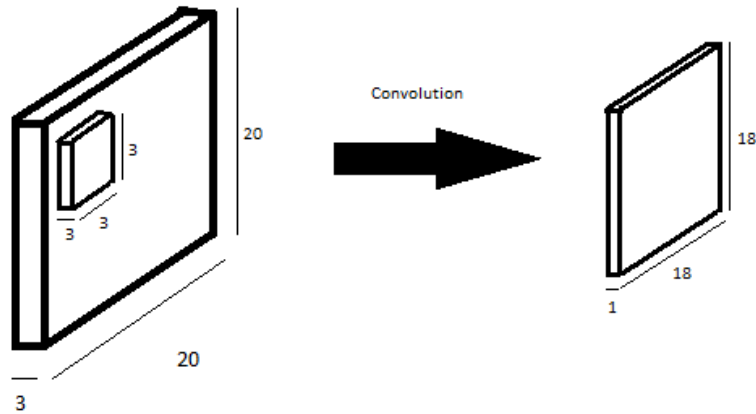


Figure 4 - Applying a filter of 3x3x3 on the input layer

Each filter is composed by a set of weights, that are multiplied by the correspondent pixel values, during the convolving process, performing the product between the weights and the inputs. These filters learn, through training, to search for a specific feature in the input by “activating” when the feature is presented in the input layer [16].

In general, the convolutions at each location of the input layer can be represented as the weighted sum between the weights and the input values [17] as demonstrated in equation 1.

$$W * X + b \quad (1)$$

W represents the matrix of filter values, **X** the matrix of pixel values and **b** the bias factor.

The filter crosses all the layer, starting in the top left corner and ending at at the lower right corner, producing an output at every location. After applying the filter at each possible location, it results on an activation or a feature map, in this case a 18x18 map, as presented in figure 4.

The size of the output depends on a parameter that defines the number of steps that the filter moves, named stride (S), for example, the case presented in figure 5 the stride is 1, i.e. the filter moves 1 cell at each step. The representation in figure 5 demonstrates how the filter moves with the stride of 1 in a 10x10 map.

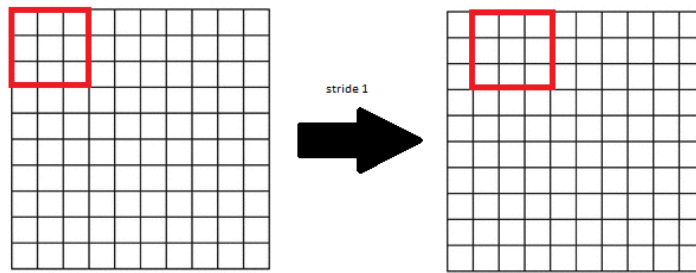


Figure 5 - Stride of 1

In figure 5, it is possible to see that the filter does 8 steps until it reaches the right end, and another 8 steps until the lower end, so the output map of that convolution is an 8x8 map.

The output size is defined by equation 2 [17],

$$\frac{W - F + 2P}{S} + 1 \quad (2)$$

where **W** represents the dimension of the input, **F** the filter size, **S** the stride and **P** is the amount of zero padding. The zero-padding consists on placing around the original map, a set of zero value pixels, as presented in figure 6. So, for $W=10$, $F=3$, $S=1$ and $P=0$, the output size will be 8.

The spatial size reduction that happens by performing convolution is not intended, because it will result in a sharp descent of the input volume, preventing deeper networks. Therefore, to prevent that size descent, it is usually used the zero-padding method as demonstrated in figure 6.

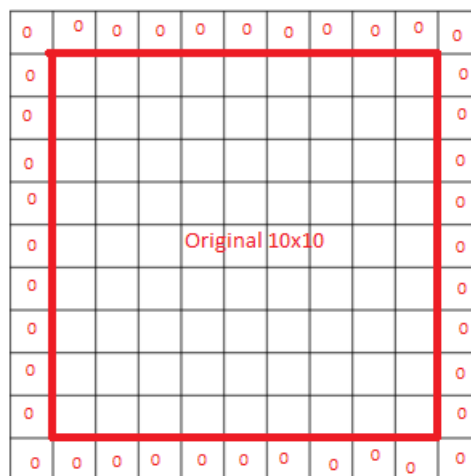


Figure 6 - Zero-padding

Therefore, by calculating the output size for $P = 1$, the output size will be 10, the same as the original map.

A typical convolutional layer is constituted by a stack of many activation maps, as the application of different filters results on various activation maps, that allows the extraction of multiple features at each location [8].

The depth of a convolutional layer is equal to the number of filters applied, for example if 5 filters were applied in the representation of figure 4, it would result in 5 activation maps, so the total volume would be $18 \times 18 \times 5$.

In figure 7 is represented the features that each filter (boxes), is searching for a specific trained model, noting that the level of the feature increases in the subsequent layer.

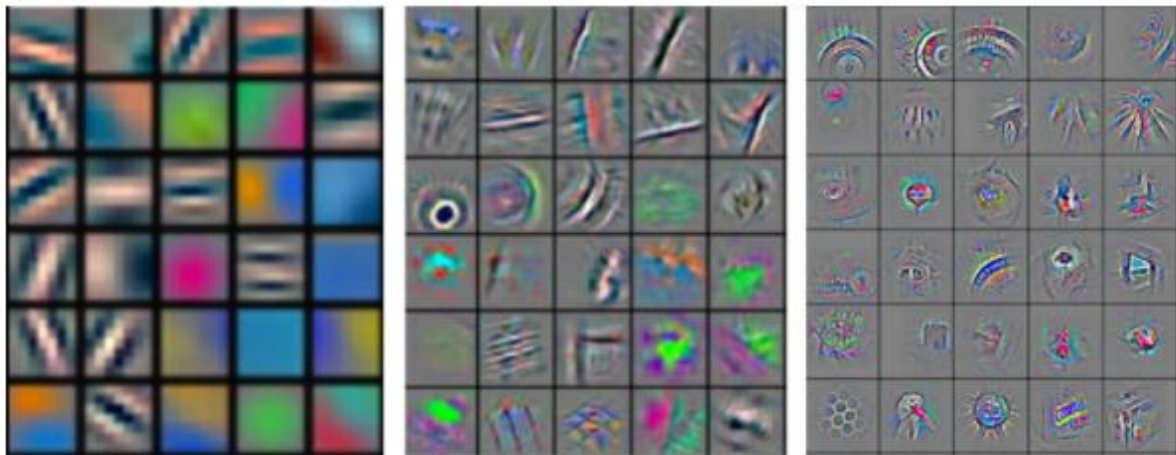


Figure 7 - Visualization of activations in subsequent layers [18]

2.1.2 Pooling layer

By using the zero-padding method on the convolutional layer, the input size will remain the same as the input passes through the network, yet a size reduction is needed to carry out, as referred. The layer that does the reduction is called pooling or sub-sampling layer.

The size reduction is greatly needed for the feature extraction, because instead of just learning the exact relative positions between features, it allows the features to shift relatively to each other, making the network more robust to variations of the features (distortions, translations) [8]. Reducing the size allows to diminish the number of parameters, hence it gradually decreases the computational cost and complexity of the network.

In similarity to the convolutional layer, the pooling layer also uses filters, but instead of performing the weighted sum, it performs arithmetic functions on all the previous activation maps.

The most common pooling methods are the max-pooling and the average pooling. The max pooling searches for the max value on the current filter position (figure 8) and the average pooling calculates the mean value (figure 9).

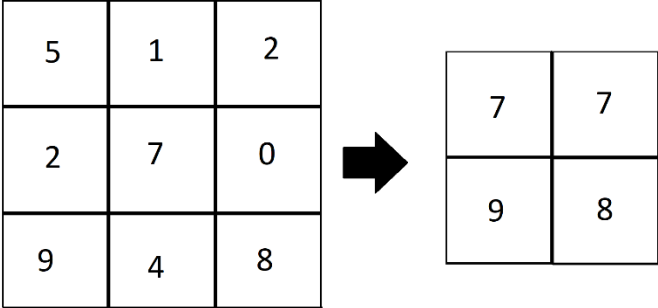


Figure 8 - Max-pooling, with filter size of 2x2, and stride 1

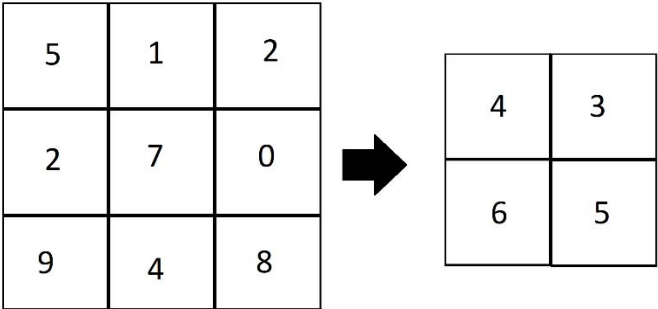


Figure 9 - Average pooling, with filter size of 2x2, and stride 1

2.1.3 Fully-connected layer

The fully-connected layer is performed once the spatial features are extracted, i.e. after the input image goes through a set of convolutional layers and pooling layers, hence it picks the high-level features and determines which features most correlate to a specific class [19]. It is the similarity that the CNNs have in relation to DNNs, where each pixel is considered as a separate neuron and each neuron is connected to all the previous ones [16].

The fully-connected layer extends all the output activation maps into a one-dimension vector, that is, the size of the first fully-connected layer depends on the size of the last map and the corresponding depth. The input vector is connected to a set of hidden layers following the output layer, that contains the output classes for the classification. An example of a fully-connected layer can be represented in figure 1.

A CNN is a combination of these layers, arranged in a specific way, and the number and the sequence of these layers depend on the way it was dimensioned [2].

Taking the example of *AlexNet* [20] presented in figure 11, the network is represented of 11 layers, excluding the input layer, and the corresponding sequence is:

- Conv->MaxPool->Conv->MaxPool->Conv->Conv->Conv->MaxPool->FC1->FC2->FC3(output layer).

The design of a typical CNN consists on defining the sequence and the structure of the layers. When the convolutional layers are defined it is necessary to select the number of filters that each layer will possess and the respective size as well as the intended non-linearity (see Appendix A pag 121). In addition, for the pooling layer, is necessary to choose the pooling operations, as well as the number of filters and their size.

2.2 Theoretical foundations

To implement a CNN model there are some factors that are needed to be considered as for example, if the network will train in respect to the ground truth, the nature of the output, the evaluation metrics to verify the performance of the network, the training processes, namely, the algorithms to train the network and the designing of the main architecture of the network. These concepts not only define the type of the network that will be used but also define the nature of learning process.

2.2.1 Supervised and unsupervised learning

Any machine learning application is divided into two groups based on the way they use the data to “learn”, the supervised and unsupervised learning. These methods define whether the data to train the network needs to be labeled or not.

The most common method is the supervised learning, where the network trains in respect to the ground truth, i.e. the network in the learning process expects not only the input image, but also the target label. Then, a supervised learning model needs the user to act as a guide to teach the model with the correct result and therefore it requires that the data used to train the network are labeled with the correct answers taking into account the outputs are previously known [21]. The supervised learning is usually used to train on classification and/or regression problems [22].

In opposition to the supervised learning there is the other method called unsupervised learning. This method does not need the output labels and the ground truth ones, because it infers the natural structure in a set of data [22] being able to separate them into groups according to the natural properties of the data. Although this method works similarly to the human brain, it is not commonly used because of the complexity of the algorithm.

2.2.2 Classification vs regression

Inside the supervised learning method there are two main algorithms which defines the nature of the output, known as the classification and the regression.

The classification problem is used when the desired output is a discrete label, more precisely, when the problem can be defined by a finite number of classes [23], for example, the MNIST dataset [24] is used to train a network to classify digits from 0 to 9 and therefore the output is composed of those 10 classes. Inside the classification problem there are two main types of classification, the binary classification and the multi-label classification [23].

The example presented above is a representation of a multi-label classification, where the model tries to detect what number is presented on the image from 0 to 9. The binary classification occurs when it is desired for the model to divide the predictions into two groups, for example, it is implemented a network to classify numbers, so the output should have only two labels, the numbers and the non-numbers (1 or 0), that is, it predicts if the supplied image contains a number, regardless of the number displayed.

On the other hand, the regression problem is used when the output is a continuous value, more precisely, when the output represents a quantity and not a set possible labels [23], for example, when it is intended to predict the price of a certain object.

2.2.3 Supervised Training

The training is the most important part of every neural network. It is where the network learns from data what are the corresponding features of a certain class, allowing them to properly predict the correct class in the input data. This learning process is performed by adjusting the network weights so that these values are the ones that best represent the model. In a CNN architecture the training will update the filter values to search for a certain feature in a class.

The supervised training process in a CNN is carried out the same way as DNNs, i.e. it is divided in 4 main tasks: the forward pass, the loss function, the backpropagation and the weight update [19].

The main goal of the training is to reduce the difference between the output predictions and the real output on a forward pass, by “discovering” the best suitable relationship between the input and the output [25]. The output prediction is the prediction of the network by the forward pass in the output layer. The real output is the information that is provided alongside the image, that is, the desired/target result.

The forward pass picks the image and passes it to the network and, at the end, a cost value is computed that indicates how bad the network is predicting the class, by calculating the loss function [26] which picks the actual predictions and the real output and calculates the error.

Afterwards, the backpropagation or the backward pass determines which values contributed most to the loss value computing the gradient of the loss function by propagating the error value through the network, from the output to the input, checking the contribution of each node to the output loss and therefore it checks how the change in the weights affects the loss, represented by the derivative of the loss function [25].

Once the derivative or the gradient of the loss is computed the weights are updated in a way that it minimizes the loss, that is, in an opposite direction of the gradient. The weight update is usually performed by an optimizer algorithm [27].

The set of the presented tasks (the forward pass, the loss function, the backpropagation and the weight update) represent an iteration or epoch. So, the training is performed for a fixed number of epochs, usually defined by the user. When all the epochs are executed, the model is trained [19].

For further understanding the neural network training procedure please see [28] and [17].

2.2.4 Intersection over Union

The IoU (Intersection over Union) is an evaluation metric, that is commonly used to measure how accurate a model is on predicting the position of an object, represented by its bounding box, in relation to the ground truth box.

To perform the IoU method, it is only needed the ground truth box, provided by the dataset, and the predicted box. The IoU (equation 3)[29], is a ratio that represents how much the predicted box is enclosed by the ground truth.

$$IOU = \frac{\textit{Area of overlap}}{\textit{Area of Union}} \quad (3)$$

The area of overlap or area of intersection, is the area that both boxes are overlapped, and the area of union is the whole area of both boxes (figure 10).

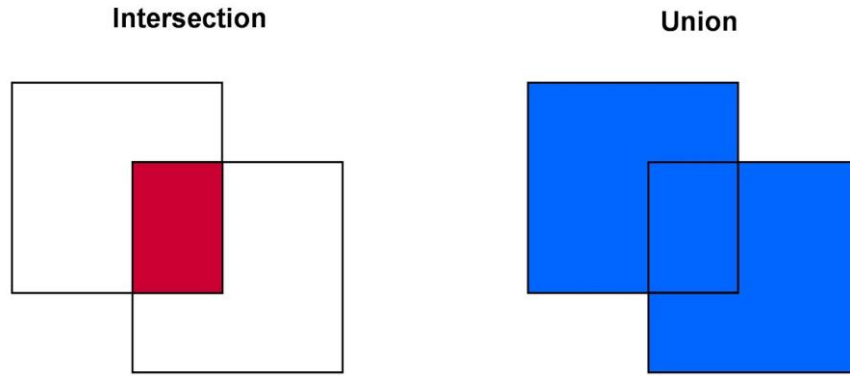


Figure 10 - Intersection and Union [30]

2.2.5 Average precision

The average precision (AP) value is the most common evaluation metric used to verify the performance of a neural network model by providing a precision value that “tells” how good a network is on performing a given task: if a network has a high average precision rate it can be implied that the network performs well the tasks otherwise the network behaves poorly.

The average precision is commonly calculated, per class, in every training epoch by using the 11-point interpolated average precision method [31]. This method consists in computing the precision and recall values to calculate the average of the maximum precisions at 11 recall levels.

The precision (equation 4) is the percentage of the predictions that are correct, that is, how accurate are the predictions.

On the other hand, the recall measures the amount of the truly relevant returned information and is represented by equation 5 [32].

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

The method picks all the predictions provided by the network and sorts them according to their probability, from the highest to the lowest value, and verifies whether the predictions are positive or negative, by checking the IoU between the predictions and the ground truth, to after compute the

precision and recall values: if the IoU is less than 0.5 it means that the predictions are false, otherwise they are true.

If is considered that the entire dataset contains 3 images of a certain class, all the predictions of the network for that class are sorted as presented in table 1.

Table 1 - Precision and recall values

Rank	Positive or Negative	Precision	Recall
1	True	1,0	0,3
2	Negative	0,5	0,3
3	Negative	0,3	0,3
4	True	0,5	0,7
5	True	0,6	1,0

For example, in the second rank the precision is $\frac{1}{2} = 0,5$ and the recall is $\frac{1}{3} = 0,3$.

According to the predictions shown in table 1, the precision and recall are related by the curve displayed in figure 11.

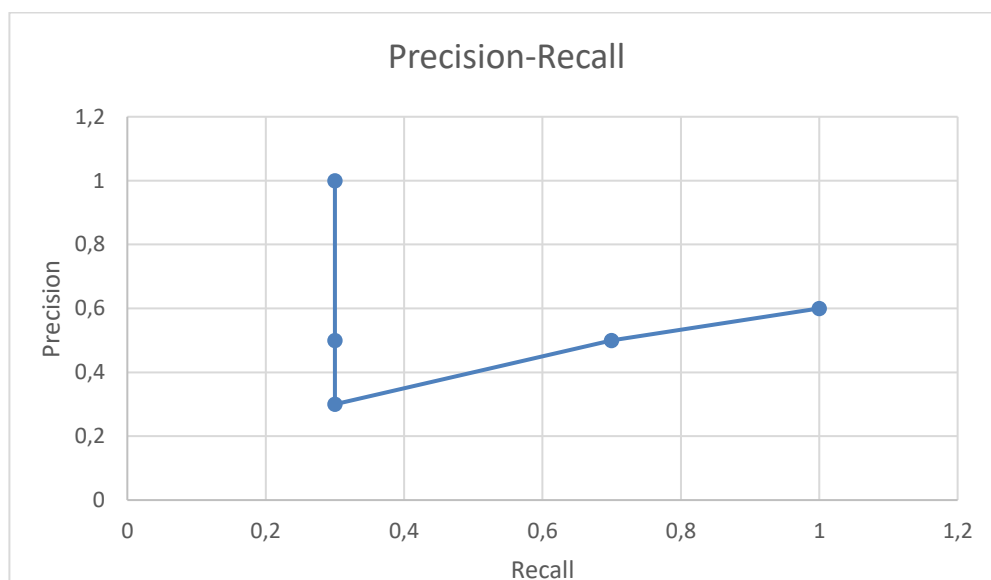


Figure 11 - Precision and recall graph

Therefore, for each recall value, r , from 0 to 1 in increments of 0.1, the precision is interpolated by replacing it with the highest precision in the next recalls $r' \geq r$ [31] (equation 6) , resulting on the red curve represented in figure 12.

$$p_{inter}(r) = \max_{r' \geq r}(r') \quad (6)$$

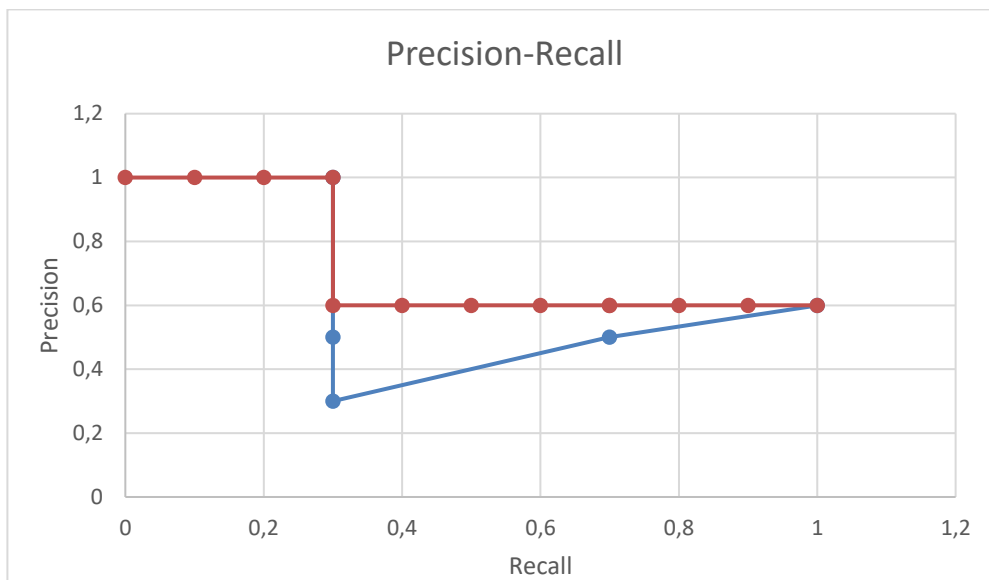


Figure 12 - precision interpolation

The average precision is then calculated by averaging the maximum precisions of the 11 recall values (0, 0.1, 0.2, ... ,0.9,1), given by the equation 7 [32]. It also can be represented as the area under the red curve (figure 12).

$$AP = \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} p_{inter}(r) \quad (7)$$

So, in this example the AP is $\frac{(4*1,0+7*0,6)}{11} = 0,75$.

The mAP (mean Average Precision) is just the mean over all the average precisions obtained in every class.

2.2.6 Convolutional neural network architectures

The convolutional neural networks are quiet a new feature that was introduced and propelled the field of deep learning. Even though the first convolutional neural network architecture was developed by Yann Lecun et al [8], in 1998, named *LeNet-5*, and can be presented in figure 13.

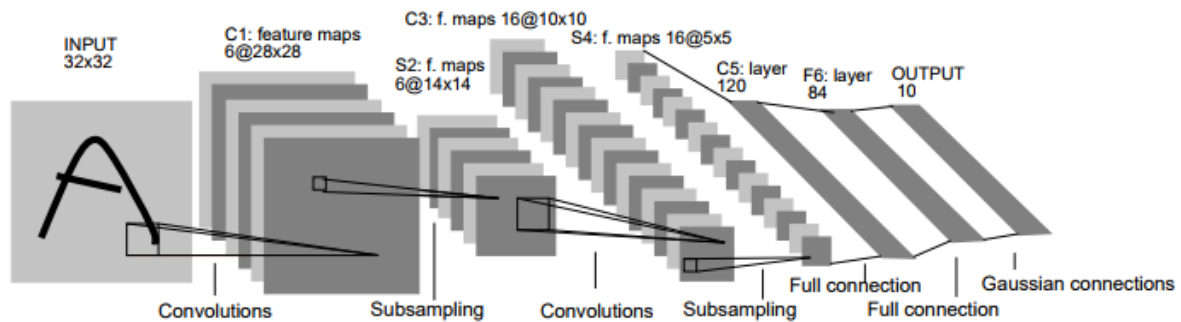


Figure 13 - *LeNet 5* architecture [8]

The *LeNet-5* architecture was fundamental to the field of deep learning for image recognition, since it was the first to state that image features are distributed across the image and the application of convolutions, more precisely, filters across the image, with learnable parameters was the best way to extract those image features [8].

As shown in figure 13, the network consists in several layers, such as:

- Input layer, is the beginning of the network, where the input raw image is provided;
- Convolutional layer, is the result of the convolution applied at the previous matrix;
- Subsampling, reduces the spatial size, the result of subsampling consists on the Pooling layer;
- Fully-connected-layer, puts the final set of matrices into a one-dimensional vector and allows each neuron (pixel) to receive data from all the neurons in the previous layers;
- Output layer, is where the classification is presented.

The sequence of this layers is a convolutional neural network, and these layers are the root of all the recent architectures of convolutional networks.

From 1998 to 2012 there was no further investigation in this area, due to computation incapacity to process these kinds of networks, at that time, because of the inexistence of GPUs and high-performance CPUs, and also due to the scarcity of image data.

Henceforth, the development of new architectures was possible with the evolution of computers, namely, faster CPUs and the introduction GPUs, and because of the constant increasing of image data in the internet.

Then, in 2012, it was introduced a new architecture of convolutional neural network, presented by A. Krizhevsky et al [20], named *AlexNet*. (figure 14) It is a network like the *LeNet-5* architecture, presented above, but with more layers, that is, deeper.

This network was developed to classify over 1.2million high resolution images in the ILSVRC (ImageNet Large Scale Video Recognition Competition) 2012 contest [33]. The ILSVRC, is an image classification contest, where teams develop neural networks, to classify a set of images in 1000 classes. The *AlexNet* network achieved a top-5¹ error of 15.3% [34], a good result compared to the later years contestants, that had an error rate above 25% [35].

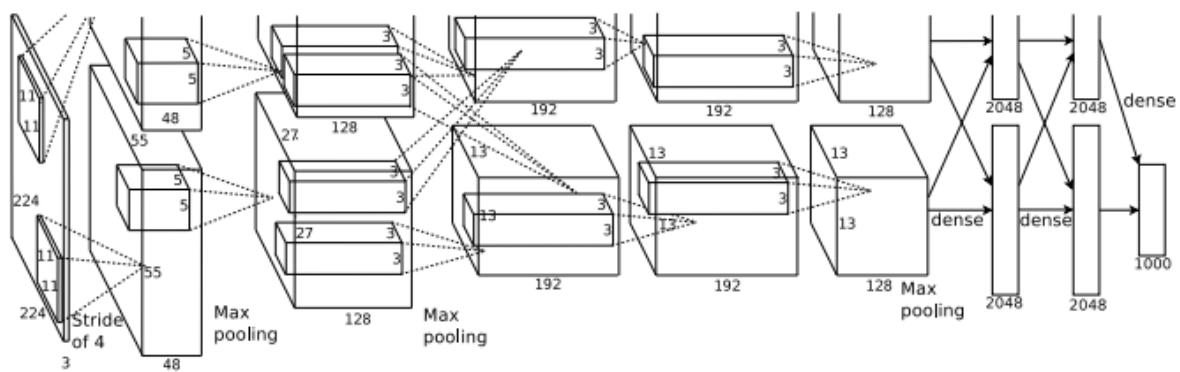


Figure 14 - AlexNet architecture [20]

The introduction of this network revolutionized the area of deep learning, which proved that CNN can be used to learn complex features on images and that was the network that drove its constant development.

In the next year, in 2013, P. Sermanet et al. [36] presented a CNN architecture, named *Overfeat*, which was in the ILSVRC–2013, with 29.8% of top-5 error on Classification plus localization challenge, and with 14.1% [37] on the classification competition. Even though, this network was a derivate from *AlexNet*, it obtained better results on the contest, demonstrating that if a CNN is trained not only to classify, but to detect and locate objects in images it boosts the network performance. The introduction of this network brought a novel method for localization and detection by accumulating predicted

¹ If the correct label is within the top 5 five classes predicted by the network

bounding boxes, that later gave rise to other case studies, like the R-CNN architectures [38]–[40], which is the one of the current leading approaches for object detection.

Also, in the same year, many other new CNN were developed, for example the *ZFNet* [41], which was one of the best networks on the classification task, obtaining 13.5% [37] of top-5 error on the challenge. In 2014, there was developed two new architectures that had outstanding results on the ILSVRC-2014 contest, the *GoogLeNet* [42] and the *VGG* network [43], with 6.6% and 7.3% [44] respectively, of classification top-5 error rate.

The *VGG* network proved that deeper networks can learn more complex patterns and thus have a better performance in image recognition. The *VGG* creators developed various architectures as displayed in figure 15, whose difference is the depth of them, that is, the number of layer that each network possesses.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv1-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 15 - VGG configurations [43]

The main problem is that if the network is deeper, more parameters must be computed, and thus it is more computational expensive, so the *VGG* architecture has 3x3 filters on all the convolutional layers, to reduce the number of parameters to be calculated. It is still one of the most used network in the object detection architectures, since it has proved that can reach a good level of abstraction. However, in comparison to the *GoogLeNet* it is somewhat computationally heavy.

The *GoogLeNet* network brought a new concept to deep neural networks, known as Inception, demonstrated in figure 16. The objective of their architecture was to reduce the computational cost,

and in the same time to maintain the performance of previous CNNs showing that CNN layers do not need to be stacked up sequentially.

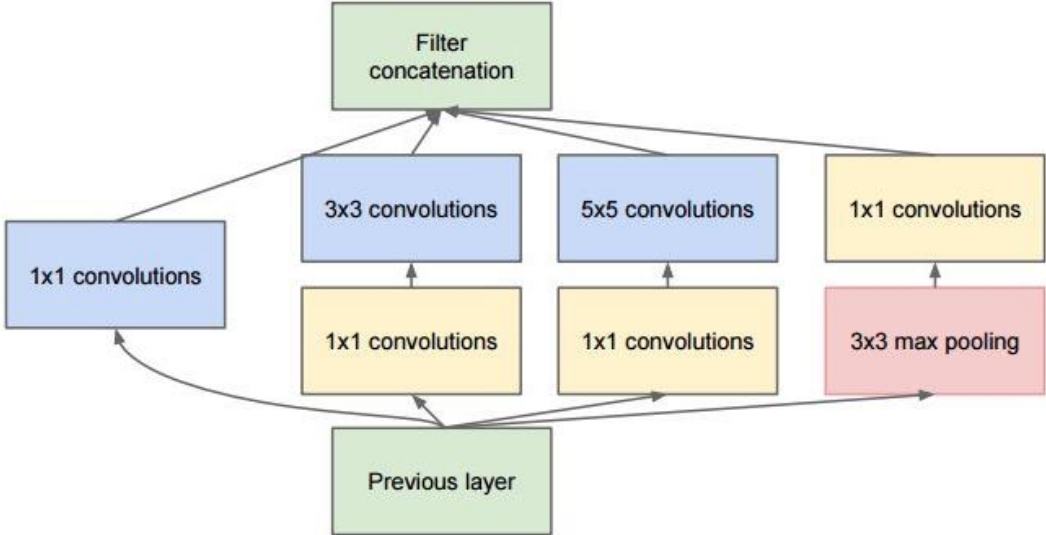


Figure 16 - Inception Module [42]

At first glance, the Inception module is the combination of parallel convolutional filters (1x1, 3x3 and 5x5), that are applied to the input layer (Previous layer), and in the end, they are all concatenated. The *GoogLeNet* network consists on a stack of Inception Modules.

The main insight of this Module is the usage of 1x1 convolutional filters after convolutions. The 1x1 convolutional filters were firstly used by M. Lin et al [45], that developed a neural network called network-in-network “NiN”. The *NiN* consists of a network with mainly 1x1 convolutional filters, that were used to provide more combinational power to the features of convolution layers. The usage of 1x1 convolutional filters allows the reduction of operations to compute, and thus it is less computational expensive.

The best architecture developed in the ILSVRC was the *ResNet* [9], in 2015, which won the contest with a 3.6% [46] error rate. Considering that Humans, in general, has an error rate between 5% and 10% the *ResNet* network outperformed the Human in classifications tasks.

Not only the network had excellent results on the classification contest but also established new records on the localization and detection tasks reaching an error of 9%.

The *ResNet* or deep residual network consists on a deep net with 152 layers of depth (much deeper than the previous year’s architectures) and introduced the idea of residual learning to deep neural networks. In this architecture, each layer of depth is called residual block (figure 17).

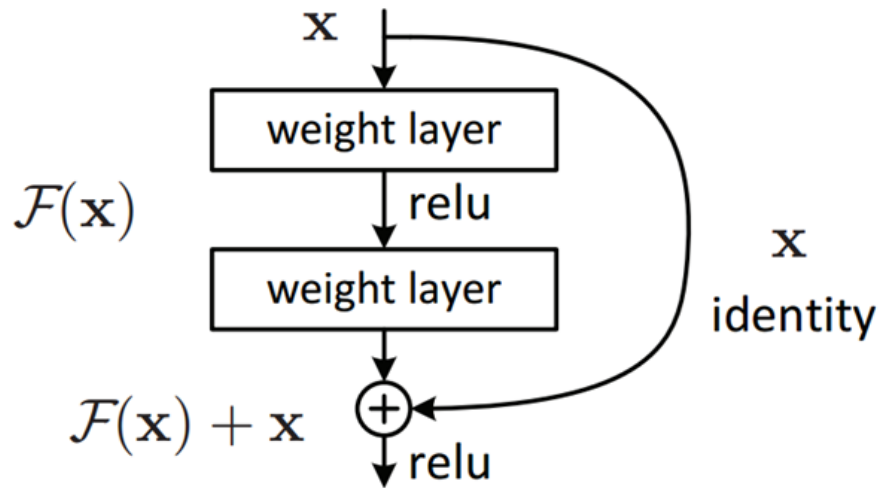


Figure 17 - Residual Block [9]

The residual block basically applies a set convolutional filters to the input x , and the result of those filters is added to the original input. The main difference of this feature on the previous architectures, is that, instead of computing the whole transformation of the input, that is, x to $F(x)$, this residual block allows to only compute the variation of the input.

The vantage of the usage of this residual block is that makes the backpropagation process easier, because of the adding operation that distributes the gradients.

These architectures previously stated are the main developed CNN until nowadays, which express the evolution of this area of study over the years, and the importance of the ImageNet classification challenge on the evolution of CNNs. They are important to this case study, since most projects or applications, in this area, employ these architectures as the core of their algorithm, such as the object detection networks.

2.3 State of the art

Recently, due to the evolution of the CNNs presented above, there were developed some CNN-based models that had outstanding results in the object detection task. As presented above, the concept of predicting bounding boxes introduced on *Overfeat* network [47], gave rise to the development of CNNs whose purpose is the detection and recognition of objects with high performances and lower frame rates, named: RCNN (Region Convolutional Neural Network); SSD (Single Shot Multibox Detector); YOLO (You Only Look Once).

While the R-CNN method is, in general, performed by firstly generating the region proposals and then those proposals are sent to classification and regression algorithms, the other methods (SSD and

YOLO) performs the detections in a different way, handling the detections as a regression problem, called regression-based object detectors.

2.3.1 Region based Convolutional Neural networks (R-CNN)

The R-CNN, was one of the firsts networks that took advantage of the CNNs qualities to accurately detect objects on images.

The R-CNN, displayed on figure 18, presented by R. Girshick et al [38], in 2014, instead of using a sliding window method to generate the image patches to be fed to the network, it uses an algorithm called Selective Search (SS) [48]. The Selective Search reduces the number of boxes to be passed to the network, to around 2000, and by using local features like color, intensity, textures it generates all the possible location of the objects [38].

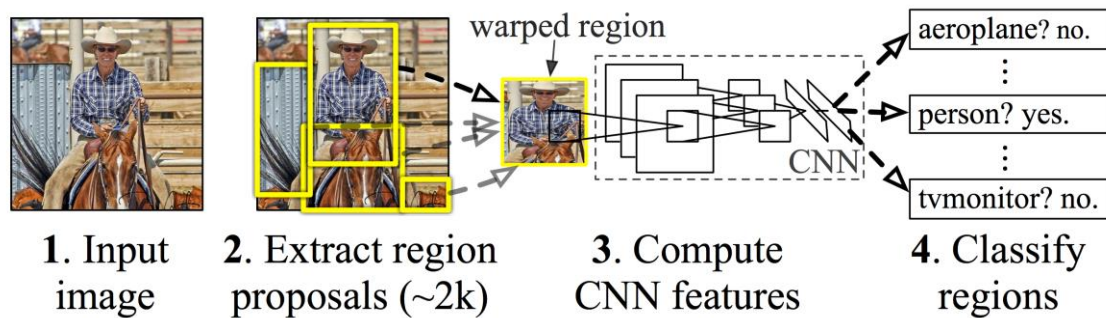


Figure 18 - R-CNN architecture [38]

Once the proposals are obtained, each region is modified to be a fixed size square and it is passed to a CNN, as shown in figure 18, it can be any of the CNNs stated above. Then at the end of the network, each region is classified, to predict if it is an object, and if so what object.

The final step, of this network is to find the box that perfectly fits the dimensions of the object, this is performed by running a linear regression algorithm on the region proposal.

In general, the R-CNN follows these 3 steps [38]:

1. Generation of the region proposals, by running Selective Search;
2. Each region proposal is cropped or warped, into a fixed size box, and fed to a pre-trained CNN, followed by a SVM classifier to get what is the object on that box;
3. Optimize the region proposals by the usage of a linear regression model, after it has been classified.

Even though, R-CNN was very slow, because all the 2000 region proposals obtained by the Selective Search had to be fed to the network, one by one with no shared computation, and in addition the whole training method involves running three separate models [39], which are: the CNN for image classification, on each proposal, the SVM classifier and the linear regression model. Therefore, it makes the train pipeline very complex, that is, slower to train.

With the goal of increasing the speed of R-CNN, it was introduced the concept of Spatial Pyramid Pooling (SPP) on *SPPnets*, in 2014, by K. He et al [49].

The SPP allowed to share the computation on the region proposal method, by firstly, computing a convolutional feature map for the entire input image, and thus generating the proposals on the last layer, by applying the SPP layer on top of it. Afterwards, each proposal is fed to a fully connected layer to be classified [49].

The SPP layer on figure 19 picks the region proposals obtained in the last CNN layer by any region proposal method, and for every region proposal it outputs multiple feature maps of different sizes (4x4, 2x2, 1x1), by using max pooling. Then, all the feature maps are concatenated into a fixed length vector, to be fed to a fully connected layer.

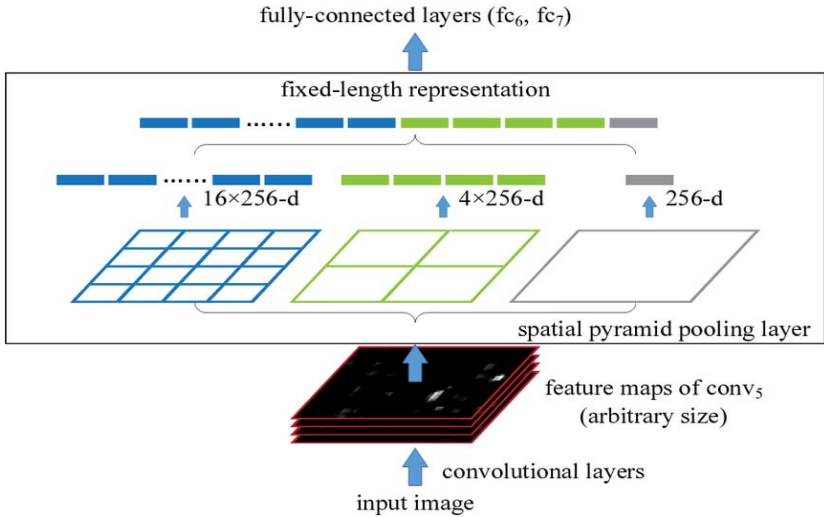


Figure 19 - SPP layer representation [49]

In general, the SPP eliminate the need of applying a region proposal generator in the raw input image, that is, it is not needed for each proposal to pass on the whole CNN, hence reducing the computational cost of running the CNN model various times. Also, it brought the possibility of passing to the network images of different sizes [49].

However, like R-CNN, training a *SPPnet* also involves multi staging, that is, running models separately, the feature extractor, training SVMs and the regression model, that increases the training pipeline.

To address the weaknesses of R-CNN and the *SPPnet*, it was, in 2015, implemented a new R-CNN network, named Fast R-CNN [39] exposed in figure 20.

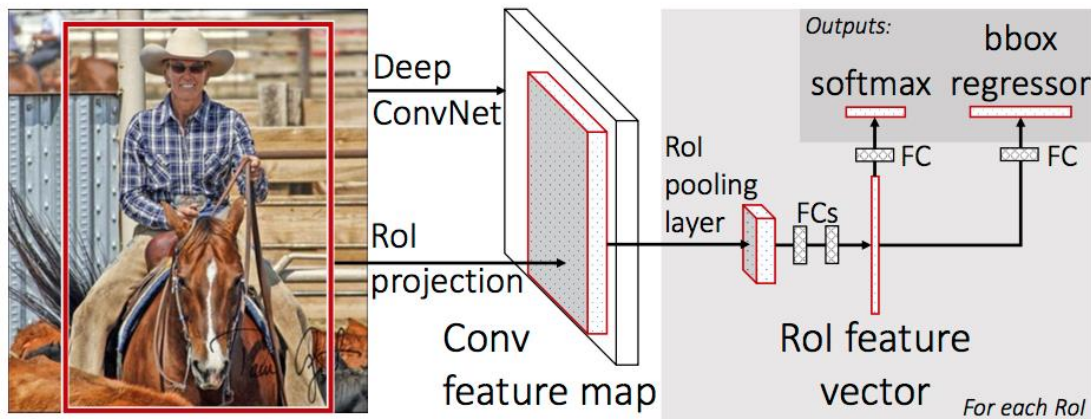


Figure 20 - Fast R-CNN network [39]

The Fast R-CNN took advantage of the method used in the SPP to first obtain the CNN representation of the input image, and then applying the region proposals on that representation. But, instead of applying the SPP on the last layer, it is applied a pooling operation developed by the authors of Fast R-CNN, named Rol pooling. Afterwards, each proposal is classified, by a *Softmax* classifier, and in parallel the bounding box is optimized by using a linear regression [39].

The new contribution that this network brought is the usage of a new type of pooling, namely the Rol pooling, and the aggregation of the feature extractor, the classifier and the linear regression into one single model, rectifying the drawbacks of R-CNN and *SSPnet* and speeding up the training process [39]. The Rol pooling works similarly to the SPP, although instead of outputting multiple features maps, it only outputs one, whose size is defined previously.

This new network achieved outstanding results, in both training and testing speed (table 2), in comparison to the R-CNN architecture.

Table 2 - Comparison of results between Fast R-CNN and R-CNN (Data source [39])

	R-CNN	Fast R-CNN	Speedup
Train time(h)	84	9.5	8.8x
Test rate(sec/img)	47	0.32	146x

Note that both the R-CNN and the Fast R-CNN have the VGG16 as the main CNN for the feature extraction, and the time consumed by the region proposal algorithm, in this case the Selective Search, is not included.

Even with all the enhancements in train and test times, as can be seen in table 2, the Fast R-CNN has a flaw, it still needs a separate algorithm to generate the region proposals. As the region proposal model is very expensive, the whole process could still be improved.

In view of that defect, Ren et al [40], in 2016, developed a new solution to speed up the Fast R-CNN model by integrating the region proposal model into the R-CNN model, they called it faster R-CNN (figure 21). The integration of the region proposal model, was achieved by implementing a Region Proposal Network (RPN) on the network.

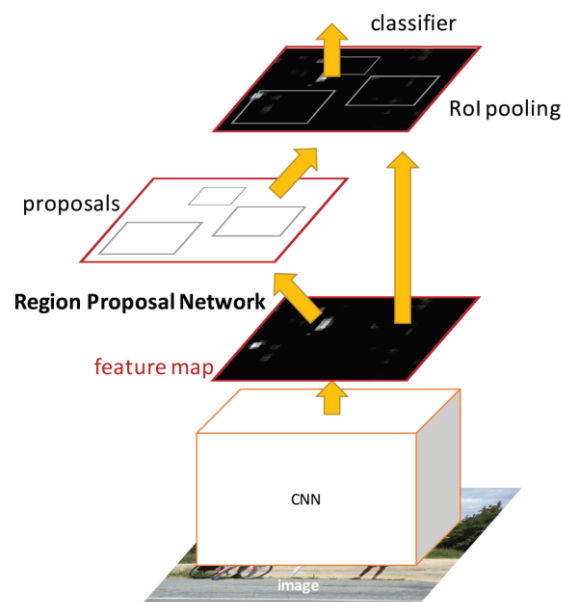


Figure 21 - Faster R-CNN [40]

As stated above the region proposal methods, like Selective Search, search for local features on the image and draw boxes on the possible objects [48]. Then, the authors took advantage of the CNN feature extractor to run their region proposal solution.

The RPN consists on running a sliding window on the feature map and every window location is matched with a set of anchors, more precisely, with a set of boxes with different aspect ratios as represented in the figure 22 [40].

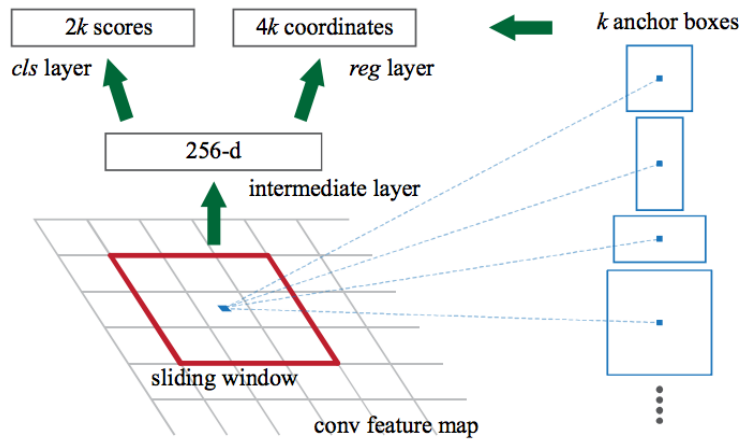


Figure 22 - RPN [40]

So, for each window location there are k possible region proposals, therefore, for each anchor is outputted a class score, that is, the probability of a class to be presented on the anchor, and the corresponding bounding box. The use of different aspect ratios allows the model to more accurately fit the box on the object, because the objects can have multiple shapes and aspect ratios [39].

Posteriorly, each bounding box obtained by the RPN, that most likely has an object, that is, with high probabilities, are passed to end part of the Fast R-CNN that classifies each box and by the regression model it adjusts the box on the object.

The faster R-CNN (figure 21) is represented by one single model, i.e. there is no need of performing outside processes, the network outputs all the possible object with their corresponding bounding box and score for each box.

This new architecture improved once more the speed of the network, because as stated on table 3 the region proposal network is almost cost free, in comparison to others region proposal algorithms.

Table 3 - Timing (ms), for each image until it reaches the end of the network. (Data source: pag.8 [40]).

Model	System	Conv	Proposals	Region-wise ²	Total	Rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	Faster R-CNN	141	10	47	198	5 fps

² The region-wise is all the processes before the region proposals, namely, the Rol pooling, the fully-connected, the classifier and the regression model

By analyzing the results that were obtained, by the authors, the faster R-CNN performed 10x faster than the Fast R-CNN. This enhancement happens mostly because of the RPN, as shown, it only takes 10ms to be performed. On the other hand, the SS takes 1510ms, that is, much slower.

The authors also proved that, the faster R-CNN not only perform the object detection in a faster way, but also it has better results in terms of precision, i.e. the precision of the network in detecting an object as represented on table 4.

Table 4 - Difference on performance between two methods, both using Fast R-CNN object detector with VGG16 (Data Source: [40])

Method	#proposals	Dataset	mAP
SS + Fast R-CNN	2000	VOC2007	66.9
SS + Fast R-CNN	2000	VOC2007+VOC2012	70.0
Faster R-CNN	300	VOC2007	69.9
Faster R-CNN	300	VOC2007+VOC2012	73.2

Until the introduction of the R-CNN there was, as presented above, a chronological evolution of the original architecture with the goal of improving it, in terms of speed and precision, and as proved the faster R-CNN is the culmination of that evolution, having outstanding results in both time and precision. However, in 2017, it was introduced a new R-CNN architecture called Mask R-CNN [50], whose finality was to, along with faster R-CNN, perform image segmentation on each proposal. Establishing a new paradigm in R-CNN networks, that is, not just only locate the object with bounding boxes but it also locates the exact pixels of an object. The Mask R-CNN framework is represented in the figure 23.

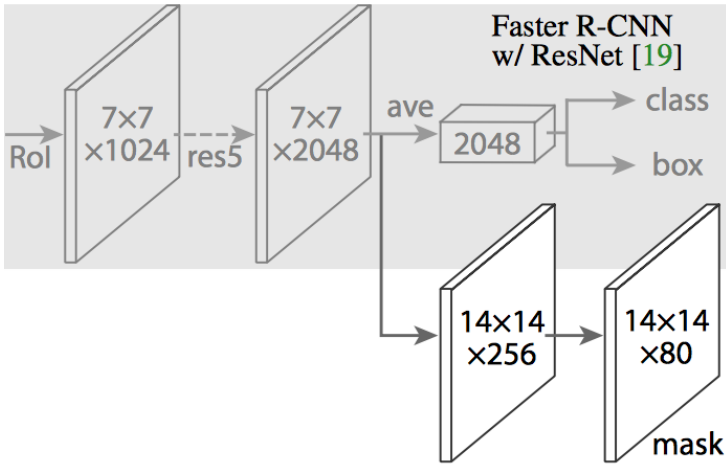


Figure 23 - Mask R-CNN tail [50]

The Mask R-CNN adds to the faster R-CNN a portion that performs the mask, as it shows on figure 23, that is, it predicts if a given pixel is part of an object or not. That portion is just a set of convolutional layers on top of the feature map.

Still, the authors had to make a change on the faster R-CNN model to improve it, namely, on the RoI pooling, because the RoIPool slightly misaligns the boxes from the regions on the original image. Hence, as the pixel-level segmentation requires a more accurate method, the authors developed the RoIAlign [50]. The whole network can be represented in figure 24.

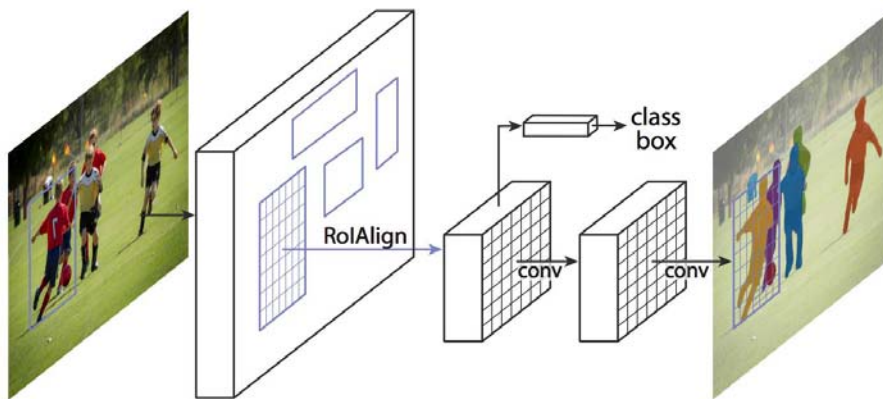


Figure 24 - Mask R-CNN architecture [50]

When the masks are generated they are combined with the region-wise part of the faster R-CNN, namely, the classifications and the bounding boxes, as presented in last part of figure 24, to perform the segmentation and the classification on each object.

2.3.2 You Only Look Once (YOLO)

The YOLO model was introduced in early 2016, by J. Redmon et al [51] and achieved a frame rate of 21fps, much faster than the faster R-CNN.

The YOLO model achieved that results because it performs the whole process of detection with only one pass, that is, a single pass on the whole network predicts at the same time the bounding boxes and the respective class probabilities. In addition, instead of just “see” the region proposals, the YOLO “see” all the input image, therefore, it promotes a better distinction between the objects and the background, enhancing the precision of the network [51].

The YOLO system as shown in figure 25, divides each image into a grid of $S \times S$ squares, and each square is responsible to predict a set of bounding boxes and a confidence score for each box.

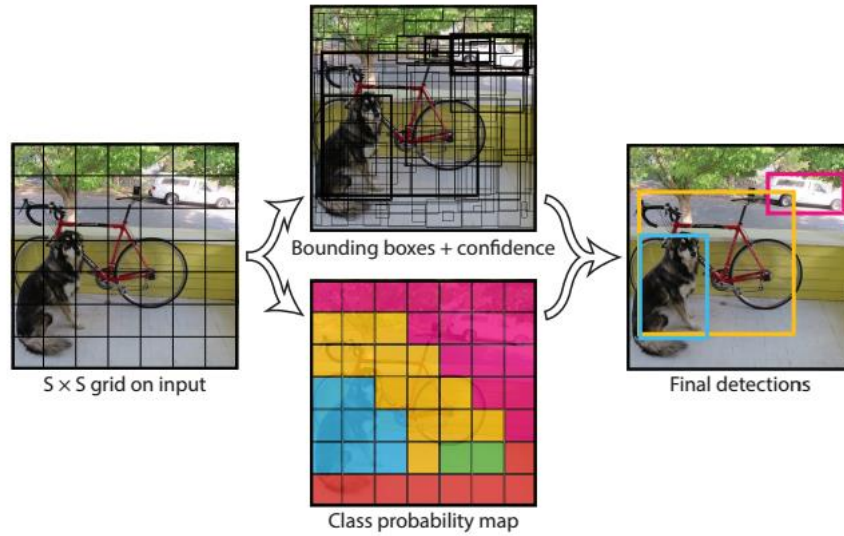


Figure 25 - YOLO model [24]

The confidence is a probability score where, in this model, it tells how accurate is the bounding box and the probability that a certain object is enclosed by that bounding box. It is defined by equation 8 [51].

$$\Pr(\text{object}) * IOU_{Pred}^{Truth} \quad (8)$$

Then, if the object exists on that cell, the confidence score should be the IoU between the predicted box and the ground truth [51].

Afterwards, each grid cell also predicts the class probabilities for each class, represented by the class probability map on figure 25.

Then the confidence score for each bounding box and the class prediction are combined into one final score as shown in equation 9 [51], that outputs the probability that a certain bounding box contains a specific type of object.

$$\Pr(\text{Class}_i | \text{Object}) * \Pr(\text{object}) * IOU_{Pred}^{Truth} = \Pr(\text{Class}_i) * IOU_{Pred}^{Truth} \quad (9)$$

The result of the class-specific confidence score is represented by the final detections, however, note that the final detections on figure 24 are thresholded, that is, it only shows the bounding boxes that have a confidence value above the threshold value.

The CNN architecture that performs the detections on YOLO is shown in figure 26.

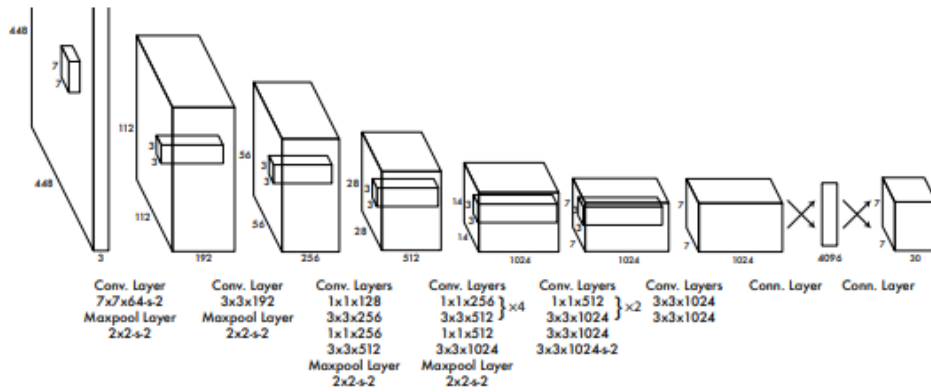


Figure 26 - YOLO CNN architecture [51]

The architecture is composed of 24 convolutional layers (Convolutional and Pooling) followed by 2 FC layers, inspired by the GoogleNet architecture, but instead of using the inception model, they used 1x1 convolutional layer followed by a 3x3 conv layer in similarity to the network in network [51]. In the overall architecture, it is important to note that the last layer is a 7x7x30 matrix.

The 7x7 represents the divisions of the image, that is, the image was divided in 7x7 grids. The x30 represents all the values that each grid cell computes, more precisely, the bounding boxes and all the class probabilities. Hence, the last layer, computes the bounding boxes and the respective class for each cell.

The last layer of Yolo can be defined by equation 10 [51],

$$S * S * (B * 5 + C) \quad (10)$$

Where **S** represented the size of each grid, **B** the number of bounding boxes for each cell and **C** is the number of classes.

The YOLO model achieved one the bests frame rates (45fps) of object detection networks, however, it has some limitations that can be relevant depending on the application. The major limitation is that the YOLO model struggles to detect objects that appears in a small group, because each grid cell can only have one class and have a limited number of bounding boxes to predict, defined by B, and it has difficulties to generalize objects that may have unusual aspect ratios or configurations [51].

Being that the YOLO, in comparison to the faster R-CNN, it is a model with less precision on detecting objects as proved in table 5.

Table 5 - Performance and speed of the object detectors, using the same dataset (Data source: [51])

	<i>mAP</i>	<i>FPS</i>
<i>Fast R-CNN [19]</i>	70.0	0.5
<i>Faster R-CNN VGG-16 [12]</i>	73.2	7
<i>Faster R-CNN ZF [12]</i>	62.1	18
<i>YOLO VGG-16 [24]</i>	66.4	21

Besides the original YOLO model, there were developed new models that had better performance, namely, the YOLOv2 or YOLO9000 [52].

2.3.3 Single Shot Multibox Detector (SSD)

The SSD is also one of the main object detection networks, based on a regression problem and has outperformed both faster R-CNN and YOLO in both accuracy and speed tasks and is current the best object detection system that has the perfect balance between speed and accuracy.

The SSD (figure 27), was introduced in late 2016, by W. Liu et al [53], and has achieved an accuracy of 74%*mAP* at 46*fps*, the table 6 represents the results of SSD in comparison to the other leading object detectors.

Table 6 - Results of the current leading object detection network using the same dataset (Data source: [53])

<i>Object detector</i>	<i>mAP</i>	<i>FPS</i>	<i>#Boxes</i>	<i>Input resolution</i>
<i>Faster R-CNN (VGG 16)</i>	73.2	7	6000	1000x600
<i>YOLO (VGG 16)</i>	66.4	21	98	448x448
<i>SSD300</i>	74.3	46	8732	300x300
<i>SSD512</i>	76.8	19	24564	512x512

The table above proves that the SSD has relatively good results regarding both accuracy and frame rate, even with more detections, represented by the number of boxes.

F

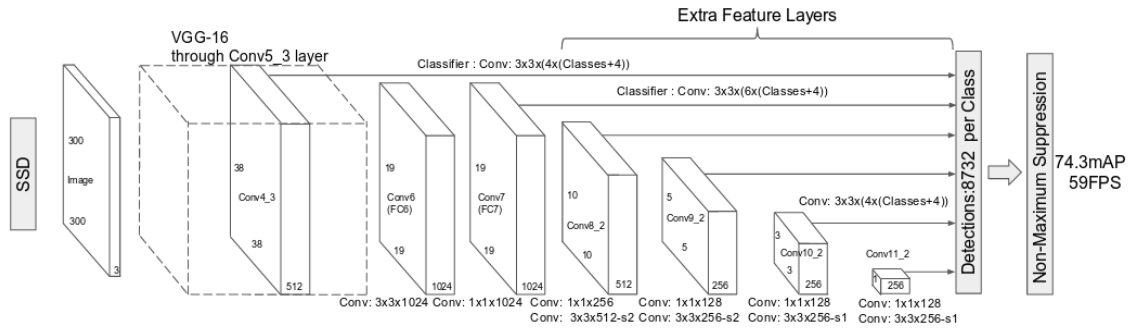


Figure 27 - SSD architecture [53]

The SSD network is built on a standard VGG-16 architecture to extract the features maps, however, the fully connected layers are switched to convolution layers that progressively reduce the size of the input thus adding other feature maps at lower scales.

The main insight of the SSD model is that it performs the detections by applying a set of convolutional filters, of size 3x3, at each feature layer. The existence of multiple layers of different scales allows these predictions to be performed at multiple scales and ratios, contributing to a better performance. Each filter outputs a score for each class and a boundary box [53].

In addition, at each cell of a feature map, for multiple maps, is associated a set of default bounding boxes centered at that cell. Therefore, instead of predicting boundary boxes from a global coordination, the box predictions are relative to a default boundary box at each cell (figure 28).

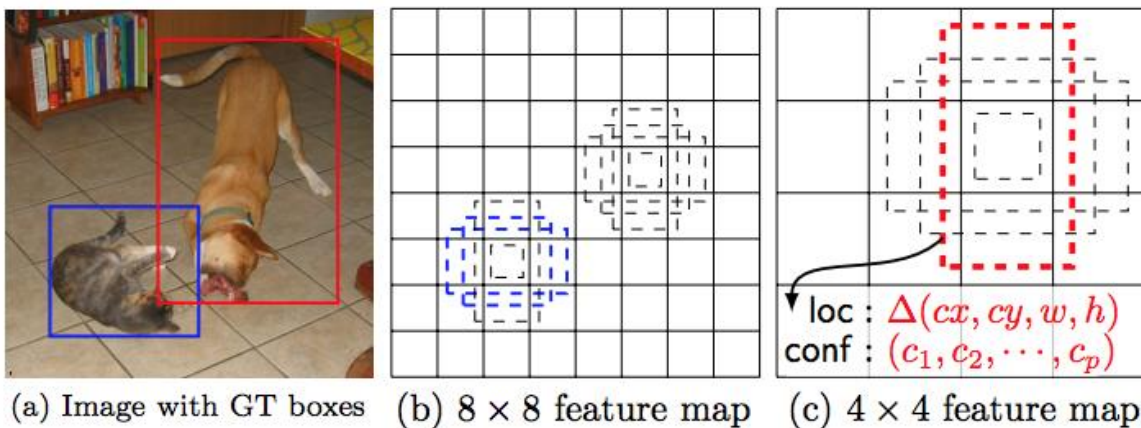


Figure 28 - Multiple scale feature maps and default boundary boxes [53]

So, for each feature map cell it is predicted the box offsets relative to the default box with the correspondent class scores, that indicates the presence of a class.

In training the default boxes are matched to the ground truth ones, to determine which default boxes correspond to a ground truth. These matches are represented by positive and negative matches. The match is positive if the corresponding default box as an IoU over 0.5, otherwise it is negative [53]. This method of computing the matches, also known as Multibox [54], provides the starting point of the training, that is, instead of starting with prediction from a random coordinate, the predictions start with the positive matches and as the training progresses the predictions tries to fit the ground truth boxes [53]. For example, in figure 28 the ground truth of the cat matches the 2 two default boxes in the 8x8 feature layer, and the ground truth relative to the dog matches to one default box in the 4x4 feature layer, all the other boxes are considered negative matches.

Also, as shown in figure 28, each feature map is responsible to match a scale of the object. Then, for each feature layer is defined a scale for the default boxes that can be represented in equation 11 [53]:

$$s_K = s_{min} + \frac{s_{max} - s_{min}}{m - 1} (k - 1), k \in [1, m] \quad (11)$$

s_{min} is the minimum scale, usually between 0,1 and 0,2, ***s_{max}*** corresponds to the maximum scale, normally 0.9, and ***m*** is the number of feature layers.

Beyond the scale, it is also defined the different aspect ratios for the various default boxes, which usually are defined as follows, $\{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$ [53].

The size of the default boxes, width, in equation 12 [53], and height, in equation 13 [53], are calculated as follows:

$$width = scale \times \sqrt{aspect\ ratio} \quad (12)$$

$$height = \frac{scale}{\sqrt{aspect\ ratio}} \quad (13)$$

In addition, for the aspect ratio of 1 is defined an extra scale that is calculated by equation 14 [53].

$$s'_k = \sqrt{s_k s_{k+1}} \quad (14)$$

Therefore, the number of default boxes in a feature map cell is defined by the chosen aspect ratios for the default boxes, normally 4 or 6.

Once the matches are computed, it is obtained a bunch of positive and negative matches, however, most of the default boxes will be a negative match, resulting in a high variance between the negative matches and the positive matches in a training set that affects the performance of the network. To surpass this drawback is used a hard-negative mining, that picks the boxes with the highest loss and keeps the ratios of positive and negative at 1:3 [53]. The negative samples are important because it “tells” the network what are the incurrent detections providing a better distinction of the background.

The authors of SSD stated that the application of data augmentation on the dataset played a crucial rule on improving the accuracy of the network up to 8.8%*mAP* [53], which allowed the network to be more robust to various object sizes.

2.3.4 Related work

These previous networks [9], [20], [42], [43], [45], [47], [55]–[58] are the main CNN based artificial vision architectures, and thus they are used in a wide range of computer vision applications, for example to detect different animals [59] for traffic sign recognition [60], for face recognition [61], [62], to colorize black and white images [63] and for image caption [64]. Being also used as the core feature extractor in the object detection networks [38]–[40], [50]–[53] as presented above.

Still, only recently, with the evolution of the CNNs, this type of deep network has been used in robotic vision applications [65]–[71].

In [65], it is used a CNN-based vision model on a mobile robot to perform obstacle avoidance, on which they use a CNN model based on the *AlexNet* network to provide to robot the movement decision, that is, where the robot must follow to avoid the collision with the obstacle, straight, left or right.

Other application presented in [66], is the use of CNNs for robot-navigation, that is, to estimate the robot heading direction using raw spherical images, they also based their CNN on the *AlexNet* architecture.

Pasquale et al [67] implemented a CNN vision system for a humanoid robot for real-world object recognition of seven different categories (laundry detergent, plate, dishwashing detergent, sponge, cup, soap and sprayer).

There is also the development of a vision system to detect pedestrians using CNNs [68], [69]. Where Hosang et al [68] used the R-CNN with *AlexNet* as the feature extractor, and in [69] was tested both

AlexNet and *GoogLeNet*. As for [70] it is used the CNNs topology to localize balls in robotics soccer, and in [71] to detect humanoid robots.

3. METHODS AND METHODOLOGIES

The model prototype was implemented on Ubuntu 16.06 operating system on a Toshiba L50-B-1JU which has an Intel 4-core CPU of 2.40GHZ and an AMD R7 GPU. As for the development environment, the main libraries are *OpenCV* and *Tensorflow*, using Python as the main programming language.

The *OpenCV* is used for the image processing tasks, namely, the reading of images, the acquisition of the RoI (Region of Interest) in images as well as the information about it, the application of transformations on images and the execution of the video cycle. It is mainly used for the creation of the dataset to train the network.

The *Tensorflow* library is a high-level library from Google for the implementation of neural networks [72], it contains built-in functions that allows the creation of the neural network model, like the creation of layers (convolutional layers, the pooling layers) with the respective variables or tensors, as well as the dimensioning of the filters for the layers. In addition, it has functions that defines the methods for the training processes, namely, the classifiers, the optimizers and the loss functions.

After the model is defined the *Tensorflow* can run the whole model by feeding an input image and the ground truth labels for training outputting the result.

The *Tensorflow* also contains an API, called *Tensorboard*, that is used to display the results of the network in the training task, namely, the average precision of the network in both training and validation tasks for all the trained classes as well as the values of the losses.

As for the Hardware, it depends on the used device, since the *Tensorflow* only has support of NVIDIA's GPUs, it is used the CPU for the all processes. The built-in camera of the laptop is used to acquire the dataset and to test the network.

3.1 Model Framework

For the development of the object detection, with artificial intelligence, it was necessary to divide the problem in three main parts, that are performed sequentially, namely, the acquisition, the training and the testing parts (figure 29).

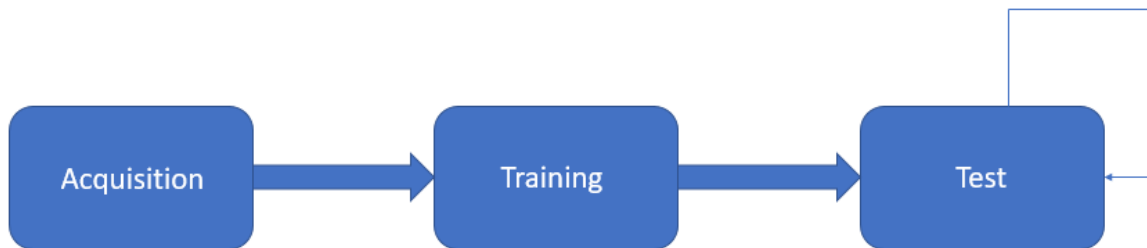


Figure 29 - Model framework

The acquisition part is where the different frames are being acquired as well as the respective ground truth information to create the dataset needed to train the network, represented in figure 30. Since the main goal is to perform object detection, the ground truth is composed of the name of the object as well as the coordinates (xmin, ymin, xmax, ymax) of the bounding box.

The training task process that information and trains the network previously dimensioned (figure 31).



Figure 30 - Acquisition of the dataset where (a) is the raw image with the bounding box and (b) is the corresponding ground truth information, namely, the box coordinates (xmin, ymin, xmax, ymax) and the respective class. Note that the coordinates presented in (b) is just an example, as it is not the corresponding box presented in (a)

```
lsgop@lar:~/codes/Tese$ tail -f out31.log
nohup: ignoring input
Dataset8: 100%|#####| 2788/2788 [00:10<00:00, 261.40samples/s]
(['[i] Data directory: ', 'data')
(['[i] Validation fraction: ', 0.1)
[i] Configuring the data source...
(['[i] # training samples: ', 2510)
(['[i] # validation samples: ', 278)
(['[i] # testing samples: ', 0)
(['[i] # classes: ', 2)
DATASET PROCESSED
[i] Epoch 1/50: 0% | 0/313 [00:00<?, ?batches/s](['[i] Project name: ', 'tests/t_data8_e50_lr5_lr5_lar')
(['[i] Data directory: ', 'data')
(['[i] VGG directory: ', 'vgg_graph')
(['[i] # epochs: ', 50)
(['[i] Batch size: ', 8)
(['[i] Tensorboard directory: ', 'tensorboard/tb_data8_e50_lr5_lr5_lar')
(['[i] Checkpoint interval: ', 10)
(['[i] Learning rate: ', 0.005)
(['[i] Learning rate decay: ', 0.97)
[i] Creating directory tests/t_data8_e50_lr5_lr5_lar...
(['[i] Starting at epoch: ', 1)
[i] Configuring the training data...
(['[i] # training samples: ', 2510)
(['[i] # validation samples: ', 278)
(['[i] # classes: ', 2)
(['[i] Image size: ', Size(w=130, h=130))
[i] Creating the model...
[i] Training...
[i] Epoch 1/50: 1% | 2/313 [00:27<1:12:04, 13.90s/batches]
```

Figure 31 - Training

The test part is the main task of the model, it is where the frames are being constantly acquired by the camera and each frame is being passed to the previously trained network, outputting all the predicted bounding boxes along with the respective confidence value and the corresponding correct class for each box, as shown in figure 32.

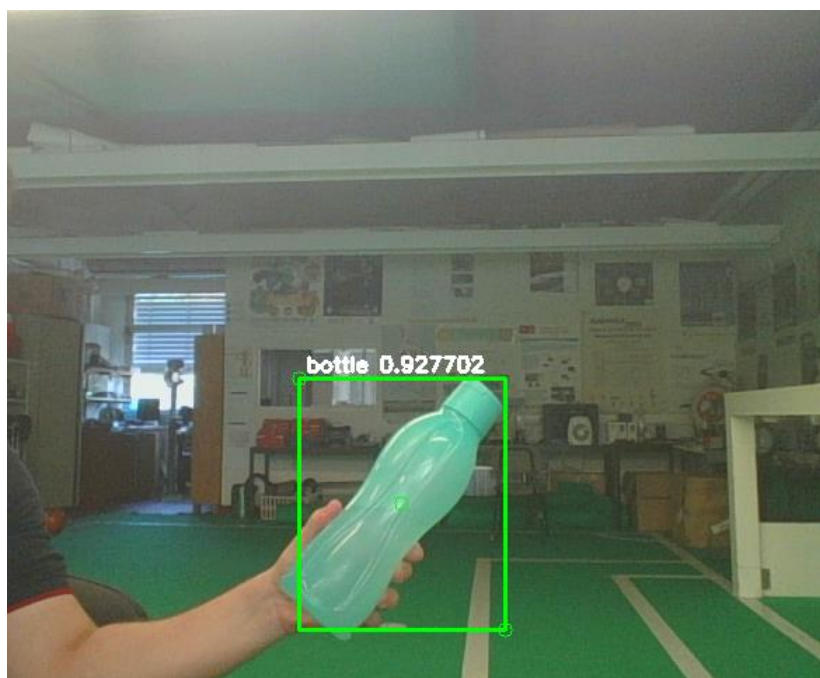


Figure 32 - Testing the trained model

3.2 Acquisition

Since this is posed as a Supervised learning problem it is needed to provide to the network the ground truth information to train the network. Hence, the goal of the acquisition part is to acquire, in real-time,

the video frames as well as the coordinates of the bounding box and the corresponding class, and save the information in a file, as presented in figure 30.

To perform that acquisition it is used computer vision algorithms using the OpenCV library, which allows to acquire video frames from the camera and perform modifications on that frame. These modifications are performed to obtain the region of interest in the frame, also called segmentation. In this case, the RoI is the object that the user wants to acquire for later learning, for example, a bottle, like in figure 33. The mainly used OpenCV functions for the segmentation are:

- findContours [73] – It picks a segmented frame, or a binary frame, and searches for contours on it, that is, points that are connected as a boundary of a shape with the same intensity;
- boundingRect [73] – It obtains the coordinates, i.e. the position of top left corner (x,y), the width and height of a contour that represents a rectangle;
- inRange [74]– Is a filter that searches in an array if there are values that are within a range of previously stipulated values, it set those values to 255 and the rest to 0;
- absdiff [74]– It calculates the absolute difference between two arrays;
- bitwise_not [74] - Inverts all the bits in an array, being that, the bits to invert can be chosen by defining the mask;
- BackgroundsubtractorMOG [75]– It performs background segmentation.

In general, the method to obtain the ground truth is represented in figure 33.

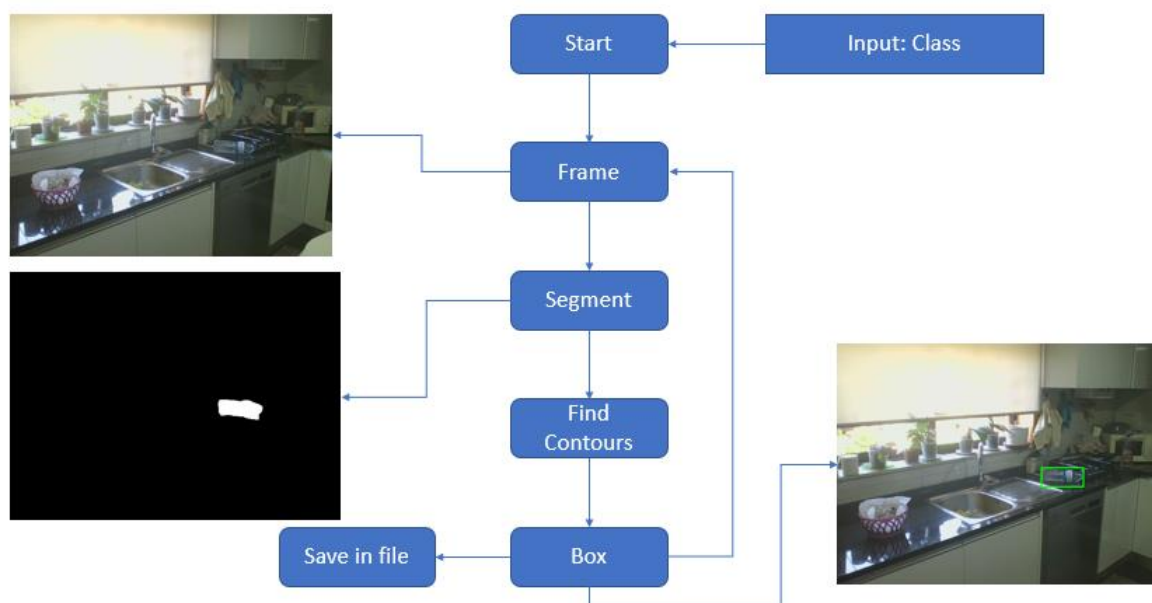


Figure 33 - Acquisition framework

Then, when the script is being executed, it is necessary for the user to place what is the desired class, that is, to which class belongs the object that will be acquired. For this, the user, when executing the Dataset script only needs to put the number of the class in the command line like in figure 34.

```
tiago@tiago-SATELLITE-L50-B:/media/tiago/Elements/codes/ssd-tensorflow$ python Dataset.py --cls 1
init done
opengl support available
('Class :', 'can')
3104
```

Figure 34 - Running the acquisition script

The number of the class corresponds to the position of the class in the label definition vector (figure 35) that represents what are the desired classes for the network to train and detect.

```
#-----
# Labels
#-----
label_defs = [
    Label('bottle',  rgb2bgr((0,255,0))),
    Label('can',  rgb2bgr((0,0,0)))]
```

Figure 35 - Classes definition

Then, the script starts to obtain the frames of the camera and starts to process each frame, segmenting the frame and performing the findContours algorithm to obtain the coordinates of the bounding box by afterwards applying the boundingRect function.

Therefore, to get the coordinates of the bounding box, it is firstly needed to perform segmentation on the frame. The segmentation is the nuclear task of almost all computer vision algorithms, it is where the background is separated from the region of interest. However it is quite difficult to perfectly perform, because as it is normally performed in real time, the frames are constantly changing which causes changes in the frame features, namely, changes in lighting and perspective.

3.2.1 Segmentation methods

The segmentation can be performed in various ways, although the most common segmentation algorithm is the segmentation through color.

The color-segmentation generally consists on changing the color space of the original frame to then apply a filter on the entire frame of a specific color value to eliminate the other colors. The acquisition model using color-segmentation can be represented in figure 36. The source code of the color segmentation is displayed on Appendix B1 page 124.

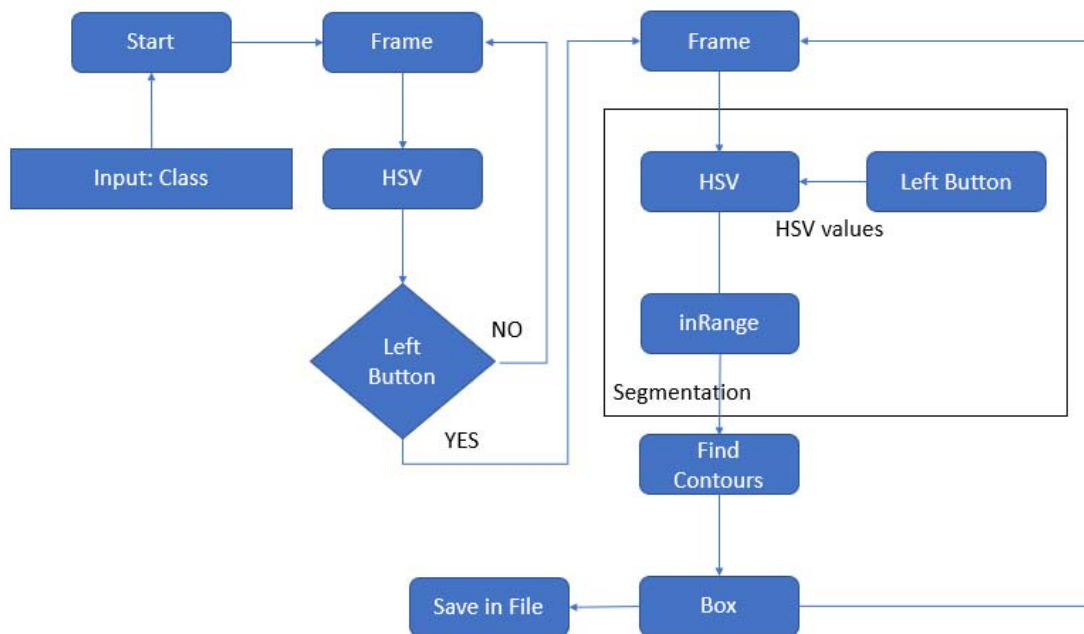


Figure 36 - Acquisition model with color segmentation

At the beginning, the model works in the same way as the general one presented above, although in the segmentation task the color space of the original frame, the BGR (Blue, Green, Red), is changed to HSV (Hue, Saturation, Value) (figure 37).

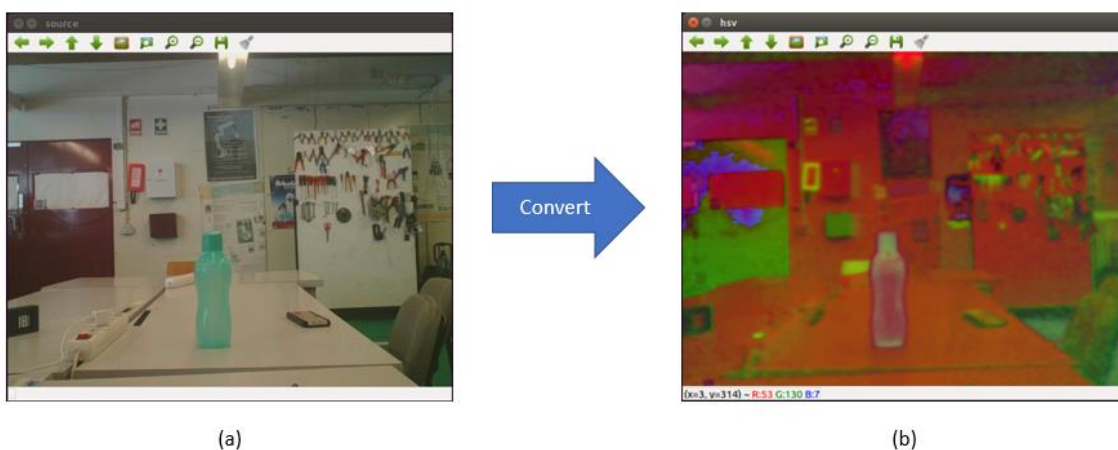


Figure 37 - Conversion of BGR to HSV. Where (a) is the source frame and (b) is the HSV frame

The HSV color space is commonly used because, unlike BGR, the color can be represented by only one value, Hue, from 0 to 360 [76] represented in table 7.

Table 7 - Hue values and the respective color

Color	Hue
Red	0-60
Yellow	60-120
Green	120-180
Cyan	180-240
Blue	240-300
Magenta	300-360

In addition, the color is also described in terms of their shade or amount of gray (Saturation), and their brightness (Value).

After the original frame is converted to HSV the color that the object contains is extracted by clicking on a random position of the object, the green bottle in figure 37(a), with the mouse left button outputting the values in figure 38.



Figure 38 - BGR and HSV values of the clicked position

Note that on the OpenCV the hue values are divided by two, for example, the green hue value instead of being between 120 and 180 it is between 60 and 90.

Then, to extract the green color that represents the bottle, a range of the HSV values is established, namely, the minimum values and maximum values, because it is not possible to segment the exact value of the color, as the color is in constant change, due to some changes that occurs in the environment, namely, variations in luminosity.

The range of the Hue value must be scaled to be a small interval between the value obtained, to prevent another color from being segmented, besides the color of the object, for example a range of 10. Then, taken the values presented in figure 38, the minimum values can be 61 (71-10), and the maximum values is 81 (71+10).

The same relationship cannot be applied for the Saturation and the Value because they are extremely influenced by the environment lighting conditions. Yet, they can be calibrated using OpenCV trackbars that allows the user to change, in real time, both values, as shown in figure 39. The maximum value of both Saturation and Value must be 255, the minimum value is calibrated by the user.

After the values are dimensioned it is applied the filter `inRange`, of OpenCV, which displays the windows on figure 39.



Figure 39 - Color Segmentation

Therefore, with color segmentation it is possible to obtain a relatively good segmented object, by calibrating the minimum value of Saturation and Value, and thus the `findContours` algorithm may obtain a contour that represents the whole object, thus obtaining a box that fits the object, as shown in figure 40.

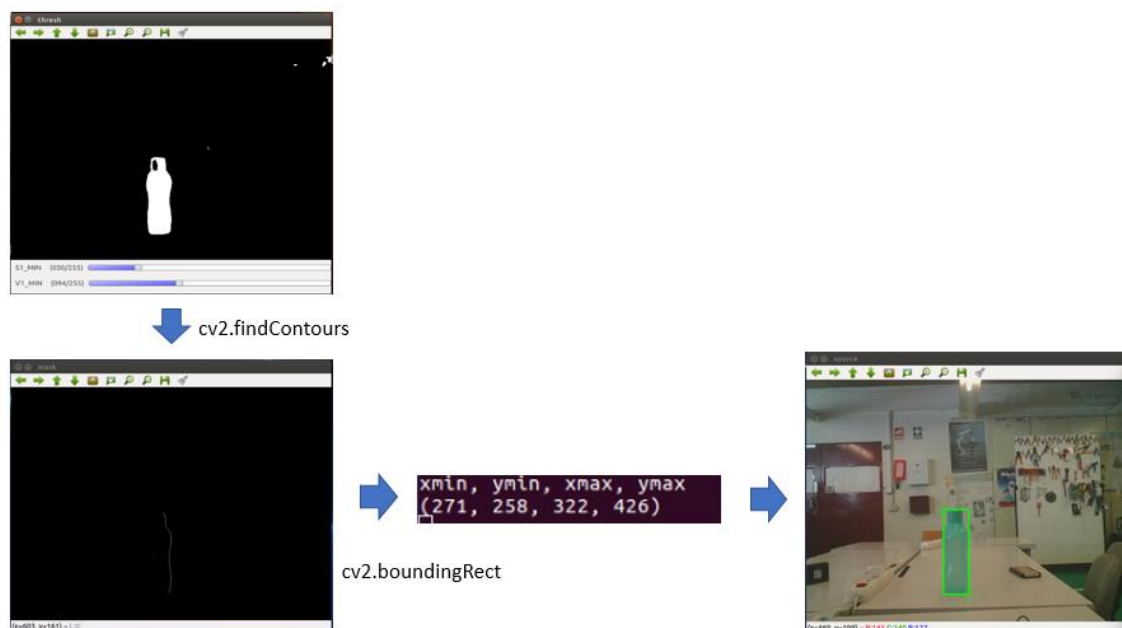


Figure 40 - Result of the acquisition model using color segmentation

The result in figure 40 proves that the color-segmentation is a good method to segment objects that are composed of a uniform color and it reflects the simplicity of the method. However, the color segmentation has many drawbacks, such as:

- The method cannot segment objects that has no color, that is, translucent objects;
- It can only segment one color at a time, then, it struggles on objects with multiple colors;
- The color segmentation can only be applied when the color of the object is different from the background;
- It cannot segment object that are black and grayscale;
- The color values are limited to the ones presented in table 5.

Therefore, the color-segmentation method, since the variety of objects is immense, fails to perform the segmentation of all the objects that are likely to be acquired, and thus it is not a good method to this case study.

On the other hand, another method that can perform object segmentation is the background subtractor, which consists in extracting the moving foreground from the static background.

The background should consist in only the environment that will surround the object, more precisely, the frame without the object, like in figure 41(a). The foreground must be in the same perspective as the background, but with the desired object (figure 41(b)).



Figure 41 - Background (a) and foreground (b) frames

The background subtractor segmentation method consists in subtracting the background frame with the foreground frame to pick the difference between them, that is, the bottle in figure 41(b).

Then, the acquisition model with the background subtractor can be represented in figure 42.

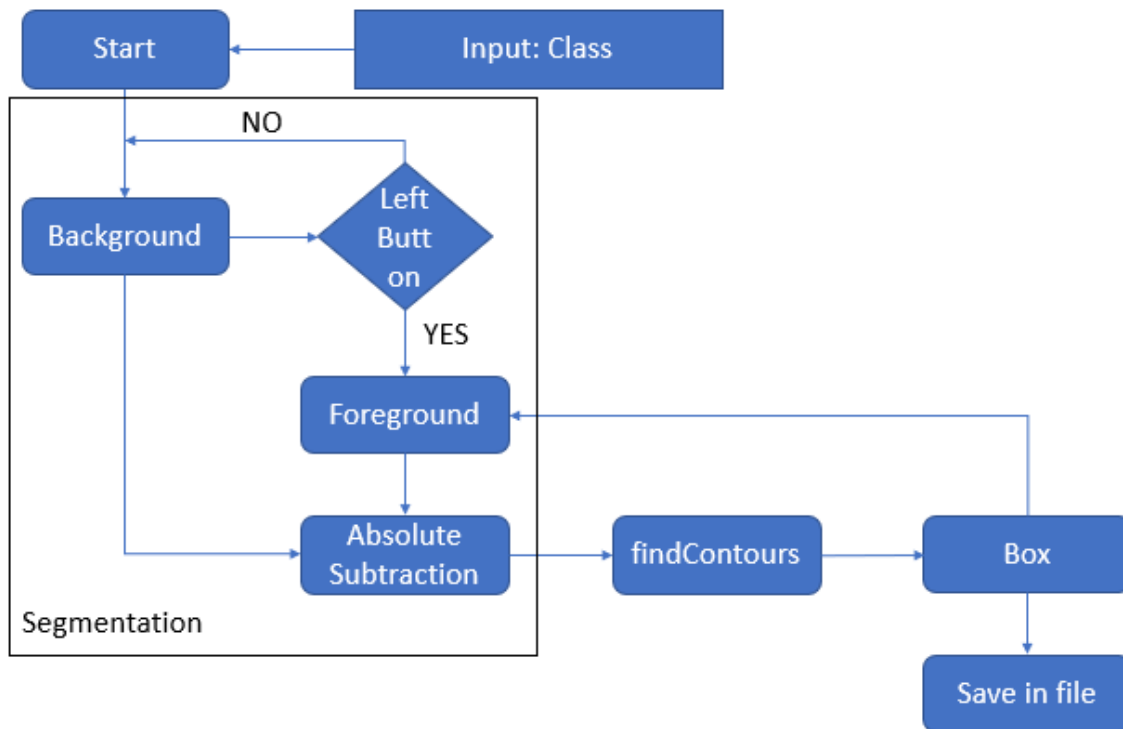


Figure 42 - Acquisition model with background subtractor

Right after the script starts, the background frame starts to be acquired until the user clicks the mouse left button. When the user clicks on it, the background frame is saved and remains the same in the whole program, and the main cycle of the script is performed. The foreground frame is being constantly acquired, and then the absolute subtraction is applied in the actual foreground frame. The absolute subtraction, picks the background frame previously saved and subtracts with the actual foreground frame, using the absdiff in OpenCV.

However, the absolute subtraction has a big problem with findContours, as stated above, the findContours expects an input vector of 1 channel, that is, a binary frame, so if the absdiff performs the subtraction between two 3D vectors, it generates a vector with the same channels, in this case 3, and therefore the findContours cannot be performed unless both background and foreground frames are thresholded, which is not desired in this method and complex to implement.

Yet, OpenCV has a function that performs that background subtraction and outputs a binary frame, or a mask, named BackgroundSubtractorMOG, which changes completely the segmentation part in figure 42 and makes it less complex, as shown in figure 43. The source code is shown in Appendix B2 on page 126.

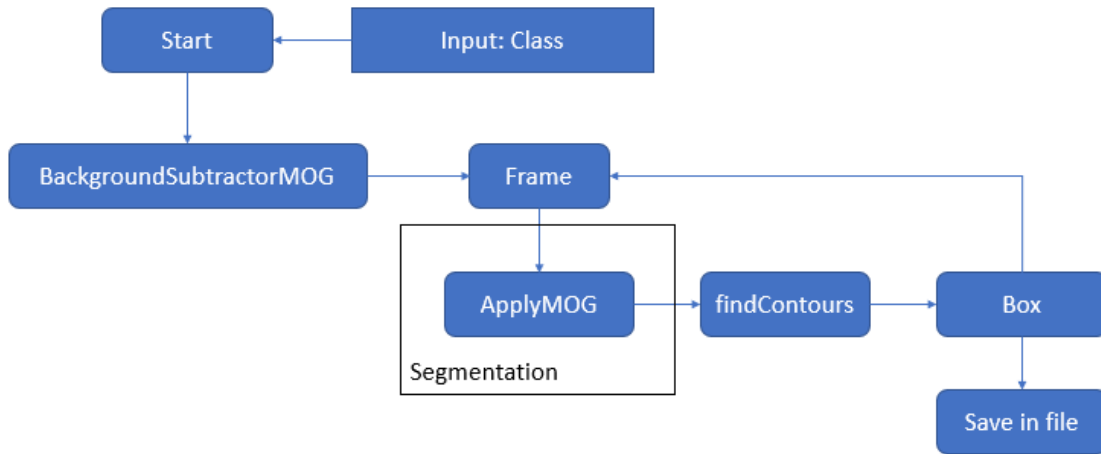


Figure 43 - Acquisition model with MOG background subtractor

The acquisition model with MOG (Mixture of Gaussian) background subtractor, presented in figure 43, reflects the simplicity of the model when the MOG algorithm is used, because the segmentation is performed by only applying the MOG on the actual frame, although, the algorithm needs to create the background subtractor before the video cycle.

As presented in figure 44, the segmentation method presented in figure 43, performs an almost perfect distinction between the background and the foreground.

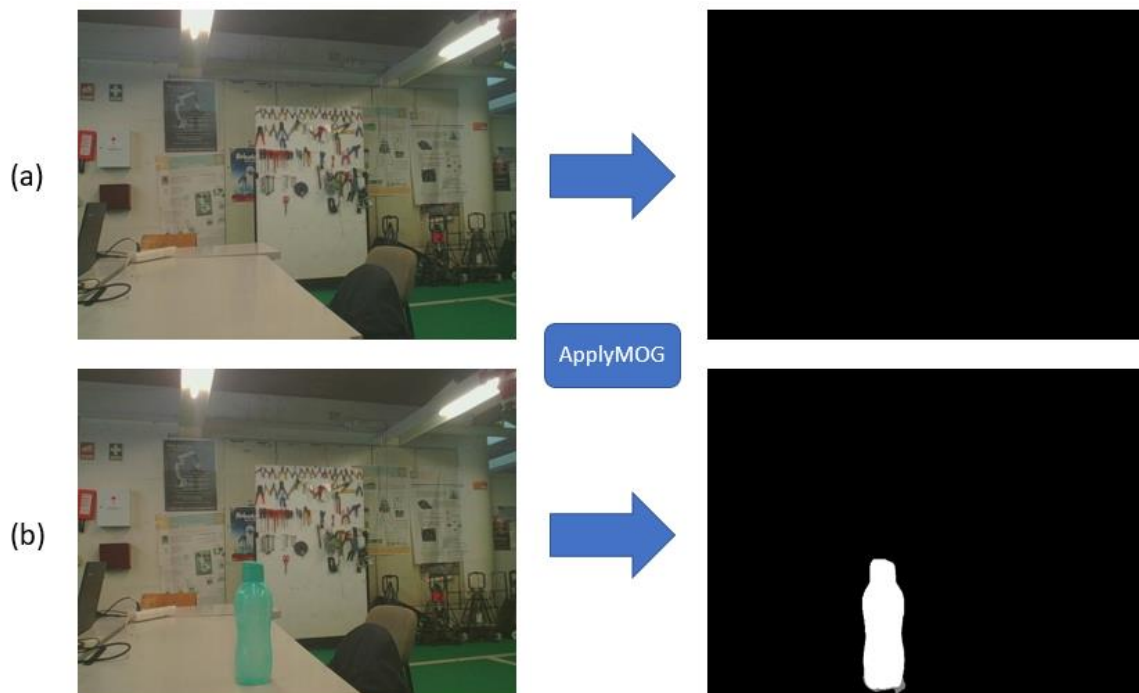


Figure 44 - Segmentation with MOG background subtractor, where (a) is the background frame and the respective segmentation and (b) represents the frame with the object, foreground, and the correspondent segmentation. (a) and (b) are successive frames.

Therefore, an almost perfect distinction reflects on an almost perfect segmentation, as shown in figure 44, and thus, the coordinates of the bounding box are even more precise.

By using this background subtractor the algorithm can acquire any type of object regardless of its color and shape, unlike color-segmentation, proving that it is a very good method to use in the acquisition framework.

However, the method has a major disadvantage, the MOG will segment every difference that happens in the background, that is, every little movement that happens on the background the MOG will extract them, as displayed in figure 45, which may happen because the frames are acquired in real time. So, the background needs to be completely static as every change will be considered as a foreground.

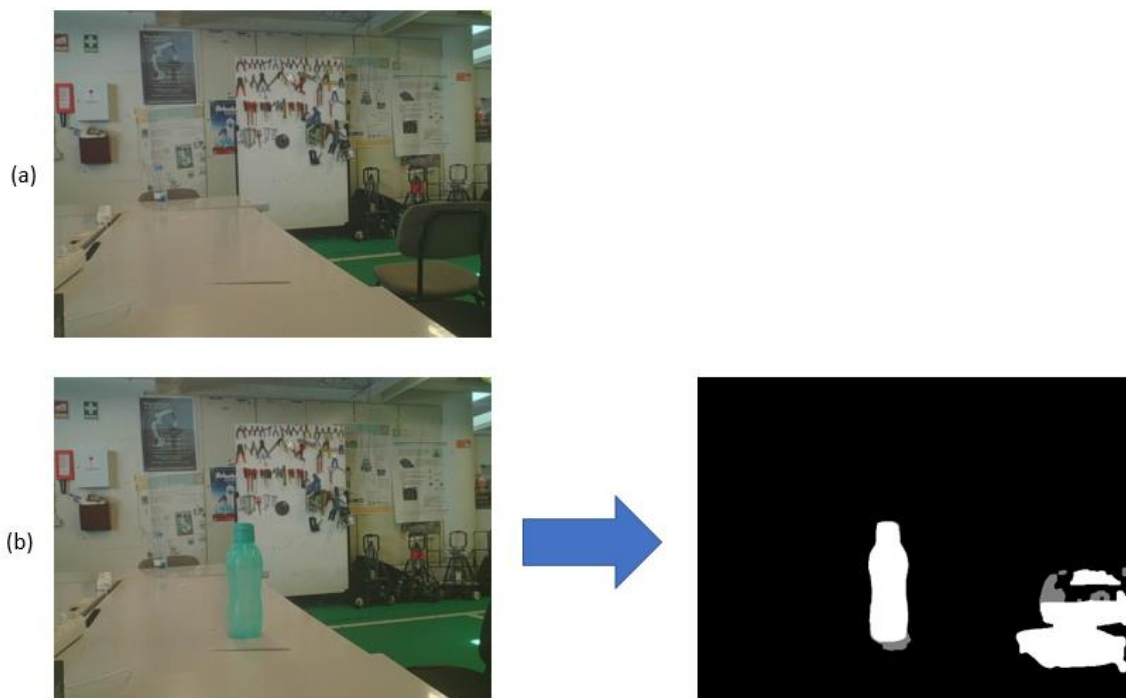


Figure 45 - MOG background subtractor segmentation, where (a) is the background frame and (b) is a consecutive frame and the respective segmentation.

In figure 45(b) it shows how a change on the background, except the object, affects the segmentation of the object, represented by the movement of the gray chair in figure 45(a). Yet, the possibility of acquiring every possible object with this method, cancels this disadvantage.

Nevertheless, one of the main objective of the acquisition task is to extract the information from the object as it moves, that allows the acquisition of the object from various positions and perspectives, for that, the user needs to use something to perform the movement in the object, for example, his own arm, like in figure 46.



Figure 46 - Acquisition with the arm

Since the MOG background subtractor will segment every movement in the background, the arm in figure 46 will also be segmented along with object (figure 47) which is not desired in this task.

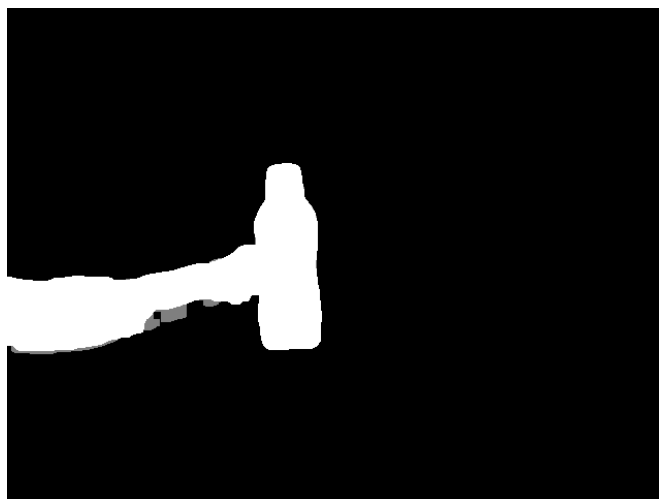


Figure 47 - Segmented frame

So, the algorithm must remove the arm from the segmentation to extract only the object, therefore it was implemented in parallel to the MOG background subtractor a color-segmentation algorithm.

The color-segmentation is used to extract the arm of the user to then perform a bitwise operation, namely the bitwise_not. The algorithm selects the MOG background subtractor mask (segmented frame) and removes what is found on the color-segmentation mask, represented in figure 48.

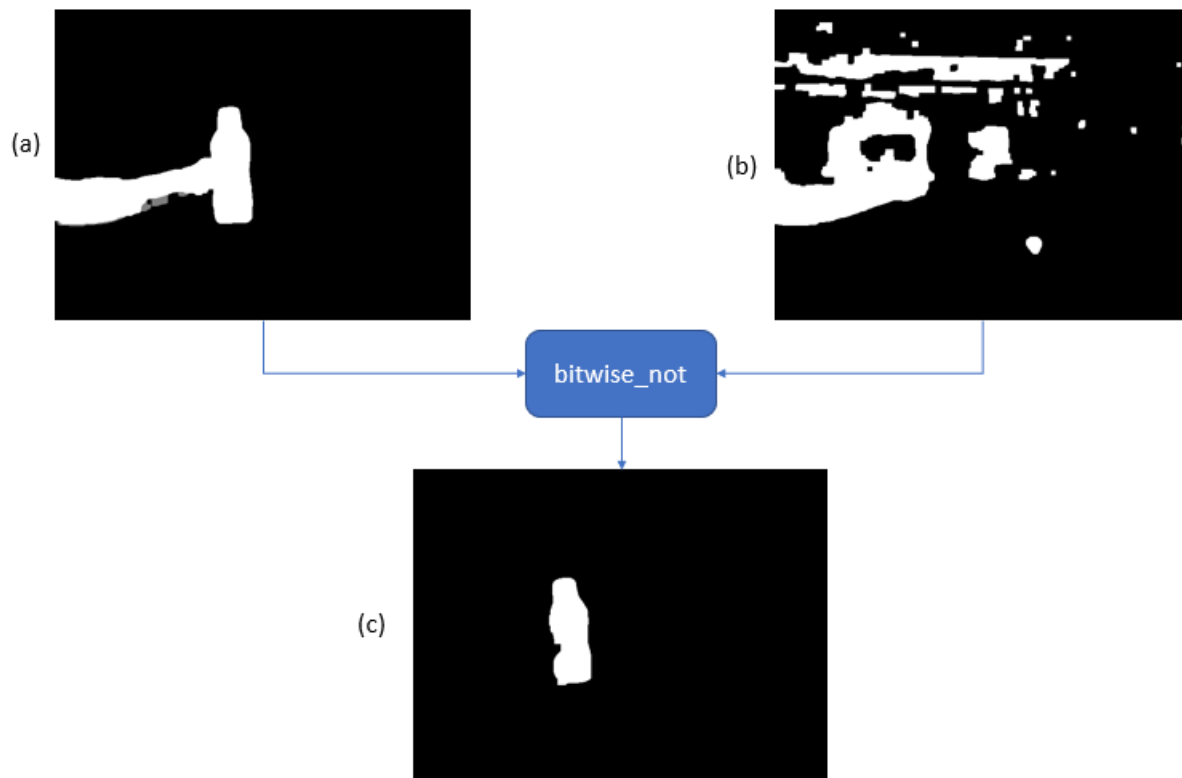


Figure 48 - Application of the bitwise NOT operation on the MOG mask (a) and on the color segmentation mask (b), resulting on the final mask (c)

As proven in figure 48(c), it is possible to extract only the desired object with the usage of both MOG background subtractor and color-segmentation when the user is moving the object with his arm. The MOG mask (figure 48(a)) is responsible to acquire the object and the arm of the user, the color-segmentation mask has the arm without the object.

The bitwise not operation just considers the segmented part of the MOG mask (figure 48 (a)), i.e. it takes the white part of the MOG mask and removes what is common in both masks, in this case, the arm. Everything besides the white part of the MOG mask is not relevant, as it does not enter into the bitwise not operation.

By using this method, it adds two disadvantages to the algorithm, the main disadvantage is that, by using the color segmentation, it removes the picked color from the MOG mask, that is, if the object is composed partially or completely of that color, the object will also be removed in the final mask. The other disadvantage is that the arm used to move the object needs to be composed by a uniformed color, because the color segmentation is only capable of segment one color. However, it adds the possibility of extracting only the object as it is being moved with the aid of the arm.

Even if the method needs to be somewhat controlled it proves that can extract every type of object regardless of its characteristics being the main purpose of the acquisition task.

3.2.2 Final method

Then, since the segmentation method was chosen the acquisition framework in figure 33 can be redesigned into the one presented in figure 49. The source is displayed on Appendix B3 page 128.

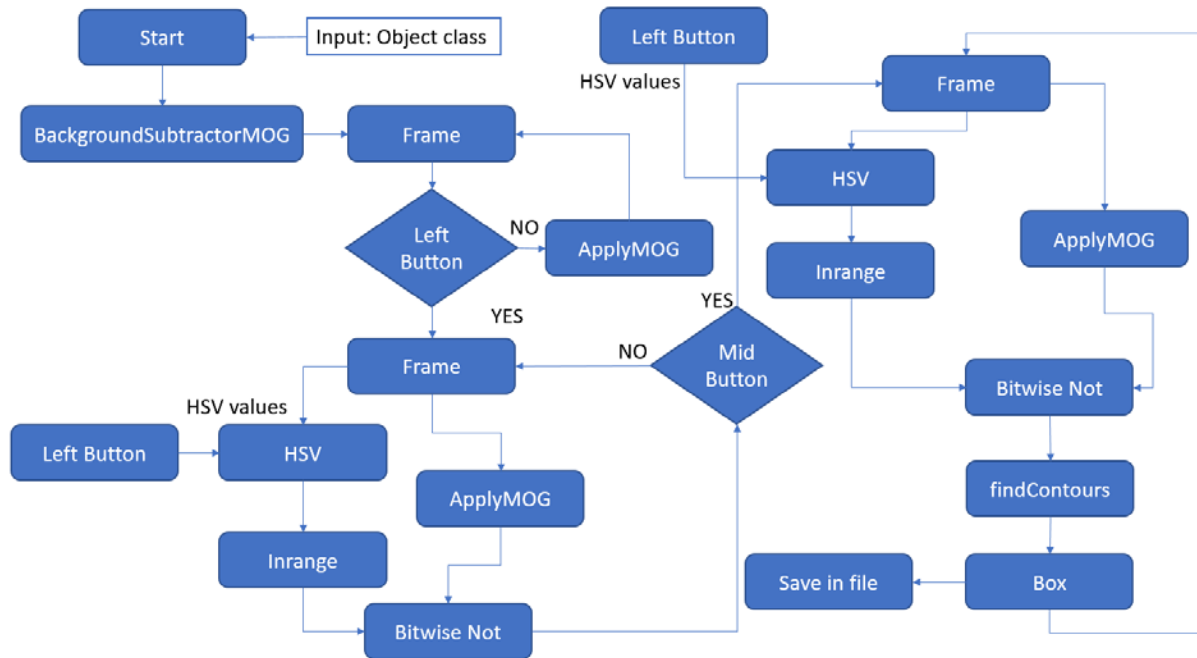
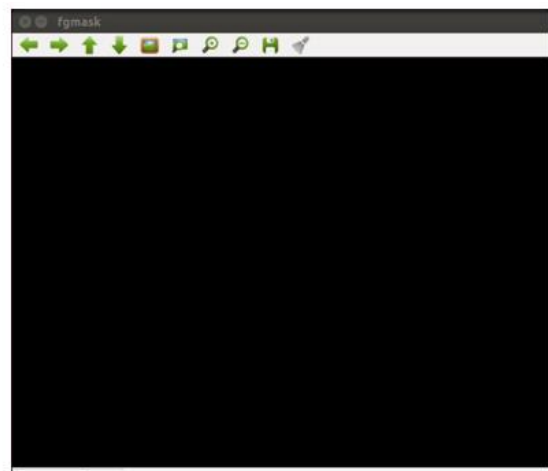


Figure 49 - Final Acquisition framework

So, when the user executes the algorithm it is needed, as presented above, to put the number of target class, like in figure 32, for example, “0” for bottle and “1” for can if the label definitions are like the ones presented in figure 31. After the program is executed, the user will be faced with two windows, the main frame, and the MOG frame (figure 50).



(a)



(b)

Figure 50 - Initial windows of the acquisition algorithm, where (a) is the main frame and (b) is the MOG mask.

These two windows will be shown until the user clicks in the main frame with the mouse left button. The mouse left button is used for the user to click on what it is desired to eliminate from the segmentation, in this case, the arm, that is, the left button will get the HSV values of the clicked position for the color segmentation part. It is also used as a validation of the methods, that is, the user can see if the methods until there are working properly, if it is, the algorithm should move to next stage by clicking on the left button.

By clicking on the arm with the left button one more window will pop up, the window of the color segmentation. This window is for the user to calibrate both Saturation and Value minimum values for the filter, it can also be used to verify if the arm is being properly segmented as displayed in figure 51.

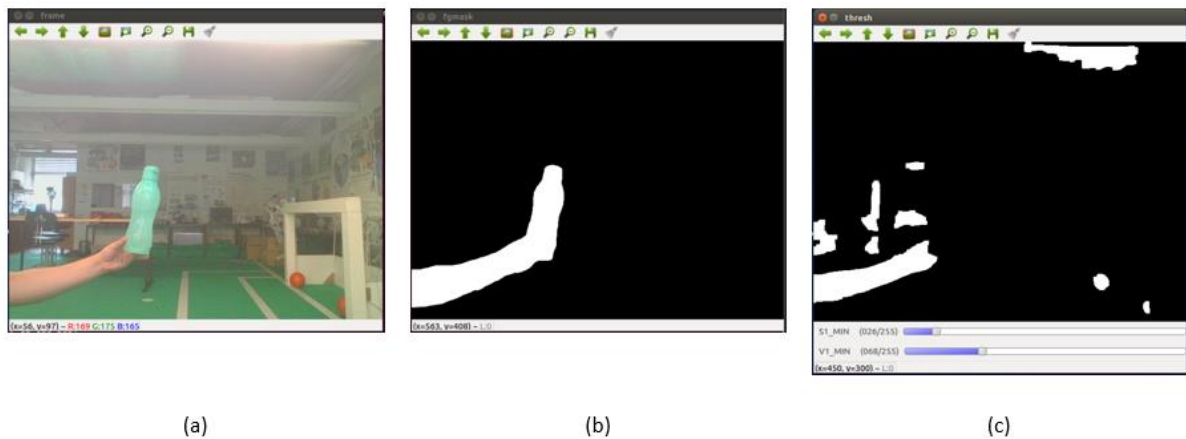


Figure 51 – Color segmentation and MOG segmentation with the arm and the desired object. (a) is the main frame, (b) corresponds to the MOG mask and (c) is the color segmentation mask

To calibrate the minimum values of Saturation and Value of the HSV and to verify if the arm is being well segmented, the cycle will remain until the mouse middle button is clicked.

When the middle button is clicked the algorithm will start to perform the findContours and thus the bounding box of the object will be shown in the main frame as shown in figure 52, and will be saved in a file along with the class of the object.

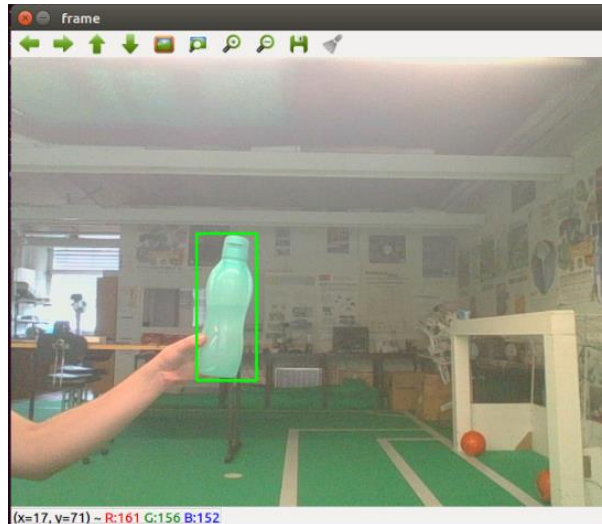


Figure 52 - Main frame with the bounding box

At the end of each cycle the object information is saved. This information is composed of the actual main frame without the bounding box and the coordinates of the bounding box with the name of the object. The frame is saved in the Images folder in the format jpg, the box information goes to the Annotations folder in txt.

To associate the image with the information file it is used an iteration in each cycle, that is, at each cycle, after the mouse middle button is clicked, the iteration value is incremented with 1 and the names of both files are the number of the actual iteration. For example, in the beginning of the dataset creation the iteration number is 1, then when the algorithm starts to acquire the bounding box at each cycle is saved one frame and the respective file with the ground truth information as is demonstrated in figure 53, and thus the first cycle will save the 1.jpg and the 1.txt and so on.

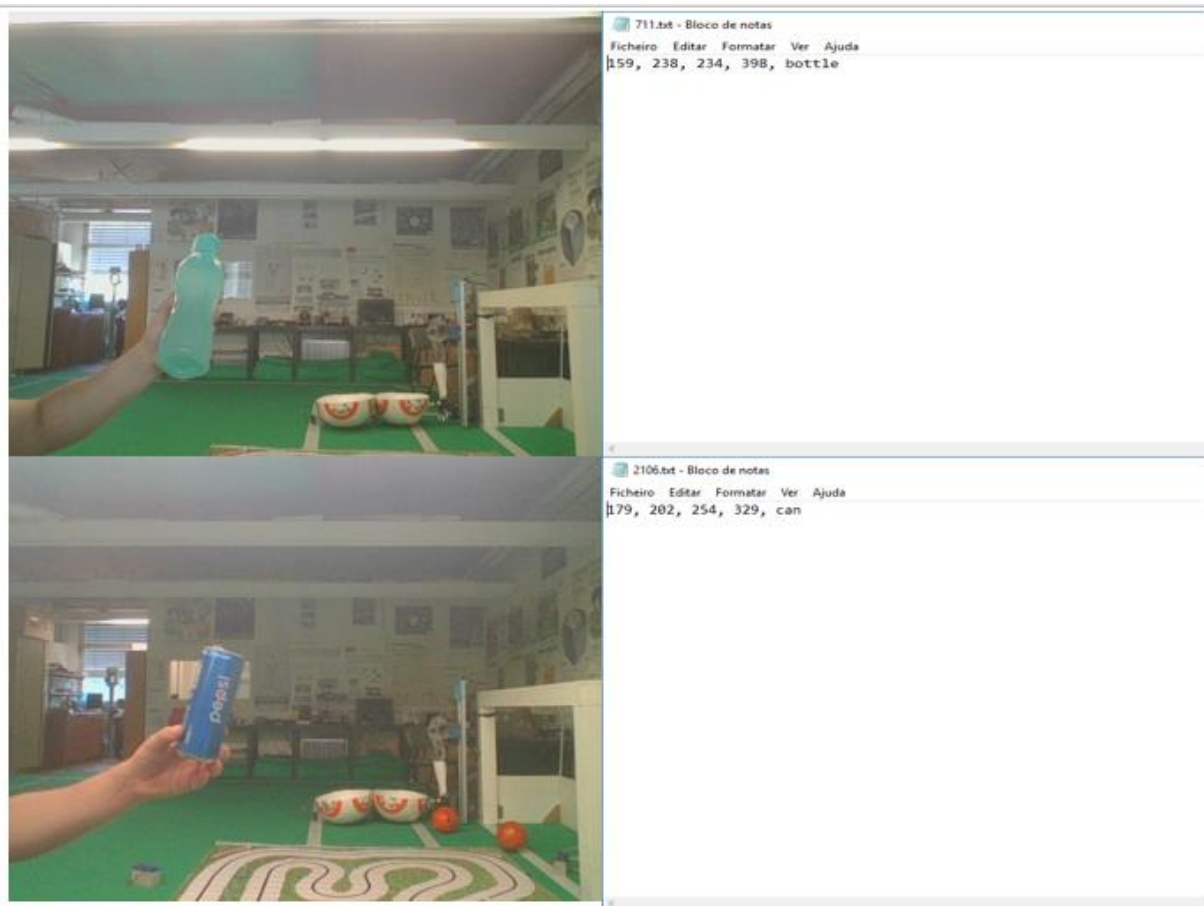


Figure 53 - Saved image frames and the respective files with the ground truth information

The ground truth information represented in figure 53 is composed of the two points that represent the limits of the bounding box, namely, the minimum point (xmin, ymin) and the maximum point (xmax, ymax), in this case (159, 238) (234, 398) for the bottle and (179, 202) (254, 329) for the can, respectively. The last part of the file is the class of the object which the box surrounds.

3.2.3 Problems

There are some situations in the acquisition method that may lead to some errors in the segmentation of the object and therefore the box may be misaligned or not centered at the object. One of the errors that may happen, shown in figure 54, is if the arm is not properly segmented.

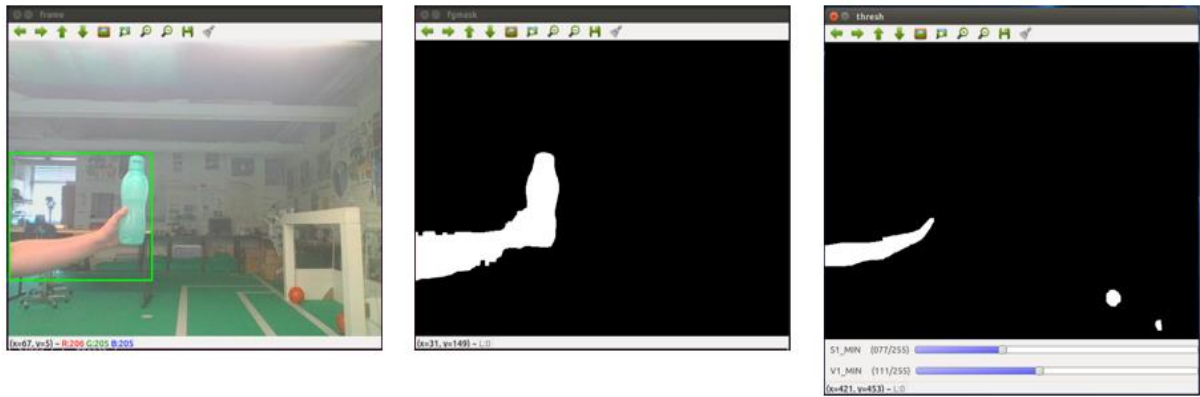


Figure 54 - Bad arm segmentation

If the arm is not properly segmented, the bounding box will not only surround the object, but will also surround the arm, and thus the color segmentation must be adjusted. Hence, the ground truth information about this bad acquisition is not wanted in the dataset, since the bounding box is not represented by the object alone.

Another problem that may happen is in the MOG segmentation which, as presented above, will extract everything that is different from the background. So, every movement will be segmented and thus it may affect the final segmentation even if the color segmentation is properly performed as displayed in figure 55.

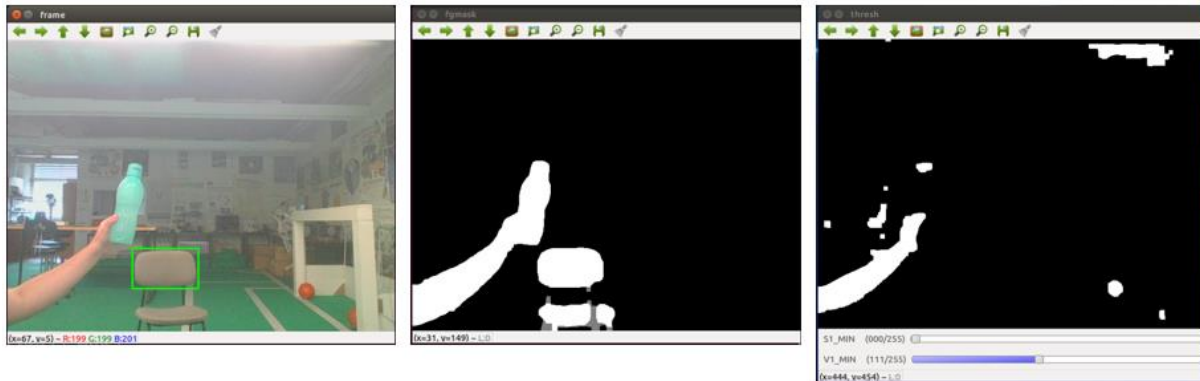


Figure 55 - Bad MOG segmentation

Therefore, if any difference occurs in the background, the difference will be displayed in the MOG mask along with the arm and the object, which will interfere with the findContours method, since it will not be eliminated with the color segmentation. If the contour is bigger than the object contour the bounding box will bound that contour like in figure 55 and incorrect information will be saved.

3.3 Training

After the dataset is created by the acquisition task, the user can train the network using the dataset previously created. The training task is composed of three parts, the first part is where the dataset is preprocessed and serialized, that is, the files that make up the dataset are randomized to form the training and the validation sequence. Afterwards, the next part is the training itself, which first creates the CNN model to then pick the previously serialized samples and feed them to the network. The model used is the SSD presented in the state of the art, because as stated is the model that has the ideal balance between accuracy and speed.

The last part, is where the final model, after the training, is exported to a file with the desired output tensors, in this case, the result tensor. This file is used to test the model, in the test task.

The implementation of this part was borrowed from this repository [77], but it was modified to function according to the configurations for this case study, namely, the ground truth information, the input size of the network, the number of feature maps, the desired classes and to train with the dataset created by the user in the acquisition task.

3.3.1 Process of the Dataset

The process part builds the samples lists and randomly reorganizes them in training and validation samples, that is, it picks in a random way what are the samples that will be used to train the network and the ones that will be used to validate the network. The number of samples for the validation are chosen by the user, for example, 5% of the totality of samples.

A sample is composed of the filename of the saved frame, in the Image folder, the respective proportional coordinates of the bounding box with the class of the object, and the size, width and height, of the corresponding frame.

The absolute coordinates of the box in the Annotations file, like in figure 53, are converted into a proportional center-width coordinate, because in SSD the anchors are represented by that configuration, that is, instead of having the min-max absolute coordinates (x_{min} , y_{min} , x_{max} , y_{max}), the box is represented by its center point (cx, cy) (equations 16, 17), the width (equation 14), and the height (equation 15).

$$width = x_{max} - x_{min} \quad (14)$$

$$height = y_{max} - y_{min} \quad (15)$$

$$cx = xmin + \frac{width}{2} \quad (16)$$

$$cy = ymin + \frac{height}{2} \quad (17)$$

In addition, they are converted to a proportional representation (equations 19, 20, 21 and 22) to allow the box information to be displayed in any frame, regardless of its size.

$$width = \frac{width}{frame_width} \quad (18)$$

$$height = \frac{height}{frame_height} \quad (19)$$

$$cx = \frac{cx}{frame_width} \quad (20)$$

$$cy = \frac{cy}{frame_height} \quad (21)$$

Therefore, a sample is composed as follows.

Sample: (Path to image folder/image name, Box (class, number class, Point (cx, cy), (width, height)), image size (width, height))

This process task generates three files, one is the file with the training samples the other is the file with the validation samples, that both contain the ground truth information of each image.

The third file is composed of all the presets for the creation of network, namely, the input size of the network, the size of the feature maps, the number of anchors and the respective aspect ratios, the number of classes and the transformations that will be performed in the training samples.

The size of the feature maps and the number of anchors is dependent of the chosen input size, if the user wants to change the input size it also needs to modify the size of the feature maps and the total number of anchors according to table 8.

Table 8 - Size of the SSD feature maps and the total number of anchors according to the preset size of the input layer

<i>Input</i>	<i>Conv4_3</i>	<i>Conv7</i>	<i>Conv8_2</i>	<i>Conv9_2</i>	<i>Conv10_2</i>	<i>Conv11_2</i>	<i>Anchors</i>
300x300	38x38	19x19	10x10	5x5	3x3	1x1	8732
250x250	32x32	16x16	8x8	4x4	2x2	1x1	6132
200x200	25x25	13x13	7x7	4x4	2x2	1x1	3924
150x150	19x19	10x10	5x5	3x3	1x1		2956
100x100	13x13	7x7	4x4	2x2	1x1		1424

The feature maps presented in table 8, from Conv4_3 to Conv 11_2, are the feature maps where the detections will be performed, as shown in the SSD architecture in figure 25.

From the input until the first feature map, the Conv4_3, the input is down-sampled three times, in the Conv1_2, in the Conv2_2 and in the Conv3_3, because the network it is built on a standard vgg16 architecture, figure 16. The Conv4_3 also reduces the spatial size, however the last map of the VGG network, the Conv5_3, is modified to not perform the down sampling.

Then, from Conv4_3 to the Conv7 the size is reduced twice in the Conv4_3 and in the Conv7. Before the Conv7 the input is always down-sampled.

Then, the input size of the feature maps, if the input size is 300x300, can be represented by a chain of events, for example as represented in table 9.

Table 9 – Input size of the feature maps

Input	Feature map	Down-sample	Output
300 × 300	Conv1_2	$\frac{300}{2}$	150 × 150
150 × 150	Conv2_2	$\frac{150}{2}$	75 × 75
75 × 75	Conv3_3	$\frac{75}{2}$	38 × 38
38 × 38	Conv4_3	$\frac{38}{2}$	19 × 19
19 × 19	Conv7	$\frac{19}{2}$	10 × 10
10 × 10	Conv8_2	$\frac{10}{2}$	5 × 5
5 × 5	Conv9_2	$\frac{5}{2}$	3 × 3
3 × 3	Conv10_2	$\frac{3}{2}$	1 × 1
1 × 1	Conv11_2		

Then, the size of each feature map is what is expected for that input to be with the consecutive down-sampling, that is, the output size of the previous layer.

The total number of anchors is the sum of every possible detections in the whole model, that is, in each feature map, as stated above, is defined a certain number of aspect ratios for each location, as shown in figure 28.

For the first map and for the last two maps, Conv4_3, Conv10_2 and Conv11_2 in the original network is defined four aspect ratios, for the rest is defined six different aspect ratios.

Then, for the input size of 300x300, the total number of anchors can be calculated as follows:

$$Anchors(300 \times 300) = (38 \times 38 \times 4) + (19 \times 19 \times 6) + (10 \times 10 \times 6) + (5 \times 5 \times 6) + (3 \times 3 \times 6) + (1 \times 1 \times 6) = 8732.$$

However, if the input size is equal or lower than 150 the last map is eliminated because the input size, with the consecutive down-samples, does not reach the Conv11_2. Also, the first feature map size is considerably low and thus instead of applying four aspect ratios on the first feature map, it is applied six. The number of anchors, for the input size equal or lower than 150 is calculated as follows:

Anchors(150 × 150)

$$= (19 \times 19 \times 6) + (10 \times 10 \times 6) + (5 \times 5 \times 6) + (3 \times 3 \times 4) + (1 \times 1 \times 4) = 2956$$

The transformations are used to perform data augmentation, which, as mentioned above, is an important task that allows the network to be more robust, by impressing modifications on the original frame. The dataset process creates a list of the various transformations that will be performed in every training sample. The list is composed of the following transformations:

- Brightness modification;
- Distortions (Contrast, Hue, Saturation);
- Reorder channels;
- Expand;
- Sampling;
- Horizontal flip;
- Resize.

These transformations will be performed before the sample is feed to the network in the training part, in a successive manner. The resize is always performed, because it is where the samples are resized to the desired input size, all other transformations can or cannot be performed, that is, they have a probability of being executed, then, for example the transformations for a specific training sample can be represented as follows:

1. Contrast Distortion;
2. Reorder channels;
3. Horizontal flip;
4. Resize.

On the other hand, the only transformation that is performed on the validation samples is the resize transformation, because the validation is not used to train the network but is used verify the performance of the network and therefore they do not need to be augmented.

3.3.2 Train

The training part builds the CNN model and picks the previously processed samples to feed them to the network to train the network.

Before the training, the SSD network is built according to the presets previously dimensioned in the process part, that is, it creates all the variables, it builds the network layers including the VGG, it creates the SSD feature maps from the presets with the respective anchors, it creates the loss functions, it builds the classifiers for each feature map and it creates the optimizer.

As for the loss function, it is used the one that was defined by the SSD authors which consists on a weighted sum of the localization loss and the confidence loss as represented in equation 23 [53],

$$L(x, c, l, g) = \frac{1}{N} (l_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \quad (22)$$

where N represents the number of the matched default boxes.

The localization loss is a smooth L1 loss [26] in relation to the predicted box (l) and the ground truth box (g). The confidence loss consists on a softmax cross entropy loss [78] over the multiples classes confidences (c).

The used classifier is a softmax [78], and regarding the optimizer, used to optimize the loss function described above, it is used the Adam optimizer [27].

After the model graph is built, the training moves to next stage, where it picks the training samples and trains the network. After the all the training samples are passed through the network and after completing all the training processes, the network is validated by passing the validation samples. The number of times this cycle is performed is chosen by the user, by defining the number of epochs.

The network is trained using the mini-batch gradient descent method [27] in which the network performs the weight update for every batch composed of a set of training samples.

The number of times that the network performs the weight update in every epoch is defined by the number of batches (equation 23).

$$N^{\circ} \text{ Batches} = \frac{N^{\circ} \text{ Training samples}}{\text{Batch size}} \quad (23)$$

Hence, there are two parts that represents an epoch, the first part is when the network is being trained with the training samples, the second and last part is when the network is being validated by the validation samples.

The training cycle first extracts the training samples from the training samples file and randomly shuffles them to print even further randomness in the samples and processes them, obtaining the

relevant information from the samples, namely, the image frame, the label of the object in the frame and the ground truth box. It also picks what are the transformations to perform at each image before inputting them into the network.

When the training batches are being feed to network, to avoid collision of data, it is created two queues, one from the samples to fill the batches and one to the batches to feed the network, more precisely, the processed samples are placed in the sample queue to feed the batches, then the batches are placed in the batch queue to be inputted to the network. The production of this batches is being performed in parallel, using the processing units (CPU or GPU), that is, the number of parallelisms is defined by the number of cores of the processing unit. Henceforth, the network receives the batch that is placed first in the queue, until there are no more batches, that is, no more samples.

Every time a batch of training samples is feed to the network, besides from updating the weights, the network also outputs the result tensor and loss value for every sample in the batch.

The result is composed of all the predictions, per class, that the network predicted for each sample to be later decoded into detections. The detection is defined as follows:

Detection: (Confidence, Box (class, id_class, center, Size(width, height)))

Therefore, every time a batch passes through the network in the training, the results are decoded for each sample in the batch and the box of the detection and the ground truth of the correspondent samples are saved in a list for later, at the end of the epoch, be used to calculate the average precision. After the training samples are finished, the training passes to the validation cycle, that performs with the same workflow as the training cycle presented above, however, the weights are not updated, that is, the network is not trained, because the validation cycle is used to verify the results of the network with data that wasn't trained with.

Then, after the validation samples are finished the algorithm picks the detection lists of both training and validation and calculates the average precision for each class for the training samples and for the validation samples and writes it in the *Tensorboard*.

This sequence of processes is performed in every epoch defined by the user and when the training is completed, the AP curves of each class is represented in the *Tensorboard* as shown in figure 56 and 57.

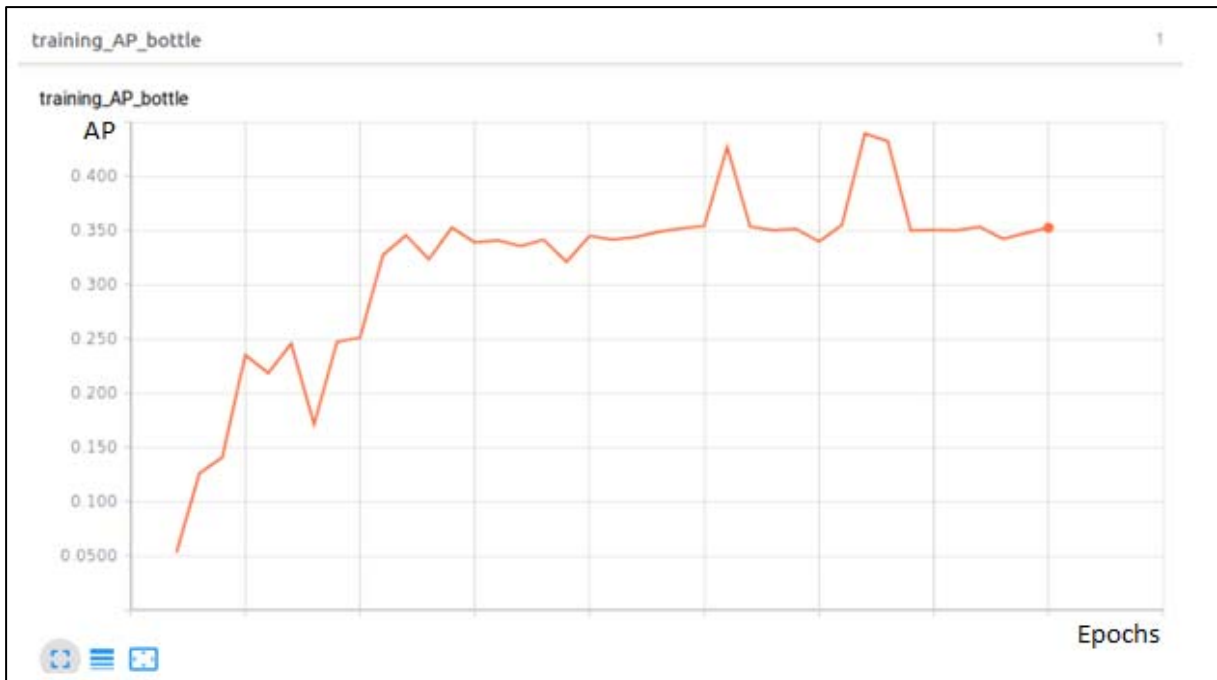


Figure 56 - Training AP precision curve on Tensorboard

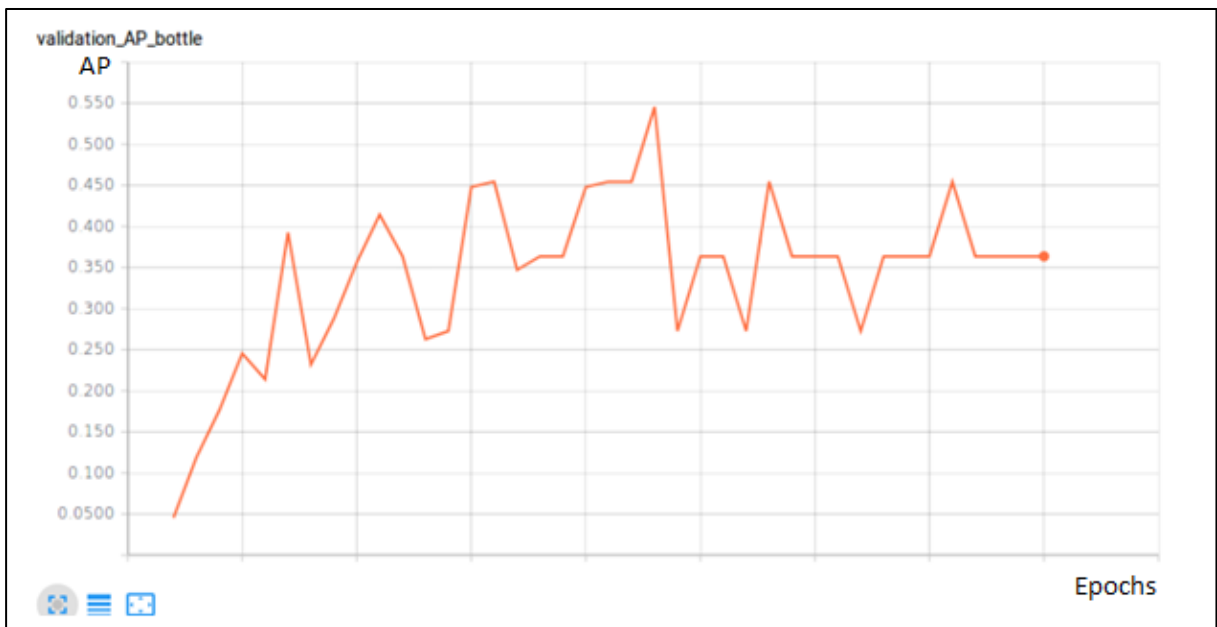


Figure 57 - Validation AP precision curve on Tensorboard

These responses will be used to later verify the performance of the network, in the Results chapter.

3.3.3 Export

At the end of the training it is generated a set of files that contain the trained model, with the respective trained variables, what the export does is that it allows the extraction of the model with the desired

output for the testing task, by generating a model file. The output, in this case, is the result tensor, then in testing, when it is inputted an image to the network, it will output the result.

To train the network, it was created a batch file that performs the three parts sequentially, that is, it first processes the dataset, then it trains the network and at last it exports the model to be tested. The batch file is named main and is executed like in figure 58.

```

ttagop@lar:~/Codes/Tese$ nohup ./main.sh tnew2_data9_e50_b4_150_lr5_lar data vgg_150 50 4 tbnw2_data9_e50_b4_150_lr5_lar tnew2_data9_e50_b4_150_lr5_lar > out38.log 2>&1 &
[1] 5825
ttagop@lar:~/Codes/Tese$ tail -f out38.log
nohup: ignoring input
Dataset: 100%|#####| 3183/3183 [00:17<00:00, 182.19samples/s]
({{}} data directory:      'data')
({{}} Validation fraction:  '0.85')
({{}} Preset:              'vgg_150')
[{}] Configuring the data source...
({{}} # training samples:  '2948')
({{}} # validation samples: '155')
({{}} # testing samples:   '0')
({{}} # classes:          '2')
DATASET PROCESSED
[{}] Epoch 1/50: 0%| | 0/737 [00:00<?, ?batches/s]({{}} Project name:      'tests/tnew2_data9_e50_b4_150_lr5_lar')
({{}} Data directory:      'data')
({{}} VGG directory:      'vgg_graph')
({{}} # epochs:            '50')
({{}} Batch size:         '4')
({{}} Tensorboard directory: 'tensorboard/tbnw2_data9_e50_b4_150_lr5_lar')
({{}} Checkpoint interval:  '10')
({{}} Learning rate:       '0.005')
({{}} Learning rate decay: '0.97')
[{}] Creating directory tests/tnew2_data9_e50_b4_150_lr5_lar...
({{}} Starting at epoch:   '1')
[{}] Configuring the training data...
({{}} # training samples:  '2948')
({{}} # validation samples: '155')
({{}} # classes:          '2')
({{}} Image size:         'Size(w=150, h=150)')
[{}] Creating the model...
[{}] Training...
[{}] Epoch 1/50: 1%| | 4/737 [00:31<1:35:25, 7.81s/batches]

```

Figure 58 - Train the network

The main batch expects seven arguments, that the user needs to put to define some parameters, which are:

1. The name of the training;
2. The dataset directory;
3. The preset, which is chosen accordingly to the desired input;
4. The number of epochs;
5. The size of the batch;
6. The name of the tensorboard folder to save the tensorboard files;
7. The name of the output model.

Once the batch file is completed, the network can be tested.

3.4 Test

The test part, as stated above, is the main part of the model framework, because it is where, after the network is trained, the frames from the camera are being successively processed by the network which provides the bounding box, the object name and the confidence of the presented object.

Before the video cycle, the test algorithm loads the model from the file exported in the export part and picks the preset data for the anchors.

After the frame goes through the network it is obtained the result tensor and afterwards, the result is decoded, as presented above. Then at every detection is performed the NMS (non-maximum suppression) to suppress the overlaps. The NMS picks the box with the biggest confidence and calculates the IoU of the other boxes in relation to that box. If the calculated IoU is bigger than a threshold it is discarded. This eliminates boxes that are close to each other, as proved in the figures 59, 60, 61 and 62.

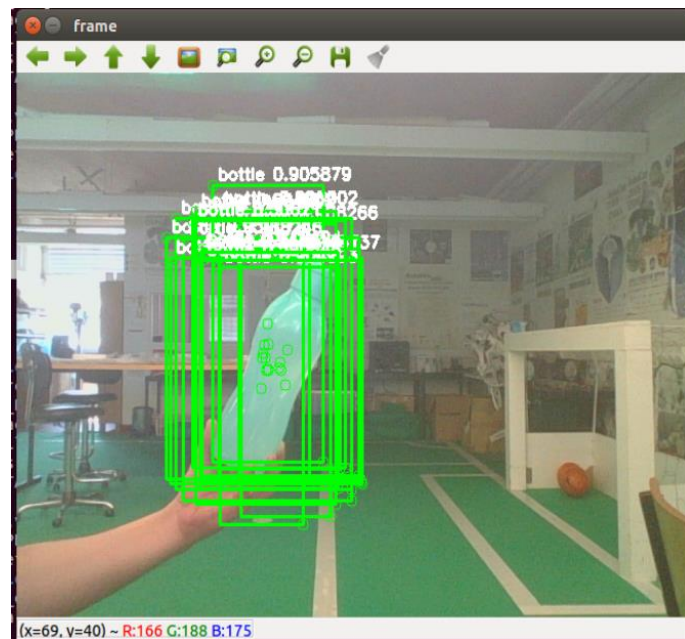


Figure 59 - Test frame with no NMS

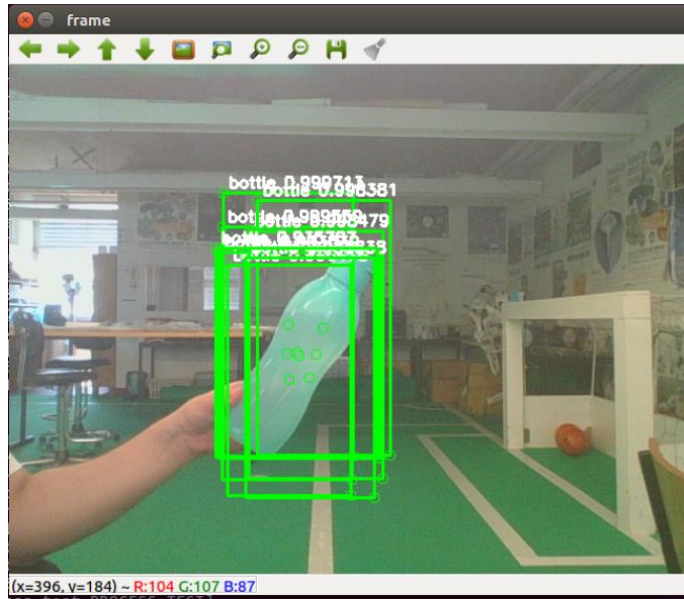


Figure 60 - Test frame using NMS to suppress overlaps that are over 0.75

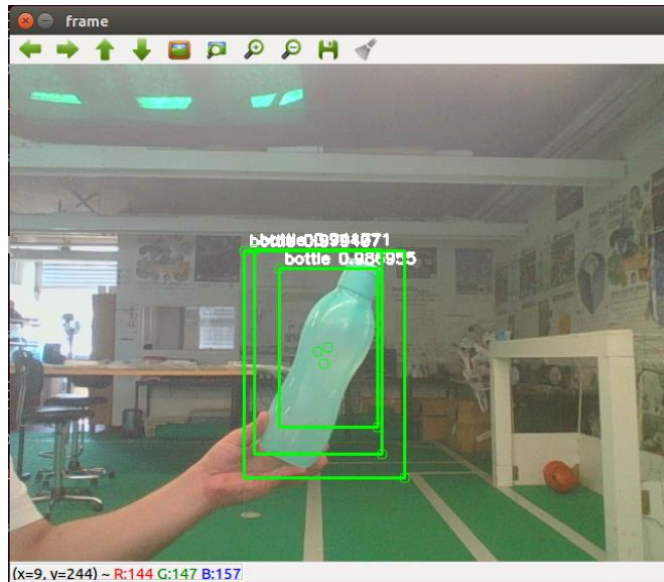


Figure 61 - Test frame using NMS to suppress overlaps that are over 0.5

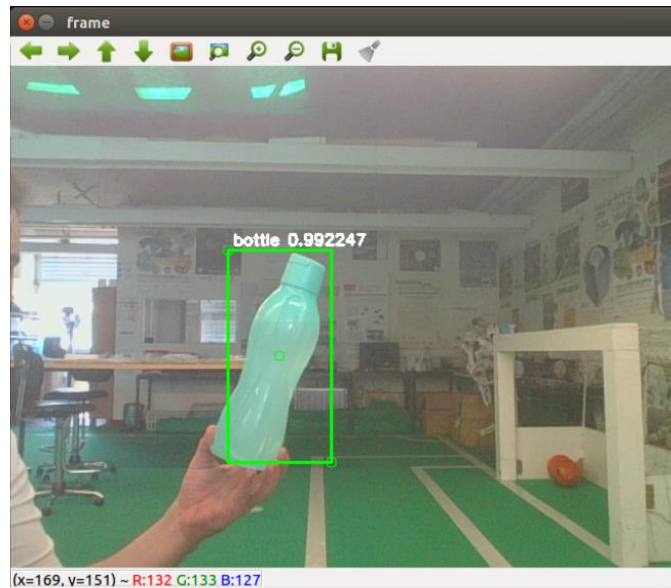


Figure 62 - Test frame using NMS to suppress overlaps that are over 0.1

As shown in figure 59, if the detections are not suppressed with the NMS, every detection, with a confidence value above 0.5, will be decoded and displayed on the frame and therefore there will be a lot information to be processed for just one object, although, if the NMS is applied (figures 60, 61 and 62) the boxes will be suppressed.

After the decoding and the suppression of the boxes, every box is displayed on the window with respective confidence and class as represented in figure 62.

4. RESULTS

This chapter presents the many tests executed on the CNN network that were carried out to choose the best parameters that represents the best results in terms of speed and precision, i.e. the input size, the learning rate and the dataset.

It will also demonstrate the results obtained by training and testing the final prototype.

4.1 Frame rate

The first tests performed were to choose the range of the input size that represents the acceptable range in terms of frame rate.

The original network, from [77], consists on an SSD network that expects inputs of 300x300 (SSD300) or 512x512 (SSD512), however, since it is used a CPU, with 4 cores, for all the processing, due to the impossibility of using AMD GPUs on Tensorflow, it is not possible to achieve real-time with that input as shown in figure 63.

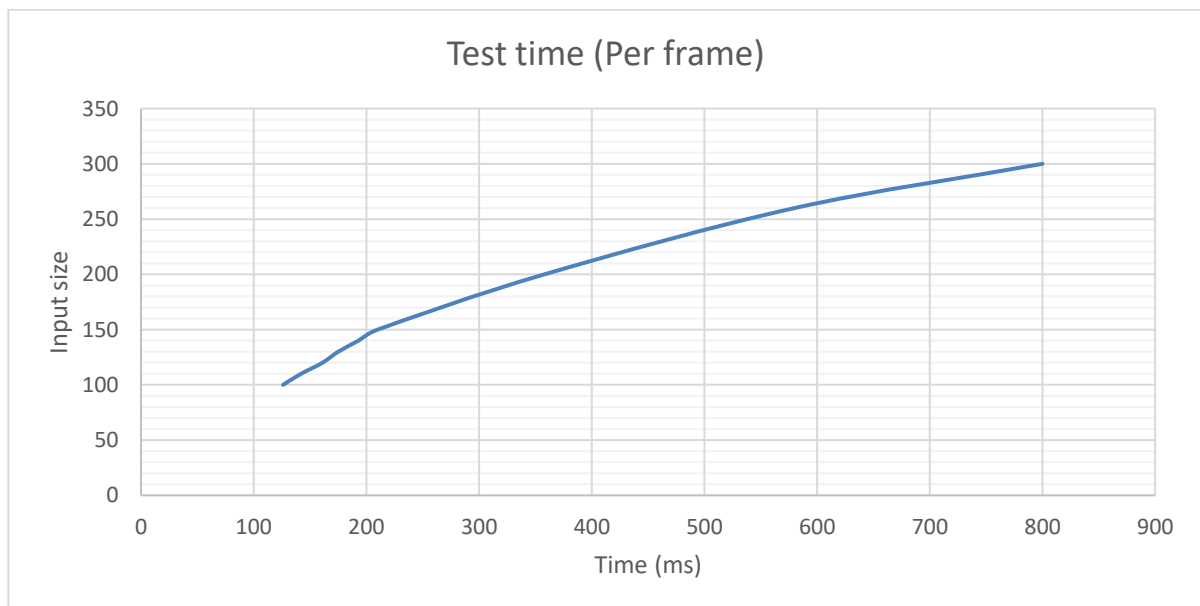


Figure 63 - Frame rate

The frame rate increases as the input size is bigger, being that with the size of 300x300 the input frame takes 800ms, almost 1FPS, to cross all the network. Hence, with higher resolution frames the frame rate will be even lower (<1FPS).

Still, 800ms is too high to be considered real-time, the same is considered up to 200ms (5FPS), correspondent to, approximately, the input of 150x150. Then, the inputs from 150x150 to 100x100 will

be used to train and test the network, due to the higher frame rate in test, approximately 5FPS and 10FPS on 150x150 and 100x100 respectively. Inputs smaller than 100x100 are low resolution frames, so they are not suitable for the problem because it is necessary to detect real objects, since the lower resolution inputs will not be able to properly extract the object features.

4.2 Data and parameter selection

After choosing the appropriate range for the size of the inputs, the network is trained with a given dataset to verify the performance of the network by comparing the training and validation AP values. The network was trained several times with different parameters and datasets to understand the changes in the performance of the network to the various data and parameters modifications, for that, in every training it is obtained the curves of the AP of both training and validation samples being that, the curves are composed of the AP values at the end of every epoch. In addition, the last values of AP are extracted, that is, the values correspondent to the last epoch, which represent the final AP of the network.

The first dataset created (Data1) is composed of 586 samples of only bottles, with respectively 5 types of bottles represented on figure 64.



Figure 64 – Types of Bottles in the dataset

Note that 5% of the total samples are used to validate the network, therefore, it is used 557 samples to train the network and the other 29 are the validation set.

The following parameters were initially defined to train the first network with Data1:

- Learning rate = 0.003;
- Input = 100x100;
- Suppress overlaps above 0.45;
- Batch size = 4.

The network was then trained four times with different number of epochs and the results were obtained, being that, the figures 65 and 66 represents the training and validation precision curves respectively, and table 10 shows the last AP values of both training and validation.

Table 10 - Results on Data1, with different number of epochs

	<i>Train AP</i>	<i>Validation AP</i>	<i>Training Time</i>
<i>20 epochs</i>	0,4268	0,3636	0d 2h 54min 43s
<i>30 epochs</i>	0,3475	0,4416	0d 3h 50min 54s
<i>40 epochs</i>	0,3526	0,3636	0d 5h 11min 12s
<i>50 epochs</i>	0,3584	0,2614	0d 6h 29min 34s

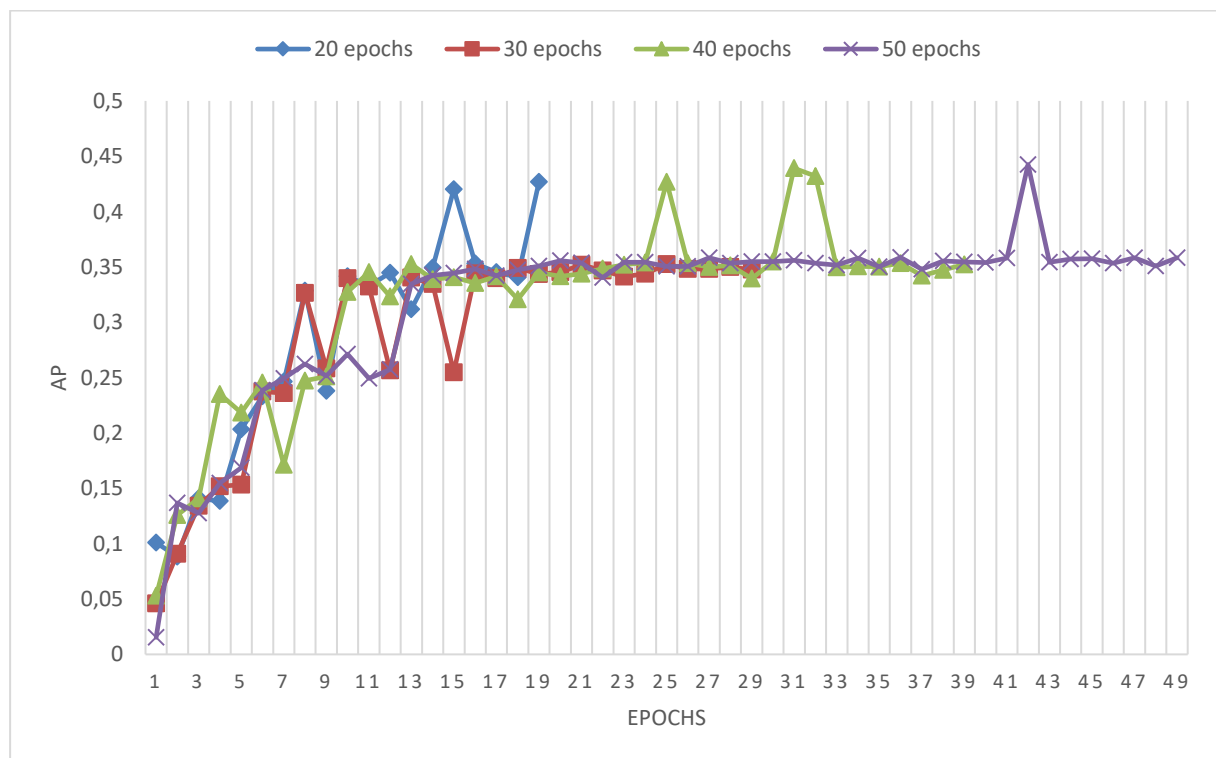


Figure 65 - Data1 training precision curves

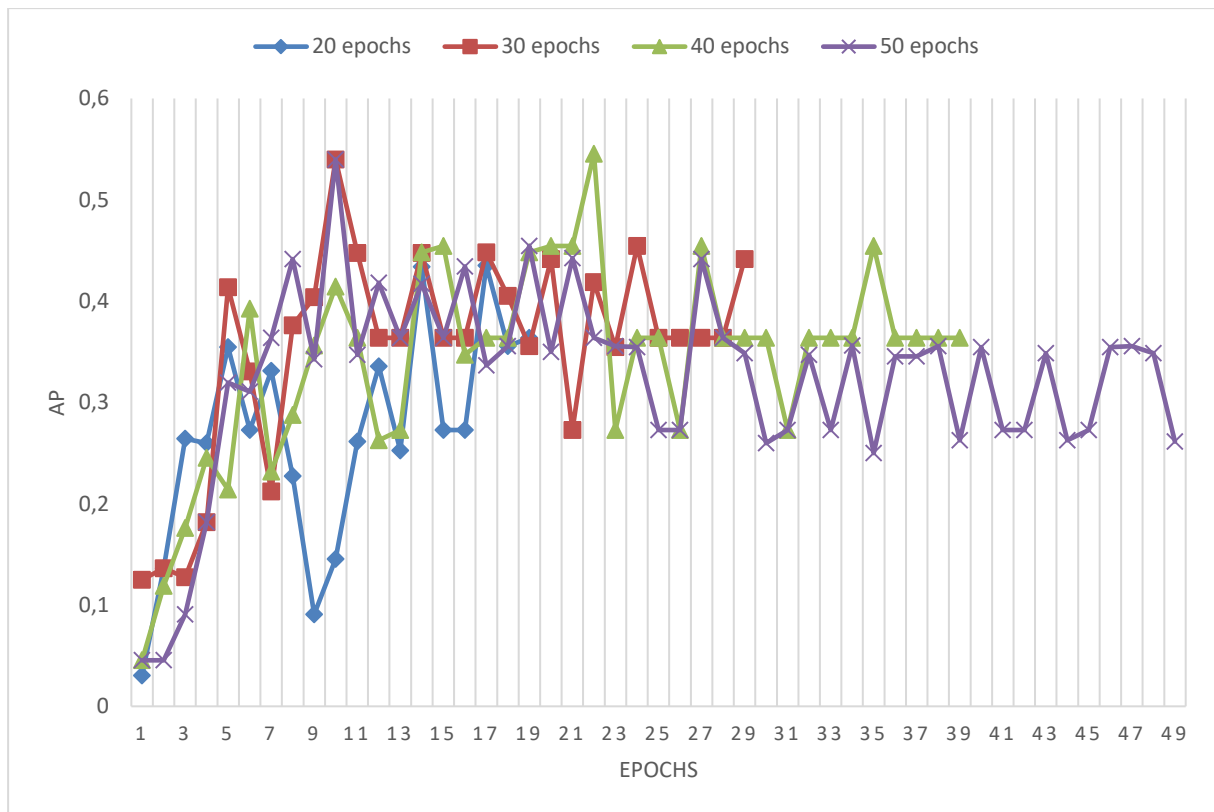


Figure 66 - Data1 validation precision curves

As presented in table 10, the network, on this dataset, performs poorly because in almost every result obtained the training AP has a higher or nearly equal precision value than the validation ones, except for the result with 30 epochs. In addition, it is possible to see, comparing the curves in figures 65 and 66, that the network is more stable in the training.

This occurrence, when the network performs better on the training data, is called overfitting, where the network learns the “data” instead of finding the best relationship between the inputs and the outputs that generalizes to all type of data.

Since the validation set is not used to train, is considered “new” data, the network is supposed to have higher values on the validation samples.

The result obtained with 30 epochs is a case where it was found a good configuration of training and validation samples that does not overfit the network. That happens because, before every training it is executed the processing task, as presented above, in which all samples in the dataset are randomly reorganized into training and validation samples and it is chosen the transformation for each training sample, then the processed samples were arranged in way that the network does not overfit.

However, if the network was trained again with 30 epochs the network will, most likely, overfit by following the other results.

With these results obtained either the dataset contains few data for the quantity and complexity of the objects presented in the dataset or the chosen parameters are not the most appropriate. Yet, since there is no evolution, as the number of epochs increases, as shown in figure 66, in the validation precision, only random changes, the cause of these poor results can be due to the low amount of data in the data set.

Hence, it was added new data to the Data1, now called Data2. The Data 2 is now composed of 1185 samples, 1126 training and 59 validation samples.

The tests performed was the same as the ones performed in Data 1, with the same parameters, the results are presented in figures 67 and 68 displaying the training and validation precision curves respectively and on table 11 that shows the network precision values.

Table 11 - Results on Data2, with different number of epochs

	<i>Train AP</i>	<i>Validation AP</i>	<i>Training Time</i>
<i>20 epochs</i>	0,3479	0,1818	0d 4h 58min 57s
<i>30 epochs</i>	0,3480	0,2587	0d 7h 44min 29s
<i>40 epochs</i>	0,3414	0,3636	0d 11h 19min 28s
<i>50 epochs</i>	0,3493	0,3593	0d 14h 13min 55s

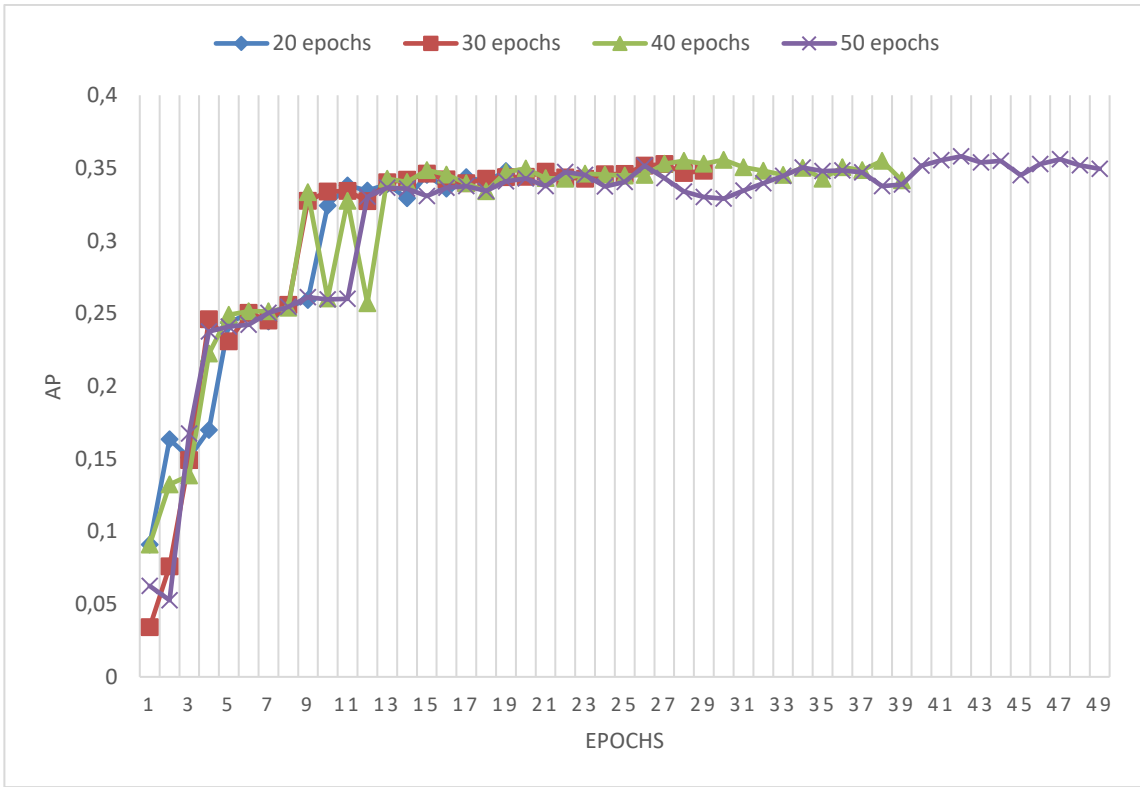


Figure 67 - Data2 training precision curves

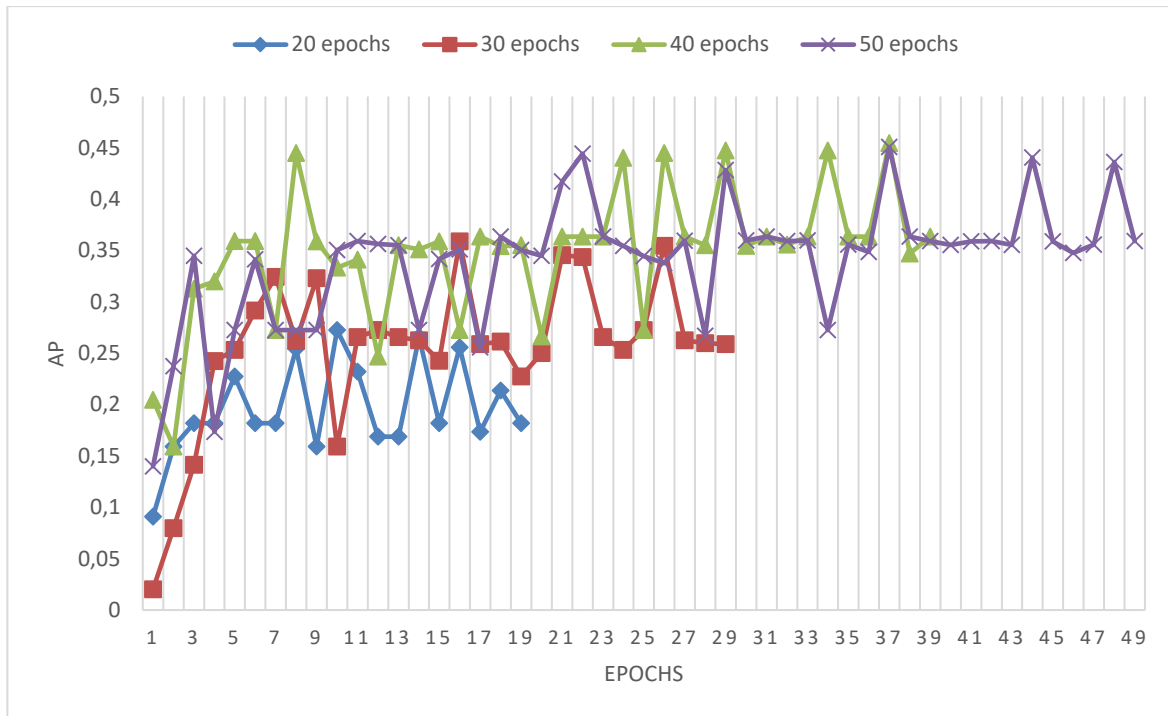


Figure 68 - Data2 validation precision curves

The results presented in the table 11 shows that the increasing of data was not enough to the improvement of the network performance.

By comparing the curves in figure 65 and 67, it is possible to see that the increasing of data stabilized the training curves even further, with all curves in figure 67 reaching an approximately equal value, as also shown in table 11 training AP values. However, in validation on figure 68 similarly to figure 66, the network performance is poor, since the curves do not represent a consistent and stable evolution, and thus, as shown on table 11, the validation AP values are lower or approximately equal to the training AP values, which is not desired.

This means that the parameters are not suitable for the network to learn the features of the object represented in the dataset, since the performance of the network didn't quiet change with the increasing of data, consequently the network could not learn more features than the ones in the tests on table 8, which would represent on higher results of validation AP.

Therefore, the following results, figures 69, 70 and table 12, were obtained by training the network with the same dataset previously used (Data2) with a different input size corresponding to the maximum input previously stipulated, that is, 150x150.

Table 12 - Results on Data2, with different number of epochs, with 150x150 as the input size

	Train AP	Validation AP	Training Time
20 epochs	0,5292	0,8926	0d 7h 16min 16s
30 epochs	0,5368	0,8182	0d 11h 9min 10s
40 epochs	0,5392	0,8024	0d 20h 16min 15s
50 epochs	0,6224	0,8182	1d 1h 38min 19s

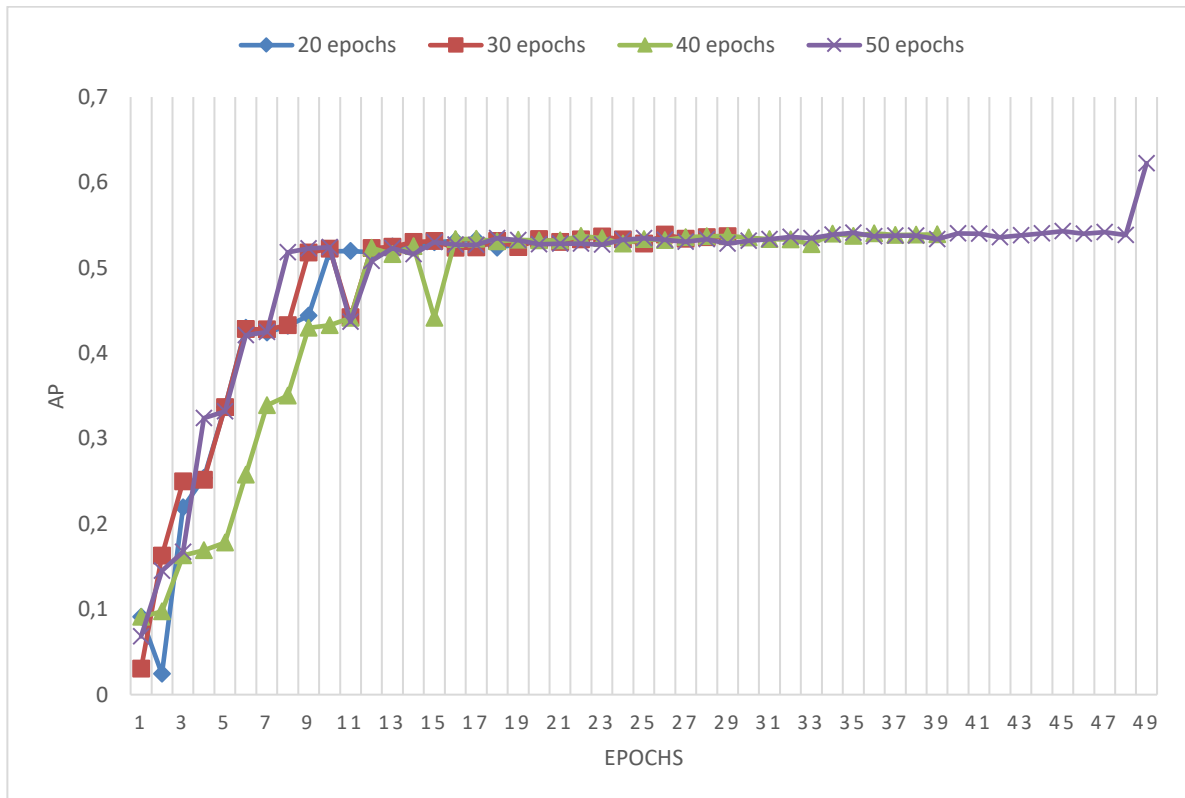


Figure 69 - Data2 training precision curves, with 150x150 as the input size

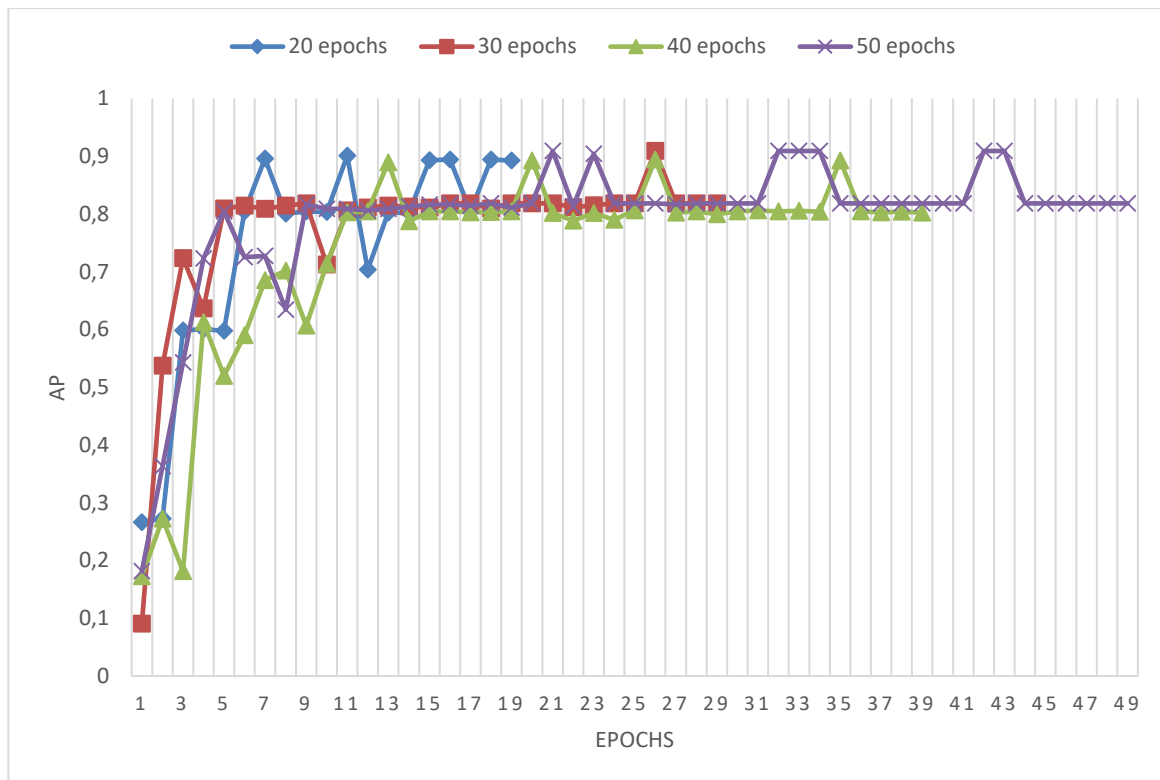


Figure 70 - Data2 validation precision curves, with 150x150 as the input size

The increasing of the input size from 100x100 to 150x150 allowed, as shown in the table 12 and on the figures 69 and 70, the enhancement of the network in detecting the objects represented by the increasing of all the AP values and the drastic change in the validation curves as shown in figure 70.

The most visible and important change was the large increase in the validation AP values from 15-35% to 80-90% of AP in the validation samples which, as stated above, represents the performance of the network in detecting objects on new data.

This sharp change in the network performance is due to the ability of the network to now detect more and better features that represents the bottle object, in comparison to the previously results, which was allowed by the increasing of the input size.

So, the input size provides a high increase in the network performance, however there is a parameter that needs to be taken into account, the learning rate. The learning rate may not influence directly the precision of the network, although, it can affect the behavior of the network by significantly modifying the training and validation curves, since the learning rate defines the size of the weight update, that is, the speed at which the network learns the features.

Therefore, the network was trained with different learning rates to check the influence of the learning rate parameter on the network performance.

The learning rates used are:

- lr1=0,001;
- lr3=0,003;
- lr5=0,005;
- lr10=0,01.

The results on Table 13 and figure 71 were obtained by training the network with different learning rates using the same parameters and dataset previously used, with only 20 epochs being used to train the network because it is easier to visualize the learning speed.

Table 13 - Results on Data2, with different learning rates, with 20 epochs and 150x150 as the input size

	<i>Validation AP</i>
<i>lr1</i>	0,7273
<i>lr3</i>	0,8926
<i>lr5</i>	0,8182
<i>lr10</i>	0,7835

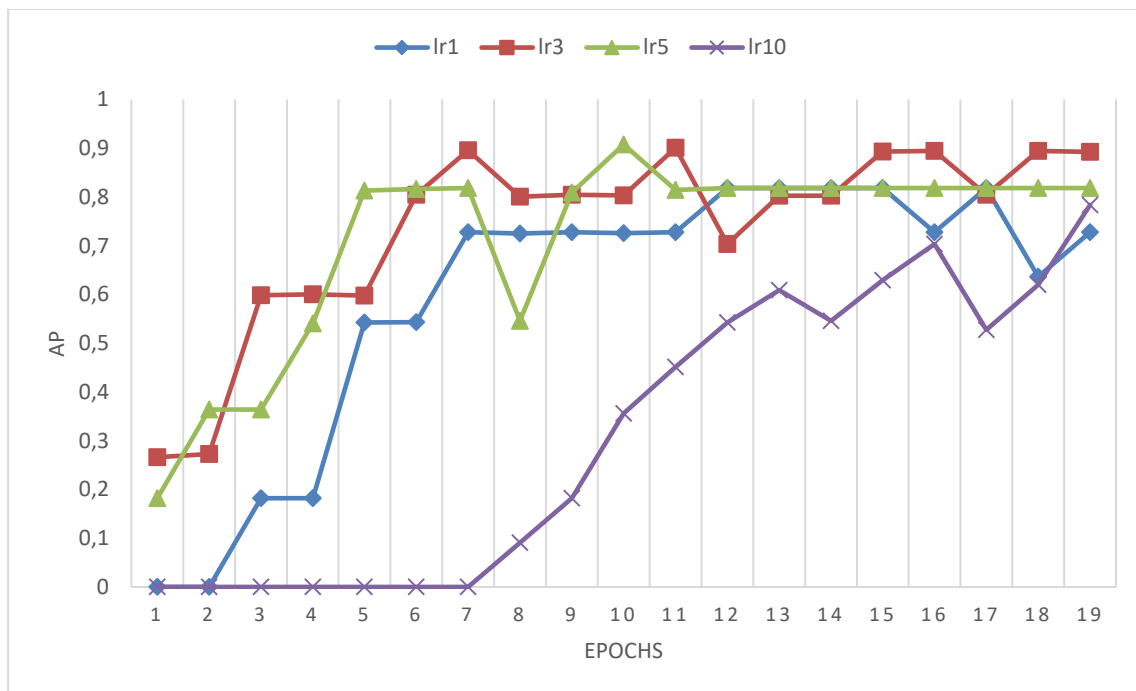


Figure 71 - Data2 validation precision curves, with different learning rates, with 20 epochs and 150x150 as the input size

At first sight, it is possible to see that the results obtained in the learning rates lr3 and lr5 were the best ones, because, it is the result that have the higher validation AP, also, on figure 71, they are the curves that has the best response, that is, with a more consistent and faster response to reach the stable

point. However, the curve of lr5 is the fastest to stabilize, as shown on figure 71, so the lr5 is the most appropriate parameter to use.

The curve of lr3, even though it ends at a higher precision value, which is optimal, it is still not on a stable point, that is, it is still learning unlike the lr5 curve that from the 11th curve the network stopped the learning because it always maintained the same AP value. This may prove to be worst with higher datasets, because with more data the network must learn more features, and thus, the network could take much time to learn them.

On the lr1 curve, it is verified that the learning rate is lower because it takes more time to increase the precision value at every epoch, in addition, in the first two values the precision is at 0, that happens because the weight update is so lower that it does not change enough to cause an increase in the AP.

On the other hand, the curve of lr10 is the other extreme as it is shown that the curve moves faster than all the others, although, it takes 7 epochs to move from the point of 0 precision, that is, the change on the weights is so high that the network instead of converging it diverges from the minimum error point. Yet, when the network starts to converge the climb is quite steep.

The usage of a high learning rate is not the most adequate, because the precision of the network will often oscillate at the point of stability, using higher number of epochs. The lower learning rate can be used, but the network will take much time to learn the features, it is needed to use a bigger number of epochs.

Henceforth, the parameters that will be used are:

- Learning rate = 0,005;
- Input size = 150x150;
- Suppress overlaps over 0.45;
- Batch size = 4.

Hence, to check the performance of the network with new parameters it was added more data to the Data2, now called Data3. The Data3 is composed of only bottles, the same bottles as the ones in Data1 and 2, but with 2104 bottle images, that is, 1999 training and 105 validation images.

The table 14 and figures 72 and 73 are the results obtained by training the network with Data3 and with the parameters previously stated.

Table 14 – Results on Data3, with different number of epochs

	Train AP	Validation AP	Training Time
20 epochs	0,3553	0,8037	0d 16h 42min 20s
30 epochs	0,5178	0,9052	1d 1h 59min34s
40 epochs	0,5371	0,9044	1d 11h 17min 15s
50 epochs	0,5324	0,9016	1d 20h 32min 35s

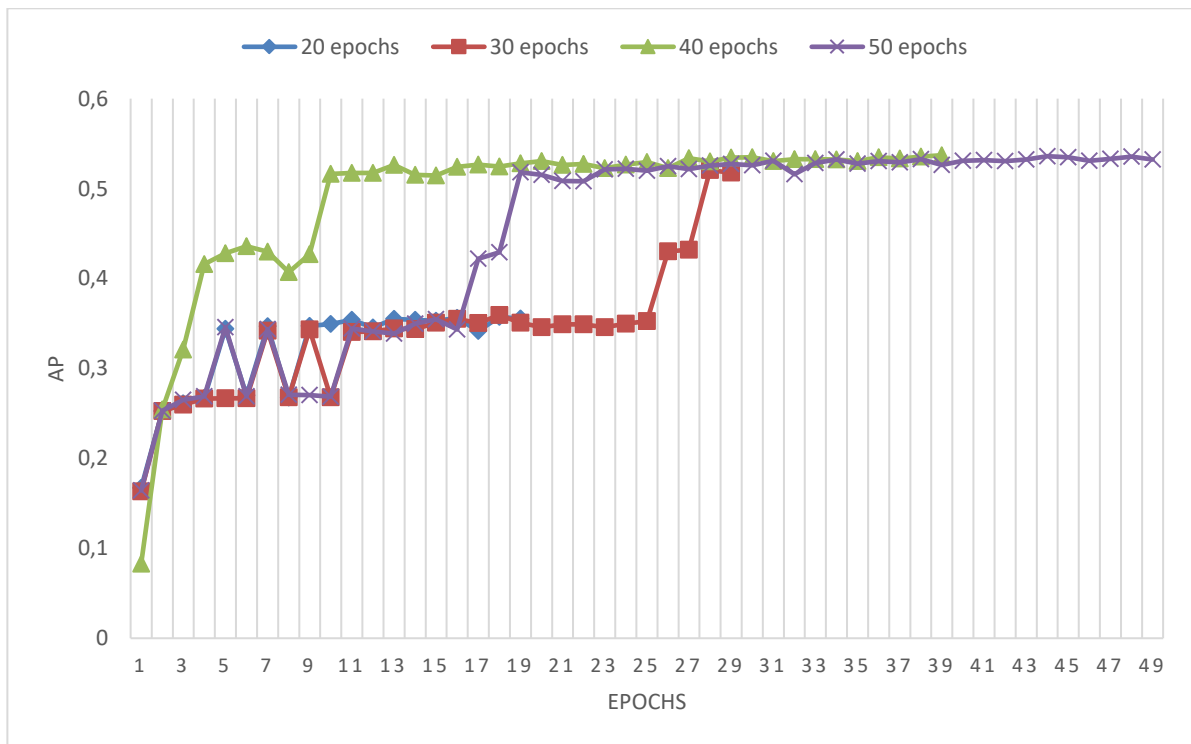


Figure 72 - Data3 training precision curves

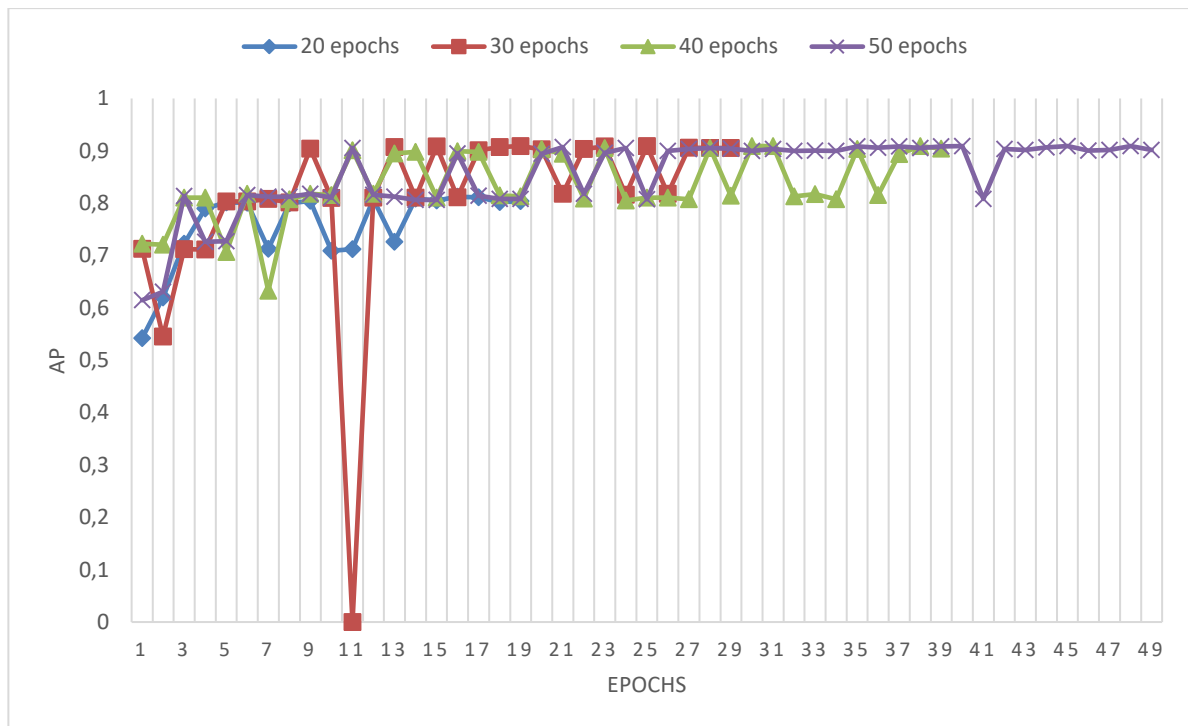


Figure 73 - Data3 validation precision curves

The results obtained, presented on table 14 and in both figures 72 and 73, shows that the network performs well on detecting bottles, with a validation precision rate between 80-90%, even with more data, which means that the parameters are appropriate to this problem and the network was able to learn the features of the class that allows its detection.

Also, on figure 73, it is possible to see that the network reaches the stable point in a fastest way due to the new learning rate used, since all the curves are near to each other.

In the 30-epoch curve, on figure 73, an unexpected value happened on the eleventh epoch. This downfall occurred because the network diverged in drastic way from the convergence point that may have been caused by some miscalculation of the gradient, which along with the learning rate caused a great drop on the average precision. However, since the network in the next epoch goes to normality this drop is not relevant in the network behavior.

As the network have a high performance in the validation set as shown in figure 73, in order to check the performance in the real application, that is, the quality of the object detection when the frames are being acquired by the network in real-time, the previously trained network, with 50 epochs, was tested. The figure 74 displays the detection of the object when it is being grabbed by the user, in figure 75 is shown the network detecting various bottles in a different perspective, at last in figure 76 is demonstrated a bad detection.

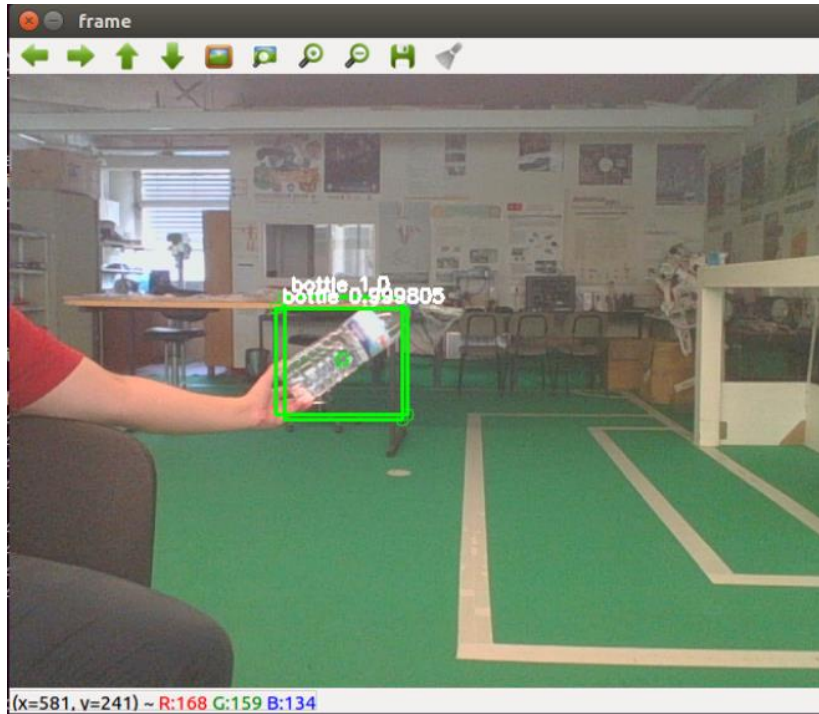


Figure 74 – Detection when the bottle is being grabbed

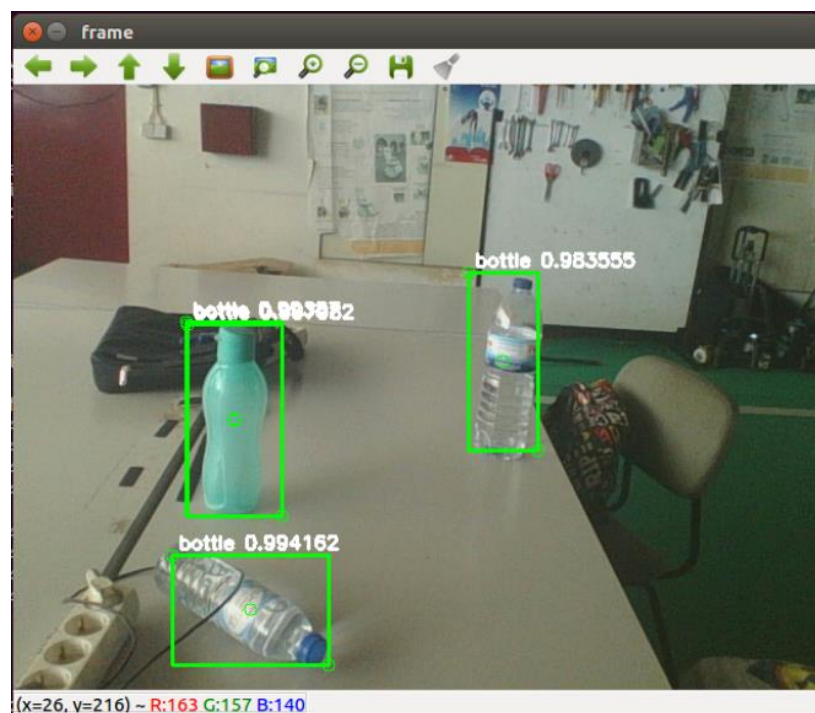


Figure 75- - Detection of various bottles in other perspective

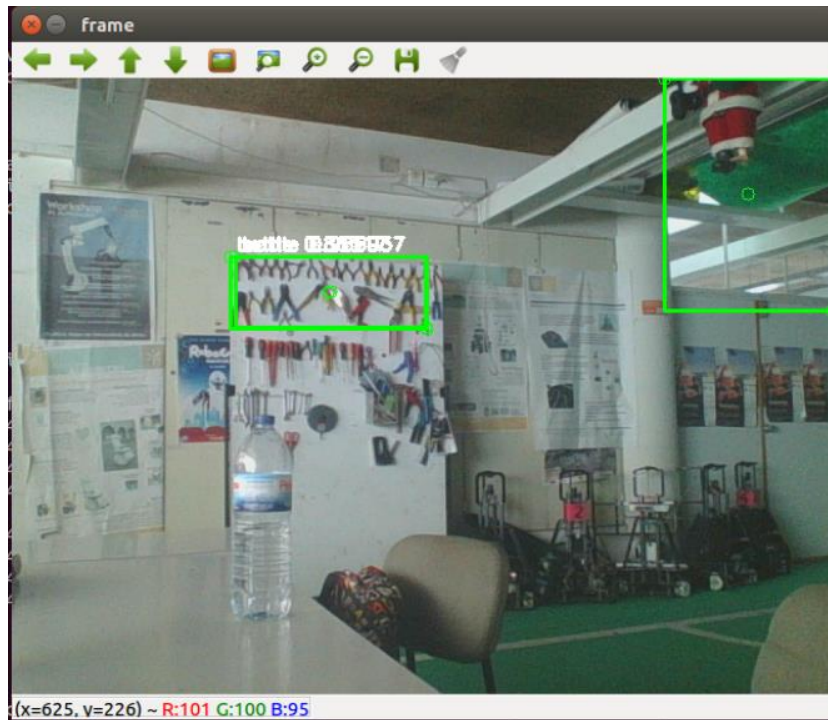


Figure 76 – Bad bottle detection

The result presented in figure 74 demonstrates the network precision on detecting bottles, where the bounding boxes almost perfectly surrounds the bottle. As for the detections in figure 75 it shows the capability of the network to detect various bottles in the same frames and that it can “see” them in a different perspective that it was trained with.

The detections presented in figure 76 allows to verify that even if the network has a high precision value in the validation and that it is able to detect very well the desired object, as proved in figures 74 and 75, the network can still have some misunderstandings in the detections. These unintended detections happen because the network only knows what a bottle is, that is, every object, besides the bottle, that has features that are the same or similar to the features that the network extracted from training, the network considers them as a bottle, as happened in figure 76.

Yet, the network is still detecting one class, which makes it easier to detect since the network only learns the features of a class in the whole network, that is, it does not need to distinguish between features and which one corresponds to a class.

Therefore, it was added another class to the network, the can class. The can class is composed of two different objects, represented in figure 77.



Figure 77 – Types of cans in the dataset

As shown in the figure 77, this new can class is less complex than the bottle one, since there are only two types of cans, however, they have some similarities to the bottle class, which is the cylindrical form. The network is, now, supposed to perform the distinction between these two classes.

Then, new data were added to the Data3 which corresponds to the addition of the new class, now called Data4. The Data4 is composed of 2818 images, which 2104 are bottle samples and the other 714 are can samples.

In table 15 is demonstrated the AP values of both classes, in figures 78 and 80 is represented the bottle training and validation curves respectively and in figure 79 and 81 is shown the can class curves, of both training and validation.

Table 15 - Results on Data4, with different number of epochs

	<i>Train AP bottle</i>	<i>Train AP can</i>	<i>Val AP bottle</i>	<i>Val AP can</i>	<i>Training time</i>
<i>20 epochs</i>	0,5004	0,2501	0,8182	0,6294	0d 22h 35min 50s
<i>30 epochs</i>	0,5284	0,3512	0,9082	0,5455	1d 11h 10min 42s
<i>40 epochs</i>	0,5269	0,3434	0,9091	0,6107	1d 23h 49min 2s
<i>50 epochs</i>	0,5312	0,3506	0,9055	0,7273	2d 12h 21min 3s

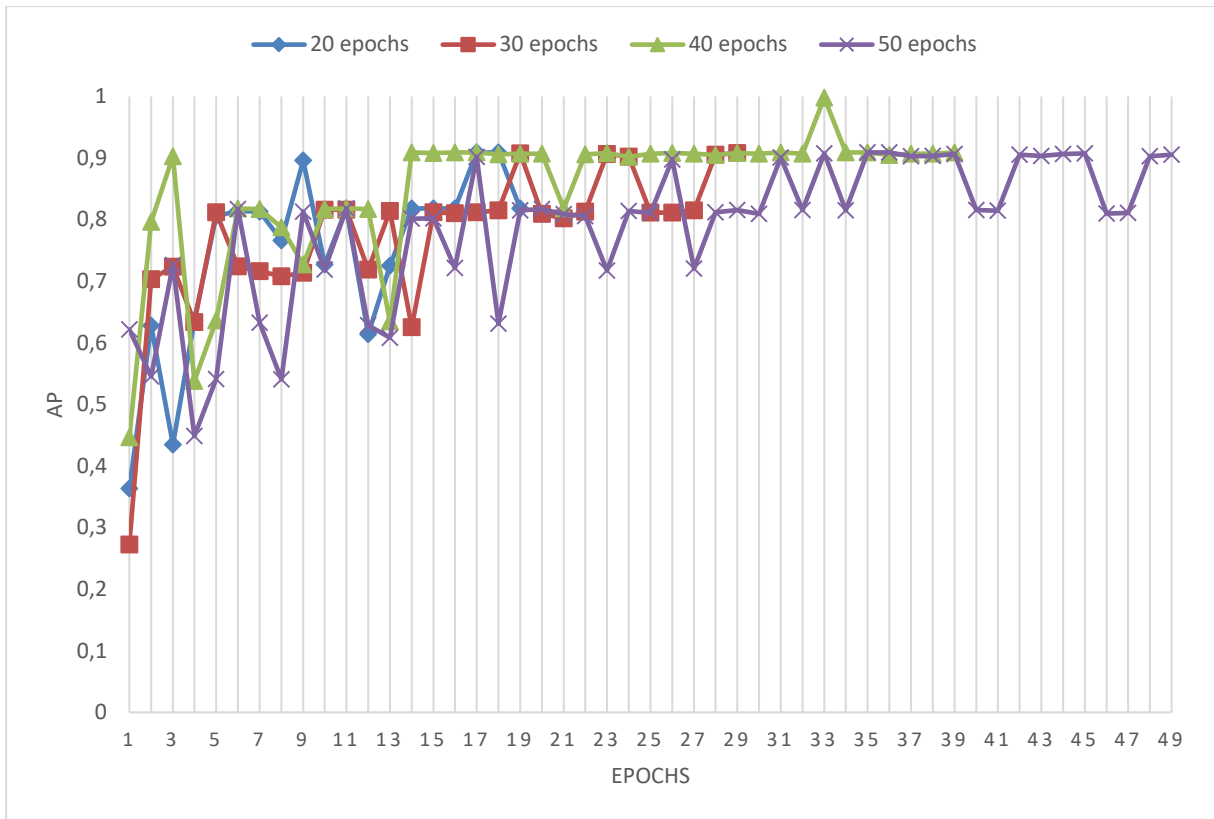


Figure 78 - Training AP curves of the bottle class

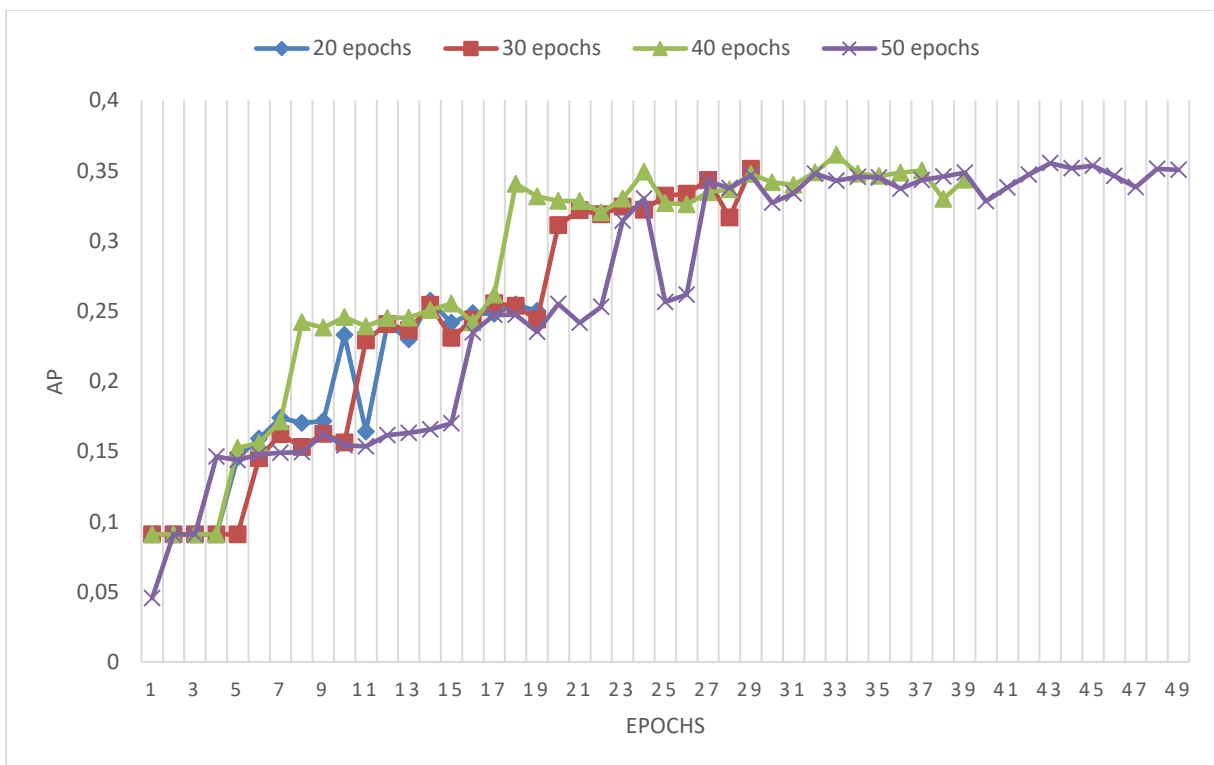


Figure 79 - Training AP curves of the can class

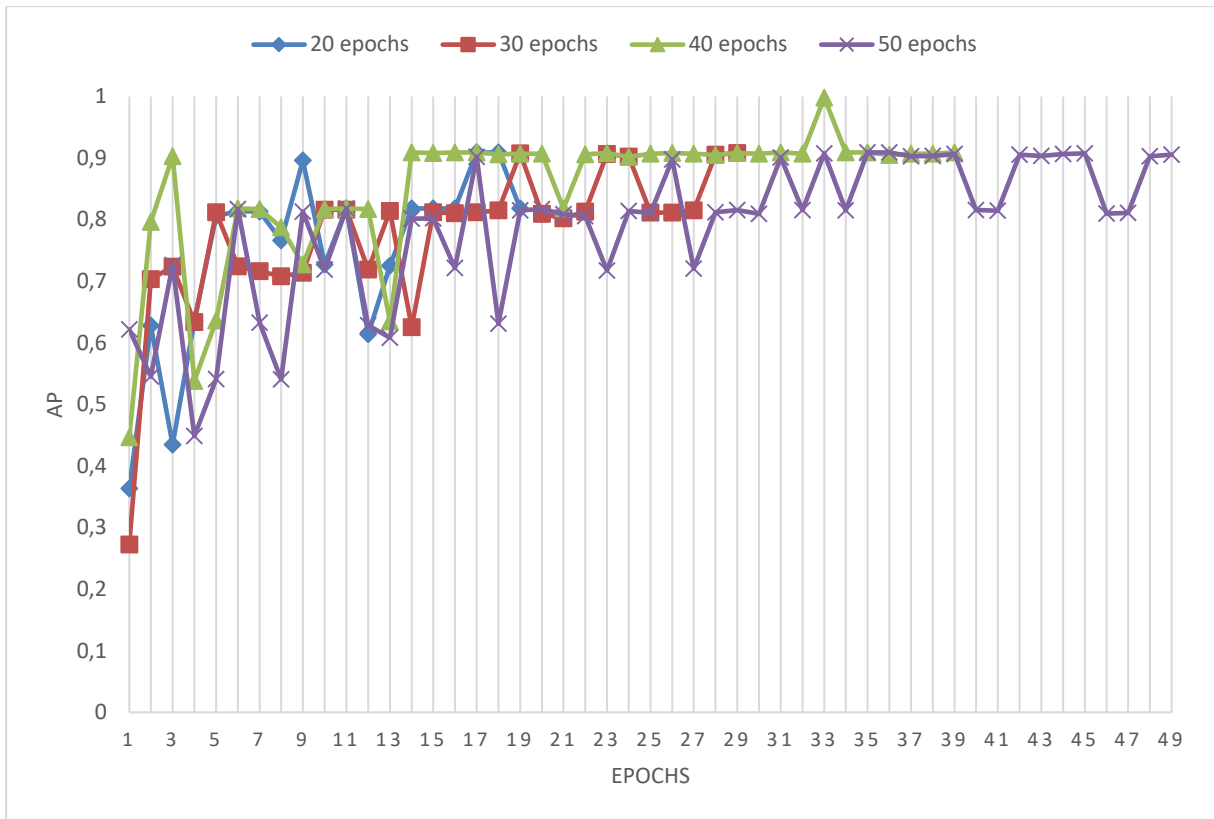


Figure 80 - Validation AP curves of the bottle class

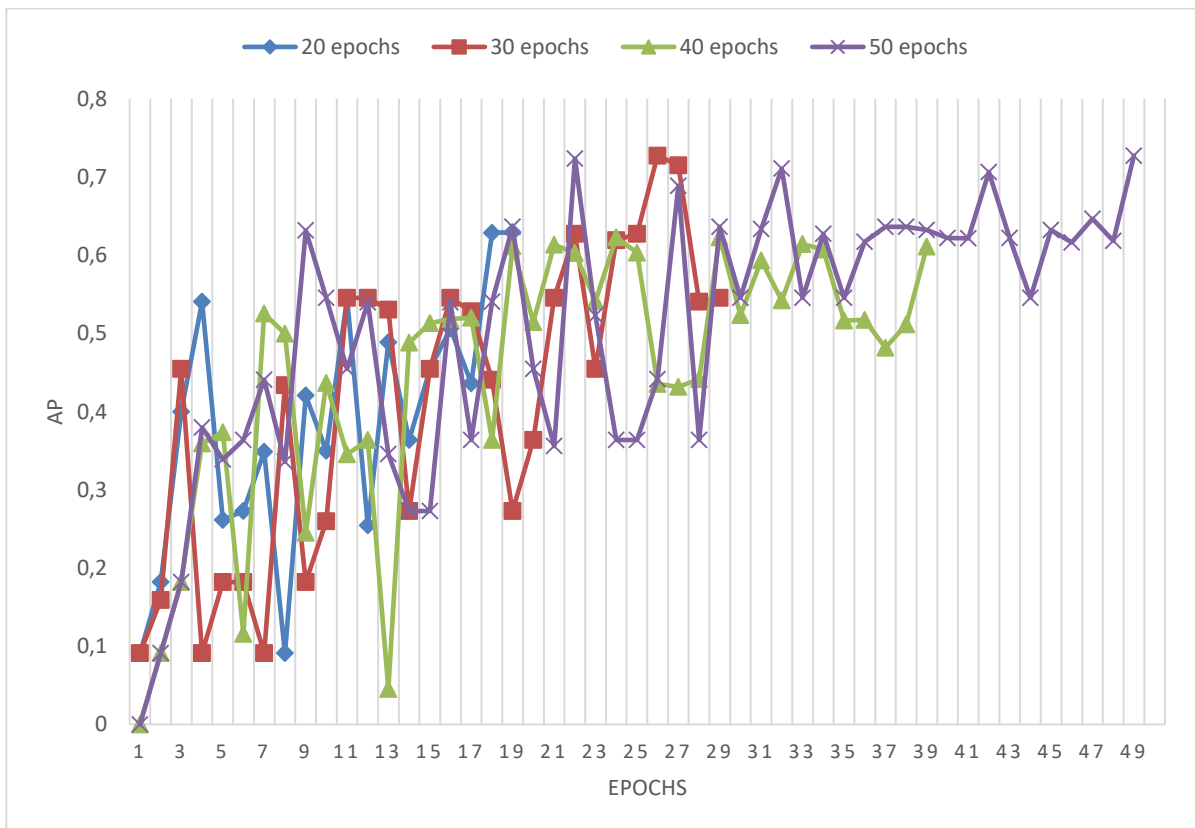


Figure 81 - Validation AP curves of the can class

At first glance, with the addition of another class, the network was able to maintain the performance on the bottle having precision rates over 80%, also by comparing the bottle training and validation curves on Data3 and on Data4, that are similar to each other.

However, the network performance on the can class is way lower than on the bottle class as demonstrated by the training and validation AP values on table 15 as well as the instability revealed in the curves on figure 81 in comparison to the validation bottle curves on figure 80.

This difference on the performance of the network in detecting both classes can be due to large disparity of data for each class, since that in this dataset there are almost three times more data of bottles than cans. This disparity on the number of data allows one class to be trained more times than the other class, consequently, the network will learn more features of the class with the bigger number of data, and thus, having a better precision rates.

Hence, the new dataset, named Data5, is composed of 4044 samples of which 2104 are bottles and the other 1940 are cans.

Therefore, the network was trained with the Data5, this time with only 50 epochs, because it guarantees that the results are the most correct and it is where it is possible to verify a better response. The table 16 shows the final AP values of both classes of the network trained with the Data5, in figures 82 and 83 is demonstrated the training curves of the trained network with Data4 and Data5 datasets, with 50 epochs, and in figures 84 and 85 is displayed the respective validation curves. The results of the Data4 are the ones obtained by the previous training, represented as the 50 epochs curves on the figures 78, 79, 80 and 81.

Table 16 – Final AP values of the network with 50 epochs, using the Data5

	<i>Train AP bottle</i>	<i>Train AP can</i>	<i>Val AP bottle</i>	<i>Val AP can</i>	<i>Training time</i>
<i>50 epochs</i>	0,5274	0,2661	0,9091	0,4503	3d 15h 22min 43s

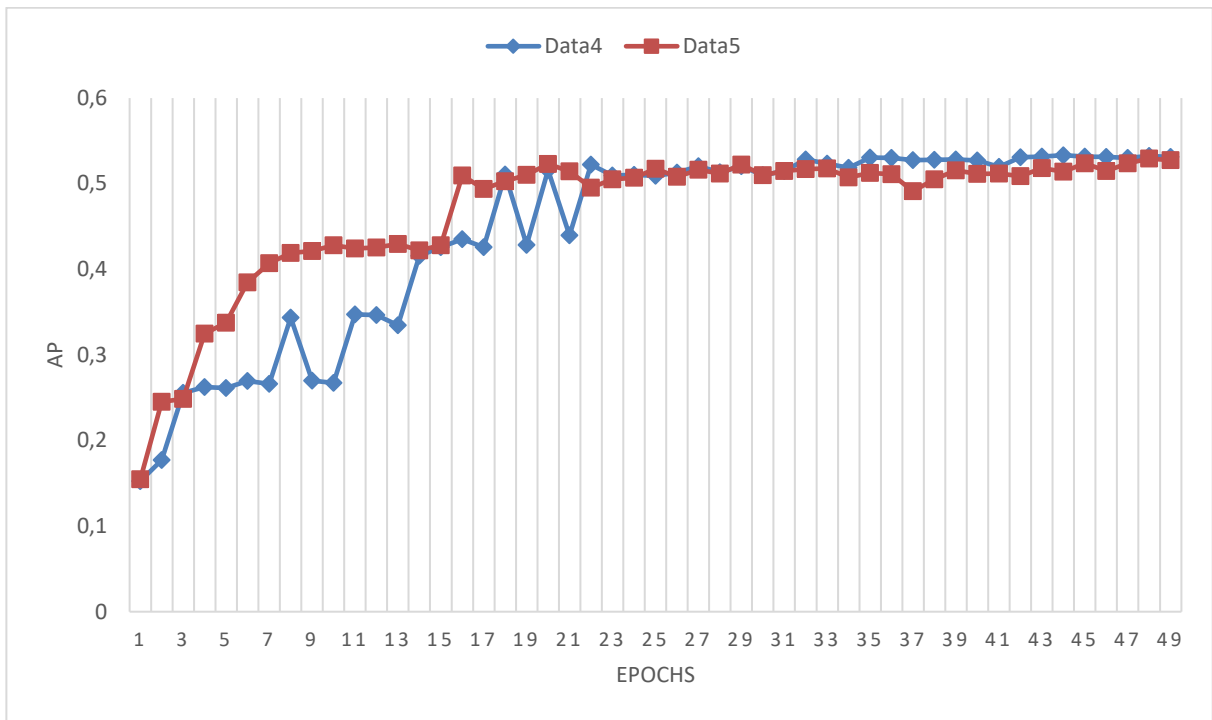


Figure 82 – Training AP of Data 4 and Data5 curves of the bottle class

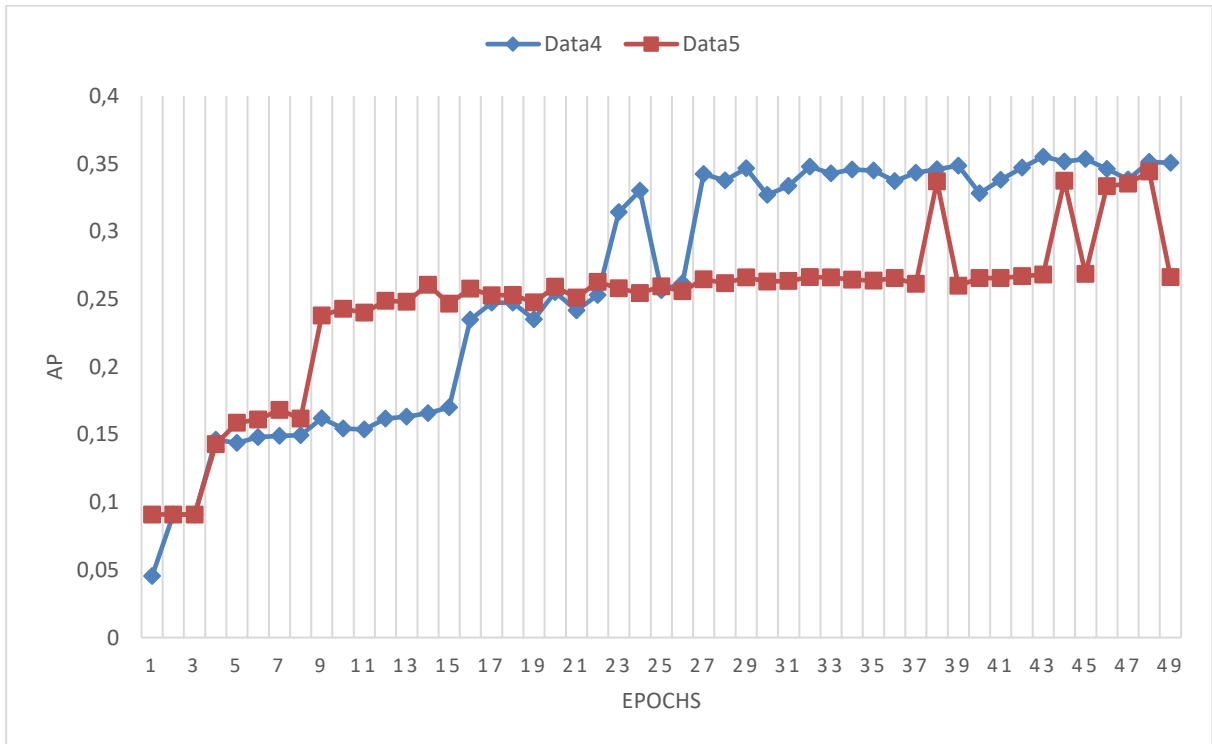


Figure 83 – Training AP of Data4 and Data5 curves of the can class

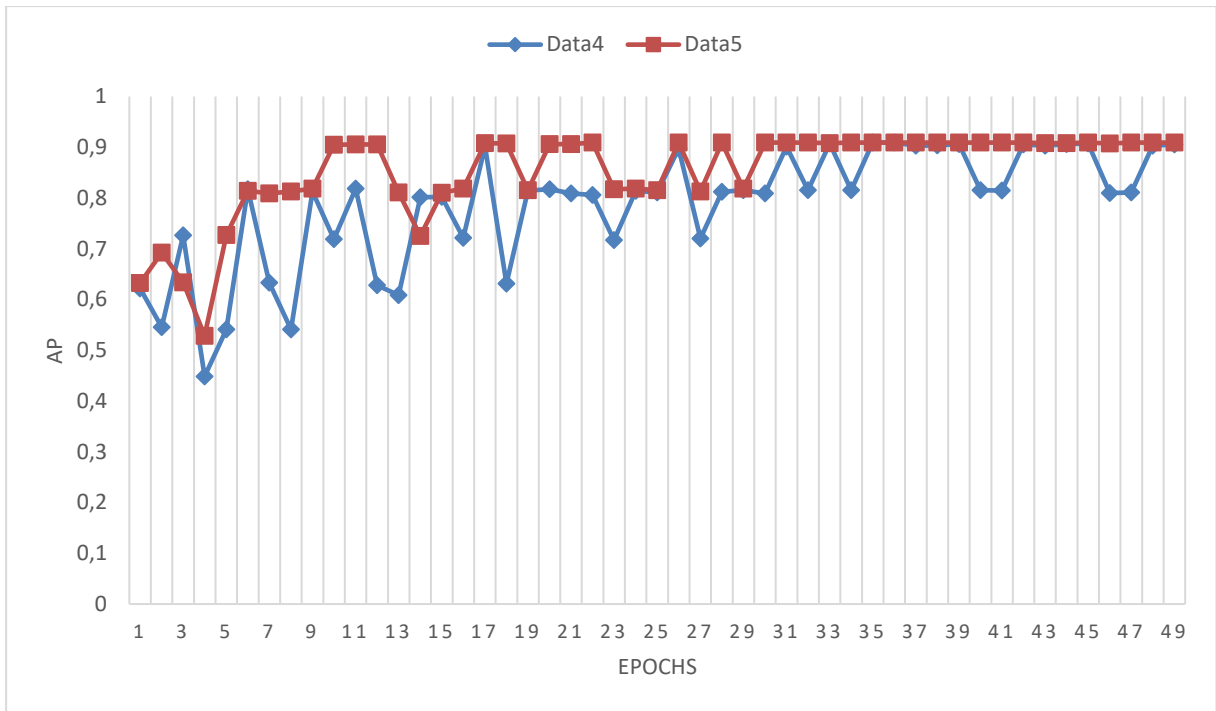


Figure 84 – Validation AP of Data4 and Data5 curves of the bottle class

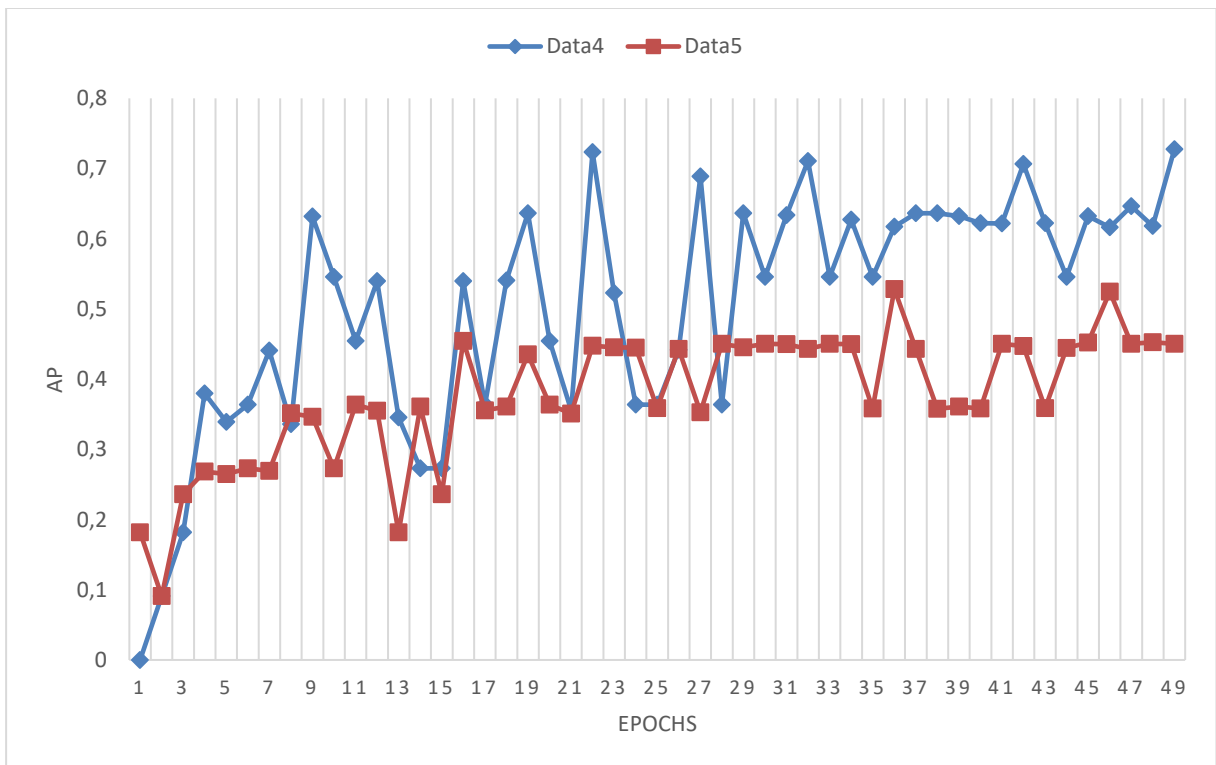


Figure 85 – Validation AP of Data4 and Data5 curves of the can class

As can be seen, the increasing of the can data did not increase the network performance on detecting cans, on the contrary it reduced its performance, as shown in figure 83 and 85 and by the AP values presented in table 16.

Which proves that previous problem of having poor performance in detecting cans was not due to the disparity of data, but because the network is not distinguishing perfectly the cans from the bottles, that is, in the detections there are some cans samples that the network considers them as bottles which caused this poor performance. By increasing the can data, the number of those bad detection also increased, thus reducing the AP values even further, as can be seen in figure 83 and 85.

This happened because, as mentioned above, both classes have similar features and therefore the network could not extract the features that most promote the distinction between the two classes, being that, the features extracted in the training process are somewhat similar, because the input size is not enough to allow the extraction of the features that are unique to each class.

Hence, this problem could be solved by increasing the input size, which would allow the network to extract more elemental features for each class, allowing the network to better distinguish between classes, although, as presented in the frame rate test, higher inputs would result on higher frame rates which does not met the requirements.

The figures 86 and 87 demonstrates the results of the trained network, with data5, on the test part.

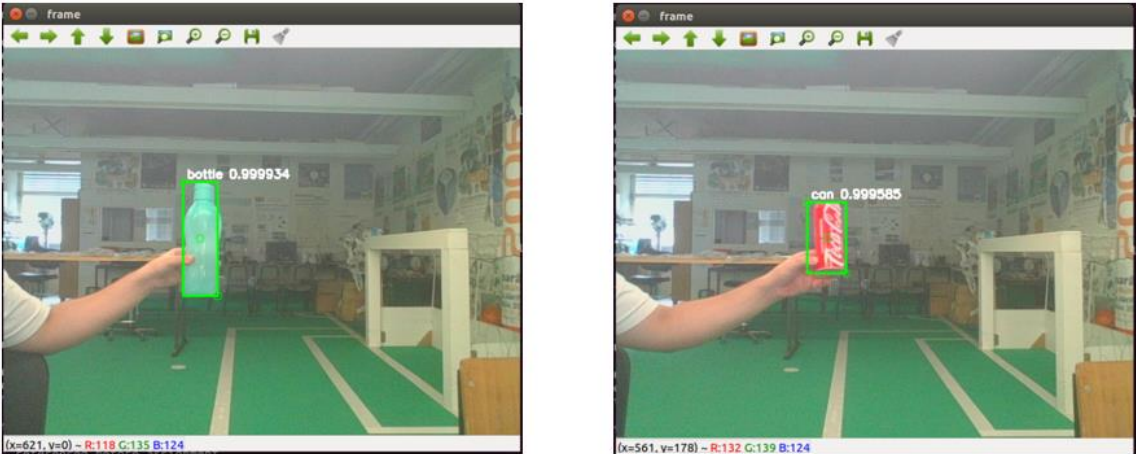


Figure 86 – Detections of a single object in a frame

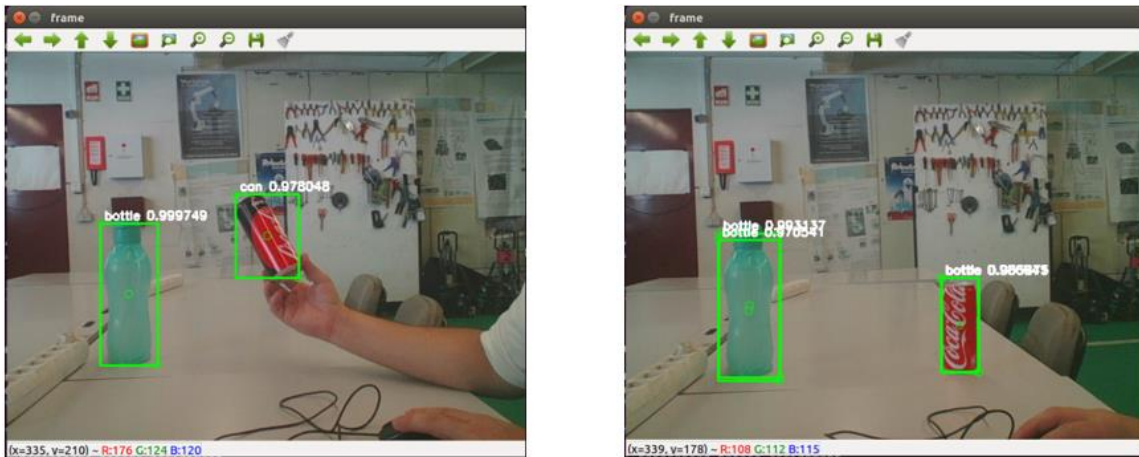


Figure 87 – Detections of both classes

In figure 86 it shows that even with a poor performance on the cans, the network can still detect very well a single can.

On the other hand, on figure 87 is shown that the network can detect both classes, however, as stated above, the network in some perspectives cannot distinguish the cans from the bottles, which lead to the detections presented in the second frame on figure 87.

As for the bottle class, even with the can class, the network is able to almost perfectly detect the bottles presented in the frame.

For the final prototype it was added a third class to the dataset in order to increase the complexity of the training by adding one more class to the network extract the features.

This new class, named bag of chips, is composed of 3 types of objects as shown in figure 88.



Figure 88 - Types of bag of chips in the dataset

The bag of chips class consists of a larger, a medium and a small bag in which they have similar features between them, although, as shown in figure 88, they can take several shapes representing a

greater complexity in relation to can class. Yet, as demonstrated in figure 88, the features that represent the bag of chips are distinct from the other two classes.

The new dataset, Data 6, is constituted by 5936 samples where 2104 are bottles, 1940 are can samples and the others 1892 are the new added class.

Then, the network was trained with the new dataset using the same parameters as the latest training performed with 50 epochs. The tables 17 and 18 displays the AP values in training and in validation of each class and the figure 89 and 90 shows the respective AP response of each class.

Table 17 – Training AP values of the network using Data6

	Train AP bottle	Train AP can	Train AP bag of chips	Training time
50 epochs	0,5342	0,2686	0,5368	5d 8h 26min 6s

Table 18 – Validation AP values of the network using Data6

	Val AP bottle	Val AP can	Val AP bag of chips
50 epochs	0,9091	0,5418	0,8182

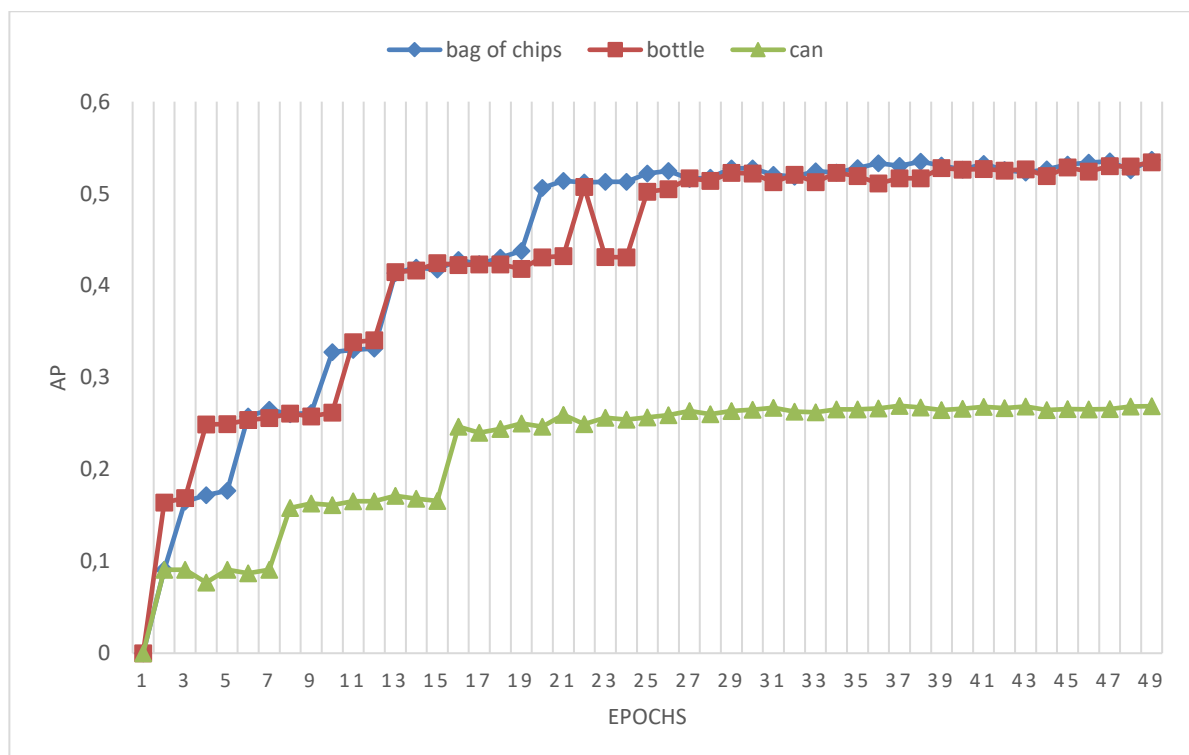


Figure 89 – Training curves of each class

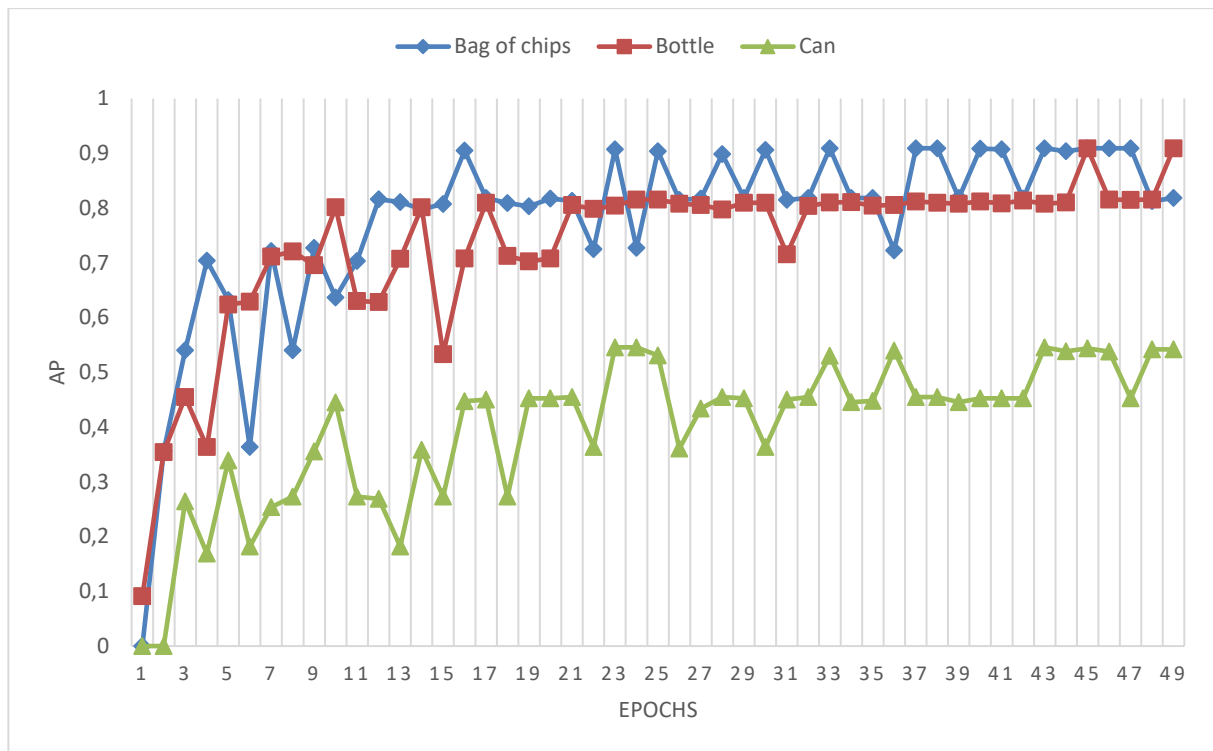


Figure 90 - Validation curves of each class

As shown by the results obtained in the training, the bag of chips class has a similar performance to the bottle class, which proves that the network is able to distinguish the bag of chips from both bottles and cans. It also reinforces that the network confuses the cans from the bottles which represents the lower performance on the can class.

However, even with the addition of another class, the network maintained the performance on the other classes and thus the selected parameters provide coherent results.

In order to verify the hypothesis of the network of having better performances with larger inputs, the network was trained with 200x200 image inputs with the same training parameters and dataset used on previously trained network.

Therefore, the next results displayed on tables 19 and 20, and figures 91 and 92, displays the performance of the new trained network.

Table 19 - Training AP values of the network using Data6 and input of 200x200

	Train AP bottle	Train AP can	Train AP bag of chips	Training time
50 epochs	0,5370	0,3576	0,6244	8d 12h 59min 2s

Table 20 – Validation AP values of the network using Data6 and inputs of 200x200

	Val AP bottle	Val AP can	Val AP bag of chips
50 epochs	0,9091	0,6272	0,9033

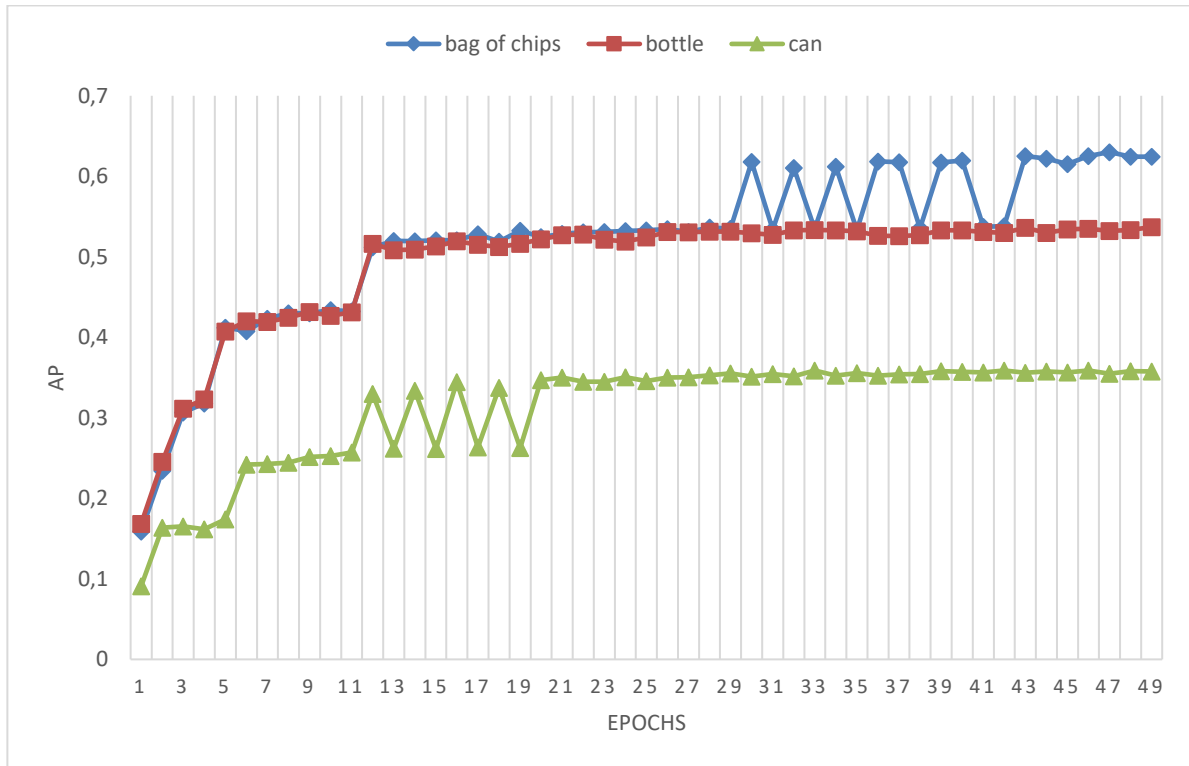


Figure 91 - Training curves of each class using 200x200 as the input size

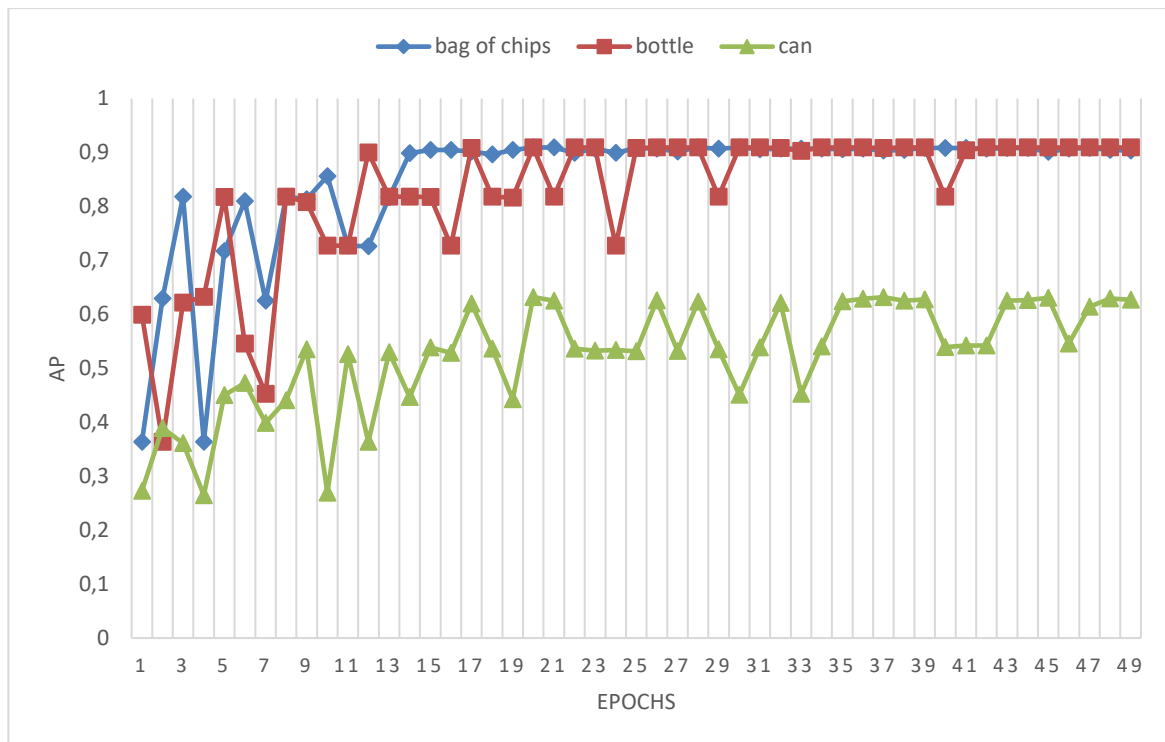


Figure 92 - Validation curves of each class using 200x200 as the input size

At first sight, it is possible to see that even with the increasing of the input size from 150x150 to 200x200 the network is still struggling on distinguishing from classes as proved by figure 92 curves so, it can be inferred that, in this dataset, the input needs to be even higher.

However, the network slightly improved its overall AP, as demonstrated in table 19 and 20, and as demonstrated by the AP curves in figure 91 and 92 which are much smoother than the previous ones, proving that the increase of the input size has somewhat provided and increase in the network performance on this dataset.

4.3 Final prototype

The final prototype is the network who was trained with following training parameters:

- Dataset = Data6;
- Number of epochs = 50;
- Input = 150x150;
- Learning rate = 0,005;
- Batch size = 4;
- NMS > 0,45.

Henceforth the network will be tested in the final part of the model where the network is receiving constant frames from the camera and the results of the network is being displayed, in real time.

The following figures, 93, 94 and 95, demonstrates some frames and the respective detections performed by the network for each class.

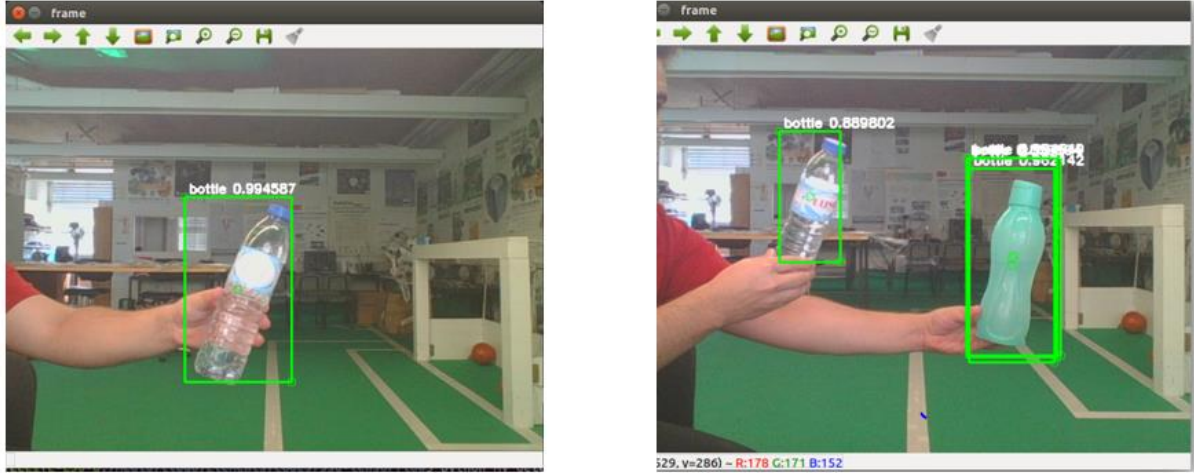


Figure 93 - Bottle detections

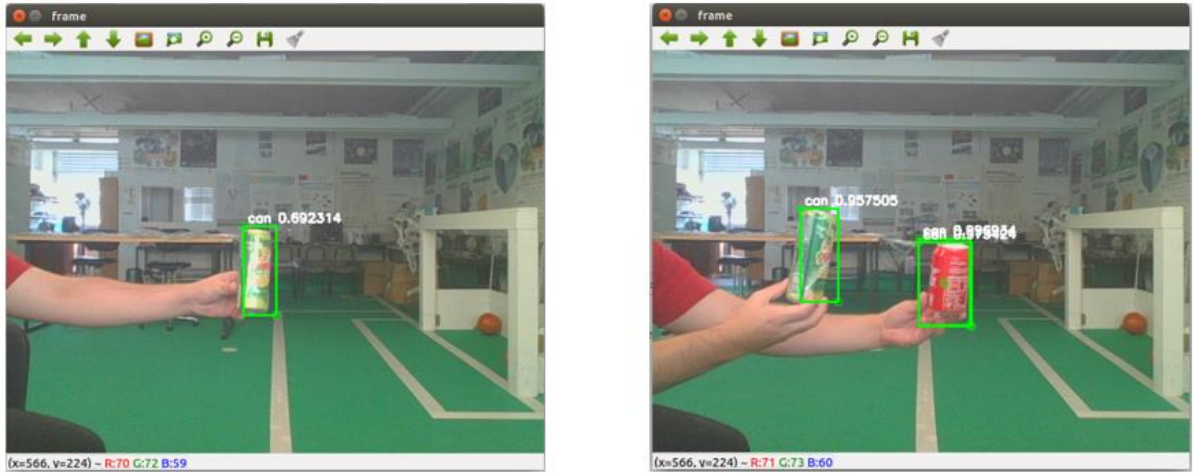


Figure 94 - Can detections

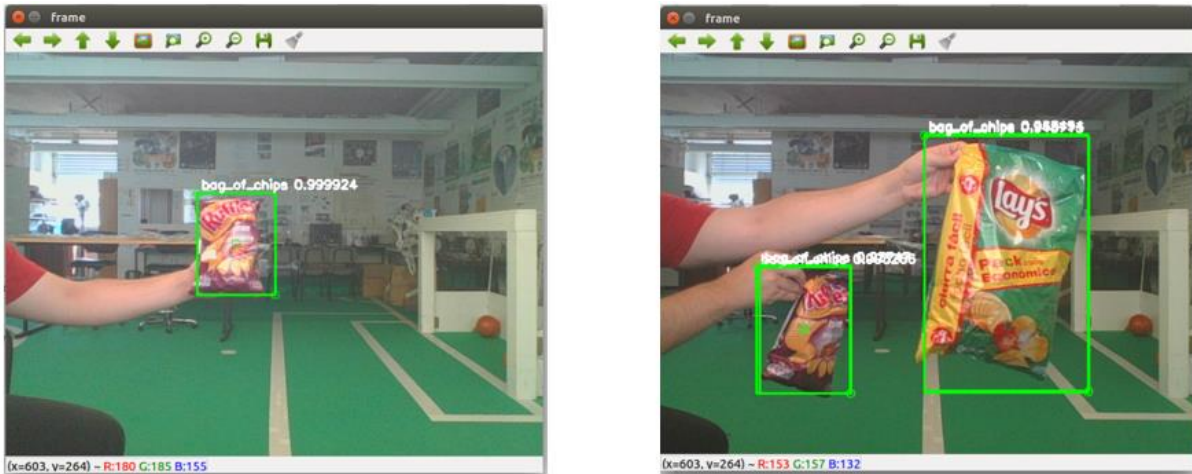


Figure 95 - Bag of chips detections

These detections represented in figures 93, 94 and 95, evidence that the network can follow the variety of each class presented in the dataset, performing good and somewhat accurate detections and by recognizing what is the correct object displayed in the environment.

The figure 96, shows the network detections, when it is represented more than one type of object in the environment, in successive frames.

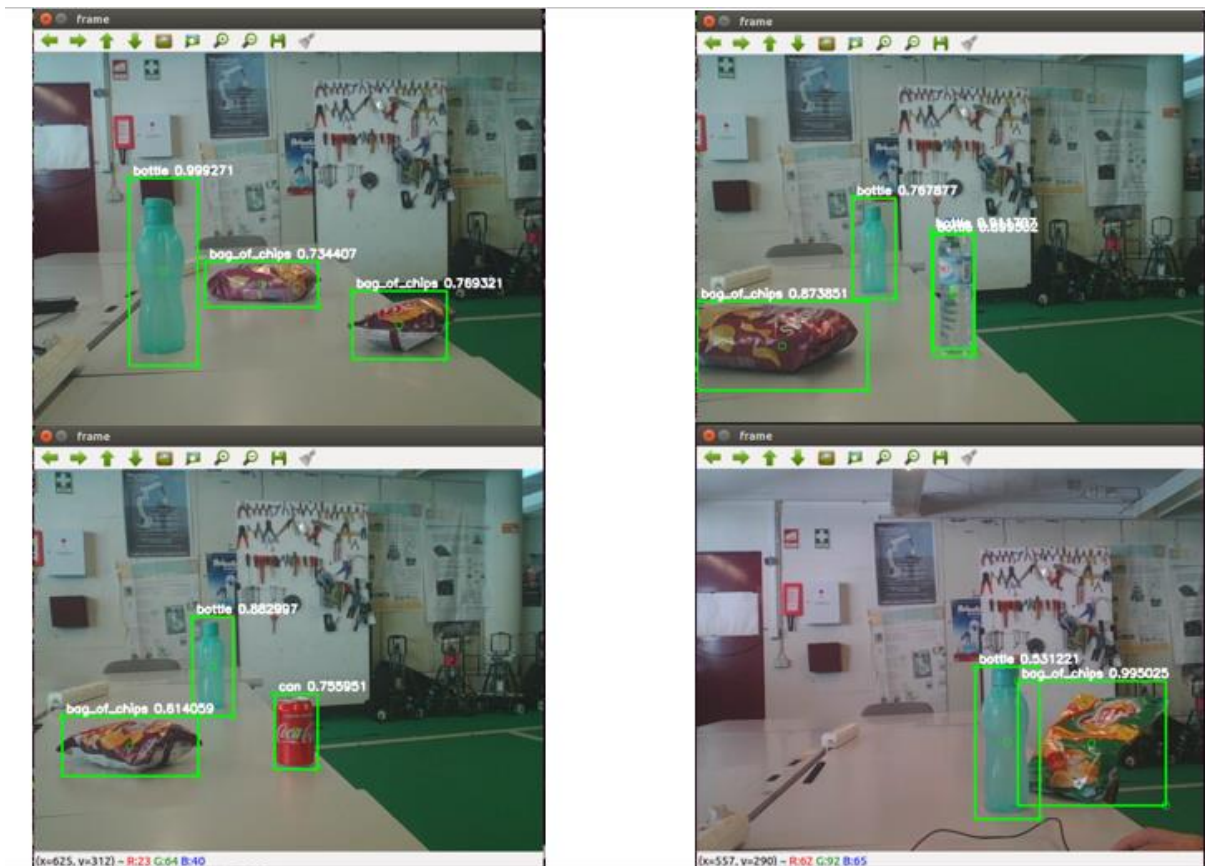


Figure 96 - Network detections in more than one class

The results presented in figure 96 demonstrates the capability of the network to distinguish between classes and to detect the correct position of the respective objects placed in different positions and perspectives.

Considering that all models are not ideal, this model still performs some mistakes on detecting and/or recognizing the object in the environment. However, some of the errors could be solved with larger inputs and therefore deeper networks, one of which is the performance on the can class where the network cannot differentiate in some perspectives the bottle from the can as represented in figure 97.

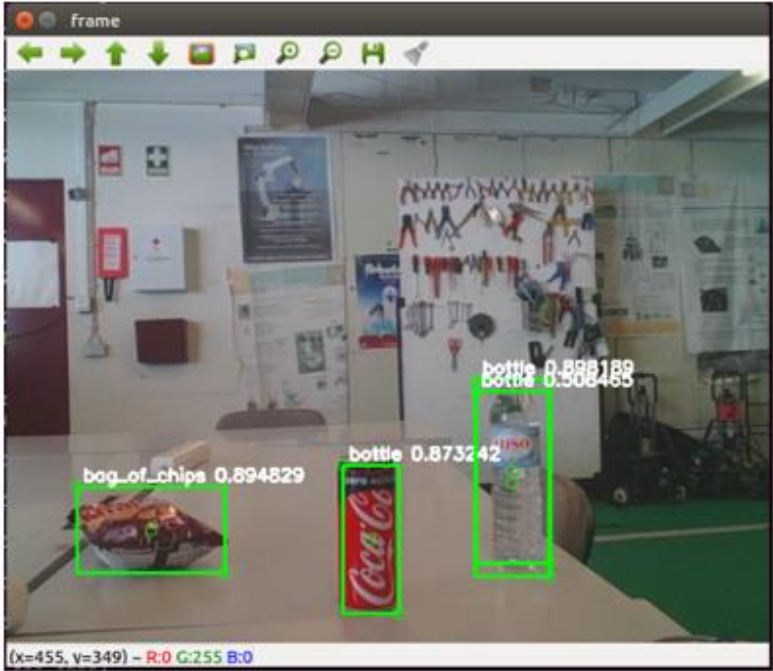


Figure 97 –Bad recognition

Moreover, in some perspectives, even when the objects are presented in the image, the network does not detect them, as shown in figure 98.

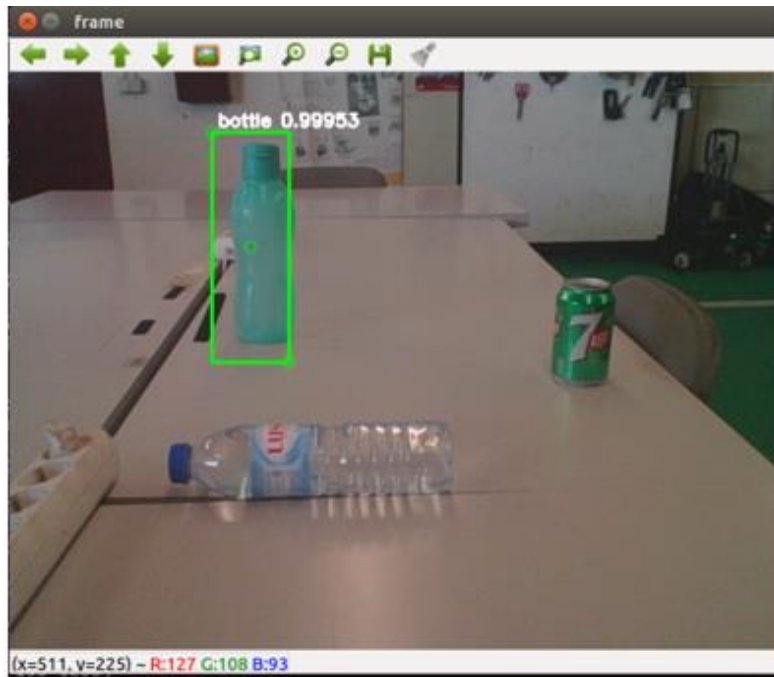


Figure 98 - Nonexistent detection

Nevertheless, the detections presented on figures 93, 94, 95 and 96, proves that even with low resolution inputs (150x150), the network is still able to detect the trained objects, presented in the environment, regardless of its perspectives and positions.

5. CONCLUSIONS

After the completion and the acquisition of the results, presented above, it can be concluded that the proposed objectives were fulfilled, as well as the requirements placed to carry out the problem with the available resources.

One of the most visible conclusion, is that, the usage of CPUs in a CNN based vision model greatly limits the range of the training parameters, principally the size of the input, which prevents the use of deeper networks, since that the inputs above 150x150 have a frame rate of less than 5FPS. Moreover, training these networks, with CPUs, takes a lot of time with longer datasets.

However, when verifying the obtained results, it was possible to see that the network, even with considerably low-resolution inputs (150x150) the network was able to detect and classify a set of objects presented in the field of view proving the viability of this model to solve vision problems even with low resources. Still, the network struggled on distinguishing relatively similar objects, due to the incapability of the network to extract the features that most relates to a specific object, that is, the features extracted in the training process do not represented the maximum level of abstraction per class, caused by the use of lower resolution inputs, which somewhat affected the consistency of the model, concluding that higher inputs would result on better performances. Therefore, it is highly recommended to use, if possible and depending on the problem, higher input resolutions.

It can also be concluded that the use of own made dataset allows to train the network with only the desired classes for the problem, instead of training the model with datasets that have more classes for the problem occupying unnecessary space. Yet, to create such dataset, it is needed to implement a system that can obtain the information for the problem, in this case for object detection, the coordinates of the bounding box and the name of the object, which can be somewhat complex. The implemented algorithm for the acquisition, as proved, was able to acquire that information in any kind of object in real-time, even though it must be somewhat controlled.

The acquisition in real-time proved to be effective in training the model, being that, the model was able to detect not only static objects but also when the object is moving, it also allowed to provide to the network innumerous perspectives and representations of the same object, frames with different light conditions, reinforcing the model on any type of variants. Moreover, the use of data augmentation is highly recommended since it applies modification to the frames to prevent the network to learn the easiest representations, namely, the color of the objects, the distance of the features, increasing the variability of the data.

Hence, it was concluded that to create a dataset, a large number of images of the same type of object is required, to allow the network to learn a greater number of features that allow its detection in several possible ways.

In general, for robotic vision systems, this model is feasible, considering that in most robotic systems the resources are not very high, since there are usually cost limits, although, if the system is highly dependent of the vision system is it recommended to use GPUs, as it allows higher inputs. In computer vision tasks the resources are usually high, for example, the presence of GPUs and thus it is advised to use them, as it does not limit the parameters and it will allow better performances.

6. FUTURE WORK

For future work, there are several works that could be performed on this system that would increase not only its performance but also its complexity, such as:

- The resolution of the main issue presented on this system, which is the use of considerably low inputs, as has been concluded. To perform this resolution, it could be used GPUs or in some way it could be used any kind of optimization that would increase the system speed;
- Use different resolution cameras to train and test the network, to see if it influences the network behavior;
- Increase the classes as well as the dataset used to train the network to increase the variability of the detections;
- Automatize, if possible, the acquisition method by making it less dependent, by eliminating the use of the color segmentation;
- Add to the dataset as well as to the network the z coordinate, which would represent the distance of the object;
- Use other CNN architecture as the main feature extractor instead of the VGG-16 network, to compare the network performance with different feature extractors.

REFERENCES

- [1] D. Kragic and M. Vincze, "Vision for Robotics," *Found. Trends Robot.*, vol. 1, no. 1, pp. 1–78, 2009.
- [2] Y. A. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [3] N. Dalal and B. Triggs, "HOG," in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005, vol. 1, pp. 886–893.
- [4] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, pp. 1150–1157 vol.2.
- [5] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, vol. 3951 LNCS, pp. 404–417.
- [6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proceedings of the IEEE International Conference on Computer Vision*, 2011, pp. 2564–2571.
- [7] J. Ruiz-del-solar, P. Loncomilla, and N. Soto, "A Survey on Deep Learning Methods for Robot Vision," *arXiv Comput. Vis. Pattern Recognit.*, pp. 1–43, 2018.
- [8] Y. Lecun, L. Eon Bottou, Y. Bengio, and P. Haaner, "Gradient-Based Learning Applied to Document Recognition RS-SVM Reduced-set support vector method. SDNN Space displacement neural network. SVM Support vector method. TDNN Time delay neural network. V-SVM Virtual support vector method," *PROC. IEEE*, 1998.
- [9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition."
- [10] D. O. Hebb, "The Organization of Behavior," *Organ. Behav.*, vol. 911, no. 1, p. 335, 1949.
- [11] M. A. Nielsen, "Neural Networks and Deep Learning." Determination Press, 2015.
- [12] "Artificial Intelligence Neural Networks." [Online]. Available: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm. [Accessed: 12-Sep-2018].
- [13] "A Beginner's Guide to Neural Networks and Deep Learning | Skymind." [Online]. Available: <https://skymind.ai/wiki/neural-network>. [Accessed: 11-Sep-2018].
- [14] D. Hubel and T. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *J. Physiol.*, pp. 215–243, 1968.

- [15] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *J. Physiol.*, vol. 148, no. 3, pp. 574–591, 1959.
- [16] K. O'shea and R. Nash, "An Introduction to Convolutional Neural Networks."
- [17] "Neural Networks Tutorial - A Pathway to Deep Learning - Adventures in Machine Learning." [Online]. Available: <http://adventuresinmachinelearning.com/neural-networks-tutorial/>. [Accessed: 12-Sep-2018].
- [18] "Identifying Traffic Signs with Deep Learning – Towards Data Science." [Online]. Available: <https://towardsdatascience.com/identifying-traffic-signs-with-deep-learning-5151eece09cb>. [Accessed: 11-Sep-2018].
- [19] "A Beginner's Guide To Understanding Convolutional Neural Networks – Adit Deshpande – CS Undergrad at UCLA ('19)." [Online]. Available: <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>. [Accessed: 12-Sep-2018].
- [20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks."
- [21] "Supervised vs. Unsupervised Machine Learning." [Online]. Available: <https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms>. [Accessed: 05-Sep-2018].
- [22] "Supervised vs. Unsupervised Learning – Towards Data Science." [Online]. Available: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. [Accessed: 12-Sep-2018].
- [23] "Classification Versus Regression – Intro To Machine Learning #5." [Online]. Available: <https://medium.com/simple-ai/classification-versus-regression-intro-to-machine-learning-5-5566efd4cb83>. [Accessed: 05-Sep-2018].
- [24] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: <http://yann.lecun.com/exdb/mnist/>. [Accessed: 12-Sep-2018].
- [25] "Neural networks and backpropagation explained in a simple way." [Online]. Available: <https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e>. [Accessed: 12-Sep-2018].
- [26] "Loss Functions in Neural Networks | Isaac Changhau." [Online]. Available: https://isaacchanghau.github.io/post/loss_functions/. [Accessed: 12-Sep-2018].
- [27] S. Ruder, "An overview of gradient descent optimization algorithms," Sep. 2016.
- [28] "How do we 'train' neural networks? – Towards Data Science." [Online]. Available:

- <https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73>. [Accessed: 12-Sep-2018].
- [29] “Intersection over Union (IoU) for object detection - PyImageSearch.” [Online]. Available: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Accessed: 12-Sep-2018].
- [30] “Object detection with neural networks – Towards Data Science.” [Online]. Available: <https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491>. [Accessed: 11-Sep-2018].
- [31] “Evaluation of ranked retrieval results.” [Online]. Available: <https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html>. [Accessed: 12-Sep-2018].
- [32] “mAP (mean Average Precision) for Object Detection – Jonathan Hui – Medium.” [Online]. Available: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. [Accessed: 12-Sep-2018].
- [33] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
- [34] “ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012).” [Online]. Available: <http://www.image-net.org/challenges/LSVRC/2012/results.html>. [Accessed: 01-Nov-2017].
- [35] “ImageNet Large Scale Visual Recognition Competition 2011 (ILSVRC2011).” [Online]. Available: <http://image-net.org/challenges/LSVRC/2011/results>. [Accessed: 30-Oct-2017].
- [36] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” 2014.
- [37] “ImageNet Large Scale Visual Recognition Competition 2013 (ILSVRC2013).” [Online]. Available: <http://www.image-net.org/challenges/LSVRC/2013/results.php>. [Accessed: 12-Sep-2018].
- [38] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.
- [39] R. Girshick, “Fast R-CNN,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, vol. 2015 Inter, pp. 1440–1448.
- [40] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

- [41] M. D. Zeiler and R. Fergus, "LNCS 8689 - Visualizing and Understanding Convolutional Networks," 2014.
- [42] C. Szegedy *et al.*, "Going Deeper with Convolutions."
- [43] K. Simonyan and A. Zisserman, "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," 2015.
- [44] "ImageNet Large Scale Visual Recognition Competition 2014 (ILSVRC2014)." [Online]. Available: <http://www.image-net.org/challenges/LSVRC/2014/results.php>. [Accessed: 12-Sep-2018].
- [45] M. Lin, Q. Chen, and S. Yan, "Network In Network," *arXiv Prepr.*, 2013.
- [46] "ImageNet Large Scale Visual Recognition Competition 2015 (ILSVRC 2015)." [Online]. Available: <http://www.image-net.org/challenges/LSVRC/2015/results.php>. [Accessed: 12-Sep-2018].
- [47] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *arXiv Prepr. arXiv*, 2013.
- [48] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition."
- [49] K. He, X. Zhang, S. Ren, and J. Sun, "SPP-net," *IEEE Trans. Pattern Anal. Mach. Intell.*, no. Figure 2, pp. 1–14, 2014.
- [50] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, vol. 2017–Octob, pp. 2980–2988.
- [51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, " (2016 YOLO) You Only Look Once: Unified, Real-Time Object Detection," *Cvpr 2016*, pp. 779–788, 2016.
- [52] J. Redmon and A. Farhadi, "YOLO9000:Better , Faster , Stronger," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 7263–7271, 2017.
- [53] W. Liu *et al.*, "SSD," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016.
- [54] D. Erhan, C. Szegedy, A. T. Dragomir, and A. Google, "Scalable Object Detection using Deep Neural Networks."
- [55] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks."
- [56] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and."
- [57] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder

- Architecture for Image Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017.
- [58] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017–Janua, pp. 2261–2269.
- [59] M. Sadegh Norouzzadeh *et al.*, “Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning Background and Related Work,” 2017.
- [60] “Traffic Sign Recognition using Convolutional Neural Network - ProggBlogg.” [Online]. Available: <https://tomaszkacmajor.pl/index.php/2017/10/15/traffic-sign-recognition-using-cnn/>. [Accessed: 12-Sep-2018].
- [61] O. M. Parkhi, A. Vedaldi, and A. Zisserman, “Deep Face Recognition,” in *Proceedings of the British Machine Vision Conference 2015*, 2015, p. 41.1-41.12.
- [62] H. Jiang and E. Learned-Miller, “Face Detection with the Faster R-CNN,” in *Proceedings - 12th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2017 - 1st International Workshop on Adaptive Shot Learning for Gesture Understanding and Production, ASLAGUP 2017, Biometrics in the Wild, Bwild 2017, Heteroge*, 2017, pp. 650–657.
- [63] R. Zhang, P. Isola, and A. A. Efros, “Colorful Image Colorization.”
- [64] A. Karpathy and L. Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions.”
- [65] C. Liu, B. Zheng, C. Wang, Y. Zhao, S. Fu, and H. Li, “CNN-Based Vision Model for Obstacle Avoidance of Mobile Robot,” *MATEC Web Conf.*, 2017.
- [66] L. Ran, Y. Zhang, Q. Zhang, and T. Yang, “Convolutional Neural Network-Based Robot Navigation Using Uncalibrated Spherical Images.,” *Sensors (Basel)*, vol. 17, no. 6, Jun. 2017.
- [67] G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale, “Real-world Object Recognition with Off-the-shelf Deep Conv Nets: How Many Objects can iCub Learn?”
- [68] J. Hosang, M. Omran, R. Benenson, and B. Schiele, “Taking a Deeper Look at Pedestrians.”
- [69] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, “Deep convolutional neural networks for pedestrian detection.”
- [70] D. Speck, P. Barros, C. Weber, and S. Wermter, “Ball Localization for Robocup Soccer Using Convolutional Neural Networks,” 2017, pp. 19–30.
- [71] D. Albani, A. Youssef, V. Suriani, D. Nardi, and D. D. Bloisi, “A Deep Learning Approach for Object Recognition with NAO Soccer Robots.”

- [72] "TensorFlow." [Online]. Available: <https://www.tensorflow.org/>. [Accessed: 06-Sep-2018].
- [73] "OpenCV: Structural Analysis and Shape Descriptors." [Online]. Available: https://docs.opencv.org/3.3.1/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a. [Accessed: 14-Sep-2018].
- [74] "OpenCV: Operations on arrays." [Online]. Available: https://docs.opencv.org/3.4.3/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981. [Accessed: 14-Sep-2018].
- [75] "OpenCV: Background Subtraction." [Online]. Available: https://docs.opencv.org/3.3.0/db/d5c/tutorial_py_bg_subtraction.html. [Accessed: 14-Sep-2018].
- [76] "What Is the HSV (Hue, Saturation, Value) Color Model?" [Online]. Available: <https://www.lifewire.com/what-is-hsv-in-design-1078068>. [Accessed: 06-Sep-2018].
- [77] "Single Shot Detector (SSD) from scratch in TensorFlow - Lukasz Janyst's web site." [Online]. Available: <http://jany.st/post/2017-11-05-single-shot-detector-ssd-from-scratch-in-tensorflow.html>. [Accessed: 12-Sep-2018].
- [78] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: <http://cs231n.github.io/linear-classify/>. [Accessed: 16-Sep-2018].
- [79] "Activation Functions: Neural Networks – Towards Data Science." [Online]. Available: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. [Accessed: 11-Sep-2018].

APPENDIX A


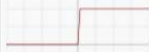

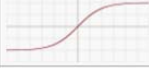



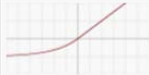

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Figure 99 – Activation functions [79]

APPENDIX B

This appendix will exhibit the several algorithms implemented to perform the dataset acquisition, presented in the methods and methodologies chapter.

Appendix B1

Code for the acquisition using color segmentation.

```
import cv2
import os
import sys, argparse
import numpy as np
from utils import Label, rgb2bgr

img = np.zeros((256,256,3), np.uint8)
kernel = np.ones((7,7), np.uint8)
dir_path = os.path.dirname(os.path.realpath(__file__))
path = dir_path + '/data/Dataset1/Images'
x1=-1
y1=-1
values = []
flgs = [0,0,0]

def nothing(x):
    pass

cv2.namedWindow('thresh')
cv2.createTrackbar('S1_MIN', 'thresh', 0, 255, nothing)
cv2.createTrackbar('V1_MIN', 'thresh', 0, 255, nothing)

#-----
#Mouse event function
#When the left mouse button is clicked it saves the clicked point
#-----
def onMouse(event, x, y, flags, param):
    global x1, y1, flgs
    if event == cv2.EVENT_LBUTTONDOWN:
        x1=y
        y1=x
        flgs[0] = 1

#-----
# Labels
#-----
label_defs = [
    Label('bottle', rgb2bgr((0,255,0))),
    Label('can', rgb2bgr((0,0,0))),
    Label('bag of chips', rgb2bgr((255,0,0)))
]

def main():
    #-----
    #get the object label to create the dataset
    parser = argparse.ArgumentParser(description='Dataset')
```

```
parser.add_argument('-cls', help='classes:' + str(label_defs), required=False)
```

```
args = parser.parse_args()
```

```
try:
```

```
    obj_name = label_defs[obj_label][0]
except(IndexError) as e:
    print('The desired class dont exist', str(e))
    return 1
```

```
    print('Class :!', obj_name)
```

```
#-----
```

```
#read the iteration from file
```

```
    f1 = open(dir_path + '/iteration.txt', 'r+')
    i = f1.read()
    print(i)
```

```
#-----
```

```
cap = cv2.VideoCapture(0)
cv2.namedWindow('frame')
cv2.setMouseCallback('frame',onMouse)
save = 0
```

```
while(True):
```

```
    #open the annotation file
    f = open(dir_path + '/data/Dataset1/Annotations/%s.txt%i,w')
    _, frame = cap.read()
    src = frame
    #apply a gaussian filter
    blur = cv2.GaussianBlur(frame,(9,9),0)
    #change the frame color space
    gray = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
```

```
    if flgs[0] == 1:
        #get the hsv value on the clicked position and save it
        values = hsv[x1, y1]
        flgs[0] = 0
        flgs[2] = 1
        cv2.setTrackbarPos('S1_MIN', 'thresh', values[1])
        cv2.setTrackbarPos('V1_MIN', 'thresh', values[2])
```

```
    elif flgs[2] == 1:
        #obtain the value of the current position of the trackbar
        S1min = cv2.getTrackbarPos('S1_MIN', 'thresh')
        V1min = cv2.getTrackbarPos('V1_MIN', 'thresh')

        #establish the range of the values
        lower = np.array([values[0]-10,S1min,V1min])
        upper = np.array([values[0]+10,255,255])

        #perform the segmentation with the previous range values
        mask = cv2.inRange(hsv, lower, upper)
```

```
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```

# get the bounding rect and display only the bigger contour,
# in order to display only one rect
if len(contours) > 0:
    filename ='%s.jpg%i'
    cv2.imwrite(os.path.join(path , filename), src)
    #pick the bigger contour
    c = max(contours, key = cv2.contourArea)
    x, y, w, h = cv2.boundingRect(c)
    # draw a green rectangle to visualize the bounding rect
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
    cv2.circle(frame, (x,y), 10, (0,255,0))
    cv2.circle(frame, (x+w,y+h), 10, (0,255,0))
    #save the coordinates
    f.write(str(x) + ', ' + str(y) + ', ' + str(x+w) + ', ' + str(y+h)+ ', ' + str(obj_name))
    #rewrite on the iteration file the current value of the iteration
    f1.seek(0)
    f1.write(str(i))
    i=int(i)+1

cv2.imshow('mask', mask)

cv2.imshow('hsv', hsv)
cv2.imshow('source', src)
cv2.imshow('frame', frame)
if cv2.waitKey(1) & 0xFF == 27:
    break

f.close()
f1.close()
cap.release()
cv2.destroyAllWindows()

if __name__ == '__main__':
    sys.exit(main())

```

Appendix B2

MOG background subtractor code for the acquisition.

```

import sys
import cv2
import numpy as np
from utils import Label, rgb2bgr

dir_path = os.path.dirname(os.path.realpath(__file__))
path = dir_path + '/data/Dataset1/Images'

img = np.zeros((256,256,3), np.uint8)
kernel = np.ones((11,11),np.uint8)
kernel1 = np.ones((5,5),np.uint8)

# -----
# Labels
# -----
label_defs = [
    Label('bottle', rgb2bgr((0,255,0))),
    Label('can', rgb2bgr((0,0,0))),

```

```

Label('bag of chips' , rgb2bgr((255,0,0)))

def main():
    #-----
    #get the object label to create the dataset
    parser = argparse.ArgumentParser(description='Dataset')
    parser.add_argument('-cls', help='classes:' + str(label_defs), required=False)

    args = parser.parse_args()

    try:
        obj_name = label_defs[obj_label][0]
    except(IndexError) as e:
        print('The desired class dont exist', str(e))
        return 1

    print('Class :', obj_name)

    #-----
    #read the iteration from file

    f1 = open(dir_path + '/iteration.txt', 'r+')
    i = f1.read()
    print(i)

    #-----
    cap = cv2.VideoCapture(0)
    cv2.namedWindow('frame')
    cv2.setMouseCallback('frame',onMouse)
    fgbg = cv2.BackgroundSubtractorMOG2()

    while(True):
        #open annotation file
        f = open(dir_path + '/data/Dataset1/Annotations/%s.txt%i,'w')
        #obtain the current frame
        _, frame = cap.read()
        src = frame
        mask = frame
        #apply a blurring filter on the frame
        blur = cv2.GaussianBlur(frame, (15,15), 0)

        #apply the MOG on the actual frame
        fgmask = fgbg.apply(blur)
        #morphological filters
        fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel1)
        fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel1)

        contours, hier = cv2.findContours(fgmask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        # get the bounding rect and display only the bigger contour,
        # in order to display only one rect
        if len(contours) > 0:
            filename ='%s.jpg%i'
            cv2.imwrite(os.path.join(path , filename), src)
            #pick the bigger contour
            c = max(contours, key = cv2.contourArea)
            x, y, w, h = cv2.boundingRect(c)
            # draw a green rectangle to visualize the bounding rect
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            cv2.circle(frame, (x,y), 10, (0,255,0))

```



```

        cv2.circle(frame, (x+w,y+h), 10, (0,255,0))
        #save the coordinates
        f.write(str(x) + ', ' + str(y) + ', ' + str(x+w) + ', ' + str(y+h) + ', ' + str(obj_name))
        #rewrite on the iteration file the current value of the iteration
        f1.seek(0)
        f1.write(str(i))
        i=int(i)+1

    cv2.imshow('frame', frame)
    cv2.imshow('fgmask', fgmask)

    if cv2.waitKey(1) & 0xFF == 27:
        break

    f.close()
    f1.close()
    cap.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    sys.exit(main())

```

Appendix B3

Code for the final acquisition model.

```

import cv2
import os
import sys, argparse
import numpy as np

from utils import Label, rgb2bgr

img = np.zeros((256,256,3), np.uint8)
kernel = np.ones((11,11), np.uint8)
kernel1 = np.ones((7,7), np.uint8)
kernel2 = np.ones((3,3), np.uint8)

dir_path = os.path.dirname(os.path.realpath(__file__))
path = dir_path + '/data/Dataset12/Images'

x1=-1
y1=-1
values = []
flgs = [0,0,0,0]

def nothing(x):
    pass

cv2.namedWindow('thresh')
cv2.createTrackbar('S1_MIN', 'thresh', 0, 255, nothing)
cv2.createTrackbar('V1_MIN', 'thresh', 0, 255, nothing)
#-----
#Mouse event function
#When the left mouse button is clicked it saves the clicked point
#When the middle button is clicked in passed the algorithm to the next stage
#-----

```

```

def onMouse(event, x, y, flags, param):
    global x1, y1, x2, y2, flgs
    if event == cv2.EVENT_LBUTTONDOWN:
        x1=y
        y1=x
        flgs[0] = 1

    if event == cv2.EVENT_MBUTTONDOWN:
        flgs[3] = 1

# -----
# Labels
# -----
label_defs = [
    Label('bottle', rgb2bgr((0,255,0))),
    Label('can', rgb2bgr((0,0,0))),
    Label('bag of chips', rgb2bgr((255,0,0)))]

def main():
    # -----
    #get the object label to create the dataset
    # -----
    parser = argparse.ArgumentParser(description='Dataset')
    parser.add_argument('-cls', help='classes:' + str(label_defs), required=True)

    args = parser.parse_args()

    obj_label = int(args.cls)

    try:
        obj_name = label_defs[obj_label][0]
    except(IndexError) as e:
        print('The desired class dont exist, type -help', str(e))
        return 1

    print('Class :', obj_name)
    # -----
    #read the iteration from file

    f1 = open(dir_path + '/iteration.txt', 'r+')
    i = f1.read()
    print(i)

    # -----
    cap = cv2.VideoCapture(0)
    cv2.namedWindow('frame')
    cv2.setMouseCallback('frame',onMouse)
    save = 0
    fgbg = cv2.BackgroundSubtractorMOG2()

    while(True):
        f = open(dir_path + '/data/Dataset12/Annotations/%s.txt%i,w')
        _, frame = cap.read()
        src = frame
        mask = frame
        blur = cv2.GaussianBlur(frame, (15,15), 0)

```

```

blur1 = cv2.medianBlur(frame, 17)
hsv = cv2.cvtColor(blur1, cv2.COLOR_BGR2HSV)

fgmask = fgbg.apply(blur)
fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel1)
fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel1)

if flgs[0] == 1:
    values = hsv[x1, y1]
    print(values)
    flgs[0] = 0
    flgs[2] = 1
    cv2.setTrackbarPos('S1_MIN', 'thresh', values[1])
    cv2.setTrackbarPos('V1_MIN', 'thresh', values[2])

if flgs[2] == 1:
    S1min = cv2.getTrackbarPos('S1_MIN', 'thresh')
    V1min = cv2.getTrackbarPos('V1_MIN', 'thresh')

    lower = np.array([values[0]-5, S1min, V1min])
    upper = np.array([values[0]+5, 255, 255])

    mask = cv2.inRange(hsv, lower, upper)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel1)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
    thresh = mask

    final_mask = cv2.bitwise_not(mask, mask=fgmask)
    final_mask = cv2.morphologyEx(final_mask, cv2.MORPH_OPEN, kernel)
    final_mask = cv2.morphologyEx(final_mask, cv2.MORPH_CLOSE, kernel)

    contours, hier = cv2.findContours(final_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # get the bounding rect and display only the bigger contour,
    # in order to display only one rect
    if len(contours) > 0 and flgs[3] == 1:
        filename ='%s.jpg'%i
        cv2.imwrite(os.path.join(path , filename), src)
        c = max(contours, key = cv2.contourArea)
        x, y, w, h = cv2.boundingRect(c)
        # draw a green rectangle to visualize the bounding rect
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        f.write(str(x) + ', ' + str(y) + ', ' + str(x+w) + ', ' + str(y+h)+ ', ' + str(obj_name))
        f1.seek(0)
        f1.write(str(i))
        i=int(i)+1

    #cv2.imshow('mask', final_mask)
    cv2.imshow('thresh', thresh)

cv2.imshow('frame', frame)
cv2.imshow('fgmask', fgmask)
if cv2.waitKey(100) & 0xFF == 27:
    break

f.close()
f1.close()

```

```
        cap.release()
        cv2.destroyAllWindows()

if __name__ == '__main__':
    sys.exit(main())
```

