



UNIVERSIDADE FEDERAL DE PERNAMBUCO  
CENTRO DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Roberto Costa Fernandes

**Embedded Object Detection and Position Estimation for RoboCup Small Size  
League**

Recife

2023

Roberto Costa Fernandes

**Embedded Object Detection and Position Estimation for RoboCup Small Size League**

A M.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

**Concentration Area:** Computer Engineering

**Advisor:** Edna Natividade da Silva Barros

Recife

2023

Catálogo na fonte  
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

F363e      Fernandes, Roberto Costa  
              *Embedded object detection and position estimation for RoboCup Small Size League* / Roberto Costa Fernandes – 2023.  
              82 f.: il., fig., tab.

              Orientadora: Edna Natividade da Silva Barros.  
              Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2023.  
              Inclui referências.

              1. Engenharia da computação. 2. Robótica. 3. RoboCup. I. Barros, Edna Natividade da Silva (orientadora). II. Título

              621.39              CDD (23. ed.)              UFPE - CCEN 2023 – 100

**Roberto Costa Fernandes**

**“Embedded Object Detection and Position Estimation for  
RoboCup Small Size League”**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia da Computação.

Aprovado em: 15/03/2023

**BANCA EXAMINADORA**

---

Prof. Dr. Cleber Zanchettin  
Centro de Informática / UFPE

---

Prof. Dr. Filipe Cordeiro Rolim  
Departamento de Computação / UFRPE

---

Profa. Dra. Edna Natividade da Silva Barros  
Centro de Informática / UFPE  
**(Orientadora)**

## **ACKNOWLEDGEMENTS**

I would like to thank all my family, who guided me through my life, leading me to who I am now, and always doing all to give me all the best. Also, I would like to thank my parents, Ana and José, and my sisters, Gabriela, Roberta, Danielle, and Janaina, for always putting me and my education first and providing me with the best home education and school education.

I would like to thank my advisor, Prof. Edna Barros. Edna was present during my whole bachelor's and master's, supporting all my projects which gave me the opportunity to be here and be who I am today. Edna also led me and motivated me on my worse days. Without her support wouldn't be possible to make this work. Edna taught me with her example to put love and passion into all I do during my life and be a good role model to future generations.

I also would like to thank my friends Walber Macedo (Binho), Lucas Henrique (Lukita), and João Guilherme, who helped me make some project decisions, listening to my ideas and their knowledge. Besides, they were also a point of support and inspiration to keep me going on the bad days.

I also would like to thank all my friends who always supported me and helped me through my university years. I would like to thank all my colleagues from RobôCIn, who became my friends and family. Being part of RobôCIn was the best opportunity I had during the university. It taught me all the qualities and skills I would need during my professional life.

## ABSTRACT

In the RoboCup Small Size League (SSL), there is the challenge of giving more autonomy to the robots, so they can perform some tasks without receiving any external information. To achieve this autonomy, the robot has to detect and estimate the position of other objects on the field so it can score goals and move without colliding with other robots. Object detection models often use monocular images as the input, but calculating the relative position of an object given a monocular image is quite challenging as the image doesn't have any information on the object's distance. The main objective of this work is to propose a complete system to detect an object on the field and locate it using only a monocular image as the input. The first obstacle to producing a model to object detection in a specific context is to have a dataset labeling the desired classes. In RoboCup, some leagues already have more than one dataset to train and evaluate a model. Thus, this work presents an open-source dataset to be used as a benchmark for real-time object detection in SSL. Using this dataset, this work also presents a pipeline to train, deploy, and evaluate Convolutional Neural Networks (CNNs) models to detect objects in an embedded system. Combining this object detection model with the global position received from the SSL-Vision, this work proposes a Multilayer Perceptron (MLP) architecture to estimate the position of the objects giving just an image as the input. In the object detection dataset, the MobileNet v1 SSD achieves 44.88% AP for the three detected classes at 94 Frames Per Second (FPS) while running on a SSL robot. And the position estimator for a detected ball achieves a Root Mean Square Error (RMSE) of 34.88mm.

**Keywords:** position estimation; deep learning; object detection; robotics; computer vision.

## RESUMO

A categoria Small Size League (SSL) da RoboCup tem o desafio de aumentar o nível de autonomia dos robôs para que eles possam realizar algumas tarefas sem receber nenhuma informação externa. Para garantir essa autonomia o robô tem que ser capaz de detectar e estimar a posição dos objetos no campo, para que ele possa marcar gols e se movimentar sem colidir com outros robôs. Modelos para detecção de objetos geralmente utilizam imagens monoculares como entrada, no entanto é desafiante calcular a posição relativa desses objetos, já que a imagem monocular não tem nenhuma informação da distância. O principal objetivo dessa dissertação é propor um sistema completo para detectar um objeto e calcular sua posição relativa no campo, usando uma imagem monocular como entrada. O primeiro obstáculo para treinar um modelo para detectar objetos em um contexto específico é ter um dataset de treinamento com imagens anotadas. Outras categorias da RoboCup já possuem dataset com imagens anotadas para treinar e avaliar um modelo. Assim, esse trabalho também propõe um dataset para a categoria SSL para ser usado como referência de comparação para detecção de objetos nessa categoria. Utilizando esse dataset, esse trabalho apresenta um fluxo para treinar, avaliar e realizar a inferência de uma Convolutional Neural Networks (CNNs) para detecção de objetos em um sistema embarcado. Combinando a detecção de objetos com a posição global recebida do SSL-Vision, esse trabalho ainda propõe uma arquitetura baseada em Multilayer Perceptron (MLP) para estimar a posição dos objetos usando somente a imagem monocular como entrada. Na detecção de objetos, o modelo MobileNet v1 SSD alcançou 55.77% AP para as três classes de interesse rodando a 94 Frames Per Second (FPS) em um robô de SSL. O modelo para estimar a posição de um objeto da classe Bola atingiu um Root Mean Square Error (RMSE) de 34.88mm.

**Palavras-chave:** estimacão de posicão; apredizado profundo; detecção de objetos; robótica; visão computacional.

## LIST OF FIGURES

Figure 1 – Sensing and acting scheme for AMRs. . . . .	16
Figure 2 – Examples of RoboCup leagues: (a) a typical SSL game. Image provided by the author; (b) a game from Middle Size League (MSL). Source: (XIAO et al., 2017); (c) a NAO robot on the Standard Platform League (SPL). . . .	17
Figure 3 – Overview of SSL-Vision, the global vision system used in SSL. . . . .	18
Figure 4 – Perceptron model, where $x_0, x_1 \dots x_n$ are the inputs, $w_0, w_1 \dots w_n$ are the weight of each input and $b$ is the bias. . . . .	22
Figure 5 – Sigmoid function . . . . .	23
Figure 6 – Hyperbolic Tangent function . . . . .	23
Figure 7 – ReLU function . . . . .	24
Figure 8 – ReLU - 6 function . . . . .	25
Figure 9 – Differences between <i>AND</i> , <i>OR</i> that their outputs can be separable by a line, and <i>XOR</i> , that does not exist such a line that can separate true and false outputs. . . . .	26
Figure 10 – MLP showing the Input Layer, the Hidden Layer, and the Output Layer. $x_1, x_2 \dots x_n$ are the inputs of the model, $w_{11}, w_{21} \dots w_{4n}, w_{211} \dots w_{2m4}$ are the weights and $y_1 \dots y_m$ are the outputs. . . . .	26
Figure 11 – Comparison between underfitted, overfitted, and perfect fitted function approximation. . . . .	31
Figure 12 – Example of convolution in an image matrix. . . . .	33
Figure 13 – Example result of a max pooling and average pooling layer. . . . .	33
Figure 14 – Differences between classifying only one object on an image and detecting multiple objects on an image. . . . .	35
Figure 15 – Example of a YOLO output containing the bounding boxes for each cell with its class probability and the final predicted bounding box. . . . .	36
Figure 16 – Standard Convolution (a) replaced by Depthwise Convolution (b) and Point-wise Convolution(c) . . . . .	37
Figure 17 – DSC block used in MobileNet v1 . . . . .	37
Figure 18 – MobileNet v1 architecture to image classification. . . . .	38
Figure 19 – Bottleneck Residual block used in MobileNet v2 . . . . .	39



Figure 20 – MobileNet v2 architecture to image classification. $t$ is the expansion factor, $c$ is the number of output channels, $n$ is the number of layers using this configuration, and $s$ is the stride of each layer. . . . .	40
Figure 21 – MobileNet v1 architecture for object detection using SSD framework . . . .	41
Figure 22 – MobileNet v2 architecture for object detection using SSD framework . . . .	41
Figure 23 – Fused Inverted Bottleneck layer, where the first Pointwise and the Depthwise convolution were fused on a $K \times K$ regular convolution. This regular convolution expands the $H_1 \times W_1 \times C_1$ input by the expansion factor, $s > 1$ , and outputs a $H_2 \times W_2 \times C_2$ block. . . . .	42
Figure 24 – Tucker Layer similar to DSC, but the first Pointwise convolution reduces the input size by the input compression factor $s < 1$ and the $K \times K$ regular convolution instead of the Depthwise convolution. This regular convolution changes the number of filters from $s \times C_1$ to $e \times C_2$ , with an output compression factor $e < 1$ . . . . .	42
Figure 25 – Best MobileDet architecture to EdgeTPU. $s$ refers to the stride, $e$ refers to the expansion factor, and 0.25 – 0.75 on the Tucker layer refers to the input and output compression factors. . . . .	43
Figure 26 – Pipeline for the proposed approach followed on this work. . . . .	49
Figure 27 – Pipeline for the proposed approach to building a model to detect SSL relevant objects. . . . .	49
Figure 28 – Pipeline for the proposed approach to building a model to estimate SSL relevant objects position. . . . .	50
Figure 29 – Pipeline used to create the dataset. . . . .	50
Figure 30 – Sample images from the dataset, showing ground-truth detection. The leftmost column (a) has images from public SSL images, the middle column (b) has images collected from a smartphone inside the field, and the rightmost column (c) has images taken in a camera placed in the robot. . . . .	51
Figure 31 – Examples of (a) iconic images and (b) non-iconic images present in the proposed dataset. . . . .	53
Figure 32 – Number of categories per image. . . . .	53
Figure 33 – Number of instances per image. . . . .	54

Figure 34 – Instance division per class and size. Small objects have less than $32 \times 32$ (1024) pixels, medium objects have a size between $32 \times 32$ (1024) pixels and $96 \times 96$ (9216) pixels, and large objects are bigger than $96 \times 96$ (9216) pixels. . . . .	55
Figure 35 – Pipeline used to train a model and validate the robot. . . . .	56
Figure 36 – Graphical representation on how to calculate IoU of two bounding boxes. . . . .	57
Figure 37 – Modifications on the architecture and data flow of the new robot. . . . .	59
Figure 38 – Pipeline for creating the dataset for position estimation. . . . .	60
Figure 39 – Some examples of the dataset containing detected balls . . . . .	61
Figure 40 – Ball's relative position to the robot with the camera. Consider the robot located on position (0,0) and having looking to the same direction as the X-axis . . . . .	62
Figure 41 – Predicted and ground truth positions for MLP architecture without using normalization . . . . .	70
Figure 42 – Predicted and ground truth positions for MLP architecture normalizing the input . . . . .	71
Figure 43 – Predicted and ground truth positions for MLP architecture normalizing the input and the output . . . . .	71

## LIST OF TABLES

Table 1 – Sigmoid, Tanh, ReLU, and ReLU - 6 activation functions and their derivative.	30
Table 2 – Hyperparameters modifications on Mobilenet v1 SSD, MobileNet v2 SSD and MobileDet models. . . . .	44
Table 3 – Number of images divided into train and test set. . . . .	52
Table 4 – Number of instances of each class in the dataset. . . . .	52
Table 5 – Number of each instance size in the dataset. . . . .	55
Table 6 – Hyperparameters used on the Grid search to find the best MLP architecture.	63
Table 7 – Average Precision (AP) for each model by Intersection over Union (IoU) threshold, in $AP_{50}$ the threshold used is 0.5 and $AP_{75}$ is 0.75, and detected object area, where $AP_S$ , $AP_M$ , and $AP_L$ stands for the result by each detection size, Small, Medium or Large. . . . .	65
Table 8 – Average Precision (AP) for each model by classes, where $AP_{Ball}$ , $AP_{Robot}$ and $AP_{Goal}$ is the Average Precision for each of class Ball, Robot and Goal respectively. . . . .	66
Table 9 – Average Recall (AR) for each model by maximum detection per image, $AR_1$ for at most 1 object per image and $AR_{10}$ for 10 objects per image, and detected object area, where $AR_S$ , $AR_M$ , and $AR_L$ stands for the result by each detection size, Small, Medium or Large. . . . .	66
Table 10 – Mean inference frequency in Frames Per Second (FPS) for each tested model.	67
Table 11 – Average Precision (AP) for each model using the dataset dividing the instances by size. . . . .	67
Table 12 – K-Means results to find the best aspect ratio configuration for the anchors.	68
Table 13 – Average Precision (AP) for each model using the best anchors aspect ratio using k-means. . . . .	68
Table 14 – Average Precision (AP) for each model using the dataset dividing the instances by size and the best anchors aspect ratio using k-means. . . . .	68
Table 15 – Average Precision (AP) for each model separated by all the approaches tested on this work. . . . .	69
Table 16 – RMSE for the proposed architecture compared with the results from (MELO; BARROS, 2022) . . . . .	71

Table 17 – Inference time for the proposed architecture . . . . . 72

## LIST OF ABBREVIATIONS AND ACRONYMS

<b>AMRs</b>	Autonomous Mobile Robot
<b>AP</b>	Average Precision
<b>AR</b>	Average Recall
<b>Avg Pool</b>	Average Pooling
<b>CNNs</b>	Convolutional Neural Networks
<b>COCO</b>	Common Objects in Context
<b>CPS</b>	Cyber-Physical Systems
<b>DSC</b>	Depthwise Separable Convolution
<b>FC</b>	Fully Connected
<b>FPS</b>	Frames Per Second
<b>GD</b>	Gradient Descent
<b>GPUs</b>	Graphical Processing Units
<b>IBNs</b>	Inverted Bottlenecks
<b>ILSVRC</b>	ImageNet Large Scale Visual Recognition Challenge
<b>IoT</b>	Internet of Things
<b>IoU</b>	Intersection over Union
<b>IPM</b>	Inverse Perspective Mapping
<b>LiDAR</b>	Light Detection And Ranging
<b>ML</b>	Machine Learning
<b>MLP</b>	Multilayer Perceptron
<b>MSL</b>	Middle Size League
<b>NAS</b>	Neural Architecture Search
<b>NNs</b>	Neural Networks
<b>ReLU</b>	Rectified Linear Unit
<b>ResNets</b>	Residual Nets

<b>RMSE</b>	Root Mean Square Error
<b>SGD</b>	Stochastic Gradient Descent
<b>SPL</b>	Standard Platform League
<b>SSD</b>	Single Shot Detection
<b>SSL</b>	Small Size League
<b>SVR</b>	Support Vector Regression
<b>Tanh</b>	Hyperbolic Tangent

## CONTENTS

<b>1</b>	<b>INTRODUCTION . . . . .</b>	<b>16</b>
<b>2</b>	<b>THEORETICAL BACKGROUND . . . . .</b>	<b>21</b>
2.1	NEURAL NETWORKS . . . . .	21
<b>2.1.1</b>	<b>Neuron Models . . . . .</b>	<b>21</b>
2.1.1.1	<i>Sigmoid</i> . . . . .	22
2.1.1.2	<i>Hyperbolic Tangent (Tanh)</i> . . . . .	23
2.1.1.3	<i>Rectified Linear Unit (ReLU)</i> . . . . .	24
<b>2.1.2</b>	<b>Multilayer Perceptron . . . . .</b>	<b>25</b>
<b>2.1.3</b>	<b>Training . . . . .</b>	<b>27</b>
2.2	CONVOLUTIONAL NEURAL NETWORKS . . . . .	31
<b>2.2.1</b>	<b>Image Classification . . . . .</b>	<b>34</b>
<b>2.2.2</b>	<b>Object Detection . . . . .</b>	<b>35</b>
2.3	OBJECT DETECTION MODELS . . . . .	36
<b>2.3.1</b>	<b>MobileNet v1 . . . . .</b>	<b>36</b>
<b>2.3.2</b>	<b>MobileNet v2 . . . . .</b>	<b>38</b>
<b>2.3.3</b>	<b>MobileNets SSD . . . . .</b>	<b>39</b>
<b>2.3.4</b>	<b>MobileDets . . . . .</b>	<b>41</b>
<b>2.3.5</b>	<b>Training MobileNets and MobileDets . . . . .</b>	<b>43</b>
<b>2.3.6</b>	<b>YOLO v4 tiny . . . . .</b>	<b>44</b>
2.4	METRICS . . . . .	44
<b>3</b>	<b>RELATED WORK . . . . .</b>	<b>45</b>
3.1	GENERAL PURPOSE DATASETS . . . . .	45
3.2	RELATIVE POSITION ESTIMATION . . . . .	46
3.3	OBJECT DETECTION IN ROBOCUP . . . . .	47
<b>4</b>	<b>PROPOSED APPROACH . . . . .</b>	<b>49</b>
4.1	OBJECT DETECTION DATASET . . . . .	50
<b>4.1.1</b>	<b>Dataset Creation . . . . .</b>	<b>50</b>
<b>4.1.2</b>	<b>Dataset Statistics . . . . .</b>	<b>52</b>
4.2	TRAINING OBJECT DETECTION MODELS . . . . .	55
4.3	RUNNING AND EVALUATING . . . . .	57

4.4	INFERENCE ENVIRONMENT . . . . .	58
4.5	DATASET TO ESTIMATE THE POSITION . . . . .	59
4.6	TRAINING EVALUATING POSITION MODEL . . . . .	62
4.7	EVALUATING POSITION DATASET . . . . .	64
<b>5</b>	<b>RESULTS . . . . .</b>	<b>65</b>
5.1	OBJECT DETECTION RESULTS . . . . .	65
5.2	POSITION ESTIMATOR RESULTS . . . . .	69
<b>6</b>	<b>CONCLUSION . . . . .</b>	<b>73</b>
	<b>REFERENCES . . . . .</b>	<b>75</b>



# 1 INTRODUCTION

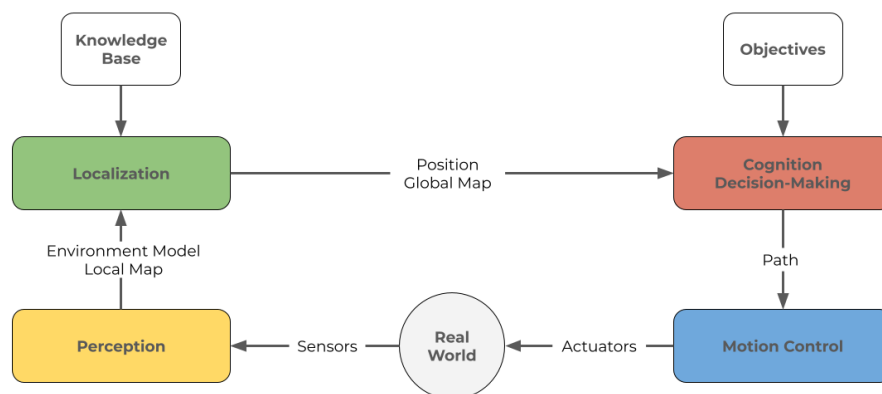
The Fourth Industrial Revolution, or Industry 4.0, focuses on increasing autonomy in the industrial production chain (KAGERMANN; WAHLSTER; HELBIG, 2013). This evolution is possible via Cyber-Physical Systems (CPS), embedded systems capable of communicating and cooperating, converging the virtual and physical worlds. Industry 4.0 has four design principles (HERMANN; PENTEK; OTTO, 2016): Interconnection, Information Transparency, Decentralized Decisions, and Technical Assistance.

Interconnection allows machines, devices, sensors, and people to communicate with each other, Internet of Things (IoT) emerge from this principle. With this communication, they can share all information they can access, achieving Information Transparency. Furthermore, having access to all the environmental information allows Decentralized Decisions for better decision-making. Thus, CPS can help humans perform unsafe tasks, implying Technical Assistance.

One technology that is a building block for Industry 4.0 is Autonomous Mobile Robot (AMRs) (RÜSSMANN et al., 2015). AMRs relies in four pillars (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011): Perception, Localization, Cognition, and Motion Control. This concept means that a robot must use its sensors to perceive the world and build its world model to locate and understand the environment. With this information and its objective, the robot plans its desired path to use its motion control algorithm to follow this path. Figure 1 shows these pillars and steps.

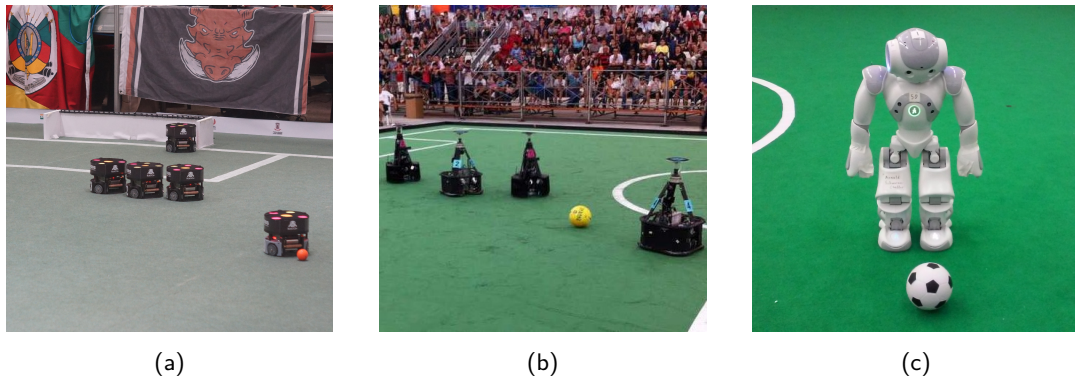
To move this research field further, RoboCup was created in 1997 to focus on collaborative mobile robots to solve dynamic problems (KITANO et al., 1997). This competition consists of

Figure 1 – Sensing and acting scheme for AMRs.



Source: Adapted from: (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011)

Figure 2 – Examples of RoboCup leagues: (a) a typical SSL game. Image provided by the author; (b) a game from Middle Size League (MSL). Source: (XIAO et al., 2017); (c) a NAO robot on the Standard Platform League (SPL).



Source: (STANDARD PLATFORM LEAGUE TECHNICAL COMMITTEE, 2021)

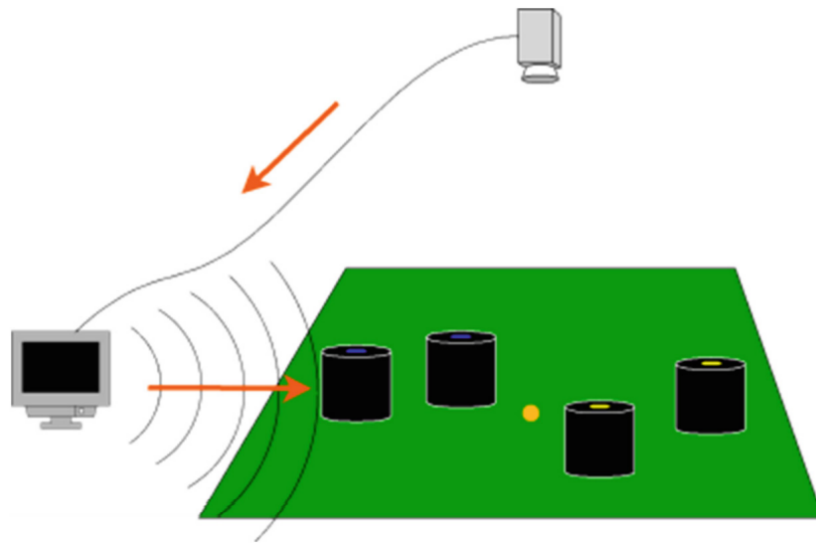
a robot team autonomously playing soccer against another team and aims to win a game against the FIFA World Cup champions of 2050 (BURKHARD et al., 2002). This competition has some leagues, each trying to develop technologies such as design principles of AMRs, real-time decision-making, computer vision, and sensor fusion.

Besides being important in robotics, those fields of study also have crucial contributions in other areas. For instance, autonomous cars use computer vision to detect objects and plan the best route without colliding with any obstacle. Besides those cross contributions, some leagues on RoboCup aim to develop domestic robots to help the elderly and people with disability.

Small Size League (SSL) is one of the most traditional leagues in the RoboCup. In this league, the robots are constrained to fit in a cylinder with a height of 15cm and a diameter of 18cm (SMALL SIZE LEAGUE TECHNICAL COMMITTEE, 2021), making it possible to precisely perform a wide range of dynamic plays every time moment during a game. The decision-making process during each play needs to be fast due to the game speed, in which robots usually move at  $3m/s$ , and the ball reaches  $6.5m/s$ . A typical SSL game occurs with two teams with eleven robots each, using an orange golf ball in a field of  $12m \times 9m$  (SMALL SIZE LEAGUE TECHNICAL COMMITTEE, 2021). Figure 2a shows a image from an SSL game.

The robots in SSL are semi-autonomous (FERREIN; STEINBAUER, 2016) due to the use of an external global vision system, the SSL-Vision (ZICKLER et al., 2010). The SSL-Vision uses a camera placed above the field to locate robots and balls using color segmentation, as each robot has different color tags above it, and the ball is orange. For example, on Figure 2a, the tag above the robot is visible, where the central tag indicates the team color, yellow or blue, and the other four tags identify the robot. Figure 3 show an overview of the SSL-Vision.

Figure 3 – Overview of SSL-Vision, the global vision system used in SSL.



Source: (WEITZENFELD et al., 2015)

In other RoboCup leagues, as MSL (MIDDLE SIZE LEAGUE TECHNICAL COMMITTEE, 2021) and SPL (STANDARD PLATFORM LEAGUE TECHNICAL COMMITTEE, 2021) instead of using external information, each robot has its camera and vision system, limiting the information to which they have access. Thus, they are considered a fully autonomous system because each robot can perform a tactic without receiving external information. A MSL robot fits in  $52 \times 52 \times 80\text{cm}$ , and a SPL uses the NAO Robot as the standard platform. Figure 2b shows an example of a MSL robot and Figure 2c shows a NAO used in SPL.

The future of the SSL and MSL is to merge in one category as the research in robotics advances (BURKHARD et al., 2002). Thus, a technical challenge (SMALL SIZE LEAGUE TECHNICAL COMMITTEE, 2020) was introduced in 2019 to evolve the SSL to bring autonomy to a SSL robot in a similar way to MSL and SPL. In this technical challenge, a SSL robot has to execute four skills without receiving external information from the SSL-Vision. These skills are:

1. Grab a stationary ball somewhere on the field;
2. Score with the ball on an empty goal;
3. Score with the ball on a statically defended goal;
4. Move the robot to specific coordinates;

A SSL robot, to perform the first three steps of this challenge, has to detect a Robot, a Ball, and a Goal autonomously and calculate their position relative to it. This technical

challenge encourages the teams to develop and propose a local vision system following the league's requirements. As a result, it brings autonomy to an SSL robot in a similar way to MSL and SPL. However, a MSL can equip a full-size computer, and a SPL robot can not be modified since it uses a standard platform, and this challenge keeps all the size constraints, except for the height restriction. So, robots can be taller than usual, creating a small room for hardware improvements.

The straightforward option to detect these objects uses scan lines and color segmentation to detect the ball (Seel, Fabio and Jut, Sabolc, 2019), as the league uses an orange golf ball. However, this approach can not detect robots and goals because they do not have a unique pattern. For instance, a team can use robots of any color, which makes it harder to use this technique. Besides, the color segmentation approach must be re-calibrated on each slight environment variation, as uneven illumination or field changes (NEVES et al., 2011).

The state-of-the-art object detection relies on Convolutional Neural Networks (CNNs) (BOCHKOVSKIY; WANG; LIAO, 2020), which, given a labeled dataset, trains a model once and does not need any other calibration or modification. Besides, this approach is robust to deal with occlusion, scale transformation, and background switch (ZHAO et al., 2019), which makes Convolutional Neural Networks (CNNs) strong candidates to use in the SSL. For other RoboCup's leagues, like SPL (ALBANI et al., 2016), and MSL (Luo et al., 2017), there are public object detection datasets.

After detecting the objects on the field, a SSL robot has to add the detected object to its world model to start planning its movements toward its objective. To do so, the robot needs a reasonable estimation of each detected object's relative position to itself. One possibility could be adding a distance sensor as a Light Detection And Ranging (LiDAR) to combine with the detected object and measure the object distance. However, the size constraints of a SSL robot made it harder to add these sensors to the robot because it would have to be located near the ground to use its full range and measure the distance of small objects as the ball. Although, a SSL field has its global vision that could be used with the detected objects to train a model to estimate those distances.

Therefore, given the SSL technical challenge, the league constraints, and the lack of an open-sourced dataset, this work has as its main objective:

- Build a complete embedded system on a SSL robot, able to detect relevant objects on this league and estimate the ball position, respecting the robot's size constraints.

To achieve this main objective, our work has the following specific objectives:

- Create a dataset for object detection to be a comparison baseline on SSL.
- Compare object detection models that could be able to run in real-time (24 Frames Per Second (FPS)) on a SSL robot.
- Create a dataset for estimating the ball position on a SSL field.
- Evaluate the model to estimate the ball's position using monocular vision.

## 2 THEORETICAL BACKGROUND

This chapter will present some basic concepts that were used in this thesis. Section 2.1 will explain the concepts of Neural Networks (NNs), including the basics of how a neuron work and how this idea evolved while research advanced. Section 2.2 will present how the initial idea of a neuron got more power using convolution operations and which applications took advantage of this approach. Section 2.3 shows some of the classical CNNs models. Section 2.4 present some metrics used to evaluate a Machine Learning (ML) model.

### 2.1 NEURAL NETWORKS

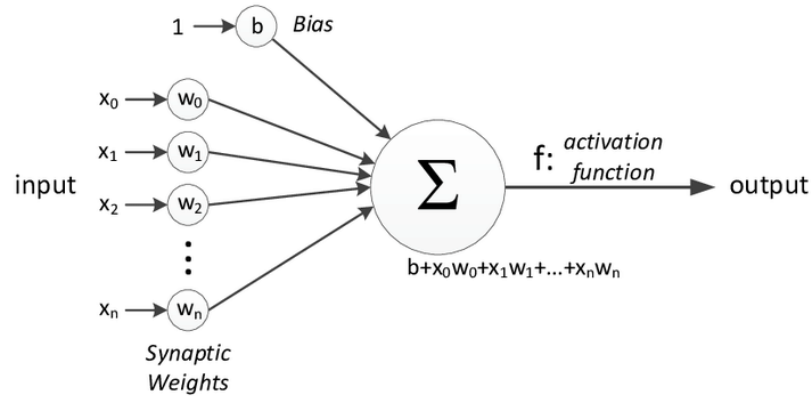
#### 2.1.1 Neuron Models

The human being is always trying to understand better how the body of a living creature works. One of the most used approaches is mathematical models making it easy to replicate a body's behavior. For instance, a human arm's joints are studied to create a kinematics model of it and use it to control robotics arms (STROEVE, 1999; GOMI; OSU, 1998; NAKAMURA et al., 2005). Another body system with a high interest in building mathematical models to help replicate and understand how it works is the nervous system.

The nervous system is based on neurons, a type of cell that communicates with others by electrical signals, known as synapse (BEAR; CONNORS; PARADISO, 2020). The first mathematical model of a biological neuron was proposed by MCCULLOCH; PITTS in 1943 (MCCULLOCH; PITTS, 1943). This model considers a neuron with multiple Boolean inputs, 0 or 1, where these inputs could be applied to a logical *NOT* before entering the neuron. The output of this neuron model is given by using logical operators *AND* and *OR* to the inputs. Then comparing this result with a threshold, if this value is greater than the threshold, it outputs 1, otherwise outputs 0.

This model was fundamental and inspired other models, but it has some main flaws, as it does not accept real value inputs and uses the same importance for all inputs. To address the problems of the MCCULLOCH; PITTS model, in 1958, the Perceptron model was proposed by ROSENBLATT (ROSENBLATT, 1958). The Perceptron model is similar to the MCCULLOCH; PITTS model, which has multiple inputs and only one output. However, the inputs and the output are real values, and the inputs are multiplied by weights to have distinctive importance.

Figure 4 – Perceptron model, where  $x_0, x_1 \dots x_n$  are the inputs,  $w_0, w_1 \dots w_n$  are the weight of each input and  $b$  is the bias.



Source: (FOUNTAS, 2021)

Thus, this model outputs the addition of the inputs and a bias. Figure 4 shows the Perceptron model.

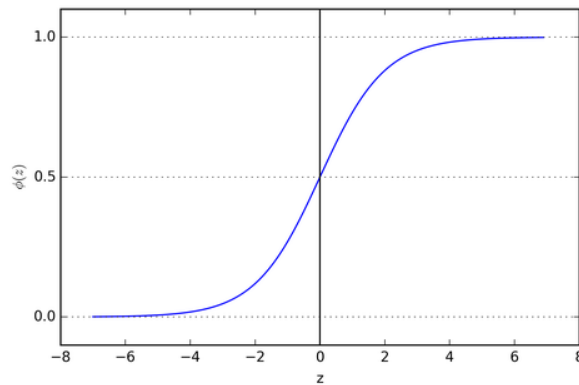
When using the MCCULLOCH; PITTS models, the output of a neuron can be thinking as an output applied to a step function. The result of this function is always mapped to be 0 or 1, which limits this model to handle binary problems. Besides, the Perceptron model puts an activation function before the output value. This activation function uses the previous result to calculate the neuron output. These activation functions have the restriction of being differentiable and monotonic. During the years, some functions were proposed to be used on the perceptron model, such as Sigmoid, Rectified Linear Unit (ReLU), and Hyperbolic Tangent (Tanh).

#### 2.1.1.1 Sigmoid

The Sigmoid function is a non-linear function that is graphically similar to the step function used in the MCCULLOCH; PITTS model. However, while the step function outputs 0 or 1, the Sigmoid function outputs any real value between 0 and 1. This function is widely used in neurons projected to output a probability. Figure 5 shows the graphical representation of the Sigmoid function and given an input  $x$ . The Sigmoid function is defined as:

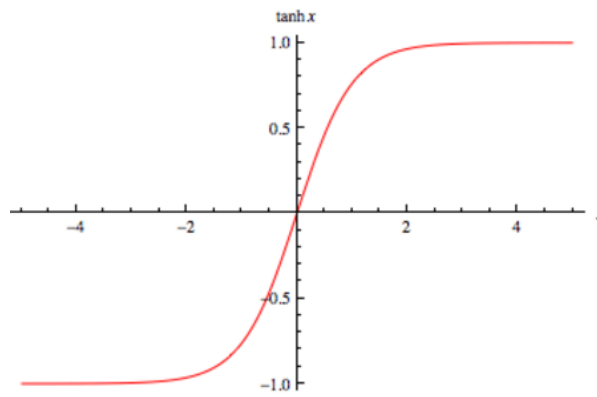
$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.1)$$

Figure 5 – Sigmoid function.



Source: Collected from the internet.<sup>1</sup>

Figure 6 – Hyperbolic Tangent function.



Source: Collected from the internet.<sup>2</sup>

#### 2.1.1.2 Tanh

The Tanh function has the same s-shape as the Sigmoid function, but it is slightly different as its output is in the  $(-1, 1)$  range instead of  $(0, 1)$ . This function is considered better than the regular Sigmoid because it is centered in origin. Its derivative has bigger values than the Sigmoid derivative, which will help the learning process (LECUN et al., 2012). Figure 6 shows the Tanh graphical representation. The Tanh function is defined as:

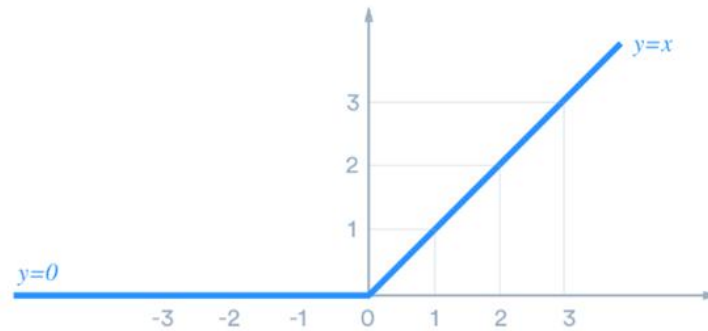
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.2)$$

<sup>1</sup> <<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>>. Accessed July 14, 2021

<sup>2</sup> <<https://www.i2tutorials.com/explain-hyperbolic-tangent-or-tanh-relu-rectified-linear-unit>>. Accessed July 14, 2021



Figure 7 – ReLU function.



Source: Collected from the internet.<sup>3</sup>

### 2.1.1.3 ReLU

Sigmoid and Tanh were widely used, keeping the output in a well-defined range, but they have two main problems. First, these functions are complex to implement because they use exponential functions internally. Furthermore, these functions saturate when the input is large or small, but they are sensitive to inputs close to 0. This saturation means that when the input is too large or too small, any change in it does not change the output. However, any slight difference in inputs close to 0 significantly impacts the output. Sigmoid and Tanh functions also have some problems with the training process, which will be discussed later.

ReLU (NAIR; HINTON, 2010) emerges as an activation function to address these problems. ReLU is defined as:

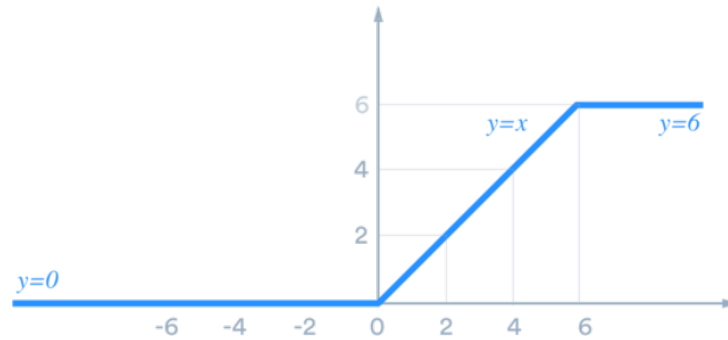
$$f(x) = \max(0, x) \quad (2.3)$$

This function returns the input if the input is more significant than zero and returns zero otherwise. Figure 7 shows the graphical representation of this function. This function is more straightforward to implement than Sigmoid and Tanh, as they need to apply a threshold to the neuron output. Besides, the linearity for values greater than zero avoids the saturation problems from other activation functions.

ReLU started to be the most used activation function. Thus some variations of it started to be used (MAAS et al., 2013) (KRIZHEVSKY; HINTON, 2010). One of those is the ReLU - 6 (KRIZHEVSKY; HINTON, 2010), which sets a maximum value of 6 to the output of the traditional

<sup>3</sup> <<https://www.programmersought.com/article/81614868733/>>. Accessed July 14, 2021

Figure 8 – ReLU - 6 function.

Source: Collected from the internet. <sup>4</sup>

one. This new restriction to ReLU made the neuron learn sparse features faster. Figure 8 shows the graphical representation of ReLU - 6. ReLU - 6 is defined as:

$$f(x) = \max(\max(0, x), 6) \quad (2.4)$$

### 2.1.2 Multilayer Perceptron

The model presented by ROSENBLATT in 1958 was based on a single neuron, which limits its application. In 1969, MARVIN; SEYMOUR stated in their book that a single Perceptron only works for linear separable functions and wouldn't work for complex functions (MARVIN; SEYMOUR, 1969). For instance, this model does not handle simple non-linear functions such as *XOR*. Figure 9 shows the differences between linear functions, such as *AND* and *OR*, and non-linear functions, such as *XOR*.

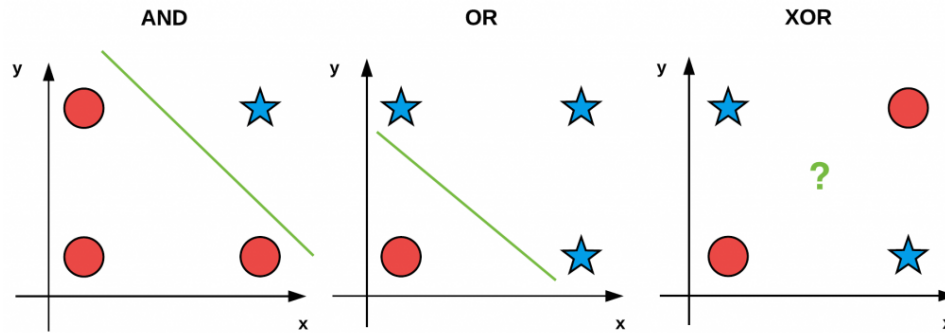
This book greatly impacted Artificial Intelligence research in the 1970s and early 1980s. In these years, researchers stopped using connectionism approaches and focused on symbolic models. However, this scenario started to change in the 1980s when some works began to prove that the book from MARVIN; SEYMOUR had some problems not well described. For example, CYBENKO proved that a NNs with at least one hidden layer could approximate any non-linear function, such as *XOR* (CYBENKO, 1989).

Multilayer Perceptron (MLP) is the idea of combining multiple layers of Perceptrons, where the output of one layer is the input of the next layer. A MLP consists of one Input Layer,

<sup>4</sup> <<https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>>. Accessed July 14, 2021

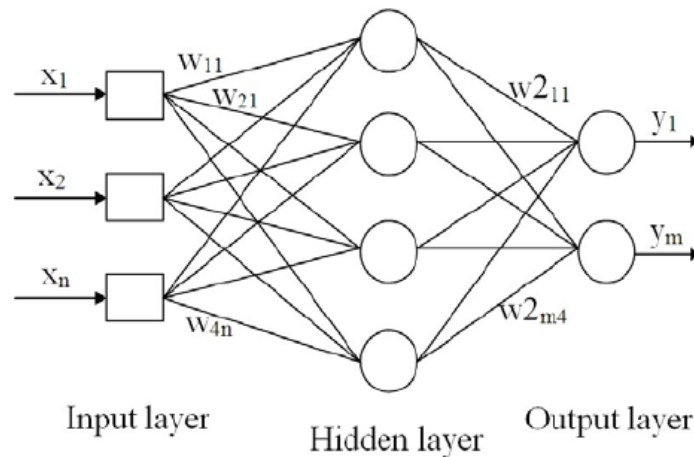
<sup>5</sup> <<https://www.pyimagesearch.com/2021/05/06/implementing-the-perceptron-neural-network-with-python/>>. Accessed July 14, 2021

Figure 9 – Differences between *AND*, *OR* that their outputs can be separable by a line, and *XOR*, that does not exist such a line that can separate true and false outputs.



Source: Collected from the internet. <sup>5</sup>

Figure 10 – MLP showing the Input Layer, the Hidden Layer, and the Output Layer.  $x_1, x_2, \dots, x_n$  are the inputs of the model,  $w_{11}, w_{21}, \dots, w_{4n}, w_{2_{11}}, \dots, w_{2_{m4}}$  are the weights and  $y_1, \dots, y_m$  are the outputs.



Source: (MOKHTAR; MOHAMAD-SALEH, 2013)

multiple Hidden Layers, and one Output Layer (RUSSELL; NORVIG, 2002). The Input Layer has many inputs as the problem need. For instance, if the input is an image of  $32 \times 32 \times 3$  pixels, the model receives 3072 input signals. The Hidden Layers combine one or more hidden units in multiple layers. The Output Layer is responsible for interpreting the signals received from the Hidden Layers and outputs one or more values as output. Figure 10 shows a MLP model with only one hidden layer.

MLP were proposed by ROSENBLATT together with the Perceptron work. However, it wasn't used at first because of the difficulty of updating the weight on the Hidden Layers (RUSSELL; NORVIG, 2002). In 1986, it was proposed the usage of backpropagation algorithm (LINNAINMAA, 1970) to train MLP, giving new possibilities to Perceptron model (RUMELHART; HINTON; WILLIAMS, 1986).

### 2.1.3 Training

The most important thing about NNs is to learn a function from examples without any given rule. This section will show how to train a NNs and uses the book from MITCHELL as reference (MITCHELL, 1997). Then, it will start talking about the simplest way to train a Perceptron, passing by more complete alternatives as Gradient Descent (GD) and Stochastic Gradient Descent (SGD), to get on the Backpropagation algorithm used in MLP models.

Let's consider a set of training examples  $D$ , where each is represented by the input  $\vec{x} = (x_1, \dots, x_n)$  and the target output  $t$ . The Perceptron training rule to update each weight element of a given Perceptron  $i$ , given its weights vector  $\vec{w} = (w_1, \dots, w_n)$  and the predicted output  $o$  is:

$$w_i = w_i + \Delta w_i \quad (2.5)$$

where,

$$\Delta w_i = \eta(t - o)x_i \quad (2.6)$$

On (2.6),  $\eta$  is the learning rate that regulates how much the weights are modified at each step. Using these two equations, the training process of a Perceptron initialized with random outputs converges in a finite number of steps under two circumstances: the training examples are linearly separable, and the learning rate is sufficiently low (MARVIN; SEYMOUR, 1969).

Although Perceptron does not fit non-linear functions, it can be used to approximate such functions. The previous training rule cannot be used for this task, as it does not converge. However, the delta rule can be used to fit this approximation. The delta rule uses Gradient Descent (GD) to find the best weight on the search space for the current step. This approach is best understood considering a Perceptron neuron without a threshold, where the output is given by:

$$\vec{o} = \vec{w} \vec{x} \quad (2.7)$$

Besides this definition, the delta rule approach needs a measure of the training error of the training set  $D$ , also called the loss function. Many errors functions can be used, one of them is half of the sum of the squared difference of the expected output  $t$  and the neuron output  $o$ , given by:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \quad (2.8)$$

This error function is just a function of  $\vec{w}$  because the training examples and their output remains the same during the training phase. So, the neuron output  $o$  modifies when the weights change. Thus, the GD will be used to find the global minimum value for the error function. GD is an optimization method to find global minima for differentiable functions (CAUCHY et al., 1847). In the case of training a neuron, it is desired to find the best weights to have the predicted outputs close to the desired ones. So, the derivative of the error function defined in (2.8) is

$$\Delta E(\vec{w}) = \left[ \frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \dots, \frac{\delta E}{\delta w_n} \right] \quad (2.9)$$

Where each  $\frac{\delta E}{\delta w_i}$  is the partial derivative of the error function for each weight, and it is a component of the derivative error vector  $\Delta E(\vec{w})$ . This derivative error vector points towards the direction to increase the error function, so the negative of this vector is used, as the delta rule wants to decrease the error. The weight update using GD is

$$\vec{w} = \vec{w} + \Delta \vec{w} \quad (2.10)$$

Where

$$\Delta \vec{w} = -\eta \Delta E(\vec{w}) \quad (2.11)$$

For each weight the update is the same as (2.5), but now  $\Delta w_i$  is given by

$$\Delta w_i = -\eta \frac{\delta E}{\delta w_i} \quad (2.12)$$

With these equations to iterative update the weights, the remaining part of finalizing the algorithm, the partial derivatives of the error function, has to be calculated. This last step is the reason for choosing to error function as (2.8) that has a derivative easy to calculate, as:

$$\begin{aligned}
 \frac{\delta E}{\delta w_i} &= \frac{\delta}{\delta w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
 &= \frac{1}{2} \sum_{d \in D} \frac{\delta}{\delta w_i} (t_d - o_d)^2 \\
 &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\delta}{\delta w_i} (t_d - o_d) \\
 &= \sum_{d \in D} (t_d - o_d) \frac{\delta}{\delta w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
 \frac{\delta E}{\delta w_i} &= \sum_{d \in D} (t_d - o_d)(-x_{id})
 \end{aligned} \tag{2.13}$$

Where  $x_{id}$  is the  $i^{th}$  input component of the training example  $d$ . And  $\frac{\delta E}{\delta w_i}$  can be used in (2.12) as

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \tag{2.14}$$

Therefore the GD algorithm to train a Perceptron has three phases. The initialization, where the weights are randomly initialized. The update step, where each training example is used to update the weights using the equations presented above. And the stop condition, which can be a maximum number of iterations or the smallest value to consider that the error function converged.

Stochastic Gradient Descent (SGD) is one variation of the standard GD, but instead of updating the weights after iterating over all training examples, it updates after each training example. This modification on SGD sometimes helps the training algorithm avoid local minima, a common problem on both algorithms when searching for global minima. On SGD, the weights are updated using

$$\Delta w_i = \eta(t - o)x_i \tag{2.15}$$

Table 1 – Sigmoid, Tanh, ReLU, and ReLU - 6 activation functions and their derivative.

Function Name	Equation	Derivative
<b>Sigmoid</b>	$f(x) = \frac{1}{1+e^{-x}}$	$\frac{\delta f(x)}{\delta x} = f(x)(1 - f(x))$
<b>Tanh</b>	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\frac{\delta f(x)}{\delta x} = 1 - f^2(x)$
<b>ReLU</b>	$f(x) = \max(0, x)$	$\frac{\delta f(x)}{\delta x} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
<b>ReLU - 6</b>	$f(x) = \max(\max(0, x), 6)$	$\frac{\delta f(x)}{\delta x} = \begin{cases} 0 & \text{if } x < 0 \text{ and } x > 6 \\ 1 & \text{if } 0 \leq x \leq 6 \end{cases}$

Source: Author

On Subsection 2.1.1 was stated that the activation functions of a neuron have to be differentiable, and this is because of the learning algorithm. The third step on (2.13) assumed the output as (2.7), but when it is used as an activation function, the output of a neuron is:

$$\vec{o} = \sigma(\vec{w} \vec{x}) \quad (2.16)$$

Where  $\sigma$  is the activation function used on the neuron. Thus, the derivative of this output is given by:

$$\begin{aligned}
\frac{\delta}{\delta w_i} \vec{o} &= \frac{\delta \sigma(\vec{w} \vec{x})}{\delta w_i} \\
&= \frac{\delta}{\delta w_i} \sigma(\vec{w} \vec{x}) \frac{\delta}{\delta w_i} (\vec{w} \vec{x}) \\
\frac{\delta}{\delta w_i} \vec{o} &= x_i \frac{\delta}{\delta w_i} \sigma(\vec{w} \vec{x})
\end{aligned} \quad (2.17)$$

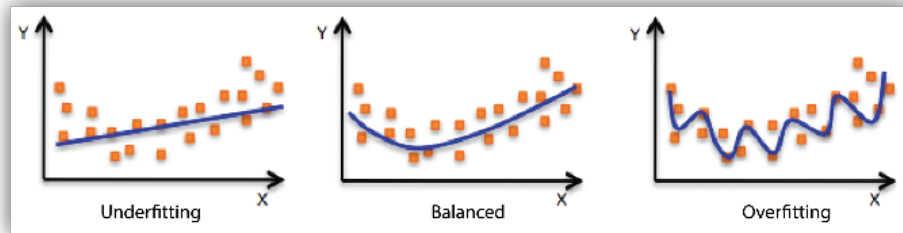
where  $\frac{\delta \sigma}{\delta w_i}$  is the derivative of the activation function. Table 1 shows the activation functions and their derivative.

Backpropagation is the generalization of SGD to train a MLP network. In Backpropagation, the loss function is similar to (2.8), but summing error over the outputs units:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 \quad (2.18)$$

where *outputs* is the set of output units,  $t_{kd}$  is the calculated output of unit  $k$  of training example  $d$  and  $o_{kd}$  is the ground truth output of unit  $k$  of training example  $d$ . (2.18) denotes

Figure 11 – Comparison between underfitted, overfitted, and perfect fitted function approximation.



Source: Collected from the internet. <sup>6</sup>

the error for the network. This equation and its derivative can calculate the output units' weights directly. Then the weights for the layers just before the output layer can be used to propagate the error and its derivatives from the output units. The rule of calculating the error using the next unit error can be repeated up to the input layer to calculate the weights for the network.

Repeating the backpropagation steps over some iterations will approximate the network into the desired function, but it will result in an approximation. The stop conditions have to be well-defined to avoid overfitting or underfitting on the training set. Overfit occurs when the network approximates too much to the desired function, losing the generalization, and underfit is the opposite when the approximation generalizes too much, being a poor approximation. For example, Figure 11 compares a perfect fit function approximation with an overfitted and an under-fitted approximation.

## 2.2 CONVOLUTIONAL NEURAL NETWORKS

Since the beginning of studies in NNs in the 1950s and 1960s, those models were developed to be used in tasks that seemed hard to humans, such as playing Chess or Go. Other functions naturally done by humans, such as sight and processing natural language, didn't have the same attention because they seemed easy to reproduce. However, it turns out to be the opposite. Models learned how to beat human champions on Go (SILVER et al., 2016) and Chess (CAMPBELL; JR; HSU, 2002), and the human visual system was poorly understandable.

One measure that represents how complex is the human vision system is that the ability to process visual information uses more than 50% of the cortex, the surface of the brain (HAGEN,

<sup>6</sup> <<https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>>. Accessed October 4, 2021



2012). The vision system uses this processing power to be invulnerable to many highly invariant competencies such as scaling, and translation (HAN et al., 2020). This invariance seems natural to humans, as we can identify objects with different sizes or the same object resized and rotated on the field of view. Unfortunately, this is one of the drawbacks not framed by the early NNs researchers.

In 1982, MARR released a book focused on a framework to replicate the human vision system (MARR, 1982). This framework didn't consider the image representation only in a neuron but in levels of representation. It was divided into 3 phases:

- Primal Sketch of the scene with low-level features such as lines, edges, curves, and boundaries.
- 2.5D Sketch with information on orientation, shades, textures, and discontinuities to provide information on image depth.
- 3D Sketch of the scene containing all the three-dimensional information.

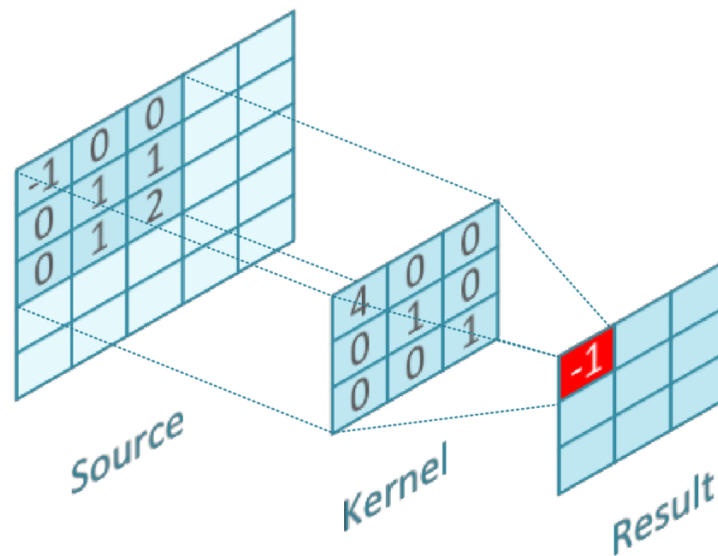
Using this idea of different levels of representation, CNNs emerges as a field in Deep Learning that tries to replicate the visual cortex. A CNNs model carries the framework presented by MARR, the first layers of the network map the low-level features, and as the network gets deeper, they map more complex features. After mapping all the crucial features, a Fully Connected (FC) layer is commonly used to classify the detected features.

The first CNNs model was the LeNet introduced in 1989 by LECUN et al. to classify numbers from hand-written numbers. In this model, instead of having neurons with weights multiplying the input, the neurons are convolutional kernels. With this change, how the model interprets the input image, as in the activation after a single layer, is a map of features of the output.

The feature extraction is done with concepts of image convolution, where a kernel passes through the image matrix generating a new matrix of values. Two different layers are used on CNNs. The first one is a simple convolution, where each value on the kernel is a trainable parameter, Figure 16 shows an example of how the convolution layer work. After the convolutional layer, it uses an activation function. Most of the time, ReLU or one of its variations is used.

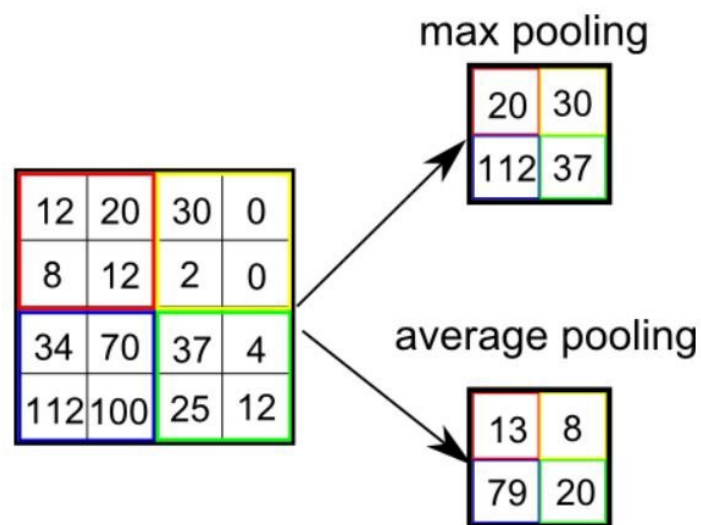
The second layer type is pooling, which downsamples a signal to reduce the image dimensions. This idea is applied to avoid that slight movement on the image results in a different feature map, making it harder to train a CNNs. Usually, a max pooling is used, which gets the

Figure 12 – Example of convolution in an image matrix.



Source: (WICHT, 2018)

Figure 13 – Example result of a max pooling and average pooling layer.

Source: Collected from the internet. <sup>7</sup>

higher pixel values in the kernel boundary, or an average pooling, which receives the average value of the pixels. For example, Figure 13 shows an example of max pooling and average pooling on a matrix.

There are two key advantages of this approach over NNs (LECUN et al., 1989), first a FC network would have a lot of weights to input all the image pixels, and the number of weights increases as much as the dimension of the image increases. And secondly, a FC architecture ignores all the topology of the input. However, in image inputs, the pixel values are highly

<sup>7</sup> <<https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>>. Accessed November 4, 2021

correlated with pixels near it.

### 2.2.1 Image Classification

The idea used by LeNet (LECUN et al., 1989) is to, given an input image, return a single class that represents all the images. This is also called Image Classification. This model receives as input an image of a handwritten digit from the US postal services and outputs the digit value. Besides, each input only has a unique number from zero to nine. The input used in this model was the start of a dataset of digits (LECUN; CORTES, 1998). This input type represents an example of iconic images, with only a single class occupying most of the image area.

2009 presented the ImageNet dataset containing more than 14 million iconic images hand-annotated in more than 20000 classes (DENG et al., 2009). This dataset was the core of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (RUSSAKOVSKY et al., 2015), a competition of image classification created in 2010. The idea of this competition was to use this dataset as a benchmark for image classification models.

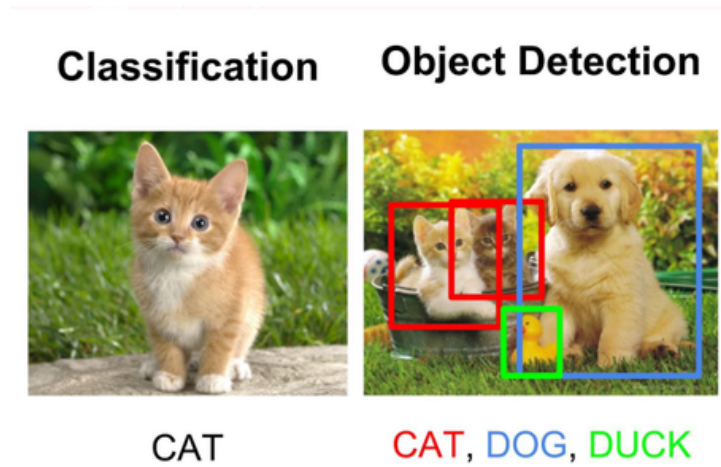
In the first years of competition, the proposed model didn't use CNNs as a classifier. In 2012, AlexNet was the first to propose the use of CNNs on this competition (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). This idea was a breakthrough in the competition and achieved a classification error of 15.3%, 10.8 percentage points lower than the runner-up.

AlexNet used an architecture similar to LeNet with some improvements. First, it was deeper than LeNet, making it possible for the model to learn finer features. Besides, it used ReLU instead of Sigmoid as the activation function, which improved the gradient propagation. Lastly, it used some optimizations, such as dropout and data augmentation. These improvements showed that CNNs had more potential than it was explored.

Over the years, more models with different improvements showed better results. For example, in 2015, the GoogLeNet presented the Inception block, which consists of small modules with convolution kernels of different sizes (SZEGEDY et al., 2015). Those blocks took advantage of various level of feature levels in the same layer. As a result, it achieved a 6.7% classification error.

Another significant progress was the Residual Nets (ResNets) (HE et al., 2016), which first used a residual block on CNNs, which allowed having deeper networks without having the problem of vanishing gradient. These residual blocks bypass the input of a layer to a block in the following few layers. So, with this bypass, the gradient has a shortcut to get on layers

Figure 14 – Differences between classifying only one object on an image and detecting multiple objects on an image.



Source: Collected from the internet.<sup>8</sup>

where it was getting too small. As a result, the ResNets achieved 3.6% classification error.

### 2.2.2 Object Detection

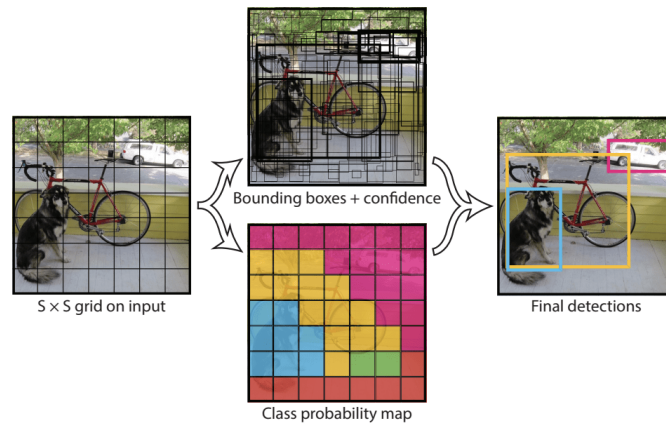
CNNs was an excellent alternative to classify images. Still, the classification problem is limited to classifying a single object on an image, but images commonly have more than one object. In that case, only one class would be predicted in the image. Object detection is the alternative to address this problem, as it can detect and locate multiple objects in an image. An object detector outputs the class and a bounding box for the predicted objects in an image. Figure 14 shows an example of an image classifier predicting only a cat in an image and an object detector outputting multiple objects and their location in an image.

Alongside with object detection challenge, some datasets were presented. One most notable is Common Objects in Context (COCO) dataset (LIN et al., 2014), which has annotations for object detection, image segmentation, and key points detection. COCO dataset has been used as a benchmark on object detection competitions since its launching. One model that had become famous for its good accuracy and good inference time on COCO dataset was YOLO (REDMON et al., 2016; BOCHKOVSKIY; WANG; LIAO, 2020).

The first YOLO model was released in 2016, and since then, some improvements have been proposed to the original model. The core idea from YOLO is to divide the prediction into a grid of cells, where which cell outputs a fixed number of bounding boxes centered on this

<sup>8</sup> <<https://www.kaggle.com/getting-started/169984>>. Accessed November 9, 2021

Figure 15 – Example of a YOLO output containing the bounding boxes for each cell with its class probability and the final predicted bounding box.



Source: (REDMON et al., 2016)

cell and the confidence of each predicted class confidence on this bounding box. Figure 15 shows an example of an output from YOLO divided into cells, with the predicted bounding boxes, the predicted classes, and how this information is merged to give the final prediction. Besides, YOLO continues to use a CNNs as the backbone of its detection.

## 2.3 OBJECT DETECTION MODELS

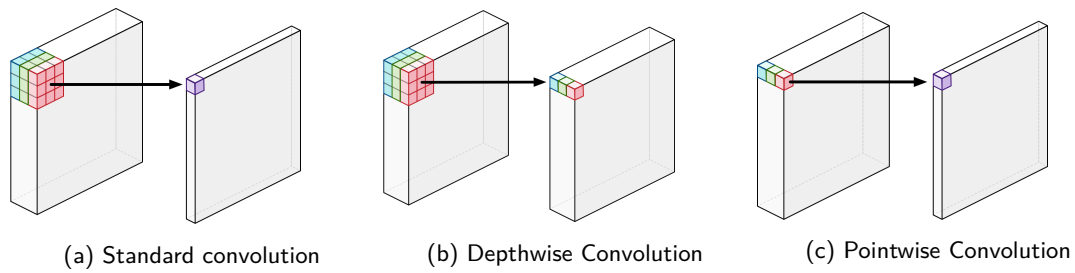
### 2.3.1 MobileNet v1

MobileNets are one of the most used classes of CNNs models used in embedded systems. The MobileNet v1 was introduced in 2017 by HOWARD et al., aiming to reduce the number of parameters and the number of operations to reduce inference time and network size. This model relies on using Depthwise Separable Convolution (DSC), introduced by (SIFRE; MALLAT, 2014), which divides a regular convolution into two steps: a Depthwise Convolution and a  $1 \times 1$  Pointwise Convolution.

A standard convolution layer is defined by  $N$  filters of dimension  $D_K \times D_K \times M$ , where  $D_K$  is the spatial dimension of the squared filter and  $M$  is the number of input channels. While in the DSC, the Depthwise Convolution Filters use  $M$  filters of  $D_K \times D_K \times 1$ , and the Pointwise Convolutions use  $N$  filters of  $1 \times 1 \times M$ . Figure 16 shows the difference between these convolutions.

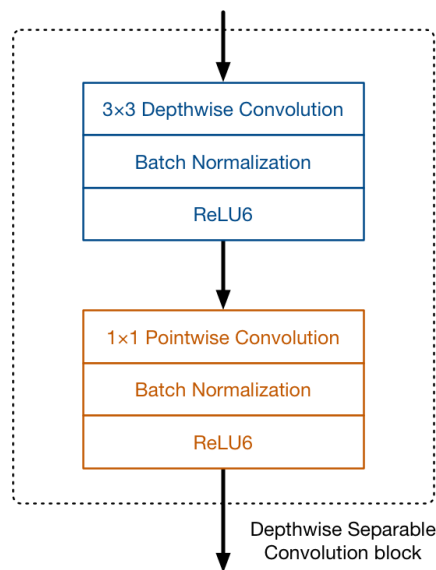
<sup>9</sup> <<https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/>>. Accessed May 24, 2021

Figure 16 – Standard Convolution (a) replaced by Depthwise Convolution (b) and Pointwise Convolution(c).



Source: Collected from the internet. <sup>9</sup>

Figure 17 – DSC block used in MobileNet v1.



Source: Collected from the internet. <sup>10</sup>

The MobileNet v1 defines as a DSC block, where each Depthwise and Pointwise Convolution are followed by a Batch Normalization Layer and a ReLU6, as the activation function. In this block, the Depthwise Convolution filters the input, and Pointwise Convolution creates a new set of features by combining the filtered layers. Figure 17 shows the DSC block used in the MobileNet v1.

The authors of MobileNet v1 proposed a model with a standard convolution on the first layer followed by thirteen blocks of DSC. After these convolutions layers, an input image of  $224 \times 224 \times 3$  is converted into  $7 \times 7 \times 1024$  feature maps. Those layers are the core idea used in a MobileNet v1 model, and they can be used in different applications, such as image classification, object detection, and semantic segmentation.

For instance, in an image classification model based on the MobileNet v1, those layers are

<sup>10</sup> <<https://machinethink.net/blog/mobilenet-v2/>>. Accessed May 27, 2021

Figure 18 – MobileNet v1 architecture to image classification.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5×	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
	Conv dw / s2	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 1024$
	Conv dw / s2	$3 \times 3 \times 1024$ dw
	Conv / s1	$1 \times 1 \times 1024 \times 1024$
	Avg Pool / s1	Pool $7 \times 7$
	FC / s1	$1024 \times 1000$
	Softmax / s1	Classifier

Source: (HOWARD et al., 2017)

followed by an Average Pooling (Avg Pool) layer, a FC layers, and a Softmax to perform the classification. In this approach, the trained model only outputs the most likely class for the whole image. Figure 18 shows this architecture for image classification.

### 2.3.2 MobileNet v2

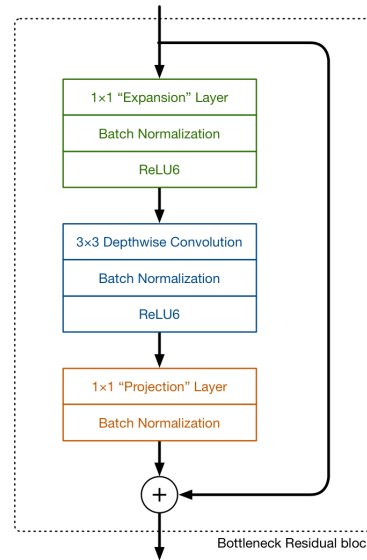
The use of DSC in MobileNets showed a significant reduction in the number of parameters and the number of operations (HOWARD et al., 2017). However, in the MobileNet v1, the layers in deeper filters increase to a  $7 \times 7 \times 1024$  on the last DSC layer. Therefore, in 2019, SANDLER et al. proposed MobileNet v2, an improvement to optimize the v1, with fewer parameters and fewer operations.

The MobileNet v2 (SANDLER et al., 2019) applies some of the improvements presented by ResNets (HE et al., 2016). The main improvement of MobileNet v2 is based on a Inverted Bottlenecks (IBNs) block that reduces the number of filters used in v1. Figure 19 shows the block used in MobileNet v2. Compared with the convolution block used in v1, this new block has a  $1 \times 1$  convolution before the v1 convolutions and the residual connection.

The new block used in v2 aims to reduce the number of channels through the model.

<sup>11</sup> <<https://machinethink.net/blog/mobilenet-v2/>>. Accessed May 27, 2021

Figure 19 – Bottleneck Residual block used in MobileNet v2.



Source: Collected from the internet. <sup>11</sup>

This reduction occurs because of a change in how the  $1 \times 1$  layers are used. In MobileNet v1, the  $1 \times 1$  Pointwise Convolution only increases the number of channels to the next layer. However, in v2, there are two types of  $1 \times 1$  convolutions, the Expansion Layer and the Projection Layer. The Projection Layer compresses the information to the next block reducing the number of channels, while the Expansion Layer uncompresses this data to input in the Depthwise Convolution. The Expansion Layer expands its input by the expansion factor of a given layer. Besides the IBNs, MobileNet v2 also uses the residual connection, as the residual block in ResNets (HE et al., 2016), that avoid gradient from vanishing.

These ideas reduce the input size for the next IBNs, thus shrinking the number of parameters in the model. MobileNet v2 has 18 layers, where the first layer is a standard convolution layer, followed by 17 IBNs blocks. This network converts a  $224 \times 224 \times 3$  image in a  $7 \times 7 \times 320$  feature map. For instance, the image classification version of MobileNet v2 uses this backbone followed by a standard  $1 \times 1$  convolution layer, an Avg Pool layer, and a  $1 \times 1$  convolution layer, used as a classifier. Figure 20 shows the MobileNet v2 architecture for image classification, using an input of  $224 \times 224 \times 3$  image.

### 2.3.3 MobileNets SSD

MobileNets models presented in subsections 2.3.1 and 2.3.2 were described as image classifiers due to the output of only one class for the whole image. However, this work aims to



Figure 20 – MobileNet v2 architecture to image classification.  $t$  is the expansion factor,  $c$  is the number of output channels,  $n$  is the number of layers using this configuration, and  $s$  is the stride of each layer.

Input	Operator	$t$	$c$	$n$	$s$
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Source: (SANDLER et al., 2019)

detect multiple objects in an image, which can be used to achieve that. As stated before, what makes the classification step in these models are the last layers, the Avg Pool layer, FC layer, and Softmax layer. In comparison, the convolutions layers extract features from the image.

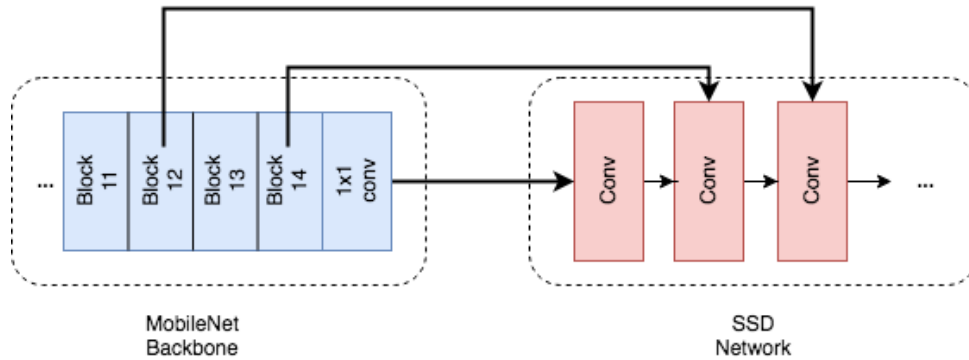
MobileNets can detect multiple objects in an image by using these models as a backbone to the Single Shot Detection (SSD) framework (LIU et al., 2015). The SSD framework proposed adding extra convolutions layers to use VGG16 (SIMONYAN; ZISSERMAN, 2014), an image classification model as an object detection model. These additional layers make the model predict multiple bounding boxes for various classes instead of a class for the entire image.

This idea can be extended to use SSD framework on MobileNet v1 and v2. To use MobileNet v1 as a backbone of SSD framework, the last three layers of the image classification model (Avg Pool, FC, and Softmax) are removed. While in the MobileNet v2, the Avg Pool and last convolution layer are removed. After that, the SSD framework is attached to the last three DSC layers. Figure 21 show the last of remaining layers of MobileNet v1 connected to the SSD framework and Figure 22 does it similar to MobileNet v2 as backbone to the SSD framework.

<sup>12</sup> <<https://aditya-kumar-52859.medium.com/object-detection-with-ssd-and-mobilenet-aedc5917ad0>>. Accessed June 9, 2021

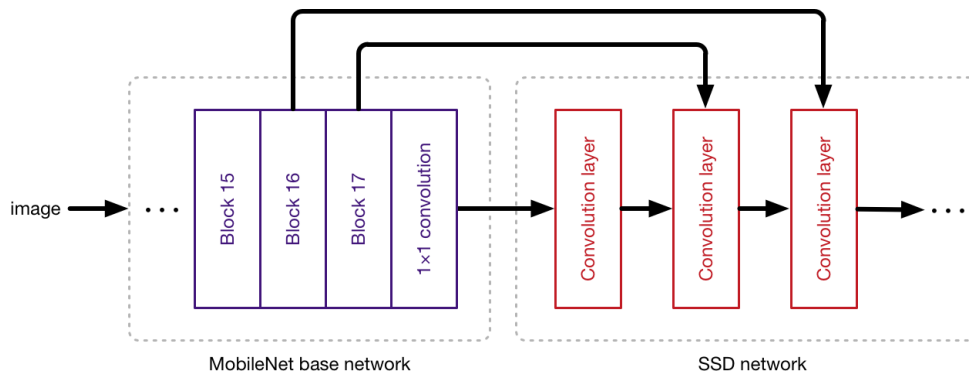
<sup>13</sup> <<https://machinethink.net/blog/mobilenet-v2/>>. Accessed May 27, 2021

Figure 21 – MobileNet v1 architecture for object detection using SSD framework.



Source: Collected from the internet. <sup>12</sup>

Figure 22 – MobileNet v2 architecture for object detection using SSD framework.



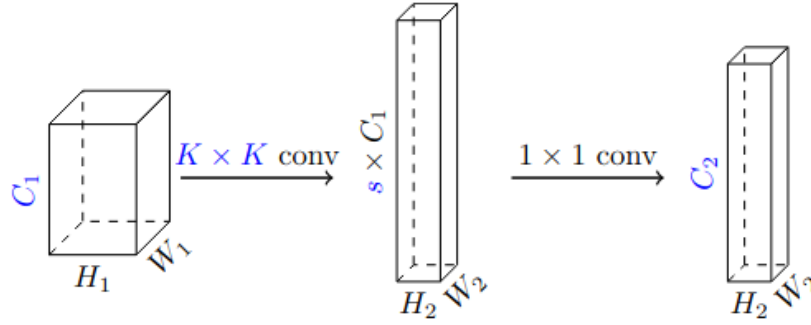
Source: Collected from the internet. <sup>13</sup>

### 2.3.4 MobileDets

MobileDets (XIONG et al., 2020) emerges in the increasing number of embedded applications using NNs and the rise of dedicated processors to CNNs frameworks. Thus, work from MobileDets investigates the latency-accuracy trade-off of using regular convolutions and DSC in mobile network architectures. This comparison is made by using TuNAS (BENDER et al., 2020), a Neural Architecture Search (NAS) that uses Reinforcement Learning to find the best hyper-parameters without using the manual tune.

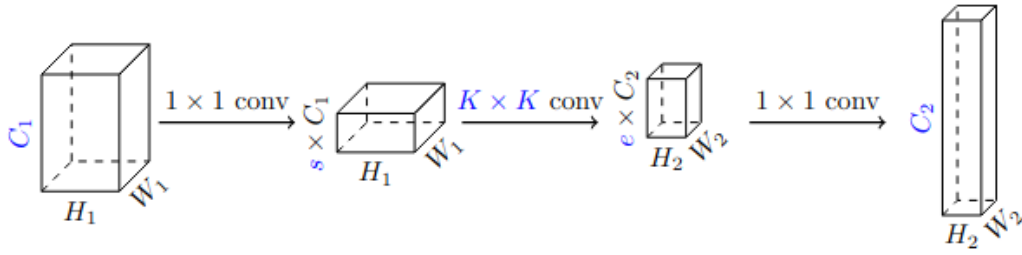
In MobileDets work, the authors consider that DSC reduce the number of parameters in the network. Still, the latency might not be shortened in dedicated processors such as Edge TPUs. Therefore, they presented two new convolution blocks to be used together with the IBNs block in the search space. The Fused Inverted Bottleneck layer fuses the first Pointwise and the Depthwise convolution of the IBNs to behave as an expansion layer,  $s > 1$ . Figure 23 shows this Fused Inverted Bottleneck layer. The Tucker Layer is similar to the IBNs, but the

Figure 23 – Fused Inverted Bottleneck layer, where the first Pointwise and the Depthwise convolution were fused on a  $K \times K$  regular convolution. This regular convolution expands the  $H_1 \times W_1 \times C_1$  input by the expansion factor,  $s > 1$ , and outputs a  $H_2 \times W_2 \times C_2$  block.



Source: (XIONG et al., 2020)

Figure 24 – Tucker Layer similar to DSC, but the first Pointwise convolution reduces the input size by the input compression factor  $s < 1$  and the  $K \times K$  regular convolution instead of the Depthwise convolution. This regular convolution changes the number of filters from  $s \times C_1$  to  $e \times C_2$ , with an output compression factor  $e < 1$ .



Source: (XIONG et al., 2020)

first Pointwise convolution has a compression ratio  $s < 1$ . Figure 24 shows the Tucker Layer.

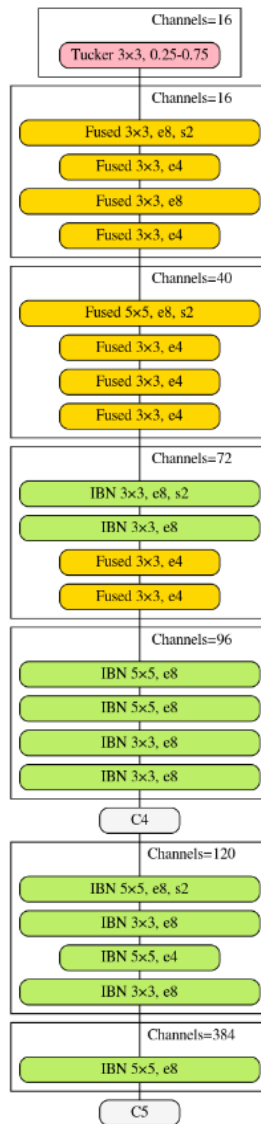
Besides the types of convolution used in each layer, MobileDets work proposes searching the hyper-parameters of the architecture, such as the input and the output size of each layer ( $C_1$  and  $C_2$  in Figure 23 and Figure 24), the size of filters ( $K$  in Figure 23 and Figure 24), and the expansion and compression factors ( $s$  and  $e$  in Figure 23 and Figure 24). The reward function used in this work was the following:

$$R(M) = mAP(M) + \tau \left| \frac{c(M)}{c_0} - 1 \right| \quad (2.19)$$

Where  $mAP(M)$  denotes the mean average precision of the model  $M$ ,  $c(M)$  is the inference time of the model  $M$ ,  $c_0$  is the maximum inference time, and  $\tau < 0$  regulates the trade-off between precision and inference time of the model.

Figure 25 shows the best architecture found by the work to the EdgeTPU, using the three kinds of layers. This architecture is used as the backbone of a SSD-Lite framework, which

Figure 25 – Best MobileDet architecture to EdgeTPU.  $s$  refers to the stride,  $e$  refers to the expansion factor, and 0.25 – 0.75 on the Tucker layer refers to the input and output compression factors.



Source: (XIONG et al., 2020)

is similar to the approach presented on Subsection 2.3.3, but it uses DSC instead of regular convolutions.

### 2.3.5 Training MobileNets and MobileDets

TensorFlow Object Detection API (HUANG et al., 2017) was used to train MobileNets and MobileDets models, making modifying and adjusting some hyper-parameters easy. Furthermore, the training process of these models was improved using data augmentation techniques such as Horizontal Flip, Image Crop, Image Scale, Brightness Adjustment, Contrast Adjust-

Table 2 – Hyperparameters modifications on Mobilenet v1 SSD, MobileNet v2 SSD and MobileDet models.

Hyperparameter	Used Values
<b>Input Size</b>	$224 \times 224$
<b>Batch Size</b>	32
<b>Data Augmentation</b>	Horizontal Flip, Image Crop, Image Scale, Brightness Adjustment, Contrast Adjustment, Saturation Adjustment, and Black Patches

Source: Author

ment, Saturation Adjustment, and Black Patches. Table 2 shows the modifications made to the training parameters in these models.

### 2.3.6 YOLO v4 tiny

YOLO v4 tiny is a shallow version of the YOLO v4 (BOCHKOVSKIY; WANG; LIAO, 2020), designed to run in an embedded system. It already uses CutMix, Mosaic, Class Label Smoothing, and Self-Adversarial Training, so this architecture does not need any extra data augmentation technique. Furthermore, due to limitations of the portability process for Google Coral Edge TPU, the YOLO v4 tiny uses ReLU rather than Leaky ReLU as an activation function.

## 2.4 METRICS

In ML, there are several ways to measure how well a model performs a task. Two of the most simple and efficient metrics are precision and recall. Precision measures the percentage of correct predictions that matches a ground truth, and recall measures the probability of a correct prediction. Those metrics can be calculated as follow:

$$precision = \frac{\text{correct predictions}}{\text{all predictions}} \quad recall = \frac{\text{correct predictions}}{\text{all ground truth}} \quad (2.20)$$

### 3 RELATED WORK

Object detection has been one of the most studied fields in computer vision since the first use of CNNs (LECUN et al., 1989). Over the years, datasets were released alongside competitions to encourage new approaches and techniques to detect objects in an image, pushing the state-of-the-art further. Section 3.1 will present other datasets for object detection, such as the state-of-the-art COCO for a diverse set of objects. Section 3.3 will show datasets and object detection techniques used on RoboCup leagues. Section 3.2 presents some approaches to estimate the distance and the relative position using monocular images.

#### 3.1 GENERAL PURPOSE DATASETS

One of the first publicly released image datasets was the ImageNet (DENG et al., 2009), together with a yearly competition, to push forward image classification techniques initially. This dataset contains over 14 million labeled images distributed over 21,000 categories. Before ImageNet, most image datasets were relatively small and focused on a small number of categories. ImageNet changed this by providing a much larger and more diverse dataset that covers a wide range of object categories and visual concepts. In 2017 this competition was considered a solved problem since the best model achieved an error of 2.3%.

The most famous and used dataset for object detection is COCO (LIN et al., 2014), released in 2014. This dataset contains 328.000 images collected from Flickr to avoid getting iconic-object images containing a single object centered in the image. Thus, the COCO dataset focuses on non-iconic images, meaning images with multiple categories in a diverse context. This strategy helps trained models to generalize objects instances, given the numerous contexts.

The classes used in the COCO dataset were chosen among 255 candidates given by children from 4 to 8 years old. The authors then voted on these categories based on how often each category occurred, and the most voted ones were selected, resulting in 80 classes. This dataset consists of 2.5 million instances, and as a result, each image averages 7.7 instances per image. It took 77.000 working hours to label all of these instances.

### 3.2 RELATIVE POSITION ESTIMATION

Relative position estimation using a monocular image is a problem related to distance estimation since having the relative position, it is possible to calculate the distance, and the way around is possible to calculate the relative angle using the camera angle range. Estimating the distance of detected objects is a typical task for autonomous cars. For example, in the autonomous car environment, the KITTI dataset (GEIGER; LENZ; URTASUN, 2012) has more than 200 thousand detected objects with their relative distance.

One of the most traditional methods to estimate the distance using a monocular image is to use Inverse Perspective Mapping (IPM) (REZAEI; TERAUCHI; KLETTE, 2015). The IPM method is based on transforming the image on a bird's-eye view so it can estimate the distance based on the distance in pixels from the bottom of the image to the bottom of the detected bounding box. However, this method has a confidence interval of  $60cm$ , which wouldn't be feasible in SSL context since this is ten times bigger than the ball diameter.

Some other works use a trainable model, so given the detected bounding box, it outputs the distance for this object. One approach used on small drones is using a Support Vector Regression (SVR) to estimate the distance of other drones using the bounding box information, such as width and height (Gökçe et al., 2015). This approach has a median error of  $18cm$  for the distance estimation, but a SVR can only be used to distance since it just outputs one value. In this approach, it is used a dataset of 8876 images collected from embedded videos. Therefore, using a trainable model to estimate the distance shows a better result than IPM.

Another work proposes a new MLP architecture called DisNet (HASEEB et al., 2018) to estimate the distance using bounding box information. This work uses a 2000 bounding box dataset focused on detecting four classes Person, Bicycle, Car, and Truck. To find the best network architecture, it divides the dataset into 80% for training, 10% for validation, and 10% for testing. It found that the best MLP architecture has 3 hidden layers containing 100 neurons each. This architecture achieved a distance accuracy close to 90%. This search for hyperparameters idea could be an excellent solution to check the trade-off between the model evaluation and the inference time.

Another option to predict the distance using monocular vision would be to use the same CNNs outputting the detected bounding box with the calculated distance. (ZHU et al., 2019) proposes adding a key point on each bounding box and changing the final layers of a CNNs model adding 3 new FC layers that have the last feature maps as the input and outputting the

object distance. This work used almost 1500 images containing 48000 objects from the KITTI dataset. This work tested this approach using a ResNets and a VGG and achieved the best result, a Root Mean Square Error (RMSE) of  $6.8m$ . However, it had to use a large dataset annotated with the distance to all the objects to make this approach possible.

There is also the possibility of a model predicting the depth for all the points in the image. This work (MASOUMIAN et al., 2021) combines the result of a DepthNet (KUMAR; BHANDARKAR; PRASAD, 2018) to predict the distance of all the points in the image and a YOLO to have the bounding boxes. It used the DepthNet already trained with the KITTI dataset and the YOLO trained with the COCO dataset and tested with 100 new images collected from them, achieving a RMSE of  $0.20m$ .

In the RoboCup SSL, a work proposes a method to locate a detected object based on a monocular camera (MELO; BARROS, 2022). In this work, the author presents an approach to calculate the position in two steps. First, it finds the point where the detected ball touches the ground, and then it uses this point and the camera's intrinsic parameters to estimate the position.

(MELO; BARROS, 2022) creates a dataset of 30 images and manually marks the pixel where the detected ball touches the ground. To have the ground-truth position, it uses a measuring tape to have the actual position of each point. Using this dataset, fit a Linear Regression model inputting the bounding box position and outputting the pixel where the ball touches the ground. Then it finds the camera pose to estimate its intrinsic parameters and calculates the object position using the ground point. This work achieved a RMSE of  $67.32mm$ .

### 3.3 OBJECT DETECTION IN ROBOCUP

The SPL has an open-source tool to create and share datasets for object detection (FIEDLER; BESTMANN; HENDRICH, 2018) that has several images labeling Robot, Ball, and Goalpost. This dataset shows the critical classes that need to be detected on a RoboCup league. However, those datasets couldn't be used on SSL context as those classes are represented differently. For instance, SSL uses an orange golf ball, the robot is a cylinder with a  $15cm$  height, and the goalpost has an area of  $100cm \times 16cm$ , while the SPL uses a black and white ball with a diameter of  $10cm$ , the robot is a humanoid, and the goalpost has  $150cm \times 80cm$ .

In other RoboCup leagues, some promising approaches to detect and locate objects. For



instance, on MSL, there is a work (Luo et al., 2017) that presents a pipeline to locate other robots in the field. They propose a new dataset for MSL annotating only robot instances and use this dataset to train a CNNs. After detecting the robots, they combine this image with a depth point cloud to estimate the position of those robots.

They collected and annotated 1456 images to construct the dataset, dividing them into 70% for the training set and 30% for the testing set. This dataset uses pictures from two sources, pictures taken from the robot camera and images taken from outside of the field from different competitions, to increase the variety of the dataset. Therefore this work gives a good direction for collecting photos and creating a dataset for a league in RoboCup. It reasonably estimates how many images a new dataset should have.

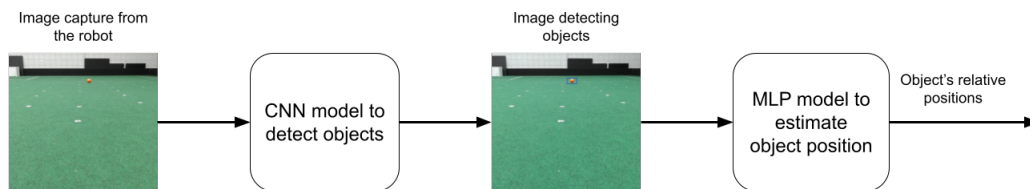
For object detection, they proposed a CNNs architecture of 9 convolutional layers, of which only the first 6 have the pooling operation. They presented this architecture to run the inference for this network on a Jetson TX2. They achieved a Average Precision (AP) of 70.65% with an inference time of  $58ms$ , close to 17 Frames Per Second (FPS). This result shows that achieving a good AP is possible while keeping the detection close to real-time inference for embedded devices.

In this work, the image used to detect objects is combined with the depth point cloud from a Kinect v2 sensor to locate the detected objects. Thus, they get the distance of the central point of the bounding box and use it to estimate the detected robot position. Therefore, using this point cloud approach, they achieved a mean error of  $30.57mm$ , which is expected since this cloud sensor is close to  $20mm$  and gives a baseline to the localization error. However, it is impossible to fit a sensor like the one used in this work on a SSL robot, as it wouldn't fit the size requirements.

## 4 PROPOSED APPROACH

This work is constructed having the central objective of estimating the position of an object relative to a SSL robot without receiving any external information. To achieve that, a camera was attached to the robot to capture the input images. Figure 26 shows the pipeline it was used. First, the camera placed on the robot captures an image from the field, and this image is used as input on a CNNs model trained to detect the relevant objects on a SSL game. Then the bounding boxes' positions from the detected objects are used on a MLP model to estimate the object position relative to the robot with the camera.

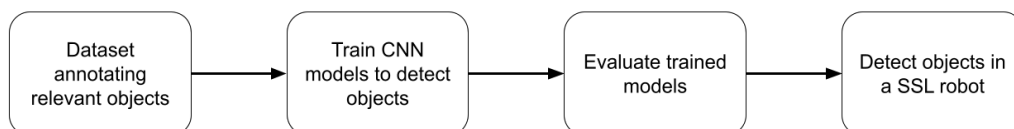
Figure 26 – Pipeline for the proposed approach followed on this work.



Source: Author

On Figure 26, two boxes need to be filled, the model to detect the objects and the model to estimate the object position. Figure 27 shows the pipeline this work followed to build the model to detect objects. First, a dataset annotating the relevant objects to SSL context was annotated. The creation of this dataset is detailed on Section 4.1. Next, CNNs models were trained using this dataset to detect the relevant objects. Section 4.2 presents the approach to train those models. Section 4.3 shows how those models were evaluated and Section 4.4 explains how the environment on the robot runs those models' inferences.

Figure 27 – Pipeline for the proposed approach to building a model to detect SSL relevant objects.

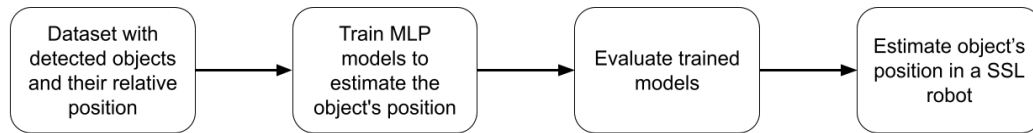


Source: Author

Figure 28 shows how the model to estimate the object's position was trained and deployed to fill the second box on Figure 26. Section 4.5 shows hows the dataset for estimating object

position was created using the object detection model and the global position from SSL-Vision. Section 4.6 shows how the MLP models were trained using this dataset. Section 4.7 presents how the models were evaluated and which metrics were used.

Figure 28 – Pipeline for the proposed approach to building a model to estimate SSL relevant objects position.



Source: Author

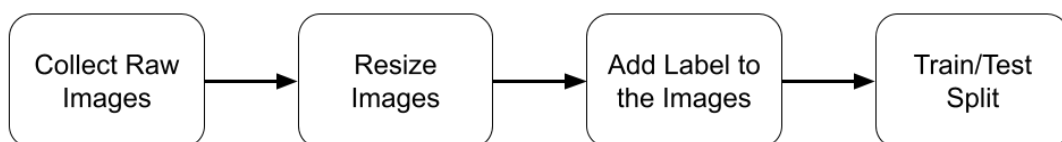
## 4.1 OBJECT DETECTION DATASET

This chapter will present the proposed dataset for training and validating models to detect objects in SSL. Subsection 4.1.1 will detail how the dataset was created and Subsection 4.1.2 will present some statistics about this dataset. Although this work uses SSL as a study case, the presented approach can be used to create datasets used to object detection.

### 4.1.1 Dataset Creation

The methodology used to create the dataset used in this work has four steps: collect unlabeled public images, resize the images, add the desired label to the images, and divide the images into train and test sets. Figure 29 shows this sequence of steps.

Figure 29 – Pipeline used to create the dataset.



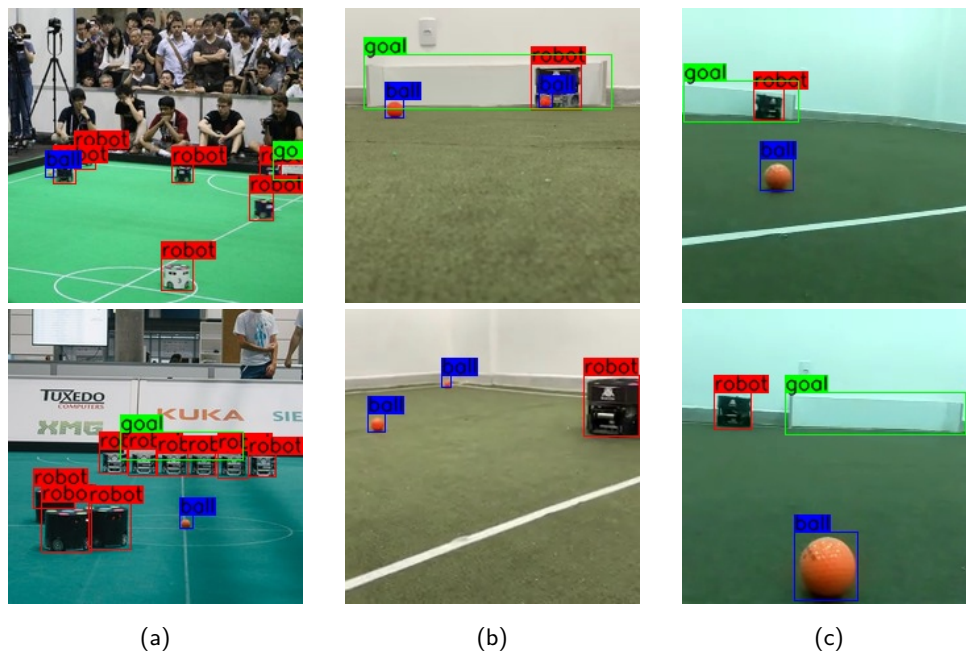
Source: Author

The proposed dataset's images come from three sources using images taken under different conditions and angles. The first set of images consists of 259 pictures taken outside of the SSL field, obtained from public image repositories of league teams. This set contains a variety of robot models and images taken under various light conditions. The second set has 516 brand-new pictures taken for this dataset from a smartphone camera inside a university laboratory

field. Furthermore, the remaining 156 images were collected using a camera placed on the SSL robot. The combination of those sets results in a dataset of 931 images.

After collecting the images, they were resized to a standard resolution of  $224 \times 224$  pixels as used by (HOWARD et al., 2017; SANDLER et al., 2019). This resizing allows the models to have fewer pixels to process and be faster running the inference. Figure 30 shows some labeled examples from this dataset. Each column of this set of images has two examples of each collected source. This figure also shows the ground truth class labels.

Figure 30 – Sample images from the dataset, showing ground-truth detection. The leftmost column (a) has images from public SSL images, the middle column (b) has images collected from a smartphone inside the field, and the rightmost column (c) has images taken in a camera placed in the robot.



Source: Author

The next step of creating the proposed dataset was to add labels to the objects in images. The proposed dataset defined three objects class to label: Robot, Ball, and Goal. These classes are the distinctive and relevant ones to detect in an SSL game. Each image on this dataset can contain multiple labels, including none in the image. To annotate those labels, a tool, Labellmg (Tzutalin, 2015), was used to add the ground-truth bounding boxes on the images. This tool outputs the ground-truth detection in YOLO and Pascal VOC format.

After labeling the images, they were randomly divided into a train and a test sets using the 70/30 proportion as in other robotics' object detection works (Luo et al., 2017), Table 3 shows the final result of this division. This creation process took 160 working hours, most of

them manually adding labels to each image. The proposed dataset is available on the author's GitHub <sup>1</sup>.

Table 3 – Number of images divided into train and test set.

	Number of images
<b>Train</b>	651 (69,92%)
<b>Test</b>	280 (30,08%)
<b>Dataset size</b>	931

Source: Author

#### 4.1.2 Dataset Statistics

One crucial factor used to analyze a dataset is class balancing. As the proposed dataset detects multiple instances in one image, there will have more instances than images. Table 4 shows the instance division on the proposed dataset. For example, the Goal class has fewer examples than Robot and Ball classes because not all images have a Goal instance, and when it appears, there is only one instance per image. However, this would state a problem, since the goal instances are very similar. Besides, the COCO dataset (LIN et al., 2014) also have some imbalanced classes and this didn't present a problem. The dataset has 4182 instances, averaging 4.5 instances per image, which helps mitigate the low number of images.

Table 4 – Number of instances of each class in the dataset.

Object Class	Instances per class
<b>Robot</b>	1886
<b>Ball</b>	1711
<b>Goal</b>	585
<b>Number of instances</b>	4182

Source: Author

When creating an image dataset, the images can be grouped into iconic and non-iconic images (BERG; BERG, 2009). Iconic images are the ones that have only one big object represented per image, and they are usually used for image classification. In comparison, non-iconic images have multiple objects in a non-uniform background. In addition, iconic images can represent all the details of one object. However, these images make it harder for the model

<sup>1</sup> <<https://github.com/bebetocf/ssl-dataset>>

to generalize and identify a class in multiple contexts (TORRALBA; EFROS, 2011). Figure 31 shows some examples of iconic and non-iconic images in the dataset.

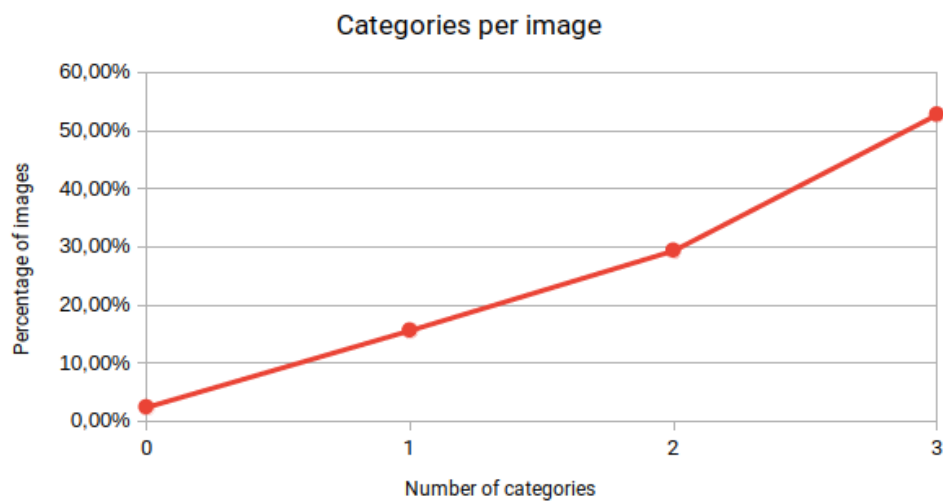
Figure 31 – Exemples of (a) iconic images and (b) non-iconic images present in the proposed dataset.



Source: Author

Some dataset statistics, such as the number of categories per image, instances per image, and object size, are used to analyze the presence of iconic and non-iconic images. Figure 32 shows the dataset's number of categories per image. It is noticeable that more than 80% of the images have more than two categories per image, which helps the dataset to have multiple representations of a class in different images.

Figure 32 – Number of categories per image.

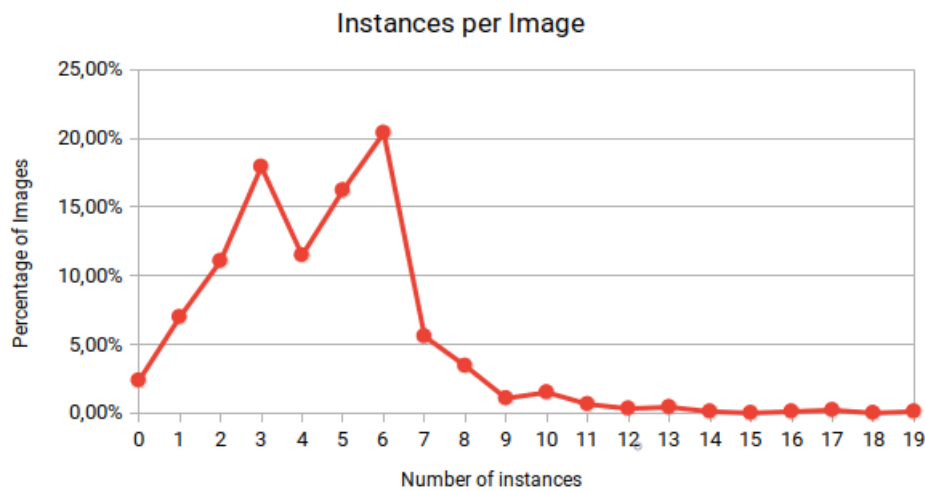


Source: Author

Another factor that represents iconic images is the number of instances per image. Usually,

this type of image has only one example per image. However, as the proposed dataset's main objective is to detect objects in distinct game situations that could have multiple objects, it is crucial to have multiple instances in each image. Figure 33 shows the number of cases per image. It is possible to see that most images have more than one instance, so there are more class instances than images. This increases the variability of the classes, helping the models to generalize the objects information.

Figure 33 – Number of instances per image.



Source: Author

The objects in the proposed dataset are classified by their area, Small, Medium, and Large, similarly to COCO (LIN et al., 2014). Small objects have an area less than  $32 \times 32$  (1024) pixels. Medium objects have an area between  $32 \times 32$  (1024) pixels and  $96 \times 96$  (9216). Moreover, Large objects are bigger than  $96 \times 96$  (9216) pixels. The dataset has 2919 Small objects, 1225 Medium objects, and the remaining 38 objects are Large. The low number of Large objects wouldn't be a problem since the input image has  $224 \times 224$  pixels and most objects aren't that close to the camera.

Table 5 summarizes this division per object size. Most objects concentrate in the Small area class, approximately 70% of all objects, due to the low-resolution images. The bigger presence of Small objects is beneficial to the dataset because these objects help a model to generalize.

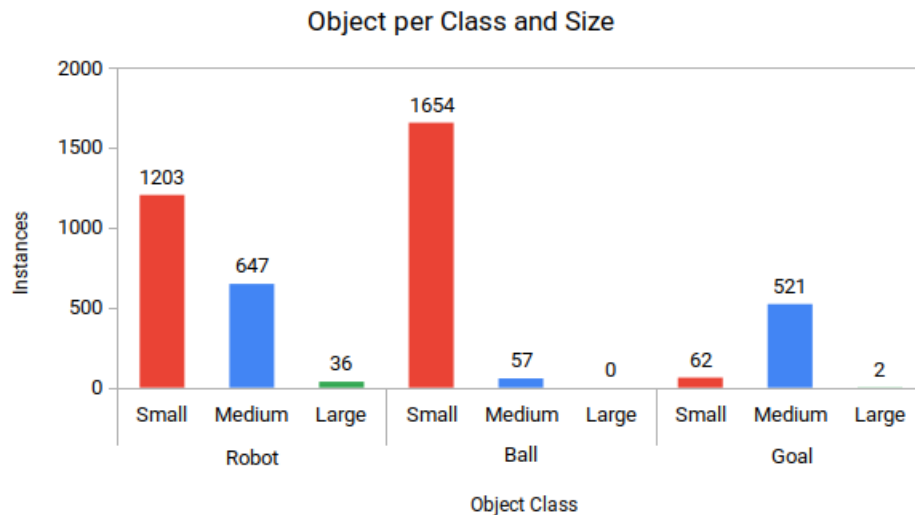
Table 5 – Number of each instance size in the dataset.

Object Size	Instances per object size
Small	2919
Medium	1225
Large	38
<b>Number of instances</b>	<b>4182</b>

Source: Author

Figure 34 shows each instance's division by the class and their area size. It is possible to see that almost all of the Ball instances are Small due to their size. More than half of the Robots' examples are Small due to images from the first set taken from outside the field, where robots are far from the camera. Furthermore, most of the Goals' samples are in the Medium class.

Figure 34 – Instance division per class and size. Small objects have less than  $32 \times 32$  (1024) pixels, medium objects have a size between  $32 \times 32$  (1024) pixels and  $96 \times 96$  (9216) pixels, and large objects are bigger than  $96 \times 96$  (9216) pixels.



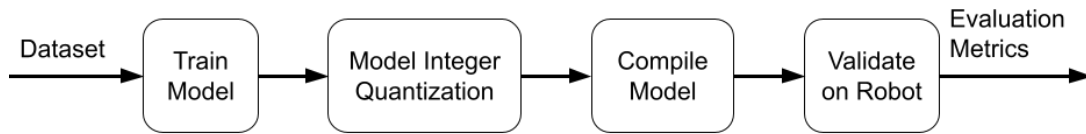
Source: Author

## 4.2 TRAINING OBJECT DETECTION MODELS

The pipeline to train, run and evaluate any model follows the same standards for each approach. This pipeline has four steps. First, the considered model is trained with the dataset. This trained model is quantized and compiled to infer using Google Coral USB Accelerator. This USB Accelerator is an Edge TPU coprocessor that enables Machine Learning high-speed inference. Then, the model is deployed on the robot to be evaluated. Figure 35 shows this pipeline's steps.



Figure 35 – Pipeline used to train a model and validate the robot.



Source: Author

The models were optimized using Integer Quantization, which reduces the inference time in 50% while reducing network precision in 1.8% (JACOB et al., 2017). This method converts the network weight from floating-point numbers to integer values. After training, the models were quantized and converted to a TensorFlow Lite compatible model, which is required to compile the model to run on a Google Coral Edge TPU accelerator.

Instead of training the CNNs model with random weights, transfer learning was used. This method uses a pre-trained model with other datasets as a training start point, which helps solve problems that don't have much training data available (Pan; Yang, 2010). To detect objects in SSL context, there isn't enough data to train a detection model from scratch, so transfer learning was used to speed up the training and keep the learned low-level features. In this work, all the tested models were pre-trained using the COCO dataset before starting training with the SSL dataset.

The proposed dataset was evaluated using MobileNet v1 SSD (HOWARD et al., 2017), MobileNet v2 SSD (SANDLER et al., 2019), MobileDet (XIONG et al., 2020), and YOLO v4 Tiny (WANG; BOCHKOVSKIY; LIAO, 2021), which are state-of-the-art embedded object detection models.

During the tests it was perceived poor results on small objects, trying to make them better, some modifications were tested. First, the three classes were divided into nine classes. Those classes were created by dividing each class by their instance size, small, medium, or large. For an object instance to be considered small, it needs to have an area less than  $32 \times 32$ (1024) pixels, more than that but less than  $96 \times 96$ (9216) the instance is considered medium, and the remaining are considered large.

Another test conducted aiming to increase the AP for all the classes was modifying the anchors' configuration on the training. Anchors are the windows used on the bounding box search during the training. The MobileNets and MobileDet models use a fixed amount of anchors and a fixed aspect ratio, which would better cover the bounding boxes configuration on a specific dataset.


On the first test, it was used a general configuration of anchors. However, each dataset has its own type of bounding box. For instance, objects from the Goal class are usually rectangular and displayed horizontally, while Ball instances are small squares and Robot instances are medium or large squares. By default, the MobileNets and MobileDet models use aspect ratios of 0.33, 0.5, 1.0, 2.0, 3.0, where 1 the bounding box is a perfect square, values less than 1 are vertical rectangular and bigger than 1 are horizontal rectangular.

### 4.3 RUNNING AND EVALUATING

When using a trained model on an SSL robot, the primary constraint is inferring in real time. A model has to run in at least 24 FPS to be considered a real-time inference. This frame rate is acceptable with the league's objects' speed since a ball, the fastest object in the field, with a maximum speed of  $6.5m/s$  (SMALL SIZE LEAGUE TECHNICAL COMMITTEE, 2021), would move only  $27cm$  between inferences.

Object detection uses the Intersection over Union (IoU) between a predicted bounding box and a ground truth bounding box to consider it correct. The IoU value ranges from 0 to 1, where 0 means no overlap between the bounding boxes, whereas 1 means the bounding boxes perfectly match. Therefore, it is set a threshold value to consider a prediction correct. Figure 36 shows how IoU is calculated using two bounding boxes areas.

Figure 36 – Graphical representation on how to calculate IoU of two bounding boxes.



$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Source: Collected from the internet.<sup>2</sup>

Tuning the IoU threshold, it is possible to increase or decrease the precision and the recall of a trained model. For instance, the model will have better but fewer predictions when

<sup>2</sup> <<https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>>. Accessed November 12, 2021

increasing the IoU threshold. Thus the precision will increase, and the recall will decrease. Likewise, lowering the threshold will reduce the precision and improve the recall.

So, different IoU thresholds will imply different values on precision and recall, and the better way to measure this trade-off is to plot the *precision*  $\times$  *recall* curve. This curve plots the precision on the y-axis and the recall on the x-axis for different values of IoU threshold.

The models were evaluated using the metrics as the COCO dataset, which are AP and Average Recall (AR). Those metrics aim to measure the weighted mean of precisions at each IoU threshold for detecting objects in an image.

The AP for an IoU is calculated considering the area under the *precision*  $\times$  *recall* curve, and there are several ways to calculate this area. AP is calculated using a 101-point interpolation introduced by the COCO dataset (LIN et al., 2014). This interpolation consists of discretizing the *precision*  $\times$  *recall* curve in 101 points and getting those points to estimate the area under the curve. Besides that, the AP presented on the COCO dataset uses the mean of the AP calculated over 10 IoU threshold, from 0.5 to 0.95 at a step frequency of 0.05.

AR describes the area doubled under the *recall*  $\times$  *IoU* curve. This *recall*  $\times$  *IoU* curve plots the recall values for each IoU value from 0.5 to 1.0. Finally, the evaluation is made using an open-source tool (PADILLA et al., 2021), which outputs the COCO metrics to compare each model's result, given the ground-truth labels and predictions.

#### 4.4 INFERENCE ENVIRONMENT

One of the main drawbacks of using a CNNs is the requirement of a Graphical Processing Units (GPUs) to infer achieving a frequency to use in real-time detection (WANG; WEI; BROOKS, 2019). Besides, GPUs are too big to use in an SSL robot, and they have a high power consumption for the battery that fits in one of these robots. However, improvements in dedicated processors, such as the Google Coral Edge TPU, are making it possible to run inference in an embedded system with a low inference time.

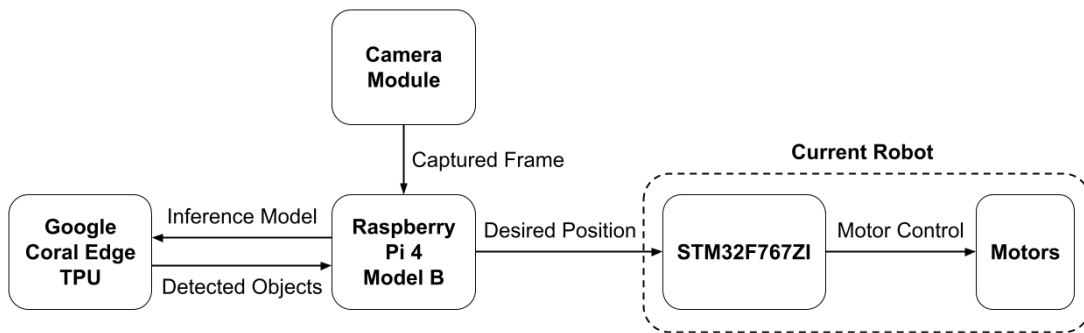
The primary constraint to this work is the environment delimitation due to the league's size restrictions (SMALL SIZE LEAGUE TECHNICAL COMMITTEE, 2021). A SSL has to fit in a cylinder with a height of 15cm and a diameter of 18cm but the technical challenge does not have a height restriction, making it possible to add another floor to the robot to fit additional hardware.

The robot used to test was a modified version of the RobôCIn v2020 (SILVA et al., 2020). All the additional modifications should have low power consumption, as the robot uses a LiPo 2200mah 4S 35C battery. This battery is enough to supply four brushless motors of 50W each and all the other robot's necessities without any modification.

The modifications on this robot consist of adding a Raspberry Pi 4 Model B, a Google Coral Edge TPU accelerator, and a camera module to the robot, composing the vision system. The camera can capture images up to 90 FPS in a resolution of  $640 \times 480$  pixels. These new components tackle the lack of computational power in the main microcontroller, an STM32F767ZI. The power consumption of a Raspberry Pi 4 with the camera module is up to 7.5W, and the Google Coral is 4.5W, which fits the power supply of the robot's battery.

The vision system's inclusion is desirable to avoid modifying the architecture and data flow of the current robot. In the current robot, the microcontroller controls the motors to operate at the desired speeds. In the new system, the Raspberry Pi receives the camera's captured frames and uses them as the input of the inference model running on the Google Coral Edge TPU. After the inference, the model outputs the detected objects to the Raspberry Pi, which computes where the robot should go and sends this position to the microcontroller. Figure 37 shows the new system architecture of the robot.

Figure 37 – Modifications on the architecture and data flow of the new robot.



Source: Author

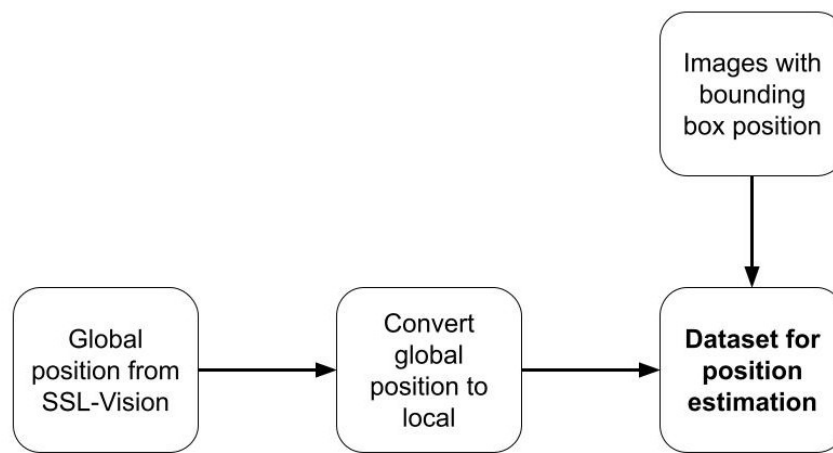
#### 4.5 DATASET TO ESTIMATE THE POSITION

The first three steps of the SSL technical challenge depend on finding and going to the ball, so this work validated the position estimation focusing only on balls. The Position Dataset needs to be constructed by having the position of the bounding boxes in a monocular image and the position from this object relative to the point where the camera is. To achieve that,

a modified SSL robot was positioned on a field with an active SSL-Vision sending the global position of the objects on the field.

Figure 38 shows the pipeline to create the dataset for position estimation. First, a SSL robot using a camera is placed on the field. This robot is actively receiving the global position of the objects on the field from the SSL-Vision. This global position is converted to a local position relative to this robot. Then, this robot takes several photos from the field while logging this data to create the dataset.

Figure 38 – Pipeline for creating the dataset for position estimation.



Source: Author

In this dataset, the bounding box position is represented by 4 values, 2 representing the bottom-left corner, and 2 representing the top-right corner. So, those points are an integer value representing a pixel. The robot and ball global positions given by the SSL-Vision are expressed in millimeters, and they are relative to the field central point. The direction in which the robot is looking is also provided in radians.

After setting the robot, the robot was placed in a static position on the field to collect the data. Each data from this dataset was compiled by putting the ball in several different positions and saving the information of that moment. The valuable information for this dataset is the bounding box position, the ball's global position, and the robot position.

The first procedure on this dataset is to transform the ball's global position to the position relative to the robot. Thus, it needs to apply two transformations on the ball position, first, moving the origin to the robot position and then rotating the axis, having the X-axis pointing in the same direction as the front of the robot.

The first transformation uses (4.1) where  $(x_{ball}, y_{ball})$  is the detected global ball position,  $(x_{robot}, y_{robot})$  is the detected robot global position, and  $(x'_{ball}, y'_{ball})$  is the ball position dislocating the origin to match the robot position.

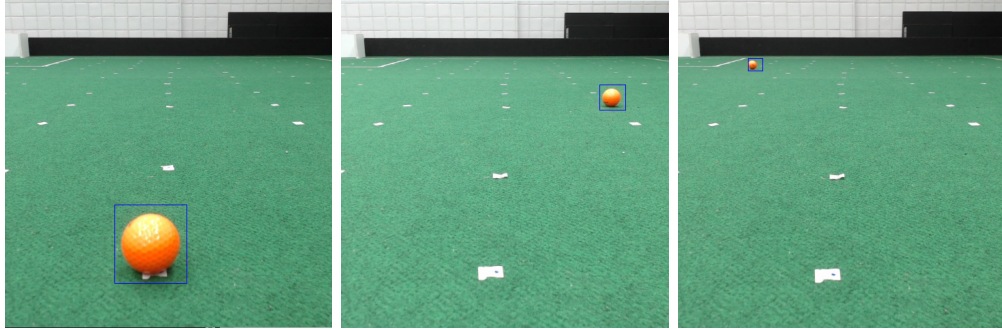
$$\begin{cases} x'_{ball} = x_{ball} - x_{robot} \\ y'_{ball} = y_{ball} - y_{robot} \end{cases} \quad (4.1)$$

The coordinates rotation uses (4.2) where  $\theta_{robot}$  is the robot angle, and  $(x_{ball_{robot}}, y_{ball_{robot}})$  is the ball position relative to the robot.

$$\begin{cases} x_{ball_{robot}} = x'_{ball} \cos(\theta_{robot}) + y'_{ball} \sin(\theta_{robot}) \\ y_{ball_{robot}} = -x'_{ball} \sin(\theta_{robot}) + y'_{ball} \cos(\theta_{robot}) \end{cases} \quad (4.2)$$

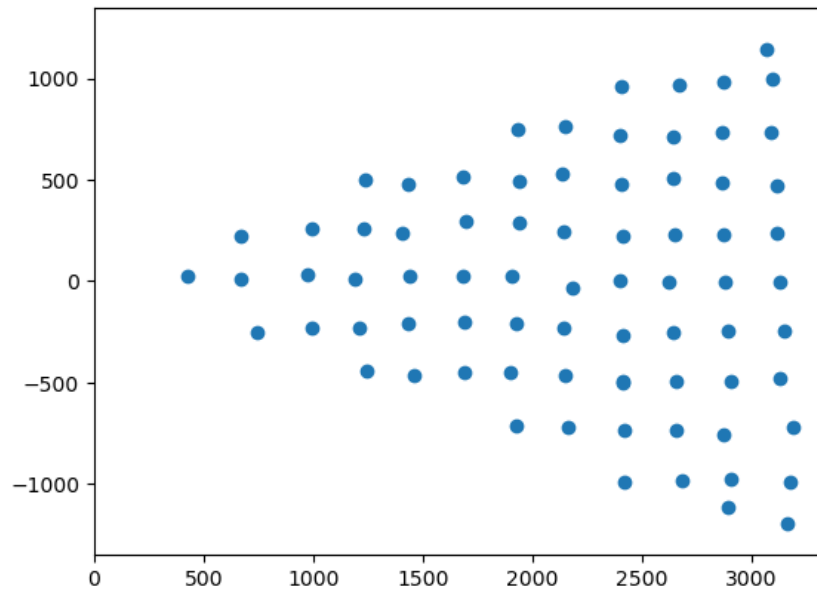
This dataset has 76 examples of different ball positions, which were collected using this procedure. Furthermore, Figure 39 shows examples of images and bounding boxes used on this dataset. Finally, Figure 40 shows the position of each point on this dataset relative to where the camera is located.

Figure 39 – Some examples of the dataset containing detected balls



Source: Author

Figure 40 – Ball's relative position to the robot with the camera. Consider the robot located on position  $(0, 0)$  and having looking to the same direction as the X-axis



Source: Author

Besides those 76 training points, it was also gathered the same 30 examples used (MELO; BARROS, 2022), so it would be possible to compare the results for the approach presented in this work to it.

#### 4.6 TRAINING EVALUATING POSITION MODEL

The Position Dataset defines the bounding box position as the input and the position relative to the robot as the output, so the model to estimate the object's position it would have four integer values as input and two real values as the output. As this problem doesn't have the same complexity as detecting an object on an image, this work fixed the desired model as MLP and used a hyperparameter search to find the best network architecture.

Other works such as DisNet (HASEEB et al., 2018) used a similar approach of searching the best MLP architecture. In the DisNet work, it only searched the number of hidden layers and how many neurons each layer should have, and it found the best architecture for their context with 3 hidden layers, each having 100 neurons.

For the proposed approach, besides searching for the number of hidden layers and how many neurons are in each layer, it was also added the learning rate on the search. This hyperparameter was also added to avoid the model being stuck in a sub-optimal solution.

The dataset proposed by us is divided into a training set having 76 examples and a test set with 30 instances, close to a 70/30 widely used on ML context. Besides, the dataset must have a validation set to search for the best architecture. This is needed to avoid overfitting in the test set on the validation. So, in the training phase, a K-Fold cross-validation strategy divided the training set into 5 folds.

After that, it is necessary to define the hyperparameter to search and their range on the search. The plan for this model is to run the inference embedded in a robot, so one of the parameters it has to be optimized is the number of hidden layers and how many neurons each layer would have. Another optimized parameter was the learning rate to avoid the model being stuck on an early solution or not learning enough from the data.

As the training and validation dataset has only 76 instances, this model doesn't need to be that deep, as it wouldn't learn much. So, in the parameters search, a network with 2, 3, or 5 hidden layers was used, and 10, 30, 50, or 100 neurons per layer. For the learning rate, it was used 0.01 and 0.001 on the search. The search strategy was a grid search since it searches for 3 hyperparameters, and the search space is small. Table 6 summarizes all the parameters used on the MLP architecture search and the fixed parameters.

Table 6 – Hyperparameters used on the Grid search to find the best MLP architecture.

Hyperparameter	Searched Values
Number of Hidden Layers	2, 3, 5
Neurons per Hidden Layers	10, 30, 50, 100
Learning Rate	0.01, 0.001
Solver	Adam
Maximum iterations	50000

Source: Author

The image used on the object detection has  $224 \times 224$  pixels, so the input is in that range. The output distance is given in millimeters and could be in the range of the field size  $9000mm \times 6000mm$ . It is noticeable that the input and the output have a range that could impact the training process. So, one of the tests used in this work was to normalize only the input or normalize the input and the output before training and evaluation. The normalization method uses normalized values to the training set minimum and maximum values.



## 4.7 EVALUATING POSITION DATASET

This work used the RMSE metric to evaluate the trained models. This metric measures the mean error between a ground truth instance and a predicted output. This metric is useful when the model wants to keep a low error in all predicted samples. RMSE is defined as (4.3) where  $N$  is the number of samples,  $\hat{y}_i$  is the predicted value, and  $y_i$  is the desired value. This metric is the right one for this context because the best model needs to have the slightest error in estimating the position.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2} \quad (4.3)$$

## 5 RESULTS

This chapter is divided into two sections. Section 5.1 details the results achieved on the object detection models trained with the proposed object detection dataset. Section 5.2 presents the result for the position estimation models.

### 5.1 OBJECT DETECTION RESULTS

Table 7 shows the AP for the four models separated by IoU threshold and object area. AP for Medium and Large objects shows how powerful these models can be in less challenging scenarios where the object is much closer to the robot. However, the results for Small objects are worse than for Medium and Large objects, which indicates a high false-positive rate. This error occurs due to the low information on objects of Small size.

Table 7 – Average Precision (AP) for each model by Intersection over Union (IoU) threshold, in  $AP_{50}$  the threshold used is 0.5 and  $AP_{75}$  is 0.75, and detected object area, where  $AP_S$ ,  $AP_M$ , and  $AP_L$  stands for the result by each detection size, Small, Medium or Large.

Method	AP	$AP_{50}$	$AP_{75}$	$AP_S$	$AP_M$	$AP_L$
<b>MobileNet v1 SSD</b>	<b>55.77%</b>	<b>84.67%</b>	<b>59.66%</b>	<b>32.51%</b>	<b>78.61%</b>	<b>89.62%</b>
<b>MobileNet v2 SSD</b>	43.41%	74.39%	44.14%	23.18%	64.35%	86.27%
<b>MobileDet</b>	36.37%	66.54%	36.83%	16.60%	60.67%	81.53%
<b>YOLOv4 Tiny</b>	42.17%	62.24%	54.09%	27.34%	69.05%	58.84%

Source: Author

From the AP perspective, the MobileNet v1 SSD had the best result overall and for Large objects. However, the AP for Large objects on the YOLO v4 tiny model was worse than for Medium objects, which is a peculiar behavior since the other models achieve better AP when detecting Large objects. This result can indicate that YOLO v4 tiny needs more labeled data with a large size, as there are only 38 objects with this size on the proposed dataset.

Table 8 shows the result for the detection dataset separating the AP by each object class. Based on these results, it can be seen that all three classes had a good result overall, achieving a AP greater than 80% on the MobileNet v1 model. The classes Ball and Robot had a worse result than the Goal class, and this can be explained by the fact that those classes has more small objects.

Table 8 – Average Precision (AP) for each model by classes, where  $AP_{Ball}$ ,  $AP_{Robot}$  and  $AP_{Goal}$  is the Average Precision for each of class Ball, Robot and Goal respectively.

Method	$AP_{Ball}$	$AP_{Robot}$	$AP_{Goal}$
<b>MobileNet v1 SSD</b>	<b>83.54%</b>	<b>80.20%</b>	<b>94.40%</b>
<b>MobileNet v2 SSD</b>	69.61%	67.38%	90.14%
<b>MobileDet</b>	63.32%	64.62%	75.88%

Source: Author

Table 9 shows the AR results separated by maximum detection per image and detected object size. A high AR is important for Robot and Goal classes because the robot uses it to avoid colliding with another robot when navigating and helps the robot identify the Goal faster. The Robot and Goal classes' represents all of the Large objects and 95% of Medium objects, as shown in Figure 34.

Table 9 – Average Recall (AR) for each model by maximum detection per image,  $AR_1$  for at most 1 object per image and  $AR_{10}$  for 10 objects per image, and detected object area, where  $AR_S$ ,  $AR_M$ , and  $AR_L$  stands for the result by each detection size, Small, Medium or Large.

Method	$AR_1$	$AR_{10}$	$AR_S$	$AR_M$	$AR_L$
<b>MobileNet v1 SSD</b>	<b>37.75%</b>	<b>62.87%</b>	<b>40.62%</b>	<b>82.18%</b>	<b>91.00%</b>
<b>MobileNet v2 SSD</b>	34.76%	50.60%	29.28%	66.21%	87.00%
<b>MobileDet</b>	30.62%	43.66%	22.96%	65.41%	85.00%
<b>YOLOv4 Tiny</b>	36.72%	45.36%	30.41%	74.16%	64.00%

Source: Author

The obtained result of AR for Medium and Large objects sizes shows a high detection rate, with the Mobilenet v1 SSD as the best AR results overall. However, it was also observed that YOLO v4 tiny had worse results for Large objects, which supports the necessity of more samples of Large objects for this model.

Table 10 shows each model's inference frequency, where the MobileNet v1 SSD had the best FPS overall, but MobileNet v2 SSD and MobileDet had a rate that fits the requirement of at least 24 FPS. However, the YOLO v4 Tiny had a bad result with only 10 FPS, which is caused by the lack of architectural optimizations on the network compared with the other evaluated models. As the YOLO v4 Tiny showed a bad FPS, it was removed for further tests since none of them aimed to improve the inference speed.

Table 10 – Mean inference frequency in Frames Per Second (FPS) for each tested model.

Method	FPS
<b>MobileNet v1 SSD</b>	<b>94</b>
<b>MobileNet v2 SSD</b>	78
<b>MobileDet</b>	87
<b>YOLOv4 Tiny</b>	10

Source: Author

Table 11 shows the AP results for the dataset with the objects instances separated by size. Comparing these results with the ones on Table 7 shows that for MobileNet v1 and MobileDet improved the AP for the small objects, showing that division could help the model to detect objects by their size. However, this approach hurt the AP for all tested models' medium and large objects detection. This happened since those type of objects has fewer instances to learn. Furthermore, this approach also decreases the AP for small objects in MobileNet v2.

Table 11 – Average Precision (AP) for each model using the dataset dividing the instances by size.

Method	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<b>MobileNet v1 SSD</b>	50.40%	68.30%	54.66%	<b>48.58%</b>	<b>68.30%</b>	56.93%
<b>MobileNet v2 SSD</b>	36.06%	57.46%	40.79%	10.92%	49.33%	51.71%
<b>MobileDet</b>	<b>50.47%</b>	<b>68.86%</b>	<b>55.52%</b>	35.81%	67.36%	<b>63.76%</b>

Source: Author

To find the best anchor configuration, this work used the bounding box on the training set as input for k-means to group the bounding boxes by their aspect ratio. Before running the k-means, the number of groups has to be defined. The proposed approach runs the k-means using 3, 5, 7, 9 groups.

Table 12 shows the result for each of the number of groups. Diving the anchors in nine aspect ratios showed the best accuracy to fit the bounding box in the training dataset. Based on these results, the horizontally rectangular goal instances represented large values of aspect ratio, 3.2 and 3.87. Moreover, as this dataset has no vertical rectangular, the k-means didn't find any centroid with low aspect ratios.

Table 12 – K-Means results to find the best aspect ratio configuration for the anchors.

# of clusters	Accuraccy	Aspect Ratio Centroids
3	65.46%	0.9, 1.0, 1.87
5	74.71%	0.89, 0.91, 0.92, 1.0, 3.87
7	77.85%	0.89, 0.92, 0.92, 1.0, 1.0, 1.09, 3.87
9	<b>80.91%</b>	0.87, 0.89, 0.92, 0.94, 1.0, 1.0, 1.06, 3.2, 3.87

Source: Author

Table 13 shows the AP results for the dataset using the best anchors values using the nine aspect ratios found using the k-means on the anchors' configuration. Comparing this result with the ones on Table 7, it only shows a slight improvement on the MobileDet model. In the other two MobileNets, it didn't change the AP significantly.

Table 13 – Average Precision (AP) for each model using the best anchors aspect ratio using k-means.

Method	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<b>MobileNet v1 SSD</b>	<b>50.18%</b>	<b>79.27%</b>	<b>54.22%</b>	31.98%	<b>70.65%</b>	<b>91.98%</b>
<b>MobileNet v2 SSD</b>	35.70%	63.13%	36.29%	22.76%	48.47%	77.54%
<b>MobileDet</b>	48.65%	77.23%	52.51%	<b>34.39%</b>	64.41%	86.55%

Source: Author

Table 14 shows the AP results for the dataset combining the objects separated by size and the anchors' aspect ratio found using k-means. Combining those approaches showed to be the worst option since it hurts the AP for all the models.

Table 14 – Average Precision (AP) for each model using the dataset dividing the instances by size and the best anchors aspect ratio using k-means.

Method	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<b>MobileNet v1 SSD</b>	<b>46.03%</b>	<b>66.48%</b>	<b>50.61%</b>	34.35%	<b>62.61%</b>	44.62%
<b>MobileNet v2 SSD</b>	30.81%	50.10%	32.99%	29.50%	39.21%	40.94%
<b>MobileDet</b>	40.79%	61.00%	45.17%	<b>36.56%</b>	49.89%	<b>54.53%</b>

Source: Author

Table 15 summarizes the AP results for all the approaches tested on this work. The MobileNet using the raw dataset had the best result overall, but adjusting the dataset to separate

Table 15 – Average Precision (AP) for each model separated by all the approaches tested on this work.

Method	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<b>MobileNet v1</b>						
<b>Raw Dataset</b>	<b>55.77%</b>	<b>84.67%</b>	<b>59.66%</b>	32.51%	<b>78.61%</b>	89.62%
<b>Dataset by size</b>	50.40%	68.30%	54.66%	<b>48.58%</b>	68.30%	56.93%
<b>Anchor Kmeans</b>	50.18%	79.27%	54.22%	31.98%	70.65%	<b>91.98%</b>
<b>Dataset by size + Anchor Kmeans</b>	46.03%	66.48%	50.61%	34.35%	62.61%	44.62%
<b>MobileNet v2</b>						
<b>Raw Dataset</b>	43.41%	74.39%	44.14%	23.18%	64.35%	86.27%
<b>Dataset by size</b>	36.06%	57.46%	40.79%	10.92%	49.33%	51.71%
<b>Anchor Kmeans</b>	35.70%	63.13%	36.29%	22.76%	48.47%	77.54%
<b>Dataset by size + Anchor Kmeans</b>	30.81%	50.10%	32.99%	29.50%	39.21%	40.94%
<b>MobileDet</b>						
<b>Raw Dataset</b>	36.37%	66.54%	36.83%	16.60%	60.67%	81.53%
<b>Dataset by size</b>	50.47%	68.86%	55.52%	35.81%	67.36%	63.76%
<b>Anchor Kmeans</b>	48.65%	77.23%	52.51%	34.39%	64.41%	86.55%
<b>Dataset by size + Anchor Kmeans</b>	40.79%	61.00%	45.17%	36.56%	49.89%	54.53%
<b>YOLOv4 Tiny</b>						
<b>Raw Dataset</b>	42.17%	62.24%	54.09%	27.34%	69.05%	58.84%

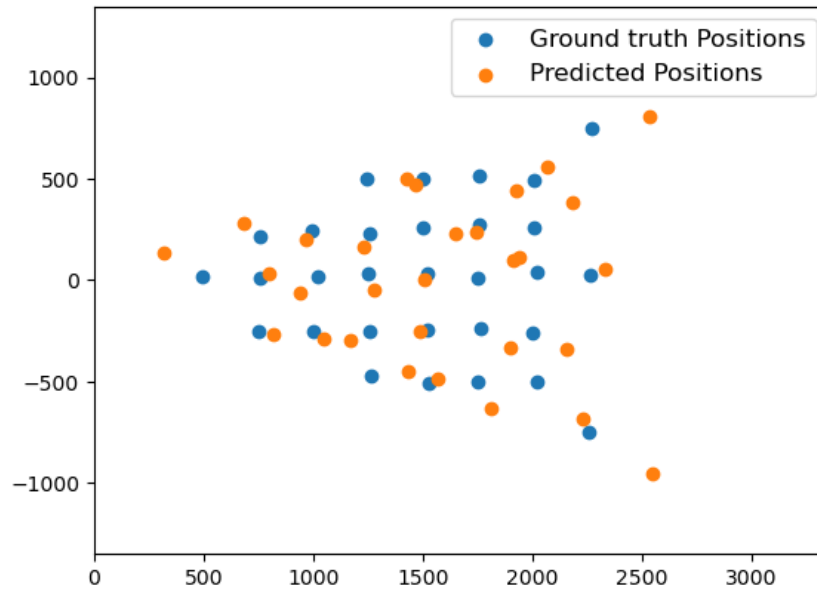
Source: Author

the instances by size, showed that it could improve the detection of small objects.

## 5.2 POSITION ESTIMATOR RESULTS

The position estimator results will be divided into two parts, without any normalization on the data, with a normalization just on the input and normalizing both the input and the output.

Figure 41 – Predicted and ground truth positions for MLP architecture without using normalization



Source: Author

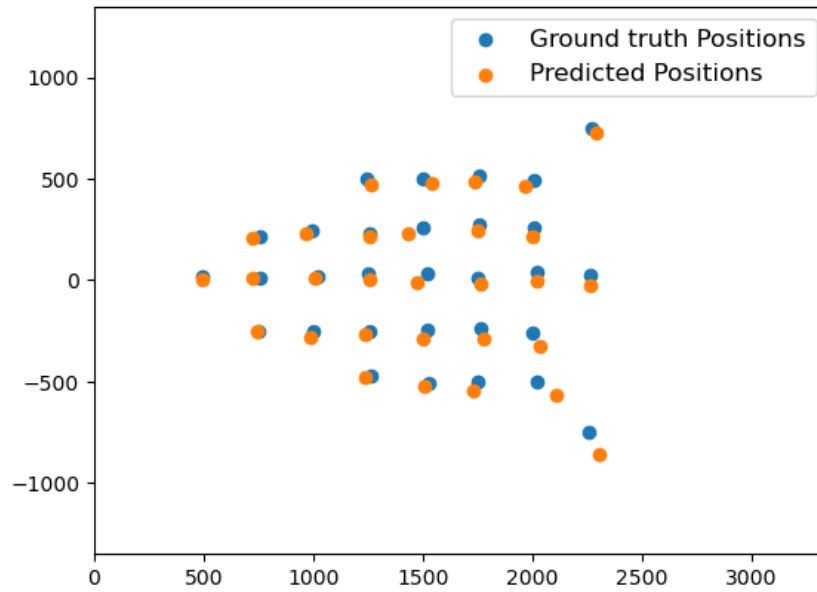
The best model without normalizing the data had only two hidden layers, with 10 neurons in each layer, using a learning rate of 0.001. This model achieved a RMSE of  $109.48mm$ . Figure 41 shows the ground-truth position for the test set and the predicted position for this model.

Normalizing just the input achieved a RMSE of  $35.17mm$ , and the best architecture found was having 5 hidden layers with 50 neurons each and a learning rate of 0.001. This improved the result from the model without normalization because the normalization made it easy for the model to train since the variation on the input is much smaller. Figure 42 shows the ground-truth position for the test set and the predicted position for this model.

Using the normalization in the input and the output, the best architecture found was with a learning rate of 0.001 and 5 hidden layers with 100 neurons each. This best model achieved a RMSE of  $34.88mm$ . This improvement is due to the output being in millimeters, and it is sensitive to any small change. Figure 43 shows the ground-truth position for the test set and the predicted position for this model.

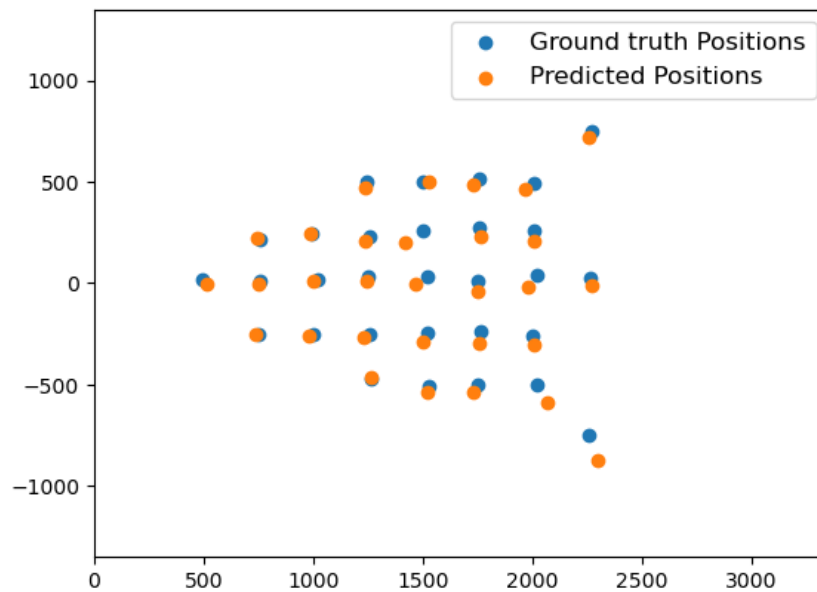
Table 16 compares our result to the result presented by (MELO; BARROS, 2022). It shows that the results normalizing the data are close to 50% better than those presented by (MELO; BARROS, 2022). Besides that, the construction of the dataset presented in our work is much simpler because their work had to hand annotate the points where the ball touched the ground in each image.

Figure 42 – Predicted and ground truth positions for MLP architecture normalizing the input



Source: Author

Figure 43 – Predicted and ground truth positions for MLP architecture normalizing the input and the output



Source: Author

Table 16 – RMSE for the proposed architecture compared with the results from (MELO; BARROS, 2022)

Method	RMSE
(MELO; BARROS, 2022)	67.32mm
MLP with raw data (ours)	109.48mm
MLP normalizing the input (ours)	35.17mm
MLP normalizing the input and the output (ours)	<b>34.88mm</b>

Source: Author



Table 17 – Inference time for the proposed architecture

Method	Inference time
<b>MLP with raw data (ours)</b>	$0.385ms$
<b>MLP normalizing the input (ours)</b>	$0.055ms$
<b>MLP normalizing the input and the output (ours)</b>	$0.069ms$

Source: Author

Table 17 presents the inference time for the tested architectures to estimate the ball position using MLP. It shows that all the proposed approaches can be considered real-time since real-time would be run in less than  $41.67ms$ . Besides, normalizing the data will give a faster than just use the raw data.

## 6 CONCLUSION

This work had as the main objective to give the ability to a SSL robot to detect nearby objects and estimate their position. It was divided into two phases, first, detect the objects on a field and then use those detected objects to estimate their position. To detect the objects, it presented an open-source labeled dataset and a benchmark for object detection for SSL. The proposed dataset guarantees its variety with images extracted from different sources under distinct lighting conditions and camera configurations. The labeled objects are Robot, Ball, and Goal, which are the essential objects found during an SSL game. The dataset's images can contain multiple instances of these objects, including no objects per image.

It also presented a pipeline to train a CNNs and deploy it on an embedded device with limited computational power. The results show that CNNs are robust to variable light conditions and can detect robots with different structures. This result contrasts with using color segmentation with scan lines. Color segmentation can be easily disturbed by these circumstances since it needs fine-tuning parameters that rely on image saturation and brightness. This thesis uses the proposed dataset to evaluate AP, AR, and FPS of four different CNNs models on constrained hardware. Furthermore, this paper highlights the importance of model architectural optimizations.

The presented dataset has a similar size as other datasets used on other RoboCup leagues. However, it is smaller than the general proposed object detection datasets. So, data augmentation techniques were applied to increase diversity and model generalization. A future dataset improvement is adding images from game situations and different field configurations. Besides, increasing the number of distinctive robots' instances and the number of Large images will boost the dataset's robustness.

Using the trained model, a new dataset was created to estimate the object's position relative to the robot. This dataset combined the detected bounding boxes with the global position from SSL-Vision. Furthermore, the position estimator used a technique to search the best hyperparameters of a MLP based on the strategy already used on the DisNet. Besides that, this approach also improved the RMSE on the other approach in the same context.

Besides all the contributions stated above, the object detection dataset and the evaluation baseline were published at the RoboCup 2021 Symposium (FERNANDES; RODRIGUES; BARROS, 2022). Object detection and position estimation were also used by the RobôCIn team on the

RoboCup 2022 SSL Vision Blackout Challenge, where the team achieved second place on this challenge.

Future work from this thesis would be:

- Improve the detection of small objects in the first phase.
- Expand the position estimation for all relevant classes in the context of the SSL.
- Create a complete dataset to detect an object and estimate its position using only one model.
- Uses the idea from the position estimator to have an automatic tool to create a dataset on a SSL game.
- Improve the position estimation on objects far from the robot.
- Improve the hyperparameter search strategy for the position estimation model.
- Add a tracking system to predict the objects' positions, keeping the detected objects' position when the detection fails.
- Use better data augmentation strategies to balance the detection dataset and make it more robust to different environments and different conditions.
- Analyze and compare the results from the different methods to create the detection dataset.

## REFERENCES

- ALBANI, D.; YOUSSEF, A.; SURIANI, V.; NARDI, D.; BLOISI, D. D. A deep learning approach for object recognition with nao soccer robots. In: SPRINGER. *Robot World Cup*. [S.l.], 2016. p. 392–403.
- BEAR, M.; CONNORS, B.; PARADISO, M. A. *Neuroscience: Exploring the Brain, Enhanced Edition: Exploring the Brain*. [S.l.]: Jones & Bartlett Learning, 2020.
- BENDER, G.; LIU, H.; CHEN, B.; CHU, G.; CHENG, S.; KINDERMANS, P.; LE, Q. Can weight sharing outperform random architecture search? an investigation with tunas. *CoRR*, abs/2008.06120, 2020. Available at: <<https://arxiv.org/abs/2008.06120>>.
- BERG, T. L.; BERG, A. C. Finding iconic images. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. [S.l.: s.n.], 2009. p. 1–8.
- BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020.
- BURKHARD, H.; DUHAUT, D.; FUJITA, M.; LIMA, P.; MURPHY, R.; ROJAS, R. The road to robocup 2050. *Robotics & Automation Magazine, IEEE*, v. 9, p. 31 – 38, 07 2002.
- CAMPBELL, M.; JR, A. J. H.; HSU, F.-h. Deep blue. *Artificial intelligence*, Elsevier, v. 134, n. 1-2, p. 57–83, 2002.
- CAUCHY, A. et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, v. 25, n. 1847, p. 536–538, 1847.
- CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989.
- DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 248–255.
- FERNANDES, R.; RODRIGUES, W. M.; BARROS, E. Dataset and benchmarking of real-time embedded object detection for robocup ssl. In: ALAMI, R.; BISWAS, J.; CAKMAK, M.; OBST, O. (Ed.). *RoboCup 2021: Robot World Cup XXIV*. Cham: Springer International Publishing, 2022. p. 53–64. ISBN 978-3-030-98682-7.
- FERREIN, A.; STEINBAUER, G. 20 years of robocup. *KI-Künstliche Intelligenz*, Springer, v. 30, n. 3, p. 225–232, 2016.
- FIEDLER, N.; BESTMANN, M.; HENDRICH, N. Imagetagger: An open source online platform for collaborative image labeling. In: SPRINGER. *RoboCup 2018: Robot World Cup XXII*. [S.l.], 2018.
- FOUNTAS, Z. Imperial college spiking neural networks for human-like avatar control in a simulated environment. 07 2021.
- GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? the kitti vision benchmark suite. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2012.

- GOMI, H.; OSU, R. Task-dependent viscoelasticity of human multijoint arm and its spatial characteristics for interaction with environments. *Journal of neuroscience*, Soc Neuroscience, v. 18, n. 21, p. 8965–8978, 1998.
- Gökçe, F.; ÜÇOLUK, G.; ŞAHİN, E.; KALKAN, S. Vision-based detection and distance estimation of micro unmanned aerial vehicles. *Sensors*, v. 15, n. 9, p. 23805–23846, 2015. ISSN 1424-8220. Available at: <<https://www.mdpi.com/1424-8220/15/9/23805>>.
- HAGEN, S. The mind's eye. *Rochester Review*, v. 74, n. 4, p. 32–37, 2012.
- HAN, Y.; ROIG, G.; GEIGER, G.; POGGIO, T. Scale and translation-invariance for novel objects in human vision. *Scientific reports*, Nature Publishing Group, v. 10, n. 1, p. 1–13, 2020.
- HASEEB, M. A.; GUAN, J.; RISTIC-DURRANT, D.; GRÄSER, A. Disnet: a novel method for distance estimation from monocular camera. *10th Planning, Perception and Navigation for Intelligent Vehicles (PPNIV18)*, IROS, 2018.
- HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016.
- HERMANN, M.; PENTEK, T.; OTTO, B. Design principles for industrie 4.0 scenarios. In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. [S.l.: s.n.], 2016. p. 3928–3937.
- HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017.
- HUANG, J.; RATHOD, V.; SUN, C.; ZHU, M.; KORATTIKARA, A.; FATHI, A.; FISCHER, I.; WOJNA, Z.; SONG, Y.; GUADARRAMA, S.; MURPHY, K. *Speed/accuracy trade-offs for modern convolutional object detectors*. 2017.
- JACOB, B.; KLIGYS, S.; CHEN, B.; ZHU, M.; TANG, M.; HOWARD, A.; ADAM, H.; KALENICHENKO, D. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017.
- KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry*. München, 2013. Available at: <[http://forschungsunion.de/pdf/industrie\\_4\\_0\\_final\\_report.pdf](http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf)>.
- KITANO, H.; ASADA, M.; KUNIYOSHI, Y.; NODA, I.; OSAWA, E. Robocup: The robot world cup initiative. In: *Proceedings of the First International Conference on Autonomous Agents*. New York, NY, USA: ACM, 1997. (AGENTS '97), p. 340–347. ISBN 0-89791-877-0. Available at: <<http://doi.acm.org/10.1145/267658.267738>>.
- KRIZHEVSKY, A.; HINTON, G. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, v. 40, n. 7, p. 1–9, 2010.

- KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGESS, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012. v. 25. Available at: <<https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>>.
- KUMAR, A. C.; BHANDARKAR, S. M.; PRASAD, M. Depthnet: A recurrent neural network architecture for monocular depth prediction. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. [S.l.: s.n.], 2018. p. 396–3968.
- LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, v. 1, n. 4, p. 541–551, 12 1989. ISSN 0899-7667. Available at: <<https://doi.org/10.1162/neco.1989.1.4.541>>.
- LECUN, Y.; CORTES, C. MNIST handwritten digit database. 1998. Available at: <<http://yann.lecun.com/exdb/mnist/>>.
- LECUN, Y. A.; BOTTOU, L.; ORR, G. B.; MÜLLER, K.-R. Efficient backprop. In: *Neural networks: Tricks of the trade*. [S.l.]: Springer, 2012. p. 9–48.
- LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: FLEET, D.; PAJDLA, T.; SCHIELE, B.; TUYTELAARS, T. (Ed.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. p. 740–755. ISBN 978-3-319-10602-1.
- LINNAINMAA, S. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, p. 6–7, 1970.
- LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S. E.; FU, C.; BERG, A. C. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. Available at: <<http://arxiv.org/abs/1512.02325>>.
- Luo, S.; Lu, H.; Xiao, J.; Yu, Q.; Zheng, Z. Robot detection and localization based on deep learning. In: *2017 Chinese Automation Congress (CAC)*. [S.l.: s.n.], 2017. p. 7091–7095.
- MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. et al. Rectifier nonlinearities improve neural network acoustic models. In: CITESEER. *Proc. icml*. [S.l.], 2013. v. 30, n. 1, p. 3.
- MARR, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. New York, NY, USA: Henry Holt and Co., Inc., 1982. ISBN 0716715678.
- MARVIN, M.; SEYMOUR, A. P. *Perceptrons*. [S.l.]: MIT Press, 1969.
- MASOUMIAN, A.; MAREI, D. G. F.; ABDULWAHAB, S.; CRISTIANO, J.; PUIG, D.; RASHWAN, H. A. Absolute distance prediction based on deep learning object detection and monocular depth estimation models. *CoRR*, abs/2111.01715, 2021. Available at: <<https://arxiv.org/abs/2111.01715>>.
- MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MELO, J. G.; BARROS, E. An embedded monocular vision approach for ground-aware objects detection and position estimation. *arXiv preprint arXiv:2207.09851*, 2022.

MIDDLE SIZE LEAGUE TECHNICAL COMMITTEE. *Middle Size League Rules and Regulations for 2021*. 2021. <[https://cdn.robocup.org/msl/wp/2021/02/Rulebook\\_MSL2021\\_v22.0.pdf](https://cdn.robocup.org/msl/wp/2021/02/Rulebook_MSL2021_v22.0.pdf)>, last checked on 2021-04-27.

MITCHELL, T. M. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2.

MOKHTAR, K. Z.; MOHAMAD-SALEH, J. An oil fraction neural sensor developed using electrical capacitance tomography sensor data. *Sensors (Basel, Switzerland)*, v. 13, p. 11385–406, 09 2013.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *lcm1*. [S.l.: s.n.], 2010.

NAKAMURA, Y.; YAMANE, K.; FUJITA, Y.; SUZUKI, I. Somatosensory computation for man-machine interface from motion-capture data and musculoskeletal human model. *IEEE Transactions on Robotics*, IEEE, v. 21, n. 1, p. 58–66, 2005.

NEVES, A. J.; PINHO, A. J.; MARTINS, D. A.; CUNHA, B. An efficient omnidirectional vision system for soccer robots: From calibration to object detection. *Mechatronics*, Elsevier, v. 21, n. 2, p. 399–410, 2011.

PADILLA, R.; PASSOS, W. L.; DIAS, T. L. B.; NETTO, S. L.; SILVA, E. A. B. da. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, v. 10, n. 3, 2021. ISSN 2079-9292. Available at: <<https://www.mdpi.com/2079-9292/10/3/279>>.

Pan, S. J.; Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, n. 10, p. 1345–1359, 2010.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016.

REZAEI, M.; TERAUCHI, M.; KLETTE, R. Robust vehicle detection and distance estimation under challenging lighting conditions. *IEEE Transactions on Intelligent Transportation Systems*, v. 16, n. 5, p. 2723–2743, 2015.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015.

RUSSELL, S.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.: s.n.], 2002.

RÜSSMANN, M.; LORENZ, M.; GERBERT, P.; WALDNER, M.; JUSTUS, J.; ENGEL, P.; HARNISCH, M. Industry 4.0: The future of productivity and growth in manufacturing industries. *Boston Consulting Group*, Boston, MA, USA:, v. 9, n. 1, p. 54–89, 2015.

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019.

Seel, Fabio and Jut, Sabolc. *On-Board Computer Vision for Autonomous Ball Interception*. 2019. <[https://tigers-mannheim.de/download/papers/2019-BallIntercept\\_TC-Seel\\_Jut.pdf](https://tigers-mannheim.de/download/papers/2019-BallIntercept_TC-Seel_Jut.pdf)>, last checked on 2021-04-12.

SIEGWART, R.; NOURBAKHSI, I. R.; SCARAMUZZA, D. *Introduction to autonomous mobile robots*. [S.l.]: MIT press, 2011.

SIFRE, L.; MALLAT, P. S. *Ecole Polytechnique, CMAP PhD thesis Rigid-Motion Scattering For Image Classification Author:*. 2014.

SILVA, C.; ALVES, C.; MARTINS, F.; MACHADO, J. G.; DAMURIE, J.; CAVALCANTI, L.; VINICIUS, M.; SOUSA, R.; RODRIGUES, R.; FERNANDES, R. et al. Robôcin 2020 team description paper. 2020.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. V. D.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M. et al. Mastering the game of go with deep neural networks and tree search. *nature*, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

SMALL SIZE LEAGUE TECHNICAL COMMITTEE. *RoboCup 2020 SSL Vision Blackout Technical Challenge Rules*. 2020. <<https://ssl.roboocup.org/wp-content/uploads/2020/07/2020-ssl-vision-blackout-rules.pdf>>, last checked on 2021-04-12.

SMALL SIZE LEAGUE TECHNICAL COMMITTEE. *Laws of the RoboCup Small Size League 2021*. 2021. <<https://roboocup-ssl.github.io/ssl-rules/sslrules.pdf>>, last checked on 2021-04-12.

STANDARD PLATFORM LEAGUE TECHNICAL COMMITTEE. *RoboCup Standard Platform League (NAO) Challenges & Rule Book*. 2021. <<https://cdn.roboocup.org/spl/wp/2021/01/SPL-Rules-2021.pdf>>, last checked on 2021-04-27.

STROEVE, S. Impedance characteristics of a neuromusculoskeletal model of the human arm i. posture control. *Biological cybernetics*, Springer, v. 81, n. 5, p. 475–494, 1999.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCHE, V.; RABINOVICH, A. Going deeper with convolutions. In: *Computer Vision and Pattern Recognition (CVPR)*. [s.n.], 2015. Available at: <<http://arxiv.org/abs/1409.4842>>.

TORRALBA, A.; EFROS, A. A. Unbiased look at dataset bias. In: *IEEE. CVPR 2011*. [S.l.], 2011. p. 1521–1528.

Tzutalin. *LabelImg*. 2015. <<https://github.com/tzutalin/labelImg>>, last checked on 2021-04-12.



- WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2021.
- WANG, Y.; WEI, G.; BROOKS, D. Benchmarking tpu, gpu, and CPU platforms for deep learning. *CoRR*, abs/1907.10701, 2019. Available at: <<http://arxiv.org/abs/1907.10701>>.
- WEITZENFELD, A.; BISWAS, J.; AKAR, M.; SUKVICHAI, K. Robocup small-size league: Past, present and future. In: BIANCHI, R. A. C.; AKIN, H. L.; RAMAMOORTHY, S.; SUGIURA, K. (Ed.). *RoboCup 2014: Robot World Cup XVIII*. Cham: Springer International Publishing, 2015. p. 611–623.
- WICHT, B. *Deep Learning feature Extraction for Image Processing*. Phd Thesis (PhD Thesis), 01 2018.
- XIAO, J.; XIONG, D.; YAO, W.; YU, Q.; LU, H.; ZHENG, Z. Building software system and simulation environment for robocup msl soccer robots based on ros and gazebo. In: \_\_\_\_\_. [S.l.: s.n.], 2017. v. 707, p. 597–631. ISBN 978-3-319-54926-2.
- XIONG, Y.; LIU, H.; GUPTA, S.; AKIN, B.; BENDER, G.; KINDERMANS, P.-J.; TAN, M.; SINGH, V.; CHEN, B. *MobileDets: Searching for Object Detection Architectures for Mobile Accelerators*. 2020.
- ZHAO, Z.-Q.; ZHENG, P.; XU, S. tao; WU, X. *Object Detection with Deep Learning: A Review*. 2019.
- ZHU, J.; FANG, Y.; ABU-HAIMED, H.; LIEN, K.; FU, D.; GU, J. Learning object-specific distance from a monocular image. *CoRR*, abs/1909.04182, 2019. Available at: <<http://arxiv.org/abs/1909.04182>>.
- ZICKLER, S.; LAUE, T.; BIRBACH, O.; WONGPHATI, M.; VELOSO, M. Ssl-vision: The shared vision system for the robocup small size league. In: BALTES, J.; LAGOUDAKIS, M. G.; NARUSE, T.; GHIDARY, S. S. (Ed.). *RoboCup 2009: Robot Soccer World Cup XIII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 425–436. ISBN 978-3-642-11876-0.