



Object Detection in Omnidirectional Images

Master degree in Computer Engineering - Mobile Computing

Francisco António Agostinho Henriques

Dissertation under the supervision of Professors Catarina Silva, Joana Costa and Pedro Assunção

Leiria, November of 2020

Originality and Copyright

This dissertation is original, made only for this purpose, and all authors whose studies and publications were used to complete it are duly acknowledged.

Partial reproduction of this document is authorized, provided that the Author is explicitly mentioned, as well as the study cycle, i.e., Master degree in Computer Engineering - Mobile Computing, 2019/2020 academic year, of the School of Technology and Management of the Polytechnic of Leiria, and the date of the public presentation of this work.

Dedication

This dissertation is dedicated to my parents, Carlos Henriques and Elsa Oliveira, who supported and motivated me to finish the masters, and to my girlfriend, Sara Dionísio who inspired every day with her determination.

Acknowledgments

This work was partially supported by project ARoundVision CENTRO-01-0145-FEDER-030652, Instituto de Telecomunicações - Delegação de Leiria.

Abstract

Nowadays, computer vision (CV) is widely used to solve real-world problems, which pose increasingly higher challenges. In this context, the use of omnidirectional video in a growing number of applications, along with the fast development of Deep Learning (DL) algorithms for object detection, drives the need for further research to improve existing methods originally developed for conventional 2D planar images. However, the geometric distortion that common sphere-to-plane projections produce, mostly visible in objects near the poles, in addition to the lack of omnidirectional open-source labeled image datasets has made an accurate spherical image-based object detection algorithm a hard goal to achieve.

This work is a contribution to develop datasets and machine learning models particularly suited for omnidirectional images, represented in planar format through the well-known Equirectangular Projection (ERP). To this aim, DL methods are explored to improve the detection of visual objects in omnidirectional images, by considering the inherent distortions of ERP. An experimental study was, firstly, carried out to find out whether the error rate and type of detection errors were related to the characteristics of ERP images. Such study revealed that the error rate of object detection using existing DL models with ERP images, actually, depends on the object spherical location in the image.

Then, based on such findings, a new object detection framework is proposed to obtain a uniform error rate across the whole spherical image regions. The results show that the pre and post-processing stages of the implemented framework effectively contribute to reducing the performance dependency on the image region, evaluated by the above-mentioned metric.

Keywords: *Computer Vision, Deep Learning, Object Detection, Equirectangular Projection, Omnidirectional images;*

This page has been intentionally left blank.

Resumo

O conceito de *Computer Vision* é, atualmente, utilizado para resolver problemas do cotidiano em diversas áreas da sociedade que estão, cada vez mais, a impor novos desafios e dificuldades. Neste contexto, a captura de imagens omnidirecionais através de câmaras 360°, associada ao rápido desenvolvimento dos algoritmos de *Deep Learning* para detetar objetos, cria a necessidade de investigar novas formas de melhorar os métodos existentes, originalmente desenvolvidos para imagens planares 2D. No entanto, a distorção produzida pelos métodos de projetar a esfera em plano, em conjunto com a falta de *datasets* constituídos por imagens omnidirecionais, tem criado dificuldades na obtenção de um algoritmo de deteção de objetos neste tipo de imagens.

Esta dissertação é uma contribuição para desenvolver *datasets* e modelos de *Machine Learning*, especificamente desenhados para imagens omnidirecionais, representadas através da projeção Equirectangular. Desta forma, os métodos de *Deep Learning* são explorados para melhorar deteção de objetos em imagens omnidirecionais, tendo em conta a distorção causada por esta forma de projetar a esfera no plano. Em primeiro lugar, um estudo experimental foi executado de forma a identificar a taxa de erro e os tipos de erros associados às características das imagens equirectangulares. Com base nesse estudo, está identificado que a *performance* dos modelos de *Deep Learning* está dependente da localização do objeto na imagem.

Como consequência desta dependência, uma nova *framework* para detetar objetos com uma taxa de erro uniforme em todas as regiões esféricas da imagem é proposta. Esta dissertação mostra que a *framework* implementada permite que a taxa de erro seja independente da região da imagem onde está o objeto, através do seu fluxo de execução diferente das *frameworks* tradicionais.

This page has been intentionally left blank.

Contents

Originality and Copyright	iii
Dedication	iv
Acknowledgments	v
Abstract	vi
Resumo	viii
List of Figures	xiii
List of Tables	xvi
List of Abbreviations and Acronyms	xviii
1. Introduction	1
1.1. Context and Motivation	1
1.2. Contributions	2
1.3. Outline of the document	3
2. Background on Omnidirectional Vision	5
2.1. Introduction	5
2.2. 360° Image and Video Technology	6
2.3. Sphere-to-Plane Projections	8
2.3.1. Equirectangular Projection	9
2.3.2. Cubemap Projection	10
2.3.3. Segmented Sphere Projection.....	10
2.3.4. Craster Parabolic Projection.....	11
2.4. Image Datasets for Object Detection	12
2.4.1. ImageNet	13
2.4.2. COCO.....	14
2.4.3. Cityscapes.....	15
2.4.4. Omnidirectional Image Dataset.....	16
2.5. Final Remarks	20
3. Object Detection with Deep Learning	21
3.1. Before Deep Learning	21
3.1.1. Histograms of Oriented Gradient and Haar-like	22

3.1.2.	Artificial Neural Networks	23
3.1.3.	Support Vector Machines and AdaBoost	25
3.2.	Deep Learning.....	27
3.2.1.	Deep Learning Frameworks	30
3.3.	Object Detection	32
3.3.1.	Object Detection in Omnidirectional Images	35
3.3.2.	Performance Metrics.....	38
3.4.	Final Remarks.....	41
4.	Research Problem, Methodology, and Comparative Evaluation.....	43
4.1.	Research Problem.....	43
4.2.	Research Methodology	44
4.3.	Comparative Performance Evaluation.....	45
4.3.1.	Training Process	46
4.3.2.	Analysis of Results	52
4.4.	Final Remarks.....	55
5.	Framework for Enhancing Object Detection in Omnidirectional images	57
5.1.	Omnidirectional Image Dataset Training	57
5.1.1.	Training Process	57
5.1.2.	Analysis of Results	63
5.2.	Improved Framework	67
5.2.1.	Analysis of Results	69
5.3.	Framework Deployment	71
5.3.1.	Real-world scenarios	73
5.4.	Final Remarks.....	75
6.	Conclusion and Future Work.....	76
	Bibliography.....	78

This page has been intentionally left blank.

List of Figures

Figure 1 - <i>Camera Obscura</i> [8]	5
Figure 2 - 360 ° Camera (Insta360 Pro 2) [11]	6
Figure 3 - Panorama stitching example [12].....	7
Figure 4 - Virtual Reality Headsets [13].....	7
Figure 5 - Globe Representation on Plane example [15].....	9
Figure 6 - Equirectangular Representation [16]	10
Figure 7 - Cubemap Representation [16].....	10
Figure 8 - Segmented Sphere Projection [17].....	11
Figure 9 - Craster Parabolic Projection Example [6].....	12
Figure 10 - ImageNet database examples [21]	13
Figure 11 - COCO dataset examples [22].....	14
Figure 12 - Cityscapes dataset examples [24]	15
Figure 13 - Cityscapes Subset Label Distribution	16
Figure 14 - Example of a 360° video frame projected onto a 2D plane with ERP approach.	17
Figure 15 - LabelImg Dashboard.....	18
Figure 16 - Image labels file example. The left side demonstrated the PascalVOC format, while the right side shows the YOLO format.	19
Figure 17 - Resulting omnidirectional image dataset object classes distribution	19
Figure 18 - Example of a sample and corresponding label.....	22
Figure 19 - Haar algorithm. Features that allow defining a face. [35].....	23
Figure 20 - HOG detectors cue mainly on silhouette contours [36]. Human body feature representation is depicted.	23
Figure 21 - ANN architecture example [38]. Three neuron layers are depicted: input layer; hidden layer; output layer. Input layer neurons (i_1 ; i_2) are weighted connected to hidden layer neurons (h_1 ; h_2) which are weighted connected to output layer neurons (o_1 ; o_2). At the bottom, two bias neurons are illustrated.	24
Figure 22 - Backpropagation algorithm example	25
Figure 23 - Support Vector Machine [41]. An Optimal Hyperplane Separation (OHS) with a higher possible margin between both classes is proposed.	26
Figure 24 - Adaboost Algorithm [44].....	26
Figure 25 - Traditional Computer Vision (a) vs Deep Learning (b) [45]	27

Figure 26 - Typical CNN architecture in aircraft structural health monitoring [46].	28
Figure 27 - 5x5 convolution to produce a feature map [47].	29
Figure 28 - Example of Maxpool with a 2x2 kernel size [47].	29
Figure 29 - Regional proposal framework pipeline [58].	33
Figure 30 - YOLO framework pipeline [59].	34
Figure 31 - SSD framework pipeline [60].	35
Figure 32 - Equirectangular image example	36
Figure 33 - Poles Regions Identification. The top and bottom red overlay regions represent the north and south pole, respectively.	36
Figure 34 - First Research Process Pipeline [61]	37
Figure 35 - Intersection over union calculation.	38
Figure 36 - Real example of Intersection Over Union [64].	39
Figure 37 - Precision-Recall Curve example	40
Figure 38 - Privacy invasion cartoon [67]. UAVs can only be detected and identified using omnidirectional vision on the ground because they can appear from any direction.	44
Figure 39 - Research Methodology Stages	45
Figure 40 - Cityscapes subset detection examples with DL models trained on 2D images	53
Figure 41 - Omnidirectional image detection example with a DL model trained on a 2D image dataset (1)	54
Figure 42 - Omnidirectional image detection example with a DL model trained on a 2D image dataset (2)	55
Figure 43 - Omnidirectional image detection example with a DL model trained on a 2D image dataset (3)	55
Figure 44 - Ball chart reporting models' mean average precision (mAP) vs computational complexity	64
Figure 45 - Ball chart reporting models' mean average precision (mAP) vs inference time	65
Figure 46 - Omnidirectional mid-region image first division	68
Figure 47 - Proposed framework initial architecture.	68
Figure 48 - Proposed framework final architecture.	69
Figure 49 - Proposed framework predictions example (1)	71
Figure 50 - Raspberry Pi 4 Model B [78].	72
Figure 51 - Jetson Nano Developer Kit [79]	72
Figure 52 - PowerEdge R240 Dell Server [80]	73

This page has been intentionally left blank.

List of Tables

Table 1 - Confusion Matrix. Actual class column labels correspond to the reality labels while predicted class columns correspond to the predicted labels.	40
Table 2 - SSD Model Training Parameters	49
Table 3 - YOLOv3 Model Training Parameters	51
Table 4 - Performance results on Cityscapes subset with DL models trained on 2D image-based dataset.....	52
Table 5 - Performance results on omnidirectional image dataset with DL models trained on 2D image-based dataset.	53
Table 6 - Non-detected objects by image region with a DL model trained on a 2D image dataset	54
Table 7 - Standard YOLOv3 and YOLOv4 and YOLOv4 (800x448) network parameters.....	58
Table 8 - Tiny YOLOv3 and YOLOv4 network parameters	59
Table 9 - SSD 300x300 and SSD 512x512 network parameters.....	61
Table 10 - Mask R-CNN network parameters.....	63
Table 11 - Inference Time Results. DL model name presented in the first column, associated with the measured inference time, in the second column.....	66
Table 12 - Non-detected object by region in omnidirectional images	66
Table 13 - Inference time results through the traditional and proposed object detection framework.	70
Table 14 - Non-detected objects by image region through the traditional and proposed object detection framework.	70

This page has been intentionally left blank.

List of Abbreviations and Acronyms

2D	2-Dimensional
3D	3-Dimensional
AE	Auto Encoders
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
AR	Augmented Reality
ASIC	Application Specific Integrated Circuit
AWS	Amazon Web Services
BVLC	Berkeley Vision and Learning Center
CMP	CubeMap Projection
CNN	Convolutional Neural Network
CNTK	Microsoft Cognitive Toolkit
COCO	Common Objects in Context
CPP	Craster Parabolic Projection
CPU	Central Processing Unit
CV	Computer Vision
DL	Deep Learning
ECR	Elastic Container Registry
ERP	Equirectangular Projection
FLOP	Floating Operation per Second
FN	False Negative
FOV	Field of View
FP	False Positive
FPGA	Field Programmable Gate Array
FPN	Feature Pyramid Network
FPS	Frames Per Second
GPU	Graphics Processing Unit
HOG	Histograms of Oriented Gradient

IBM	International Business Machines
IDC	International Data Corporation
KPI	Key Performance Indicator
LSTM	Long-Short Term Memory
MB	MegaBytes
ML	Machine Learning
MS	MicroSoft
OHS	Optimal Hyperplane Separation
ONNX	Open Neural Network Exchange
OS	Operating System
PC	Personal Computer
R-FCN	Region-based Fully Convolutional Network
RNN	Recurrent Neural Network
SPP	Segmented Sphere Projection
SSD	Single Shot Detection
SVM	Support Vector Machine
TN	True Negative
TP	True Pegative
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicle
UHD	Ultra High-Definition
USD	United States Dollar
VOC	Visual Object Classes
VR	Virtual Reality
XML	Extensible Markup Language
YOLO	You Only Look Once

This page has been intentionally left blank.

1. Introduction

In this chapter, we focus on the dissertation's contextualization and goals along with its concerning motivation. Furthermore, contributions to the related community, as well as the document structure is presented.

1.1. Context and Motivation

Visual perception is one of the most important human senses, materialized when the eyes receive light patterns that are transformed into neural signals and then transmitted through the huge neural network that constitutes the human brain [1]. This process creates a huge amount of information used by humans to recognize and identify multiple objects, which is usually carried out through a process of learning and recognition.

Over the last decades, computer vision technology, through traditional or intelligent approaches, has been widely explored to solve real-world problems and improve life quality in many different domains, such as self-driving cars, accurate health diagnoses, agriculture operations improvement, etc [2]. The focus lies on trying to develop smart mechanisms capable of automatically interpreting visual content through image processing and, providing the same output as the human visual system would, preferably faster and more accurately. The computational process usually entails extracting relevant features from images or videos to identify patterns.

Generally, such technology aims to execute one of these three tasks: image classification, object detection, and semantic or instance segmentation [3]. Image classification refers to a process in computer vision that can classify an image according to its visual content, i.e., association with a category within some predefined set [4]. On the other hand, object detection methods aim to locate an object by returning bounding boxes or pixel masks. Finally, semantic or instance segmentation algorithms associate each image pixel with a given object label.

Such tasks are usually based on 2-dimensional (2D) images. However, new application requirements and fast technological advances are continuously posing new challenges which cannot be met by 2D cameras. Their limited field-of-view (FOV) and, subsequently, blind

spots do not allow all view directions, including all-around from the ground, mid-level above ground to sky, to be efficiently monitored.

Therefore, omnidirectional vision is increasingly a requirement on computer vision tasks: to identify people, vehicles, animal, etc., at the ground-level; monitor buildings, balconies, or windows at the mid-level; detect sky-level objects such as unmanned aerial vehicles (UAVs), which consists of autonomously or remote-controlled vehicles to fly over pre-defined areas. In addition to the above-mentioned requirement has been flooding into people daily life, according to [5], the 360° camera market for media and entertainment is estimated to reach USD (United States Dollar) 1,569.2 million by 2023 from USD 473.6 million by 2018, at a compound annual growth rate of 27.1% between 2018 and 2023.

Given the above-mentioned facts, to overcome such requirements, an efficient omnidirectional image-based objection detection algorithm is increasingly becoming mandatory. Nevertheless, with 360° images, new challenges have been created: huge resolution to keep high fidelity along 360 x 180° range; very high frame-rate to avoid motion sickness of viewers; distortions produced by the most common sphere-to-plane projections (equirectangular, cubemap projection, truncated square pyramid, and craster parabolic projection) [6].

1.2. Contributions

Computer vision approaches have been heavily studied in the last few years and, nowadays, several frameworks are capable of providing reasonable performance in many image and video processing tasks. However, currently available frameworks were usually designed to use 2D images as input, while specific solutions for omnidirectional data are still open for further improvement and performance optimization.

Experimental studies on detecting object regions in omnidirectional images, representing the spherical domain as planar images through the well-known Equirectangular projection (ERP), denoted that the error depends on the object spherical location in the image. Therefore, this dissertation aims to emphasize the main differences between omnidirectional and 2D image-based object detection algorithms' performance and propose a new framework to obtain uniform error rate across the whole spherical image regions.

The main contributions of this work include:

- State-of-the-art computer vision approaches for object detection on omnidirectional images.
- Omnidirectional and 2D image-based object detection algorithms' comparative analysis.
- Improved omnidirectional image-based object detection framework proposal.
- Evaluation of the proposed framework.

Moreover, a paper entitled “*Object Detection in Equirectangular Images*” was published at the 26th Portuguese Conference on Pattern Recognition (RECPAD) which aims to promote the collaboration between the Portuguese scientific community in the fields of Pattern Recognition, Image Analysis, and Processing, Soft Computing.

1.3. Outline of the document

The remainder of this document is structured as follows:

After this introductory chapter, Chapter 2 includes the background on omnidirectional vision. Within the scope of that chapter, omnidirectional image and 360° video technology is presented along with the description of the most common approaches for project spherical images onto 2D planes. Omnidirectional and 2D image datasets for object detection are detailed at the end of the chapter.

Then, Chapter 3 presents an overview of object detection tasks with intelligent approaches of Machine Learning (ML), namely Deep Learning (DL), which is introduced followed by a detailed explanation of the improvements and differences that DL approaches have brought when compared to the previous methods. Afterward, object detection algorithms are presented to allow to grasp the main challenges of applying such algorithms on omnidirectional images.

In Chapter 4, the research problem in addition to the methodology guidelines is presented to understand the motivation behind the need of detecting object regions in omnidirectional images. Then, the proposed procedure to develop an efficient framework is defined. Moreover, a comparative performance evaluation of 2D image-based object detection algorithms on 2D and omnidirectional image datasets is carried out.

Then, after presenting the experimental analysis, Chapter 5 focuses on providing a framework for enhancing object detection in omnidirectional images. This chapter starts by demonstrating a domain-specific approach for improving object detection results above-achieved. Then, a new framework that allows optimizing such results is proposed. Furthermore, to provide the reader with the information needed concerning the framework deployment, some considerations on the DL algorithm decision-making process are presented.

Finally, in Chapter 6, the work developed in this dissertation is summarized and the future work is presented.

2. Background on Omnidirectional Vision

In this chapter, general aspects of image capture are, firstly, introduced. Then, 360° image and video technology, as well as the most common approaches of mapping spherical points to 2D planes, are detailed to provide the reader with the main challenges that this technology brings. Succeeding the above-mentioned section, 2D image datasets for object detection and the process to produce an omnidirectional image dataset are described.

2.1. Introduction

During the nineteenth century, through a lightproof box with a pinhole on one side and a translucent screen on the other, named *camera obscura* (Figure 1), the first photographic image was taken. Later, with improvements in the above-mentioned process, the word “photography” was introduced to name the method of recording images by the action of light on sensitive material [7].



Figure 1 - *Camera Obscura* [8]

At that time, the image capture process used to take a long time to be completed, given that the acquisition of each image required a long light exposure time. However, the creation of a short focal lens enabled that time to be significantly reduced. Meanwhile, the process kept being improved until 1981, when the world’s first digital electronic camera was presented, already similar to nowadays’ cameras. Since then, this technology has been evolving with several improvements in image quality.

Despite substantial advances since the *camera obscura* conception, today’s traditional cameras rely on the same principles to produce an image: when light rays pass through the

center of the lens or effective pinhole they are projected onto a sensor array to provide a linear perspective image [9].

After a couple of years from the capture process invention, capturing all view directions in a single image faster became one of the research goals. Therefore, the next section aims to explore the technology behind cameras with omnidirectional view capabilities.

2.2. 360° Image and Video Technology

Camera manufacturers introduced omnidirectional cameras in 1958. Those cameras - depicted in Figure 2 -, commonly referred to as 360° or panoramic cameras, can capture the full 360° surroundings in a single picture or a video clip [10], putting in a single image left, right, and sky-level content. Moreover, depending on the display type, such technology may provide an immersive visual experience.

Despite its beginning as a non-consensual technology, 360° cameras have recently increased in popularity given that such cameras have become more affordable and easier to use at the consumer level. The 360° camera global market for media and entertainment is estimated to reach USD 1,569.2 million by 2023 from USD 473.6 million by 2018, at a compound annual growth rate of 27.1% between 2018 and 2023 [5].



Figure 2 - 360 ° Camera (Insta360 Pro 2) [11]

In contrast to 2D images, 360° images are typically captured through a set of multiple cameras or a camera that contains multiple lenses. Then, a software-based post-processing, namely panorama stitching, which consists of merging multiple images with overlapping regions, is required to provide a spherical image. This is demonstrated in Figure 3.

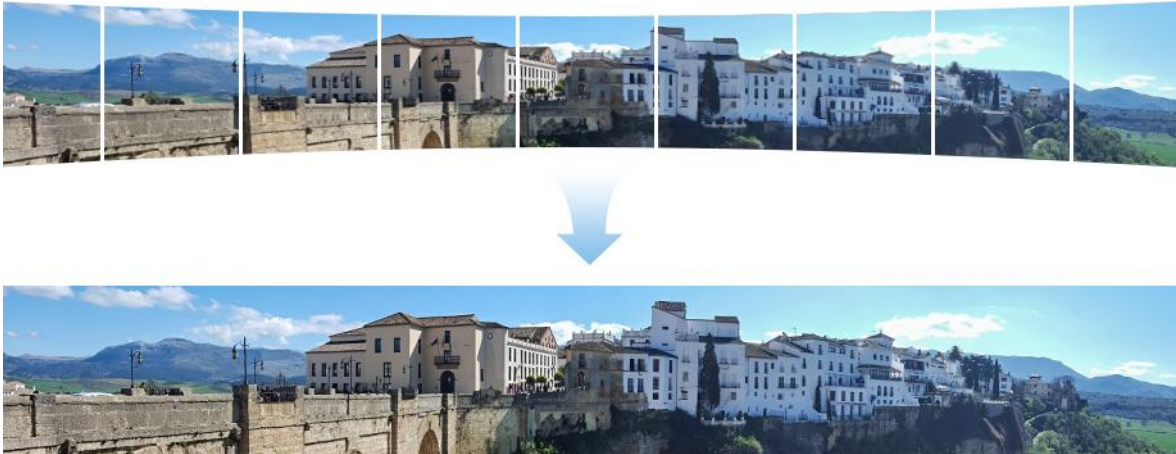


Figure 3 - Panorama stitching example [12]

Generally, the resulting image, referred to as omnidirectional or 360° image, provides an immersive experience by covering the whole 360 x 180° sphere, different from conventional images that only cover a limited plane. Therefore, any viewing angle at a given point can be recreated which leads to great opportunities to improve the visual experience and expand the functionalities of currently available applications.

The well-known Virtual Reality (VR) and Augmented Reality (AR) are among the most common applications of omnidirectional vision that attempts to simulate real-life experiences by merging different types of visual content. Thereby, its users, usually through headsets (Figure 4), can turn their heads in any direction and see programmed content, just like a human does in the real-world [13].



Figure 4 - Virtual Reality Headsets [13]

The growing development of 360° technology allows AR/VR to be optimized in the same way considering that AR/VR consists of 360° videos rendered via head-mounted displays, as the VR headsets. The ability to get immersive experiences through VR headsets has been

introduced as a mainstream technology since the boom of smartphones with high-density displays and 3D graphics capabilities which were a key-step on major advances.

Although this technology is still too expensive, multiple areas of people's daily tasks such as education where students' knowledge is acquired through virtual experiences, can have a huge positive impact. For instance, ClassVR, a product from Avantis Systems company, provides educational resources through a student-friendly interface by using a standalone VR headset, since 2017 [14].

Despite the opportunities and experiences provided by 360° technology, new challenges have also been raised. Given that omnidirectional images present a high level of detail - image resolution - and 360° videos require a high frequency of images to appear on display – frame-rate - to cover the whole sphere with high fidelity and avoid motion sickness, storage, and transmission issues are introduced because of the inherent very high file size. Then, to solve such issues, data can be transformed by mapping a sphere onto a plane, on a process called sphere-to-plane projection.

2.3. Sphere-to-Plane Projections

For many years, mathematicians and physicists needed to represent spheres on the plane. It started when mathematical principles were applied to get a globe's representation which gave rise to hundreds of map projections proposals. Figure 5 presents an example of globe representation on the plane. Despite the distortion caused by all sphere-to-plane projections, projection of an image onto a plane is required, particularly, when we are attempting to store, process, or transmit them.

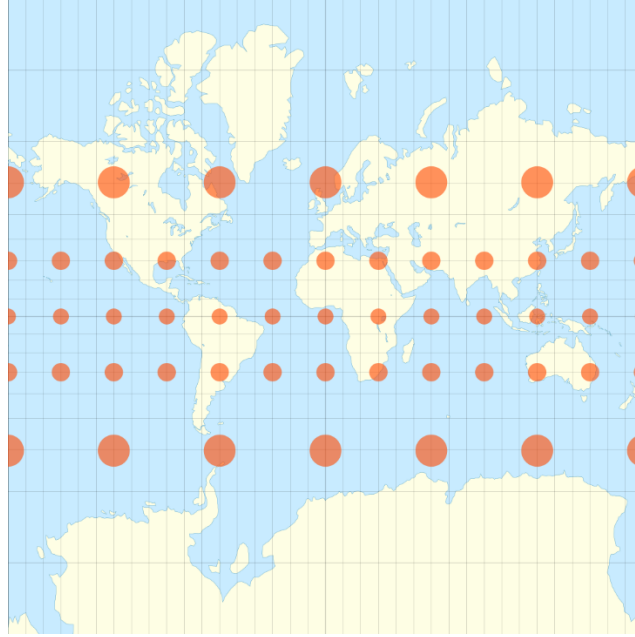


Figure 5 - Globe Representation on Plane example [15]

Then, in this section, the most common sphere-to-plane approaches are described to allow the reader to understand how omnidirectional image points are mapped to a 2D plane in computer graphics. The following projections are included: Equirectangular Projection (ERP); Cubemap Projection (CMP); Segmented Sphere Projection (SPP) and Craster Parabolic Projection (CPP).

2.3.1. Equirectangular Projection

Equirectangular projection, the most popular way to store and transmit 360° content, defines each sphere point by a horizontal angle $\theta \in [-\pi, \pi[$ and vertical angle $\phi \in [-\pi/2, \pi/2[$, as detailed in Figure 6. Then, given a sphere Σ , an equirectangular image P is obtained as follows:

$$P(i, j) = \Sigma(\theta_i, \phi_j)$$

$$\text{with } \forall_i, \theta_i - \theta_{i+1} = \delta\theta \text{ and } \forall_j, \phi_j - \phi_{j+1} = \delta\phi \text{ [16]}$$

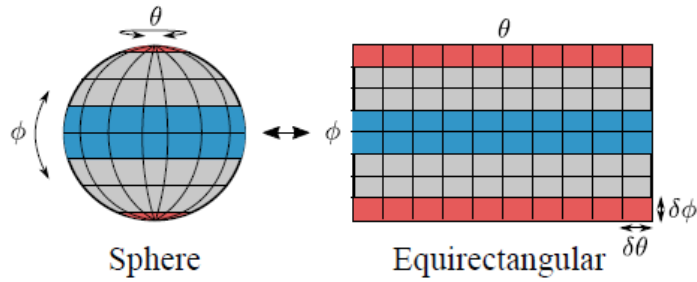


Figure 6 - Equirectangular Representation [16]

Despite this approach has become the standard sphere-to-plane projection, severe visual distortions can be caused by non-uniform sampling distance, i.e., the non-constant distance between two points. The mid-region (blue region in Figure 6) is defined with much fewer pixels than poles (red region in Figure 6), i.e., the pixel density is quite different leading to geometric distortions.

2.3.2. Cubemap Projection

Different from the ERP approach, Cubemap representation consists of decomposing the sphere into independent subregions. Generally speaking, the center of the cube is used to perform a perspective projection of the sphere on each face of the cube (Figure 7) [16]. Although this solution provides less radial distortion than ERP, it creates frontiers in the image which may lead to object split sometimes. Consequently, the same object may appear in more than one face of the cube at the same time.

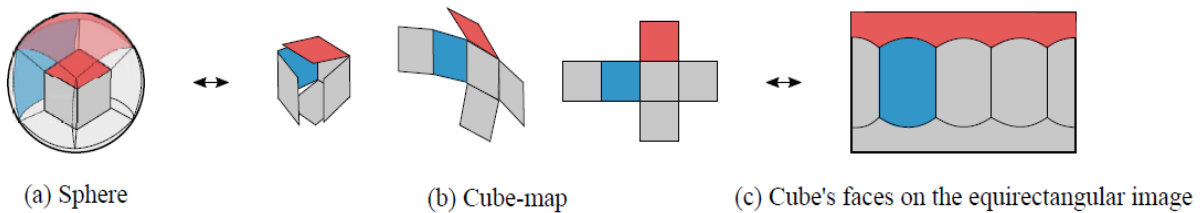


Figure 7 - Cubemap Representation [16]

2.3.3. Segmented Sphere Projection

Another method of mapping an omnidirectional image onto a 2D plane, namely Segmented Sphere Projection, aims at providing 3 different segments of the sphere: the north pole, the

equator, and the south pole. While north and south poles are mapped into 2 circles, the equatorial segment into 4 squares, as detailed in Figure 8.

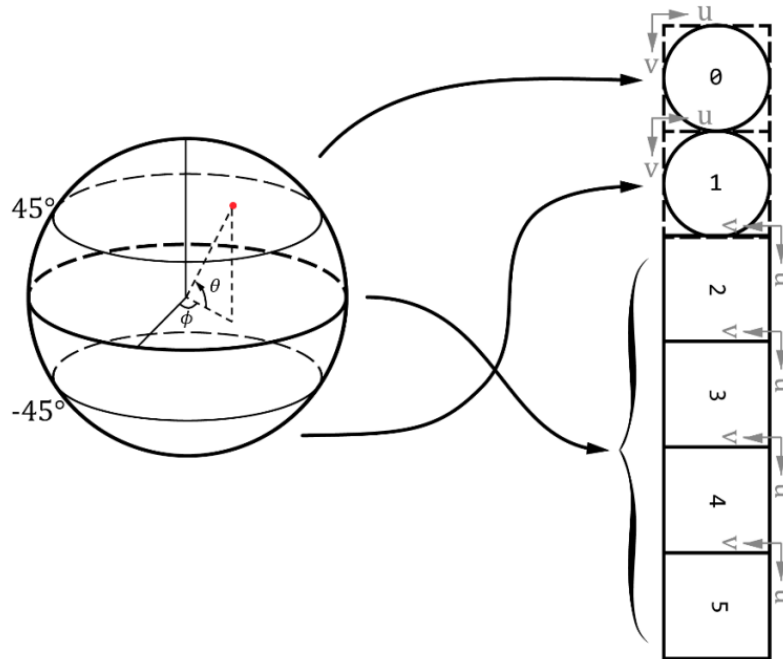


Figure 8 - Segmented Sphere Projection [17]

Notice that regions labeled as “0” and “1” are north and south poles, respectively. On the other hand, regions from “2” to “5” in Figure 8 belong to the equatorial segment, where the same projection as ERP is applied. Furthermore, to map a point (m, n) on the face to a point (ϕ, θ) on the sphere, a different mathematical equation is used, depending on the face [17].

2.3.4. Craster Parabolic Projection

Craster Parabolic Projection (CPP) implements an algorithm that, given a sphere with radius $R = 1$ and a spherical point (θ, ϕ) that needs to be mapped to a 2D point (m, n) , the following equation is applied [17]:

$$m = R\theta \left[\left(2 \cos \frac{2\phi}{3} \right) - 1 \right]$$

$$n = \pi R \sin \frac{\phi}{3}$$

Similar to other projections, CPP also produces distortion after being applied, more precisely, near outer meridians at high latitudes, where the distortion is severe, as depicted in Figure 9.



Figure 9 - Craster Parabolic Projection Example [6]

2.4. Image Datasets for Object Detection

The appearance of digital cameras and the advancement of technology along with the development of communication infrastructures have made photography fairly easy for any user. Nowadays, almost every single person at any time and anywhere can take a picture with suitable digital equipment, such as a smartphone. Therefore, large digital data volumes have become available from heterogeneous sources in a fast way and, according to International Data Corporation (IDC) report, the volume of data will reach 90 Zettabytes¹ in 2025 [18].

The resulting data started to be structured to meet requirements across a wide range of applications. One of those applications, deeper analyzed in Chapter 3, involves developing Computer Vision (CV) techniques to locate and identify objects in images. Such technology usually requires a large amount of image data, referred to as image datasets, to achieve efficient results [19]. However, the higher availability of 2D image-based capture devices when compared to devices with omnidirectional capabilities tends to make open-source omnidirectional image datasets more difficult to find out. Therefore, this section aims to provide the reader with an overview of the 2D and omnidirectional image datasets used within the scope of this dissertation, focusing on their main characteristics.

In recent years, to improve and study CV algorithms, the list of available 2D image datasets has largely increased. This list includes datasets like PASCAL Visual Object Classes (VOC) [20], ImageNet [21], Common Objects in Context (COCO) [22], Modified National Institute of Standards and Technology (MNIST) [23] and, Cityscapes [24] datasets. Although the

¹ 1 Zettabyte = 1 000 000 000 000 Gigabytes

wide diversity of available datasets, we just focus on those that were used in our experiments: ImageNet, Microsoft COCO, and Cityscapes datasets.

2.4.1. ImageNet

ImageNet [21] is an ongoing effort to provide researchers open access to an image database that is organized according to the WordNet nouns hierarchy. WordNet [25] is a large lexical database of English that labels the semantic relationships among words. As a result, the nouns hierarchy consists of a collection of nodes (entities), starting at a root node.

Furthermore, this lexical database distinguishes between types (common nouns) and instances (specific persons, countries, or geographic entities) where instances are always terminal nodes in their hierarchies [26]. For example, Lionel Richie is an instance of a singer while an armchair is a type of chair.

ImageNet includes 1000 nodes (object labels) with an average of over five hundred images per node to fill the need for structured data from the researches [21]. In Figure 10, ImageNet examples are depicted, representing the cycling noun hierarchy. The cycling node belongs to the sports and athletics root node which has three leaves (terminal) nodes: bicycling, dune cycling, and motorcycling. Besides, information about the number of pictures in this node and its distribution is also provided. In this particular case, 1.364 pictures are available, most of them labeled as motorcycling or bicycling.



Figure 10 - ImageNet database examples [21]

This database offers a huge number of images in an attempt to reach a great diversity, given that objects may have different appearances, positions, viewpoints, poses, and backgrounds. Furthermore, to correctly label each image, when a dataset is created, it needs to be reviewed by humans. In the ImageNet case, a voting system was introduced to link an image with an object label and then, an image was considered positive only if it got a convincing majority

of votes [27]. In addition to associate object labels to images, the exact location of objects is also stored in bounding box format - coordinates of the rectangular border that fully encloses an object.

Generally, it has been a useful resource to help the community in the research field of CV to overcome the lack of 2D labeled image data that those algorithms usually use as input in object detection tasks.

2.4.2. COCO

Microsoft Common Objects in Context (COCO) [22] is another large-scale dataset that contains 330.000 images, of which 220.000 are labeled. In opposition to ImageNet, COCO only provides 80 object categories. However, it has more instances per category. This dataset was also labeled with object categories (person, chair, car...) and also “stuff” categories (sky, street, grass...) given that those categories could provide relevant contextual information.

Likewise, in the ImageNet database, the COCO dataset also stores objects' location in an image, however, the COCO dataset goes deeper: it associates each pixel of an image to a category. Figure 11 depicts some examples of images from the COCO dataset to demonstrate how can each pixel be labeled with one category.



Figure 11 - COCO dataset examples [22]

As hinted by the name, its images are taken from everyday scenes thus attaching “context” to the objects captured in the scenes. Then, objects are not isolated in the image which allows CV algorithms fed by the COCO dataset to take advantage of the given data to be more accurate.

2.4.3. Cityscapes

Additionally, the investigation process behind this dissertation led us to study open-source available labeled datasets related to the urban environment. Despite the wide range of available datasets that the resulting research provided, the Cityscapes dataset [24], due to its huge diversity and application scenarios seemed to be a wise choice. The selected dataset was a researchers' effort for semantic urban scene understanding tasks by proposing a dataset that exceeds previous attempts in terms of size, annotation richness, and scene variability and complexity.

This dataset provides 25.000 annotated images captured among 50 cities for several months to get different weather conditions. Moreover, image frames included in the dataset were manually selected to guarantee a large number of dynamic objects with a high diversity of layouts and backgrounds. Dataset examples as well as their corresponding semantic annotations are depicted in Figure 12.



Figure 12 - Cityscapes dataset examples [24]

The above-mentioned dataset includes 30 classes (including road, person, car, sky, traffic sign, etc.) clustered into 8 groups (flat, human, vehicle, construction, object, nature, sky, and void). However, given the high dataset diversity that was not necessary for our research, a subset of the Cityscapes dataset was proposed, only involving images that include objects belonging to the person, car, truck, bus, and motorcycle labels.

Consequently, the resulting subset provided 3451 images, containing 31822 car, 21413 person, 888 motorcycle, 582 truck, and 483 bus instances, producing a label distribution as demonstrated in Figure 13.

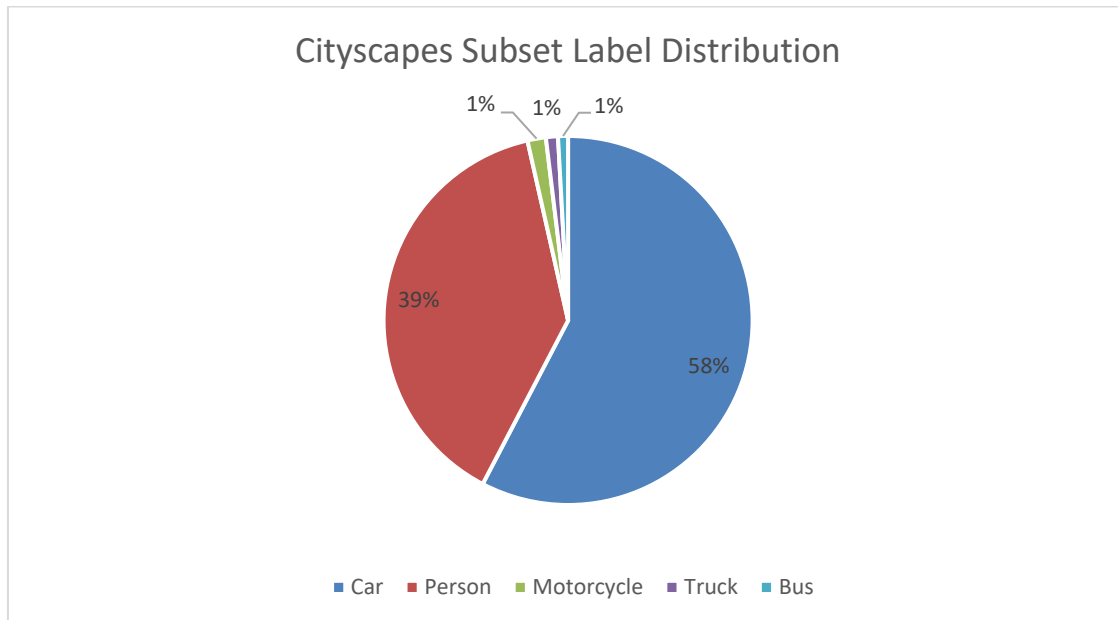


Figure 13 - Cityscapes Subset Label Distribution

In opposition to the original dataset, in this subset, a bounding box conversion process was required to keep the same format as the previous datasets. This conversion process was implemented, simultaneously, with the label filtering process, by selecting not only the minimum x and y values but also polygon maximum x and y values of each object location to produce, respectively, bounding box top left and bottom right coordinates.

Despite the acquisition of the above-mentioned 2D image datasets, the lack of open-source labeled omnidirectional image datasets was still not solved after the investigation process. That fact led us to create an omnidirectional dataset by ourselves, accepting all the work behind the procedure: video and image capturing and labeling process.

2.4.4. Omnidirectional Image Dataset

To gather content for acquiring the proposed omnidirectional image-based dataset, different approaches to collect videos through an Ultra High-Definition (UHD) 360° video camera was carried out. The 360° video camera was used to capture an urban environment to include different visual objects of all possible regions of spherical images in the dataset. For that purpose, the camera was, firstly, placed on a highly congested traffic location to produce

video recordings where people and vehicles were visible. Then, to enrich the dataset with high diversity viewpoints, object poses, and weather conditions, the same camera was mounted on the roof of a car, and videos were recorded while the car was moving. Finally, to fill the lack of aerial objects, an unmanned aerial vehicle was controlled over pre-defined regions, simulating aerial intrusion in a private property, while the 360° camera was recording, playing the role of an omnidirectional surveillance camera.

Succeeding the initial capturing process, to make the image dataset diversified, video frames had to be extracted from the resulting video content. At this stage, it was important to keep in mind that not all video frames were required to be grabbed given that differences between consecutive frames could not be relevant to achieve the desired diversity. Furthermore, the very-high-resolution videos that 360° cameras, usually, provide, needed to be taken into consideration.

Due to its ability to transform and filter multimedia content and its high-portability which allows running across Linux, Mac OS X, and Microsoft Windows², the FFmpeg [28] framework was used to overtake both concerns. On the first hand, the acquired UHD 360° videos were resized to a Full HD resolution (1980x1080 pixels). Later, one video frame was grabbed per second (which means a frame-rate of 1 FPS³) and the resulting image was projected onto a 2D plane with the ERP approach. Figure 14 depicts an image produced by the described process.



Figure 14 - Example of a 360° video frame projected onto a 2D plane with ERP approach.

² Operating Systems

³ Frame per second

After finishing processing the whole acquired video content, the image dataset had to be labeled. This process, as above-referred, consists of identifying objects in an image and save their location as well as their corresponding label class, in a human-readable format. To reduce the manual effort this task requires, LabelImg [29], an annotation tool, was used to facilitate the labeling process, providing a graphical image tool to create and save bounding boxes for each image.

Generally, this tool allows the selection of a working directory that contains the image dataset that needs to be labeled, as demonstrated in Figure 15. Then, a rectangle box could be drawn for each identified object, associating an object label class to each one and saving the final output in the most suitable format.



Figure 15 - LabelImg Dashboard

Object labels could be stored in PascalVOC or YOLO format. On the first hand, PascalVOC stores the annotation in Extensible Markup Language (XML) format and it includes information about the data source, image size, and location of the identified object as well as its classification. The object location was defined as a bounding box that includes four fundamental values: $xmin$, $ymin$, $xmax$, $ymax$. In opposition, in the YOLO format, each file line corresponds to each object instance. Each object is defined with 5 values: index of label class, x , y , $width$, and $height$, where the last four values consist of float values relative to the image width and height, and x and y values represent the center of the rectangle box. Moreover, the YOLO format does not store absolute values, using the relative format of each one. Both formats are depicted in Figure 16.

```

<annotation>
  <folder>images</folder>
  <filename>image_0001.png</filename>
  <path>D:\images/image_0001.png</path>
  <source>
    <database>360_Cam</database>
  </source>
  <size>
    <width>1920</width>
    <height>1080</height>
    <depth>3</depth>
  </size>
  <object>
    <name>vehicle</name>
    <bndbox>
      <xmin>87</xmin>
      <ymin>531</ymin>
      <xmax>169</xmax>
      <ymax>588</ymax>
    </bndbox>
  </object>
</annotation>

```

0	0.135937	0.521296	0.025000	0.025926
0	0.540625	0.518981	0.025000	0.021296
0	0.951562	0.502315	0.075000	0.050926

Figure 16 - Image labels file example. The left side demonstrated the PascalVOC format, while the right side shows the YOLO format.

At the end of the acquisition and labeling stages, the omnidirectional image dataset followed a label distribution as presented in Figure 17, containing a total of 779 images distributed by six object label classes: car, truck, bus, motorcycle, person, and unmanned aerial vehicle where the prevailing class is the car.

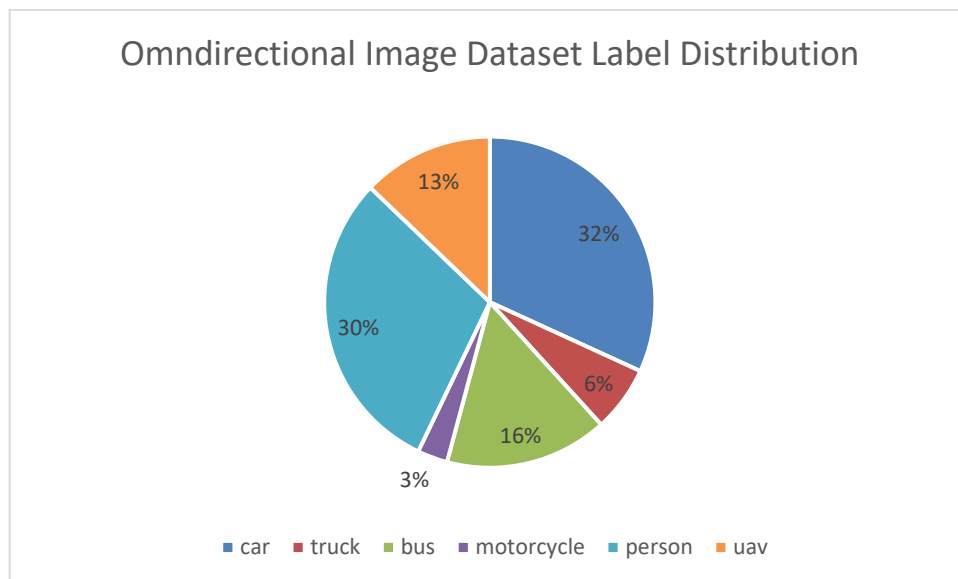


Figure 17 - Resulting omnidirectional image dataset object classes distribution

The resulting dataset allows the initial lack of omnidirectional image dataset to be overtaken and it was an effort to help the CV community to further investigate object detection techniques on this image type.

2.5. Final Remarks

Since the first camera invention, image acquisition technology has been completely transformed, however, the principles to produce an image are still the same. Although in the digital era, the vast majority of people have access to digital equipment, such as smartphones, allowing images to be captured from diverse perspectives, omnidirectional view capabilities are increasingly a requirement in the most demanding applications to overcome the blind spots that 2D image-based capture devices usually have.

In a virtual-reality perspective, the possibility of moving around at a concert or festival to find the position that users prefer comes up with a set of 360° cameras that were strategically positioned. Moreover, immersive adventures at stadiums for experiencing an event as if a user is there, are also potential applications where this technology plays a crucial role [30].

Additionally, in a non-entertainment perspective, professional 360° camera rigs allow video surveillance to be more efficient. However, given that CV approaches to automatically detect objects on such surveillance systems are usually fed by 2D images, omnidirectional image-based systems require further investigation. Such technology usually involves a large amount of data to make the system more efficient. For that reason, initial research on both 2D and omnidirectional image datasets was carried out.

Consequently, this chapter presented an overview of three of the most common 2D image dataset, namely ImageNet, Microsoft COCO, and Cityscapes. With regards to omnidirectional content, the lack of open-source labeled data made an omnidirectional image dataset to be proposed. The acquisition process required video content to be captured across diverse conditions and the resulting image data to be labeled.

As a result of the initial research has been completed, the next chapter aims to give the reader the knowledge needed about object detection approaches to understanding not only the research problem behind this dissertation but also the experiments carried out.

3. Object Detection with Deep Learning

The goal of this chapter is to present an overview of object detection approaches with Deep Learning (DL), a field that has attracted much research attention in the last years. The chapter starts with an introduction to DL, followed by an explanation of the DL concept along with a description of popular DL frameworks used by the community. Object detection with DL applied to 2D images is, then, introduced to describe the state-of-the-art methods for detecting objects in omnidirectional images. Finally, the evaluation metrics to measure and compare object detection models' performance used in this dissertation are defined and explained.

3.1. Before Deep Learning

Nowadays, a relevant application of CV deals with functions for recognizing and identifying specific objects in images, such as a person or a road, attempting to replicate the human being able to identify objects through vision, memory, and knowledge [31]. This technology aims to produce meaningful information from an image and it usually includes methods for acquiring, processing, and understanding images to achieve its goal [32].

There are many CV applications such as object detection and identification, video tracking, object pose estimation, motion estimation, and image restoration [32]. This thesis focuses on object detection algorithms through ML approaches, which have been receiving a lot of attention in recent years, not only in academia but also in industry.

Machine Learning is a subfield of Artificial Intelligence (AI) that uses computational algorithms to turn data into usable models [33] and the associated methods are, generally, classified as supervised or unsupervised. Supervised learning methods aim at developing predictive models from a labeled dataset to make predictions in an unlabelled set of samples. Each sample in the learning dataset has an associated label, which consists of the desired forecast for that sample. For example, a label representing the character "4" can be associated with the sample in Figure 18.



Figure 18 - Example of a sample and corresponding label

In opposition, unsupervised approaches try to identify patterns without access to the labeled dataset. Generally, such methods aim at grouping learning data with inherent similarities and classifying each group with its corresponding label [34].

To perform object detection tasks, traditional algorithms of ML were, firstly, developed. Identifying an object in a given image requires each object region to be located along with those identified regions to be classified. To achieve these goals, traditional methods are usually subdivided into three sub-steps: informative region selection, feature extraction, and classification [3]. The informative region selection stage consists of scanning the whole image with a multi-scale sliding window because objects differ in position, aspect ratio, or size. Usually, this task is computationally expensive due to the large number of windows needed to cover all possibilities.

Succeeding the informative region selection, the second sub-step produces a semantic and robust image representation, such as Histograms of Oriented Gradient (HOG) and Haar-like, through feature extraction algorithms. Finally, to associate a category to an object and improve the representations for visual recognition, in the classification stage, Support Vector Machine, AdaBoost, and Artificial Neural Networks are often chosen as classifiers.

3.1.1. Histograms of Oriented Gradient and Haar-like

Paul Viola and Michael Jones proposed a feature-based image representation that operates much faster than a pixel-based representation. The well-known Haar algorithm is commonly used to identify objects in images, however, it has been demonstrated to be very useful on face detection tasks.

The implemented system uses three types of features: two-rectangle feature, which value is the difference between the sum of the pixels within two rectangular regions; three-rectangle feature that computes the sum within two outside rectangles subtracted from the sum in a center rectangle; four-rectangle feature, representing the difference between diagonal pairs of rectangles [35]. Figure 19 depicts an example of features extracted from a person's face.

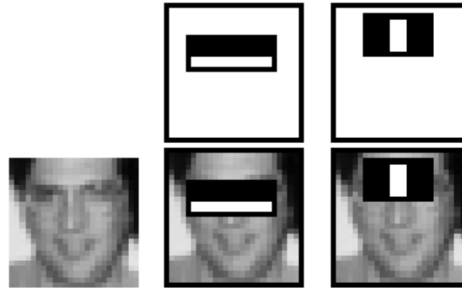


Figure 19 - Haar algorithm. Features that allow defining a face. [35]

Through a different approach, Histograms of Oriented Gradients (HOG) is also a feature-based image representation. It is calculated by computing vertical and horizontal gradients and then, gradient magnitude and angle to find the structure of the object. This feature representation outperforms human detection tasks [36], as illustrated in Figure 20.



Figure 20 - HOG detectors cue mainly on silhouette contours [36]. Human body feature representation is depicted.

3.1.2. Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired by human brain-behavior to gain abilities to solve problems. Like the human brain, ANNs are composed of a large set of weighted connections of units (neurons) where each one is responsible for performing a specific task [37]. Figure 21 depicts an example of an ANN architecture.

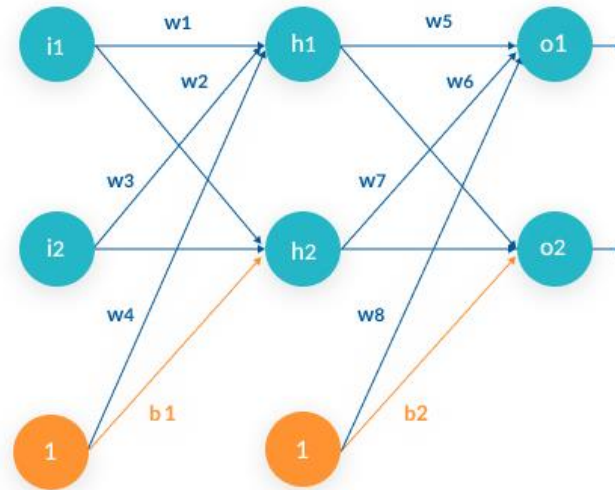


Figure 21 - ANN architecture example [38]. Three neuron layers are depicted: input layer; hidden layer; output layer. Input layer neurons (i_1 ; i_2) are weighted connected to hidden layer neurons (h_1 ; h_2) which are weighted connected to output layer neurons (o_1 ; o_2). At the bottom, two bias neurons are illustrated.

Each artificial neuron has one or more indispensable activation functions in the intermediary layers. Generally, an activation function calculates a weighted sum from the input layers, and it adds a bias. Depending on the function used, the result value can be different, however, it is usually mapped between 0 to 1 or -1 to 1.

The structure of ANN depends on its type. Nevertheless, we focus on a multilayer feedforward network, which consists of three types of layers: input, output, and hidden. The input layer receives the input in the form of a multidimensional vector and transfers it to the hidden layers. Then, each hidden layer will make decisions and weigh up the level of impact that a hypothetical change has on the final output based on the previous layer. In the end, after data had been processed, output layers receive the final output [38].

In object detection problems, training multilayer feedforward networks with a backpropagation algorithm is a common approach. The backpropagation algorithm, illustrated in Figure 22, aims at optimizing the weights so that the neural network can correctly associate inputs to outputs. This process consists of two stages: the forward pass and the backward pass. The forward pass keeps the values of the weights unchanged and output layers values are computed from the input data. On the other hand, in the backward pass, the computed error is propagated back to the previous layers. Both stages together make one iteration.

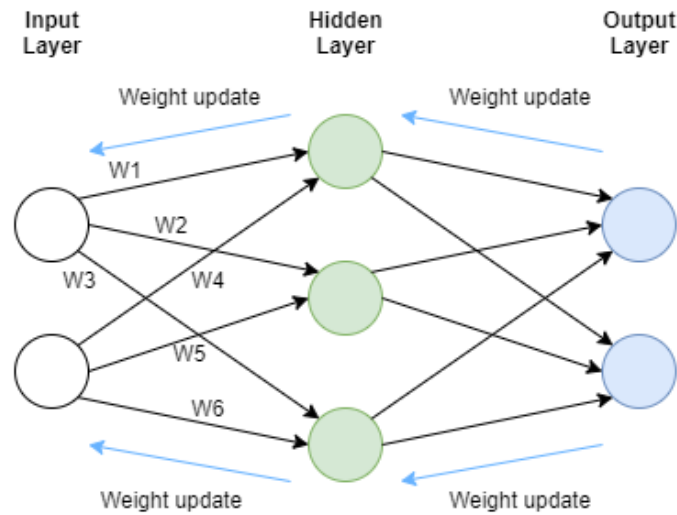


Figure 22 - Backpropagation algorithm example

Therefore, in the backpropagation process, an error is computed with the backpropagated data by each artificial neural network neuron and weights values are, then, updated. The process is repeated until reaching the input layer.

Even though ANN can be trained through supervised or unsupervised learning approaches, this research work focuses on the supervised method. This approach requires an output vector to be known for each of the input vectors. Then, to compute the error value, the difference between ANN output and known output is calculated. Therefore, the learning algorithm on this method is responsible for learning a mapping function that allows us to receive an input and provide the correct output.

Object detection algorithms that this dissertation implements are different from the above-mentioned ANN by their depth, as their name implies: DL networks. In Section 3.2, a better understanding of the DL concept is provided.

3.1.3. Support Vector Machines and AdaBoost

Support Vector Machines (SVMs), introduced by Vapnik [39], are supervised Machine Learning algorithms very often used in classification problems. Generally, SVM aim at finding an optimal hyperplane to separate a dataset into two classes in an n -dimensional space, as presented in Figure 23. Mathematically, the problem can be described as follows. Given a set of training data $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ where x_i belongs to R^d , a d -dimensional space, and the data is labeled according to an unknown probability distribution

$P(x, y)$ and a loss function $V(y, f(x, y))$ is defined to measure the error, when, for a given x , $f(x)$ is calculated instead of the actual value y [40].

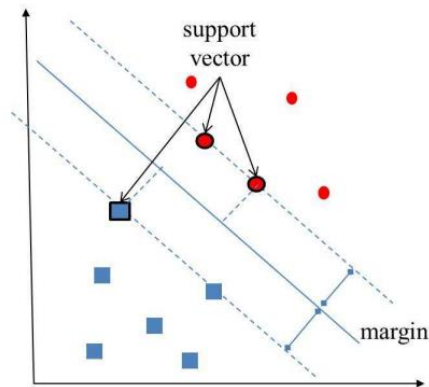


Figure 23 - Support Vector Machine [41]. An Optimal Hyperplane Separation (OHS) with a higher possible margin between both classes is proposed.

This algorithm has provided new solutions to important real-world problems and specific application scenarios. For instance, in 1999, it was proposed to apply a set of SVM classifiers to medical Tuberculosis from photomicrographs of Sputum smears, the first time it was used in medical problems, which has allowed medical experts to be supported on their very important decisions [42].

On the other hand, another approach for solving a classification problem was proposed, namely, Adaptive Boosting (AdaBoost) which allows the creation of a strong and robust classifier from a set of weak classifiers through an iterative learning algorithm [43], as denoted in Figure 24.

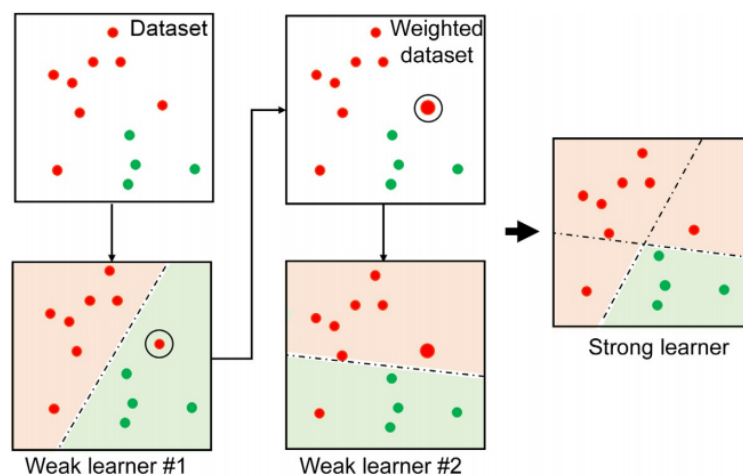


Figure 24 - Adaboost Algorithm [44]

In Adaboost's process, the weight of a sample misclassified by the previous decision tree is boosted so that the previously misclassified sample is correctly classified by the subsequent tree. As long as more weak classifiers are added in series to the model, classification accuracy increases [44].

3.2. Deep Learning

Deep Learning is part of a wide range of ML methods, which aims at replicating the human of learning from multiple levels of information using layer-based approaches. Before the DL emergence, feature engineering was required to extract descriptive information from images through CV techniques, as demonstrated in Figure 25. Moreover, it was necessary to select which information was relevant to train ML algorithms.

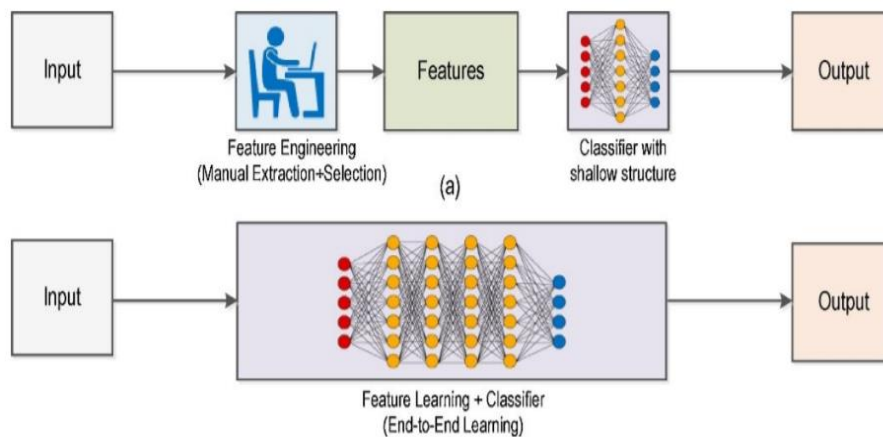


Figure 25 - Traditional Computer Vision (a) vs Deep Learning (b) [45]

In opposition to traditional techniques, DL methods, based on ANNs, have introduced the end-to-end learning concept. Essentially, the learning algorithm architecture allows automating the hand-crafted feature extraction and selection. Consequently, a unified learning framework is provided instead of a multiple-step approach.

Deep Learning approaches have become more popular due to major advances in network structures and training approaches. In 1997, in [23], the authors proposed to replace the hand-crafted feature extraction step by operating directly on pixel images. This approach applied to a character recognition research problem proves that was possible to develop accurate machine learning algorithms without a manual extraction and selection and it brings huge research efforts on this topic.

Over the last decades, different Deep Neural Networks types have been developed: Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), Auto Encoders (AE), etc. However, we focus on Convolutional Neural Networks (CNNs). Convolutional Neural Network is a multiple layer architecture inspired by the natural visual perception mechanism of living creatures that usually consists of a set of three types of layers: convolutional, pooling, and fully connected layer [46]. Different from the other neural networks, neurons in CNN are arranged in width, height, and depth dimensions, as depicted in Figure 26.

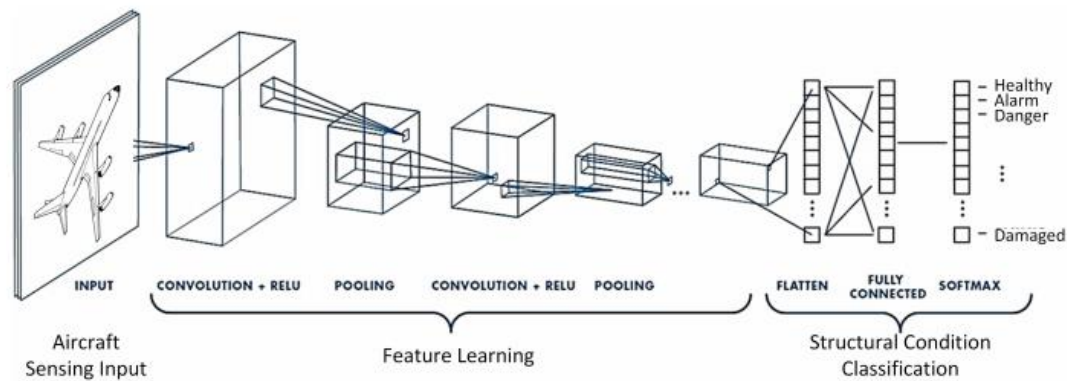


Figure 26 - Typical CNN architecture in aircraft structural health monitoring [46].

Convolutional layers, as the name implies, are a crucial part of CNN operation mode. Their main goal is extracting features from the given input through multiple convolutional kernels that learn feature representations. The convolutional kernel or filter consists of an array of numbers that represent a certain pattern in its area. This filter slides around the input image, multiplying, and adding the values in the filter with the image pixels. This sliding process is also known as the convolving process and it starts at the top left corner. The final feature map results from applying an activation function on the convolved results. The most used activation functions are *sigmoid*, *ReLU*, and *tanh*. Therefore, the above-depicted figure demonstrates a set of convolutional layers with *ReLU* activation function.

The above-mentioned process is demonstrated in Figure 27, where a 2D pixelated image is received as input and the convolutional process is applied, with a kernel size of 5x5 to produce a feature map.

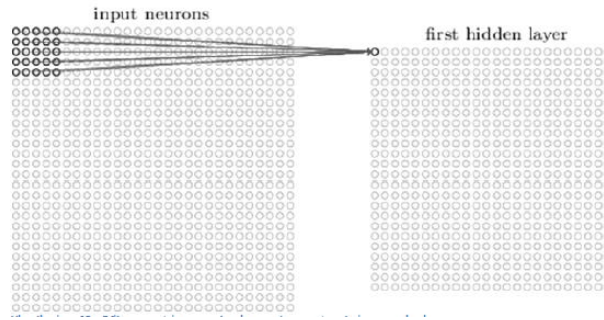


Figure 27 - 5x5 convolution to produce a feature map [47]

A convolutional layer is usually followed by a pooling layer which aims at reducing the number of elements of feature maps computed from the previous layer. As well as convolutional layers, such layers also require a kernel size to be defined. Max or mean pooling are some examples of commonly used filters. In Figure 28, a max-pooling with a 2x2 kernel size is demonstrated, where the maximum number in every subregion where the filter convolves is chosen.

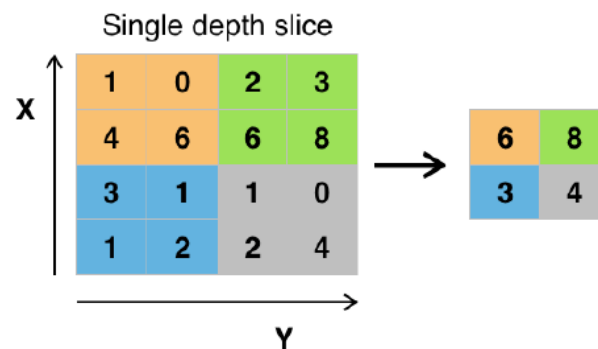


Figure 28 - Example of Maxpool with a 2x2 kernel size [47]

Finally, fully connected layers produce an n-dimensional vector output from a given input, where N represents the number of possible classes. For example, if we were developing a CNN for recognizing Latin alphabet characters, the N value would be 26 (Latin alphabet length). This layer usually uses an output function to normalize the output vector of CNN. In the architecture represented in Figure 26, the *softmax* output function is used.

To train a DL model, a labeled dataset is required. Both ImageNet and MS COCO datasets are commonly used for training models, given that they offer a huge amount of image labeled data. When a model has never been trained before, a random initialization of weights is performed, which is often known as a train from scratch.

Given that training a model from scratch is computationally expensive, the transfer learning approach is mostly used. This process consists of taking a pre-trained model on a large dataset and fine-tuning the model with our dataset. The main idea is taking advantage of the feature extracting ability of the pre-trained model and replacing the classifier. Considering the Latin alphabet example, rather than training the whole network, we can use a pre-trained model on ImageNet (which has 1000 classes) and train it on an alphabet character images dataset (with only 26 classes).

Object detection models based on CNN architecture enable CV engineers to achieve better accuracy in complex tasks when comparing to traditional approaches. By eliminating the manual feature extracting step, the less expert analysis given that it is not necessary to choose which features are important in each image. Additionally, the emergence of DL approaches was also promoted by the DL frameworks.

3.2.1. Deep Learning Frameworks

Due to the growth of the DL community, a lot of open-source frameworks have been introduced to facilitate the execution of the most common DL algorithms. Each one aims at trying to optimize ML algorithms' performance through different implementations. This section focuses thereby on benchmarking the most popular DL frameworks by identifying the main advantages and disadvantages of each one and explaining their implementation.

Recent advances in hardware technology have enabled research across different implementations to explore deep learning algorithms' performance over different hardware environments. Despite the Central Processing Unit (CPU) is the mainstream technology, the Graphics Processing Unit (GPU) outperforms on neural network training given that their internal cache, high-speed bandwidth, and quick parallel performances. Furthermore, like GPU, both Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) devices can accelerate model training due to their parallel computing capacity [48].

Depending on the goal and domain of the task, each hardware device can take advantage. On the first hand, ASIC has an optimized architecture to achieve low-energy consumption, low latency, computing performance, and scalability. On the other hand, FPGA takes advantage given its chip price. Finally, GPU is also able to achieve energy efficiency and outperforms on compatibility, upgradability, and ubiquitous computing.

Mainstream DL frameworks have different compliant hardware platforms and base implementations. Thus, their main characteristics and differences are below-described to understand each implementation [49].

Different DL frameworks have, obviously, different compliant hardware platforms and base implementation. To better understanding the main ideas behind each framework, five toolkits are bellow described. Generally, we include the most popular frameworks amongst the researchers, according to [50]: TensorFlow, Microsoft Cognitive Toolkit, PyTorch, Caffe, and Keras.

- **TensorFlow** [51], developed by Google Brain, provides a flexible architecture through a single data flow graph that expresses all numerical computations, including mathematical and communicational operations. Furthermore, it supports distributed training given that computation can be deployed to one or more CPUs or GPUs on different hardware.

Although is written in Python programming language, math operations are written as high-performance C++ binaries. Python is easy to learn and works with and it provides high-level programming abstractions, which justifies TensorFlow implementation.

- **Microsoft Cognitive Toolkit (CNTK)** [52] is an open-source toolkit, which aims at providing tools for training and testing neural networks through multiple GPUs. Due to its effective way to manage memory resources, this framework is computationally very efficient.

Microsoft-CNTK was one of the first DL frameworks to support an open-source model representation for framework interoperability and shared optimization, known as Open Neural Network Exchange (ONNX).

- **PyTorch** [53] is a Python-based ML framework based on a prior framework know as Torch [54]. In opposition to frameworks like TensorFlow, which requires a computational graph to be designed before running the model, in PyTorch, the graph can be dynamic.

This ML framework provides users with CPU and GPU neural network training options. Moreover, it offers great flexibility and speed due to its optimal implementation.

- **Caffe** [55] provides researchers and engineers with a DL framework to train and deploy DL algorithms which have been maintained and developed by the Berkeley Vision and Learning Center (BVLC) and its GitHub⁴ community.

Its main advantages include its modularity and speed given that the Caffe framework it allows to be extended to new data formats and network layers. However, it is not efficient in a wide range of domains. The main application area is computer vision or image classification problems.

- **Keras** [56] is an open-source DL library, written in python that works on top of other DL frameworks such as TensorFlow or Microsoft-CNTK. Generally, it facilitates the process of prototyping neural networks and it supports a wide range of network layers.

Keras Application Programming Interface (API) development reduces cognitive load given that it is user-friendly and easy to extend. Moreover, new modules are simple to add.

3.3. Object Detection

In recent years, precisely determining the location of objects contained in each image by outputting the bounding box around the object has been attracting much attention. This task, referred to as object detection, provides a lot of opportunities in real-world scenarios. However, it also faces hard challenges such as partial occlusion; different illumination conditions, poses, and scale [57]. This subsection aims at clarifying DL-based object detection approaches to understand how this task is performed.

Deep Learning algorithms for object detection can be mainly subdivided into two types of approaches. In the first type, the algorithm has two-steps, including a regional proposal

⁴ Distributed version control and source code management tool

generation and a classification stage. On the other hand, the second approach consists of just one step. The first pipeline, as shown in Figure 29, consists of three modules. From the input image, around 2000 regions are extracted and then proposed, producing a set of candidate regions. The second module takes as input each region and computes features through a CNN that is later associated with a class on the third module where a set of SVMs are available [58].

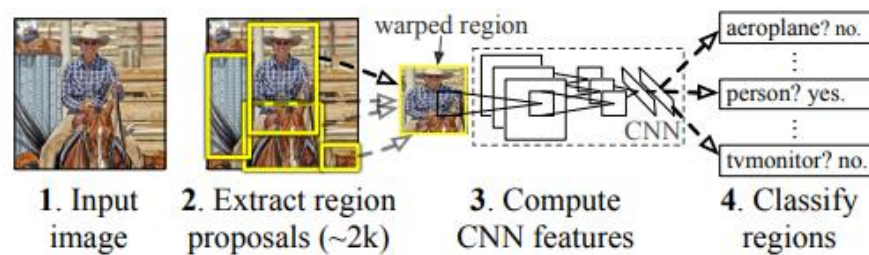


Figure 29 - Regional proposal framework pipeline [58]

This architecture was the basis for Regions with CNN features (R-CNN), in 2014, which brought accurate results when compared to the previous methods. Later, this algorithm was subsequently upgraded with the introduction of SPP-net that becomes more efficient on object detection tasks due to its improvements in locating objects with different scales. Despite the implemented changes in the initial architecture, new solutions were continuously developed until reach most recent solutions: Region-based Fully Convolutional Network (R-FCN), Feature Pyramid Network (FPN), and Mask R-CNN [3].

Therefore, R-FCN, by adding a new network layer, has improved its inference time due to the generation of scores for each proposal region. On the other hand, FPN architecture has been widely used to achieve better results on scale invariance scenarios in object detection systems. Finally, Mask R-CNN provides a two-step architecture to detect all objects in an image and perform an instance segmentation.

In contrast to this approach, the second object detection pipeline, based on global regression/classification, does not split the process into two stages, allowing the direct extraction of bounding boxes with associated classes from input images and, consequently, reduce time expenses. Mainstream object detection frameworks based on this pipeline include You Only Look Once (YOLO) and Single Shot Detection (SSD).

In the YOLO framework pipeline, illustrated in Figure 30, the process starts by dividing the input image into a grid of $S \times S$ dimension. Then, the grid cell that contains the center of an

object takes the responsibility of detecting it. Moreover, each grid cell produces bounding boxes and confidence scores or, in other words, how confident that the model is that the box contains an object. Besides, at the same time, a class probability map is computed where each grid cell predicts the class associated with an object [59].

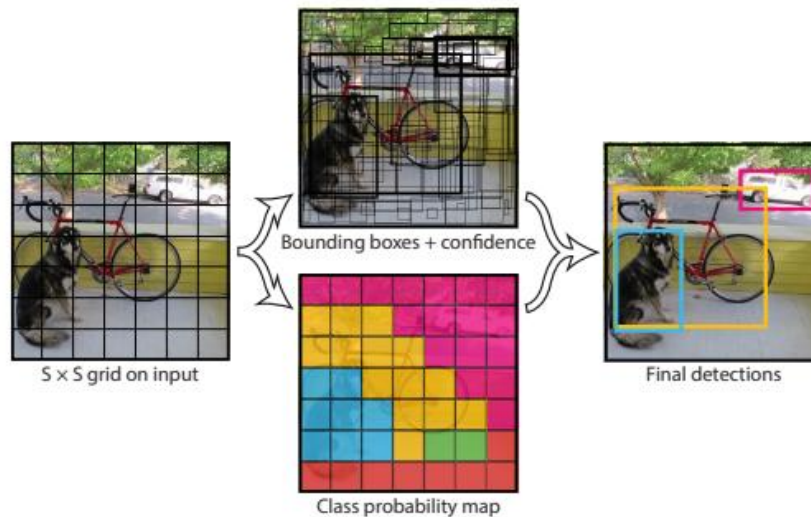


Figure 30 - YOLO framework pipeline [59]

Considering, now, the YOLO model architecture, its first version has 24 convolutional layers for feature extraction, followed by 2 fully connected layers for computing predictions, however, the last improved version of YOLO, YOLOv3, consists of 53 convolution layers. This structure provides great results on image processing in real-time scenarios: 45 Frames Per Second (FPS) with 320x320x3 input size YOLOv3 version and 220 FPS with a simplified version.

On the other hand, the SSD framework is also based on a feed-forward convolution network that aims to detect objects in images, producing bounding boxes and confidence scores, as the YOLO framework does. Despite their common goal, the SSD framework pipeline consists of receiving an input image with ground truth boxes which are computed by a set of convolutional layers, where the above-mentioned feature extraction process is performed. After this step is completed, the feature map obtained is computed and bounding boxes of different sizes and aspect ratios are returned. Finally, confidence scores (“*conf*” in the Figure) are calculated for each bounding box (“*loc*” in the Figure), as depicted in Figure 31 [60].

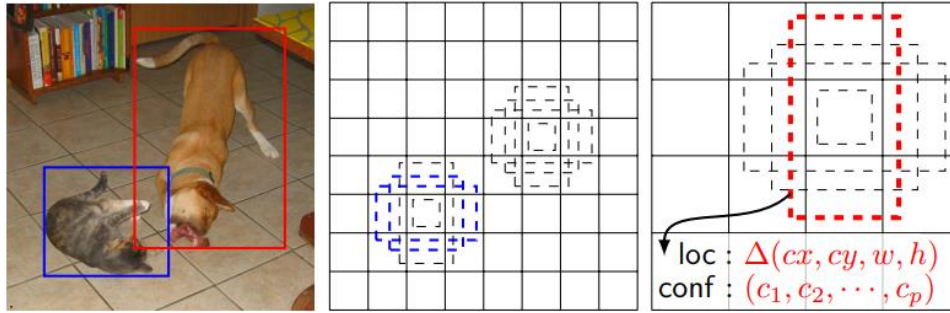


Figure 31 - SSD framework pipeline [60]

Different from YOLO, the SSD model architecture consists of a truncated base network followed by a set of convolutional layers whose size is decreased progressively, providing predictions at multiple scales. Then, feature maps computed are going through a 3x3 convolution to produce bounding boxes.

As a result of the diversity of DL model architectures, different open-source frameworks have been developed to achieve good performance on object detection. However, the fact that such frameworks are usually fed by 2D images, led recent research works to evaluate object algorithms on omnidirectional images.

3.3.1. Object Detection in Omnidirectional Images

Object detection in 2D images with DL approaches was a great achievement that technology advances have promoted to solve prior challenges. However, due to the emergence of 360° technology, new challenges have been created. The first challenge results from the need of projecting a 360° image onto a 2D plane which can be achieved through different methods, as mentioned in Chapter 2.

Given its simple approach to convert a spherical plane into a cartesian grid, the ERP has been established as the mainstream sphere-to-plane conversion method for project 360° content. Therefore, this dissertation relies on the ERP approach rather than on the cubemap, SPP, or CPP projections. Consequently, this section provides an overview of state-of-art algorithms and frameworks for detecting objects in ERP images.

Unfortunately, ERP applied to images captured from 360° capture devices create severe distortion that is mostly visible in objects near the poles [61]. In Figure 32, an ERP image example is depicted, followed by the poles region identification, in Figure 33. The visible

distortion can hamper the use of 2D image-based object detection models in omnidirectional images.



Figure 32 - Equirectangular image example



Figure 33 - Poles Regions Identification. The top and bottom red overlay regions represent the north and south pole, respectively.

In addition to the visual changes that ERP produces, another issue has been identified by state-of-the-art researches. The difficulty to overtake the lack of omnidirectional open-source labeled image datasets, as already denoted in our initial research experiments, makes an accurate omnidirectional image-based object detection algorithm a hard goal to achieve.

These problems led the DL community to propose different approaches to keep a good object detection performance even facing ERP image challenges. “Object Detection in Equirectangular Panoramas” [61] and “Pano-RSOD Dataset” [62] were two of the first research works that have been pursued focusing on this field.

The first work [61] proposes a multi-projection variant of the YOLO detector that tries to solve identified problems through multiple stereographic sub-projections. In their experiments, a data set extracted from 22 4k-resolution videos with 6431 objects was

considered. The solution, as presented in Figure 34, consists of three stages: stereographic projection, detection, and bounding box post-processing.

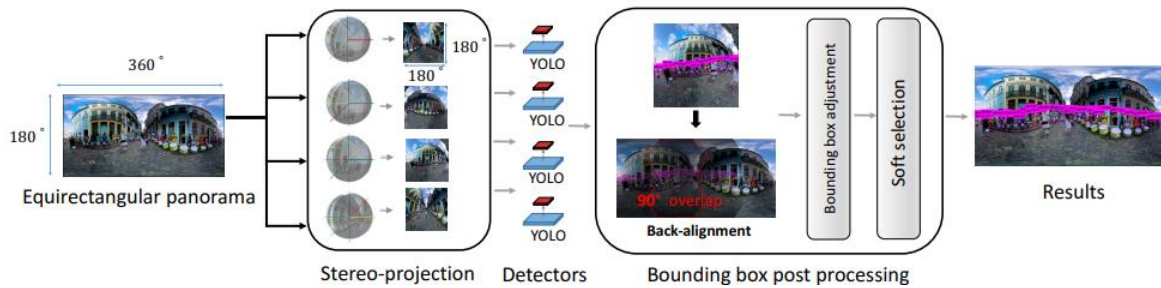


Figure 34 - First Research Process Pipeline [61]

First, to cover the whole image two projections with a horizontal and vertical span of 180 degrees are required. Nonetheless, in this implementation, object distortion is still large, and, for that reason, four sub-projections with an overlap of 90 degrees were defined for processing. After the stereo projection stage, each sub-projection is processed by the YOLO detector, producing a set of detections with bounding boxes, confidences, and class names associated. Finally, bounding boxes returned by the last step are re-aligned and results are presented.

The second work [62] starts by creating a non-open-source dataset of 9402 images with 2048 x 1024 pixels extracted from the streetscape of downtown Zhongshan City, Guangdong Province, China, and each image has 9 objects on average [63]. Then, through LabelImg open-source tool [29], all the images have been labeled and reviewed and, finally, the dataset consisted of 4 categories with a total of 87542 bounding boxes.

The authors evaluated different object detection algorithms based on both one-stage and two-stage frameworks. Generally, YOLOv3 outperforms the other methods. In terms of speed, it performs better, reaching 13 milliseconds of processing time per detection. Furthermore, considering the accuracy metric, it achieved top performance, 1% above than the second-best method, Faster R-CNN. The accuracy of car and person classes also obtained higher values when using the YOLOv3 algorithm, however, for sign and line categories, the highest values were reached with Faster R-CNN.

Among the presented research works, the need for evaluating and comparing algorithms' performance was pointed out. In the same way, this dissertation requires an overview of the performance metrics used to provide the researchers with useful information at the decision-

making stage. Therefore, the next section aims to present a detailed explanation of such performance metrics' computation.

3.3.2. Performance Metrics

Given the need for comparing object detection algorithms' performance, a set of common evaluation metrics have been introduced. Therefore, the evaluation performance metrics section provides a useful explanation of mean average precision (mAP), intersection over union (IoU), and floating operation per second (FLOPs). These metrics along with a deep analysis of provided results, are fundamental to choose the solution that better fits the target goal.

First, one of the most common metrics used to analyze the accuracy of predictions produced by trained deep neural networks is the IoU. Generally, it computes the similarity between the bounding box predicted by the model and the ground truth bounding box (desired model output). This performance metric, as depicted in Figure 35, is calculated by dividing the overlapping area and the area of union.


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


Figure 35 - Intersection over union calculation

Taking into consideration a real example where the target is detecting traffic signs, the trained model has predicted the red bounding box demonstrated in Figure 36, while the desired output was the green bounding box. By applying the above-mentioned formula, the IoU is computed to evaluate the model's performance. Its values are within the range between 0 and 1 (or 0 and 100 if we are looking at percentage values) and the more accurate prediction, the higher is IoU value.

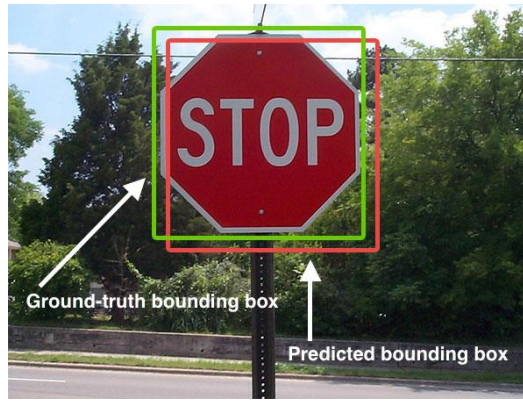


Figure 36 - Real example of Intersection Over Union [64]

In addition to IoU, mean average precision (mAP) has been a typical performance metric to evaluate the results produced by DL models. Before explaining the mAP, recall, and precision metrics and, subsequently, their inherent error types used should be clarified. Firstly, when we are facing a binary classification problem, there are four possible prediction outcomes: true positive (TP), false positive (FP), false negative (FN), and true negative (TN).

Assuming that our trained model identifies the presence of a person in an image. Then, we have a positive class, person, and a negative class, no person. Therefore, a true positive occurs when the model correctly predicts the positive class. In the same way, a true negative is an outcome where the model correctly predicts the negative class. On the other hand, false-positive is the given classification when the model incorrectly predicts a positive class, while false negative is the outcome when the negative class is incorrectly predicted. Table 1 summarizes these concepts applied to the given scenario.

		Actual Class	
		P	N
Predicted Class	P	<p>True Positive (TP)</p> <p>Reality: Person Predicted: Person</p>	<p>False Positive (FP)</p> <p>Reality: No Person Predicted: Person</p>
	N	<p>False Negative (FN)</p> <p>Reality: Person Predicted: No Person</p>	<p>True Negative (TN)</p> <p>Reality: No Person Predicted: No Person</p>

Table 1 - Confusion Matrix. Actual class column labels correspond to the reality labels while predicted class columns correspond to the predicted labels.

From these values, it is important to analyze not only the proportion of correct positive predictions (precision) but also the proportion of actual positives that were correctly identified (recall). The above-mentioned metrics, namely, precision and recall are computed as follows:

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

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

After understanding the above concepts, the standard performance measure for object detection, mAP, can be introduced. Although a consensual definition of the way it should be computed is not established yet among researchers, this dissertation follows the most common definition: the average of areas under the recall-precision curve (Average Precision - AP) for all the classes. In Figure 37, a precision-recall curve example is depicted.

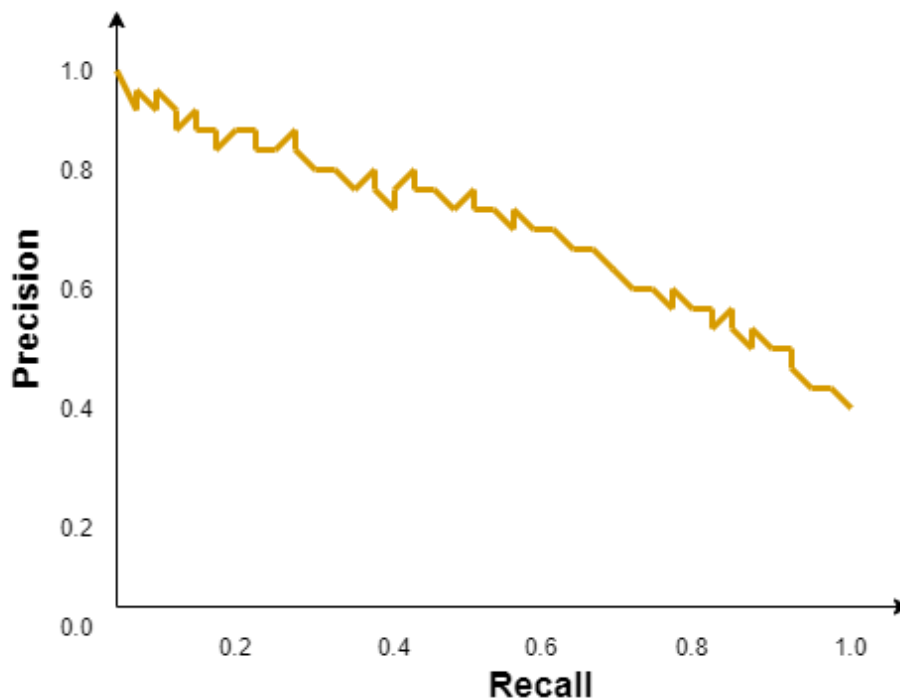


Figure 37 - Precision-Recall Curve example

As above-demonstrated, average precision (AP), which corresponds to just one label class, can be computed by finding the area below the precision-recall curve (orange line in Figure 37). Finally, to compute the desired mAP, the average AP for all classes must be calculated. These metrics are usually associated with IoU. The IoU metric defines the minimum threshold to consider a correct prediction. For instance, “*AP@0.5*” means that the values of AP were measured considering that correct predictions have, at least, 0.5 of IoU.

Lastly, to evaluate models’ performance, the floating-point operations per second (FLOPs) metric was also taken into consideration in this research. According to International Business Machines Corporation (IBM) [65], FLOPs value is a critical measure of computing power and speed. Consequently, hardware resources needed to perform predictions are usually estimated by analyzing this useful measure.

3.4. Final Remarks

With the rapid development of technology, different approaches and tools have been recently introduced. Computer Vision approaches have been explored to provide algorithms that allow us to identify patterns through image feature extraction, creating new application scenarios, such as object detection.

Object detection has been performed through traditional and DL approaches: traditional methods are subdivided into informative region selection, feature extraction, and classification steps, otherwise DL methods do not need a manual feature extracting process from raw input data. Due to this great advantage, DL has been mostly explored over traditional methods and, consequently, new DL methods have been introduced.

Object detection with DL techniques that are based on 2D images has shown a reasonable performance. However, over the last few years, due to the 360° cameras market growth and the new requirements that the technology brings, an efficient object detection method is required. Current research works on this field identified the lack of omnidirectional image datasets and the distortion that ERP produces as the main challenges that current algorithms could face.

For that purpose, we aim to evaluate the performance of 2D image-based algorithms on the omnidirectional image dataset acquired in the next chapter. Before that section, the research problem along with the methodology is described in detail.

4. Research Problem, Methodology, and Comparative Evaluation

Chapter 4 focuses on providing a clear description of the proposed research problem and research methodology. Firstly, an introduction to the problem is carried out to give the reader the knowledge needed for understanding its motivation. Succeeding that section, the research problem is presented, followed by an explicit methodology definition, including well-defined steps.

Finally, a comparative evaluation of 2D image-based object detection algorithms on 2D and omnidirectional image datasets used as the baseline for our research is detailed. Then, the training process with the inherent steps, as well as the analysis of results is presented.

4.1. Research Problem

Omnidirectional vision on object detection frameworks to capture the full field of view (FOV) of 360° is increasingly becoming a requirement on the most demanding systems. Technology advances in many areas made 2D cameras not enough to ensure the efficiency of a surveillance system. Their limited FOV and, subsequent blind spots, have to be covered to allow simultaneous surveillance in all view directions, including all-around from the ground, mid-level above ground to sky.

Therefore, 2D video cameras no longer comply with the concept of an environment where everything should be possible to be observed, scrutinized, and identified (obviously subject to the legal conditions in force). For instance, the limited FOV and relatively low resolutions of most current systems, are constraining factors for such types of requirements, which may not allow achieving the target performance levels specified for systems with intelligent functionalities.

Different view directions mean, obviously, different challenges. At the ground level, threats such as people, vehicles, animals, or door fronts require special attention. On the other hand, surveillance at the mid-level above ground allows buildings, windows, or balconies to be efficiently monitored. Finally, at the sky-level, it comes one of the most recent threats of privacy invasion, for instance: unmanned aerial vehicles (UAVs), depicted in Figure 38. These vehicles, which can be autonomously or remote-controlled, can fly over a target area which may lead to invasion of private properties, for example [66].



Figure 38 - Privacy invasion cartoon [67]. UAVs can only be detected and identified using omnidirectional vision on the ground because they can appear from any direction.

Although different view directions can be covered by multiple 2D capture devices, in the long term, investing in an omnidirectional camera is usually a better solution to reduce hardware costs. Additionally, processing images from different sources requires object detection frameworks to be rearranged to aggregate the results from the execution of object detection tasks in each image.

Given the fact that most object detection systems do not use omnidirectional, such format on its own poses implicit challenges to current DL algorithms. The main concerns are caused, not only by the huge resolution of each image but also by the inherent geometric distortions that may occur as a result of the planar projection used in their representation.

Therefore, this work is a contribution to the DL community by exploring the well-known object detection algorithms applied to omnidirectional images. Then, the next section aims to describe the research methodology that was followed within the scope of this dissertation.

4.2. Research Methodology

The initial investigation on 2D image datasets and the development of an omnidirectional dataset, stated in Section 2.4, allowed us to define the research methodology carried out in this dissertation, following the workflow demonstrated in Figure 39.

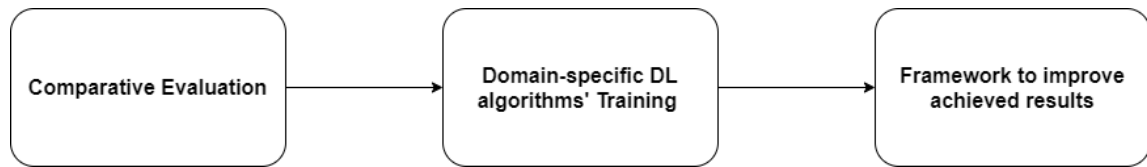


Figure 39 - Research Methodology Stages

As depicted in the above Figure, this research consisted of three fundamental stages comprised of multiple sub-steps. On the first hand, a comparative performance evaluation of object detection algorithms trained in 2D image datasets on 2D and omnidirectional datasets had to be performed to allow identifying the main drawbacks and differences between the execution of object detection tasks on both image types. For that purpose, the omnidirectional image dataset and the Cityscapes subset were defined as the source data to develop the comparative performance evaluation on recognizing “urban objects” in omnidirectional images.

Then, after proceeding to the analysis of results achieved in the previous stage, a set of diversified object detection algorithms were trained in the acquired omnidirectional image dataset. This stage allowed us to carry out a domain-specific approach to improve object detection accuracy when compared to 2D image-based methods. The resulting DL algorithms were benchmarked to provide the DL community with the information needed to understand each algorithm’s behavior.

Moving on to the final stage, the development of a framework for enhancing object detection accuracy on omnidirectional was carried out. This stage involved identifying and denoting the main failure points of trained algorithms and investigating an optimized approach to overcome such problems. Finally, the proposed framework was evaluated and compared to the previous methods.

4.3. Comparative Performance Evaluation

This section aims to establish the comparative performance evaluation of currently available networks trained on conventional resolutions and FOV when compared to an omnidirectional image dataset. Firstly, not only the definition of DL algorithms to be used but also a short justification for each choice is provided. At this stage, the main differences between networks are demonstrated.

The evaluation environment was still the same on both datasets and the goal was measuring two Key Performance Indicators (KPIs): mAP and IoU. Moreover, all network configuration parameters used are described to repeat this experiment several times, but not consecutively (e.g. one month later). Finally, performance results and specific aspects analysis, as well as similarities and differences between evaluation experiments, were registered.

4.3.1. Training Process

To perform the training process for the comparative evaluation, Single Shot MultiBox Detector (SSD) in addition to You Only Look Once (YOLO) version 3 (v3) were selected. On the first hand, the SSD [60] approach, due to its unified framework, training and inference speed, and accuracy performance demonstrated on COCO and PascalVOC datasets were used during this experiment. Our implementation of the SSD algorithm follows an open-source Keras-based implementation [68] and it consists of retraining the model on the Cityscapes subset. Therefore, by getting model weights from the ImageNet dataset training process and using the parameters described in Table 2, the training process was carried out.

Property	Description	Value
Network Parameters		
img_height	Network input height	512
img_width	Network input width	512
img_channels	Network input channels	3
swap_channels	The color channel order (BGR, RGB,...)	[2, 1, 0]
scales	List of anchor boxes scaling factors	0.07
		0.15
		0.3
		0.45

		0.6 0.75 0.9 1.05
aspect_ratios	List of aspect ratios for the anchor boxes	[1.0, 2.0, 0.5], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333] [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5], [1.0, 2.0, 0.5]
normalize_coords	Use relative instead of absolute coordinates	True
batch_size	Number of images processed in one batch	8
final_epochs	Max. number of epochs	100
n_classes	Number of output label classes	5
Optimizer Parameters		
optimizer	---	Adam

learning rate	The initial learning rate for training	0.001
beta_1	The exponential decay rate for the first moment estimates	0.9
beta_2	The exponential decay rate for the second-moment estimates	0.999
epsilon	Constant for numerical stability	1^{-8}
decay	How the learning rate of the optimizer changes over time	0.0
Loss Function Parameters		
Loss Function	---	SSD_Loss
neg_pos_ratio	The maximum ratio of negative to positive ground truth boxes to include in the loss computation	3
n_neg_min	The minimum number of negative ground truth boxes to enter the loss computation, per batch	0
alpha	A factor to weight the localization loss in the computation of the total loss	1
Model Checkpoint Callback Parameters		
save_best_only	Save all models or only the best	True

save_only_weights	Save the whole model or only weights	False
monitor	Metric to be monitored	val_loss

Table 2 - SSD Model Training Parameters

Initial experiments on the SSD training process demonstrated that using our local hardware resources was not a feasible option to proceed with the process, as a consequence of exhaustive hardware resource consumption. Then, a cloud-based solution was implemented, allowing to manipulate on-demand computing-accelerated instances, according to the approach's needs.

One of the most-known cloud-based solution provider, Amazon Web Services (AWS) [69], offers computing instances to enable individuals or organizations to train machine learning models through their Sagemaker service [70]. After performing a cost-benefit analysis, *ml.p2.xlarge* instance seemed to be the most appealing instance to be selected. This instance provides *4 virtual CPU (vCPU)*, *1 K80 GPU*, *61 Gibibytes (GiB)* memory, and high network performance.

Following a recommended implementation, Sagemaker service was used along with Simple Cloud Storage Service (S3), also provided by AWS. Both services combined allowed model training progress to be more dynamic given that, best models were, successively, uploaded to the storage service at the time they are available.

Although Sagemaker service offers pre-built object detection frameworks that provide high-level abstraction during the training stage, a Sagemaker algorithm was implemented for controlling more efficiently the whole process. Therefore, the implementation involved developing a lightweight, standalone, executable package of software (known as docker container) that contains everything needed to run an application, including code, system libraries, settings, etc. [71]. This approach is very helpful when we are attempting to isolate the application from its running environment.

The above-mentioned Keras-based source code implementation of SSD was modified to produce a docker container, a process known as *dockerizing* an application. Finally, after its rearrangement, the resulting Docker container was published to another AWS service,

Elastic Container Registry (ECR) [72], which allows SSD model training in Sagemaker service.

In addition to SSD DL model training implementation, the YOLOv3 [73] training framework was also used to enhance the diversity of the proposed reference performance. Our YOLOv3 implementation follows an open-source repository implementation which also provides a step-guide to train this neural network on a custom dataset through transfer-learning techniques. Given the above-mentioned implementation requires a specific framework to be executed, Darknet [74] was locally compiled and, subsequently, installed. This open-source neural network framework, written in C and CUDA supports both CPU and GPU computation.

During the initial stage of pipeline implementation, Cityscapes subset annotations, stored in PascalVOC format for the first training process, were converted to YOLO format and then, validated to ensure conversion process efficiency. Afterward, neural network parameters were adjusted to our dataset. These parameters are identified in Table 3.

Property	Description	Value
Network Parameters		
img_height	Network input height	512
img_width	Network input width	512
img_channels	Network input channels	3
momentum	How much history affects the further change of weights	0.9
batch	Number of images processed in one batch	64
subdivisions	Number of mini-batches to be processed by the GPU at once	32
decay	Weaker updating of weights for eliminating disbalance in the dataset	0.0005
angle	Random changes on images rotation on training	0
saturation	Random changes on images saturation on training	1.5
exposure	Random changes on images brightness on training	1.5

hue	Random changes on images color on training	0.1
learning rate	The initial learning rate for training	0.001
burn_in	For the first X batches, slowly increase the learning rate until its initial value defined above.	1000
max_batches	Max. number of iterations	10000
policy	Policy for changing the learning rate	steps
steps	At these number of iterations, the learning rate is multiplied by the <i>scales</i> factor	8000, 9000
scales	---	0.1, 0.1
classes	Number of output label classes	5
anchors	Object bounding box ratios. Each pair of values is, respectively, height and width.	7, 18, 18, 26, 10, 48, 32, 47, 19, 93, 56, 80, 38, 186, 96, 136, 133, 279

Table 3 - YOLOv3 Model Training Parameters

YOLOv3 training process was computationally hard to accomplish through available hardware resources, identically to the SSD training process. However, at this time, AWS Sagemaker service, due to its associated costs, was replaced by Google Colaboratory (or Colab) [75]. Colab is an open-source framework that allows notebooks to be executed on Google's cloud servers, providing hardware accelerators, including Tensor Processing Unit (TPU) and GPU options.

The final training pipeline consisted of, firstly, downloading the Darknet framework from its repository to Google Drive, which was the storage service used to replace the AWS S3 service. Then, after preparing the labeled dataset and changing the YOLOv3 network configuration parameters file to include the above-described parameters, a Python 3 notebook was created in Google Colab along with the GPU hardware accelerator option enabled. Later, our Google Drive was mounted in Colab's notebook to share resources between both services. Finally, by providing a neural network with pre-trained weights for

the convolutional layers, the DL model was trained with, successive weights uploads to the drive.

4.3.2. Analysis of Results

The comparative performance results analysis section aims to present the main drawbacks identified as well as the defined metrics results to show the behavior of trained models resulting from the training process above-demonstrated.

Firstly, the resulting model provided by the SSD training was evaluated on the Cityscapes subset. This experiment did not require a cloud-based approach given that the inference process needs less sophisticated hardware requirements than DL training. Therefore, the resulting performance for SSD implementation was achieved through a local approach with a Personal Computer (PC) with the following hardware specifications: *Intel Core i7-8750h CPU @ 2.20GHz-2.21 GHz; 16GB RAM; GeForce GTX 1050*. On the other hand, YOLOv3 inference analysis was still executed on Google Colab by using the last trained weights and DL neural network configurations of the training process.

Performance results of DL models on defined Cityscapes subset are demonstrated in Table 4. Values for both AP and mAP values were computed with a minimum intersection over union (IoU) threshold of 0.5, represented in table column headers as *AP@0.5* and *mAP@0.5*, respectively.

	AP@0.5 (%)					mAP@0.5 (%)
	car	truck	bus	motorcycle	person	
SSD	73.2	64.2	65.8	50.1	74.3	65.5
YOLOv3	76.3	63.8	67.1	51.9	75.3	66.9

Table 4 - Performance results on Cityscapes subset with DL models trained on 2D image-based dataset.

The results table shows that in terms of IoU and mAP metrics, the YOLOv3 DL model outperforms the SSD model. While the first-mentioned model achieved a mAP@0.5 of 66.9%, the other only achieved 65.5%. One of the pieces of evidence that led us to find a good reason for this outperforming scenario is the neural network input size that was defined for each one. The most accurate model, YOLOv3, has an input size bigger (608x608) than SSD (512x512), which could be a fundamental aspect to achieve the final results.

Going deeper into the analysis, car and person labels are the most accurate labels on both models due to their higher number of samples on the training dataset. In opposition, given the lack of motorcycle samples, it was the most inaccurate label. In Figure 40, some examples of detections are depicted. On the left side, two people who were walking around were successfully detected although their proximity. On the other hand, the right side shows efficient car detection, even in difficult conditions such as small objects located at the mid-level.

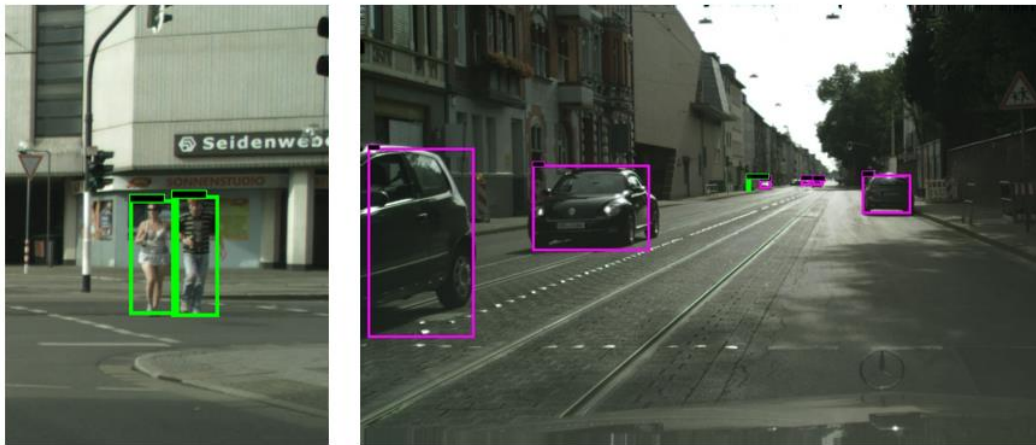


Figure 40 - Cityscapes subset detection examples with DL models trained on 2D images

After analyzing trained model results on the Cityscapes subset, the same models were, then, evaluated on the omnidirectional presented dataset. Given that the acquired dataset does not contain all object classes covered by DL models, evaluation results only contain the performance for car, bus, and person labels. Table 5, which provides performance results on the 360° dataset, follows the same data pattern that was used on the previous evaluation results table.

	AP@0.5 (%)			mAP@0.5 (%)
	car	bus	person	
SSD	47.1	28.3	41.5	39.0
YOLOv3	49.6	30.1	44.7	47.7

Table 5 - Performance results on omnidirectional image dataset with DL models trained on 2D image-based dataset.

Comparing with the evaluation performed on the Cityscapes dataset, the current performance on the omnidirectional dataset has dramatically decreased. Deep learning models have shown clear difficulties to detect an object in this type of image. Car and person labels, the

most accurate classes in the previous experiment, have changed from 76.3 and 75.3 to 49.6 and 44.7 AP@0.5 values in the YOLOv3 case-study, respectively. In the SSD model, the performance impact was very similar, given that AP@0.5 values have decreased from 73.2 and 44.3 to 47.1 and 41.5. An image example of an evaluation performance experiment is depicted in Figure 41, which contains a well-detected car, driving in a roundabout.



Figure 41 - Omnidirectional image detection example with a DL model trained on a 2D image dataset (1)

Although the Cityscapes subset includes cars, buses, and people in different poses, viewpoints, and climate conditions, omnidirectional images have particular aspects that can lead DL models not to detect objects with high accuracy. Firstly, ERP makes images to contain objects in an unusual view pose which made the detection procedure, an unstable process. On the other hand, the object size at the images' mid-region is usually lower.

By splitting the image into three regions (left, center and right), we noted that 63% of non-detected objects are at the center, while just 37% of the remaining failures are distributed by the other regions. These results are demonstrated in Table 6 and led us to conclude that the most problematic objects are located at the center of the image.

Left	Center	Right
16%	63%	21%

Table 6 - Non-detected objects by image region with a DL model trained on a 2D image dataset

Figure 42 and Figure 43 depict the above-mentioned identified problem. Even though the left and right image region objects were well identified, cars located at the mid-level were not. As demonstrated, at the mid-level of images, objects are usually smaller and trained-models' inaccuracy was even more clear.



Figure 42 - Omnidirectional image detection example with a DL model trained on a 2D image dataset (2)



Figure 43 - Omnidirectional image detection example with a DL model trained on a 2D image dataset (3)

As long as object detection algorithms based on 2D images did not meet the accuracy requirements needed in the most demanding contexts, new approaches to optimize the acquired results were investigated.

4.4. Final Remarks

In most recent years, the object detection field has been perhaps one of the most researched topics. People's daily routine has been impacted by such technology however, new

challenges have been raised. Due to 2D camera limitations, UHD 360° cameras have been proposed to perform computer vision tasks to overcome those challenges.

This chapter presented and described the research problem to provide the DL community with a useful research study about object detection and recognition in omnidirectional images. Therefore, three fundamental stages were defined: comparative performance evaluation, domain-specific DL algorithms' training and, proposal of a framework for improving results of object detection accuracy in this image type.

The comparative performance carried out in this chapter denoted that the accuracy significantly decreases from 2D to omnidirectional image dataset, as expected. Moreover, both 2D image-based algorithms demonstrated more difficulty to detect objects near the image center than elsewhere. Given the lack of stability, the next chapter presents mechanisms for improving the achieved results, starting with a domain-specific training approach.

5. Framework for Enhancing Object Detection in Omnidirectional images

In this chapter, a framework for improving object detection algorithms in omnidirectional images is proposed. Firstly, domain-specific DL training experiments to overcome the problematic situations identified in the comparative performance evaluation is carried out. Then, the resulting omnidirectional image-based algorithms are benchmarked to understand the main differences between such models.

Following that section and taking as input the analysis of results from the above-mentioned experiments, an improved framework architecture is presented and evaluated. Finally, aspects concerning the deployment of the proposed framework are considered to provide the reader with the fundamental considerations of the DL algorithms in the decision-making stage.

5.1. Omnidirectional Image Dataset Training

This section details the training experiments specifically applied to the omnidirectional image dataset which was used as input. The same above-used network architectures were selected: SSD and YOLO. However, at this time, we evaluated different versions of each one along with a new architecture: Mask R-CNN.

Different network input sizes were verified, attempting to establish a relationship between performance metrics and network input size and complexity. In terms of performance metrics used, mAP, model complexity, and FLOPs were taken into consideration. Moreover, as in the previous experiments, due to the lack of hardware resources to perform required experiments, AWS and GoogleColab cloud-computing providers were used to accelerate the training process.

5.1.1. Training Process

Firstly, different variations of YOLO were trained on the omnidirectional image dataset. Trained models include the standard version of YOLOv4 and YOLOv3 with an input size of 608x608x3, a faster and less complex version of YOLO, called Tiny-YOLO, in both third and fourth versions with an input size of 416x416x3 and, finally, standard YOLOv4 with an

input size of 800x448x3. Neural networks' parameters that were used during YOLO models' training are depicted in Table 7 and Table 8.

Property	YOLOv3	YOLOv4	YOLOv4 (800x448)
Network Parameters			
img_height	608	608	800
img_width	608	608	448
img_channels	3	3	3
momentum	0.9	0.9	0.9
batch	64	64	64
subdivisions	16	16	16
decay	0.0005	0.0005	0.0005
angle	0	0	0
saturation	1.5	1.5	1.5
exposure	1.5	1.5	1.5
hue	0.1	0.1	0.1
learning rate	0.001	0.0013	0.0013
burn_in	1000	1000	1000
max_batches	12000	12000	12000
policy	steps	steps	steps
steps	9600, 10800	9600, 10800	9600, 10800
scales	0.1, 0.1	0.1, 0.1	0.1, 0.1
mosaic	----- ⁵	1	1
classes	6	6	6
anchors	12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401	12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401	12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401

Table 7 - Standard YOLOv3 and YOLOv4 and YOLOv4 (800x448) network parameters

⁵ YOLOv3 model does not support 'mosaic' parameter

Property	Tiny-YOLOv3	Tiny-YOLOv4
Network Parameters		
img_height	416	416
img_width	416	416
img_channels	3	3
momentum	0.9	0.9
batch	64	64
subdivisions	16	16
decay	0.0005	0.0005
angle	0	0
saturation	1.5	1.5
exposure	1.5	1.5
hue	0.1	0.1
learning rate	0.001	0.00261
burn_in	1000	1000
max_batches	12000	12000
policy	steps	steps
steps	9600, 10800	9600, 10800
scales	0.1, 0.1	0.1, 0.1
classes	6	6
anchors	10,14, 23,27, 37,58, 81,82, 135,169, 344,319	10,14, 23,27, 37,58, 81,82, 135,169, 344,319

Table 8 - Tiny YOLOv3 and YOLOv4 network parameters

In addition to YOLO model experiments, SSD architecture was also evaluated. Experiments included two variations of this architecture, modifying not only the network input size but also loss function attributes. As well as demonstrated for YOLO models, Table 9 presents SSD with 300x300x3 of input size and SSD with 512x512x3 of input size network parameters.

Property	SSD 300x300	SSD 512x512
Network Parameters		
img_height	300	512
img_width	300	512
img_channels	3	3
swap_channels	[2, 1, 0]	[2, 1, 0]
scales	0.07, 0.15, 0.3, 0.45, 0.6, 0.75, 0.9, 1.05	0.07, 0.15, 0.3, 0.45, 0.6, 0.75, 0.9, 1.05
aspect_ratios	[1.0, 2.0, 0.5], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5], [1.0, 2.0, 0.5]	[1.0, 2.0, 0.5], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5, 3.0, 0.33333333], [1.0, 2.0, 0.5], [1.0, 2.0, 0.5]
normalize_coords	True	True
batch_size	8	8
final_epochs	100	100
n_classes	6	6
Optimizer Parameters		
optimizer	SGD	Adam
learning rate	0.001	0.001
momentum	0.9	----

beta_1	----	0.9
beta_2	----	0.999
epsilon	----	1^{-8}
nesterov	false	----
decay	0.0	0.0
Loss Function Parameters		
Loss Function	SSD_Loss	SSD_Loss
neg_pos_ratio	3	3
n_neg_min	0	0
alpha	1	1
Model Checkpoint Callback Parameters		
save_best_only	True	True
save_only_weights	False	False
monitor	val_loss	val_loss

Table 9 - SSD 300x300 and SSD 512x512 network parameters

Finally, an instance segmentation DL network, namely Mask R-CNN, was prepared to provide the models' benchmarking analysis with diversified network structures. For that purpose, an open-source online available implementation [77] to adjust the provided solution to our context and dataset was extended. The model's parameters used in the training stage are described in Table 10. The process was dramatically accelerated by using transfer-learning techniques by taking pre-trained weights as a starting point for our train. In this specific case, we took advantage of weights provided by a Mask R-CNN train on the MS COCO dataset.

Mask R-CNN	
Property	Value
num_classes	6
gpu_count	1
images_per_gpu	2
backbone	resnet101
compute_backbone_shape	None
backbone_strides	[4, 8, 16, 32, 64]

fpn_classif_fc_layers_size	1024
top_down_pyramid_size	256
rpn_anchor_scales	(32, 64, 128, 256, 512)
rpn_anchor_ratios	[0.5, 1, 2]
rpn_anchor_stride	1
rpn_nms_threshold	0.7
rpn_train_anchors_per_image	256
pre_nms_limit	6000
post_nms_rois_training	2000
post_nms_rois_inference	1000
use_mini_mask	True
mini_mask_shape	(56, 56)
image_resize_mode	square
image_min_dim	800
image_max_dim	1024
image_min_scale	0
image_channel_count	3
mean_pixel	np.array([123.7, 116.9, 103.9])
train_rois_per_image	200
roi_positive_ratio	0.33
pool_size	7
mask_pool_size	14
mask_shape	[28, 28]
max_gt_instances	100
rpn_bbox_std_dev	np.array([0.1, 0.1, 0.2, 0.2])
bbox_std_dev	np.array([0.1, 0.1, 0.2, 0.2])
detection_max_instances	100
detection_nms_threshold	0.3
learning_rate	0.001
learning_momentum	0.9
weight_decay	0.0001

loss_weights	{“rpn_class_loss”: 1., “rpn_bbox_loss”: 1., “mrcnn_class_loss”: 1., “mrcnn_bbox_loss”: 1., “mrcnn_mask_loss”: 1.}
use_rpn_rois	True
train_bn	False

Table 10 - Mask R-CNN network parameters

5.1.2. Analysis of Results

The analysis of results section focuses on providing a detailed evaluation of the trained models' report. As previously referred, three fundamental performance metrics were measured: mAP, to evaluate models' accuracy, FLOPs, for taking into consideration the computation cost of each deep neural network, and, finally, the model complexity, given by the number of learning parameters. Furthermore, each model inference speed was computed by measuring the elapsed time between the exact moment when the algorithm receives an image and the moment when its predictions are available.

To be possible getting a nonsubjective analysis, the evaluation environment was still the same for measuring all the performance metrics: a Windows 10 machine with *Intel® Core™ i7-8750H CPU @2.20GHz 2.21 GHz; 16,0 GB RAM; NVIDIA GeForce GTX 1050*.

Results were later aggregated so that it is easier to compare each one accordingly to different criteria. Figure 44 and Figure 45 present the final report of the deep neural networks evaluation process. The first figure relates the models' mAP@0.5 (y-axis) with their computational cost (x-axis), as well as their complexity (circle diameter). On the other hand, the second figure depicts not only the relationship between the models' mAP and computed inference time but also their complexity.

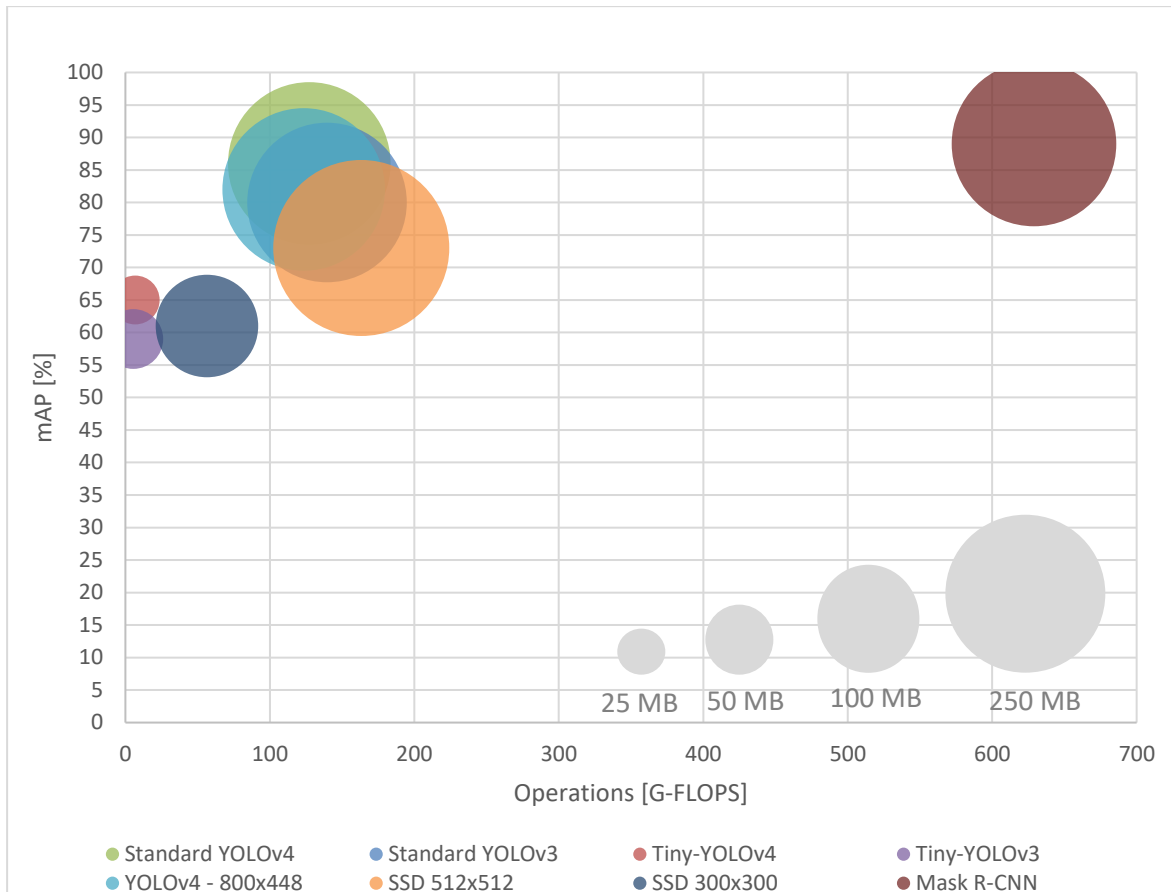


Figure 44 - Ball chart reporting models' mean average precision (mAP) vs computational complexity

In regard to mAP, the Mask R-CNN DL algorithm provided the highest score (89%), whereas Tiny YOLOv3 seemed to have more difficulties in detecting objects, given that it only achieved 59% of mAP. In the same way as the less accurate model, SSD 300x300 and Tiny YOLOv4 did not efficiently detect objects with high accuracy. Afterward, standard YOLOv3 and YOLOv4 800x448 reached a similar result: 80% and 82%, respectively. In opposition, standard YOLOv4 models outperformed the aforementioned methods by providing 86% of mAP. Finally, the remaining model, SSD 512x512, demonstrated some problems in getting a high-level accuracy rate, by achieving a 73% mAP.

In terms of complexity, three groups of similar models were identified. Firstly, Tiny YOLOv4, followed by the Tiny YOLOv3 DL algorithm belongs to the less complex group, having a measured complexity of around 25 megabytes (MB). Different from the first group, the medium-complexity group just includes the SSD 300x300 model with about 100 MB. Lastly, the group where the vast majority of models fit aggregates models that have a measured complexity close to 250 MB. That group contains the standard YOLOv3 and YOLOv4, YOLOv4 800x448, SSD 512x512, and Mask R-CNN.

Unfortunately, no satisfactory correlation between the number of parameters of models and mAP was found. However, models with fewer parameters tend to be ineffective at detecting objects. Concerning the relation between mAP and computational complexity, we noticed that the model which requires more complex hardware resources is also the most accurate. The same pattern was not followed by the remaining models given that, in some cases, less complex models outperformed more complex models. For instance, Tiny YOLOv4 was more accurate than SSD 300x300, although its minimal cost in terms of hardware resources.

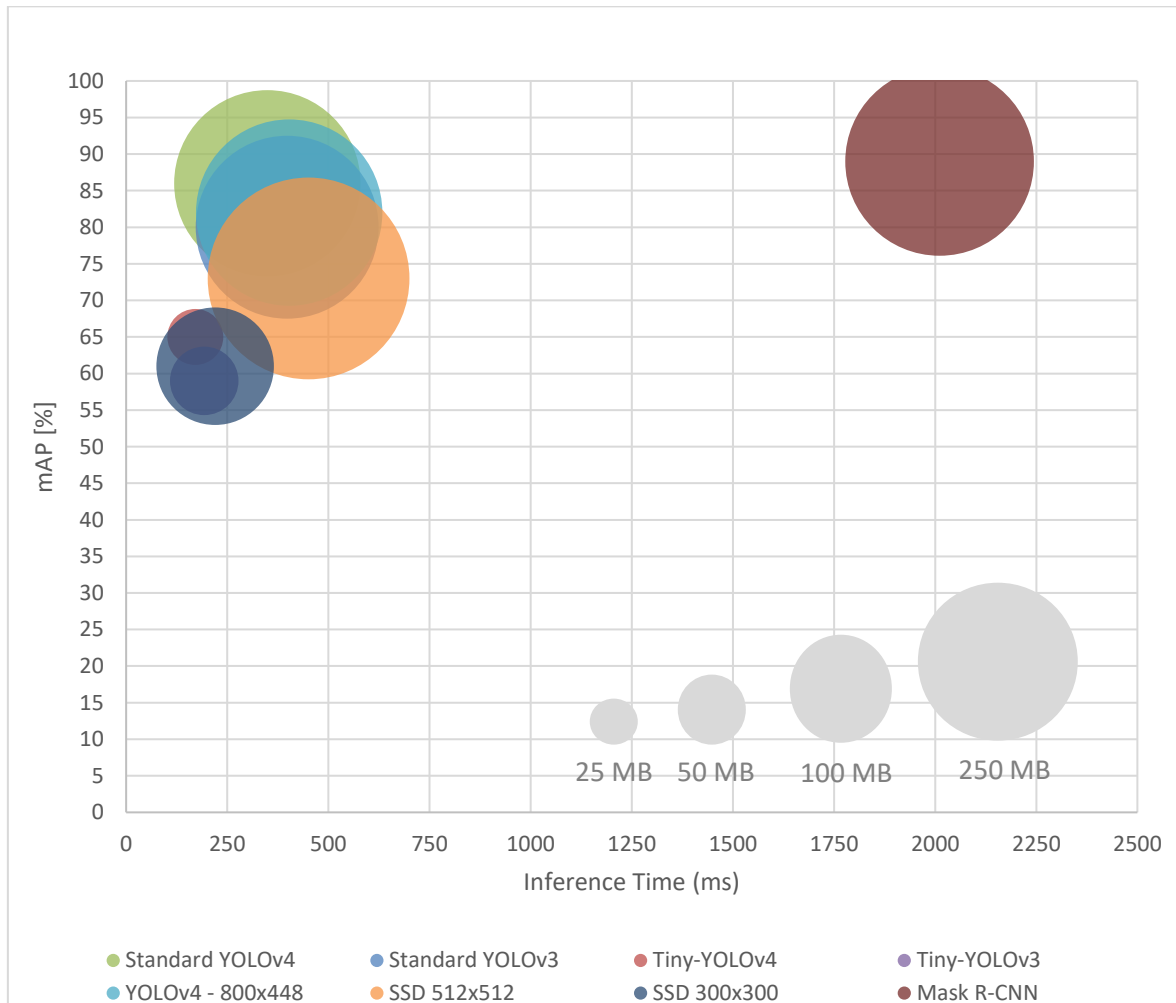


Figure 45 - Ball chart reporting models' mean average precision (mAP) vs inference time

In opposition to the previous figure, Figure 45 shows the relation between the mAP and the time each model needs to detect objects in a single image, referred to as inference time. As above-described, the evaluation conditions of all models were the same to ensure the validation of analysis and comparison of results. To clarify such results, Table 11 presents each trained DL model associated with the measured inference time as well as its standard deviation.

DL Model	Inference Time (ms)
Tiny-YOLOv4	171 ± 3.21
Tiny-YOLOv3	193 ± 2.98
SSD 300x300	220 ± 5.46
Standard YOLOv4	349 ± 5.83
Standard YOLOv3	398 ± 6.41
YOLOv4 - 800x448	403 ± 5.95
SSD 512x512	451 ± 8.23
Mask R-CNN	2011 ± 4.23

Table 11 - Inference Time Results. DL model name presented in the first column, associated with the measured inference time, in the second column.

By comparing each model individually, Mask R-CNN was undoubtedly the model that requires more time to return its detection results (2011 ms). Despite the long processing time, these results were more accurate in comparison to the remaining models. On the other hand, for the models with the lowest inference time, the values are usually the least accurate, which tends to be a pattern on the results obtained in this study. Tiny-YOLOv4, Tiny-YOLOv3 and, SSD 300x300 belong to the fastest models' group, however that group is characterized by its lack of accuracy.

Moving on to the mid-level algorithms, their measured inference time was very similar, ranging between 349 and 451 ms. In this final group, standard YOLOv4 reached better inference time and accuracy values, while SSD 512x512 did not ensure high-accuracy values and it required more time to process an image. Finally, standard YOLOv3 and YOLOv4 - 800x448 achieved 398 and 403 ms, respectively.

Although omnidirectional image-based DL algorithms demonstrated to be more accurate than 2D image-based models, the middle region of images is still the most problematic. As seen in Table 12, objects located at the center tend to be more difficult to detect, while left and right-positioned objects are easier detected.

Left	Center	Right
25%	42%	33%

Table 12 - Non-detected object by region in omnidirectional images

When compared to Table 6, where the same metric for 2D image-based algorithm was depicted, results followed a more uniform distribution given that the mid-region error rate decreased from 63% to 42%. With regards to the remaining regions, the left and right regions' error rate increased from 16% to 25% and 21% to 33%, respectively.

However, current values for the above-presented performance measure did not allow an omnidirectional image-based object detection framework to fulfill current requirements on most common applications. Then, a framework for making the non-detected objects' rate more uniform is proposed in the next section.

5.2.Improved Framework

The improved framework proposed in this section is explained utilizing specific mechanisms to uniformize the error rate across the whole spherical image regions. The initial architecture is, firstly, presented, followed by the second version of the framework, which was devised to overcome the initial drawbacks.

In opposition to left and right regions, objects located at the center have a propensity to be smaller, which could be a crucial fact to justify the results of the previous experiments. For that reason, the proposed framework involves applying two parallel pipelines: the first one focusing on the whole image, and the second just concentrating on the middle region.

The first pipeline follows the same pattern as traditional object detection frameworks, where the whole image is processed by a given DL algorithm and the inherent predictions are returned. On the other hand, only the image mid-region is processed in the second stage, however, instead of processing that region once, the proposed framework requires the mid-region to be separated into two blocks, as depicted in Figure 46.

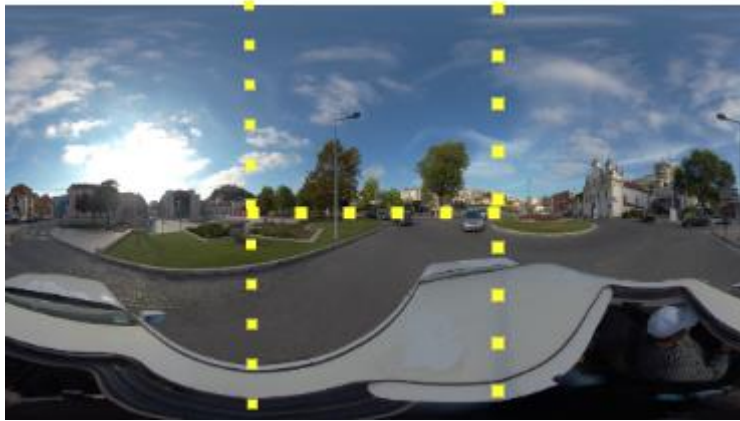


Figure 46 - Omnidirectional mid-region image first division

As long as two image blocks are provided by the second pipeline, multiple inference processes are also required. Given that multiple object detection processes produce multiple results, the final prediction output requires such results to be rearranged in a post-processing stage. The proposed framework architecture, presented in Figure 47, shows both parallel pipelines with a pre-processing step in the bottom pipeline which crops image mid-region into sub-images. All images are processed by the DL algorithms and all results are later aggregated in the framework's last stage.

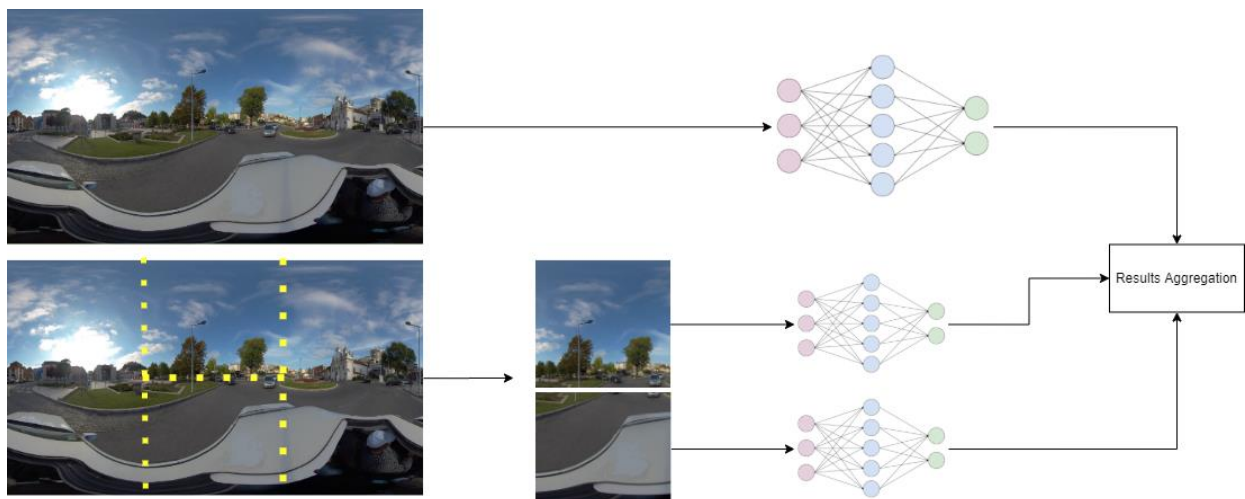


Figure 47 - Proposed framework initial architecture

Although the initial expectations on the framework results, such experiments demonstrated that an eventual issue concerning objects located at the second pipeline sub-division blocks could invalidate its success. The identified issue occurs when an object instance overlaps both sub-regions at the same time, eventually producing multiple object counts to the final

results. Consequently, a new framework version was proposed, following a workflow as depicted in Figure 48.

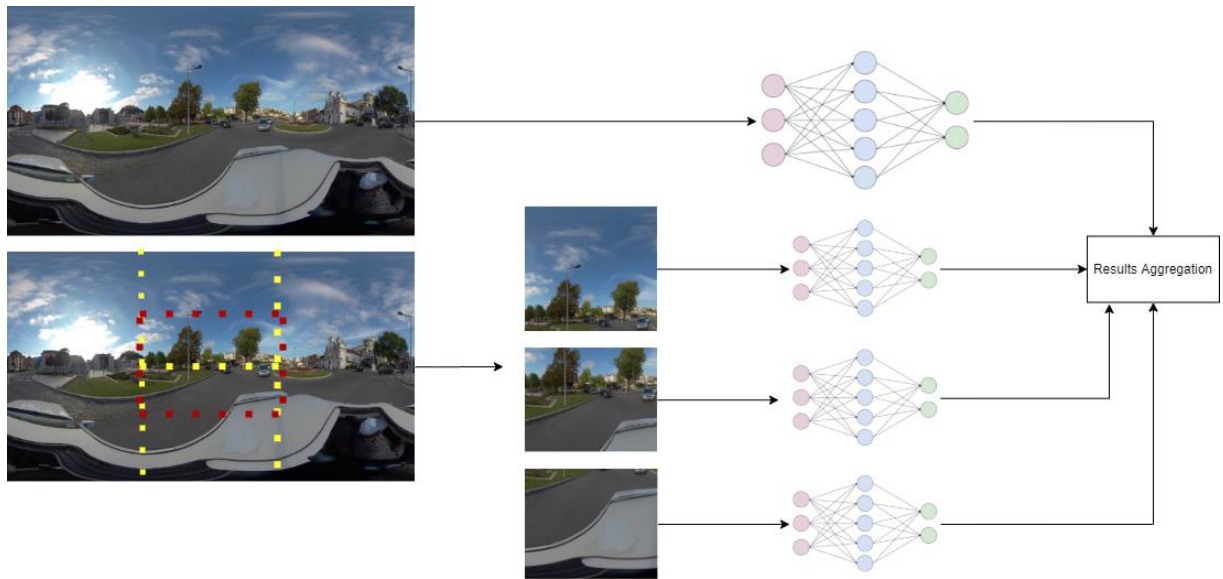


Figure 48 - Proposed framework final architecture

The new version introduces a third subdivision which covers the initial regions intersection limit to prevent errors associated with objects at such limit. Therefore, objects detected from the DL algorithms in both top and bottom sub-regions that are almost touching the block's bottom and top, are discarded. On the other hand, mid-region predictions remain to be later aggregated in the post-processing stage.

To evaluate the proposed framework efficiency, we focused not only on comparing its inference time but also on non-detected objects' error rate by image region, the same metrics presented in the previous experiments.

5.2.1. Analysis of Results

This section aims to demonstrate evaluation results through the above-presented object detection framework architecture. To ensure testing veracity, the evaluation environment was still the same.

In terms of inference time, given the higher complexity when compared to the conventional object detection frameworks, we expected the inference time performance to decrease which was verified, as depicted in Table 13. Object detection in omnidirectional images increased from 349 ms to 1152 ms through the proposed framework.

Traditional Framework	Proposed Framework
Inference Time (ms)	
349 ± 5.83	1152 ± 8.45

Table 13 - Inference time results through the traditional and proposed object detection framework.

Considering the non-detected objects' rate by image region, the proposed framework improved that evaluation metric value from 42% to 39%, providing a more uniform error distribution. Left and right regions achieved 27% and 34%, respectively, as depicted in Table 14.

	Left	Center	Right
Traditional Framework	25%	42%	33%
Proposed Framework	27%	39%	34%

Table 14 - Non-detected objects by image region through the traditional and proposed object detection framework.

Although the object detection framework's inference time increased when compared to the traditional framework, some mid-region located object instances which were not previously detected were detected through this new approach. Figure 49 supports this sentence by depicting an example where the traditional framework did not identify mid-region located cars, probably, because of their small size, are now successfully detected through the proposed framework.

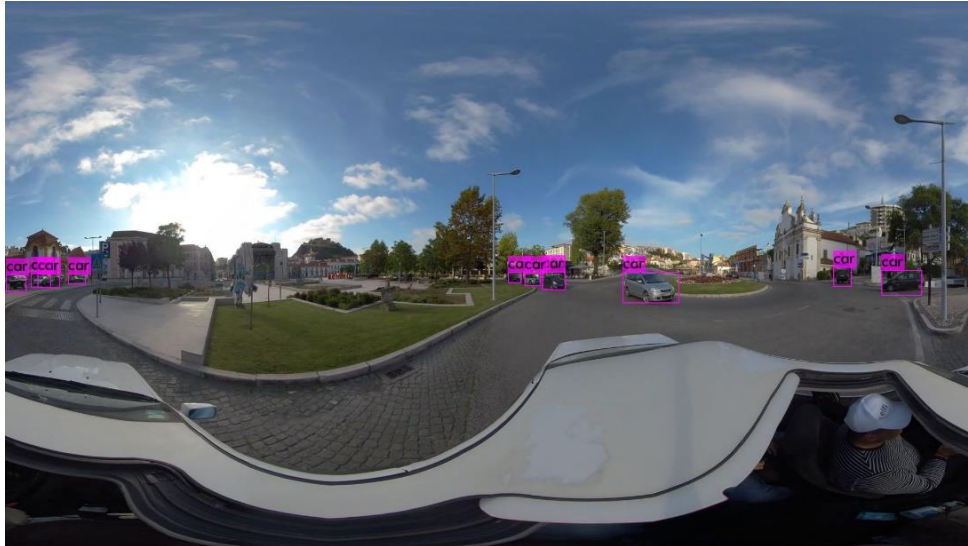


Figure 49 - Proposed framework predictions example (1)

After presenting the results concerning DL models trained on ERP images through both traditional models and the proposed framework previously presented, the next section describes the framework deployment and DL algorithms that better fit a given scenario.

5.3. Framework Deployment

Nowadays, not only in object detection domain problems but also in most cases involving technology, scalability is a mandatory requirement. The ability to ensure the availability of service is maintained according to the number of users, available hardware resources, or even, from a business-level perspective, the financial plan offered for the project are important aspects, upon which the final decision relies.

Consequently, to develop a technological product, some imposed constraints drastically change how the project proceeds. In terms of hardware resources, three main approaches are usually available depending on financial and connectivity constraints. In a non-existent or weak internet connection scenario, or when the latency is critical to ensure the operation's success, physical (non-cloud-based) hardware resources are a wise option. Moreover, prices are usually lower than cloud-based solutions, although their inherent maintenance costs.

Automatically associating object detection tasks with high-computational resources tends to overestimate the required hardware to perform such tasks. However, tiny, low cost and credit-card-sized devices, such as Raspberry Pi 4 Model B (Figure 50) [78], are ready to run less complex DL algorithms. In opposition, given the limited resources available on these

devices, deploying algorithms that execute a large number of operations and require a considerable time to provide detected objects is not a feasible task.



Figure 50 - Raspberry Pi 4 Model B [78]

Different from those simple devices, more powerful computers that let object detection tasks to be performed through multiple neural network parallel execution [79] are also a good physical framework option. This device group of devices includes Jetson Nano (Figure 51), developed by NVIDIA, which allows more complex DL models, to be run through GPU acceleration at an accessible cost. It is also important to note that both Raspberry Pi and Jetson Nano support camera modules to facilitate image capture which is a fundamental stage on object detection tasks.



Figure 51 - Jetson Nano Developer Kit [79]

Still regarding local devices, however, in a higher budget level, local servers are also a suitable option for DL algorithms deployment scenarios that require high-computational hardware resources. A local server provides different computational capabilities, depending on its specifications and pricing. Figure 52 depicts an example of a local server, namely, PowerEdge R240 Dell Server.



Figure 52 - PowerEdge R240 Dell Server [80]

In situations that do not require low-latency values and internet connection is not a typical issue, cloud-based frameworks make scalability, management, and simple deployment easier to achieve. Given their organized infrastructures, they allow to dynamically change the machine which hosts the framework without any concerns about its maintenance. Furthermore, cloud-based frameworks are new deployment compliant in terms of logistics which facilitates new product releases.

Despite cloud-based frameworks' advantages, their cost is not always affordable to companies' budgets, and, depending on requirements, they could not be the most suitable solution.

5.3.1. Real-world scenarios

Considering that is not possible to select a single model for meeting all established requirements, this section presents a list of scenarios with real-world application to understand the situation where each DL algorithm presented in this dissertation outperforms the others.

The first scenario is described as a non-critical situation whose main goal consists of automatically recognizing and locating cars, motorcycles, trucks, and buses for later extract some kind of statistical metrics to identify rush hours. On that premise, a medium-value of accuracy is enough to mitigate the initial problem due to the error-margin allowed. In addition to the above-mentioned fact, there are no strict time limits for performing object recognition tasks. For that last reason, DL models' selection range is very wide however, to minimize framework costs, algorithms that require less computational hardware resources, such as Tiny-YOLOv4, Tiny-YOLOv3, or SSD 300x300 seem to be an optimal choice.

Taking into consideration the deployment environment, any constraint that forces a cloud or non-cloud/local deployment was defined, so it depends on the system architecture as well as the financial plan associated with the project.

Secondly, the next scenario requires an automatic framework for detecting UAVs, people, and vehicles to avoid the invasion of privacy issues which means that real-time alerting is a mandatory requirement. In opposition to the first example, a non-detected object is a critical point of failure on the system and puts the whole system efficiently at risk.

In this specific example, high object detection accuracy rather than minimizing solution' cost defines the framework architecture. Moreover, finding a sweet spot between processing/inference time and accuracy is the first goal given that one of the pre-defined constraints includes real-time monitoring. Subsequently, an alarm should be triggered as soon as possible to minimize the reaction time.

From the previously demonstrated experiments, the high-mAP model group is comprised of Mask R-CNN, standard YOLOv3 and YOLOv4, and YOLOv4 800x448. Although in terms of this performance metric, Mask R-CNN outperforms the remaining algorithms, Table 11 shows that it is also the model that requires more time for processing a single image.

Therefore, the framework development should start with standard YOLOv4 by itself evaluation, followed by an analysis of standard YOLOv4 through this dissertation proposed framework. Depending on accuracy results, a choice should be taken, considering that the system efficiency must not be negotiated. Generally, critical scenarios require a higher hardware resources investment so that results are available near real-time with the minimum error associated, which tends to make non-cloud deployment a not suitable option.

Finally, the last presented scenario involves developing an Application Programming Interface (API) for locating vehicle license plates. As a consequence of requiring an online availability, a cloud-based deployment through any cloud solution existing in the market has to be considered. Besides, accurate and fast results should be provided to guarantee the financial return.

Given the above-mentioned constraints, standard YOLOv4 with a cloud-based deployment seems to fulfill specified requirements. It provides satisfactory accuracy results with reasonable inference time measured and it does not require high-computational resources which allows saving cloud resources costs.

Summing up, considering that a DL model is better in general terms is not fair. All presented models have their applications where their main advantages could be emphasized to achieve project requirements.

5.4. Final Remarks

In this chapter, for improving object detection algorithms on detecting objects in omnidirectional images, a domain-specific training stage was carried out. That stage involved training multiple DL algorithms fed by omnidirectional images.

Although the resulting models' accuracy performance increased when compared to the previous experiments, we noted that the error rate by image region is still not uniform across the whole spherical image regions. For that reason, a framework for improving results and reducing the error rate was proposed. In opposition to traditional frameworks, two parallel stages are performed: the whole image processing and middle image sub-regions predictions processing. Additionally, the framework introduces a post-processing stage for results aggregation.

Performance results concerning the above-mentioned framework were analyzed, by measuring not only the error rate associated with both traditional and proposed approaches but also the average inference time measured for image processing. As long as the framework requires more processing time, inference time increased. However, in terms of error rate by image region, we were able to improve the previous experiments' results.

Finally, to provide the reader with more information on the DL model decision-making, a set of real-world scenarios were presented. Then, each algorithm was associated with the presented scenarios to explain that is not fair to consider a model better than the other, everything depends on the scenario.

6. Conclusion and Future Work

In this dissertation, a new method for improving object detection in omnidirectional images was proposed. Such research study required an omnidirectional image dataset acquisition stage which involved 360° video capture and image labeling given that open-source labeled omnidirectional image datasets on the urban environment were not easily available.

Initial experiments on comparing 2D image-based DL algorithms' accuracy on 2D planar and omnidirectional images provided interesting results. That comparative performance allowed us to note not only an accuracy performance decrease from 2D to omnidirectional image dataset but also a non-uniform error rate across the whole spherical image regions. Such fact led our research to focus on a domain-specific DL model training process.

Then, a benchmarking report of DL algorithms trained with the omnidirectional acquired image dataset was presented and analyzed. The results achieved supported our initial thoughts of providing more accurate results through a domain-specific approach when compared to a generic algorithm. Moreover, a comparison between a set of DL algorithms in terms of accuracy, complexity, and inference time was demonstrated to understand the main differences between DL models.

Although the resulting DL algorithms from the above-mentioned training provided more accurate results, the error rate was still not uniform across the whole image regions. Objects located at both left and right image regions tended to be easier to identify than mid-region objects which led us to propose a new approach to overcome the identified issue.

The proposed approach consists of adding pre and post-processing stages to the traditional object detection framework across two parallel pipelines. The first pipeline focuses on the whole image and follows the same pattern as traditional object detection frameworks, where the whole image is processed by a given DL algorithm and the inherent predictions are returned. On the other hand, only the image mid-region is processed in the second stage. However, instead of processing that region once, the proposed framework requires that region to be divided into three blocks which are individually processed to return predictions. All predictions are aggregated in the post-processing stage.

This framework allowed the error rate to be more uniform across the whole image regions however, given that more processing is involved in the framework, the inference time increased, as demonstrated in the results section.

Regarding future work, implementing this solution in a real-world scenario and evaluate its accuracy would be the first step to take, since the actual procedure was only evaluated in a controlled environment. Further evaluation could create opportunities for identifying issues on the current framework and, consequently, add more robustness to the implemented approach.

Then, taking as input the evaluation results and after implementing the inherent improvements, an automatic video surveillance system with capabilities of detecting objects in all view directions would be developed. That system could completely transform current video surveillance systems and solve security and privacy issues imposed by the recent technological advances.

Bibliography

- [1] B. J. Baars and N. M. Gage, "Vision," in *Cognition, Brain, and Consciousness*, Academic Press, 2010, pp. 156-193.
- [2] Q. Wu, Y. Liu, Q. Li, S. Jin and F. Li, "The application of deep learning in computer vision," *Chinese Automation Congress (CAC)*, no. 17469740, pp. 6522-6527, 2017.
- [3] Z.-Q. Zhao, P. Zheng, S.-t. Xu and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, 2019.
- [4] K. Balaji and K. Lavanya, "Medical Image Analysis With Deep Neural Networks," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, 2019, pp. 75-97.
- [5] "360-degree camera, Press Releases," 31 12 2019. [Online]. Available: <https://www.marketsandmarkets.com/PressReleases/360-degree-camera.asp>.
- [6] M. Xu, C. Li, S. Zhang and P. Le Callet, "State-of-the-Art in 360° Video/Image Processing: Perception, Assessment and Compression," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 1, pp. 5-26, 2020.
- [7] H. E. R. Gernsheim, A. Grundberg, B. Newhall and N. Rosenblum, "History of photography," Encyclopædia Britannica, inc., [Online]. Available: <https://www.britannica.com/technology/photography>. [Accessed 10 10 2020].
- [8] "Encyclopædia Britannica," [Online]. Available: <https://www.britannica.com/technology/camera-obscura-photography#/media/1/90865/118840>. [Accessed 10 10 2020].
- [9] D. Kocak and B. Ouyang, "Underwater imaging: photographic, digital and video techniques," in *Subsea Optics and Imaging*, Woodhead, 2013, pp. 275-293.
- [10] T. Jokela, J. Ojala and K. Väänänen, "How people use 360-degree cameras," *Proceedings of the 18th International Conference on Mobile and Ubiquitous Multimedia*, no. 18, pp. 1-10, 10 2019.
- [11] "Insta 360," [Online]. Available: <https://www.insta360.com/product/insta360-pro2>. [Accessed 25 03 2020].

- [12] W. Zhang, X. Peng, J. S. Shi and Z. Guo, “Computing Foundations for Computational Science Final Project,” Harvard University, 2018. [Online]. Available: <https://cs205-stitching.github.io/>. [Accessed 5 9 2020].
- [13] A. Mehrfard, J. Fotouhi, G. Taylor, T. Forster, N. Navab and B. Fuerst, A *Comparative Analysis of Virtual Reality Head-Mounted Display Systems*, 5 12 2019.
- [14] “Class VR,” Avantis Systems Ltd, [Online]. Available: <https://www.classvr.com/school-virtual-reality/>. [Accessed 25 04 2020].
- [15] E. Ghaderpour, “Map Projection,” *York University Department of Earth and Space Science and Engineering, Toronto, Canada*, 16 12 2014.
- [16] Maugey, Thomas, O. L. Meur and L. Zhi, “Saliency-based navigation in omnidirectional image,” *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP), Luton*, pp. 1-6, 2017.
- [17] Y. Ye, E. Alshina and J. Boyce, *JVET-H1004: Algorithm descriptions of projection format conversion and video quality metrics in 360Lib*, 7 2018.
- [18] “Seagate,” [Online]. Available: <https://www.seagate.com/pt/pt/our-story/data-age-2025/>. [Accessed 05 09 2020].
- [19] A. Mikołajczyk and G. Michał, “Data augmentation for improving deep learning in image classification problem,” *2018 International Interdisciplinary PhD Workshop (IIPhDW), Swinoujście*, pp. 117-122, 2018.
- [20] M. V. G. L. W. C. K. I. W. J. a. Z. A. Everingham, “The PASCAL Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision*, vol. 88, pp. 303-338, 2010.
- [21] “Image Net,” [Online]. Available: <http://image-net.org/>. [Accessed 28 03 2020].
- [22] “COCO Dataset,” [Online]. Available: <http://cocodataset.org/>. [Accessed 28 03 2020].
- [23] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
- [24] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

- [25] G. A. Mille, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39-41, 1995.
- [26] P. University, "WordNet," Princeton University, 2010. [Online]. Available: <https://wordnet.princeton.edu/>. [Accessed 10 10 2020].
- [27] "Image Net," [Online]. Available: http://www.image-net.org/papers/imagenet_cvpr09.pdf. [Accessed 28 03 2020].
- [28] "FFmpeg," [Online]. Available: <https://www.ffmpeg.org/>. [Accessed 21 05 2020].
- [29] Tzutalin, "LabelImg," *GitHub*, no. <https://github.com/tzutalin/labelImg>, 2015.
- [30] "360designs," [Online]. Available: <http://360designs.io/eye/applications/>. [Accessed 5 9 2020].
- [31] N. Terashima, Intelligent Communication Systems, 2002.
- [32] Y. Tan, GPU-based Parallel Implementation of Swarm Intelligence Algorithms, 2016.
- [33] T. W. Edgar and D. O. Manz, Research Methods for Cyber Security, 2017.
- [34] G. Shobha and S. Rangaswamy, Handbook of Statistics, 2018.
- [35] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA*, pp. I-I, 2001.
- [36] N. Dala and B. Triggs, *Histograms of Oriented Gradients for Human Detection*, p. 8.
- [37] L. Vanneschi and M. Castelli, "Multilayer Perceptrons," in *Encyclopedia of Bioinformatics and Computational Biology*, 2019, pp. 612-620.
- [38] K. O' Shea and N. Ryan, "An Introduction to Convolutional Neural Network," *ArXiv: Neural and Evolutionary Computing*, 2015.
- [39] V. N. Vapnik, "An Overview of Statistical Learning Theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988-999, 1999.
- [40] T. Evgeniou and M. Pontil, "Support Vector Machines: Theory and Applications," *Lecture Notes in Computer Science*, vol. 2049, pp. 249-257, 1 2001.
- [41] N. Surantha, S. M. Isa, T. Fennia Lesmana and A. Setiawan, "Sleep Stage Classification using the Combination of SVM and PSO," *1st International Conference on Informatics and Computational Sciences (ICICoS)*, 2017.

- [42] K. Veropoulos, G. Learmonth, C. Campbell, B. Knight and J. Simpson, “Automated identification of tubercle bacilli in sputum. A preliminary investigation,” *Anal Quant Cytol Histol.*, pp. 277-82, 1999.
- [43] L. H. Thai, T. S. Hai and N. T. Thuy, “Image Classification using Support Vector Machine and Artificial Neural Network,” *International Journal of Information Technology and Computer Science*, p. 4, 5 2012.
- [44] S. Misra and H. Li, “Noninvasive fracture characterization based on the classification of sonic wave travel times,” in *Machine Learning for Subsurface Characterization*, 2020, pp. 243-287.
- [45] J. Walsh, N. O'Mahony, S. Campbell and A. Carvalho, “Deep Learning vs. Traditional Computer Vision,” *Proceedings of the computer vision conference (CVC 2019)*, p. 128–144, 25-26 4 2019.
- [46] I. Tabian, H. Fu and Z. Sharif Khodaei, “A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures,” *Sensors*, vol. 19, no. 4933, 2019.
- [47] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.
- [48] W. Dai and D. Berleant, “Benchmarking Contemporary Deep Learning Hardware and Frameworks: A Survey of Qualitative Metrics,” *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI), Los Angeles, CA, USA*, pp. 148-155, 23 11 2019.
- [49] A. Shatnawi, G. Al-Bdour, R. Al-Qurran and M. Al-Ayyoub, “A comparative study of open source deep learning frameworks,” *2018 9th International Conference on Information and Communication Systems (ICICS), Irbid*, pp. 72-77, 4 2018.
- [50] “365datascience,” [Online]. Available: <https://365datascience.com/deep-learning-frameworks-2019/>. [Accessed 16 05 2020].
- [51] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, “Tensorflow: a system for large-scale machine learning,” *OSDI*, vol. 16, pp. 265-283, 2016.
- [52] F. Seide and A. Agarwal, “Cntk: Microsoft’s open-source deep-learning toolkit,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. ACM, p. 2135, 2016.

- [53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, Kopf, reas, E. Yang, Z. DeVito, M. Raison and Tejani, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024--8035.
- [54] R. C. a. S. B. a. J. Mariéthoz, “Torch: a modular machine learning software library,” *IDIAP*, 2002.
- [55] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *Proceedings of the 22nd ACM international conference on Multimedia*, p. 675–678, 2014.
- [56] “Keras,” [Online]. Available: <https://keras.io/>. [Accessed 15 05 2020].
- [57] A. R. Pathak, M. Pandey and S. Rautaray, “Application of Deep Learning for Object Detection,” in *Procedia Computer Science*, 2018, pp. 1706-1717.
- [58] R. Girshick, J. Donahue, T. Darrell and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” pp. 1-3, 22 10 2014.
- [59] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779-788, 2016.
- [60] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single Shot MultiBox Detector,” *Lecture Notes in Computer Science*, pp. 21-37, 29 12 2016.
- [61] W. Yang, Y. Qian, F. Cricri, L. Fan and J.-K. Kamarainen, *Object Detection in Equirectangular Panorama*, pp. 1-6, 21 5 2018.
- [62] Y. Li, G. Tong, H. Gao, Y. Wang, L. Zhang and H. Chen, “Pano-RSOD: A Dataset and Benchmark for Panoramic Road Scene Object Detection,” *Electronics*, vol. 329, no. 3, p. 8, 11 3 2019.
- [63] “Pano-RSOD Dataset,” [Online]. Available: <https://pan.baidu.com/share/init?surl=H9RsXfXCCfBgpF2bY2LGeA>. [Accessed 9 5 2020].
- [64] A. Rosebrock, “PyImageSearch,” 7 11 2016. [Online]. Available: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Accessed 1 1 2020].

- [65] “IBM,” [Online]. Available: ibm.com/ibm/history/ibm100/us/en/icons/petaflopbarrier/. [Accessed 20 06 2020].
- [66] A. Jaimes, S. Kota and J. Gomez, “An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles UAV(s),” *IEEE International Conference on System of Systems Engineering*, pp. 1-6, 2008.
- [67] “Gogladly,” [Online]. Available: <https://gogladly.com/>. [Accessed 07 11 2020].
- [68] P. Ferrari, “SSD: Single-Shot MultiBox Detector implementation in Keras,” *Github repository*, no. https://github.com/pierluigiferrari/ssd_keras, 2018.
- [69] “Amazon Web Services,” [Online]. Available: <https://aws.amazon.com/>. [Accessed 6 5 2020].
- [70] “Sagemaker - Amazon Web Services,” [Online]. Available: <https://aws.amazon.com/sagemaker/>. [Accessed 6 5 2020].
- [71] “Docker,” [Online]. Available: <https://www.docker.com/resources/what-container>. [Accessed 30 5 2020].
- [72] “Amazon Elastic Container Registry,” [Online]. Available: <https://aws.amazon.com/pt/ecr/>. [Accessed 30 05 2020].
- [73] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv*, 2018.
- [74] J. Redmon, “Darknet: Open Source Neural Networks in C,” 2016. [Online]. Available: <http://pjreddie.com/darknet/>. [Accessed 2 6 2020].
- [75] “Google Colab,” [Online]. Available: <https://colab.research.google.com/>. [Accessed 6 5 2020].
- [76] F. Giacosa, “Decay Law and Time Dilatation,” *Acta Physica Polonica B*, vol. 47, no. 9, p. 2135, 2016.
- [77] W. Abdulla, “Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow,” *Github repository*, 2017.
- [78] “Raspberry Pi,” [Online]. Available: <https://www.raspberrypi.org/>. [Accessed 27 08 2020].
- [79] “Jetson Nano Developer Kit,” [Online]. Available: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. [Accessed 27 08 2020].

[80] “Dell,” [Online]. Available: <https://www.dell.com/pt/empresas/p/poweredge-r240/pd>. [Accessed 29 08 2020].