

Universidad de Alcalá

Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Detection of abnormal passenger behaviors on ships, using RGBD
cameras

ESCUELA POLITECNICA
SUPERIOR

Author: Antonio Carlos Cob Parro

Advisor: Cristina Losada Gutiérrez

2020

UNIVERSIDAD DE ALCALÁ
ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

**Detection of abnormal passenger behaviors on ships, using
RGBD cameras**

Author: Antonio Carlos Cob Parro

Advisor: Cristina Losada Gutiérrez

Tribunal:

President: Manuel Mazo Quintas

1st Vocal: Roberto J. López Sastre

2nd Vocal: Cristina Losada Gutiérrez

Calification:

Date:

“El trabajo duro vence al talento natural”

Acknowledgements

Gracias a mis padres Antonio y Paloma, que de una manera incansable me han dado todo lo que tengo y han formado todo lo que soy.

Gracias a Andrea, por ser mi combustible, mi toma a tierra y la causante de una sonrisa perpetua.

Gracias a Cristina, mi tutora, que con su increíble paciencia ha conseguido que todo esto sea capaz. Ya que este trabajo es tanto suyo como mio.

Gracias a mis amigos Fernando y Lucas, mis hermanos, muestra viviente de que la sangre no hace la familia.

Gracias a Marta por darme la oportunidad de iniciarme en el mundo laboral con un proyecto tan increíble.

Gracias a todos por todo.

Resumen

El objetivo de este trabajo fin de Máster (TFM) es el diseño, implementación, y evaluación de un sistema inteligente de videovigilancia, que permita la detección, seguimiento y conteo de personas, así como la detección de estampidas, para grandes embarcaciones. El sistema desarrollado debe ser portable, y funcionar en tiempo real. Para ello se ha realizado un estudio de las tecnologías disponibles en sistemas embebidos, para elegir las que mejor se adecúan al objetivo del TFM. Se ha desarrollado un sistema de detección de personas basado en una MobileNet-SSD, complementado con un banco de filtros de Kalman para el seguimiento. Además, se ha incorporado un detector de estampidas basado en el análisis de la entropía del flujo óptico. Todo ello se ha implementado y evaluado en un dispositivo embebido que incluye una unidad VPU. Los resultados obtenidos han permitido validar la propuesta.

Key words: Detección de personas, Detección de estampidas, Redes neuronales, Aprendizaje máquina y Flujo óptico.

Abstract

The aim of this Final Master Thesis (TFM) is the design, implementation and evaluation of an intelligent video surveillance system that allows the detection, monitoring and counting of people, as well as the detection of stampedes, for large ships. The developed system must be portable and work in real time. To this end, a study has been carried out of the technologies available in embedded systems, in order to choose those that best suit the objective of the TFM. A people detection system based on a MobileNet-SSD has been developed, complemented by a Kalman filter bank for monitoring. In addition, a stampede detector based on optical flow entropy analysis has been incorporated. All this has been implemented and evaluated in an embedded device that includes a Vision Processing Unit (VPU) unit. The results obtained have allowed the validation of the proposal.

Key words: People detection, Stampedes detection, Neurons networks, Machine learning y Optical flow.

Contents

Resumen	ix
Abstract	xi
Contents	xiii
List of Figures	xvii
List of Tables	xxi
List of Acronyms	xxv
1 Introduction	1
1.1 Objective	2
1.2 Proposed solution	2
1.3 Memory structure	3
2 State of the art	5
2.1 People detection and counting	5
2.2 Anomalous behaviour recognition	8
2.3 Datasets	10
3 Commercial hardware solutions	15
3.1 Cameras	15
3.1.1 Intel D400 series	15
3.1.2 Microsoft Kinect and Kinect V2	17
3.1.3 Jetson cameras	18
3.1.4 Summary of camera characteristics	20
3.2 Embedded system	20
3.2.1 Intel Movidius Myriad X VPU	21
3.2.2 Up-Squared AI Vision Development Kit	21
3.2.3 Next Unit of Computing (NUC)	22

3.2.4	Nvidia Jetson AGX Xavier Developer Kit	22
3.2.5	EIC MS Vision 500	23
3.2.6	Summary of embedded system characteristics	23
3.3	Conclusions	23
4	Theoretical background	25
4.1	Neural networks	25
4.1.1	Artificial Neural Networks	26
4.1.2	Learning procedure and parameters	28
4.1.3	Convolutional neural networks	30
4.1.3.1	Layers in a CNN	31
4.1.3.2	Activation functions	33
4.1.3.3	CNN architectures	35
4.1.4	Solutions for object detection	35
4.2	OpenVino toolkit	40
4.3	Kalman Filter	41
4.4	Optical Flow	43
4.4.1	Lucas-Kanade method	45
4.4.2	Gunnar-Farneback method	46
5	Implementation	47
5.1	People detection	48
5.1.1	Image pre-processing	49
5.1.2	Network architecture	49
5.1.2.1	Feature extraction	50
5.1.2.2	Classification with the SSD	54
5.1.3	Data Post-Processing	55
5.2	People tracking and counting	56
5.2.1	Filter assignment	57
5.2.2	Object association	58
5.2.3	Kalman filter iteration	58
5.2.4	Filter elimination	59
5.3	Crowd stampede detection	59

6	Experimental results	63
6.1	Experimental setup	63
6.1.1	Datasets for people detection, tracking and counting	63
6.1.2	Datasets for stampede detection	64
6.1.3	Metrics	66
6.1.3.1	IOU and GIoU for people detection	67
6.1.3.2	Precision, accuracy and recall	68
6.2	Results	69
6.2.1	People detection	70
6.2.1.1	Qualitative evaluation	70
6.2.1.2	Quantitative evaluation	71
6.2.2	Stampede detection	72
6.2.2.1	Qualitative analysis	72
6.2.3	Quantitative evaluation	75
6.2.4	Computational Cost	76
7	Conclusion and future issues of research	79
7.1	Conclusions	79
7.2	Future issues	80
8	Budget	81
8.1	Hardware resources	81
8.2	Software resources	81
8.3	Human resources	82
8.4	Total cost budget	82
8.5	Contract expenses	82
8.6	Contract earning	83
8.7	Global Budget	83
9	Tools and resources	85
9.1	Hardware resources	85
9.2	Software resources	85
	Bibliography	87

List of Figures

1.1	General block diagram of the proposed system.	3
2.1	Example of people detection and tracking with kalman filter and HOG [1].	6
2.2	Difference between a RGB and depth image [2].	6
2.3	Feature extraction method designed by [3].	7
2.4	Sample of crowd density classes [4].	8
2.5	Convolutional neural network for crowd stampede detection [5].	9
2.6	Segmented figure in which segment information is processed for the study of abnormal behaviour.	9
2.7	Stampede detecting from head counting [6].	10
2.8	Example of activity map [7].	10
2.9	Environments provided by the UMN dataset.	11
2.10	RGB and depth scenes from EPFL-CORRIDOR and EPFL-LAB.	12
2.11	Hall of the Polytechnic of the University of Alcalá de Henares.	12
2.12	Some sample images for different actions included in the GBA2016 dataset.	13
2.13	Example of crowd behaviour of the GBA dataset.	13
2.14	Example of two activities carried out by the Weizman dataset	14
3.1	Intel Realsense Depth Camera D-435i [8].	16
3.2	Intel Depth Camera D415 [9].	16
3.3	Microsoft kinect [10].	17
3.4	Microsoft kinect v2[11].	18
3.5	STEEReoCAM [12].	19
3.6	TaraXL [13].	19
3.7	Intel Movidius Myriad X Vision Processing Unit [14].	21
3.8	Up-Squared [15].	22
3.9	Intel® NUC with the AI on PC Development Kit [16].	22
3.10	Jetson AGX Xavier Developer Kit [17].	22
3.11	EIC-MS-VISION-500 [18].	23

4.1	Comparison between the structure of a biological neuron and an artificial neuron [19].	27
4.2	Example of neural network structure with three layers [20].	27
4.3	Example of the convolution process with a 3x3 kernel.	31
4.4	Effect of downsampling in an output volumen.	32
4.5	Examples of the two different types of pooling [21].	32
4.6	Example of general diagram of a fully connected network [19].	33
4.7	Example of CNN architecture [19].	33
4.8	Rectified Lineal Unit function.	34
4.9	Sigmoid function.	34
4.10	Hyperbolic tangent function.	35
4.11	R-CNN structure [22].	36
4.12	Fast R-CNN structure [23].	37
4.13	Faster R-CNN structure [24].	37
4.14	You Only Look Once (YOLO) workflow [25].	38
4.15	YOLO network architecture [25].	38
4.16	Multiboxes generated by the SSD [26].	39
4.17	Single-Shot Detector (SSD) network architecture [26,27].	40
4.18	OpenVino structure [28].	40
4.19	Kalman structure.	42
4.20	Example of a kalman filter cycle [29].	43
4.21	Optical flow example [30].	44
4.22	Optical flow example [31].	45
4.23	Optical flow example [32].	46
5.1	General block diagram of the developed system.	47
5.2	System diagram where the people detection module is highlighted in red.	48
5.3	MobileNet-SSD architecture.	50
5.4	The basic architecture of Conv_Dw_PW [33].	50
5.5	Depthwise separable convolution [34].	51
5.6	Standar Convolution kernels [35].	51
5.7	Depthwise Convolution kernels [35].	52
5.8	Pointwise Convolution kernels [35].	52
5.9	Feature maps [26].	54
5.10	Example of the effect of the Non-Maximum Suppression filter [36].	55
5.11	Modules of the tracking system highlighted in red.	56
5.12	Counting and tracking system workflow.	57
5.13	General block diagram of the crowd stampede detector.	59

5.14	Example of image before and after the pre-processing stage.	59
5.15	Example of activity Map.	60
5.16	Comparison between the values of TOV (on the left) and Entropy (on the right) for a sample sequence with a Stampede.	61
5.17	Threshold selection.	61
6.1	ROI used for people detection in GBA2016 dataset.	64
6.2	Environments forming the EPFL dataset.	64
6.3	Sample images correspondent to each of the three scenarios in the UMN dataset.	65
6.4	GBA stampede sample frame.	66
6.5	IOU graphical representation [37].	67
6.6	Difference between precision and recall [38].	69
6.7	Results in two frames from the EPFL-corridor datasets.	70
6.8	Results in two frames from the EPFL-lab dataset.	70
6.9	People detection results in different images from the GBA2016 dataset.	71
6.10	Example of the system working in different stampede scenarios.	72
6.11	Comparative graphs of the Ground-truth (GT) with the stampede detection system.	73
6.12	Example of the system working in different stampede sequences in the GBA stampede dataset.	74
6.13	Comparative graphs of the GT with the stampede detection system.	75

List of Tables

- 3.1 Main features of the Intel D415 and D435 Red Green Blue and Depth (RGBD) cameras. 16
- 3.2 Main features of the Kinect. 17
- 3.3 Main features of the Kinect v2 camera. 18
- 3.4 Main features of the STEEReoCAM. 18
- 3.5 Main features of the TaraXL. 19
- 3.6 Summary of the main technical features of the analyzed cameras. 20
- 3.7 Summary of the main technical features of the analyzed embedded systems. 23

- 5.1 Convolution layers. 53
- 5.2 SSD parameters. 55

- 6.1 Summary of the distribution of videos in each of the three levels of difficulty for EPFL and GBA2016 datasets. 66
- 6.2 Experimental results for the people detection algorithm in the EPFL and GBA2016 datasets. 71
- 6.3 Metrics UMN. 75
- 6.4 Metrics GBA. 76
- 6.5 Computational cost (in ms) for the different available hardware platforms. 77

- 8.1 Price of the different devices used in the TFM. 81
- 8.2 Price of the different software used in the TFM. 82
- 8.3 Hourly rate of human resources TFM. 82
- 8.4 Total cost budget. 82
- 8.5 Contract expenses. 82
- 8.6 Contract earning. 83
- 8.7 Sum of work costs. 83

List of Acronyms

AI	Artificial Intelligenge.
ANN	Artificial neural networks.
API	Application Programming Interface.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
DDR4	Double Data Rate type four Synchronous Dynamic Random-Access Memory.
DNN	Deep Neural Networks.
DW	Depth-wise-layers.
eMMC	embedded MultiMediaCard.
EPFL	École polytechnique fédérale de Lausanne.
FN	True Negative.
FOV	Field of View.
FP	False Positive.
FPGA	Field-programmable gate array.
FPS	Frames per Second.
GB	Gigabyte.
GBA2016	GEINTRA Behaviour Analysis 2016.
GEINTRA	Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte.
GF	Gunnar-Farneback.
GIoU	General Intersection over Union.
GPIO	General Purpose Input/Output.
GPU	Graphical Processing Unit.
GT	Ground-truth.
HDL	Hardware Description Language.
HOG	Oriented Gradient Histogram.

HW	Hardware.
IoU	Intersection over Union.
IR	Intermediate Representation.
Ir	Infrared.
LK	Lukas-Kanade.
LPDDR4x	Low-Power Double Data Rate Synchronous Dynamic Random Access Memory.
ML	Maximum Likelihood.
MP	Megapixels.
NMS	Non-Maximum Suppression.
NUC	Next Unit of Computing.
OF	Optical Flow.
OpenVINO	Open Visual Inference and Neural network Optimization.
PC	PersonalComputer.
PCA	Principal Components Analysis.
PW	Point-wise-layers.
RAM	Random Access Memory.
RCNN	ResNet Convolutional Neural Network.
RGB	Red Green Blue.
RGBD	Red Green Blue and Depth.
ROI	Region of interest.
ROM	Read-Only Memory.
SDK	Software Development Kit.
SOC	System on Chip.
SSD	Single-Shot Detector.
SVM	Support Vector Machine.
SW	Software.
TFM	Final Master Thesis.
TN	True Negative.
ToF	Time of Flight.
TOV	Temporal Occupancy Variation.
TP	True Positive.

USB Universal Serial Bus.

VPU Vision Processing Unit.

YOLO You Only Look Once.

Chapter 1

Introduction

Nowadays it is very common to find video-surveillance systems anywhere. In general, these systems are centralized, which means that a set of sensors captures environment information and sends it to a central node. Furthermore, the received information are stored and supervised by a person. This can be a problem because humans do not have the capacity to pay attention continuously. In addition errors can happen after a long time of monitoring.

The incorporation of automatic surveillance processing systems [39–41] that, for example, issue an alert when a person appears or if there is a suspicious movement, make it possible to improve the accuracy and efficiency of video surveillance systems. These automatic systems used to be centralized, so they can have problems related to their reliability and scalability.

In this context, the main objective of this TFM is the design, development and evaluation of a decentralized system for automatic video-surveillance which detects anomalous behaviours that can be a risk for people (such as overcrowded areas, stampedes, people falling, etc.) as well as counting them [2, 42, 43] from Red Green Blue (RGB) or RGBD video sequences.

With the above, the objective of this work is to build a portable and decentralized system with the following characteristics:

- **Portable:** the system must be portable, so it allows an easy transportation and set up. Furthermore, this system must be functional on a embedded platform such as the Intel NUC [16] or the UP Squared Artificial Intelligence (AI) Vision Development Kit [15].
- **Multiple information:** the system must be able to work with different data types such as RGB, depth or RGBD. To do this, it is possible to use different sensors such as D-415 [9], D-435 [8] or Kinect [11].
- **Decentralized:** this system must not depend on any other for the processing of the information. So it must be able to receive the information from the sensor and process it, generating a response that can be sent to a central node.
- **Functional:** the system must be able to detect, track, count and detect anomalous situations such as stampedes. The processing of all these actions should be carried out in the embedded system.

To summarize, the developed system must meet two main requirements: the device has to be portable, so it can be settled in different positions in the area under surveillance and it has to be able to process all the video streaming in real time.

This work has been developed within the Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte (GEINTRA) research group [44], in the framework of the research project “PALAEMON A holistic passenger ship evacuation and rescue ecosystem” [45]. In this context, it is focused on the automatic control of people in large ships or navies, controlling the passengers security in case of an evacuation or rescue. This need arises as in this kind of transports, passengers can be randomly moving during the alarm moment. In this way it is possible to guaranty a full supervision of their conduct and send a constant flow information to the crew, being able to act in the most effective way.

1.1 Objective

As it has been stated before, the main aim of this TFM is to design and implement a system which allows both, people counting and tracking, as well as detection of anomalous behaviours (such as falls, overcrowds, stampedes, etc) of passengers on ships, from the access points on board ships. It is carried out using RGB or RGBD data, from cameras located in the environment, that must be processed in real time in an embedded system.

This global objective can be divided in several specific goals. They are detailed below.

- Search and study of different systems in the scientific literature, as well as commercial solutions for people detection, counting, tracking and behaviour analysis.
- Analysis and comparison of different embedded systems, with several technologies, for the subsequent selection of the one which better fits with the specifications, allowing thus, the image processing in real time.
- System design and development, that includes two different tasks:
 - Design and development of a robust and reliable algorithm for people detection, tracking and counting.
 - Design and development of the algorithm for behaviour analysis which includes: falling and stampedes detection.
- Implementation of the developed algorithms in the chosen embedded system, and code optimization for working in real-time.
- Exhaustive experimental evaluation of the proposed system, and proposal of possible improvements.

1.2 Proposed solution

This TFM proposes a solution that integrates different modules to design an autonomous and decentralized system, for people detection, tracking and counting, and stampede detection. Figure 1.1 shows a general block diagram of the proposed system. This design uses a smart camera model for data acquisition. These images are processed to extract the location of the people who are in each frame. The obtained information allows people tracking and counting, as well as determining the number of people entering and leaving the area under surveillance. Furthermore, if the number of people is over a predefined threshold, there is included a module for detecting stampedes.

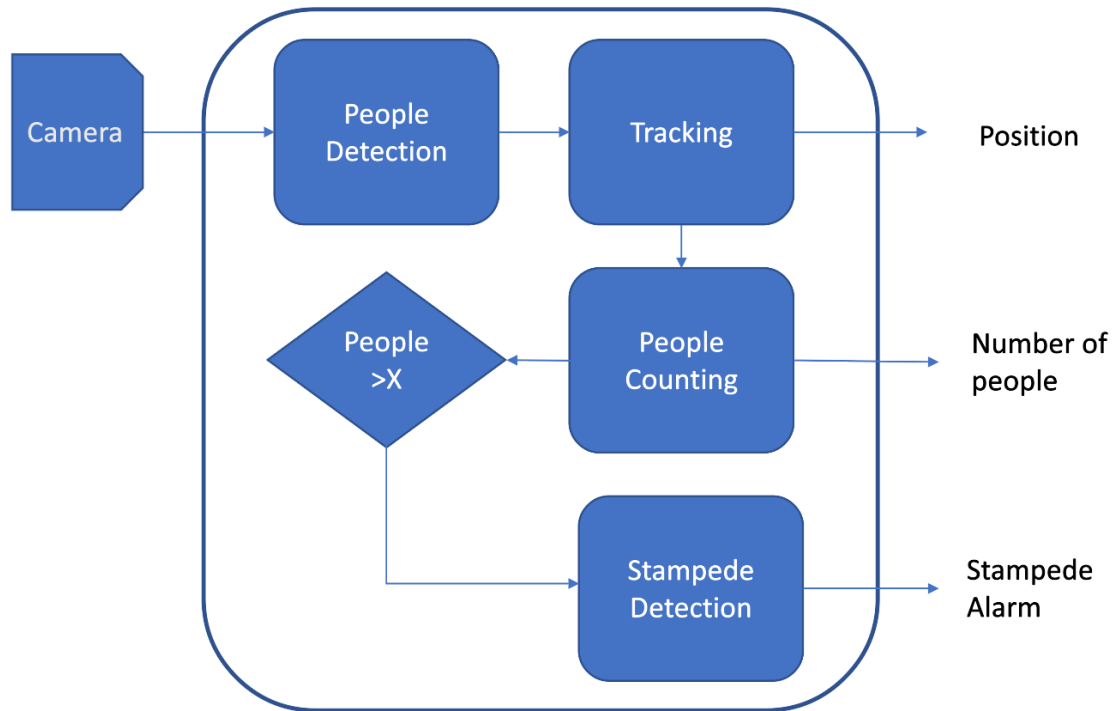


Figure 1.1: General block diagram of the proposed system.

The main modules are the ones in charge of detecting, tracking and detecting stampedes.

The workflow of the system starts in the detection module, which is in charge of detecting people in the environment through a MobileNet-SSD network. This module generates predictions of where people are in the frame. These predictions are passed to the tracking module, which uses a Kalman filter to predict where people are. Later, using this prediction, a count and representation of the people located by the system is made. In addition, this module allows the generation of a tracking of the predictions of the MobileNet-SSD network, making the system robust to overlapping.

The system also includes a stampede detector, which collects the counting information provided by the tracking module. When more than 6 people are detected in the system, this module starts working and predicts a possible stampede, generating an alert signal.

1.3 Memory structure

This work is structured in [6 chapters](#), which are set out below with a brief explanation of their contents:

- Chapter 1, the [TFM](#) introduction: where it is exposed the system features and the form to successfully address the project.
- Chapter 2, includes the State of the art: it is a summary of different theoretical solutions and devices which are compared to each other to select the best for the most optimal solution.
- Chapter 4 in which it is explained the theoretical framework of this [TFM](#). At the end of the chapter there is a conclusion explaining the implemented solution selected.

- Chapter 5, describes the implementation: in this chapter is exposed the different ways to get a functional system. It is explained the development to detect, track, count, detect anomalous behaviour and how it is moved to embedded system.
- Chapter 6 presents the main experimental results: in this section sets out in a clearly and orderly way the results achieved. In this chapters, it is also described the experimental setup.
- Chapter 7, Conclusion and future issues of investigation: is the last chapter of this TFM and it is evaluated the achieved obtained. Also, there is a part of possible improvements and how it can be used in future projects.

Chapter 2

State of the art

In this chapter there is presented the state of the art related to the main modules explained in the introduction. This chapter refers to methods and models that are capable of detecting and tracking people, as well as recognizing anomalous or dangerous behaviours. For this purpose a study of different works has been made to show possible solutions.

To organize the information, the chapter is divided into three sections. Firstly, the work related to detection and counting of people is presented. In this section both classical methods and methods employing artificial intelligence are analyzed. In addition, those jobs that use **RGB** or depth information are also analyzed.

Then, in the next section, some of the work is described in the context of recognizing abnormal behaviour. Where different methods for the study of crowd behaviour are shown.

And to finish the chapter, the main data sets of the literature that allow to evaluate and test the **TFM** in a more adequate way to the requirements imposed are described.

2.1 People detection and counting

Detection and counting is defined as the ability of a system to indicate the position of an object and the number of such objects. This can be extrapolated to anything that is identifiable with a camera. For this work, as discussed in the introduction, a detection and counting of people is desired. The literature related to people detection is very extensive [46, 47], and the different proposals can be classified depending on the type of information used (**RGB**, **RGBD** or depth), and the type of approach. Within the different methods mentioned, they can be distinguished in two large groups: those that are based on classical methods, which make use of mathematical resources for the detection of people [1, 48–56] and modern approaches, based on machine learning [3, 57–65].

Most of the classical methods use **RGB** images. This means that the cameras used by the systems, acquire information in the visible spectrum. Furthermore, these proposals include two different steps: first, they extract features from the image, that then are classified to detect if there are people in the scene.

Within the classical methods, one of the most popular is the one proposed by Dalal and Triggs [48], that obtains the image features using Oriented Gradient Histogram (HOG), and then include a Support Vector Machine (SVM) classifier. **SVM** is a supervised learning algorithm [66] that can be used for binary classification or regression, being very popular in applications such as: extraction of natural

language characteristics [67], speech recognition [68], image recognition based on feature extraction [69], etc.

Another type of widely used classical technique for classification is Principal Components Analysis (PCA). It can be defined as a technique used to describe a data set in terms of new uncorrelated variables ("components"). The components are ordered by the amount of original variance they describe, so the technique is useful for reducing the dimensionality of a data set. As it happens with SVM, this technique is used with others to increase its robustness and accuracy, this happens with [49] that implements a PCA with HOG, for the robust detection of people and the detection of their heads and shoulders.

There are also some proposals that address both, people detection and tracking using different techniques. In this context, the proposal in [1] propose a method that combine a Kalman filter and HOG features, whereas [50] describes an approach based on image segmentation and a Kalman filter. Figure 2.1 shows an example of the results obtained in [1].

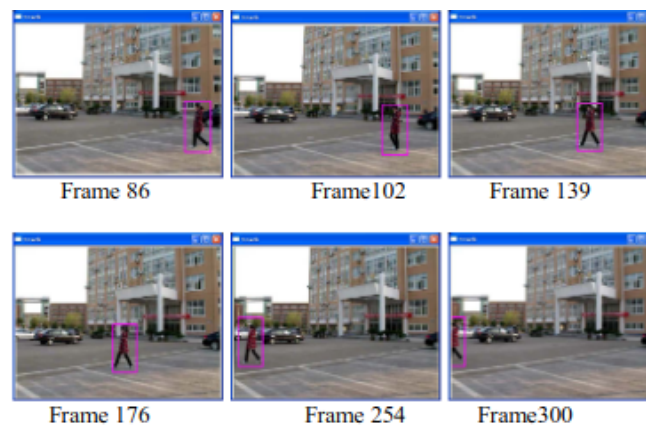
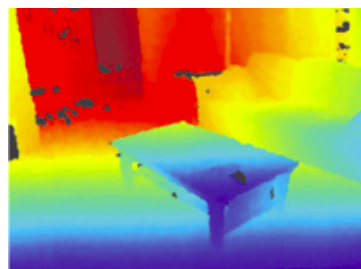


Figure 2.1: Example of people detection and tracking with kalman filter and HOG [1].

In recent years, there have appeared RGBD [8, 10, 70] cameras that provide both, RGB images and depth (distance from each 3D point to the camera) information. This information can be obtained using different technologies, such as stereo vision, structured light or Time of Flight (ToF). Figure 2.2 shows an example of an RGB image and a depth image. As mentioned, in the RGB image, the visible color spectrum is represented. While in Depth, a colour gradient is represented, associating distances to the colors. Besides, the appearance of this kind of sensors has caused the development of numerous works that use this information for people detection [51, 53–56]. Some of these works use HOG features obtained from RGBD data [52, 54, 55], whereas other approaches are based on PCA [53]. Furthermore, there are also works that use only depth information for privacy preserving [56].



(a) RGB image.



(b) Depth image.

Figure 2.2: Difference between a RGB and depth image [2].

The use of **RGBD** data allows improving the robustness of the proposals. However, this technology has some limitations, since the maximum depth is limited, and they can have problems in outdoor environments due to solar radiation, so most of the systems that use this kind of data are used only in indoor environments.

These methods are often used as complementary systems to **RGB**, generating much more robust and flexible systems. The biggest limitation that this type of images have is the little distance that they are able to cover. That is why most of these systems are used indoors.

In recent years, the improvements in the capacity of hardware systems and the availability of large dataset has led to an increase in the numbers of proposals based on deep learning. This does not mean that deep learning based approaches have displaced the classical methods, but that they started to be used as an alternative or as a complement to them.

As with classical methods, deep learning based ones can also work with either **RGB** or **RGBD** information. In this context, there are numerous works that use Convolutional Neural Network (CNN) for objects and people detection and classification. This type of systems are characterized by the extraction of characteristics from the images in the first layers, and the subsequent analysis of them in the last layers. This process is carried out by using convolutions. When talking about detection using **CNN**, it is possible to find works such as [64], which is able to detect people in real time. This type of systems are not only used to detect people, but also allows the interpretation of actions or emotions as is the case of [63]. Another point of view within **CNNs** is the design of specialized networks for detecting people in certain types of actions such as the jumping of people into positions with protrusions [3], as shown in the example of figure 2.3.

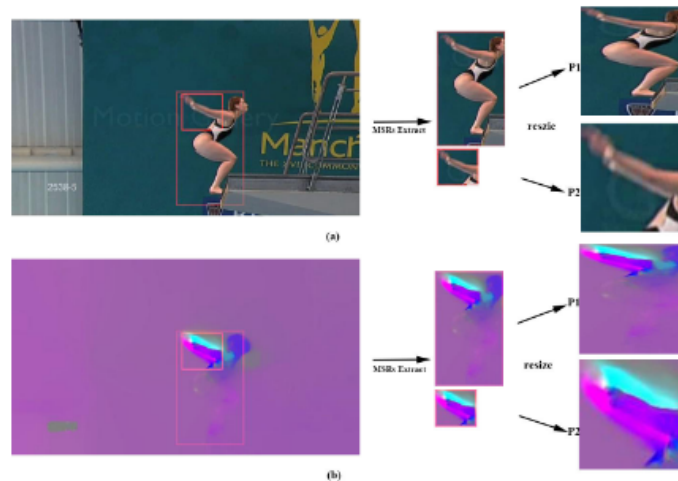


Figure 2.3: Feature extraction method designed by [3].

ResNet Convolutional Neural Network (RCNN) [57] and Faster RCNN [58, 59] have also been proposed for people detection with **RGB** images.

As mentioned above, deep learning based approaches can also work with **RGBD** data or depth images. In this field it is possible to find works such as [60] which is used to generate a robust people detection system using a depth-oriented **CNN** and a Kalman filter. There can also be found other approaches such as the one described in [61] or [62], which apart from detecting people, are able to extract their pose and orientation. Whereas other works detect people with a high accuracy using only depth information [65].

The decision about which method to use, whether a classic or a modern one is complicated, and it depends on the available hardware characteristics system to be mounted and many other factors. For this reason, it is necessary to carry out a previous study of the design of the model and the environment in which the developed systems going to work. For this work, it is desirable a system that is able to detect different types of people in different locations, due to its portability feature. Another important point to take into account is the available hardware resources. In the case of this work, as explained in section 3, it is going to be used a portable mini PersonalComputer (PC) with an specialized VPU for image processing. That is why for this TFM it has been decided to use a CNN based approach.

2.2 Anomalous behaviour recognition

In this section, there is presented a study of the state of the art regarding anomalous behaviour and how to identify them using computer vision.

Anomalous behaviour is defined as that behaviour carried out by an individual or group of individuals that is out of the ordinary. As defined, there can be a great number of anomalous behaviours, such as: falls, jumps, dry stops, stampedes, etc. These behaviours are closely related to the location where they occur, as what may be serious or dangerous in one place may be normal in another. For this reason it is necessary to limit the type of behaviour to be analyzed. Due to the TFM issue in which a video surveillance system is designed for large vessels, the anomalous behaviour that is studied is focused on crowd stampedes. This is useful to warn the personnel in charge of the security of a ship that in some section of the ship some kind of stampede is taking place.

Since the concept of a stampede is not understood if there is not a minimum number of people, there are also analyzed the proposals for detecting crowds in different scenarios, for low and moderate people density. Some examples of the different crowd density situations are shown in figure 2.4

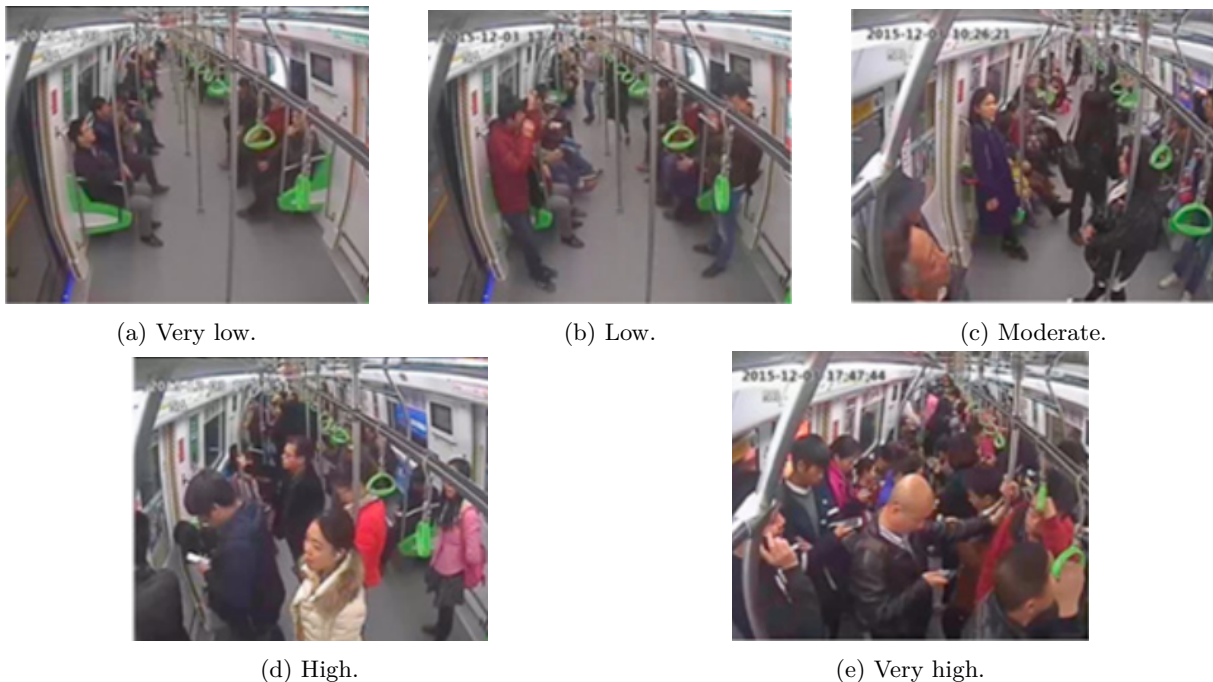


Figure 2.4: Sample of crowd density classes [4].

Regarding stampede detection, there is a diversity of methods to detect both crowds and anomalous events. Nowadays, most of the proposals are based on the use of neural networks. In the literature, there are several works that elaborate personalized networks for detecting stampedes [5], whereas other approaches use conventional networks [71], [72], [73]. Figure 2.5 shows an example of the neural network proposed in [5].

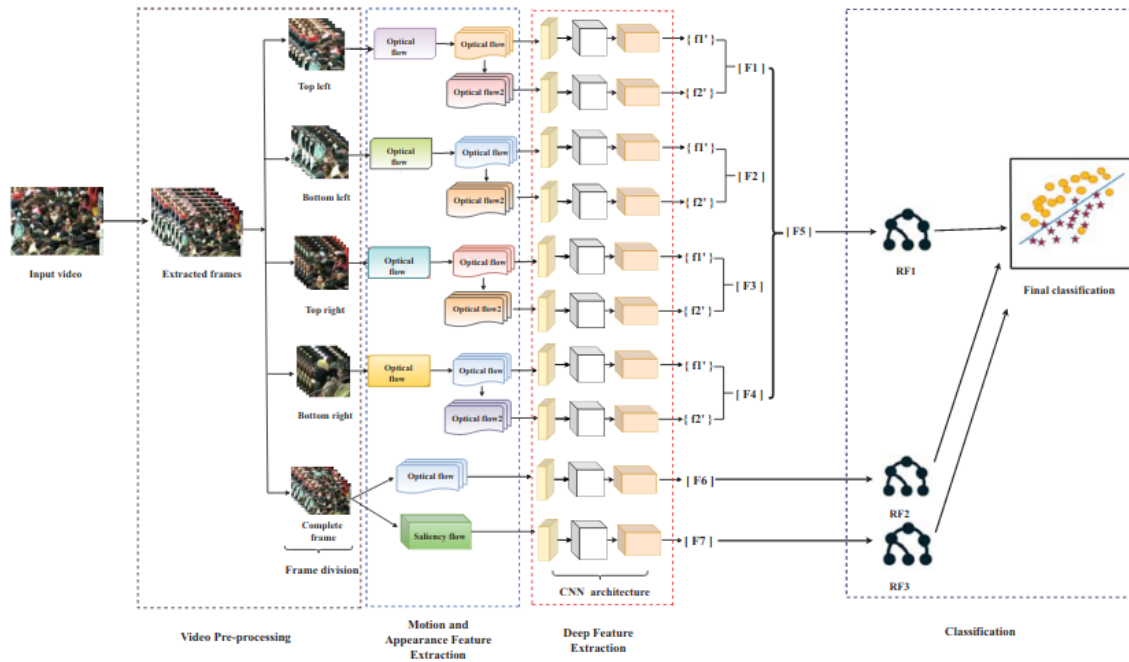


Figure 2.5: Convolutional neural network for crowd stampede detection [5].

There are also methods for stampedes and crowds detection that do not implement neural networks. This is the case of [74], which uses *SVM* and *HOG* to determine the density of a crowd, and if a space is very dense, it warns of a possible danger of stampedes. There are also cases such as [75], which are based on image segmentation and study characteristics such as: Crowd collectiveness, crowd conflict, mean motion, etc. To predict possible stampedes. An example is shown in figure 2.6.



Figure 2.6: Segmented figure in which segment information is processed for the study of abnormal behaviour.

Another interesting method is that of the work [6], which performed a type of head-count algorithm and with it was able to predict future stampedes. Figure 2.7 shows graphically how the algorithm [6] is able to detect crowds of people in a crowd through head detection.

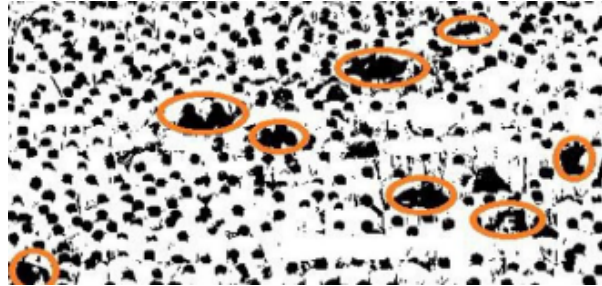


Figure 2.7: Stampede detecting from head counting [6].

Other methods are based on the use of optical flow for crowd analysis, in which the variations of pixels between frames are studied and this change is analyzed, being able to detect sudden movements, transit of people or stampedes, as is the case of [7]. Figure 2.8 shows a representation of the optical flow through a representation of an activity map, in which the variations in the scenes are represented as a change in the entropic level.



Figure 2.8: Example of activity map [7].

After analyzing different approaches, it was concluded that due to the hardware limitations of a portable system and the need for medium-density crowding analysis, the approaches based on the analysis of optical flow are the more adequate for the objectives of this TFM, since these approaches are able to obtain results similar to the other works that use neural networks with a lower computational cost.

2.3 Datasets

A dataset can be defined as a set of ordered and classified data. This data can be images, numerical data, sounds, etc. Datasets are used in the field of AI for training and testing neural networks. The parts corresponding to the training and the testing must be different, so the bigger and more extensive the dataset is, the better the training and testing parts will be. The datasets shown below have been chosen because they are the most commonly used for detecting people and stampedes. They also offer environments that could be similar to the conditions found on passenger ships.

Nowadays there are a large amount of available datasets, some of them are briefly explained below.

- **UMN Dataset**

It is a public dataset provided by the University of Minnesota [76] that consists of a set of videos with normal and abnormal situations. The videos contain different scenes, both in indoor and outdoor environments. The videos have the characteristic of always starting with a situation of normality followed by a panic situation in which a stampede occurs. The dataset consists of videos with images in gray scale and RGB. Figure 2.9 shows the three scenarios of which the dataset is composed: being 2 outdoor (a field 2.9a and a square 2.9b) and one indoor (university hall 2.9c).



(a) Outdoor with RGB image in green field.



(b) Outdoor with RGB image in square.



(c) Indoor in gray scale.

Figure 2.9: Environments provided by the UMN dataset.

- **EPFL RGB-D pedestrian dataset**

The École polytechnique fédérale de Lausanne (EPFL) RGB-D Pedestrian dataset [77] consists of over 13000 RGBD images, acquired using a Kinect camera. People instances are annotated, including both: fully exposed and highly occluded people, because the authors manually labeled the entire dataset, generating bounding boxes for each person, and adding an occlusion percentage value.

The dataset includes two subsets recorded in two different scenarios: a laboratory (EPFL-LAB) and a corridor (EPFL-CORRIDOR). EPFL-LAB includes frames with up to 4 people facing the camera, whereas EPFL-CORRIDOR contains scenes with up to 8 people.

Some sample images from the EPFL-CORRIDOR and EPFL-LAB scenes are shown in figure 2.10. Figures 2.10a, 2.10c show the RGB image acquired by the system, while 2.10d and 2.10b show the image in depth.

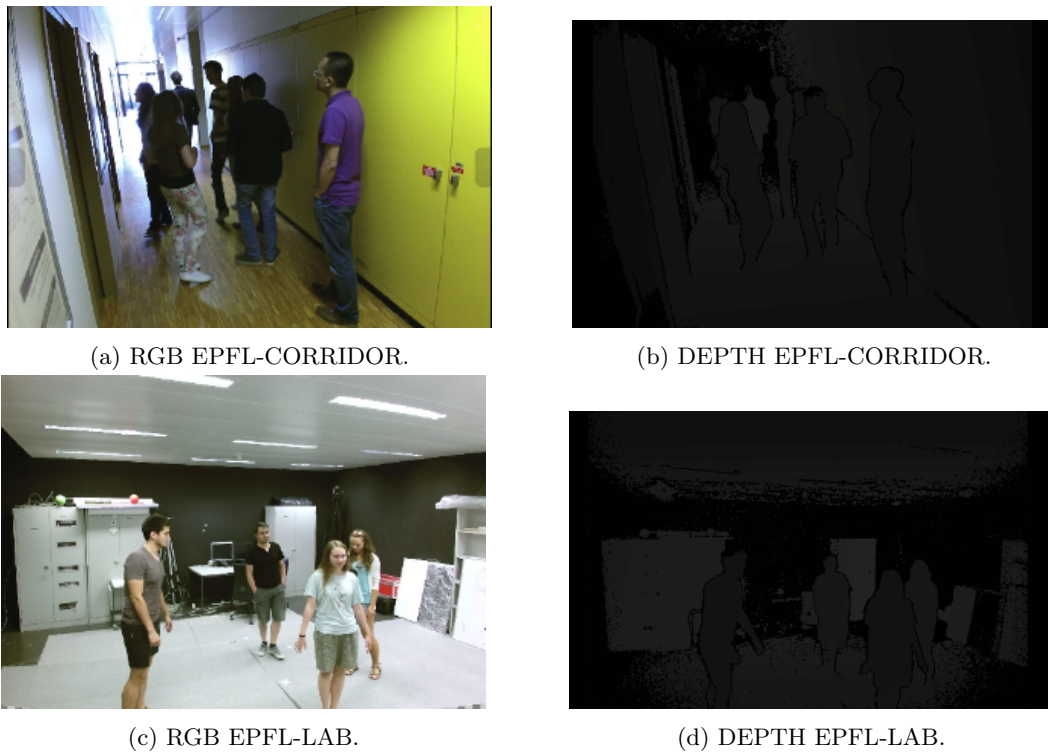


Figure 2.10: RGB and depth scenes from EPFL-CORRIDOR and EPFL-LAB.

- **GBA2016**

GEINTRA Behaviour Analysis 2016 (GBA2016) dataset [78] comprises a set of videos recorded on the Polytechnic school of the University of Alcalá, using a stationary camera FullHD 2.7K, the GoPro HERO4 [79], with a resolution of 1920x1080 pixels. The recording is produced at 60 Frames per Second (FPS) and has an ultra-wide field of view. Figure 2.11 shows the area in which this dataset has been recorded.

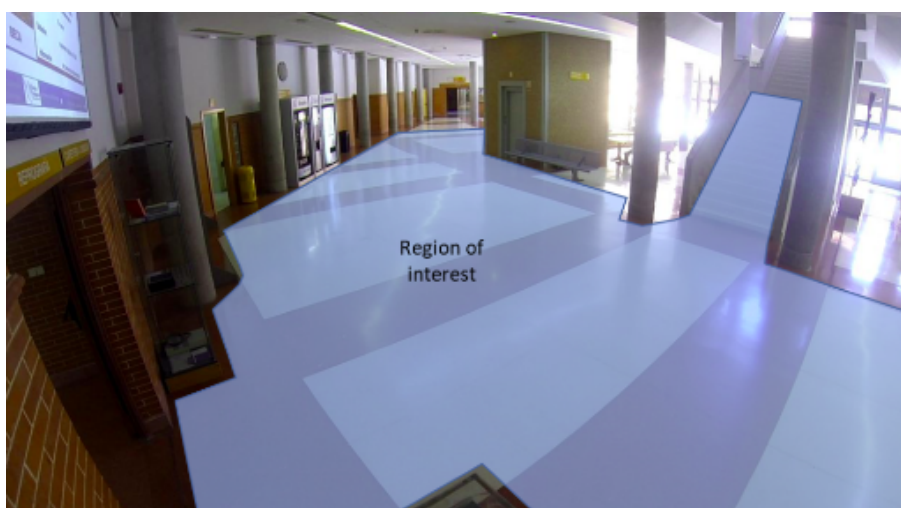


Figure 2.11: Hall of the Polytechnic of the University of Alcalá de Henares.

GBA includes a total of 40 videos where different types of scenes such as walking, sitting, falling, ascending stairs, etc. are presented. In addition, the dataset includes a set of 12 videos showing

group behaviour, such as stampedes, static groups, walking groups, etc. Figure 2.12 shows different individual actions such as sitting 2.12a, falling 2.12b and climbing stairs 2.12c.



(a) Sitting.



(b) Climbing stairs.



(c) Falling.

Figure 2.12: Some sample images for different actions included in the GBA2016 dataset.

This dataset also includes a subset for crowd stampede detection, named GBA stampede. Figure 2.13 shows an example of this subset.



Figure 2.13: Example of crowd behaviour of the GBA dataset.

- **WEIZMANN**

WEIZMANN dataset [80] includes videos of 9 people performing 10 different actions: walking, running, jumping, handshaking, jumping in a place, jumping jack, etc. The videos are recorded using a static camera with a resolution of 180x114 pixels. Figure 2.14 shows some sample images extracted from the WEIZMANN dataset,



Figure 2.14: Example of two activities carried out by the Weizman dataset .

Chapter 3

Commercial hardware solutions

Usually, the basic parts corresponding to a video surveillance system are the camera and the processing node. In this section there are described and evaluated several commercial cameras and embedded systems that are capable of satisfying the minimum requirements explained in the introduction of this [TFM](#). In this section, the different devices that have been proposed in the design phase are presented along with their technical characteristics. Furthermore, at the end of the chapter the devices chosen in this work are indicated.

3.1 Cameras

The camera is the device that captures the information from the environment and sends it to the node to be processed, for this reason, the choice of this device is so important for this work. Nowadays, there are a large number of different camera models with distinct features and prices.

The choice has been based on the study of features offered by the camera such as: Field of View (FOV), FPS, resolution and the different data provided by the camera [RGB](#), [RGBD](#) or depth. Furthermore, a compromise between price and quality has been searched.

The main characteristics of the analysed cameras are detailed below. Additionally, a table summarizing these technical characteristics [table 3.6](#) is included at the end of this section.

3.1.1 Intel D400 series

The Intel D400 series includes several cameras designed by Intel for easy setup and portability. The D400 series models are characterized by high resolution depth and active Infrared (Ir) stereo with a wide field of view. This type of cameras are characterized by high precision of movement both indoors and outdoors. In addition, these models allows developing a large number of vision projects.

- **Intel D-435i** [\[8\]](#): the Intel RealSense Depth Camera D435 includes wide field of view depth and [RGB](#) sensors. This camera is versatile and it can be used for different applications such as robotics, computer vision or virtual reality. The design, size and form of this camera (shown in [figure 3.1](#)) allows for easy and fast installation and portability. In addition, the Intel D-435i camera is powered by USB. It also comes with the D435 Cross-platform Software Development Kit (SDK) 2.0, that allows the use of the camera in a much more versatile way, allowing the use of more than one

programming language, wraps, as well as many tools and a list of examples to facilitate their learning and use. The technical specifications of this [RGBD](#) sensor are summarized in table 3.1.

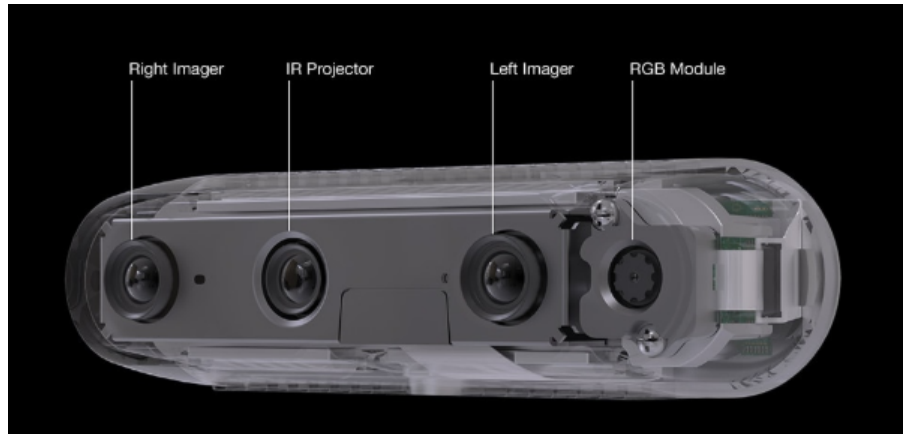


Figure 3.1: Intel RealSense Depth Camera D-435i [8].

- **Depth Camera D415** [9]: the D-415 camera (figure 3.2) is a previous version of the D-435 [8]. This model has fewer features than D-435 but it is still a very versatile model. Due to the versatility of this device, it is used in many different fields such as: hardware prototyping, software development or education. This camera shares many of the features of the D435 version. Its design, size and form of D-415 allows for easy and fast installation and portability, and it is also powered by Universal Serial Bus (USB). Moreover, the D415 camera comes with D415 Cross-platform [SDK](#) 1.0. The technical specifications of this [RGBD](#) sensor are summarized in table 3.1.

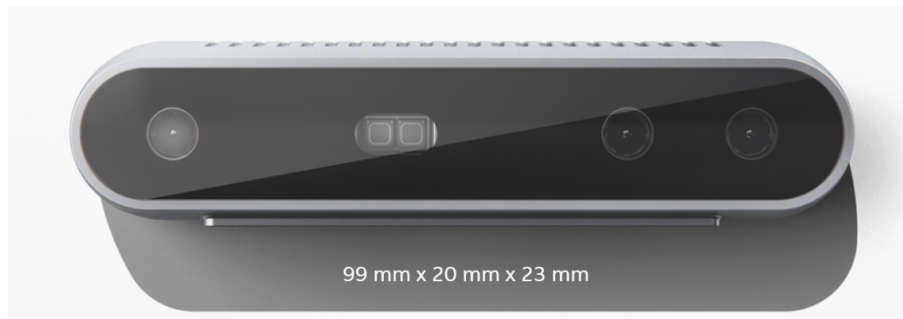


Figure 3.2: Intel Depth Camera D415 [9].

Feature		Depth	RGB
Depth technology		Active Ir Stereo	-
Resolution (pixels)		Up to 1280 x 720	1920 x 1080
Frame rate (FPS)		Up to 90	30 FPS
Connectors		USB-C 3.1 Gen 1	
FOV (H x V x D)	D415	$65^{\circ} \times 40^{\circ}$	$69^{\circ} \times 42^{\circ}$
	D435	$87^{\circ} \times 58^{\circ}$	
Size (mm)	D415	99 x 20 x 23	
	D435	90 x 25 x 25	

Table 3.1: Main features of the Intel D415 and D435 [RGBD](#) cameras.

3.1.2 Microsoft Kinect and Kinect V2

Kinect [10] and Kinect V2 [70] belong to a commercial line started by Microsoft in 2010 with the launch of the first Kinect as a peripheral for the Xbox 360. Besides, the Kinect V2 was released in 2014, being a updated model with a new depth technology and characteristics. Both devices were designed to play video-games, however, their characteristics and price have made them widely used for computer vision due to their quality and price [81]. Kinect devices include a set of sensors (RGB camera, Ir sensor and microphones) that allow obtaining RGBD and audio data. This opens a wide range of opportunities in the scientific world. However, the first version of Kinect was discontinued in 2016, and the V2 in 2018.

- **Kinect:** the Kinect RGBD camera (figure 3.3) is a Microsoft device. It was released on November 2011 as a peripheral for the Xbox 360, featuring an RGB camera, an Ir stereo sensor, a multi-array microphone and a custom processor running the patented software, which provides 3D full-body motion capture, facial recognition and voice recognition capabilities. The Kinect array of microphones allows the Xbox 360 to perform sound source location and ambient noise suppression, enabling participation in Xbox Live chat without the use of a headset. The technical specifications of this RGBD sensor are summarized in table 3.2

Feature	Depth	RGB
Depth technology	Ir Structured light	-
FOV (H x V)	$57^\circ \times 43^\circ$	$57^\circ \times 43^\circ$
Resolution (pixels)	640 x 480	640 x 480
Frame rate (FPS)	30 FPS	30 FPS
Size (mm)	$279 \times 76 \times 76$ mm	
Connectors	USB standar 2.0	

Table 3.2: Main features of the Kinect.



Figure 3.3: Microsoft kinect [10].

- **Kinect v2:** Kinect v2 (figure 3.4) is a Microsoft device [70], it was released on November 2013 as a peripheral for the Xbox One. Apart from that, this camera can also be used for situations outside the scope of video games [82, 83]. The Kinect v2 camera is based on ToF technology, and it also include an RGB camera.

This camera is able to work without visible light, since it includes an active light source. It also has a high quality/price ratio. However, Microsoft discontinued the production of this camera model in 2018. The main characteristics of the Kinect V2 are summarized in table 3.3

Feature	Depth	RGB
Depth technology	ToF (Time of flight)	-
FOV (H x V)	$70^\circ \times 60^\circ$	$84^\circ \times 53^\circ$
Resolution (pixels)	Up to 512 x 424	1920 x 1080
Frame rate (FPS)	30 FPS	30 FPS
Size (mm)	249 × 66 × 67 mm	
Connectors	USB standar 3.0	

Table 3.3: Main features of the Kinect v2 camera.



Figure 3.4: Microsoft kinect v2[11].

3.1.3 Jetson cameras

The Jetson cameras are a type of cameras use for Nvidia Jetson boards. This graphic boards are oriented to image processing. In addition this family has models with developer features that allow to use the board like a mini computer. These types of camera are only compatible with the NVidia Jetson boards. Below, there are described some of the available Jetson cameras:

- **STEEReoCAM**: STEEReoCAM [12] is a camera model oriented to the Nvidia Jetson Nano (figure 3.5), AGX Xavier and TX2 boards. It is characterized by being a 3D MIPI Stereo camera with a resolution of 2 Megapixels (MP), this allows it to have a good precision and depth range.

Feature	Depth	RGB
Depth technology	Stereo	-
FOV (H x V)	$54^\circ \times 49.5^\circ$	$54^\circ \times 49.5^\circ$
Resolution (pixels)	1600 x 1300	1600 x 1300
Frame rate (FPS)	30 FPS	30 FPS
Size (mm)	143 × 27.5 × 29 mm	
Connectors	Micro Coaxial	

Table 3.4: Main features of the STEEReoCAM.



Figure 3.5: STEEReoCAM [12].

- **TaraXL:** TaraXL [13] is a model oriented and optimized to be used by NVidia boards such as the jetson AGX, Xavier or Tx2 (figure 3.6). It is a stereo camera powered and communicated via USB. The TaraXL is a development kit which comes with an SDK. Its main characteristics are summarized in table 3.5.

Feature	Depth	RGB
Depth technology	ToF	-
FOV (H x V)	$54^\circ \times 36^\circ$	$54^\circ \times 36^\circ$
Resolution (pixels)	752 x 480	752 x 480
Frame rate (FPS)	60 FPS	60 FPS
Size (mm)	143 × 27.5 × 29 mm	
Connectors	USB 3.0 A to Micro-B Type	

Table 3.5: Main features of the TaraXL.



Figure 3.6: TaraXL [13].

3.1.4 Summary of camera characteristics

Below, the tabla 3.6 summarize the main characteristics of the analyzed cameras.

Features		Intel D400		Microsoft Kinect		NVIDIA Jetson	
		D-435	D-415	Kinect	Kinect v2	SteereoCam	TaraXL
Depth	Technology	Active Ir		Structured light	ToF	Stereo	
	FOV (H,V)	$87^\circ \times 58^\circ$	$65^\circ \times 40^\circ$	$57^\circ \times 43^\circ$	$70^\circ \times 60^\circ$	$54^\circ \times 49^\circ$	$54^\circ \times 36^\circ$
	Resolution (pixels)	1280 × 720		320 × 240	512 × 424	1600 × 1300	752 × 480
	Frame rate (FPS)	90		30		30	60
RGB	FOV (H,V)	$69.4^\circ \times 42.5^\circ$		$57^\circ \times 43^\circ$	$84^\circ \times 53^\circ$	$54^\circ \times 49^\circ$	$54^\circ \times 36^\circ$
	Resolution (pixels)	1920 × 1080		620 × 480	1920 × 1080	1600 × 1300	752 × 480
	Frame rate (FPS)	30		30		30	

Table 3.6: Summary of the main technical features of the analyzed cameras.

3.2 Embedded system

One of the objectives set for this work is the development of a portable and decentralized system. To do so, it is important to use some kind of embedded system from those available in the market. This system must be able to process the information provided by the and generate a result that is easily interpreted

For the selection of the embedded system, a set of commercial systems with different characteristics and technologies are proposed. This set of devices is studied both as a whole and individually. Thus, in the last section of this chapter, a conclusion is established explaining why the choice of the device was made.

The following is an explanation of the different types of technologies that make up the various embedded systems. A short explanation has been made to explain the basic fundamentals.

- **System on Chip (SOC):** is a full solution that combines hardware and software schemes under a single device. These devices are composed of a Central Processing Unit (CPU), input and output ports, Random Access Memory (RAM) memory and Read-Only Memory (ROM) memory. It can also include an Field-programmable gate array (FPGA) area for hardware designs.
- **FPGA:** is a programmable device composed of logic blocks whose interconnection and functionality can be reconfigured according to the design needs. Several Hardware Description Language (HDL) (such as VHDL or Verilog) are usually used for programming these devices.
- **Microprocessor:** is the central integrated circuit of a computer system in which logic and arithmetic operations (calculations) are performed to enable the execution of programs, from the operating system to the application software.

Besides the hardware unit to process the information, the following issues must be considered to select the more suitable embedded approach:

- **Size:** the searching has been focused on embedded system models with small size and easy installation.
- **Technical characteristics:** For the processing of artificial vision and artificial intelligence it is necessary to have devices that are capable of running programs with high computer load. For this purpose, devices with **RAM** and **ROM** memory large enough to support the system are needed. Graphics cards or specialized devices in the video process such as **VPU**s. And to finish a microprocessor that is able to manage the whole system.
- **Compatibility and communication:** the system must have communication channels to be connected with other devices and systems. Also compatibility features should be considered.
- **Image Processing options:** to compute images is a complex issue in any processing system. Thus, the option of specific and adapted image processing tools for an embedded solution must be explored in order to reduce the complexity and accelerate the codification stage.

Below, there are described different commercial solutions that have been analysed to be used for the purpose of this **TFM**. In this section the technical characteristics of the devices will be presented in an orderly way. In addition, at the end of the section there is a table with a summary of all these features to facilitate their understanding.

3.2.1 Intel Movidius Myriad X VPU

Is a type of device called a "Vision process unit" **VPU** [14](figure 3.7). This device has a specialized hardware design in the use of deep neural networks. The internal structure of the Myriad includes 16 powerful **SHAVE** [84] cores and a high throughput intelligent memory fabric that makes the Myriad X a leader in the deep neural network industry for computer vision applications. Myriad X is the third generation of this type of device specialized in deep neural network processing, being today the most powerful on the market.

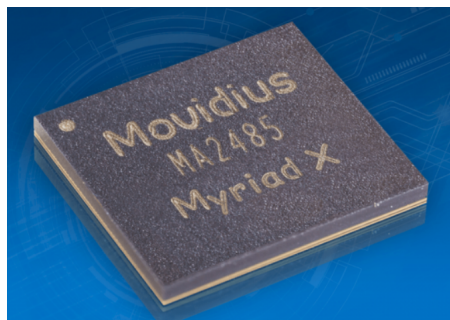


Figure 3.7: Intel Movidius Myriad X Vision Processing Unit [14].

3.2.2 Up-Squared AI Vision Development Kit

Is a specialized kit for the development of machine vision and **AI** projects. At the hardware level (figure 3.8), the Up-Squared is characterized by: an Intel Atom x7-E3950 microprocessor. An 8 Gigabyte (GB) **RAM** memory of the Double Data Rate type four Synchronous Dynamic Random-Access Memory (DDR4) type. A 64GB embedded MultiMediaCard (eMMC) **ROM** and a deep learning module of Intel Movidius Myriad X [14] that includes a specialized reference in **CNN**. In addition to all these hardware features the system comes with the OpenVino framework installed and an Ubuntu 16.04 operating system.



Figure 3.8: Up-Squared [15].

3.2.3 NUC

This AI Development Kit (figure 3.9) [16] is designed to generate an environment where systems containing AIs can be developed. The hardware features of this device that allow the development of Maximum Likelihood (ML)s are: a seventh generation Intel Core I7 microprocessor and an integrated Iris Plus graphics. It also include a DDR4 RAM of 16 GB and a ROM of 512 GB. Finally, the development kit includes an intel Movidius Myriad X VPU. Finally this device comes with the Open Visual Inference and Neural network Optimization (OpenVINO) framework preinstalled.



Figure 3.9: Intel® NUC with the AI on PC Development Kit [16].

3.2.4 Nvidia Jetson AGX Xavier Developer Kit

Is a developed kit by the Nvidia (figure 3.10) [17] brand, designed for the development of applications and systems with artificial intelligence. It is characterized by a low power consumption(<30W) which allows it to be a portable and easy to install system. In addition, within its hardware characteristics, the system includes: a 64 bits and 8 cores ARM CPU, a 512 cores Volta Graphical Processing Unit (GPU), a 64 GB Low-Power Double Data Rate Synchronous Dynamic Random Access Memory (LPDDR4x) RAM and a 32 GB eMMC ROM. This system is specially designed for the development of robot drones and other types of autonomous machines.



Figure 3.10: Jetson AGX Xavier Developer Kit [17].

3.2.5 EIC MS Vision 500

Is a SOC [18] device (figure 3.11), which includes peripherals such as a camera, a speaker, a microphone, etc. Apart from the peripherals, the system is characterized by the use of a Qualcomm QCS603 microprocessor, a 4GB LPDDR4x RAM and a 16GB eMMC ROM. This device requires an internet connection for working, since the information is processed in a remote server.



Figure 3.11: EIC-MS-VISION-500 [18].

3.2.6 Summary of embedded system characteristics

To finish this section, table 3.7 summarizes the main features of the previously described embedded systems.

Technical features of embedded systems			
Device	Core	RAM memory	ROM memory
EIC MS Vision 500	Qualcomm QCS603	LPDDR4x (4GB)	eMMC (16GB)
Up-Squared AI Vision X Developer	Intel Atom x7-E3950	DDR4 (8 GB)	eMMC (64 GB)
Nvidia jetson xavier	GPU Volta, 512 cores, CPU 64 bits ARM, 8 cores	LPDDR4x (64 GB)	eMMC(32GB)
Intel® NUC Mini (AI Dev Kit)	Intel Core i7, Iris Plus Graphics	DDR4 (16 GB)	SSD (512 GB)

Table 3.7: Summary of the main technical features of the analyzed embedded systems.

3.3 Conclusions

There has been carried out an intensive study comparing different options that are available on the market, analysing their main characteristics and studying the balance between price and characteristics. This exhaustive analysis has allowed choosing the most suitable hardware for the objectives of this TFM: the RGBD camera Intel D-435 and two different embedded systems (the Intel NUC and the Up-Squared).

The chosen camera (Intel D-435) offers plenty of flexibility, allowing it to be used with different embedded systems. In addition, it can provide both RGB and depth data that can be used in the future to improve the system.

The next point to be discussed is the choice of the embedded system. In this case, and unlike the cameras, there have been selected two different devices:

The first one is the Up-Squared [AI Vision X Developer](#), a [SOC](#) which has a great capacity of image processing. In addition, it has the advantage of providing several General Purpose Input/Output (GPIO)s for communication. Furthermore, the small size and low power consumption of the device make it easy to install in real environments.

The second device is the Intel [NUC](#). It is a mini [PC](#), oriented to image processing. This device could be classified as a higher range at hardware and processing level compared to the Up-Squared. The only difference is the impossibility of using [GPIOs](#), but it supplements it with a higher processing capability. The size of the [NUC](#) is similar to that of the Up-Squared, what allows to facilitate its installation.

Chapter 4

Theoretical background

In this chapter, there are described several topics and theoretical concepts that are necessary to understand the developed system and its functionality. As mentioned in the chapter 1, the objective of the system is the detection tracking and counting of people and the recognition of anomalous behaviours in crowds. In addition, the system has to be portable, which means that the system has to be able to run on an embedded system.

The first step of the system is the detection of people. This module is very important, since every task such as counting, tracking or detecting anomalous behaviours depends on detection. During the study of the state of the art (described in chapter 2), there was realized that there is great diversity of alternatives to carry out this task.

In this TFM, it has been decided to use a CNN based approach for people detection for two reasons: first, because the chosen hardware is optimized for the implementation of neural networks, as shown in the conclusions of chapter 3, and the second reason is the efficiency and precision observed in the different detection works shown in the chapter 2.

In addition the other tasks: counting and tracking depend on the result obtained from the detection module. These two tasks are designed to combine the CNN result with a bank of Kalman filters that allows tracking and counting people in each frame.

Regarding to the anomaly recognition stage, in this work, the anomaly to identify in the scenarios of interest is the crowd stampede, using a proposal based on the analysis of the optical flow entropy.

Throughout this chapter, four of the most important theoretical concepts of TFM are explained:

1. The concept of artificial intelligence, the use of basic neural networks and the use of CNNs.
2. The Kalman filter as the optimal filter for target tracking.
3. The framework that allows to run the AI in the embedded system providing the real time characteristics.
4. The concept of optical flow and the two alternatives studied in this TFM.

4.1 Neural networks

Nowadays, the term AI can be defined as: a branch of computer science that aims to create a technical equivalent of human intelligence and not only computer scientists work in it, but also experts from other

fields of knowledge. But this explanation does not define what it is, but says what it is supposed to look like.

As it has been observed, the concept of AI is not entirely clear. For this reason, in 1950, the mathematician Alan Turing developed a test to measure AI. With a series of questions, the test of Turing determines whether a machine is recognizable as such. If the answer of the machine are indistinguishable from those of a person, then its artificial nature is confirmed. However, for today AI technology, this definition is not very helpful, as it is developed especially for technical areas. Here it is less a question of the machine being able to communicate, but rather of the machine performing highly specialized tasks efficiently. For this type of technology a restricted version of the test of Turing is used, in which, if in the same area a technical system has the same capabilities as a human being (such as a medical diagnosis or a chess game), it is considered an AI system. At this point, two definitions of artificial intelligence emerge: one “strong” and another one “weak”.

- The definition of strong AI refers to an intelligence that is capable of replacing people in their totality with their several capacities. This universal approach to man as a machine has existed since the Enlightenment, but it is still fiction.
- The definition of weak AI focuses on the development and implementation of AI in clearly defined areas of application. This is precisely the point where current AI research is to be found, whose areas of application, almost in their entirety, are today in the field of weak but widely specialized AI such as, for example, the development of autonomous cars, the elaboration of medical diagnostics or the creation of intelligent search algorithms. In this TFM, a type of intelligence of this type is going to be developed in which a system is capable of performing video surveillance tasks.

These AIs are created using what are known today as Artificial neural networks (ANN), among which are the CNNs used in this work. The basic operation of an ANN and a CNN is explained in detail below.

4.1.1 Artificial Neural Networks

The classical systems as the ANN are inspired in the biological neural networks that form the human brains. Such systems are able to learn to perform tasks just showing them some examples. One example of this behaviour is in image recognition, these systems might learn to identify certain types of image or to extract specific information from them.

The ANNs are formed by a collection of connected units, this type of connected units are called nodes or artificial neurons. In the figure 4.1, it is possible to compare the structure of an artificial neuron with a real neuron. The artificial neuron has several inputs that are modified with different weights depending of the importance of these inputs, this part is comparable with the dendrites in a real neuron. After that, in the neuron core the weights are summed and processed for a non-linear system. This part corresponds to the real neuron body. At the end, the output of the non-linear system is sent to other neurons, this part is similar to axon of the real neuron.

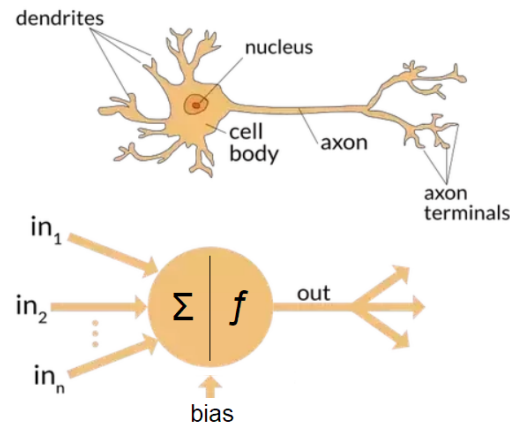


Figure 4.1: Comparison between the structure of a biological neuron and an artificial neuron [19].

The main aim is to generate a system that can solve a problem in the same way that a human brain do. To do this, it is necessary to connect several neurons as in a human brain, creating a set of nodes like the one shown in the example in figure 4.2. Each layer of this network has a number of nodes which does not have to be the same. The layers can be divided in three types: the input layer, that is the first to come into contact with the data. The hidden layer, which may consist of more than one layer, and receive its name because the user has not access to these layers. And the output layer, that is characterized for the number of class of the system.

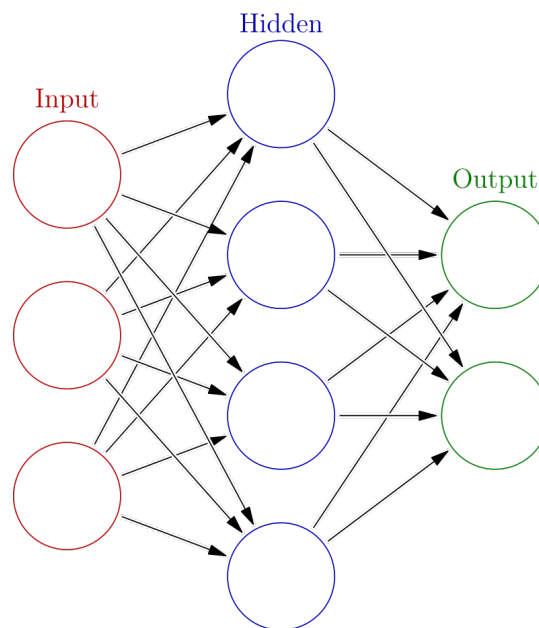


Figure 4.2: Example of neural network structure with three layers [20].

The main components of the ANNs are the detailed below:

- **Neurons:** or artificial neurons represent the biological concept of neuron. When it receives an input, this input is combined with its internal activation state and in some times uses a threshold through an activation function. This whole process generates an output that is used as an input to other neurons or as a result of the system. But the most important point is the activation function,

this function provides a smooth result. This means that, there is a differentiable transition as input values change.

- **Connection and weights:** like biological neurons in [ANNs](#), there are connections that join the output of one neuron to the input to another neuron. All connections have a weight used to represent the importance of this connection. The weights are generated during the training.
- **Propagation function:** this function calculates the input from the output of previous neurons and the connections as a weighted sum. The propagation function is ordered to add the bias term, this node has not input and it is used to generate a displacement in the result.
- **Architecture:** as it is said below, the typical organization used in the [ANNs](#) is a set of layers, especially in deep learning. The layers are sets of neurons, that are connected only to other neurons of the next adjacent layer. The only layer that receives external data is the Input layer, this is the first layer of the network. The last layer produce the result, and it is named Output layer. In between them, there may be zero or more hidden layers. Between two layers, multiple connection patterns are possible. They can be fully connected, with every neuron in one layer connecting to every neuron in the next layer, or pooling, where a group of neurons in one layer is connected to a single neuron in the next layer, thereby reducing the number of neurons in that layer. Neurons with only such connections form a directed acyclic graph and are known as feedforward networks. Alternatively, networks that allow connections between neurons in the same or previous layers are known as recurrent networks.
- **Hyperparameters:** are those constant values that defined the [ANN](#) before the learning process. These values can be: learning rate, batch size and the number of hidden layers. The value of some hyperparameters can be dependent of other hyperparameters.

4.1.2 Learning procedure and parameters

Learning is one of the most important aspects in the [ANNs](#). The learning of [ANN](#) can be explained as the adaptation ability of the network to better handle a task. During the training process, the weights of the network are modified and fitted. This fit tries to minimize the errors, and improve the accuracy. The learning process is completed when the error stops reducing its value. This error evaluation is done through the cost function that is evaluated periodically during learning, until the cost error stops decreasing.

Respect the learning there are two important elements: the learning rate and the cost function.

The **Learning rate** is defined as the size of the steps taken by the model to adjust the error at each observation. This hyperparameter can be used to define an important part of the training. The used of a high learning rate involve shorter training times but with lower accuracy, on the other hand a lower learning rate requires more time to train but there is a improvement in the accuracy. The best solution is to reach a compromise between time and precision and the learning rate is the hyperparameter that controls it.

The **Cost function** or loss function is a function used to measure the performance of a model. The main aim of this function is to quantify the error between the predicted values and expected ones, returning that difference as a real number. There are different types of cost functions that allow to quantify the error of the systems, Among the best known are:

- **Mean Square Error (MSE):** function (eq. 4.1) that measures the euclidean distance between two points. For a set of values it is the sum of the squares of the distances between the real value(Y_r)

and the predicted value (Y_p). The objective of this function is to make the value as close as possible to 0.

$$MSE(Y_r, Y_p) = \sum (Y_r - Y_p)^2 \quad (4.1)$$

- **Mean Absolute Error (MAE):** this function (eq. 4.2) is characterized by the absolute value of the differences between the real value (Y_r) and the predicted value (Y_p). Unlike MSE, this cost function does not penalize those predicted values that are farther away from the actual value.

$$MAE(Y_r, Y_p) = \sum |Y_r - Y_p| \quad (4.2)$$

Regarding the alternatives for training the networks, there are three major learning paradigms: supervised, unsupervised and reinforcement learning.

Supervised Learning [85] is based on a set of data that has already been labeled. The data, which are each of the elements that make up the data set, are composed of a series of fields of characteristics or attributes and a target field, which is the one that is labelled in the training data. The objective of this type of learning is to extract a set of rules that allow predicting the target field for new case studies.

Supervised Learning problems are divided into two categories: regression and classification. These two categories are essentially distinguished by the type of target field, which is numerical in the case of regression and categorical in the case of classification.

- **Regression:** the aim is to predict what the value of the target field will be for a new data, that is, given the properties of a case for which the value of the target field is not known, the model obtained after the training process must be able to predict as correctly as possible. This prediction is made from the values of the variables and the relationship between them. The ability to obtain more or less information from the variables depends largely on the way in which the data have been prepared and also on the algorithm used in the training process.
- **Classification:** problems are those that aim to predict which category the target field of each data belongs to from a list of possible categories. In cases where there are only two possible categories, they are called binary classification or detection problems; while cases that require complex answers and predictions between multiple categories (more than two) correspond to multi-classification problems.

In contrast with supervised learning, in the unsupervised learning, there is no need to label the data, as the aim is to find relationships of similarity, difference or association in the data set. Depending on the objective, the problems can be classified into three different types: Clustering, Detection of Anomalies and Associations.

- **Clustering:** the objective is to generate groupings or clusters by looking for data that are similar to each other. Once the model is obtained, it is possible to predict to which group a new data will belong.
- **Anomaly detection:** unlike in clustering models, what is sought in the detection of anomalies are the data that differ from the others.
- **Associations:** the aim is to find relationships between the different values that take up the fields of a data. In this way, it is possible to deduce rules of association that indicate that when one of the

fields takes a certain value, in general, another of the fields tends to take a particular value much more frequently than if this happened randomly.

Finally, the reinforcement learning this kind of learning is in between supervised and unsupervised approaches. In this mode, there is not labelled data but the system is rewarded when it has a great result and is not rewarded when it fails.

4.1.3 Convolutional neural networks

Besides the classical neural networks, nowadays, the improvements in the hardware and the availability of large datasets have allow the emergence of the Deep Neural Networks (DNN). In this contexts, one of the most used networks are the CNNs, which are having great results in different classification tasks [5, 63]. The CNNs are being widely used in different fields such as image classification [86], natural language processing [87], financial time series[88], image and video recognition and many others.

CNNs are based on biological processes where the connectivity pattern between neurons is very similar to the organization of the animal visual cortex [89]. This is because the brain, in particular the vision part, does not function as a single block. It has a set of layers, which process the information collected by the eyes. In each layer of the brain, characteristics are extracted which are then used in an unconscious way to recognise patterns and objects. In the CNN, something similar occurs, there is a set of layers, which are responsible for extracting the characteristics of the images. Then, with those characteristics, a classification process is carried out.

Another advantage offered by this type of neural network is that it does not need a pre-processing stage, like other classification algorithms. This is because the network learns the filtered actions that in traditional algorithms would be necessary. That allows an independence of knowledge in feature design.

To understand how this kind of networks work, it is necessary the concept of convolution, that can be defined as: the mathematical operation of two real functions (f and g), which produces a third function ($f * g$) which is the superposition of f with a shifted and inverted version of g . Functions f and g can be defined both in the continuous domain 4.3 and in the discrete domain 4.4.

$$f * g(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau. \quad (4.3)$$

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m] \quad (4.4)$$

This definition can be extrapolated to the field of computer vision. Where normally the function f is defined by an image and the function g is defined by what is known as kernel or filter. The kernel is used to extract characteristics from the image f . The output generated from this type of convolution is called a feature map. When working with matrices instead of mathematical functions, it is called matrix convolution. An example of this kind of convolution, with a 3x3 kernel is represented in the figure 4.3.

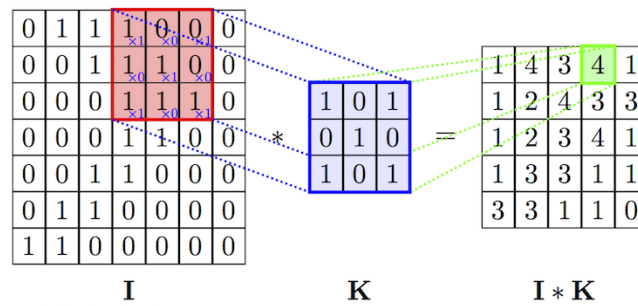


Figure 4.3: Example of the convolution process with a 3x3 kernel.

4.1.3.1 Layers in a CNN

The basic architecture of the CNN, in the simplest case, is formed by a set of layers that transform the volume of the image, reducing it and extracting its characteristics. The three main layers that form the CNNs are: the convolutional layers, the pooling layers and the fully connected layers.

- **Convolutional Layer:** the CNNs use a tensor as input whose size is (number of images) X (image width) X (image height) X (image Depth). The input data is processed in the convolutional layer, and the system extract a feature map whose shape is (number of images) x (feature map height) x (feature map width) x (feature map channels). These convolutional layers have the following attributes:

- Convolutional kernels, defined by a height and width.
- The number of input and output channels.
- Depth, that in convolutional filters is equal to the number of convolutional kernels.

The convolutional layer takes the input and this is convolved and passed to the next layer. This mechanism is very similar to the one in a neuron in the visual cortex to a specific stimulus.

In CNN the vectors of weights and bias are denominated as filters and represent particular features of the inputs. An important point of this networks is that many neurons can share the same filter. This avoids a bad use of the memory in the hardware.

- **Pooling Layer:** these layers are periodically inserted between the convolutional layers. The aim of this type of layer is to reduce the spatial size of the feature map, thus reducing the amount of parameters and computation of the CNN. In most of the networks two downsamples are used, reducing to the half the size in width and height of the feature map, in the figure 4.4 it is shown the as the downsample affects to the spatial volume.

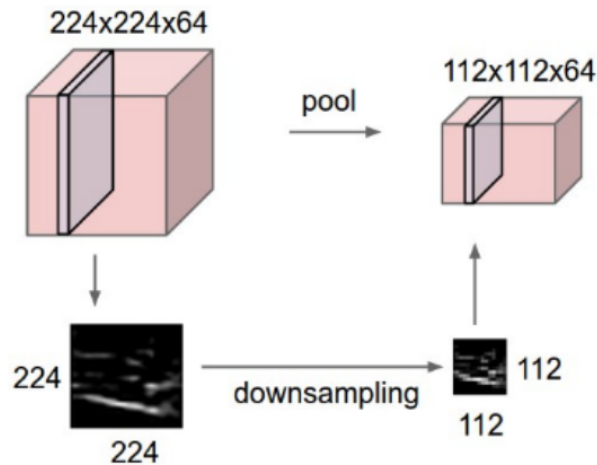


Figure 4.4: Effect of downsampling in an output volume.

There are two ways to carry out the downsampling 4.5, by selecting the maximums of the pool or by the average of the pool. Figure 4.5b shows the average pool and figure 4.5a shows the max pool.

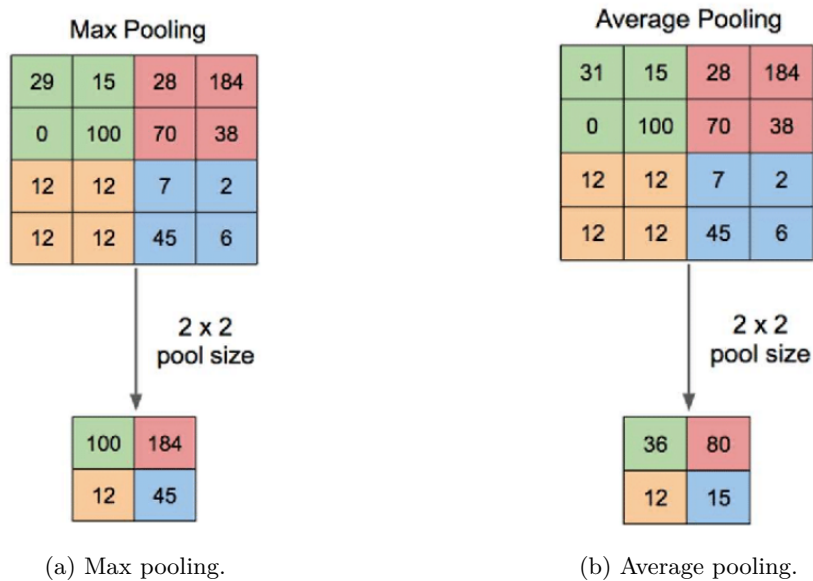


Figure 4.5: Examples of the two different types of pooling [21].

- **Fully connected Layer:** is a network of ANNs that can have more than one layer where the neurons that form it are all interconnected. This type of network is located at the end of the CNN, and its purpose is to collect the characteristics extracted along the convolutional layers and to classify the image input in the corresponding class. The use of this layer allows a classification using non-linear combinations of the extracted characteristics. Figure 4.6 shows a representation of the shape of one of these layers.

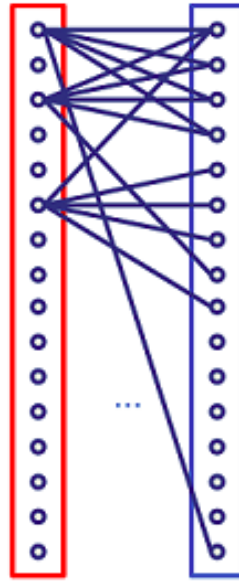


Figure 4.6: Example of general diagram of a fully connected network [19].

- **Softmax layer:** fully connected layers are usually connected to a softmax output. It allows the sum of probabilities of each class of one. In this way the probabilities are normalized. In equation 4.5 the softmax function is shown in a mathematical way

$$P(z) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K \quad (4.5)$$

Finally, the global scheme of an CNN is similar to the one shown in figure 4.7. It must be taken into account that all these parts can be combined to generate an infinite number of different networks that, depending on the architecture, can be more useful for some application or for another.

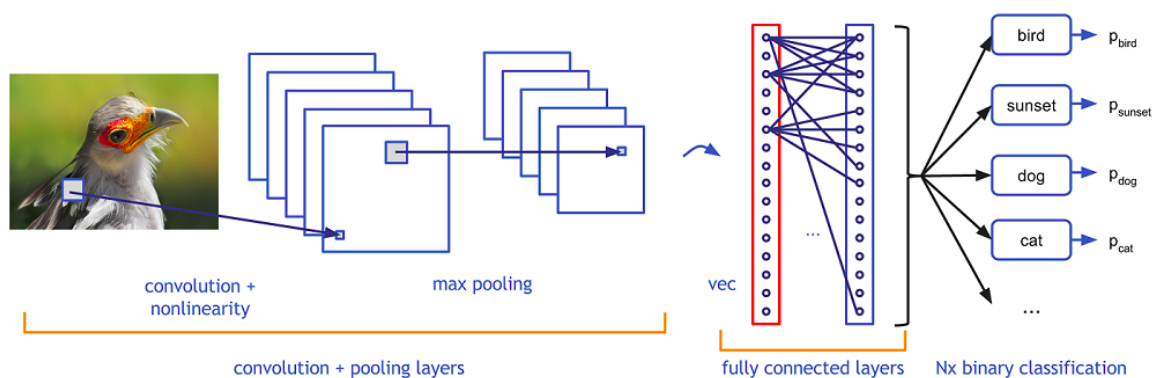


Figure 4.7: Example of CNN architecture [19].

4.1.3.2 Activation functions

When the convolutions between the image and the kernel are carried out, the values that the resulting images have are usually not limited. This is a problem, because as CNN advances, and more convolutions are made, these values can grow more. In order for these values not to grow indefinitely, the convolutional layers are usually followed by activation functions that return an output from an input value. Usually

the set of output values in a given range such as (0,1) or (-1,1). There are different types of activation functions such as: RELU, SIGMOID, softMax, Hyperbolic Tangent, etc.

- **Rectified Linear Unit:** the Rectified Linear Unit function transforms the entered values by cancelling the negative values and leaving the positive ones as they come in. Equation 4.6 shows the way to represent mathematically this function

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{for } x < 0, \\ x & \text{for } x \geq 0 \end{cases} \quad (4.6)$$

Figure 4.8 shows a graphic representation of the ReLU function.

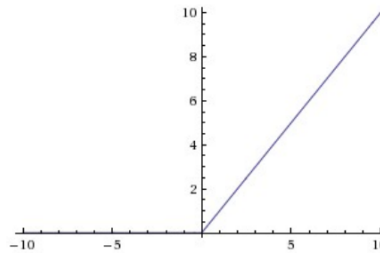


Figure 4.8: Rectified Lineal Unit function.

- **Sigmoid:** the sigmoid function transforms the entered values to a scale (0,1), where high values tend asymptotically to 1 and very low values tend asymptotically to 0. Equation 4.7 shows the way to represent mathematically this function.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (4.7)$$

Figure 4.9 shows a graphic representation of the Sigmoid function.

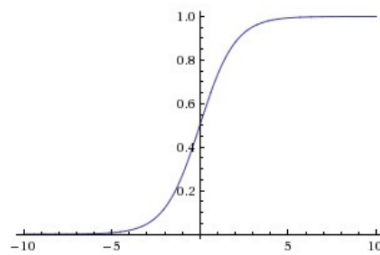


Figure 4.9: Sigmoid function.

- **Hyperbolic tangent:** function transforms the entered values to a scale (-1,1), where high values asymptotically have 1 and very low values tend asymptotically to -1. It can be seen that this function is very similar to the one explained above (sigmoid). Equation 4.8 shows the way to represent mathematically this function

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (4.8)$$

Figure 4.10 shows a graphic representation of the Hyperbolic Tangent function.

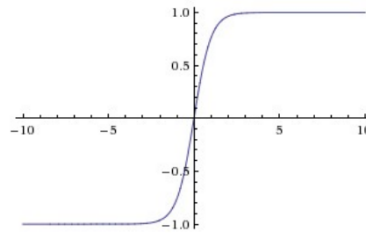


Figure 4.10: Hyperbolic tangent function.

4.1.3.3 CNN architectures

Nowadays, CNNs can be made by scratching, choosing the number of layers, the size of the input image, the activation functions, etc. But another alternative is the use predefined CNN models, which have been tested to work correctly. Among them, the following can be found.

- **LeNet:** [90] was one of the first CNN architectures used for detection and classification processes. Being able to detect digits, zip codes, etc.
- **AlexNet:**[91] was a version of LeNet but much more complex. It added layers of depth to the architecture, increasing the extraction of features from the input images. This allows to be able to recognize more difficult images at a visual level. In 2012 this architecture managed to surpass one of the records proposed by ImageNet Large Scale Visual Recognition Challenge.
- **GoogleNet:**[92] architecture designed by google as an Inception module. It manages to reduce the number of network parameters compared to other networks such as AlexNet. In addition, it added new features such as the use of avg-pooling or the insertion of the fully-connected layer at the end.
- **MobileNet:** [35] is a network designed by google in the year 2017. The aim of this network is to reduce the processing times that most CNNs had by using a different convolutional method than the normal one. It is an architecture designed to be used in portable devices with little computing capacity.
- **DarkNet:**[93] is a very light and easy to process CNN open-source. It is characterised by its low resource usage and by the network used to test the YOLO[25].

4.1.4 Solutions for object detection

From 2012 onwards, CNN architectures have been a popular tool for the task of image classification. The main advantage of using CNNs over traditional methods is, that the task of feature engineering is not required, due to the ability of CNNs to learn automatically. For this reason, many researchers began to study the tasks of detecting objects within an image using CNN. The main idea behind most approaches was to use proposals for regions of the image that could contain objects and to classify these. Based on the above idea, a brute-force approach could be proposed in which a window of, for example, a size 3x3 was taken and each square through which the window was moved in the image was classified. However, this solution is computationally very expensive and slow. Below are some of the most popular approaches that optimize this task.

- **R-CNN:** to solve the problem of selecting a large number of regions, Ross Girshick et al [22]. A method is presented in which he uses a selective search to choose 2000 proposals of regions from the image. The selective search algorithm is as follows:

1. Generate divisions in the image by means of brute force, considered as candidate regions.
2. Use a greedy algorithm to recursively combine regions with similar characteristics in larger regions.
3. Use the regions resulting from the algorithm as the proposed end regions.

Once the proposals for the final regions have been received, they are passed on to a **CNN** which converts them into characteristic vectors. Finally these vectors are sent as input to train a **SVM** classifier in order to detect the presence of objects in the candidate proposals. Figure 4.11 shows a workflow of the operation mode:

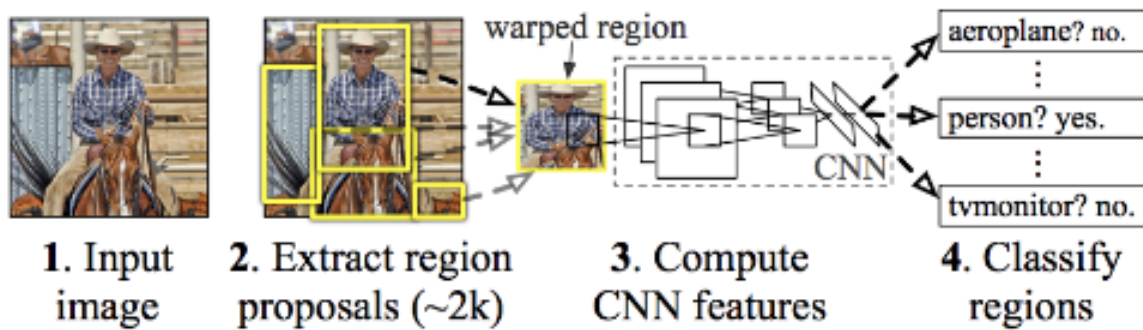


Figure 4.11: R-CNN structure [22].

R-CNN problems:

- The time needed to train a network with a minimum number of entries in order to obtain good results is very high.
 - It is not possible to implement them in real time, since detecting each image would take a large amount of time on modern hardware.
- **Fast R-CNN:** [23] this approach was presented by the same author of the previous algorithm (R-CNN) and it is intended to solve the deficiencies of this algorithm, in order to increase the speed of object detection. The difference with its predecessor is that in FAST R-CNN the Convolutional Neuronal Network does not receive the proposals of regions, but directly the input image, in order to generate a map of convolutional characteristics. From this map the proposals of regions are identified and passed through a pooling layer to resize them. Finally, a softmax layer is used in order to normalize the results and obtain the probabilities for the different classes of the proposed region.

By not having to send all region proposals to the Convolutional Neuronal Network as input, Fast R-CNN is much faster in training and validation, reaching a classification time of about 2 seconds. However, it still cannot achieve real time object detection.

Figure 4.12 shows a workflow of the operation mode:

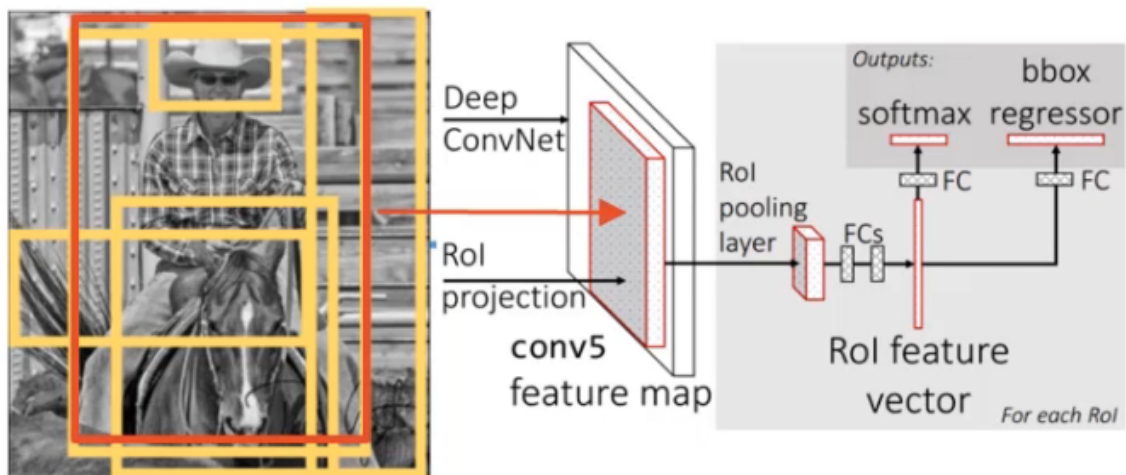


Figure 4.12: Fast R-CNN structure [23].

- Faster R-CNN:** the bottleneck of the two previous algorithms is the use of a selective search for the proposal of regions. In the article Faster R-CNN [24] published by Shaoqing Ren et al. in 2016, an algorithm is presented that avoids the use of selective search for region proposals and allows the network to generate them. The initial operation is the same as Fast R-CNN, the input image is sent to a CNN to obtain the feature map. Then, the selective search algorithm is replaced by a new neural network that predicts region proposals. Finally, these proposals are resized using a pooling layer that is used to classify the proposed region of the image. As can be seen in the following figure 4.13, Faster R-CNN achieves a detection time of 0.2 seconds per image, which implies a rate of 5 images per second, feasible for real time use.

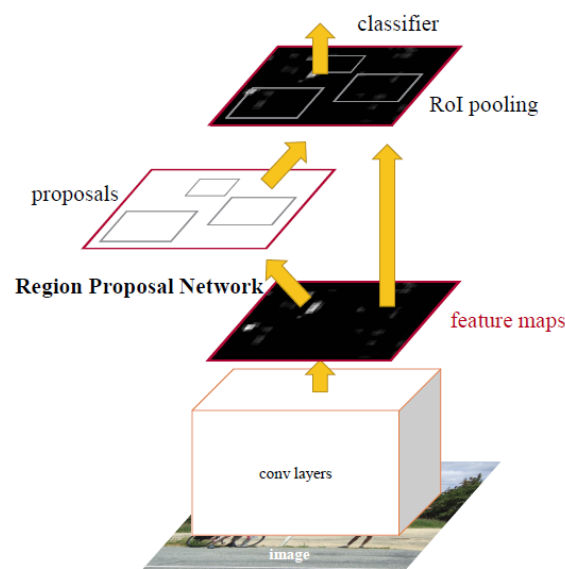


Figure 4.13: Faster R-CNN structure [24].

- YOLO:** [25] (You Only Look Once) takes a completely different approach, it is designed as a detection system and not as an adaptation of a traditional sorter. In essence, YOLO applies a

single Convolutional Neural Network to the entire image. The network divides the image into regions and for each region predicts boundary boxes and class probabilities.

The operation is as follows:

1. **YOLO** divides the input image into a matrix of x by y cells. In each cell you can detect up to a z number of different objects.
2. In each cell of the matrix, the possibility of an object within the matrix is predicted and a bounding box is created around that possible object together with the adjacent cells. The bounding box does not give information about the class of the object in question, but its thickness increases the greater this probability.
3. For each bounding box, the object that may be inside it is predicted. These are now labelled according to the class they contain, as can be seen in the image in the middle of figure 4.14.

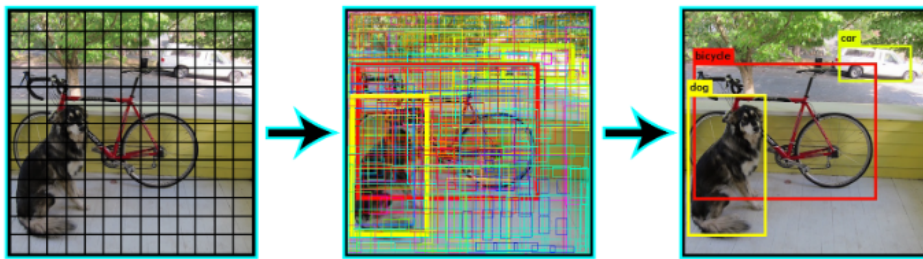


Figure 4.14: **YOLO** workflow [25].

4. Finally, a minimum threshold of probability that a bounding box must have in order to really contain a class is fixed and all the boxes that do not exceed that threshold are eliminated. This returns the expected result.

YOLO offers several different configurations of its architecture, depending on the objective of the task, whether speed or precision. **YOLO's** main architecture consists of 25 convolutional layers followed by 2 fully connected layers. Behind each group of convolution layers illustrated in figure 4.15, there is a max-pooling layer.

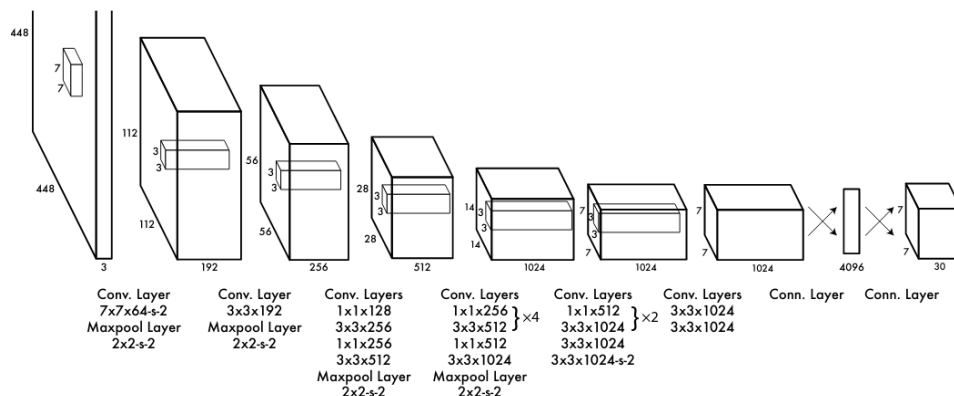


Figure 4.15: **YOLO** network architecture [25].

- **SSD**: the algorithm **SSD** [26] arises with the idea of solving the problem presented by standard convolutional networks such as **Faster R-CNN** [59], this problem is the slowness of training and detection due to the complexity of its multiple components. The main idea of this new deep learning algorithm is to include a technique known as **multibox** 4.16, inspired by the work of Szegedy et

al. [94] in the inception network that allows to detect objects in a single pass of the image, without having to depend on different branches.

The technique starts with the creation of so-called priors. Priors are fixed-size bounding boxes whose dimensions are pre-computed based on the dimensions and locations of the bounding boxes provided to the algorithm during training. They are called prior because they are based on the bayesian statistical inference method, more specifically on an a priori probability distribution of where the object will appear in the image. The priors are selected so that the Intersection over Union (IoU) is between 0.1 and 0.5 with people noted in the image. This method of computing priors is better than selecting them randomly in the image, however, it involves having to pre-train the multibox predictor, which is somewhat costly. However, there is a solution known as fixed priors, which consists of the following: given a training image, when it is introduced into a CNN and progresses through its different layers, its dimensionality is reduced (thanks, for example, to the 'pooling layers') and feature maps are obtained, what allows the input image to be discretized in cells. Each of these cells, similar to an anchor, acts as the center of a set of rectangles with different aspect ratios that can contain people. The recommendation is to use between four and six priors per cell. Adding variations in the scale and aspect ratio of these can help detect more objects, but slows down the algorithm.

By obtaining feature maps of different dimensionality (different number of cells), it is possible to detect objects of different scales in the image. Similarly, the number of feature maps that are generated can be increased with respect to the original network by adding convolution blocks, which increases the probability of detecting and classifying correctly, but slows down the algorithm.



Figure 4.16: Multiboxes generated by the SSD [26].

Regarding the architecture where the SSD algorithm is used, it is based on the architecture of the VGG-16 [27] (figure 4.17). The VGG-16 is a convolutional network that has been trained with over one million images. It is 16 layers deep and is capable of classifying images into a thousand different object categories from keyboards to animals. The SSD architecture is based on the VGG-16's architecture because of the VGG-16's high accuracy in image classification. However, the architecture of the SSD is not exactly the same as that of the VGG-16, as it replaces the fully-connected layers with 6 convolutional layers, 5 of which are responsible for detecting objects.

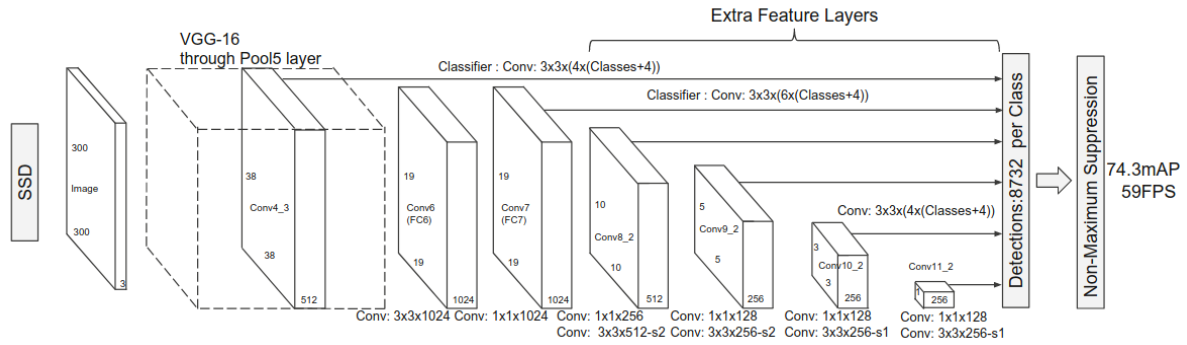


Figure 4.17: SSD network architecture [26,27].

4.2 OpenVino toolkit

OpenVINO [28] is a toolkit developed by Intel and it is used to ease the optimization of Deep Learning models. Intel(R) Distribution of OpenVINO is the version used in this TFM because it is the one pre-installed in the selected Hardware (HW) described in chapter 3.

OpenVINO is formed by two main blocks, as shown in figure 4.18: the run model optimizer that generate Intermediate Representation (IR) files through trained model and the Inference Engine, where the IR is executed. In this block, it is possible to choose the technology where it is executed (CPU, Intel Processor Graphics, VPU, FPGA, etc.). These blocks allow a high level of parallelization in systems using CNNs. A general block diagram of the OpenVINO architecture is shown in figure 4.18. Since it is a system belonging to a private company, it is not possible to access the internal technology of the main blocks.

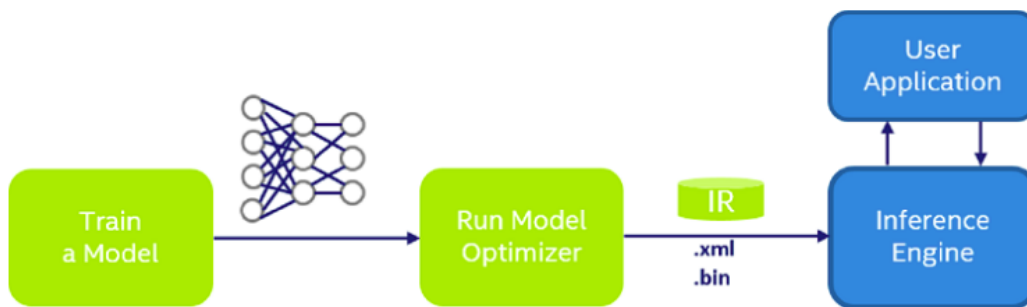


Figure 4.18: OpenVino structure [28].

- **The Model Optimizer** is a tool that works through terminal both in Windows and Linux, allowing to generate the optimized system by means of the introduction of a single command in shield. This optimizer generates a static analysis of the model and ensures that the deep learning models automatically adjust, thus achieving a model that works optimally in the final device.

The IR file describes the network once it has been optimized by OpenVINO. This representation includes two files:

- The topology file: a XML file that describes the network topology.

- The trained data file: a .bin file that contains the weights and biases binary data.

One of the advantages of the IR is that these generated files can be read, inferred and loaded into the Inference Engine. That means that independently of the machine, the trained network can be used if the OpenVINO framework is installed. This is because the Application Programming Interface (API) is unified in several platforms. In addition, the IR offers post-training optimization tools.

- **The inference engine** is defined as a set of libraries (in C++ language) provided by a common API that offers inference solutions in different types of platforms, such as: GPU, FPGA, CPU or VPU. The Inference Engine API is in charge of reading and interpreting the files generated by the IR. This way, input and output formats are set, finishing with the execution of the models in the devices. The steps followed by the inference engine are the following:

1. To take the model generated by the IR with the .xml and .bin files.
2. To optimize the execution of the inference for the target hardware.
3. To generate an inference solution, which is characterized by a reduced footprint on the embedded platforms.

The inference engine supports inferences from multiple network types and image classification frameworks: from networks like GoogLeNet[92] or ResNet[95] to other networks that are entirely convolutional like SSD[26], faster R-CNN [59], etc.

4.3 Kalman Filter

The Kalman filter is an algorithm that uses a set of measurements observed in time to provide the estimation of some unknown variables.

The filter name provides of its creator, Rudolf E. Kalman. This algorithm has numerous applications in technology. Normally this filter has been used in navigation, guidance and control of vehicles, however it is also used in other areas like signal processing [96] as a different perspective to signal processing , central nervous system [97] for the real-time study of muscle conditions using EMG signals, etc.

The Kalman filter is used for states estimation based on linear dynamic systems with or without Gaussian noise in state space format. The model defines the evolution of states from k-1 to k as:

$$x_k = Fx_{k-1} + Bu_{k-1} + w_{k-1} \quad (4.9)$$

where:

- x_k : is the actual state vector.
- F is the state transition matrix, this matrix is applied to the state vector in k-1 x_{k-1} .
- B: referred to as a control input matrix, this matrix is applied to the control vector u_{k-1} .
- w_{k-1} : is the process noise vector, which is a representation of the noise must be zero-mean Gaussian with the covariance Q , *i.e.*, $w_{k-1} \sim N(0, Q)$.

Both the process model and the measurement model are related by describing the relationship between the measurement and the state in the current state in k as:

$$z_k = HX_k + v_k \quad (4.10)$$

In the former equation:

- z_k : is the actual measurement vector.
- H : is the measurement matrix.
- v_k : is the measurement noise vector that is assumed to be zero-mean Gaussian with the covariance R , i.e., $v_k \sim N(0, R)$.

The main function of the Kalman filter is to generate an estimation x_k in a time k . This generates a initial estimate in x_0 . Then, the measurements are represented with z_1, z_2, z_3, z_k , and the matrices F, B, H, Q, R are used to describe the system. Note that these matrices have not subscripts because here it is assumed that they are time invariant, although in the real world it is necessary be aware with this issue. The covariance matrices Q and R are used to represent the statistics of the noise. In real problems these matrices are not known but for the theoretical study it is assumed that the noise is gaussian white noise.

Kalman filter algorithm has two steps, as it can be observed in figure 4.19: prediction and update.

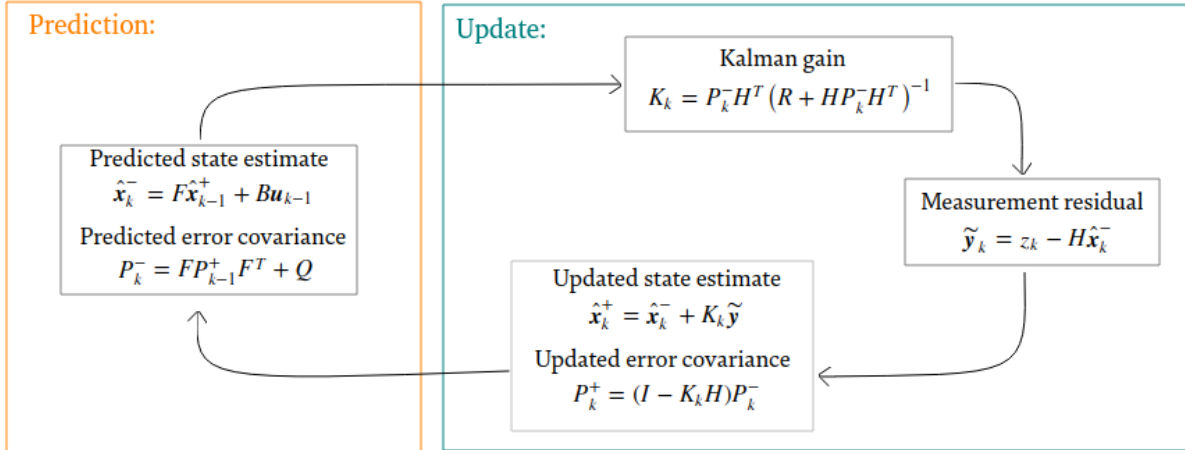


Figure 4.19: Kalman structure.

For the Kalman filter there are several operators that have different meanings depending if they are used or not. the operator " $\hat{\cdot}$ ", means an estimation of a variable. The other operators are " $-$ " and " $+$ " are used to stated predicted (prior) or update (posterior) estimates, respectively.

In the prediction stage, the previous state is generated by the value of the estimate of the previous updated state. The term P is defined as the error covariance of the state. This matrix is used to define the covariance error estimation of the filter. I is possible to define the covariance of a random variable like $cov(x) = E[(x - x^-)(-x^-)T]T$, where E indicates the expected (mean) value of its argument. The covariance error increases in the prediction stage due to addition of Q , that means that the filter more inaccurate in the state estimate after the prediction step.

In the update stage, in first place it is necessary to compute the measurement residual \tilde{y}_k , this measurement is know as innovation and it can be defined as the difference between the true measurement z_k and the estimate measurement $H\hat{x}_k^-$. With this information the filter estimates the current measurement multiplying the measurement matrix by the predicted state. The residual measurement \tilde{y}_k is then multiplied by the Kalman gain K_k , what is used to modify the predicted estimation x_k^- . Once the updated state estimation is obtained, the update error covariance is computed P_k^+ . If it is compared the predicted error covariance and the updated error covariance if the first is smaller than the second, which means the filter has more confidence in the state estimation after the measurement is used in the update stage.

A remarkable aspect is to have a starting point to run/implement the Kalman filter. For the initial values, there are necessary the state estimate x^+0 , and the error covariance matrix P^+0 . Together with Q and R , X_0^+ and P_0^+ , it is possible to obtain desired performance. Also it is possible to use the “initial ignorance”, in which the user chooses a large P_0^+ to achieve a rapid convergence.

It is important to emphasize that Kalman filter is designed to be used in linear models. The models are represented by matrices which are called F, H and B . As mentioned below, the process noise and measurement noise are additive Gaussian.

Figure 4.20 shows a graphic scheme about how Kalman filter functions work.

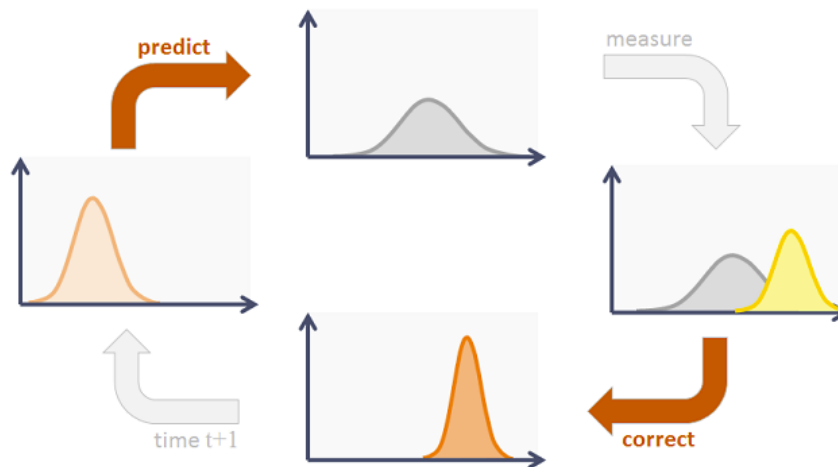


Figure 4.20: Example of a kalman filter cycle [29].

The figure shows an example of the Kalman filter cycle. The filter starts by generating a prediction (left) with the system model. Then the system takes a measurement (up). And with the prediction, the measure taken (right), depending on how good the model and measuring system is. It generates a correction that is the output of the kalman filter(down) and is taken as input from the system model [29].

4.4 Optical Flow

The Optical Flow (OF) can be defined as the pattern of motion of objects, surfaces and edges in a visual scene, caused by relation motion between two points in a scene. Another definition can be the distribution of different brightness velocities pattern in an image. This concept was not introduced for the science, but by the American psychologist James J. Gibson [98] in the 1940s to describe the visual stimulus of animals in motion through the world. Gibson pointed that this OF is vital for survival for affordable perception, that is the skill to discern possibilities for action within of a situation and an environment. Later, it was

demonstrated the importance of the OF for the perception of movement by the observer, being able to perceive shapes, distances, moving objects and the control of locomotion. In the figure 4.21, it is possible to observe a ball in different frames. With this picture, the OF concept is shown in a graphical schema, representing the white arrow the displacement vector.

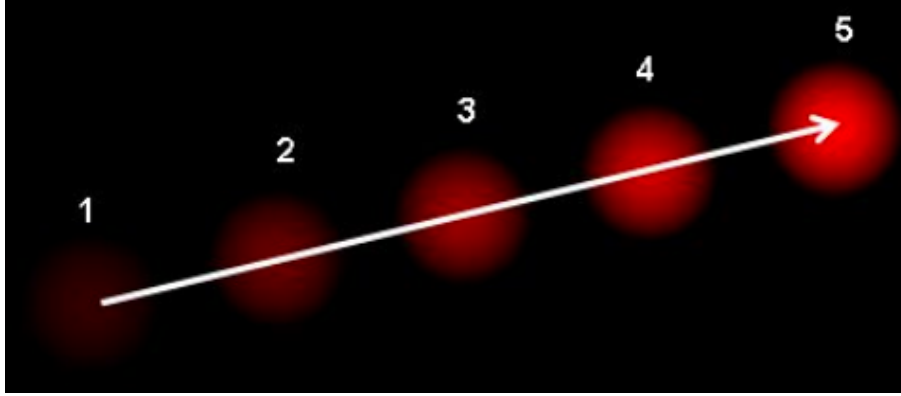


Figure 4.21: Optical flow example [30].

As time went by, branches of technology such as robotics or computer vision became interested in studying the concept of OF. Since then different techniques have been developed to measure OF in different ways. Some of these methods were based on the study of lighting changes [99], or the study of the segmentation of objects [100] in a set of images.

The OF looks for the way to compute the movement between two frames of two consecutive images ($t + \Delta t$). For this purpose, it is based on the intensity of a pixel with coordinates (x, y) at a certain time t , defined as $I(x, y, t)$, and the change of the intensity in that pixel from one frame to the next one $(\Delta_x, \Delta_y, \Delta_t)$, from what the following brightness constancy constraint can be defined:

$$I(x, y, t) = I(x + \Delta_x, y + \Delta_y, t + \Delta_t) \quad (4.11)$$

It is assumed that the movement in the image is small. Therefore the Taylor series of the brightness constancy constraint is developed as shown in the equation 4.12:

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\delta_I}{\delta_x} \Delta x + \frac{\delta_I}{\delta_y} \Delta y + \frac{\delta_I}{\delta_t} \Delta t + \text{higher order terms} \quad (4.12)$$

The higher order terms are truncated to get the equation 4.13:

$$\frac{\delta_I}{\delta_x} \Delta x + \frac{\delta_I}{\delta_y} \Delta y + \frac{\delta_I}{\delta_t} \Delta t = 0 \quad (4.13)$$

Finally, the expression in equation 4.13 can be divided by Δ_t , obtaining the following equation or OF:

$$\frac{\delta_I}{\delta_x} \frac{\Delta x}{\Delta t} + \frac{\delta_I}{\delta_y} \frac{\Delta y}{\Delta t} + \frac{\delta_I}{\delta_t} \frac{\Delta t}{\Delta t} = 0 \quad (4.14)$$

Equation 4.14 can be rewritten as:

$$\frac{\delta_I}{\delta_x} V_x + \frac{\delta_I}{\delta_y} V_y + \frac{\delta_I}{\delta_t} = 0 \quad (4.15)$$

where V_x and V_y are the velocity components or also called, **OF** of $I(x, y, t)$, and $\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y}, \frac{\delta I}{\delta t}$ are the image derivatives at (x, y, t) . These derivatives, can also be written as I_x, I_y and I_t , as shown in equation 4.16.

$$I_x V_x + I_y V_y = -I_t \quad (4.16)$$

The equation 4.16 has two unknowns, so it has no solution. For this reason, it is necessary to add another restriction that allows getting another equation to obtain the **OF**. There are different approaches for estimating the **OF** in images, being the most known alternatives the ones proposed by Lukas-Kanade (LK) [101] and Gunnar-Farneback (GF) [32], that are briefly explained below.

4.4.1 Lucas-Kanade method

Lukas-Kanade method [101] is based on the premise that the neighbouring pixels in an image have a similar motion value. This method uses a 3x3 patch around the point. In this way the 9 points around the center have the same motion. The next step is to find (f_x, f_y) for these 9 points. At this point, there are 9 equations with two unknown variables. To solve this situation, it is used the least squared fit method, obtaining a final solution shown in eq. 4.17 with two equations and two unknown variables.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \quad (4.17)$$

To obtain the flow vectors, this method takes a set of points to track. The idea is effective and simple but it has some problems: this algorithm works very well when the movements between the frames are small, but fails when the collected movements are big. This problem is solved using a pyramidal approach [102]. The concept of use the pyramid is based on doubling the size of the image, taking into account the chosen point to follow the **OF**. This allows small movements to be eliminated and large movements to become small. In this way, an **OF** vector can be obtained along with the scale.

As it has been explained, the extraction of the **OF** by means of the Lucas-Kanade method is sparse, since it is obtained only for some points in the image. This can be seen implemented in figure 4.22, which only makes a study of the **OF** of certain points of the image. This is represented in wakes corresponding to the front of the vehicles.



Figure 4.22: Optical flow example [31].

4.4.2 Gunnar-Farneback method

As it is shown in [32], the algorithm developed by Gunnar and Farneback seeks to estimate the movement of each point in the image by approximating its evolution through polynomial expressions. This algorithm provides a dense OF estimation, that is, it calculates the movement of all pixels in the image. These algorithms are usually quite slow because they have to work with a large number of points. However, the Gunnar-Farneback proposal, by using polynomial expressions, greatly simplifies the computational cost and achieves much shorter processing times than other similar algorithms.

At the end, a dot matrix is obtained, representing the movement of each pixel from its initial position to its position in the last image. These data are known as motion vectors and in figure 4.23, it is shown a representation of them in a sample image.

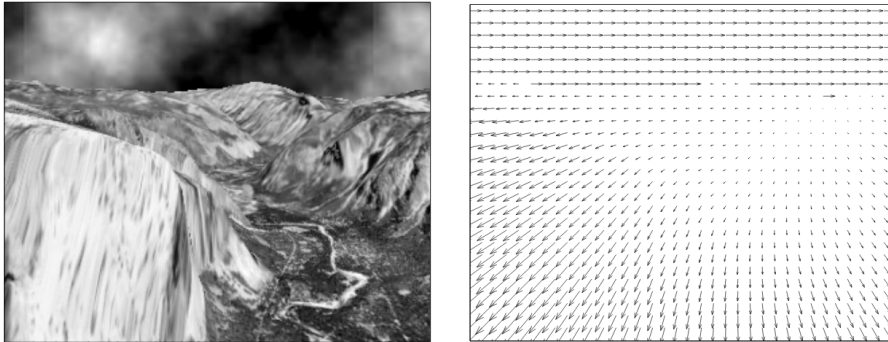


Figure 4.23: Optical flow example [32].

Chapter 5

Implementation

This chapter shows how all the theoretical concepts explained in the previous topic 4 have been applied. As it has been explained in the previous chapters, the developed system is composed of three different parts: the people detection module, the people tracking and counting stage and the anomalous behaviour recognition, that is focused on crowd stampedes.

For the development of this work, the requirements explained in chapter 1 must be taken into account. The system must be able to be executed in embedded systems and also work in real time. This implies an efficient use of the systems, both at HW and Software (SW) level. For this reason, it is proposed a system that develops to the maximum the processing capacity of the microprocessor using its cores in parallel and using the specialized HW (VPU) offered by the devices selected in chapter 3. To carry out this implementation, it has been decided to separate the basic processing part from the AI processing part. Figure 5.1 shows a diagram of how the system works and how it is structured. The concept of parallelization by means of threads can be seen in it. This allows the system to be able to run on more than one microprocessor cores, thus avoiding congestion in the cores. This allows a better management of the HW. In addition, being parallelized, allows reducing the processing time, so the objectives of a portable system in real time can be achieved.

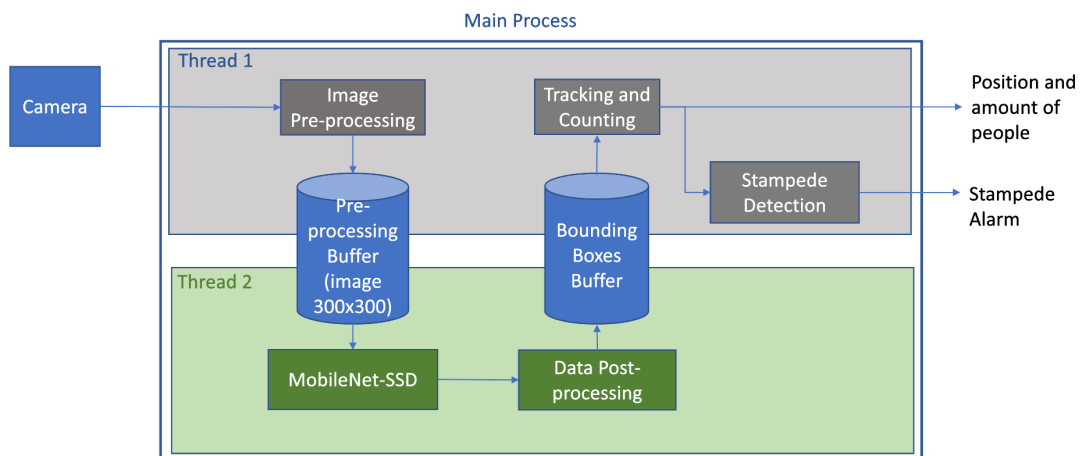


Figure 5.1: General block diagram of the developed system.

As it can be seen in figure 5.1, the system is divided in two totally independent processing areas. These areas communicate with each other through the use of buffers, in this case two. The first one, called "*Pre-processing Buffer*", is in charge of storing the images pre-processed by the system. And the second buffer, called "*Bounding Boxes Buffer*", is in charge of storing the post-processed results returned by the MobileNet-SSD.

Due to the parallelization by threads, the workflow of the system is as follows.

In the area of the first thread, the information is collected by the sensor or from memory in the case of a video. This information is pre-processed and stored in "*Pre-processing Buffer*". Then in the same thread, it is analyzed if "*Bounding Boxes Buffer*" contains data. If not, the process finishes. In case it does, the tracking and counting actions are performed. If, when counting, it is observed that the number of people exceeds a limit, the stampede system is activated and it analyzes the image sequence, generating an alert signal if necessary.

With respect to the second thread, it is responsible of executing the part corresponding to the CNN. For this purpose, it collects the pre-processed frames from the "*Pre-processing Buffer*" and transfers them to the VPU, on which the MobileNet-SSD network is running. Once the network has made its predictions, it returns a set of results, which are post-processed. Finally, the filtered information is stored in "*Bounding Boxes Buffer*".

The aim of the system is to detect, count, track and detect stampedes. In this chapter it is explained in detail the implementation and the operation of the modules shown in the figure 5.1, to achieve the objectives of this work.

5.1 People detection

In this section it is explained in detail the development of the algorithm corresponding to the detection of people within the system. As mentioned above, this stage is the most important within the system, due to the dependence of the other stages on the results of the people detector. In figure 5.2, the blocks corresponding to this stage are highlighted in red color.

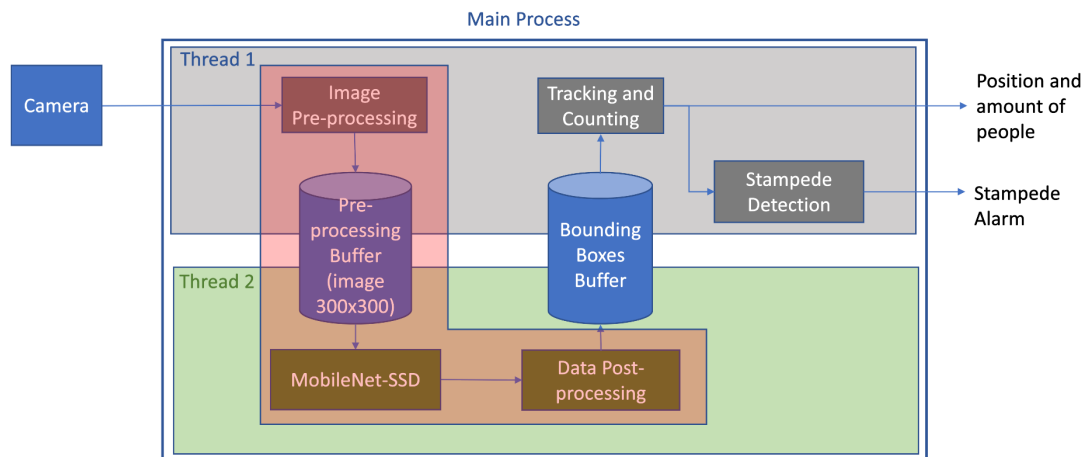


Figure 5.2: System diagram where the people detection module is highlighted in red.

As it can be seen in figure 5.2, it is composed of three parts: image pre-processing, the MobileNet-SSD network and the post-processing of the data obtained through the network. Each of these three blocks is explained in detail below.

5.1.1 Image pre-processing

The pre-processing of the input image is an important stage in the system. Pre-processing is based on, adapting and modifying the input image, so that the system is able to extract the maximum information from that image. This process has two steps:

1. **Resize and normalize values:** once the input have been acquired, and the region of interest has been located, it is necessary to change the size of these image to 300x300 pixels. This size is due to the fact that the MobileNet-SSD network works with 300x300 images. Also, this network does not work with pixel values from 0 to 256 but with values from -1 to 1. Therefore, the values of each RGB pixel must be normalized to generate an array with values between ± 1 .
2. **Matrix transposition:** due to the internal structure of the VPU, it is necessary to reorganize the values corresponding to each image once it has been normalized. After the normalization, an HxWxC matrix is obtained, in which H has a normalized value for the height, W has a normalized value for the width and C corresponds to the number of channels, in this case 3 as it is RGB. As mentioned above, in order for the VPU to work correctly according to the manufacturer's description, it is necessary to pass it an array with the following structure, CxHxW. So it is necessary to make a transposition of the normalized matrix.

5.1.2 Network architecture

This section explains how it works and why this network has been chosen for this system. The chosen network is a MobileNet-SSD, a MobileNet architecture, which uses the SSD method for object detection. The architecture that forms this network is very similar to the one used in the VGG-16 [27]. The main difference is the replacement of the VGG-16 module by the MobileNet module. The reasons for using this architecture are two: the first one, a fast architecture was needed, which could be implemented with selection algorithms such as SSD. And the second, the resource consumption was low, because the devices used in this TFM are portable and its HW is not as powerful as a high-performance PC. For this reason MobileNet was chosen, because its main feature is the speed of computing and the use of a type of convolutional layers that allow the use of fewer resources.

The architecture used is shown in figure 5.3. It can be seen that at the beginning of the network there is the MobileNet module, composed of 35 convolutional layers, which are responsible of the feature extraction. A set of 5 layers of these 35 are in charge of carrying out the classification of objects applying an SSD.

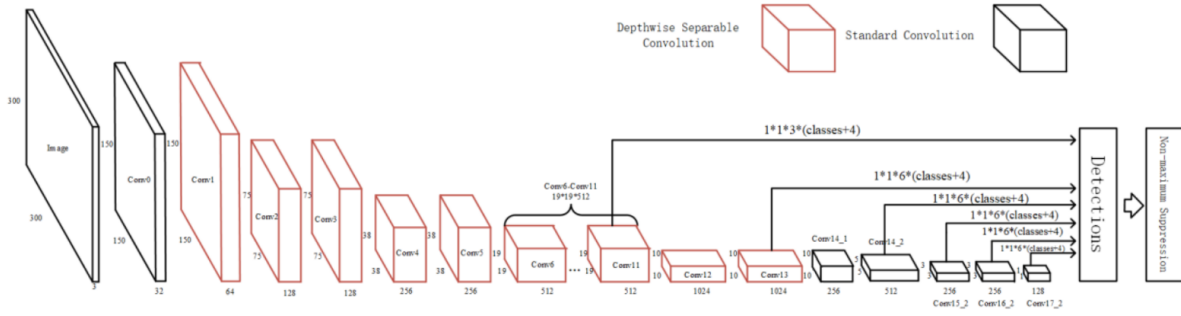


Figure 5.3: MobileNet-SSD architecture.

This architecture, apart from being fast and cheap in the cost of computational resources, can be implemented in the VPU carried by the devices selected in chapter 3, what is important because one of the objectives of this TFM is to use an architecture that allows to be executed in portable devices.

5.1.2.1 Feature extraction

The MobileNet network [35] was developed with a different approach than other networks. The main goal was to perform a precise and lightweight network, that should be able to process information at high speed. This is because the speeds of the other networks were extremely slow. To achieve this goal, maintaining the same accuracy than other networks, it was necessary to change the approach of the concept of convolution. In the convolutional networks, the main part is where the input data is convoluted by different filters that are responsible for extracting their characteristics. This has a high computational cost, so MobileNet use separable convolutions Conv_Dw_PW whose theoretical background is explained later. With this it managed to increase the speed and the extraction of more features with less depth in the network.

Due to the previously explained reasons the MobileNet replaces the VGG-16 [27] network of the original SSD for feature extraction. It was demonstrated that, for the ImageNet-1000 [103] dataset, the MobileNet is able to obtain the same results that the VGG-16, but with only 1/33 of the parameters.

Due to the limitations of HW, when using portable devices without large computing power, and to the requirements of the work, of a system that is able to work in real time, it has been decided to use this architecture. In the following points it is explained how the computational reduction and the extraction of characteristics of this architecture are achieved.

The MobileNet is based on a structure called Conv_Dw_PW, shown in figure 5.4. It is composed of two types of convolutions. The first, Depth-wise-layers (DW) which is a type of convolution where a 2D depth kernel (3x3) is applied, at each of the tensor's depth levels. And the second Point-wise-layers (PW) which is a convolution model that uses a 1x1 kernel, whose depth is equal to the number of channels in the input image and which iterates at each of the image points.

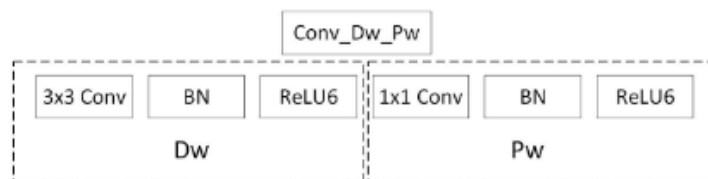


Figure 5.4: The basic architecture of Conv_Dw_PW [33].

These convolutional layers are in charge of extracting features in MobileNet. These two convolutions are executed in a chain, first the **DW** and then the **PW**. The **DW** convolution uses a two-dimensional filter to all input channels. Then the **PW** convolution applies a 1×1 filter to all the **DW** outputs across all channels by extracting the feature map. Figure 5.5 shows the Depthwise Separable convolutions process.

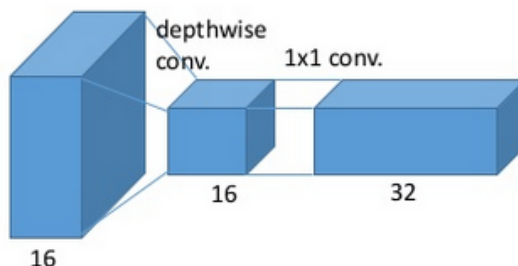


Figure 5.5: Depthwise separable convolution [34].

To understand why MobileNet has been chosen, with respect to the computational cost involved in using it. A comparison is made between the computational cost of a standard convolution explained in the section (4.1), used in any type of architecture such as the VGG-16 and the convolutions used in MobileNet.

To measure the computational cost of a standard convolution, it is first necessary to know the size of the kernels that compose it. Standard convolution kernels can be expressed graphically as shown in figure 5.6, where the number of channels in the image is M , the number of kernels needed is N , which corresponds to the number of output channels, and the size of the kernels is $D_k * D_k$.

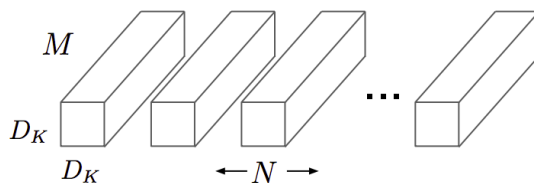


Figure 5.6: Standar Convolution kernels [35].

For a $D_F \cdot D_F$ image the computational cost of standard convolution (CC_{ORD}) is defined in equation 5.1.

$$CC_{ORD} = D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F \quad (5.1)$$

The following is the same way that the computational cost of the standard convolution has been calculated. The study of the computational cost of the **DW** and **PW** is carried out.

The **DW** kernels have the structure shown in figure 5.7. Where M is the number of input channel image and DK is the size of the kernels. It must be noted that the **DW** output size is one, so at the network output it is only one kernel. That means that in the expression of the computational cost 5.2, N is equal to 1.

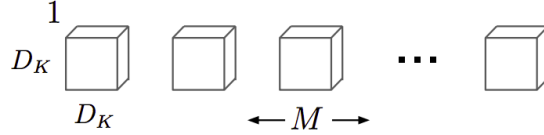


Figure 5.7: Depthwise Convolution kernels [35].

For an input image of $D_F \cdot D_F$ size the computational cost of a Depthwise convolution (CC_{DW}) is defined in equation 5.2:

$$CC_{DW} = D_K^2 \cdot M \cdot D_F^2 \quad (5.2)$$

And the **PW** kernels have the structure shown in figure 5.8. Where N is the number of kernels, M is the number of channels in the input image and the kernel size is 1×1 . This is why this type of convolution is called a point convolution.

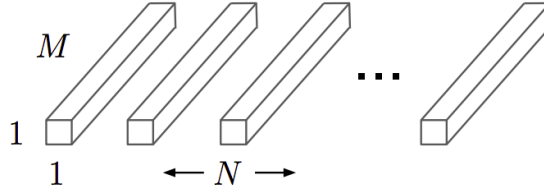


Figure 5.8: Pointwise Convolution kernels [35].

For an image of size $D_F \cdot D_F$, the computational cost of Pointwise convolution (CC_{DP}) is defined in equation 5.3:

$$CC_{DP} = MN \cdot D_F \cdot D_F \quad (5.3)$$

The combination of the computational cost of **DW** and **PW** is called depthwise separable with-convolution, and it was first named in [35]. So, the separable depthwise with-convolution can be expressed as the sum of **DW** and **PW** (equation 5.4).

$$CC_{DW+DP} = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \quad (5.4)$$

In equation 5.5, there are mathematically compared the computational cost of the separable depthwise convolution and the ordinary convolution. As it can be observed, the computational cost of a ordinary convolution is greater than that of the separable depthwise convolution. This allows the extraction of features to be faster and equally efficient.

$$\frac{CC_{DW+DP}}{CC_{ORD}} = \frac{D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2}{D_K^2 \cdot M \cdot N \cdot D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \quad (5.5)$$

Table 5.1 shows the different layers that form the network and their characteristics. It can be seen that the input data to the network are 300×300 images and that the convolutional layers analysed by the **SSD** algorithm are highlighted in bold in the table 5.1, being 11, 13, 14_2, 15_2, 16_2, 17_2 layer. From these layers the information corresponding to the object detection in the image is extracted. For

this purpose [SSD](#) makes use of the feature maps generated by these layers. The following section explains this process in detail and how the [SSD](#) has been configured.

Table 5.1: Convolution layers.

Type	Filter Shape	Input Size
Conv_0/S2	3x3x3x32	300 x 300 x 3
DW_Conv_1/S1	3x3x32 dw	150x150x32
Conv_1/S1	1x1x32x64	150x150x32
DW_Conv_2/S2	3x3x64dw	150x150x64
Conv_2/S1	1x1x64x128	75x75x64
DW_Conv_3/S1	3x3x128dw	75x75x128
Conv_3/S1	1x1x64x128	75x75x128
DW_Conv_4/S2	3x3x128dw	75x75x128
Conv_4/S1	1x1x128x256	38x38x128
DW_Conv_5/S1	3x3x256dw	38x38x256
Conv_5/S1	1x1x256x256	38x38x256
DW_Conv_6/2	3x3x256dw	38x38x256
Conv_6/S1	1x1x256x512	19x19x256
DW_Conv_7/S1	3x3x512 dw	19x19x512
Conv_7/S1	1x1x512x512	19x19x512
DW_Conv_8/S1	3x3x512 dw	19x19x512
Conv_8/S1	1x1x512x512	19x19x512
DW_Conv_9/S1	3x3x512 dw	19x19x512
Conv_9/S1	1x1x512x512	19x19x512
DW_Conv_10/S1	3x3x512 dw	19x19x512
Conv_10/S1	1x1x512x512	19x19x512
DW_Conv_11/S1	3x3x512 dw	19x19x512
Conv_11/S1	1x1x512x512	19x19x512
DW_Conv_12/S2	3x3x512 dw	19x19x512
Conv_12/S1	1x1x512x1024	10x10x512
DW_Conv_13/S1	3x3x1024 dw	10x10x1024
Conv_13/S1	1x1x1024x1024	10x10x1024
Conv_14_1/S1	1x1x1024x256	10x10x1024
Conv_14_2/S2	3x3x256x512	10x10x256
Conv_15_1/S1	1x1x512x128	5x5x512
Conv_15_2/S2	3x3x128x256	5x5x128
Conv_16_1/S1	1x1x256x128	3x3x256
Conv_16_2/S2	3x3x128x256	2x2x128
Conv_17_1/S1	1x1x256x64	1x1x256
Conv_17_2/S1	3x3x64x128	1x1x64

5.1.2.2 Classification with the SSD

The principle used to detect by the SSD algorithm is known as the MultiBox Detector [26]. The algorithm uses the feature maps generated by the convolutional layers to make detections. Figure 5.9 shows the shape of two feature maps, the left one 8x8 and the right one 4x4. In this TFM the features maps used are those generated by layers 11, 13, 14_2, 15_2, 16_2, 17_2 (table 5.1).

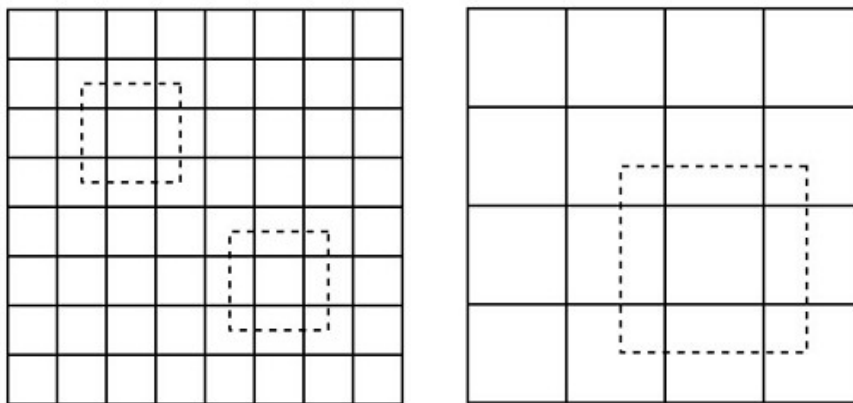


Figure 5.9: Feature maps [26].

The SSD algorithm then adds a number N of bounding boxes for each position of the feature maps. The size, ratio and number of these bounding is defined when designing the network architecture (shown in table 5.2). The size of the bounding box used by the SSD, defines how accurate the system is in detecting, and it can be defined in the SSD algorithm. To do this, first the bounding box scale is calculated for each feature map, as shown in eq. 5.6 where the highest layer is scaled to 0.9 (S_{max}) and the lowest to 0.2 (S_{min}) and all intermediate layers are evenly spaced. Being k the number of feature maps, in this case 6.

$$S_K = S_{min} + \frac{S_{max} - S_{min}}{m - 1}(k - 1), (k \in [1, m]) \quad (5.6)$$

After extracting the scaling of the bounding boxes S_k from each feature map, the SSD extracts the width and height of each bounding box. It uses S_k and the designed ratio values for each feature map. These values can be found in table 5.2. The size of the bounding boxes can be defined with the equations 5.7:

$$\begin{aligned} w_K^a &= S_K \sqrt{a_r} \\ h_K^a &= S_K / \sqrt{a_r} \end{aligned} \quad (5.7)$$

The next step that SSD performs is to center the bounding boxes at each of the feature map positions. To do this, use the size of the feature maps f_k and the positions in the feature maps i and j . The centers of each bounding box can be defined as, $(\frac{i+0.5}{f_k}, \frac{j+0.5}{f_k})$.

Layer	Feature map Size	Ratio	Number of bounding boxes
Conv11	19x19	1,2	3
Conv13	10x10	1,2,3	6
Conv14_2	5x5	1,2,3	6
Conv15_2	3x3	1,2,3	6
Conv16_2	2x2	1,2,3	6
Conv17_2	1x1	1,2,3	6

Table 5.2: SSD parameters.

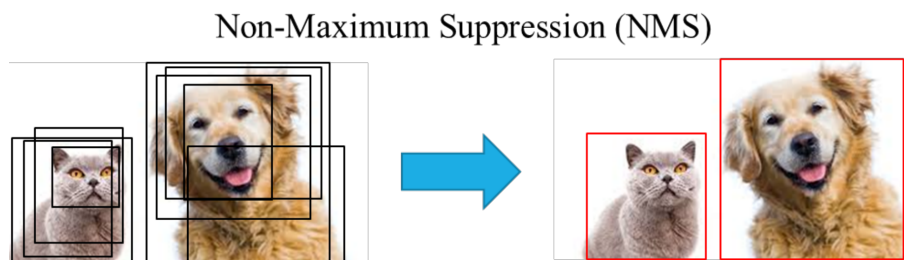
Finally, the [SSD](#) algorithm analyzes each bounding box by extracting a score for each class. The number of bounding boxes the algorithm analyzes is an important parameter. Because if the number of bounding boxes analyzed is high, the system will have a higher computational cost. In this system the number of bounding boxes used can be calculated by the eq. [5.8](#) and by the values of the table [5.2](#). Being FSX the x component of the feature map size, FSY the y component of the feature map size and NB the number of bounding boxes.

$$BoundingBoxes = \sum_{i=FirstLayer}^{LastLayer} FSX_i \cdot FSY_i \cdot NB_i \quad (5.8)$$

According to the data of the designed network and applying equation [5.8](#), the total number of bounding boxes is showed in [5.9](#):

$$BoundingBoxes = (19 \cdot 19 \cdot 3) + (10 \cdot 10 \cdot 6) + (5 \cdot 5 \cdot 6) + (3 \cdot 3 \cdot 6) + (2 \cdot 2 \cdot 6) + (1 \cdot 1 \cdot 6) = 1005 \quad (5.9)$$

After analyzing each score generated by all the bounding boxes, so that no values are repeated and only the highest and most adjusted one prevails, the values are processed by a Non-Maximum Suppression (NMS) filter function. Figure [5.10](#) shows how an [NMS](#) function is affected. The [NMS](#) returns the highest score of the detected object. This system allows to represent on the image a bounding box of the detections with the score detected by the [SSD](#) algorithm.

Figure 5.10: Example of the effect of the Non-Maximum Suppression filter [\[36\]](#).

5.1.3 Data Post-Processing

Once the raw information is obtained from the MobileNet-[SSD](#), it is necessary to filter and adapt it, since it returns a large amount of values that are not necessary for this work. The network is using a pre-trained tensorflow system. To which it has been made a training for the detection of people. This system can

locate other elements in the images, such as dogs, cats, etc. For that reason all those detections that do not have utility are filtered. And only the detections of humans are used. In addition, the network gives a margin of precision of its predictions. If this margin does not exceed a threshold of 25%, this detection is neglected. Finally, once the information is already correct, it is reorganized in dictionaries and saved in the common buffer "*Bounding Boxes Buffer*".

5.2 People tracking and counting

In this section, it is explained in detail the people tracking and counting algorithm. The red colored part of the figure 5.11 shows the blocks corresponding to this stage.

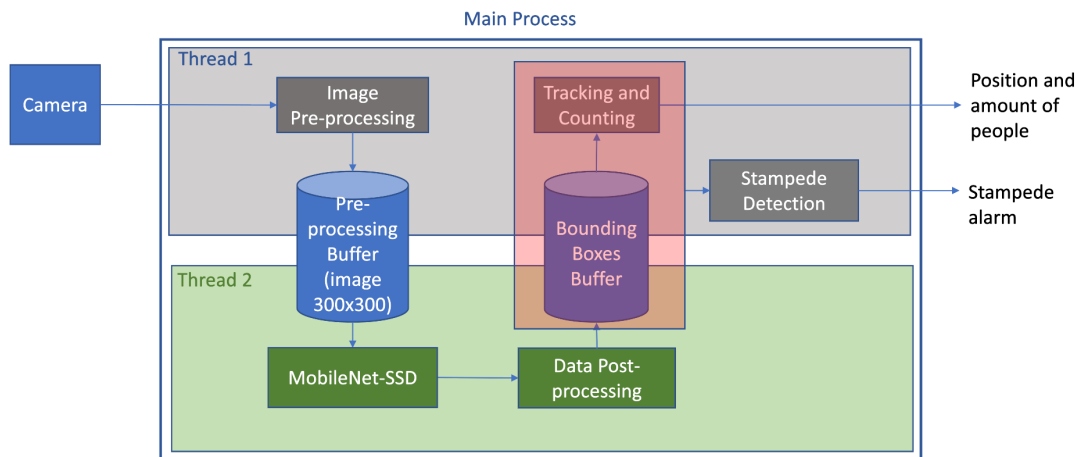


Figure 5.11: Modules of the tracking system highlighted in red.

Because there are situations where detection systems are not able to detect. A tracking system has been designed, which apart from generating a tracking of the objects detected, allows the system to give robustness to situations such as:

- **Occlusions:** can be defined as the visual phenomenon in which an element interposes itself between the detected object and the sensor used. Due to this, the system stops detecting the object, losing control of the position in which it is located. To solve this problem, in cases where the objects have uniform movements, a tracking system is proposed, which is capable of predicting where it can be. So in the case that an external element interposes itself, the system will be able to predict its position.
- **Objects crossing:** a typical problem in detection, it happens when two similar objects cross in the scene. This can be a problem in cases of identification, because the system can confuse the objects. For this reason, a tracking system that is capable of measuring the trajectory of two moving objects allows this trajectory to be used as an identifier.

A Kalman filter system (4.3)) has been used to carry out the monitoring system. Due to the HW limitations of the work, this type of system is suitable because it offers good results with a low computational cost.

It is important to emphasize that in the scene, as usual, the system must follow multiple people. There are methods that work with a single estimator to predict the position of all the objects. However,

the Kalman filter in its most basic form, as is the case here, is not capable to achieve that. This forces each person in the scene to have their own filter, thus forming a Kalman filter bank. When working in this way, the system is obliged to control the following possible moments of the bank.

- **Tracker creation:** when an object is first detected in the scene, the system must assign a Kalman filter to that new object.
- **Object association:** the system must be able to correctly identify the detections made by the detection system with the already assigned Kalman filters. This means that for each object detected, its Kalman filter must be associated.
- **Kalman filter iteration:** once the measurements of each object have been delivered to each Kalman filter. The iteration of each one of the filters must be done fulfilling the prediction and correction stages. In case an object previously identified does not appear in the image due to an occlusion or a failure of the detector, the Kalman filter can use the estimated values until the object appears again in the scene.
- **Filter elimination:** in the case that a previously identified object has not been detected during a period of time, either because it has disappeared from the scene or because the detector is not able to identify it, the system must be able to finish the estimator associated to that object.

Figure 5.12 shows the tracking system developed in this work. The operation of the system is explained in detail below:

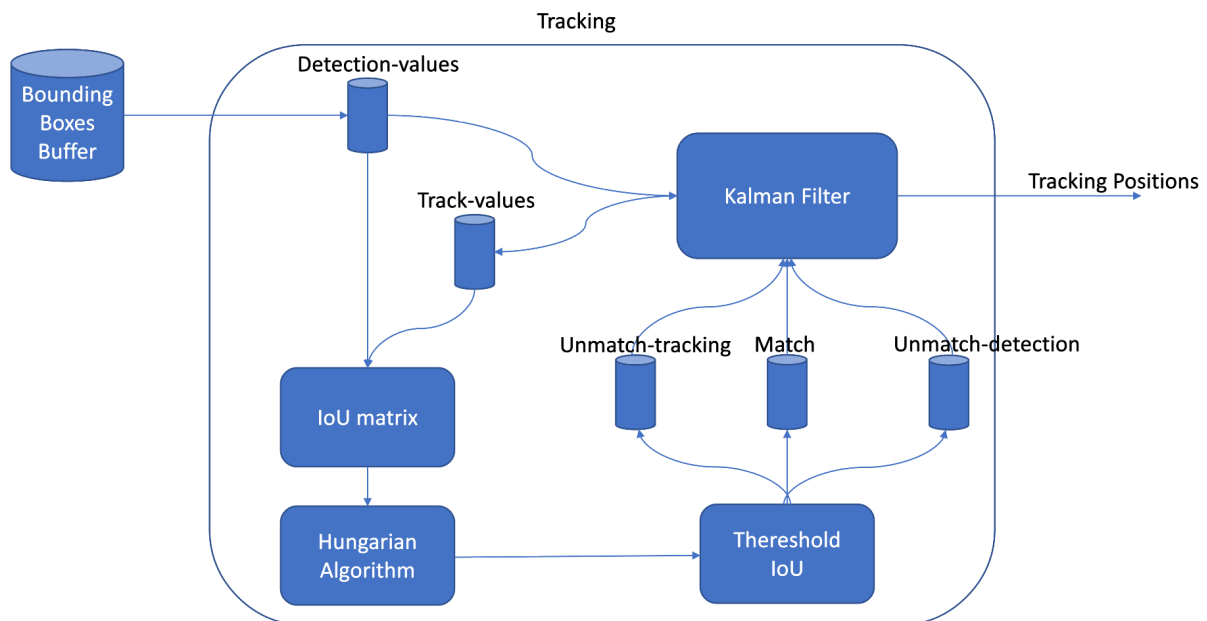


Figure 5.12: Counting and tracking system workflow.

5.2.1 Filter assignment

As mentioned in the previous section. It is necessary to make an association of filters to those new detections in the system. To do this, a list system is used to save the bounding boxes detected and those followed. To identify if a detection is new, the size of the "Detection-values" and "Track-values" lists are compared (5.12). If "I" is larger, it indicates that a new object has been detected in the scene. With that information, a Kalman filter is assigned to the bounding-box of the new detected object.

5.2.2 Object association

The next point corresponds to adding the filters created in the previous step to the detections made. At this point three situations can occur. The first is that the number of elements detected is equal to the number of filters used. The second that the number of detections is greater than the number of filters and the third that the number of filters is greater than the number of detections. To make these comparisons, the size of the "*Detection-values*" and "*Track-Values*" lists are compared. In each of the three cases, the following procedure is used.

1. If the number of detections is equal to the number of filters, this means that the detected objects have already been associated with a Kalman filter. To identify which filter corresponds to each detected element, the IoU is extracted from the bounding boxes of the detected elements with the bounding boxes generated by the Kalman filters after the correction. These percentage values are ordered in a 2D matrix. This matrix is processed by the Hungarian algorithm (also known as the Munkres algorithm [104]), which determines the optimal relationship between the bounding boxes of "*Detection-values*" and "*Track-values*". After the values determined by the Hungarian filter, a filter is performed that discards those relationships with IoUs that do not exceed a threshold. In this way it is avoided that the measurement is associated with the prediction. In this way it is identified which filters correspond to each detection. This relationship is stored in the "*Match*" list.
2. If the number of detections is higher than the number of filters. This means that a new element has been detected. To do this, the same process is carried out as in the case of the number of detections being the same as the number of filters. This process generates the correspondence between all the filters and all the bounding boxes detected except the new one. These correspondences are saved in the "*Match*" list. And the position in "*Detection-values*" of the new detected value is saved in the "*Unmatch-detection*" list, to which a filter must be assigned as indicated in the previous section.
3. If the number of detections is less than the number of filters. That means that one less element has been detected than the one that was being tracked. To do this, the same process is carried out as in the case of the number of detections being the same as the number of filters. This process generates the correspondence of all the detections with the filters except one. This filter that is not assigned, corresponds to the value not detected by the detection system. And the position in "*Track-values*" of the value corresponding to this filter is saved in the "*Unmatch-tracking*" list, the rest of the matches are saved in the "*Match*" list.

5.2.3 Kalman filter iteration

As mentioned in the previous section, the system can be found in three situations. That the system has detected a new object, there is a value in the "*Unmatch-detection*" list, so a filter is associated with that object. In this case the system still cannot make any kind of correction, so it stores the prediction of the bounding box to use it in the next frame.

In this second situation, the system has associated the object detected by the MobileNet-SSD network with its Kalman filter. At this point, in "*Match*", it is indicated which position in "*Detection-values*" corresponds to "*Track-values*". The filter makes the prediction that is stored in "*Track-values*". After that, the association of prediction and measurement ("*Detection-values*") is made. Then, the correction is carried, saving it in the "*Track-values*" position from where the correction of the previous frame had been extracted.

The third case is when the "*Unmatch-tracking*" list has the position of "*Track-values*" object that has not been detected, so the correction cannot be made, that is why only the prediction can be carried out, based on the previous frame values.

5.2.4 Filter elimination

As mentioned in the previous section, if the system does not detect a previously identified object in the image. The system generates a prediction of where it may be. This is because the object may have left the scene or may have become immobile behind an occlusion. The system cannot always be predicting its position. For that reason, when an object is saved in the "*Unmatch-detection*" list a number of N times, the filter corresponding to that object is deleted.

5.3 Crowd stampede detection

In this section, it is described the development of the proposal for stampede detection through OF techniques and entropy levels. As it has been explained in section 4.4, there are two main classical approaches for estimating the OF: LK [101] and GF [32]. For this TFM, it has been used the Farneback approach, due to its capability to provide a dense OF map.

Figure 5.13 shows a general block diagram with the phases of the abnormal behaviour detector, and each of these phases are explained below.

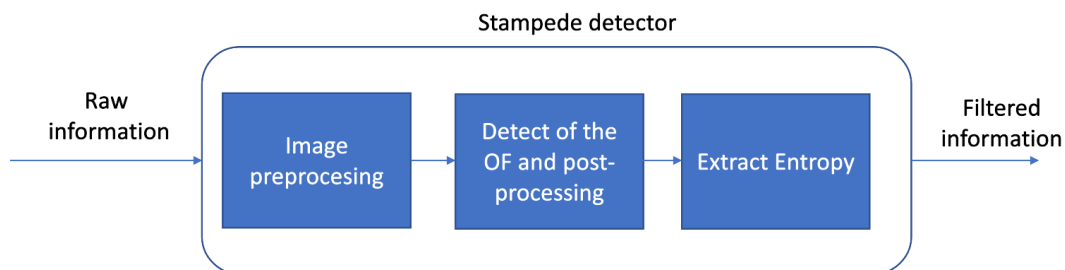


Figure 5.13: General block diagram of the crowd stampede detector.

The first step for stampede analysis is to pre-process the input image. This means, adapting the characteristics of the image to obtain a better result. This pre-processing stage consists of: converting the RGB image to grayscale and applying a Gaussian filter that smooths corners and edges. After that, he image is prepared for extracting the optical flow that is used to create and activity map. In figure 5.14, it is shown an example of an input image before (left) and after (right) the pre-processing stage.

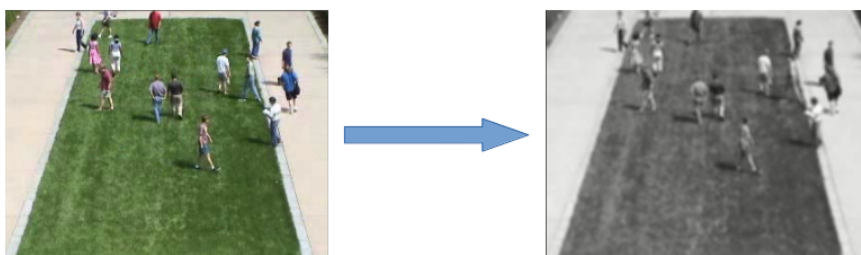


Figure 5.14: Example of image before and after the pre-processing stage.

After the pre-processing, the OF is estimated using the integrated python function that evaluates Farneback OF 4.4.2.

This function returns two values: the angle, which indicates the orientation with which each pixel has changed, and the magnitude, which indicates how much change has occurred at each pixel, both between one frame and the next one. However, only the magnitude is used for this work.

With the values obtained from the magnitude, a 3D matrix is assembled. In which the width and height correspond to the size of the frames in the video sequence. The depth correspond to the frame rate of the video, because activity map is analyzed every second. This value is considered enough for detecting stampedes, without a big increasing the computational cost. This matrix is used to extract the activity map. For this purpose the mean depth of the matrix is calculated. This generates a two-dimensional matrix of width and height, the same as the previous matrix and with values that vary between 0 and 255, due to the fact that it is in grey scale. This matrix represents the areas where the movement has been centered. This matrix is filtered by a median filter that eliminates the salt and pepper effects. The resulting matrix is defined as an activity map. An example of an activity map is shown in figure 5.15, in which the areas painted in white represent the areas where the amount of movement has been greatest.



Figure 5.15: Example of activity Map.

When there is an anomaly in the movement such as a stampede. Such events are reflected in the values of the activity map. Two ways of analysing these sudden changes in the activity map are studied. Through the use of the Temporal Occupancy Variation (TOV) and entropy.

The value of the TOV can be defined as the percentage of image space during an O_t time interval. In order to measure the image space it is used from the activity map corresponding to the current frame A_t and the previous frame A_{t-O_t} . As shown in eq. 5.10 the difference between these two activity maps is carried out generating as a result the TOV.

$$TOV = A(t - O_t) - A(t) \quad (5.10)$$

The other way to estimate the movement is through the entropy [7]. Which can be defined as, the measure of uncertainty of the image values by counting the average amount of information needed to encode the image values. The equation 5.11 shows the mathematical expression for the entropy, where n is the number of separate symbols and p_i is the frequency of the each pixel in the image. When an

anomalous event occurs between one frame and the next, the entropy value increases. This indicates that some abrupt change has occurred in the image.

$$H = - \sum p_i (\log_2 p_i) \quad (5.11)$$

For this work, it was finally decided to evaluate only entropy for stampede detection, because the TOV value in this type of action generates less stable values than entropy. In figure 5.16, two graphs are shown, one corresponding to TOV and the other to entropy. It can be seen that the values corresponding to the TOV are more unstable and difficult to measure than entropy.

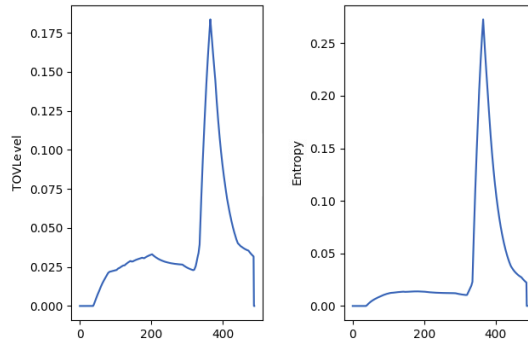


Figure 5.16: Comparison between the values of TOV (on the left) and Entropy (on the right) for a sample sequence with a Stampede.

To conclude, it is necessary to add a threshold value to determine when a stampede is taking place. The threshold value should be large enough not to be exceeded by values corresponding to the basic movement of the image and small enough to detect the stamping as quickly as possible. The threshold has been determined experimentally for each dataset. A set of videos has been reserved for this purpose, which have been used to study the dimensions of entropy and to check which threshold value was the most optimal. Figure 5.17 shows the threshold set to have the largest possible detection ratio.

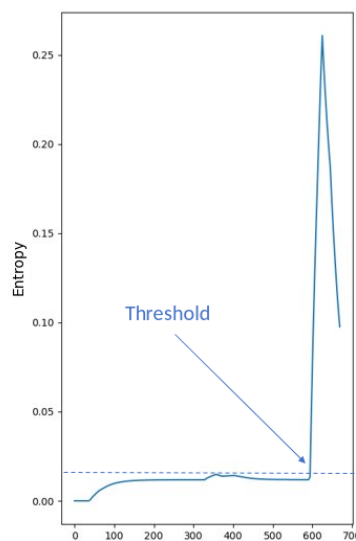


Figure 5.17: Threshold selection.

Chapter 6

Experimental results

This chapter presents the experimental results obtained in this [TFM](#), starting with the explanation of the experimental setup as well as the methods and metrics used to evaluate quantitatively the performance of the system. The exposition and discussion of the results is divided in two parts: first, there are presented the results for people detection, tracking and counting. After that, there are exposed the results related to crowd stampede detection.

In addition, a time study is presented, in which the computational cost of the system in different devices is evaluated. This evaluation is carried out to determine if the system can operate in real-time in an embedded system.

6.1 Experimental setup

In this section, there are first briefly described the datasets used for the experimental evaluation of the developed system. After that, the used metrics are also described.

6.1.1 Datasets for people detection, tracking and counting

For the evaluation of detection and counting of people, two different labelled dataset have been used. These datasets include the ground truth data for each image with the number of people in the scene and their position (bounding box).

The two used datasets contain different situations according to the difficulty in detecting people, being more difficult to detect when there is a larger number of people and occlusions in the image. Although the datasets have been already explained in section [2.3](#), below it is included a brief description of the images and the content of any of them.

- **GBA2016** [\[78\]](#): is a dataset recorded and labelled by the [GEINTRA](#) research group, at the Polytechnic School of the University of Alcalá. The videos show up to 8 people performing different actions. Both people bounding box and the corresponding action are manually labelled. The images in [GBA2016](#) dataset have a resolution of 1920x1080 pixels, at 60 **FPS**. Since as the images size are so large for the system, it has been defined a Region of interest (ROI) in which the people detection is performed. In the figure [6.1](#) it is possible to see the [ROI](#) selected for this dataset.

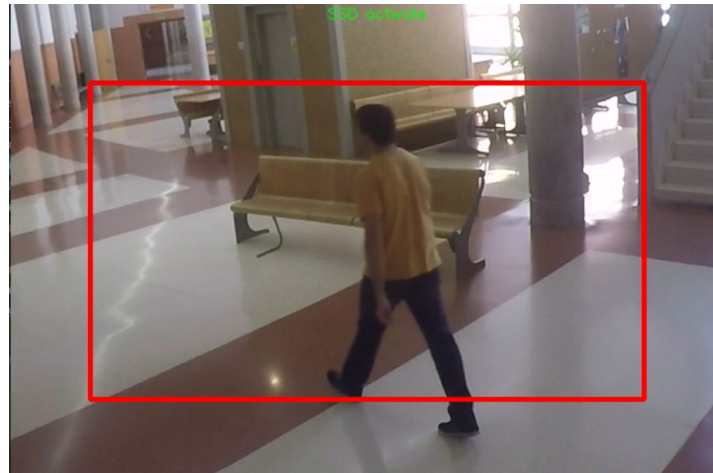


Figure 6.1: ROI used for people detection in GBA2016 dataset.

- **EPFL RGBD pedestrian dataset** [77]: created in the EPFL, is a widely used dataset that includes both RGB and depth data. It is divided into two different scenarios: a laboratory (EPFL-LAB) and a corridor (EPFL-Corridor), with up to 8 people, important occlusions and people entering and leaving the scene. Besides, the dataset is manually labelled, so it has a reliable GT of the events that occur. It is a dataset with a high complexity when detecting due to the events previously reported. It is worth highlighting that in this TFM only RGB videos are used, which are characterized by a rate of 30 FPS and a size of 512x424 pixels. Due to the size of the image, it is not necessary to apply a ROI in this dataset. Figure 6.2 shows the two environments offered by EPFL. Showing in figure 6.2a the corridor and in figure 6.2b the laboratory.



(a) Example of EPFL-CORRIDOR.



(b) Example of EPFL-LAB.

Figure 6.2: Environments forming the EPFL dataset.

6.1.2 Datasets for stampede detection

To evaluate the stampede detection system, it is necessary to clarify when a stampede begins and when it ends. A stampede can be defined as starting when two or more members of a crowd generate an abrupt escape movement. Such movement generates a chain reaction in the other members of the crowd causing the mass of people to run away, either in a scattering movement or in a herd movement. And it is possible to say that the stampede phenomenon ends when, the members of the crowd have run to a quieter area and have decided to stop or until no person is detected in the image. The system, from the moment the stampede starts until it ends, must detect the anomalous situation during the set of frames that the

situation lasts. With all this explained, it is evaluated frame by frame detecting if there is a stampede or not. To be able to carry out this stampede it is necessary to have a set of datasets that contain this information. Two datasets have been used for this purpose: UNM and GBA(stampede).

- **UMN**: is a dataset created and recorded by the University of Minnesota. In it is possible to find both normal and abnormal behaviour of crowds that are recorded in indoor and outdoor environments. The dataset has a total of 11 videos with three different environments. Two of them are outdoors and the other one is indoor. Figure 6.3 shows three scenes, corresponding to each of the three scenarios provided by the dataset.



(a) Outdoor with RGB image in green field.



(b) Indoor with Gray scale color.



(c) Outdoor with RGB image in public square.

Figure 6.3: Sample images correspondent to each of the three scenarios in the UMN dataset.

- **GBA (Stampedes)**: is an extension of the GBA dataset previously described. It is a dataset with a set of 15 videos recorded at the Polytechnic School in the University of Alcalá. This set of videos from the GBA dataset initially show a group of people walking, and after a while a stampede takes place. These videos have been hand-labelled, indicating the frame in which the stampede begins and ends. This dataset allows the evaluation of the system in indoor environments. Figure 6.4 shows the stage and an example of a frame of this dataset of these videos.



Figure 6.4: GBA stampede sample frame.

Within these datasets there are different situations with different complexities. For this reason, the analyzed videos have been classified according to difficulty, in three groups:

- **Easy:** videos with one or two people, without occlusions.
- **Medium:** videos with more than two people and less than five, in which there is some partial occlusions.
- **Hard:** videos with more than 4 people, with continuous occlusions, some of them total occlusions. In addition, there are changes in the profundity in which people move.

Table 6.1 shows the number of used videos from GBA, EPFL-LAB and EPFL-CORRIDOR, divided according to difficulty. As it can be seen, all the videos on the EPFL dataset are in the hard category. This is because they are videos with a high level of occlusion and with more than 4 people on the scene in all the cases. While the videos analyzed in GBA, are Easy and Medium videos, where there are no more than 3 people in any of the videos and the occlusion level is low.

Dataset	Easy	Medium	Hard
EPFL-LAB	-	-	1
EPFL-CORRIDOR	-	-	6
GBA2016	5	6	-

Table 6.1: Summary of the distribution of videos in each of the three levels of difficulty for EPFL and GBA2016 datasets.

6.1.3 Metrics

Metrics can be defined as those measures that are used to quantitatively evaluate the performance of a system. Below, there are described the different metrics used in this work.

To evaluate the people counting system, for each frame, the number of people identified by the system is collected and compared with the number of people given in the labelled GT. These values allow the evaluation of both the count and the tracking system. With regards to the detection, there is used a metric based on the IoU, known as the General Intersection over Union (GIoU), since it allows a more precise measurement of the overlaps of the bounding boxes. The use of the GIoU [105] is due to the fact that the IoU has two weaknesses which are detrimental to the evaluation of the system, and which make

it less accurate. Furthermore, there are also used the usual metrics: precision, accuracy and recall. Each of these metrics is explained below.

6.1.3.1 IOU and GIoU for people detection

The **IoU** is one of the most used methods to measure the good behaviour of detection systems. In particular, those detection systems that generate bounding boxes in the detected elements, such as **YOLO**, **SSD**, etc. **IoU** consists of comparing the bounding box predicted by the system with the bounding boxes defined by the **GT**. This method gives as a result, how much percentage of overlap there is between the predicted bounding box and the bounding box of the **GT**. Figure 6.5 shows graphically how the **IoU** is divided between the total area of the two bounding boxes.


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


Figure 6.5: IOU graphical representation [37].

The **IoU** is scale invariant, that means that given two bounding boxes A and B, the result is the same independent of the physical space that those two bounding boxes occupy. For a system in which all the elements are at the same depth and do not vary, the **IoU** generates very positive results, but when the detected individuals move in all directions, perspective affects detection.

The second weakness is that the **IoU** does not reflect if two bounding boxes are near or far from each other, since in case that $|a \cap B| = 0$, the **IoU**=0.

At the end, these two weaknesses can be translated as a problem by not interpreting the perspective of the recording, having a flat system that does not evaluate the 3D of the real world. This is why the work [106], propose the use of the **GIoU**. It is an algorithm that is based on the **IoU** but introduces rectifications to the weaknesses mentioned above, making an evaluation system that takes into account the depth of the image.

The **GIoU** algorithm is extracted directly from the above-mentioned paper and it is shown below:

Algoritmo 6.1: IoU and GIoU.

input: Predicted B^p and GT B^g bounding box coordinates:

$$B^p = (x_1^p, y_1^p, x_2^p, y_2^p), \quad B^g = (x_1^g, y_1^g, x_2^g, y_2^g),$$

1 For the predicted box B^p , ensuring $x_2^p > x_1^p$ and $y_2^p > y_1^p$:

$$\hat{x}_1^p = \min(x_1^p, x_2^p), \quad \hat{x}_2^p = \max(x_1^p, x_2^p),$$

$$\hat{y}_1^p = \min(y_1^p, y_2^p), \quad \hat{y}_2^p = \max(y_1^p, y_2^p),$$

2 Calculating area of B^g : $A^g = (x_2^g - x_1^g)x(y_2^g - y_1^g)$.

3 Calculating area of B^p : $A^p = (\hat{x}_2^p - \hat{x}_1^p)x(\hat{y}_2^p - \hat{y}_1^p)$.

4 Calculating intersection I between B^p and B^g :

$$(x_1^I = \max(\hat{x}_1^p, x_1^g), \quad (x_2^I = \min(\hat{x}_2^p, x_2^g),$$

$$(y_1^I = \max(\hat{y}_1^p, y_1^g), \quad (y_2^I = \min(\hat{y}_2^p, y_2^g),$$

$$I = \begin{cases} (x_2^I - x_1^I)x(y_2^I - y_1^I) & \text{if } x_2^I > x_1^I, y_2^I > y_1^I, \\ 0 & \text{otherwise} \end{cases}$$

5 Finding the coordinate of smallest enclosing box B^c :

$$(x_1^c = \min(\hat{x}_1^p, x_1^g), \quad (x_2^c = \max(\hat{x}_2^p, x_2^g).$$

$$(y_1^c = \min(\hat{y}_1^p, y_1^g), \quad (y_2^c = \max(\hat{y}_2^p, y_2^g).$$

6 Calculating area of B^c : $A^c = (x_2^c - x_1^c)x(y_2^c - y_1^c)$

7 $IoU = \frac{I}{U}$, where $U = A^p + A^g - I$

8 $GIoU = IoU - \frac{A^c - v}{A^c}$

The algorithm also allows to work with evaluation systems similar to those of **IoU**, because it also returns a generated percentage of the detected bounding-box overlap and the Bounding-box of the **GT**. This percentage can be interpreted in the same way that the **IoU** return is interpreted. By comparing with a threshold, choosing which values are valid and which are not.

6.1.3.2 Precision, accuracy and recall

In order to understand precision, accuracy and recall metrics it is first necessary to explain how to account for the successes and failures generated by the system. For this purpose, the nomenclature used is as follows.

- **True Positive (TP)**: number of solutions that the system declares positive and that are truly positive.
- **False Positive (FP)**: number of solutions that the system declares positive and that are actually negative.
- **True Negative (TN)**: number of solutions that the system declares negative and that are actually negative.
- **True Negative (FN)**: number of solutions that the system declares negative and that are actually positive.

From these results it is possible to extract metrics from the system, or in other words, to determine how well or how badly the system works by evaluating certain characteristics. For this work, the characteristics studied are:

- **Positive Predictive Value (PPV) or precision:** proportion of truly positive cases among the positive cases detected by the test. It is a fundamental value that depends on the prevalence, an index that is independent of the quality of the test. The mathematical definition is shown in equation 6.1:

$$Precision = \frac{TP}{(TP + FP)} \quad (6.1)$$

- **Recall:** proportion of positive cases that are well detected by the test. The mathematical definition is shown in equation 6.2:

$$Recall = \frac{TP}{(TP + FN)} \quad (6.2)$$

- **Accuracy:** is the ratio of true results (both true positives (TP) and true negatives (TN)) among the total number of cases examined (TP, TN, FP, FN). The mathematical definition is shown in equation 6.3:

$$Accuracy = \frac{TP + TN}{(TP + TN + FN + FP)} \quad (6.3)$$

In the figure 6.6, it is possible to see a graphical representation of the previously defined metrics.

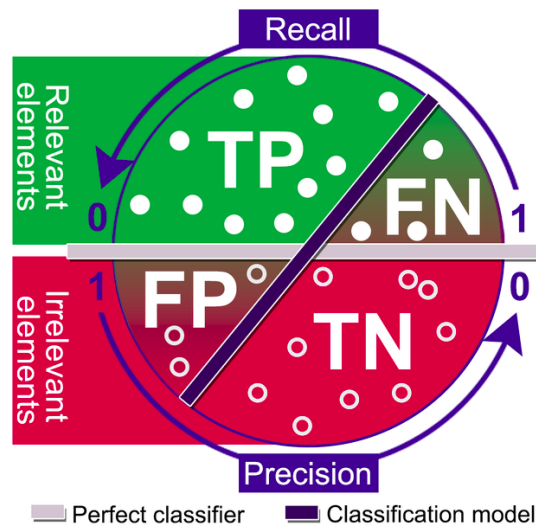


Figure 6.6: Difference between precision and recall [38].

6.2 Results

In this section, there are presented the experimental results obtained in this TFM. The results are divided in three sections: first there are described the results obtained for people detection and counting. Then, there are shown the results related to stampede detection. Finally, the computational cost of the developed algorithms is also analyzed.

6.2.1 People detection

This section evaluates the ability of the system to detect people in the two previously explained datasets: the EPFL RGBD pedestrian dataset and the GBA2016 dataset. The following, it is presented a qualitative evaluation in the aforementioned datasets, and after that, there are described the quantitative results.

6.2.1.1 Qualitative evaluation

In this dataset there are two main environments, the corridor and the laboratory. Both meet the same conditions of requirement and difficulty so they are evaluated as a whole. This dataset is characterized by a difficulty of the Hard type, in which more than 4 people are in scene, existing overlaps and continuous changes of position. The following images show several frames of the system detected by the use of this dataset.

Figure 6.7 shows two examples of the use of the detection system in the EPFL videos located in the corridor. Figures 6.7 show that it is a narrow environment, conducive to the generation of occlusions. In these dataset videos, a large number of people are shown generating constant occlusions between them. In figures 6.7a and 6.7b, both the overlap and the number of people in the videos can be seen.

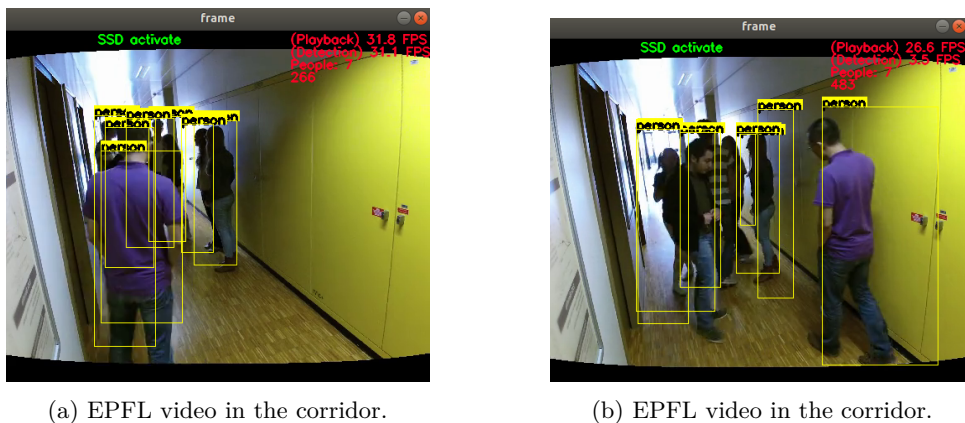


Figure 6.7: Results in two frames from the EPFL-corridor datasets.

Figure 6.8 shows two examples of the use of the detection system in the EPFL videos located in the laboratory. It is shown that it is a wider environment than those shown in the videos of the corridor of the same dataset. However, the members of the video are constantly forcing occlusions, increasing the difficulty of detection. In figure 6.8a and 6.8b these occlusions can be seen.



Figure 6.8: Results in two frames from the EPFL-lab dataset.

The next analyzed dataset is [GBA2016](#). The videos in this dataset have simpler situations regarding detection. It is a wide area, with few individuals and softer occlusions than those shown in [EPFL](#).

Figure 6.9 shows different detection situations, in which from one to three people can be found. The number of overlaps in this dataset is not strong compared to [EPFL](#).

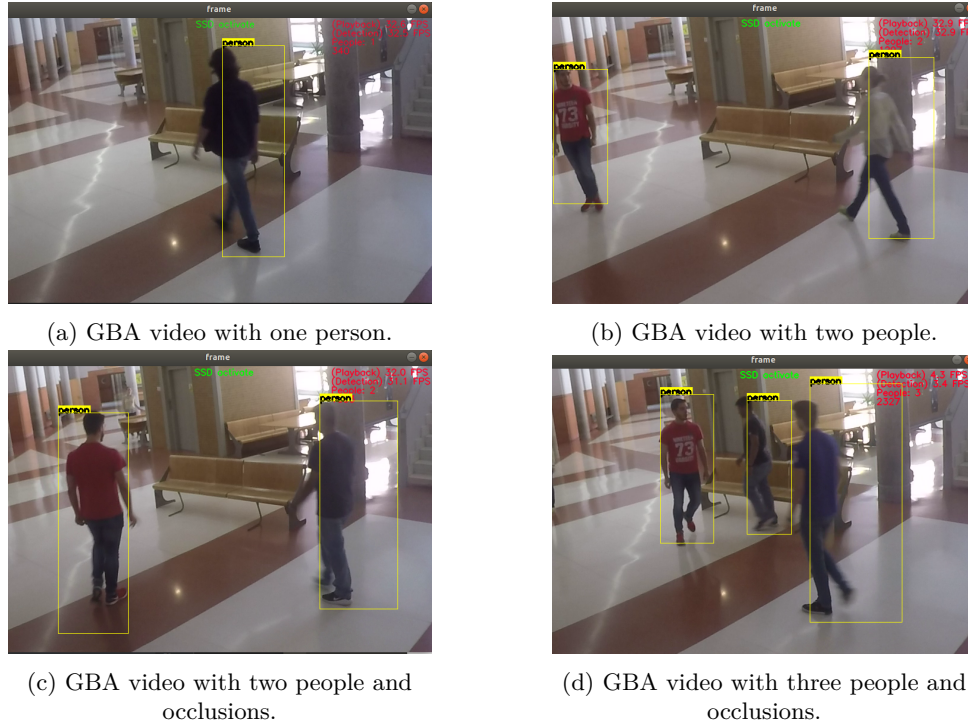


Figure 6.9: People detection results in different images from the GBA2016 dataset.

6.2.1.2 Quantitative evaluation

Table 6.2 shows the results of the metrics of the different datasets used in this work. As it has been explained before, the videos corresponding to EPFL dataset are all considered as difficult, whereas the videos in [GBA2016](#) dataset has been divided in two groups (easy and medium) depending on its difficulty.

Dataset	#frames	TP	TN	FP	FN	Accuracy	Recall	Precision	
EPFL-CORRIDOR	3875	15886	7228	3896	1902	80.71 %	85.67 %	80.33 %	
EPFL-LAB	763	2345	266	353	241	81.46 %	88.14 %	86.02 %	
GBA2016	Easy	4324	2943	1136	19	340	91.93 %	90.13 %	99.11 %
	Medium	23056	17201	24209	699	2740	92.33 %	87.23 %	96.09 %

Table 6.2: Experimental results for the people detection algorithm in the EPFL and GBA2016 datasets.

As it can be seen in the table 6.2, the developed system obtains good results both, in medium and easy environments. it can be seen that the system works well in both medium and easy difficulty environments. This is verified by observing the results given for the [GBA2016](#) dataset, with an accuracy over 91%, and a precision over 96%, with high recall values (around 90%). Regarding the difficult sequences in the [EPFL](#) dataset, the accuracy is over the 81 % for both scenarios. It is worth highlighting that most of the error are related to highly occluded people, which is difficult to detect even for an human observant.

To put these results into context, the results of other state-of-the-art works have been analyzed. For a fair comparison, the work described in [107] has been chosen, as it is the only one that provides results using only RGB information (without using depth). This work only provides the precision, that is of 80 % for RGB, comparable to those results of the proposal. Furthermore, adding depth data only allows improving the precision up to an 84.93 %.

6.2.2 Stampede detection

This subsection evaluates the results obtained by the stampede detection system. For this purpose, two datasets have been used: the UMN dataset, which is used as a basic dataset for stampede detection, and the GBA stampede dataset, which allows to observe how the system works in a totally different situation and with larger images. As in the previous section, the results are divided in two parts. First it is carried out a qualitative evaluation, and after that, there are shown the quantitative results.

6.2.2.1 Qualitative analysis

Figure 6.10 shows an example of how the developed system works in two of the available scenarios of the UMN dataset. In this figure, it is possible to see the images just before and after the stampede starts. It can also be seen the warning message launched when a stampede is detected. The figures 6.10a and 6.10c (on the left) show the situations in the videos before the stampede, whereas in the figures 6.10b and 6.10d (on the right) the situations after the stampede have been detected.

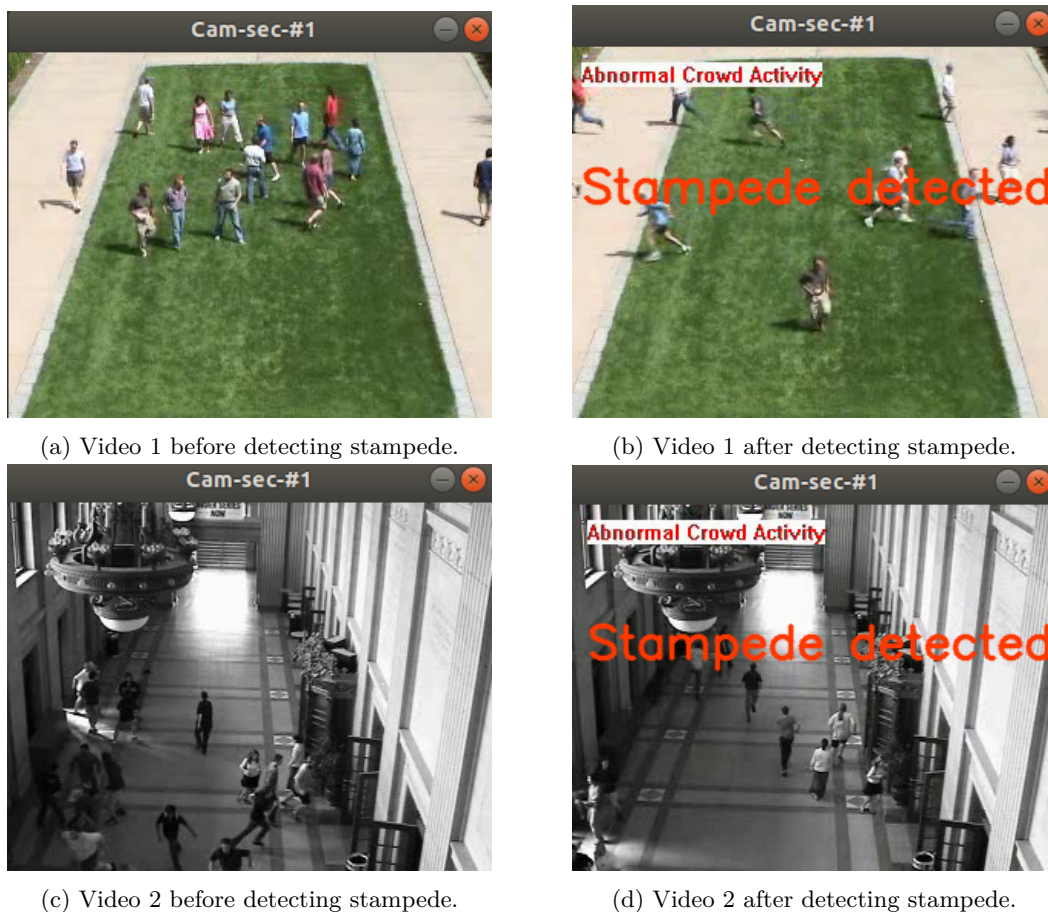


Figure 6.10: Example of the system working in different stampede scenarios.

The UMN dataset consists of a total of 6 videos. To evaluate the stampede module, one of the videos in the dataset has been used to determine the optimal threshold, while the others have been used for the evaluation of the system. The way the system has been evaluated is as follows. By using the **TP**, **TN** it has been determined as a success when the system agrees with the **GT** and has been determined as a failure (**FN,FP**) when it is ahead or behind the **GT**.

In the figures 6.11, the **GT** is shown in the corresponding frames, which is indicated as a red transparent block. And in blue it is possible to observe the entropy response given by the system developed in this **TFM**. The system has a threshold of 0.05 in the value of the entropy derivative, indicating that a stampede is being detected.

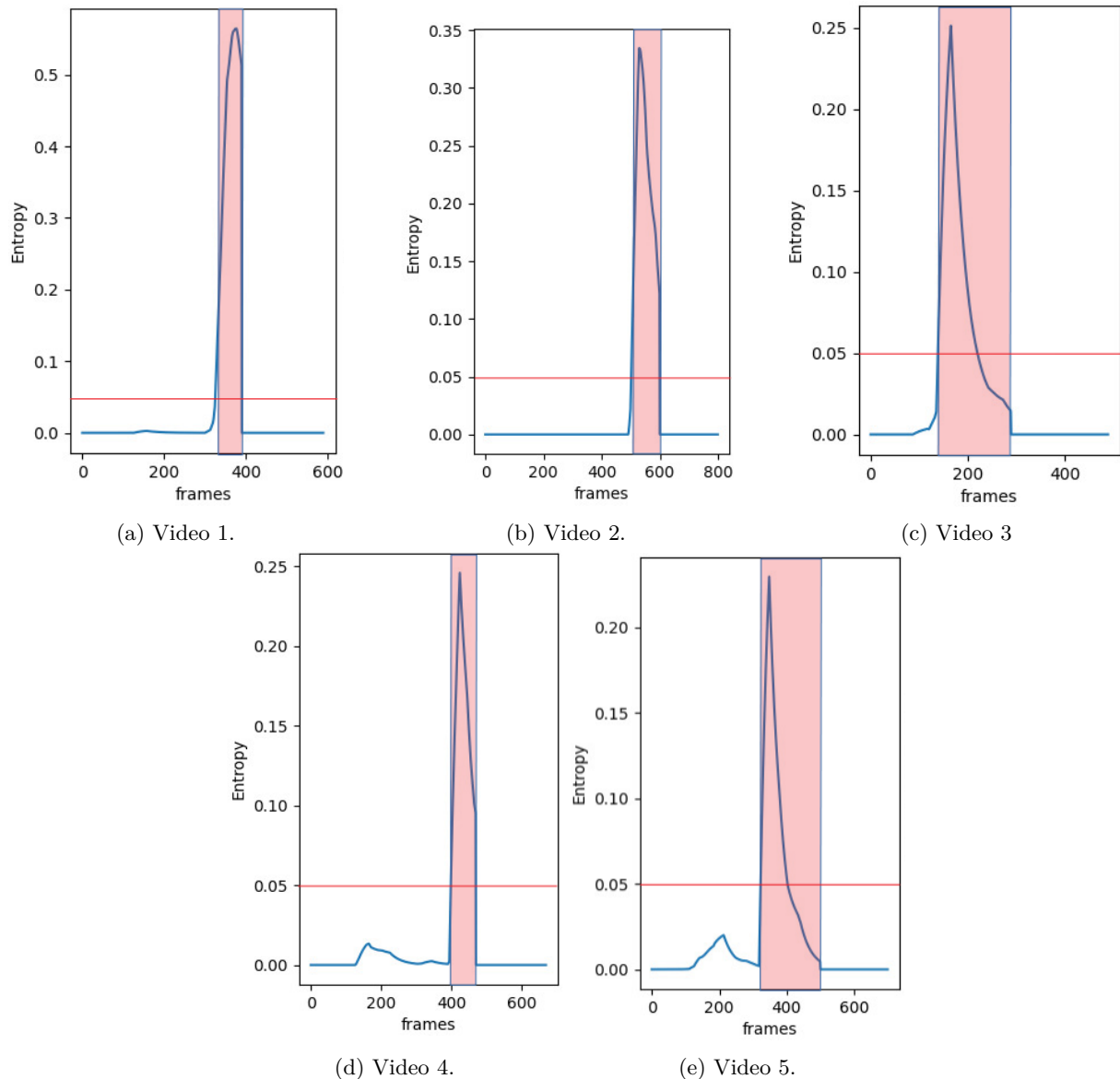


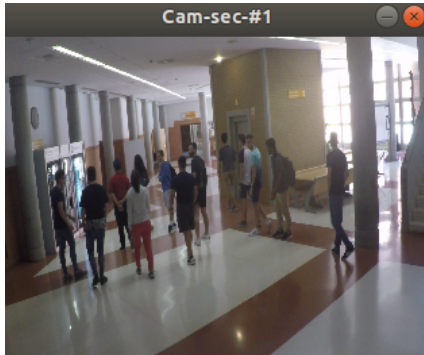
Figure 6.11: Comparative graphs of the **GT** with the stampede detection system.

As can be seen in all the figures, the system is able to detect the stampede earlier than the **GT** does. It is also possible to verify that the system does not fail at the time of a stampede having a 100 % recall for all the cases.

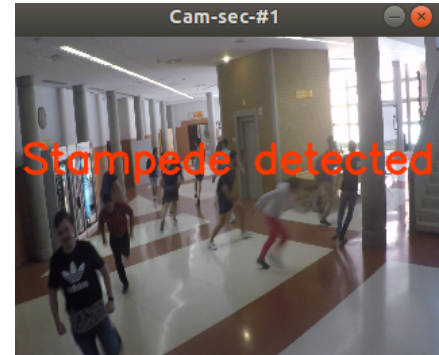
It is worth highlighting that the videos are not manually labeled, but the labels are obtained using a method based on the Hidden Markov algorithm explained in the work [108]. When studying empirically

the videos, it is observed that the system designed for this TFM is faster and more precise than the one used to label the GT.

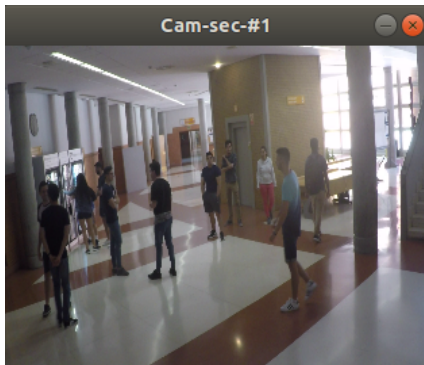
Figure 6.12 shows an example of how the developed system works in two of the available sequences from the GBA dataset. In this figure, it is possible to see the images just before and after the stampede starts. In these figures it can be seen that, when the computed entropy exceeds the threshold, it appears a warning message indicating that a stampede is taking place. The figures on the left 6.12a, 6.12c show the situations in the videos before the stampede and in the figures on the right 6.12b, 6.12d the situations once the stampede has been detected.



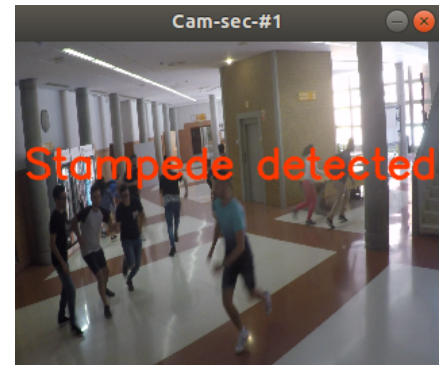
(a) Video 1 before detecting stampede.



(b) Video 1 after detecting stampede.



(c) Video 2 before detecting stampede.



(d) Video 2 after detecting stampede.

Figure 6.12: Example of the system working in different stampede sequences in the GBA stampede dataset.

GBA stampede dataset has a set of 15 videos showing abnormal behaviour in crowds. From these 15 videos, 5 of them are related to other behaviours. The other 10 videos do show situations in which stampedes happen. From the 10 videos, 3 of them have been used to determine threshold and the other 7 to evaluate the behaviour of the system.

In the following figures 6.13 the GT is shown in the corresponding frames, which is indicated as a red transparent block. And in blue it is possible to observe the entropy response given by the system developed in this TFM. The system has a threshold of 0.03 in the value of the entropy derivative, indicating that a stampede is being detected.

In figure 6.13 it can be seen that, unlike UMN, the stampede detector has a small delay in most videos, starting the stampede after the GT. This can be produced due the type of video used, because the image size of GBA is 1920x1080 pixels, and the system needs to reduce the size of the image to be able to process it. This can cause a loss of information in the images. Even so, the system is able to detect stampedes most of the time that they occur.

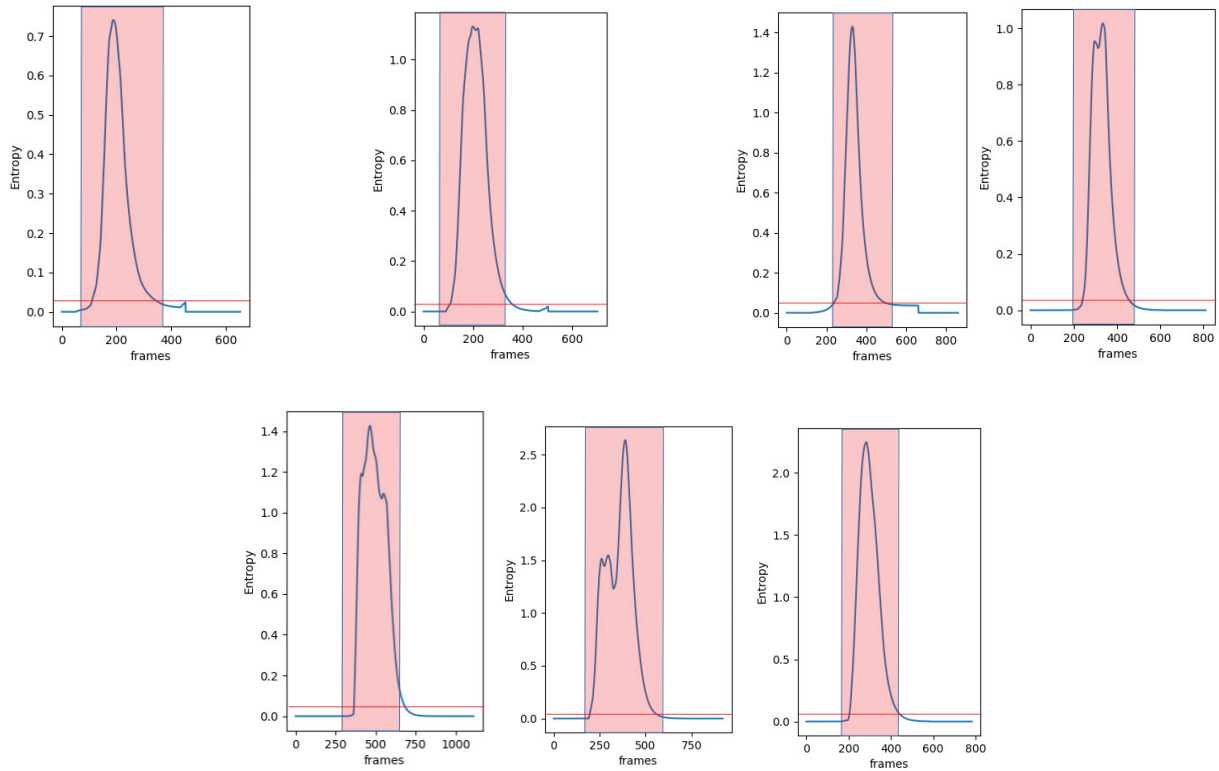


Figure 6.13: Comparative graphs of the GT with the stamped detection system.

6.2.3 Quantitative evaluation

The table 6.3 shows the precision, the accuracy and the recall of the system with respect to the GT given by the dataset.

Sequence	TP	TN	FP	FN	Accuracy	Recall	Precision
Video1 (garden)	69	320	10	0	87.34 %	100 %	97.49 %
Video2 (garden)	121	478	22	0	84.61 %	100 %	96.42 %
Video3 (hall)	161	127	13	0	92.52 %	100 %	95.68 %
Video4 (hall)	62	388	9	0	80.59 %	100 %	96.35 %
Video5 (hall)	203	296	35	0	85.29 %	100 %	93.44 %
Average:					86.07 %	100 %	95.88 %

Table 6.3: Metrics UMN.

As can be seen in the tables the accuracy value is not as high as expected, this is due to the GT used to extract the data. To better understand what is being discussed, the graphs showing the entropy value and the value of the entropy derivative are presented. These graphs also show in red the beginning of the GT stamped. What can be observed is that the system used as a GT, is not as well adapted as the designed system.

The table 6.4 shows the precision, the accuracy and the recall of the system with respect to the GT given by the dataset.

Name	TP	TN	FP	FN	Accuracy	Recall	Precision
Video 1	297	75	0	18	95.38 %	94.28 %	100 %
Video 2	254	79	31	10	89.03 %	96.21 %	89.12 %
Video 3	320	217	0	37	93.55 %	89.63 %	100 %
Video 4	210	198	0	22	94.88 %	90.51 %	100 %
Video 5	363	270	27	20	93.08 %	94.77 %	93.07 %
Video 6	437	197	0	8	98.75 %	98.20 %	100 %
Video 7	212	181	2	17	95.38 %	92.57 %	99.06 %
Average:					94.29 %	93.73 %	97.51 %

Table 6.4: Metrics GBA.

As it can be seen in tables 6.4, the accuracy is over the 89% for all the sequences, while the precision is close to a 100%.

From these results it can be concluded that the developed stampede detection system is a reliable and robust, achieving the objectives of this TFM.

6.2.4 Computational Cost

This section evaluates the computational costs associated with the execution of the system developed in this TFM. To evaluate the computational cost, the execution time in milliseconds (ms) is measured in different hardware platforms: the intel NUC, the Up-Squared and a conventional PC with a GPU. For this purpose, 10000 frames from different datasets is used, generating an average of all of them. In addition, the measurements taken correspond to each of the blocks that make up the system and to the sum total of these. This allows to compare the parallelism relations in each of the systems with the serial processing.

Although the characteristic of th NUC and the Up-Squared system have been already explained in chapter 3, the main features are repeated in this section.

- The intel NUC device is a mini-PC, with Intel Core I7 of 10th and an integrated GPU Iris Plus graphics, DDR4 RAM of 16 GB and a ROM of 512 GB. It also has a Myriad X VPU module inside for video processing and the OpenVINO framework, used for processing AI's, is also installed.
- The Up-Squared is a device that integrates an Intel Atom x7-E3950 microprocessor. An 8 GB memory RAM of the DDR4 type. A 64 GB eMMC ROM. It also has a Myriad X VPU module inside for video processing. Besides, the device has installed the OpenVINO framework, used for AI processing.
- The PC used for the development of this TFM, is a high performance computer, with an Intel i7 9th processor, a GeForce RTX 2080 Ti GPU, 32 GB memory RAM of the DDR4 type and 512 GB of ROM memory. For the development of the TFM was installed the OpenVINO framework.

Table 6.5 shows the results for the computational cost in the previously described platforms. For the Up-Squared2, there have been computed both, the computational cost using only the CPU, an also using the available VPU. Besides, for the PC, there has also been computed the computational cost with and without using the GPU.

Hardware platform		Detection	Tracking and counting	Stampede detection	Total
NUC	CPU	1.83	1.74	11.07	14.65
	GPU	1.90	1.99		14.96
Up-Squared	CPU	12.68	16.86	43.21	72.76
	VPU	4.68	5.43		53.33
PC	CPU	2.17	2.07	15.07	19.32

Table 6.5: Computational cost (in ms) for the different available hardware platforms.

As it can be seen, the **NUC** device can run all parts of the system in real-time (at a frame rate of 30 **FPS**). Either by using the **CPU** or the **GPU**. It can be seen that the **CPU** of this device is more powerful than the **GPU**, this is because the **GPU** is not a separate device like most of the graphic cards on the market. It is an integrated part of the **CPU**. This makes the computing power of this device less powerful. The processing time of the stampede detection module is not as good as that of the detection and tracking modules. This is due to the fact that this module is computationally expensive. But it would not be running frame by frame, but only when a predefined number of people in the scene is exceeded.

For the Up-Squared, it can be seen that the computing times are longer than at the **NUC**. This is due to the fact that the **HW** that composes the Up-Squared has inferior performance compared to the **NUC**. Even so, it can be observed that the system both in detection and tracking is able to work in real-time at 30 **FPS**. For this device, there have been obtained the execution times using only the **CPU**, and also using the **VPU**. When only the **CPU** is used, it is observed that the processing times increase with respect to the use of the **VPU**, being three times greater. It is also observed that when the **VPU** is used for detection, the tracking and counting system continues to be processed by the **CPU**, but the value is lower, this is because the **CPU** is less overloaded and can use more resources for processing the tracking and counting. Although the highest computational cost is observed in stampede detection as mentioned before, this system would not execute every frame, so the real time requirement is only affected during the frames that are analyzed for possible stampedes. Comparing the temporary values of the **NUC** with those of the **PC**. It can be seen that the values obtained by means of the **NUC** are better than those of the **PC** using only the **CPU**. This can be due to several causes. One is that the **CPU** used in the **NUC**, being of a later generation than the **CPU** of the **PC**, has more computing power. Or another option may be that the **HW** of the **NUC** design is more optimized than that the one of the **PC**.

Chapter 7

Conclusion and future issues of research

7.1 Conclusions

In this work, there has been designed, implemented and evaluated an intelligent video surveillance system, which is implemented by three modules. A detection module formed from a MobileNet-SSD network, which is able to detect people on the scene. A tracking and counting module, which collects the information given by the MobileNet-SSD network and through a Kalman filter bank is able to track and count people. Finally, a module has been designed that is capable of detecting stampedes by means of optical flow algorithms.

During the development of the [TFM](#), there has been carried out a search of different proposals in the literature for achieving the objectives. Furthermore, there have also been analyzed several [HW](#) platforms.

The system has been implemented on two different devices: the [NUC](#) and the UpSquared. In them there is specialized [HW](#) that allows the implementation of an [AI](#) system.

The developed system has been exhaustively evaluated in different datasets in order to test its behaviour. For this experimental evaluation, accuracy, precision and recall metrics have been used.

For the detection, tracking and counting module, in the worst case (with a high degree partial and total occlusions), average values of 80% precision, 85% recall and 82% accuracy have been obtained. And the values obtained in the best of cases reach 99% precision, 90% recall and 91% accuracy.

For the stampede detection module, values of more than 90% are observed in the different datasets used for all the metrics. Having a system that is able to detect stampedes in a robust and reliable way.

Regarding the temporal analysis, it is verified that the module of detection, counting and tracking fulfills the requirements of real-time, in the two selected platforms. The module that in this case consumes more resources is the stampede detection one, which in the case of the UpSquared, exceeds 40ms. But, this module only starts working when more than 5 people are detected.

In summary, the requirements and objectives of the [TFM](#) have been fulfilled.

7.2 Future issues

As shown in this work, the flexibility offered by this type of system is very great. Allowing to have a great flexibility and adaptability to the environment. For that reason a great number of improvements can be done to it that in principle were planned but for reasons of time they could not be implemented. These improvements are shown below.

- Parallelization in a third thread of the stamping module.
- Implementation of the use of the [VPU](#) of the [NUC](#) device.
- Detection of more abnormal behaviour such as falls. In a system like this, which is capable of detecting stampedes, it would be very useful to be able to detect if there is someone on the ground in order to provide help.
- Implementation of a docker for easy installation of the system in any machine. In this way there is no need to worry about downloading libraries or system requirements.
- Use of more Myriad X externally with the [USB](#) neural stick to implement more features to the system without overloading.
- Use of more than one camera in parallel being able to combine two information flows from different perspectives to improve results and make a more robust and reliable system.
- Implementing the MobileNet V2 architecture.

Chapter 8

Budget

This chapter summarizes the different costs that occurred during the project. The cost of hardware and software used during the project and the cost of employee time is analysed. In addition, the cost of taxes and profits is analysed.

8.1 Hardware resources

The different devices used during the development of the project are shown in the table 8.1.

Concept	Price	Quantity	Total
NUC	964,09€	1	964.09€
UpSquared	453,02€	1	453.02€
D-435	168,74€	2	168.74€
High-performance PCs	2600€	1	2600€
HP-pavilion x-360	800€	1	800€
TOTAL			5154.59

Table 8.1: Price of the different devices used in the TFM.

8.2 Software resources

The software used for the development of this design has been free of charge, due to the decision to use open-source tools. Tools like github that are paid, its license has been provided by the Alcalá de Henares university.

Concept	Price	Quantity	Total
Pycharm-community	0€	0	0€
GitHub	0€	0	0€
Git	0€	0	0€
OpenVINO	0€	0	0€
TexStudio	0€	0	0€
TOTAL			0€

Table 8.2: Price of the different software used in the TFM.

8.3 Human resources

The development, testing and validation of the project has been done by a telecommunication engineer. The cost is evaluated by the number of hours invested in the project. The table 8.3 gives that information.

Concept	Price	Quantity	Total
Engineer	100€	500	50000€
TOTAL			50000€

Table 8.3: Hourly rate of human resources TFM.

8.4 Total cost budget

In this section, the table 8.4 is added with the sum of hardware, software and human costs.

Concept	Cost
Hardware cost	5154.59€
Software cost	0€
Human cost	50000€
TOTAL	55154.59€

Table 8.4: Total cost budget.

8.5 Contract expenses

This section includes expenditure on facilities, tax and administrative costs. To estimate these expenses, a percentage of 25% of the total cost budget has been used. The Table 8.5 shows the expenses of the contract.

Concept	Cost
25% of the total cost budget	13788.65€

Table 8.5: Contract expenses.

8.6 Contract earning

The estimated earnings for this TFM have been calculated as a percentage of the total budget, with the estimated percentage being 7% as the table table 8.6 shows.

Concept	Cost
Total sum of cost budget	13788.65€
7% of the total cost budget	4826.02€

Table 8.6: Contract earning.

8.7 Global Budget

In this last section is dedicated to show a summary of the different budgets. The table 8.7 shows the global budget of this work, being.

Concept	Cost
Total cost budget	55154.59€
Contract expenses	13788.65€
Contract earnings	4826.02€
TOTAL (no VAT)	73769.26€
VAT (21%)	15481.55€
GLOBAL BUDGET	89260.80€

Table 8.7: Sum of work costs.

Chapter 9

Tools and resources

This chapter explains the different types of software and hardware used to carry out the work.

9.1 Hardware resources

The hardware resources used in this [TFM](#) are as follows:

- High performance [PC](#) with 16 [GB](#) of [RAM](#) and 512 [GB](#) of storage. Using a 10th generation intel core i7 and an NVIDIA gt force 2080 ti graphics card.
- Intel NUC for AI.
- Up-Squared.
- Camera intel D-435.

9.2 Software resources

More than one software resource has been used for the implementation of the practice.

- PyCharm community: has been used as environment to work in python 3.6.
- Microsoft Office: programs such as Power Point, Excel and Teams have been used to develop multiple task in this work.
- Ubuntu 18.04 LTS: is the Operative system used in this [TFM](#).
- \LaTeX and KbitTex: The two tools used to write this book.
- OpenVino: framework to optimizes the CNN in the devices.
- GIT and GitHub: the version control.

Bibliography

- [1] C. Li, L. Guo, and Y. Hu, "A new method combining hog and kalman filter for video-based human detection and tracking," in *2010 3rd International Congress on Image and Signal Processing*, vol. 1. IEEE, 2010, pp. 290–293.
- [2] X. Liu, D. Campbell, and Z. Guo, "Single image density map estimation based on multi-column cnn and boosting," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 1393–1396.
- [3] Z. Tu, J. Cao, Y. Li, and B. Li, "Msr-cnn: Applying motion salient region based descriptors for action recognition," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 3524–3529.
- [4] S. Pu, T. Song, Y. Zhang, and D. Xie, "Estimation of crowd density in surveillance scenes based on deep convolutional neural network," *Procedia computer science*, vol. 111, pp. 154–159, 2017.
- [5] T. Gupta, V. Nunavath, and S. Roy, "Crowdvas-net: A deep-cnn based framework to detect abnormal crowd-motion behavior in videos for predicting crowd disaster," in *2019 IEEE international conference on systems, Man and cybernetics (SMC)*. IEEE, 2019, pp. 2877–2882.
- [6] A. Musa, M. Rahman, M. Sadi, and M. Rahman, "Crowd reckoning towards preventing the repeat of '2015 hajj pilgrims stampede'," in *2017 2nd International Conference on Electrical & Electronic Engineering (ICEEE)*. IEEE, 2017, pp. 1–4.
- [7] A. Pennisi, D. D. Bloisi, and L. Iocchi, "Online real-time crowd behavior detection in video sequences," *Computer Vision and Image Understanding*, vol. 144, pp. 166–176, 2016.
- [8] "Intel d-435i," "<https://www.intelrealsense.com/depth-camera-d435/>" [Último acceso 16/marzo/2020].
- [9] "Intel d-415i," "<https://www.intelrealsense.com/depth-camera-d415/>" [Último acceso 16/marzo/2020].
- [10] J. Smisek, M. Jancosek, and T. Pajdla, "3d with kinect," in *Consumer depth cameras for computer vision*. Springer, 2013, pp. 3–25.
- [11] "Microsoft kinect v2," "<https://docs.depthkit.tv/docs/kinect-for-windows-v2>" [Último acceso 16/marzo/2020].
- [12] "Steereocam," "<https://www.e-consystems.com/nvidia-cameras/jetson-agx-xavier-cameras/stereo-camera.asp>" [Último acceso 16/marzo/2020].
- [13] "Taraxl," "<https://www.e-consystems.com/3d-usb-stereo-camera-with-nvidia-accelerated-sdk.asp>" [Último acceso 16/marzo/2020].

- [14] “Intel® movidius™ myriad™ x vpu,” ["https://www.movidius.com/myriadx"](https://www.movidius.com/myriadx) [Último acceso 16/marzo/2020].
- [15] “Up squared ai vision development kit,” ["https://marketplace.intel.com/s/offering/a5b3b00000TfbCAAS/up-squared-ai-vision-development-kit?language=en_US"](https://marketplace.intel.com/s/offering/a5b3b00000TfbCAAS/up-squared-ai-vision-development-kit?language=en_US) [Último acceso 16/marzo/2020].
- [16] “Intel® nuc with the ai on pc development kit,” ["https://software.intel.com/en-us/videos/unboxing-the-intel-nuc-with-the-ai-on-pc-development-kit"](https://software.intel.com/en-us/videos/unboxing-the-intel-nuc-with-the-ai-on-pc-development-kit) [Último acceso 16/marzo/2020].
- [17] “Jetson agx xavier developer kit,” ["https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit"](https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit) [Último acceso 16/marzo/2020].
- [18] “Eic-ms-vision-500,” ["https://www.arrow.com/es-mx/products/eic-ms-vision-500/einfochips-limited"](https://www.arrow.com/es-mx/products/eic-ms-vision-500/einfochips-limited) [Último acceso 16/marzo/2020].
- [19] D. Jurić, “A biological and an artificial neuron,” <https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7> [Último acceso 23/septiembre/2020].
- [20] “Neural network explain,” ["https://towardsdatascience.com/how-does-a-neural-network-make-predictions-6740663a63cb"](https://towardsdatascience.com/how-does-a-neural-network-make-predictions-6740663a63cb) [Último acceso 16/marzo/2020].
- [21] “Difference between max pool and avg pool,” <https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks> [Último acceso 23/septiembre/2020].
- [22] R. Nussinov and H. J. Wolfson, “Efficient detection of three-dimensional structural motifs in biological macromolecules by computer vision techniques,” *Proceedings of the National Academy of Sciences*, vol. 88, no. 23, pp. 10 495–10 499, 1991.
- [23] “Description of frcnn,” ["https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e"](https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e) [Último acceso 25/febrero/2020].
- [24] D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, *Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference*. Mit Press, 1996, vol. 8.
- [25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
- [26] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*. Springer, 2016, pp. 21–37.
- [27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [28] “Opencv official web),” <https://docs.opencv.org/latest/index.html> [Último acceso 20/octubre/2020].
- [29] D. Jurić, “Object tracking: Kalman filter with ease,” <https://www.codeproject.com/Articles/865935/Object-Tracking-Kalman-Filter-with-Ease> [Último acceso 23/septiembre/2020].
- [30] “Concept of opticalflow,” ["https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html"](https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html) [Último acceso 6/april/2020].

- [31] “Opencv-kanade-example,” ["https://www.programmingsought.com/article/84994968564/"](https://www.programmingsought.com/article/84994968564/) [Último acceso 23/septiembre/2020].
- [32] G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” in *Scandinavian conference on Image analysis*. Springer, 2003, pp. 363–370.
- [33] A. M. Husein, C. Christopher, A. Gracia, R. Brandlee, and M. H. Hasibuan, “Deep neural networks approach for monitoring vehicles on the highway,” *Sinkron*, vol. 4, no. 2, pp. 163–171, 2020.
- [34] J. Zhang, J. Xu, L. Zhu, K. Zhang, T. Liu, D. Wang, and X. Wang, “An improved mobilenet-ssd algorithm for automatic defect detection on vehicle body paint.”
- [35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [36] “Nms definition,” <https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-%E7%89%A9%E4%BB%B6%E5%81%B5%E6%B8%AC-non-maximum-suppression-nms-aa70c45adffa> [Último acceso 1/noviembre/2013].
- [37] “Iou description with opencv,” ["https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/"](https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/) [Último acceso 16/marzo/2020].
- [38] “Precision-recall comparison,” ["https://commons.wikimedia.org/wiki/File:Precision-Recall_tradeoff.png"](https://commons.wikimedia.org/wiki/File:Precision-Recall_tradeoff.png) [último acceso 10/octubre/2020].
- [39] A. Senior, *An Introduction to Automatic Video Surveillance*. London: Springer London, 2009, pp. 1–9. [Online]. Available: https://doi.org/10.1007/978-1-84882-301-3_1
- [40] R. Mandal and N. Choudhury, “Automatic video surveillance for theft detection in atm machines: An enhanced approach,” in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2016, pp. 2821–2826.
- [41] A. Goyal, S. B. Anandamurthy, P. Dash, S. Acharya, D. Bathla, D. Hicks, A. Bhan, and P. Ranjan, “Automatic border surveillance using machine learning in remote video surveillance systems,” in *Emerging Trends in Electrical, Communications, and Information Technologies*. Springer, 2020, pp. 751–760.
- [42] N. Elassal and J. H. Elder, “Unsupervised crowd counting,” in *Asian Conference on Computer Vision*. Springer, 2016, pp. 329–345.
- [43] M. Marsden, K. McGuinness, S. Little, and N. E. O’Connor, “Resnetcrowd: A residual deep learning architecture for crowd counting, violent behaviour detection and crowd density level classification,” in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2017, pp. 1–7.
- [44] “Geintra-web,” ["http://www.geintra-uah.org/"](http://www.geintra-uah.org/) [Último acceso 16/marzo/2020].
- [45] “Official page of palaemon project,” <https://cordis.europa.eu/project/id/814962/es> [Último acceso 13/octubre/2020].
- [46] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, “Computer vision and deep learning techniques for pedestrian detection and tracking: A survey,” *Neurocomputing*, vol. 300, pp. 17–33, 2018. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S092523121830290X>

- [47] C. Victoria Priscilla and S. P. Agnes Sheila, "Pedestrian detection - a survey," in *Intelligent Computing Paradigm and Cutting-edge Technologies*, L. C. Jain, S.-L. Peng, B. Alhadidi, and S. Pal, Eds. Cham: Springer International Publishing, 2020, pp. 349–358.
- [48] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893.
- [49] C. Zeng and H. Ma, "Robust head-shoulder detection by pca-based multilevel hog-lbp detector for people counting," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 2069–2072.
- [50] D. Thombre, J. Nirmal, and D. Lekha, "Human detection and tracking using image segmentation and kalman filter," in *2009 International Conference on Intelligent Agent & Multi-Agent Systems*. IEEE, 2009, pp. 1–5.
- [51] L. Zhu and K.-H. Wong, "Human tracking and counting using the kinect range sensor based on adaboost and kalman filter," in *International Symposium on Visual Computing*. Springer, 2013, pp. 582–591.
- [52] H. Zhang, C. Reardon, and L. E. Parker, "Real-time multiple human perception with color-depth cameras on a mobile robot," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1429–1441, 2013.
- [53] A. Jalal, S. Kamal, and D. Kim, "A depth video-based human detection and activity recognition using multi-features and embedded hidden markov models for health care monitoring systems." *International Journal of Interactive Multimedia & Artificial Intelligence*, vol. 4, no. 4, 2017.
- [54] B. Choi, C. Meriçli, J. Biswas, and M. Veloso, "Fast human detection for indoor mobile robots using depth images," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1108–1113.
- [55] Q. Tian, B. Zhou, W.-h. Zhao, Y. Wei, and W.-w. Fei, "Human detection using hog features of head and shoulder based on depth map." *JSW*, vol. 8, no. 9, pp. 2223–2230, 2013.
- [56] C. A. Luna, C. Losada-Gutierrez, D. Fuentes-Jimenez, A. Fernandez-Rincon, M. Mazo, and J. Macias-Guarasa, "Robust people detection using depth information from an overhead time-of-flight camera," *Expert Systems with Applications*, vol. 71, pp. 240–256, 2017.
- [57] X. Ren, S. Du, and Y. Zheng, "Parallel rcnn: A deep learning method for people detection using rgb-d images," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 2017, pp. 1–6.
- [58] Z. Li, L. Zhang, Y. Fang, J. Wang, H. Xu, B. Yin, and H. Lu, "Deep people counting with faster r-cnn and correlation tracking," in *Proceedings of the International Conference on Internet Multimedia Computing and Service*, 2016, pp. 57–60.
- [59] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017. [Online]. Available: <https://doi.org/10.1109/TPAMI.2016.2577031>
- [60] J. Angelico and K. R. R. Wardani, "Convolutional neural network using kalman filter for human detection and tracking on rgb-d video," *CommIT (Communication and Information Technology) Journal*, vol. 12, no. 2, pp. 105–110, 2018.

- [61] P. Khaire, P. Kumar, and J. Imran, "Combining cnn streams of rgb-d and skeletal data for human activity recognition," *Pattern Recognition Letters*, vol. 115, pp. 107–116, 2018.
- [62] G. Zhang, J. Liu, H. Li, Y. Q. Chen, and L. S. Davis, "Joint human detection and head pose estimation via multistream networks for rgb-d videos," *IEEE Signal Processing Letters*, vol. 24, no. 11, pp. 1666–1670, 2017.
- [63] S. Jaiswal and G. Nandi, "Robust real-time emotion detection system using cnn architecture," *Neural Computing and Applications*, vol. 32, no. 15, pp. 11 253–11 262, 2020.
- [64] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B.-Y. Choi, and T. R. Faughnan, "Real-time human detection as an edge service enabled by a lightweight cnn," in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 125–129.
- [65] D. Fuentes-Jimenez, R. Martin-Lopez, C. Losada-Gutierrez, D. Casillas-Perez, J. Macias-Guarasa, C. A. Luna, and D. Pizarro, "Dpdnet: A robust people detector using deep learning with an overhead depth camera," *Expert Systems with Applications*, vol. 146, p. 113168, 2020.
- [66] "Svm explication," <https://towardsdatascience.com/support/vector-machine-introduction-to/machine/learning-algorithms-934a444fca47> [Último acceso 1/octubre/2020].
- [67] J. Bedo, C. Sanderson, and A. Kowalczyk, "An efficient alternative to svm based recursive feature elimination with applications in natural language processing and bioinformatics," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2006, pp. 170–180.
- [68] N. Smith and M. Gales, "Speech recognition using svms," in *Advances in neural information processing systems*, 2002, pp. 1197–1204.
- [69] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, "Large-scale image classification: Fast feature extraction and svm training," in *CVPR 2011*. IEEE, 2011, pp. 1689–1696.
- [70] J. Sell and P. O'Connor, "The xbox one system on a chip and kinect sensor," *IEEE Micro*, vol. 34, no. 2, pp. 44–53, 2014.
- [71] W. Zhang, L. Tian, C. Li, and H. Li, "A ssd-based crowded pedestrian detection method," in *2018 International Conference on Control, Automation and Information Sciences (ICCAIS)*. IEEE, 2018, pp. 222–226.
- [72] Y. Liu, M. Shi, Q. Zhao, and X. Wang, "Point in, box out: Beyond counting persons in crowds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6469–6478.
- [73] Y.-H. Lai, Y.-C. Chang, and J.-W. Perng, "Development of intelligent human flow density detection system based on sensor fusion," in *Proceedings of the 2020 the 4th International Conference on Innovation in Artificial Intelligence*, 2020, pp. 59–63.
- [74] N. A. N. V. Gohokar, "Crowd density as dynamic texture: Behavior estimation and classification."
- [75] M. Marsden, K. McGuinness, S. Little, and N. E. O'Connor, "Holistic features for real-time crowd behaviour anomaly detection," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 918–922.
- [76] R. Mehran, A. Oyama, and M. Shah, "Abnormal crowd behavior detection using social force model," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 935–942.

- [77] T. Bagautdinov, F. Fleuret, and P. Fua, "Probability occupancy maps for occluded depth images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2829–2837.
- [78] C. Martinez, M. Baptista, C. Losada, M. Marrón, and V. Boggian, "Human action recognition in realistic scenes based on action bank," in *International Work-conference on Bioinformatics and Biomedical Engineering*, 2016, pp. 314–325.
- [79] "Intel d-415i," "<https://gopro.com/es/es/update/hero4>" [Último acceso 03/septiembre/2020].
- [80] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2247–2253, December 2007.
- [81] H. Sarbolandi, D. Lefloch, and A. Kolb, "Kinect range sensing: Structured-light versus time-of-flight kinect," *Computer vision and image understanding*, vol. 139, pp. 1–20, 2015.
- [82] Y. Shi, X. Dong, D. Shi, and Q. Yang, "Human detection using color and depth information by kinect based on the fusion method of decision template," *ICIC express letters. Part B, Applications: an international journal of research and surveys*, vol. 7, no. 7, pp. 1567–1574, 2016.
- [83] T. T. D. Pham, H. T. Nguyen, S. Lee, and C. S. Won, "Moving object detection with kinect v2," in *2016 IEEE international conference on consumer electronics-Asia (ICCE-Asia)*. IEEE, 2016, pp. 1–4.
- [84] "Shave of myriad -x," "https://en.wikichip.org/wiki/movidius/microarchitectures/shave_v2.0" [Último acceso 03/septiembre/2020].
- [85] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, 2018.
- [86] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "Cnn-rnn: A unified framework for multi-label image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2285–2294.
- [87] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.
- [88] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, "Stock price prediction using lstm, rnn and cnn-sliding window model," in *2017 international conference on advances in computing, communications and informatics (icacci)*. IEEE, 2017, pp. 1643–1647.
- [89] D. Jurić, "Difference between cnn and brain," <https://medium.com/@gopalkalpande/biological-inspiration-of-convolutional-neural-network-cnn-9419668898ac> [Último acceso 23/septiembre/2020].
- [90] 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.
- [91] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

- [92] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [93] "Official web page of darknet," ["https://github.com/AlexeyAB/darknet"](https://github.com/AlexeyAB/darknet) [Último acceso 16/marzo/2020].
- [94] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.
- [95] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *arXiv preprint arXiv:1603.08029*, 2016.
- [96] M. Roth, G. Hendeby, C. Fritsche, and F. Gustafsson, "The ensemble kalman filter: a signal processing perspective," *EURASIP Journal on Advances in Signal Processing*, vol. 2017, no. 1, p. 56, 2017.
- [97] L. L. Menegaldo, "Real-time muscle state estimation from emg signals during isometric contractions using kalman filters," *Biological Cybernetics*, vol. 111, no. 5-6, pp. 335–346, 2017.
- [98] J. J. Gibson, "James j. gibson." 1967.
- [99] D.-H. Yi, T.-J. Lee, D.-I. Cho *et al.*, "A new localization system for indoor service robots in low luminance and slippery indoor environment using afocal optical flow sensor based sensor fusion," *Sensors*, vol. 18, no. 1, p. 171, 2018.
- [100] Y.-H. Tsai, M.-H. Yang, and M. J. Black, "Video segmentation via object flow," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3899–3908.
- [101] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.
- [102] J.-Y. Bouguet *et al.*, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel corporation*, vol. 5, no. 1-10, p. 4, 2001.
- [103] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [104] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
- [105] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union," June 2019.
- [106] —, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 658–666.
- [107] T. Ophoff, K. Van Beeck, and T. Goedemé, "Improving real-time pedestrian detectors with rgb+ depth fusion," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2018, pp. 1–6.
- [108] "Hidden markov models)," <https://stanford.edu/~jurafsky/slp3/A.pdf> [Último acceso 1/octubre/2020].

