UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE LA TELECOMUNICACIÓN

DESIGN AND IMPLEMENTATION OF A PARKING'S OCCUPATION AUTOMATIC DETECTION SYSTEM WITH TENSORFLOW

VÍCTOR GARCÍA RUBIO

2018

TRABAJO FIN DE GRADO

Título: Desing and implementation of a parking's occupation automatic detection system with TensorFlow.

Autor: Víctor García Rubio

Tutor: Juan Antonio Rodrigo Ferrán

Ponente: José Manuel Menéndez García

Departamento: Departamento de Señales, Sistemas y Radiocomunicaciones.

Grupo de Aplicación de Telecomunicaciones Visuales.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

Suplente:

Fecha de lectura:

Calificación:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

DESIGN AND IMPLEMENTATION OF A PARKING'S OCCUPATION AUTOMATIC DETECTION SYSTEM WITH TENSORFLOW

VÍCTOR GARCÍA RUBIO

2018

Resumen

En este documento se describe el resultado del proyecto cuyo propósito ha sido el desarrollo e implementación de un sistema de detección automática de la ocupación de las plazas de un parking a partir de una imagen de las mismas.

Para ello se ha creado un modelo de inteligencia artificial cuya arquitectura esta basada en redes neuronales y, más concretamente, en redes neuronales onolucionales, un tipo de estructura de redes neuronales multicapa que consistuye el estado del arte en procesado de imagen mediante inteligencia artificial.

Para el correcto funcionamiento del modelo, éste ha sido entrenado con imágenes de las plazas individuales etiquetadas. Para obtener los *datasets* necesarios para dicho entrenamiento, se ha creado un proceso para automatizar la creación de los mismos a partir de las imágenes obtenidas mediante cámaras a cierta altura y archivos CSV con las etiquetas obtenidas manualmente para cada foto del parking general.

Una vez creados los *datasets* de cada parking, se ha entrenado la arquitectura del modelo para obtener de manera inicial un modelo adaptado a cada parking. Se han analizado de los procesos de entrenamiento para obtener las características de los datos que influían en el rendimiento del modelo, así como los parámetros de entrenamiento a modificar para mejorar la precisión de la red. Después de haber perfeccionado el modelo para tener una alta precisión para su parking particular, se han aplicado técnicas de aprendizaje profundo como el *transfer learning* para finalmente obtener un modelo general adaptable a cualquier parking.

Tras obtener este modelo general, se ha comparado su rendimiento frente a otros métodos que conforman el estado del arte en detección de la ocupación de plazas de parkings.

Finalmente, se han mostrado las conclusiones obtenidas del proyecto, así como las futuras líneas de trabajo, los impactos ambientales, económicos, sociales y éticos más destacados y el presupuesto total del proyecto.

Palabras clave

TensorFlow, Keras, Inteligencia artificial, Procesado de imagen, Python, OpenCV

Summary

In this document, the result of a project whose purpose has been the development and implementation of a parking's occupation automatic detection from a general parking's image is described.

To do so, an artificial intelligence model has been created. Its architecture is based on neural networks. More specifically, on convolutional neural networks, a neural network multilayer structure which constitutes the state of the art in image processing with deep learning.

In order to achieve the correct functioning of the model, it has been training with parking lot images previously labelled. To obtain the datasets needed for the training phase, an automatization process has been created to generate them from images obtained from a parking camera along with CSV files that storage the hand-labelled status of each parking place.

After the creation of these datasets for each parking, the model has been trained in order to obtain a model adapted to each parking. These training processes has been analyzed with the purpose of acquiring the data features which influence in the model's performance, together with the training's parameters that need to be adjusted.

Once the model has reached a significant accuracy for its particular parking, deep learning techniques such as transfer learning has been applied with the purpose of obtaining a general model adaptable to any parking.

After this general model has been obtained, its performance has been compared with other methods that compose the state of the art in parking's occupation automatic detection.

Finally, the conclusions extracted from the project has been shown. Moreover, the future lines of work of this project have been listed, along with the project's budget and the economical, social, ethical and environmental aspects influenced by the project.

Keywords

TensorFlow, Keras, Deep learning, Image processing, Python, OpenCV

Agradecimientos

Gracias a mi familia por su apoyo durante toda la carrera y a mi tutor y ponente por haberme dado la oportunidad de empezar a desarrollar mi carrera en el campo de la inteligencia artificial.

Contents

List of Figures

List of Tables

1 INTRODUCTION AND OBJECTIVES

1.1 INTRODUCTION

Automatic occupancy detection in parking lots is a useful tool for an effective management of parking lots. The ability to know the availability of parking spaces can reduce queues and the time to come across with an empty space.

An established technology for the availability measure are ground sensors. This technology stands in the need of the sensors maintenance, whose cost increases proportionally to the parking size.

As an alternative to that, image processing and its combination with deep learning techniques, emerge as an efficient method to leverage the surveillance cameras installed in the majority of parkings to identify free spaces.

In this project, I have developed an alternative solution based on Convolutional Neural Networks (CNN). It consists on a deep neural network trained with parking lot images, able to detect if a parking place is occupied. In order to do that, a camera obtains pictures of the corresponding parking lot. After that, the application will process the image along with the labelled coordinates of each lot, sending them to the trained model and marking them as empty or occupied.

1.2 OBJECTIVES

The main goal of this project is to obtain the occupancy information of a parking lot, such as the location of the empty parking spaces as well as the number of parking places available. This information is extracted from predictions generated by our deep learning model.

This objective involves smaller tasks including:

- Dataset generation from labelled images.
- Creation of a deep learning model architecture based on Convolutional Neural Networks.
- Establishment of a training and test process.
- Adjustment of model's parameters based on the training and test processes results.
- Storage and load of trained models.
- Study of different types of models: individual space model, parking lot model and general model.

2 TECHNOLOGIES AND STATE OF THE ART

This chapter illustrates the different techniques used in this project. The two main technologies employed are: Image processing, especially the OpenCV library, which will be detailed in section 2.1.1, and next in order, deep learning, whose tools: Tensorflow, Cuda and Keras have made possible to train and implement the designed model. These will be described with detail in the three subsections of section 2.2.

2.1 IMAGE PROCESSING

As detailed in [1], image processing is a method to transform an image into digital information and apply operations on it, to get important information or modify the image. Image processing systems treat the image as a two-dimensional matrix in order to extract its numerical properties.

The digital image processing process is described in Figure 2.1:

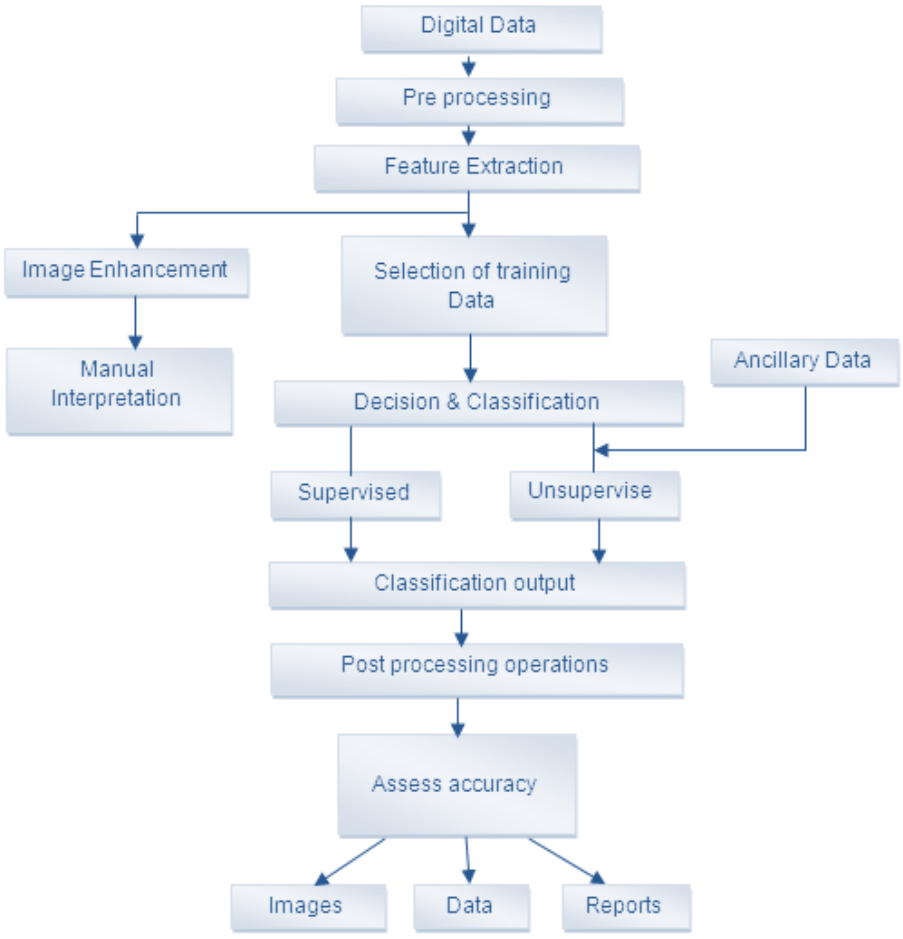

Figure 2.1 Image Preprocessing. Source: [1]

2.1.1 OpenCV

OpenCV is a library of reference in computer vision, image processing and machine learning applied to them. As indicated in [2] and [3], this open source library is under BSD license. OpenCV has interfaces for Java, Python, C++ and C and support the following operative systems: Windows, Linux, MacOS, iOS and Android. The main objective of OpenCV is to develop computational efficient applications.

There are different modules included in OpenCV functionality, some of them are described in the following list and graph:

- Core, imgproc, highgui: Image/video: Input-Output, display and processing.
- Objdetect and features2D: Object detection modules
- Calibd3D:3D and stereo computer vision
- Ml: Machine learning
- GPU: CUDA acceleration

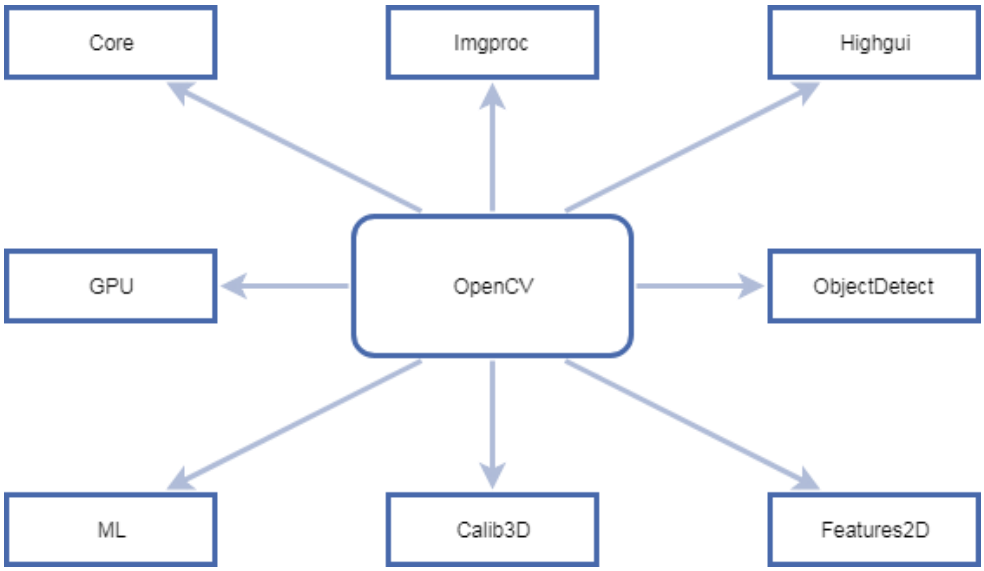

Figure 2.2 OpenCV Modules Scheme

More modules conform the complete library, which are listed in [4].

One of the most interesting features that include the latest releases of OpenCV is the GPU acceleration. More than 200 functions have been modified to take advantage of the computational capabilities of graphic cards from NVIDIA, whose CUDA library will be explained in a future section.

The performance difference has been measured in [4], comparing a Tesla C2050 versus Core i5-760 2.8 GHz, SSE, TBB. The results are shown in Figure 2.3:

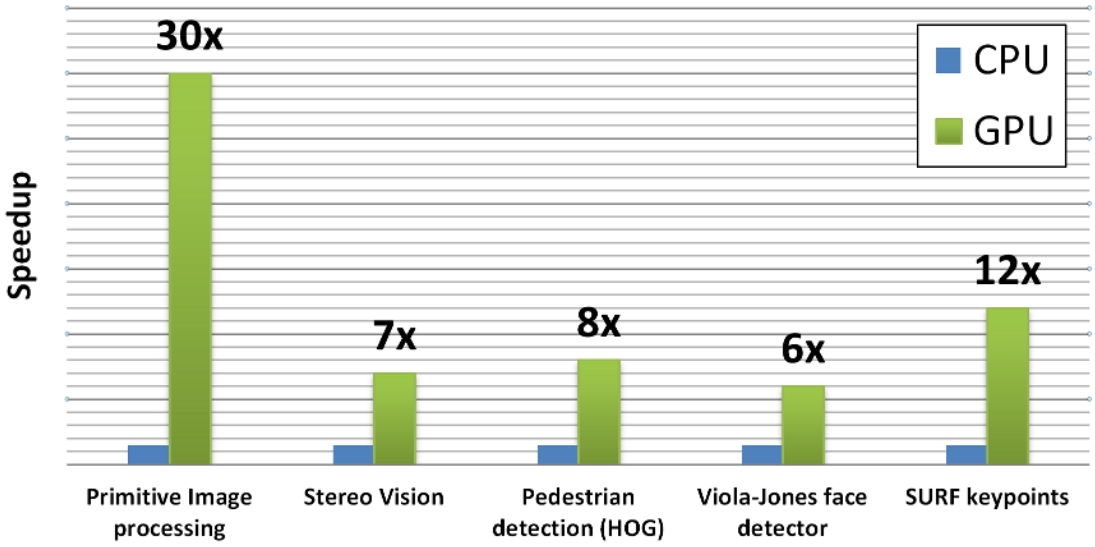

Figure 2.3 CPU vs GPU OpenCV Performance. Source [4]

2.2 DEEP LEARNING

Deep learning is a machine learning technique which employs a mathematical structure called Neural Network. In this technology, the objective is to imitate the different functionalities of the brain.

Deep learning has supposed a revolution in the technological industry, producing advances in different environments, such as the assistants Alexa and Siri or the autonomous cars. It is the state of the art technology in the industry. In the subsequent paragraphs, we will be describing how Neural Networks work, as they are its core structure

2.2.1 Neural networks

Based on the detailed explanation from MIT [5], neural networks consist in a great sum of nodes, each of them densely connected to the rest of this network elements. Normally, these networks are structured in layers. Each layer is composed of a different number of nodes. This number depends of the layer purpose in the network, as well as other variables that will be describe in the next sections.

The nodes are the computational units of neural networks. In them, the input data is combined with different coefficients, named weights, which objective is to modify the input data to the purpose of the neural network, depending on the task the network is trying to learn. The sum of modified input

values is the variable which is introduced to the activation function. This function defines the way the signal goes through the following layer or output.

In Figure 2.4, a detailed scheme of a node is shown:

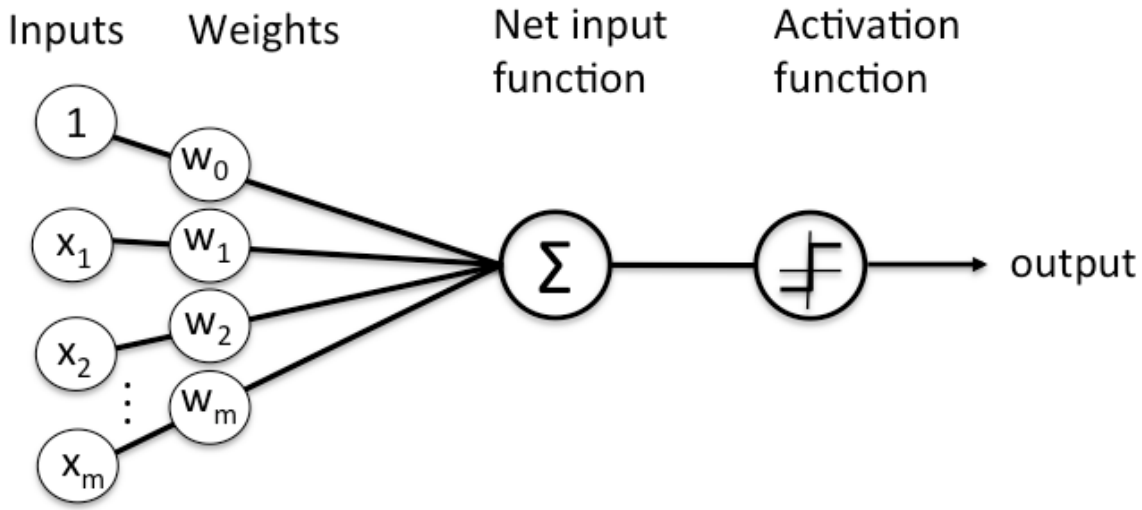

Figure 2.4 Neural Node. Source [6]

Each group of nodes on the same rows in the network's hierarchy conforms a layer. Each layer's output represents the succeeding layer's input. This process starts with the input layer, it extends through all the intermediate or hidden layers and finalises with the output layer, whose output is the objective of the neural network.

An example of a simple neural network is represented in Figure 2.5:

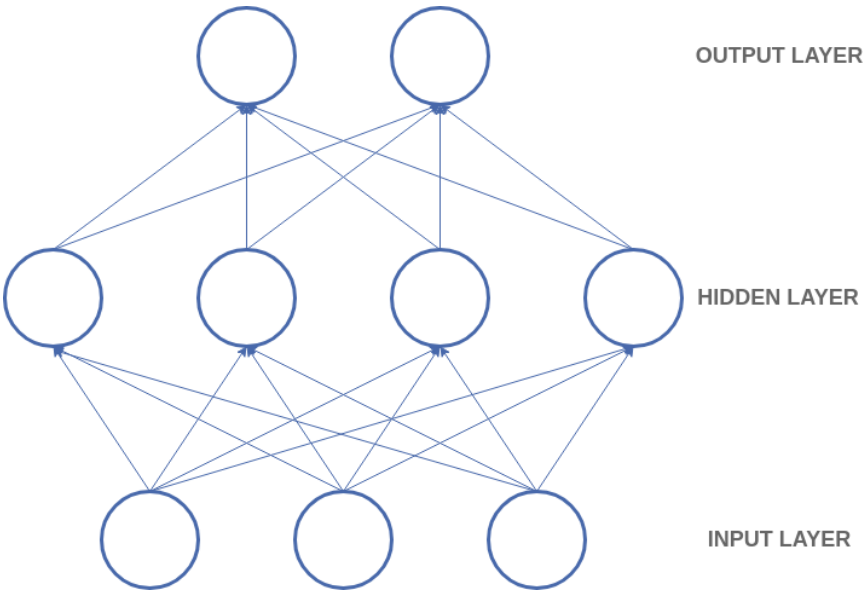

Figure 2.5 Neural Network

2.2.2 Convolutional Neural Networks (CNN)

From the assumption that the input are images, this type of neural networks are able to use more properties than other kinds of neural networks, making it more efficient and reducing the number of parameters needed to train.

Convolutional Neural Networks' main difference with regular networks is how their nodes are disposed. Its layers take advantage of the 3 dimensions of an image: width, height and depth, understanding depth as the number of colour channels of an image. Due to the fact that Convolutional Neural Networks expect always an image as an input, they dispose its neurons as 3D volumes. This disposal differs from the fully connected layers of regular networks, illustrated in Figure 2.5. This reduces significantly the number of weights for each layer. As an example,the first layer of a regular network whose input is a 200x200 RGB image has $200 * 200 * 3 = 120000$ weights. [7]

The third layer of Figure 2.6 illustrates the explanation:

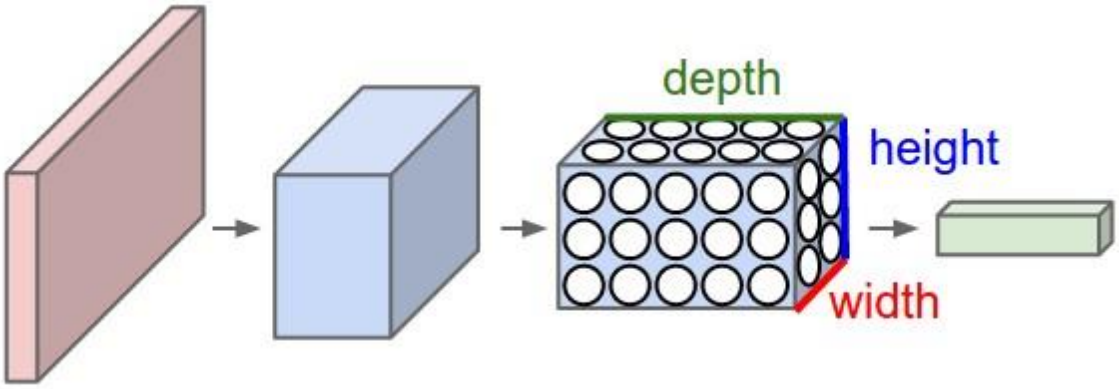

Figure 2.6 Convolutional Neural Network. Source [7]

This type of network is based on four different types of layers: Convolutional, Activation, Pooling and Fully-Connected Layers. In the following paragraphs, we will be defining the function of these layers.

- Convolutional layers compute the values of each pixel in every channel of the input image and applies the dot product between a small region defined by the number of filters used on the layer and the weights implicit on it. The result of applying this layer is a volume with the dimensions: $[Width, Height, Filters]$.

- Activation layers apply a function to the output of convolutional layers. This function may vary depending of the network's purpose. RELU functions are frequently used in intermediate or hidden layers. The RELU function is defined as: $R(z) = \max(0, z)$. The output dimensions of this network are the same as those of the previous layer.

- Pooling layer performs a down sampling to the volume, reducing its size in width and height by a pre-defined factor. Reducing these dimensions in a factor of 2:1 is the most used technique.

This three-layer structure is repeated through the network. Each set of three layers extract features from the input image and reduces its size due to the application of pooling layers.

Finally, an output function is needed to extract the information generated by the CNNs. In object detection algorithms similar to our project, input images are labelled with different classes, regarding the object depicted on them. Classes are the output of our final layer, which takes a one-dimensional vector as input to obtain this significant information. To achieve that, fully connected layers are applied.

Fully connected layers compute, for example, the scores of the different classes in detection algorithms. This type of layers resulting in a volume with dimensions: $[1,1, Classes]$

An example of a neural network with classification purpose that implements convolutional neural network is described in Figure 2.7:

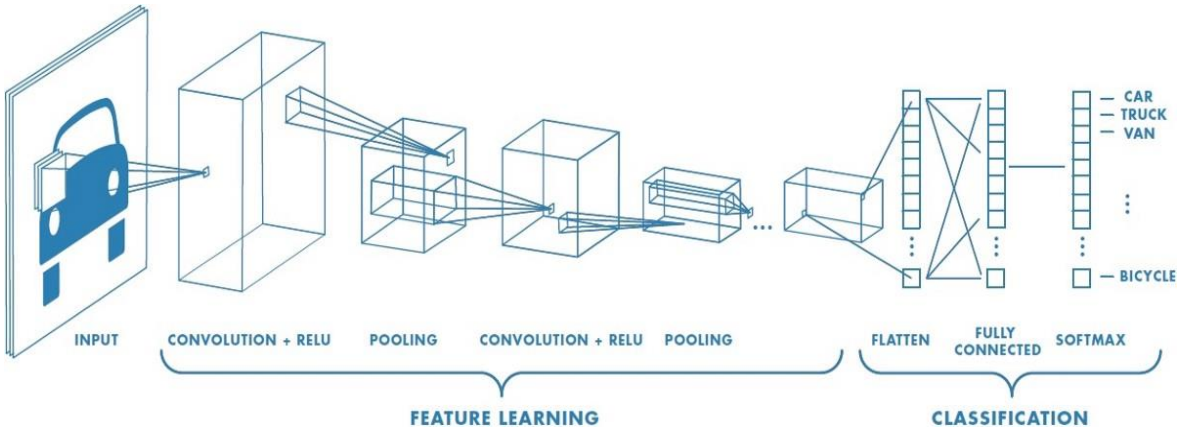

Figure 2.7 Deep Neural Network for Classification. Source [15]

2.2.3 Tensorflow

TensorFlow is an open source software library for numerical computation. It is based on the use of data-flow graphs as the main representation of the computing processes. Mathematical operations are illustrated as the nodes. The edges that join the different operations are called tensors. These tensors are multidimensional arrays which permit the movement and storage of the information obtained in the graph.

Tensorflow is mainly used in Python language, although it has API's for different languages such as C++ or JavaScript. However, the operations related to graphs are performed in high-performance binaries written in C++.

This library has been an important element in the deep learning revolution we are leaving nowadays. It aisles the software developments from the mathematical operations, which make neural networks easier to implement and train.

TensorFlow introduces the idea of tensors applied to machine learning and deep learning applications. Tensors are a generalization of matrices to augmented dimensions. In Tensorflow, tensors are represented as n-dimensional arrays. Figure 2.8 depicts the differences between scalars, vectors, matrices and the concept of tensors.

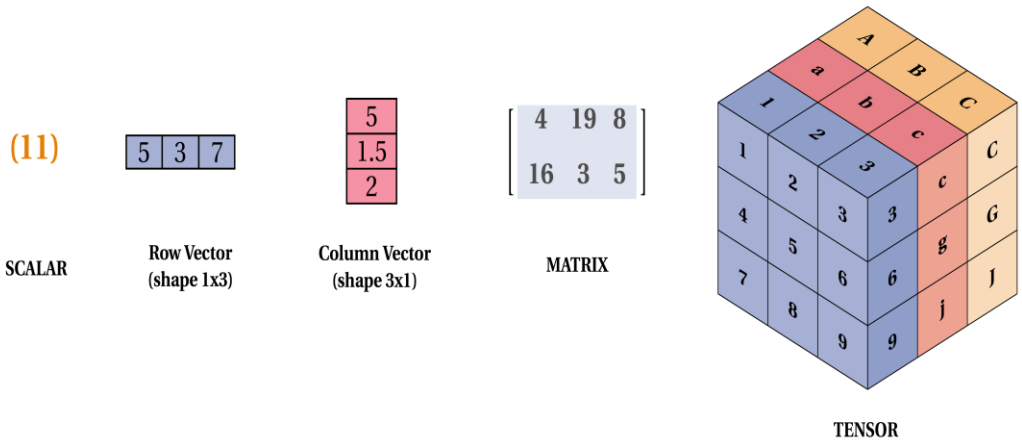

Figure 2.8 Tensor representation. Source [16]

2.2.4 Keras

Keras is a high-level API for neural network developments, written in Python. This API is able to run with backends such as Theano, Tensorflow and CNTK. Its goal is to enable fast experimentation by simplifying the mathematical operations of its backends.

It is one of the most used platforms for deep learning projects development due to the fact that it reduces the actions required for common use cases. Moreover, it offers most of the modules quoted before, i.e. input layers, convolutional pooling and activation layers. As showed in the following graph, Keras is one of the most used tools by deep learning research. As this graph from [8] shows, in October 2017, it was the second most mentioned term regarding deep learning frameworks and APIS:

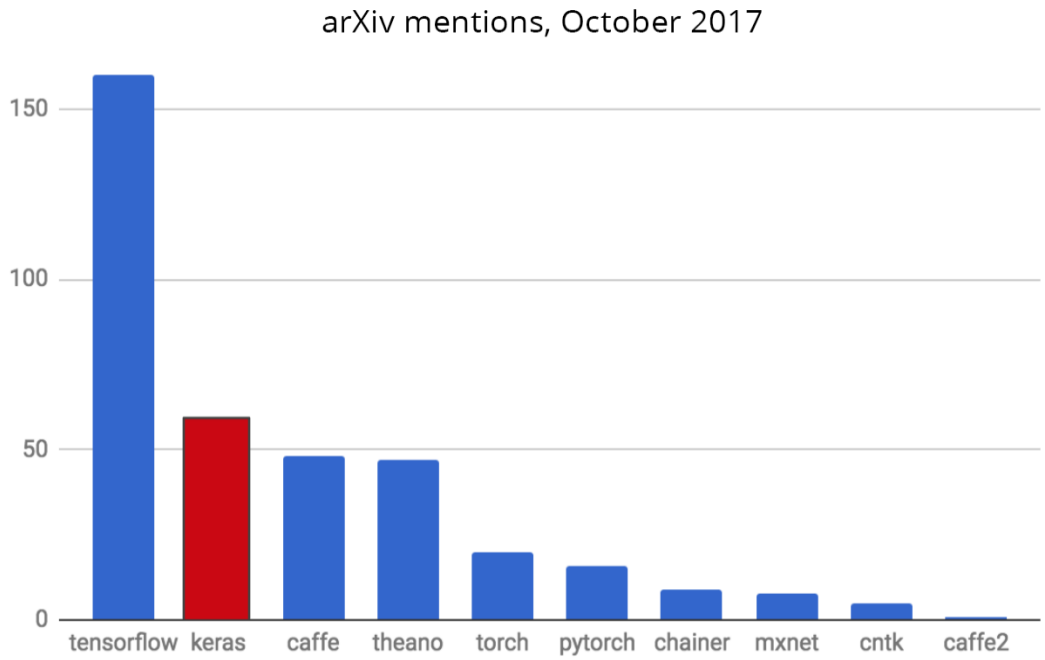

Figure 2.9 Keras mentions in axviv.org. Source [10]

In this project, Keras has been selected because of the simplicity it offers for deep learning implementation. In the following paragraphs, the most common layer of this neural network's API will be explained.

Dense layers

This type of layers are based on the following structure:

$$Output = activation(input \cdot weights + bias)$$

This layer is similar to Figure 2.5. This layer is used to obtain a predefined number of nodes which contains the most significant information obtained from the previous layers. Dense layers are used in most of deep learning implementations because they permit the adaptation of tensor's dimensions produced in layers to a pre-defined shape in order to adjust them to the algorithm purpose.

Flatten layers

Another important layer in deep neural network implementation is the flatten layer. The goal of this layers is to reshape the multidimensional vectors, mentioned before as tensors, to a one-dimensional vector, in order to apply fully connected layers such as dense or activation layers.

Dropout layers and the concepts of overfitting/underfitting

Dropout layers set a part of the neural network from a layer to 0 at each update during training, preventing overfitting. The overfitting process appears when the neural network is perfectly adapted to the training dataset, but the test process reflects a much lower accuracy. Underfitting regards algorithms which do not adapt enough to the dataset features and do not achieve the correct weights for the algorithm purpose.

The subsequent diagram represents the application of a dropout layer in the simple network defined in Figure 2.5.

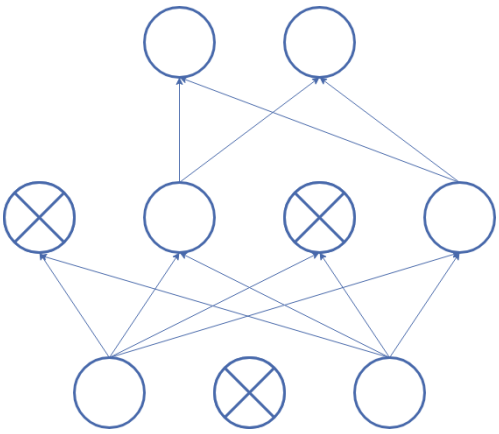

Figure 2.10 Dropout effect on Neural Network

2.3 STATE OF THE ART

Due to the fact that Convolutional Neural Networks explained on previous sections are the state of the art implementations regarding image processing with deep learning, different methods have been developed in order to approach an efficient solution to parking occupancy measuring.

In [8], an efficient implementation of CNNs is used for the detecting the occupancy of parking lots from a camera image. In this method a pre-defined architecture from a very popular deep CNN, called AlexNet is taken as base. Because of the large amount of neurons this deep neural network

has, the architecture is reduced in both the number of three-layer sets described in previous sections and the output layer classes, dimensioning them to the problem needs.

Similarly to this project, this implementation uses a custom dataset for training purposes, as well as a public dataset for training and validation processes.

The method implemented in [9] is similar to this project as a result of its use of manual segmentation for parking lots. This publication introduces the modification of the orientation of parking to complete vertical or horizontal position depending on its original angle. In addition, it proposes a new open source dataset for this type of developments named PKLot, which is an improvement for the development community. This dataset is used in both of the mentioned implementations, due to the fact that it has a large variety of parkings, weather conditions and car colors.

3 PROJECT DEVELOPMENT

This section gives a complete insight of the architecture of the project. First of all, we divide the different phases and components of the project. Subsequently, every element will be explained in detail.

3.1 IMAGE ANALYSIS

In this subsection we will be describing the processing given to the parking images in order to get the most important features for the training and testing processes.

Image analysis and processing is an essential part of this project. Regarding the different cameras position, as well as the different parkings used in this project, an extended process of image processing is needed.

In the starting point, we have thousands of images from four different camera locations. Each parking image has been hand-labelled with a specific order, creating a CSV file for each parking camera. Every row of each CSV file represents the ground truth labels of every frame, with a temporal order expressed in the image filename.

From this starting conditions, we have extracted each parking place as an independent, stored it as empty or occupied as well as the coordinates of the parking lot in the original image for dataset automatization purposes that will be described in the following sections.

3.1.1 Parking segmentation

The first phase of the image pre-processing stage is the segmentation of each image in parking lots labelled regarding its occupancy and stored properly.

In this part, the OpenCV library implemented in Python is used. It permits a fast creation of GUI for image visualization, as well as the management of keyboard and mouse events on it.

A distinguished characteristic of the segmentation process of this project is the trimming order. In order to take advantage of the hand-labelled CSV files, a specific order has to be obeyed for each parking camera. By reason of the automatization that the dataset generation process needs, the order has to be exactly the same that the ground truth labels. With this in mind, the process of dataset creation and automatic labelling described in subsections 3.2.1 and 3.2.2 is developed and a lot of time consuming task such as the dataset's folder creation and individual parking lot images labelling can be done automatically.

The process of parking segmentation has the succeeding phases:

- An individual image size is defined as a parameter of the program. This size will be conditioning the future input layer of out neural network.

- A reference image from each parking camera is defined for segmentation purposes.

- A text file is created at the beginning of the software routine for coordinate storage.

- The image is shown as an OpenCV window. This window is able to recognise mouse button event applied on them if an event subroutine is associated with the window.

- When the left button of our mouse is clicked, a new point is defined on the image. This point contains the two coordinates (x, y) of the mouse click position on the window.

- Due to the fact that we need four coordinates to define a parking lot, we have created a points array which stores the four points of each parking place. When the array is completed, a subroutine is called in order to store the points on the text file and create the individual lot image.

- When all the parking lots are labelled, the parking camera coordinates are stored in the text file. The process of coordinates disposal and storage will be detailed in subsection 3.1.3.

Coming from the previous process, the result of the segmentation process is depicted in Figure 3.1.

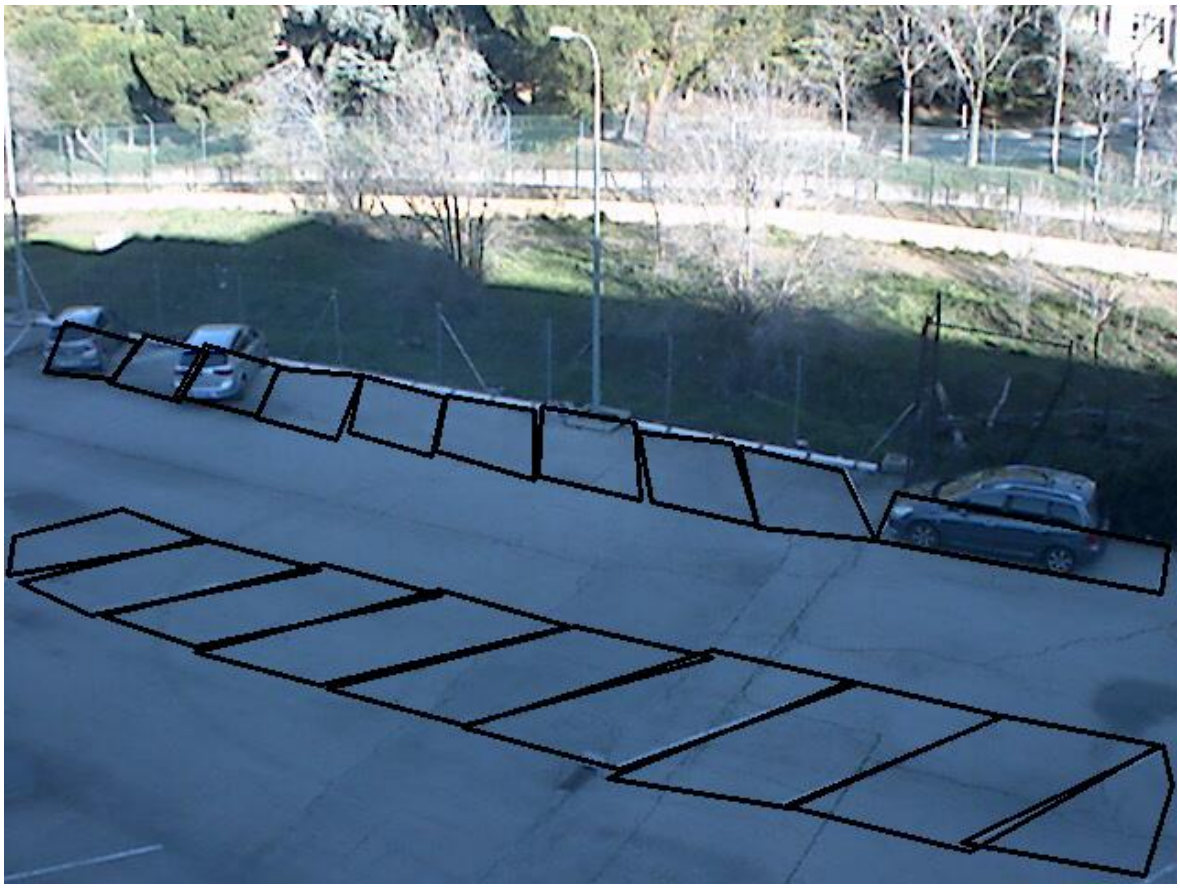

Figure 3.1 Parking Segmentation

3.1.2 Parking lot image transformation

As a result of the segmentation process applied on the parking images mentioned above, individual parking lot images are generated. These pictures sizes are determined initially by the following equations:

Defining P(x) as the x coordinates of the points array and P(y) as the y coordinates of the four points array, the initial width and height of our parking lot is calculated as:

$$Width = \max\big(P(x)\big) - \min\big(P(x)\big)$$

$$Height = \max\big(P(y)\big) - \min(P(y))$$

In order to achieve an initial rectangular image with this size from the initial points, a geometric transform is needed. Geometric transformations are linear applications which modify the properties of an image regarding the correlation extracted from pairs of points. Different types of geometric transforms are used in image processing. In this case, a perspective transform is needed, as a result of the different orientations of the parking lots.

In a mathematical form, the geometric transformations are based on matrix dot products. In the project, a 3x3 matrix as the shown below is needed in order to apply the perspective transform.

$$M = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$$

Establishing (x_1,y_1) as a point of our original trimmed parking lot and (x_2, y_2) the new coordinates of our image. The perspective transform or homography M relates them as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = M \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

To obtain the M matrix for the homography operation, we apply the function from OpenCV named *getPerspectiveTransform(source_points, dst_points)*. This method calculates a perspective transform using two points array with four elements. Applying the previous equation to each pair of two points, the matrix M is obtained. After that, the function *wrapPerspectiveTransform(image, transformMatrix, finalSize)* is applied. This routine, which also belongs to the OpenCV library, applies the obtained homography matrix to the original image and resizes the modified image to a size, defined as *finalSize* in this case.

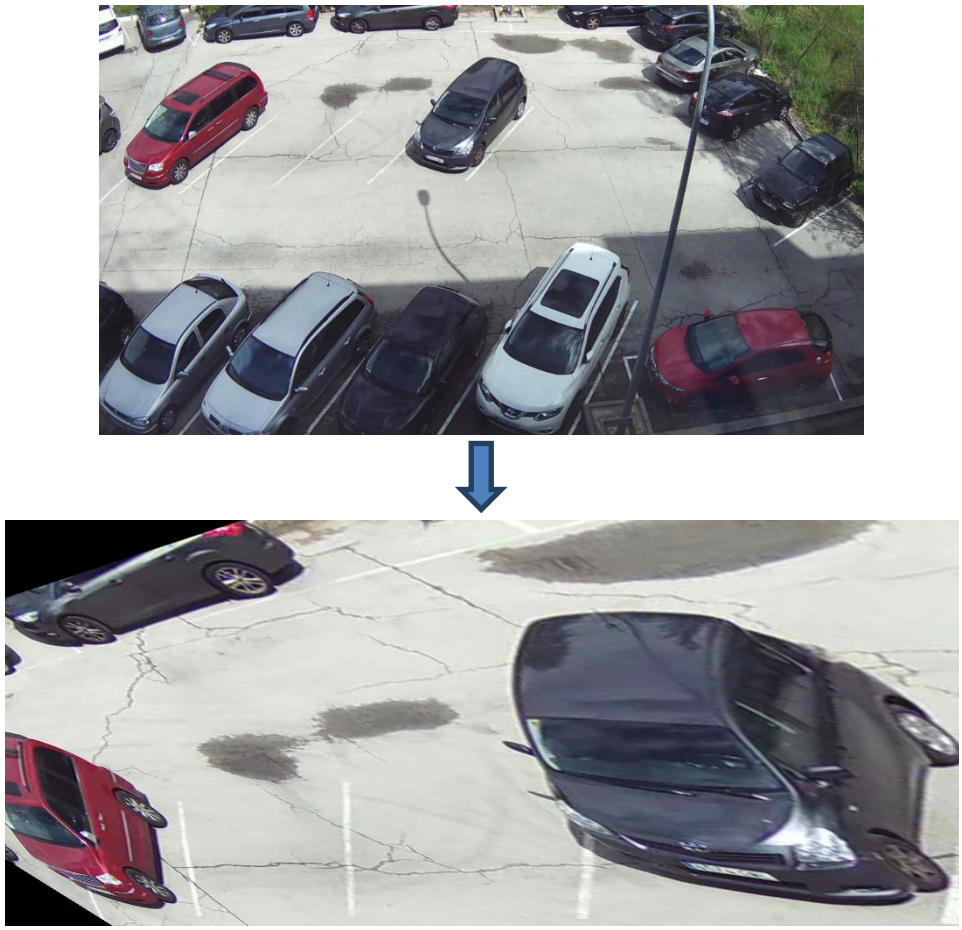

Figure 3.2 Parking lot first transformation

As a result of this first geometric transform, we have a variable size for each individual image. This feature has to be modified, because, as mentioned in preceding sections, an initial input size has to be defined for implementation purposes. To achieve this, a second perspective transformation is needed. The final size of this process has been modified along the development of the project for the sake of analysing the impact of this parameter in the training process. In subsection 4.1.1 we will be analysing the influence that image size on the final neural network.

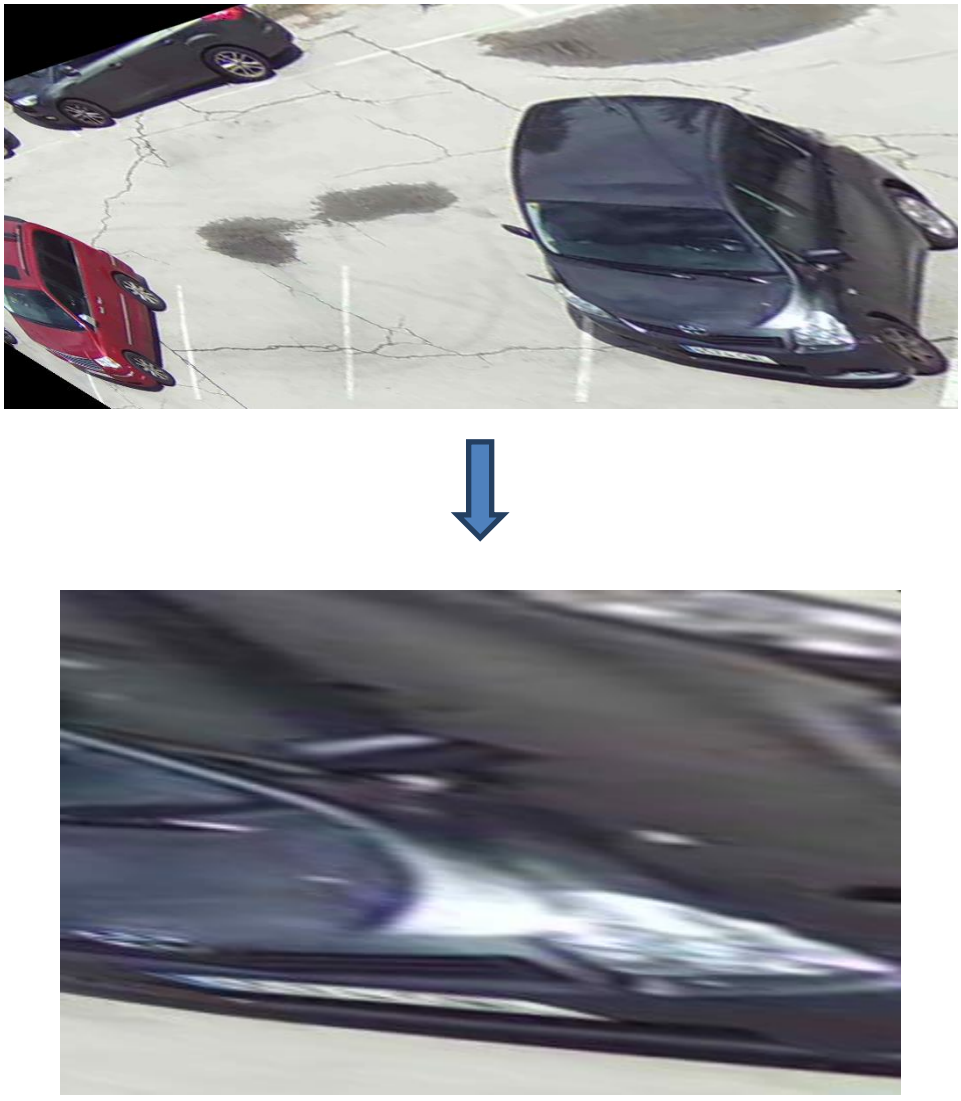

Figure 3.3 Parking lot second transformation

Sizes of 64x64, 192x192 and 256x256 have been chosen for the algorithm. The reason for the election of squared images regards in the use of convolutional neural networks. In developments including CNNs, squared images have been defined to use by different motives. The predominant purpose is to have controlled the size of the output images after every layer. When convolutional and pooling layers are applied to the pixels' matrix of images, the output size is obtained effortless.

For this reason, most of deep neural networks which include convolutional neural network structures employs equal dimensions for its input layers.

3.1.3 Coordinates storage for datasets

The second process regarding image segmentation is the storage of coordinate points. It should be outlined that the collected points for automatization purpose are the pre-transform points obtained by the mouse button subroutine named in section 3.1.1.

This method takes advantage of the fact that always four points are needed to define an individual parking lot. When the program activates the subroutine, a text file place the point's coordinates in pairs of (x, y) coordinates, separated by a space character. This character will be taken into account on the dataset generation process. After the parking camera image is completely segmented, the text file has N*4 lines of text, naming N as the number of individual parking lots on the camera image.

This process has prominent advantages as a labelling method. As a result of the decision of creating a unique text file for each different parking camera instead of every parking image, the total sum of files is reduced approximately by 50 percent. Denoting that more than our dataset is composed by more than fifty thousand parking images and only 6 text files are needed with our labelling process, the memory and disk storage preserved are a remarkable achievement.

3.2 PREPROCESSING DATA

The preprocessing of data is a crucial phase in the development of a deep neural network. The general process pipeline regarding data preprocessing for machine learning and deep learning projects is clearly described in the succeeding figure.

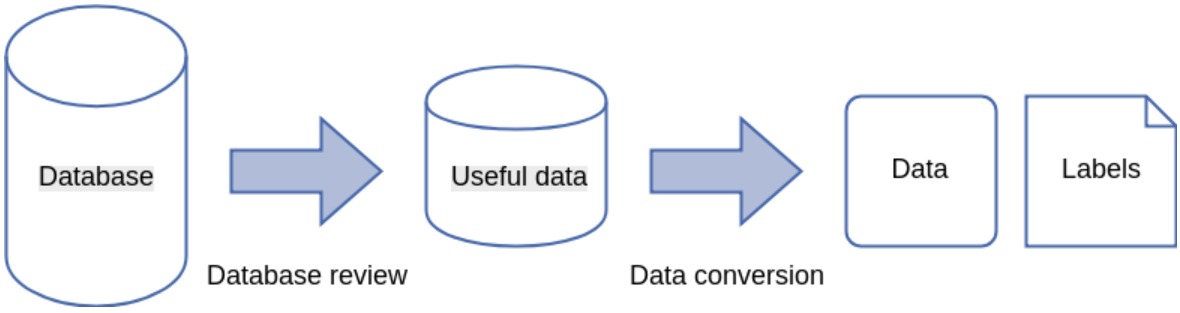

Figure 3.4 Pre-processing data scheme

The detailed process of data preprocessing in this project has included the following steps:

1. Text file association with parking camera.
2. Lecture of parking points and automatic parking lot segmentation and geometric transformation.
3. Class obtained from hand labelled CSV files in the correct position.
4. Training and testing dataset split.

The preprocessing sequence would follow this scheme:

Figure 3.5 Preprocessing procedure

3.2.1 Dataset creation

In order to automatize the dataset generation process, the module datasetGenerator is created.

The subsequent scheme represents the different parameters and processes that conform the module. In the following sections we will be explain in detail the purpose of them.

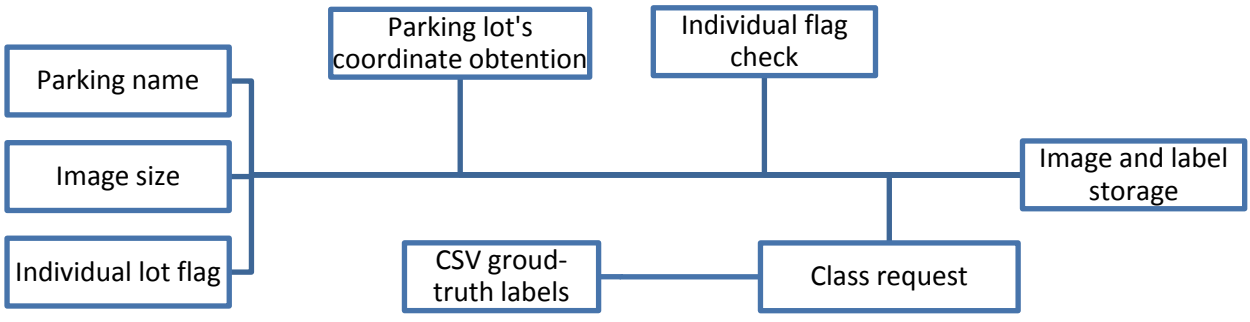

Figure 3.6 Dataset creation scheme

- Parking name: As described in preceding sections, the number of parking lot for each parking camera varies. Due to this fact, the parking name has to be defined previously to this module execution
- Image size: As mentioned, different input sizes have been picked for neural network analysis. The module need a pre-defined size in order to generate individual parking lot with the same dimensions.
- Individual place flag: In future sections we will be analysing the difference between creating a model for each parking camera place, a general model for each parking and a complete model not dependant from parking camera. This flag modifies the image saving routing, diving the dataset in each location and, inside of each location, as empty or occupied.

From these input parameters, the next information can be obtained:

- Segmentation text file: As a result of the segmentation process, the text file's filename follow the structure: "puntos_+ Parking Name+.txt", understanding the + symbols as string type concatenations in Python language. Taken advantage of this pre-defined structure, the text file can be automatically obtained from the parking name.

- Individual parking lots amount: Due to the fact that the number of parking lot is constant for each parking camera, the amount can be determined from the parking name.

- Dataset type: as mentioned above, different types of neural network models have been tested in this project. From the individual place flag we are able to determine if the dataset purpose will be for general parking training or for individual parking lot training.

3.2.2 Automatization of labelling process

Due to the significant number of individual images from both classes, whose sum in the largest dataset reaches the figure of 68817 images, an automatization perspective has to be taken in order to generate the different size datasets, including its labels, as fast as possible. Exploiting the fact that the parking images have been hand-labelled previously, a subroutine in the datasetGenerator module has been created to automatically label the individual image at the same time they are created.

To do so, the subroutine access to the CSV file which contains the ground-truth labels. 0 indicates an empty parking lot, while 1 means the individual parking lot is occupied.

As well as the text file which stores the parking lots' coordinates, the CSV file has a structure which contains the parking name taken as input. The structure is defined as: ocupaciones_+

Parking Name+.txt", where + symbols are string concatenations as well as in the text coordinates files.

As a consequence of the establishment of a pre-defined structure, where each row of the CSV file corresponds to the parking image, in alphabetical order, the connection between the parking lot and its ground truth label is made. This link is possible due to the counting variables used in the subroutine and the image filenames. An extract of one of our hand-labelled CSV files is shown in the following figure to illustrate the concepts mentioned in this section.

1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1
3	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1
4	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	1
5	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	1
6	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0	1
7	1	0	1	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0	1
8	1	0	1	0	0	1	0	1	0	1	0	0	1	0	0	1	1	0	1
9	1	0	1	0	0	1	0	1	0	1	0	0	1	0	0	1	1	0	1
10	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0
11	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0
12	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0
13	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0
14	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
15	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
18	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
19	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.7 CSV example

Finally, after the automatic labelling of each individual image, an automatic folder structure is created. This organisation is based on two parameters:

- Train/Test folders: The need of this separation will be described in the following section. Different percentages of dataset have been selected for the neural network training analysis.

- Class separation: For training and test purposes, the images have been divided in empty and occupied folders. The main reason is to take advantage of Keras pre-processing methods [10], which simplify the neural network example feeding process for binary classification problems such as this project. In future sections, we will be detailing how these functions have been implemented in our algorithm.

3.2.3 Training and test datasets

Thanks to the automatic process in image generation and labelling mentioned in above sections, different datasets have been created. As mentioned before, various image dimensions have been tested, as well as different divisions for training and test processes. In graphs from Figure 3.9 and Figure 3.8, the number of examples are detailed for the 256x256 size. Every size has the same amount of examples.

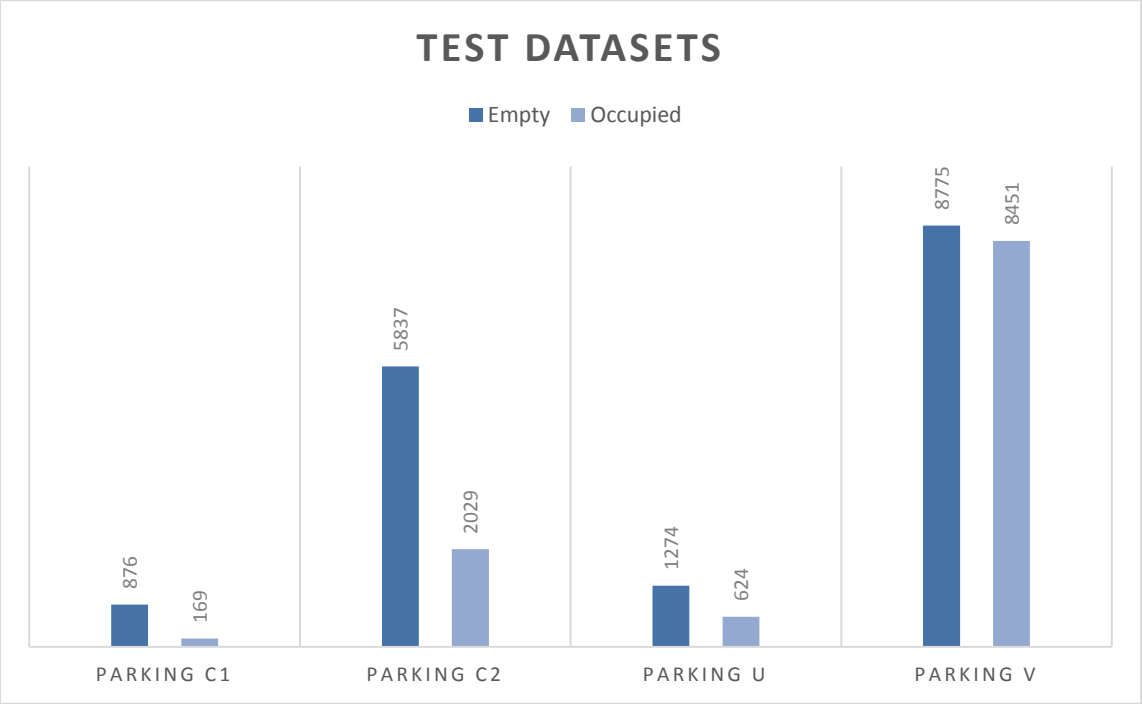

Figure 3.9 Test datasets examples

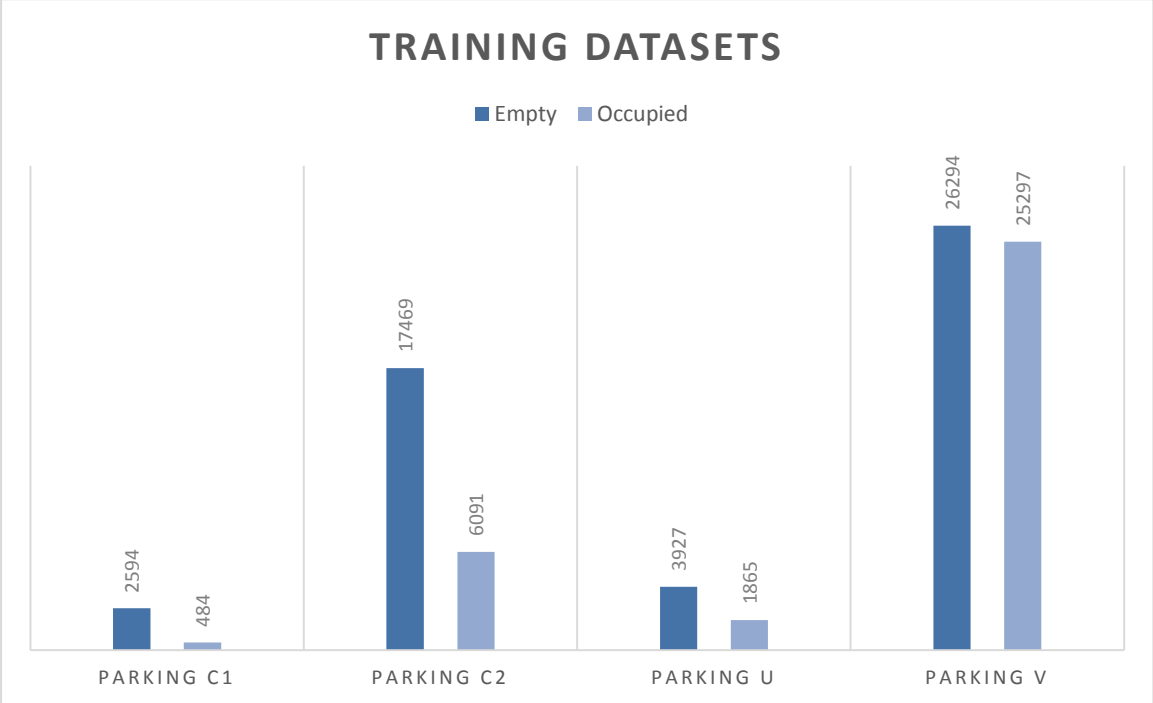

Figure 3.8 Training datasets examples

As it can be appreciated in the previous graphs, four cameras have been used for training the neural network. The next images are examples from different datasets:

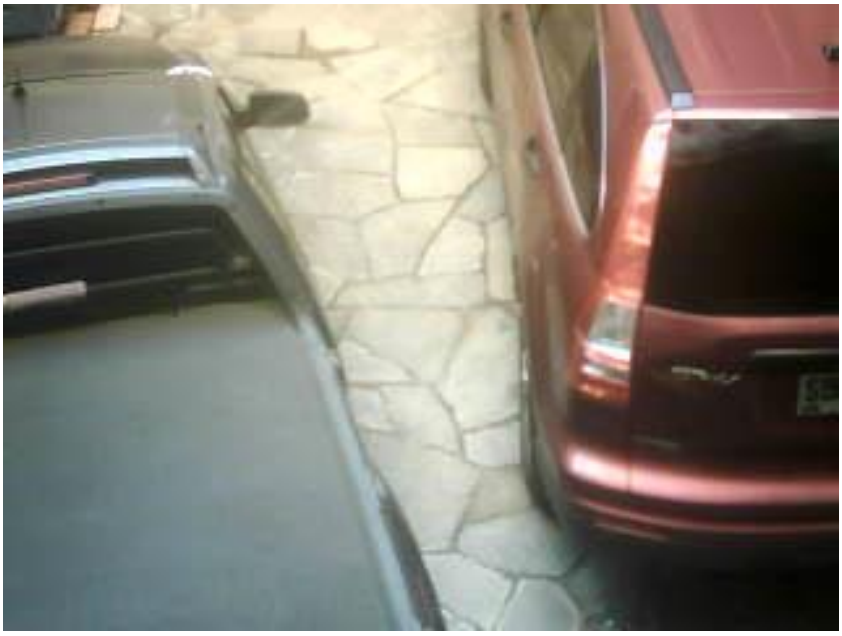

Figure 3.10 U example

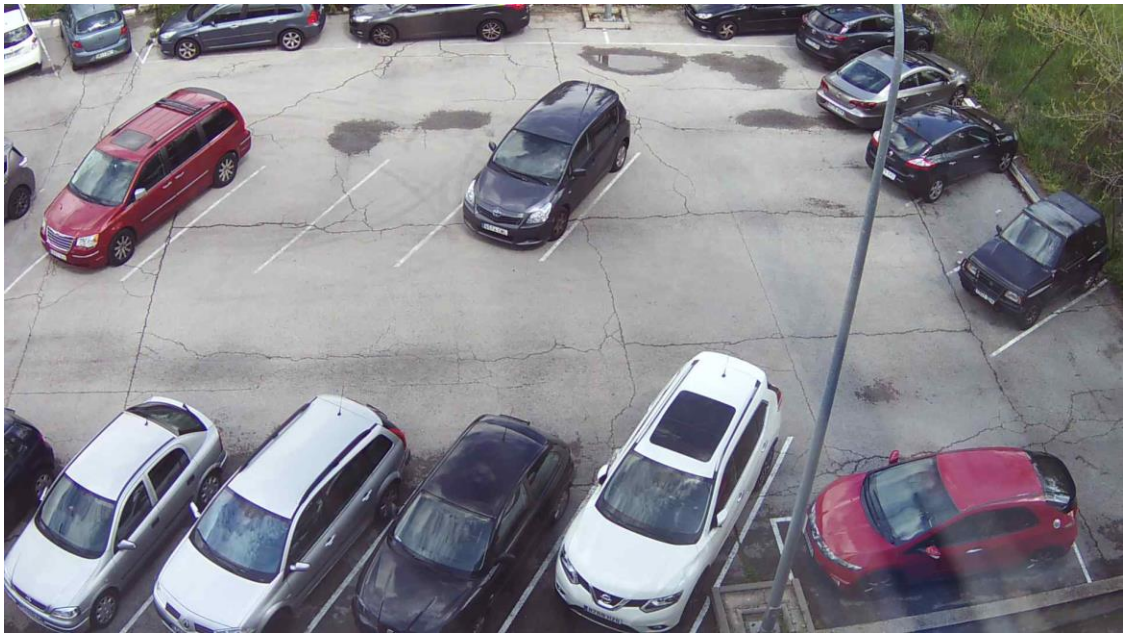

Figure 3.11 C1 example

Figure 3.12 C2 example

3.3 NEURAL NETWORK

The algorithm that will be determining whenever a parking lot is occupied is a data-preprocessing system which obtains individual parking lot images from a parking image together with its location, along with a deep neural network which predicts the availability of the parking lots As Described in detail sections 2.2.1 and 2.2.2, Convolutional Neural network structures conforms the core elements of the deep learning model used in this project.

3.3.1 Architecture

The function which creates the model, trains and tests it is named modelGenerator. This part of the algorithm obtains the parking name and the training parameters in order to produce a deep neural network architecture.

For this project, a Keras Sequential model is chosen [10]. This type of model architecture consists of a linear succession of neural network layers.

In order to create the model structure, the following parameters are necessary:

1. Image size: This parameter will determine the shape of our input layer. This type of layer converts the image format files such as PNG or JPG files to raw matrix's of RGB values. The dimensions of this input layer, with Tensorflow as backend for Keras is:

$$[Image\ width, Image\ height, 3]$$

2. Parking name: Because of the creation of a model for each parking camera, the parking name has to be introduced to this model in order to acquire the correct dataset for training and test processes.

From this parameters the architecture for parking models is created as follows:

- After the input layer, three convolutional neural network structures like the described in 2.2.2 are defined. Each convolutional neural network has the next layers:

    - Convolutional layer: With a variable number of filters: 32 for first and second CNN and 64 for the last one. The numbers have been selected as they are employed commonly in this type of layers.

    - Activation layer: As mentioned in previous sections, this layer's purpose is to extract information from the convolutional operation applied before. In this case, the layer employs a RELU function, as it is a standard in this type of deep neural networks.

    - MaxPooling layer: The purpose of these layer is to reduce the size of the matrix analysed in the neural network, in order to remove the less significant information from the matrix and reducing the number of nodes needed in the following layer.

- The output of the final CNN is a matrix whose size has been modified as a consequence of the operations applied to it. This matrix is the input of a Keras Flatten layer. As mentioned in section 2.2.4, this layer extends the matrix into one dimension, creating a vector structure.

- To process this vector structure, a Dense Keras layer is applied. With this layer we are able to reduce the length of our vector. A length of 64 elements is selected in order to balance the excessive combination and blurring of information and vector size.

- Following this layer, an Activation layer with a RELU function is applied in order to select the most significant coefficients of the vector.

- Subsequently, a Dropout layer is employed. With the intention to avoid overfitting, a dropout coefficient of 0, 5 has been stablished, giving balance between excessive removing of nodes and overfitting's risk reduction.

- The second-to-last layer is another dense layer. The purpose of this layer is to resize the vector to the final output of the model. In this project, the final result will consist on a number in the interval [0, 1].

- The final layer is a sigmoid activation function. The formula of this type of function is:

$$y = \frac{1}{1 + e^{-x}}$$

And the result of applying this equation to the interval [-10,10] is shown in the succeeding graph:

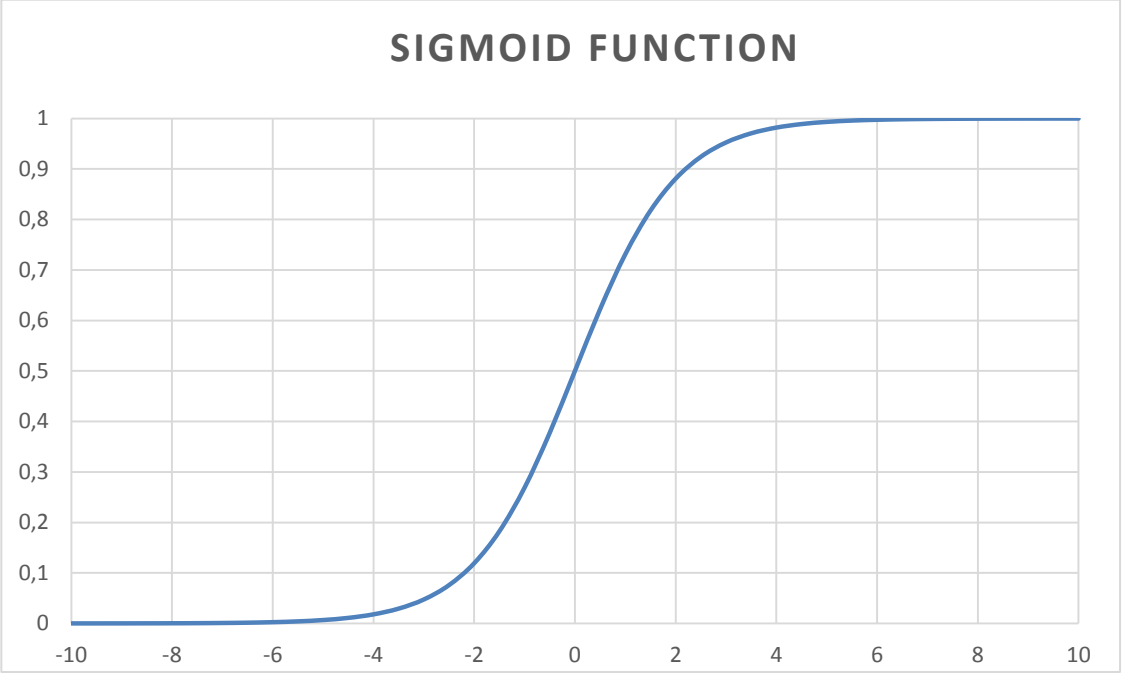

Figure 3.13 Sigmoid Function

As it can be denoted from the graph above, the sigmoid function is commonly used for binary classification problems such as the one this project is based on.

The result of this layer is, regarding the sigmoid formula, a number between 0 and 1.On the one hand, 0 means an absolute confidence that the input image is an empty image. On the other hand, 1 means the complete certain that the input image depict an occupied parking lot.

A threshold for intermediate values have been established in 0.5. Due to the result of the algorithm, this value has not changed along the development.

The model architecture detailed in Keras layers is represented in the following page.

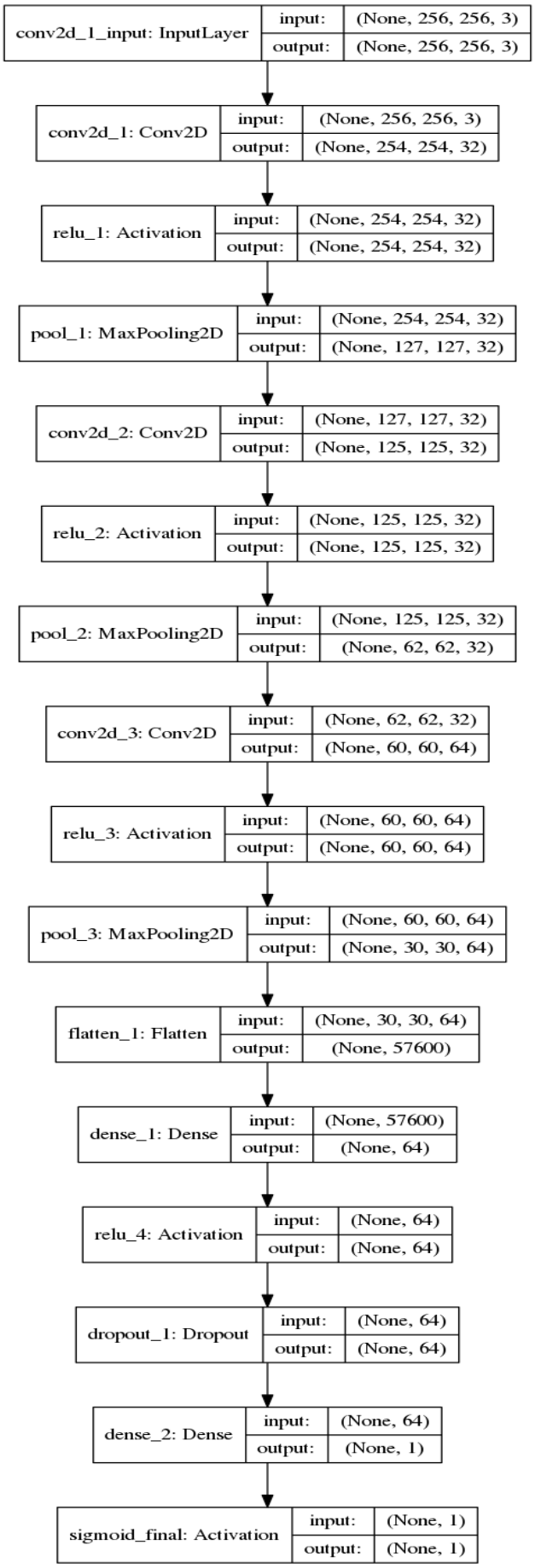

Figure 3.14 Model layer's structure

A detailed report of the matrix size between each layer, as well as the parameters used in each part of the architecture are illustrated in the subsequent figure:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 254, 254, 32)      896
_________________________________________________________________
relu_1 (Activation)          (None, 254, 254, 32)      0
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 125, 125, 32)      9248
_________________________________________________________________
relu_2 (Activation)          (None, 125, 125, 32)      0
_________________________________________________________________
pool_2 (MaxPooling2D)        (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
relu_3 (Activation)          (None, 60, 60, 64)        0
_________________________________________________________________
pool_3 (MaxPooling2D)        (None, 30, 30, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                3686464
_________________________________________________________________
relu_4 (Activation)          (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
sigmoid_final (Activation)   (None, 1)                 0
=================================================================
Total params: 3,715,169
Trainable params: 3,715,169
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.15 Model layer's parameters and dimensions

3.3.2 Model parameters

In this subsection, we will be describing the main parameters configured in the deep neural network model for the training and test processes.

Batch size

As one of the resource conditionals for the training and test processes, this value represents the number of input images stored in our GPU at the same time. Each time a new batch is introduced to the neural network, an additional ***step*** has been made in the process. As a consequence of graphic card memory limitations as well as image size, this factor may vary along the algorithm. Exponential natural values of 2 are commonly used to define the batch size value.

Epochs

This parameter represents the number of times the neural network will go across all the input examples of the dataset. Depending on the training dataset's size, this natural number could increase or decrease with the objective of creating the model as precise as possible and prove the model's strength against underfitting and overfitting.

Loss

Given an input value, which in deep learning is the prediction value of the model and a target, a ground-truth label value, the loss function calculates the differences between them. This parameter is highly conditioned by the model type. In this project, as we are facing a binary classification problem, the loss function picked is binary cross entropy. The formula of this loss function is defined in the following equation:

$$Loss = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

- N represent the total number of examples employed in loss calculation.
- y_i is the ground-truth value for the i-esim input.
- $\hat{y}_i$ is interpreted as the module's prediction value for the i-esim input.
- log is the representation of the natural logarithm operation.

The objective of every deep learning model is to minimize its loss function, which means the prediction for the model's input is equal to the labelled value. This parameter is crucial in training and testing process in deep learning.

Optimizer

The purpose of optimization algorithms is to update the weights and biases to minimize the loss function after each step. In this project, and Adam optimization algorithm is chosen.

Defined in [7] as the default optimizer for convolutional neural networks architecture, Adam optimizer has demonstrated it is an excellent election for this type of deep neural network models.

In the next graph, a comparative among different optimization algorithms is made in an MNIST neural network, which employs a similar architecture to our model.

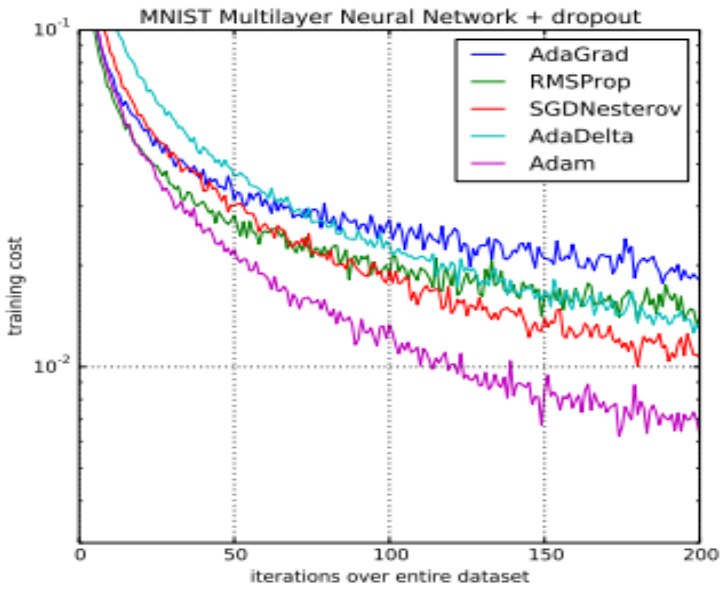

Figure 3.16 Adam optimizer comparison. Source [14]

Metrics

This function judges the performance of the model during the training process. The main difference with loss function stems from the fact that loss function are used when training the model and metrics values do not modify any parameter. Metrics are just a more comprehensive tool for understanding the model's performance.

3.3.3 Training and test processes

In order to create a complete functional deep neural network, training and testing processes are required. Training the model fits it to our dataset. The test process provides and unbiased evaluation of our final model fit on the test dataset.

To train the model, we need to transform the raw images into a tensor image data. To do so, the *ImageDataGenerator* class from Keras Image Pre-processing module is used. This class allow us to generate batches of this tensor image data with real-time data augmentation. Data augmentation regards a group of techniques which permit the generation of additional example from an original dataset. Depending on the model purpose, as well as the type of input, different techniques are applied to the dataset. In this project, we have modified the images with the following parameters:

1. Rescaling examples with 1:255 proportion
2. Shearing images in order to increase the strength versus probable image deformations
3. Zooming. The purpose of this technique is to predict from different image features
4. Horizontal flip. Due to the fact that vehicle could park in different positions, extra examples are created modifying the orientation of images.

With the modifications above, the diversity of our data is augmented significantly. After the training and test instances from ImageDataGenerator class are created, we apply the method flow_from_directory from this class. This method generates the batches of the augmented data defined in the ImageDataGenerator definition. This function has the following parameters:

- Dataset path to obtain data from.
- Image size for model's input layer
- Batch size for training process
- Class mode: The type of model regarding its output. In this project is defined as "binary" resulting from the problem approached.

After the dataset batch generator is defined, the model is ready to be trained. As a result of the election of Keras batch generator, the method fit_generator from Keras Model class is employed. The parameters needed to configure the training process are described below:

1. Training augmented data batch generator obtained from the previous process.
2. Steps per epoch: Ratio between training number of examples and batch size defined before.
3. Epochs: As defined before, the number of times the model will go through the complete dataset.
4. Callbacks: Function whose goal is to show information regarding the training process. This methods are useful to check if the training process is working correctly. In this project, a custom callback function has been developed in order to obtain the training loss and accuracy of the model at each step of the process.

After the training process is completed, the testing part begins. The aim of this element is to examine if the models have overfitted or underffited the training data. Due to the fact that this part does not need a large amount of data for its purpose, the generator for this part only applies rescaling to its dataset part.

As well as the training part, the accuracy and loss values are extracted from a callback function.

3.3.4 Model weights and architecture saving process

After both training and test phases have ended, the resultant model has to be saved for future analysis and deployments. Initially, the model weights were saved together with the model architecture in an .h5 file. This file extension comes from the HDF5 version from Hierarchical Data Format (HDF) [11], HDF defines a group of file formats in its versions HDF4 and HDF5 designed to organize and save large amounts of data.

For starting models for 64x64 images, this format has worked correctly. The moment the input data size was increased to 256x256, the training device could not afford to store the model architecture and weights in the same file format occasioned by the power of its resources. To resolve this problem, a new method of model storage is defined. From Keras Models module, the function to_json() is implemented. With this function, the program saves the model architecture in a JSON file, separating it from the model weights, which are saved in HDF5 format. This procedure allows the training device to store the model for all the different sizes of input data.

Finally, similarly to the callback function mentioned in the section before, a routine is defined to keep significant parameters of the training process for analysing possible misbehaviours and failures committed during the procedure, along with labelling the model for future use. The model's characteristics saved have been:

- Dataset employed for training and test
- Number of training and test examples of the mentioned dataset
- Size of input layer
- Batch size and number of epochs
- Optimizer and loss functions used to fit the model and upgrade the bias and weights.
- Training and test loss and accuracy
- Flag indicating if the architecture and weights are saved together or not. This will condition the model loading in analysis and deployment phases, described in future sections.
- Transfer learning indicator: Points out if the model has been based on a pre-trained model or has been trained from non-initialized weights and biases.

The diagram depict in the following page illustrates the complete process from the generation of the model, to the storage of the parameters resulting from the training/test processes.

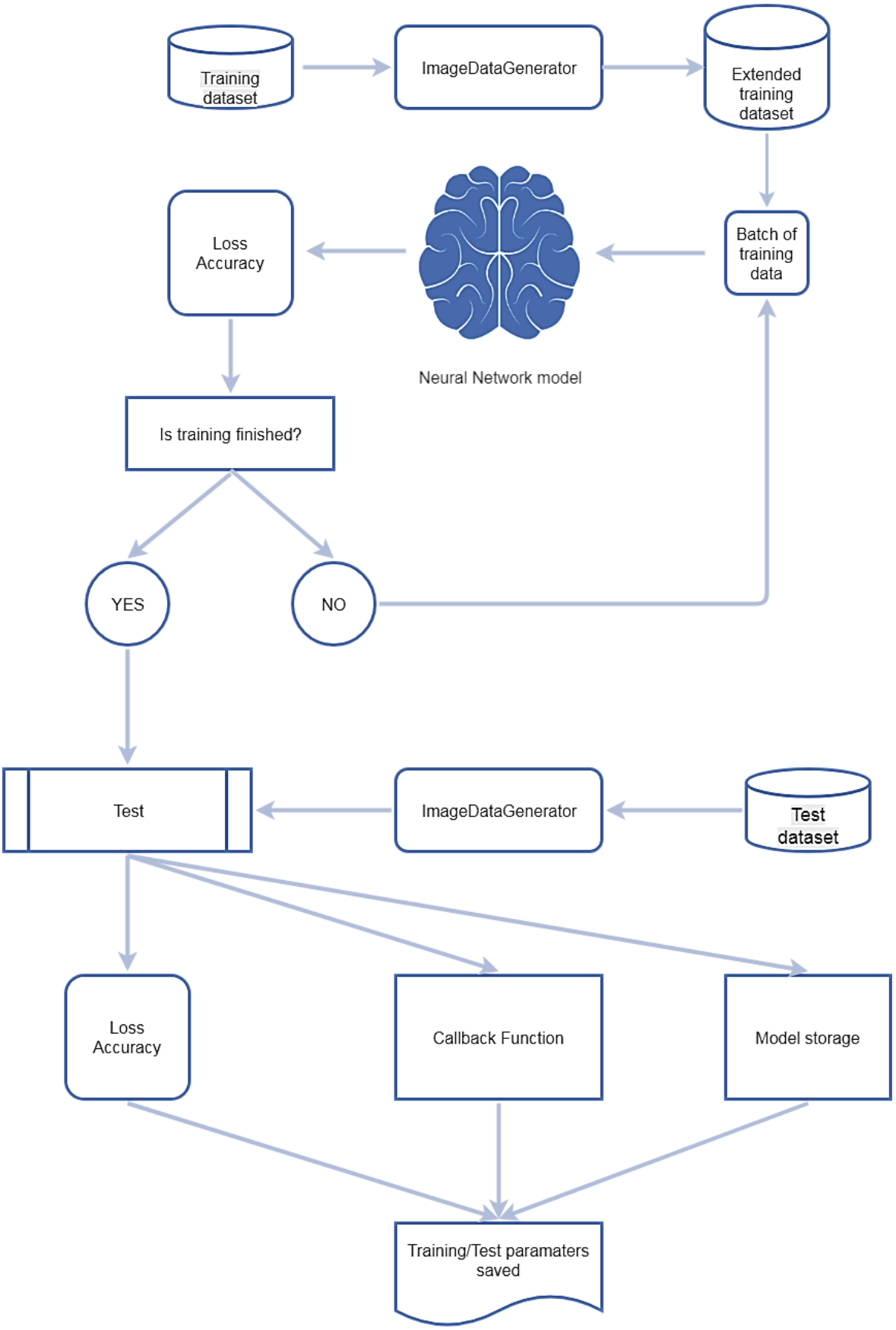

Figure 3.17 Complete model process

3.4 DEEP LEARNING MODELS

In this section, the different types of model approach will be described, as well as the procedures related to each kind and the concept of transfer learning models, a common approach for pre-trained models.

3.4.1 Parking dedicated model

The first approach to the problem was to generate a different model for each parking camera dataset. The reasons for this election are detailed below:

- Manageable and automatic dataset creation: As a consequence of the division of hand-labelled occupations in each parking camera and for dataset automatization purposes mentioned in previous sections, this approach was the most convenient for the initial development of the models

- Training process simplicity. To verify that the complete model's creation pipeline was working correctly, this design gives as the opportunity to implement a complete pipeline with less effort than other approaches such as the creation of an individual parking lot model described in the following section or a mixed parking model trained from non-initialized weights.

- Balance between model number and problem scenarios. As a result of the fact that in this project only a few cameras are analysed, this approach gives us suitable equilibrium between the number of models for our case scenarios.

From these motivations, the parking dedicated approach was initially selected. It has composed the base for the transfer learning part, which is a distinguished feature of the project.

3.4.2 Individual parking lot models

After the approach mentioned above, an individual parking lot model has been developed. In this method, each parking lot from every camera has been trained separately. The purpose of this procedure is to observe the performance difference between the previous implementation. Due to the characteristic of this approach, the following modifications were made:

- Dataset creation process changed. As a consequence of the individual parking lot training, the dataset has a different structure. The separation is done by parking lot position and after that, in empty/occupied classification.

- Model generation, training, storage and load. This approach produced various initial difficulties in the processes mentioned. The model generated has to be trained regarding its parking lot camera and number. In order to train every parking place at the same time, the

training process pipeline had to be modified for this type of training. After each model has completed the training and test phases, the storage process has to be changed as well in order to identify correctly the parking place trained for the model. In this model, loading process and increase of resources use emerges as a result of the creation of individual models.

3.4.3 Transfer learning

The concept preceding the parking independent models described in the next section is transfer learning. This procedure is employed when a pre-trained model is used as a base in a model creation process. From these base models, the architecture and weights are obtained at first. The architecture is normally modified with purpose of adapting the pre-trained model to the approach of the project together with input adjustments. After those elements are established, the new model is trained with a new dataset, adapting the weights mentioned to the new inputs and their features. This process reduces significantly the time needed for training neural networks due to the fact that these new models fit the dataset much faster if they are based on a correct pre-trained model than if they are trained from non-initialized bias and weights values.

This process is a state-of-the art methodology in deep learning developments. The main reason is the state-of-the-art deep neural networks for general image classification such as Alexnet [12], Resnet [13], which have won different deep learning competitions in platform as Kaggle.com, are trained in multiple powerful GPUs installed in professional development servers such as Google Cloud an Amazon Web Services. Most developers have significant lesser resources available for training their deep neural network, which causes a prominent increase of training time which transform the model analysis and modification in an extremely slow process. For this sum of reasons, large pre-trained models weights and architecture are used as base.

3.4.4 Parking independent model

As mentioned before, the goal of the transfer learning process is to create models from pre-trained weights and predefined architecture. In this project, the aim is to create a model able to predict as accurate as possible regardless of camera position, weather or light conditions.

To do so, a parking dedicated model has been taken as the pre-trained neural network. A transfer learning process has been applied to this model. As explained in section 3.4.3, this pre-trained model is re-trained with different datasets, in order to generalize the parameters and avoid overfitting to the original dataset. The model weights have been retrained on C1 and C2 training datasets. Because the input layer is not modified during the process, model weights are not modified in the retraining processes.

The most accurate model obtained applying this technique has been chosen as the final model applied on the project and its performance and results will be analysed in future sections.

4 RESULTS

In this section, we will go through the results obtained during the development and implementation of the detection system. First of all, we will analyze the impact of the different parameters in the model results, alongside with the influence of the dataset quality and the influence of weather. Secondly, we will compare our model with the state-of-the art systems developed for similar casuistry. Finally we will be showing the performance of the final model.

4.1 MODEL ANALYSIS

In the succeeding points we will be evaluating the conclusions extracted from the modifications made on the training processes along this project, as well as the dataset conditions which affect model performance.

4.1.1 Training parameters influence in model performance

Input image size

One of the most crucial parameters in this project has been the correct election of the input image size. These dimensions modify the model training time, as well as the number of weight value which conforms the different tensors in the neural network. As declared before, 64x64, 128x128 and 256x256 image input sizes have been tested. Figure 4.2 Loss variation by input size and Figure 4.1 Accuracy variation by input size show the performance variation of the model

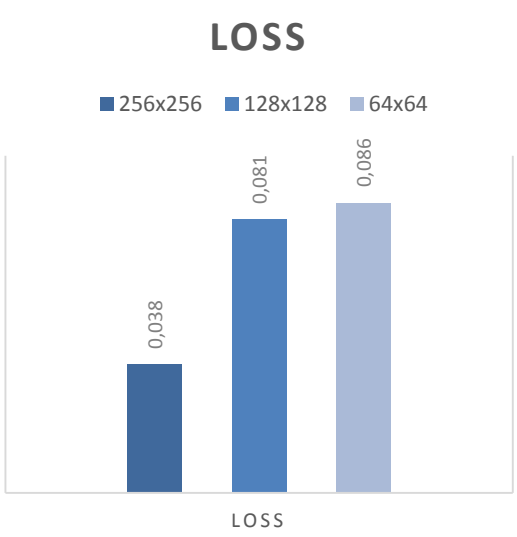

Figure 4.2 Loss variation by input size

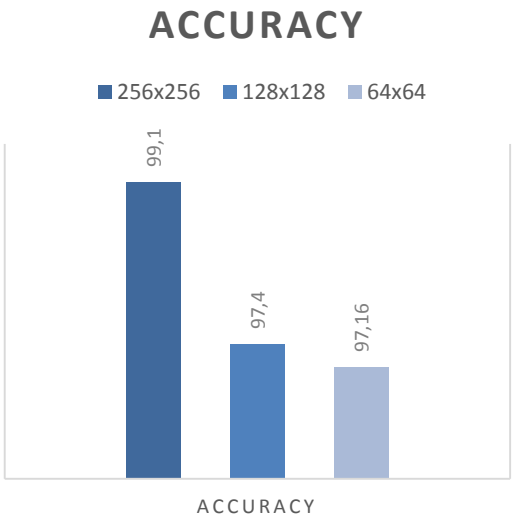

Figure 4.1 Accuracy variation by input size

varying only this characteristic.

As it can be deducted, the initial approach of 64x64 was an excessively low resolution for the optimal classification process. In a more moderate way, the image dimensions 128x128 still being a

lower resolution than the model needs to perform as precise as possible. With 256x256 inputs image, the parking lot image taken as input has a similar size to the parking lot dimension in original images. In our project, the performance of the model is prioritized over the training time, as the model is not very complex and the resources needed for a reasonable training are not elevated.

The difference of training time spent on one epoch in a dataset composed by 23560 examples with a batch size of 32 examples, is illustrated in Table 1:

Input size	256x256	128x128	64x64
Training time	9 min 30 secs	3 min 17 secs	2 min 7 secs

Table 1 Training time comparative

Batch size

Although this parameter seems to be used only for memory exhaustion prevention, its influence reaches another fundamental parameters of the model. The number of input data examples used modifies the training process, reducing its duration while increasing this value. Furthermore, the batch size implies that the neural network is able to compare with more values on each step, which gives a more accurate update to its weights. This could increase the performance of the model in more complex deep neural networks whose goal is a more difficult situation than the classification of parking occupancy levels.

4.1.2 Weather and lightning impact

Shadows

During the training and test process with C1 and C2 parking datasets, the influence of shadows in model performance has been remarkably determinant in this prediction's efficiency. As it can be appreciated in Figure 4.3 False positive due to shadows, different models have predicted false positives when a shadow is positioned over and empty parking lot.

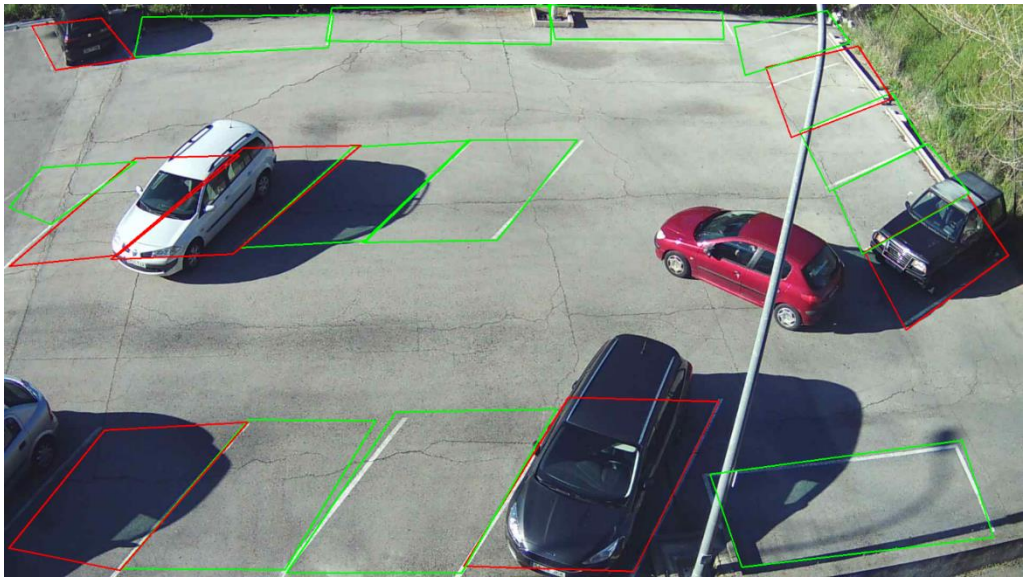

Figure 4.3 False positive due to shadows

Illumination an car's colours

As a result of the extensive range of car colours, the deep neural network has to be strong against different car colours that, in different conditions described below, the model could predict as false negatives:

- Black vehicles in low light conditions: Parkings are used in minimum light conditions. When a black vehicle is parked in this type of situation, the neural network's prediction could predict a false negative.

- High luminance colour in sunny weather: During sunny periods, vehicles with colors such as white or similar, could be interpreted as a false positive due to the high bright of the input image. Such misbehaviours have been shown during this project as depicts **¡Error! No se encuentra el origen de la referencia.**:

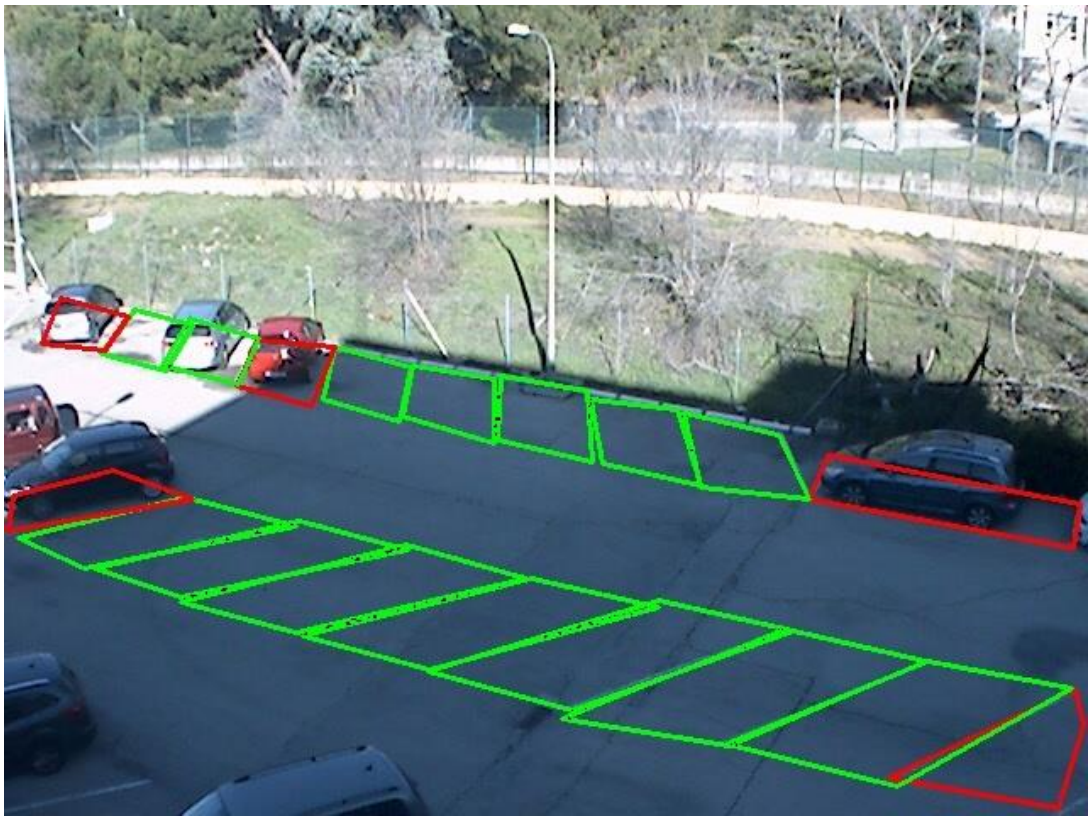

Figure 4.4 False negative because of light variation

In order to overcome these negative impacts, we have tested different input sizes. With 64x64 and 128x128, the number of false positive and false negatives are similar. Although, 256x256 increased significantly the strength of the model against this environmental variations. This is one of the reason which has produced the selection of 256x256 as input size for our final model.

4.1.3 Dataset quality

Parking camera resolution

As it can be deducted from previous parking images, the resolution of the camera is different for each parking. This condition modifies the performance of the models, especially in transfer learning situations. When the resolution is lower, less information can be extracted from each original parking image, which complicates the correct identification of the parking lot. The robustness against the variation of the image's resolution is a significant feature to take into account, as it represent the model's strength to real world conditions.

Special scenario's labelling

Another problem which influences the performance of the model is the manual labelling of the parking lots. Incorrect positions of car, which could occupy to places and the car would be divided in the image processing phase, complicate the detection process. In addition to that, this situation increases the difficulty of hand-labelling images. Differentiating between a parking lot that can be

occupied by a motorbike and not by a car, or even depending on the dimensions of the car are problematic situations in which labelling could modify the behaviour of the model.

4.2 FINAL MODEL: CREATION AND PERFORMANCE

In this subsection the creation process, efficiency and results of the final implementation model are analyzed and compared with state of the art implementations mentioned in previous sections.

4.2.1 Final model creation

As described in sections 3.4.3 and 3.4.4, the aim of this project is to create a deep neural network model for parking occupancy measurement in different locations. To do so, a model is trained on the different parking dataset following a transfer-learning process, which produces a model able to detect the occupancy levels independently of the parking location. Moreover, the final neural network is strong versus adverse weather and illumination conditions. For these reasons the model is prepared for production in real world situations. In the subsequent sections, the most significant results of training and test processes will be detailed, as well as the performance comparisons with state of the art methods described in section 2.3.

The architecture of the model is depicted in Figure 3.14. As it can be appreciated on that figure, the input size of the model is 256x256. This value has been selected due to the conclusions obtained in different test processes such as the described in 4.1.1 and 4.1.2Figure 3.14 Model layer's structure, as these dimensions have the highest accuracy along with the highest robustness against shadows and illumination problems.

4.2.2 Training and test results

Due to the use of callback functions during the training process, training and test processes' results are extracted at the moment they finished. The following table shows the performance of our model in these crucial procedures:

Parameters	*Loss*	*Accuracy*	*Examples*	*Input Size*
Training	0,01761	99,442	51591	256x256
Test	0,01605	99,522	17226	256x256

Table 2 Training and test results

The train loss and accuracy are depict in Figure 4.6 Training loss graph and Figure 4.5 Training accuracy graphobtained by our callback function.

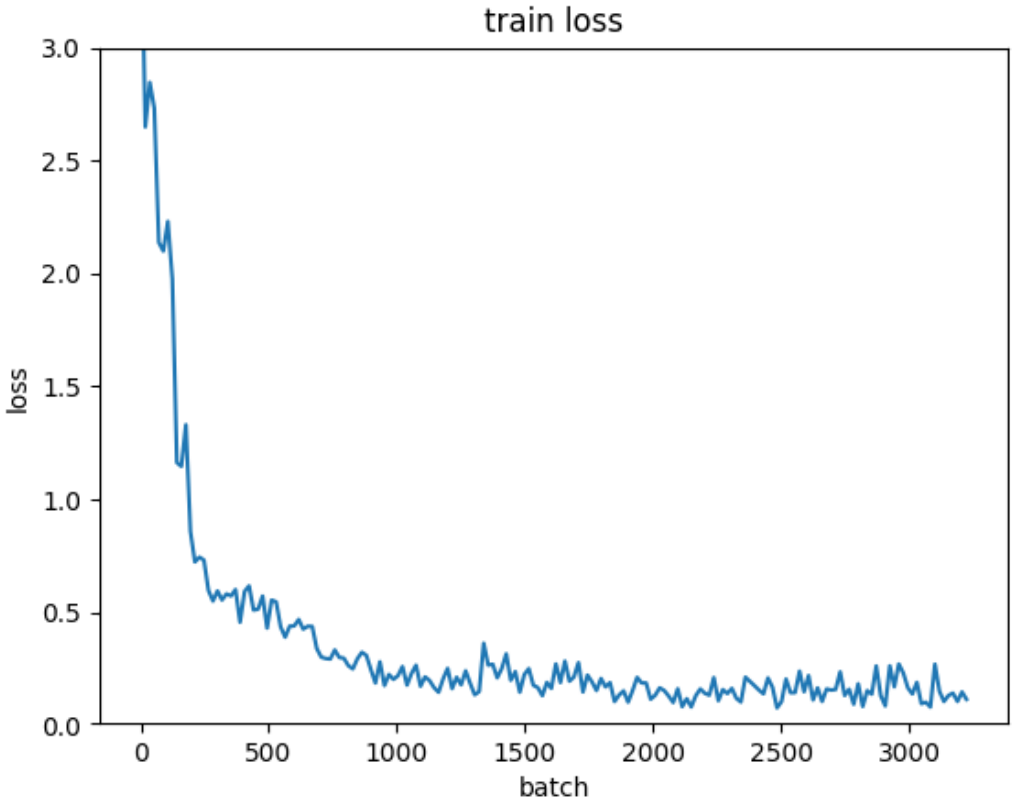

Figure 4.6 Training loss graph

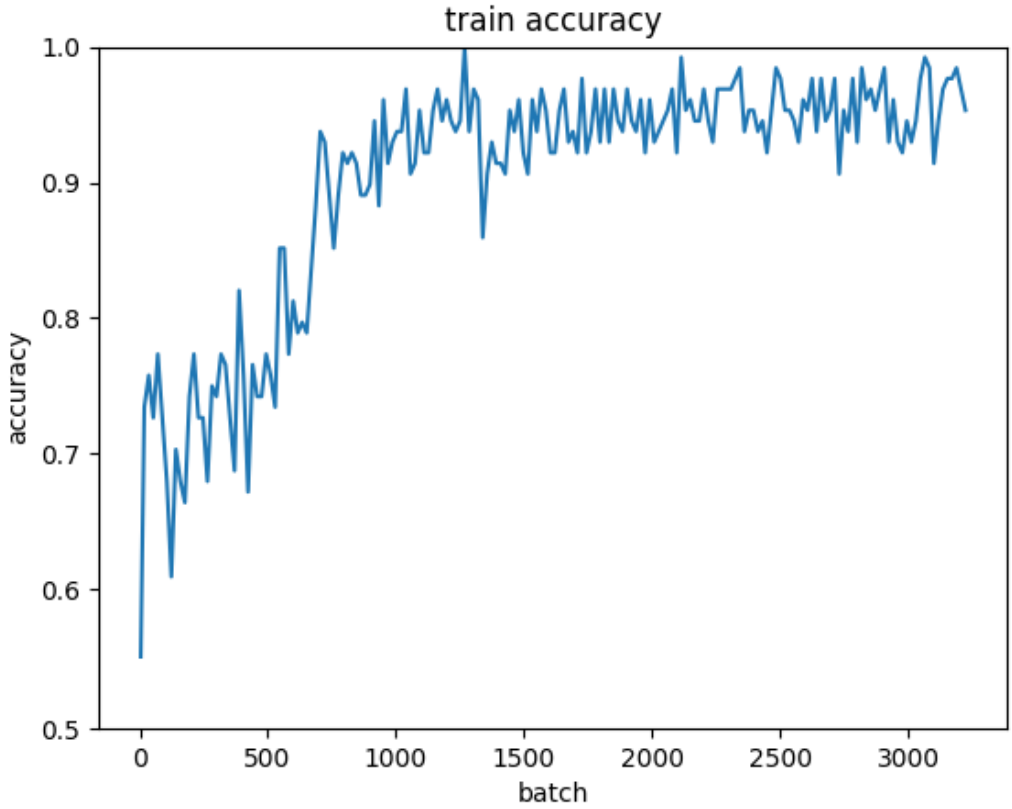

Figure 4.5 Training accuracy graph

4.2.3 Weather and shadows robustness

As cited in section 4.1.2, multiple weather conditions can produce misbehaviours in models prediction. Along the following figures, we will be showing the strength of the final model against difficult condition:

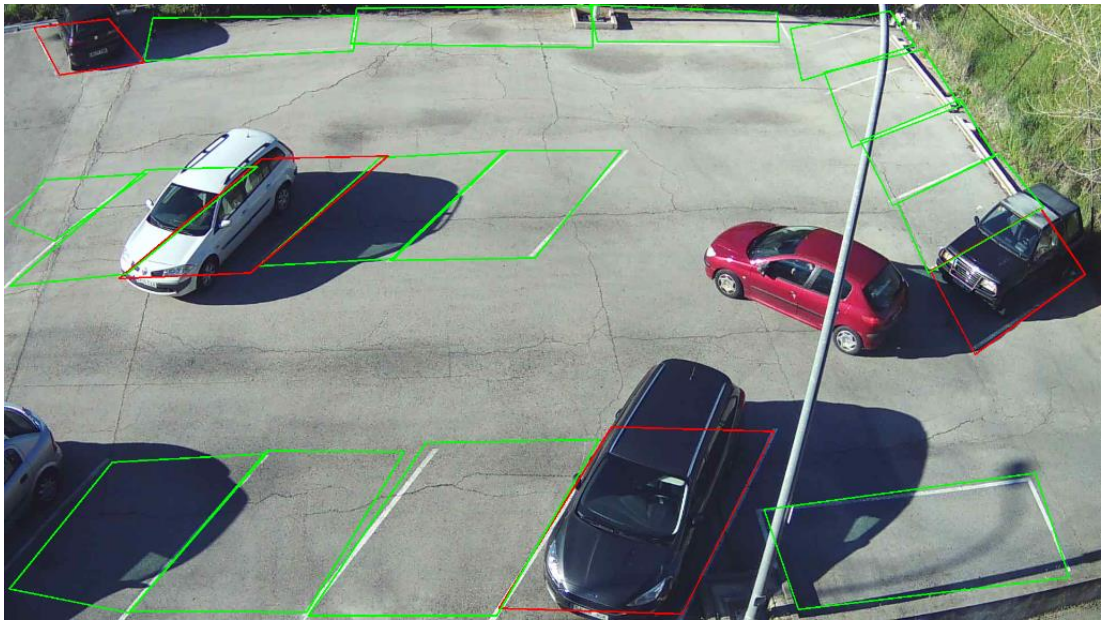

Figure 4.8 Model's shadow robustness

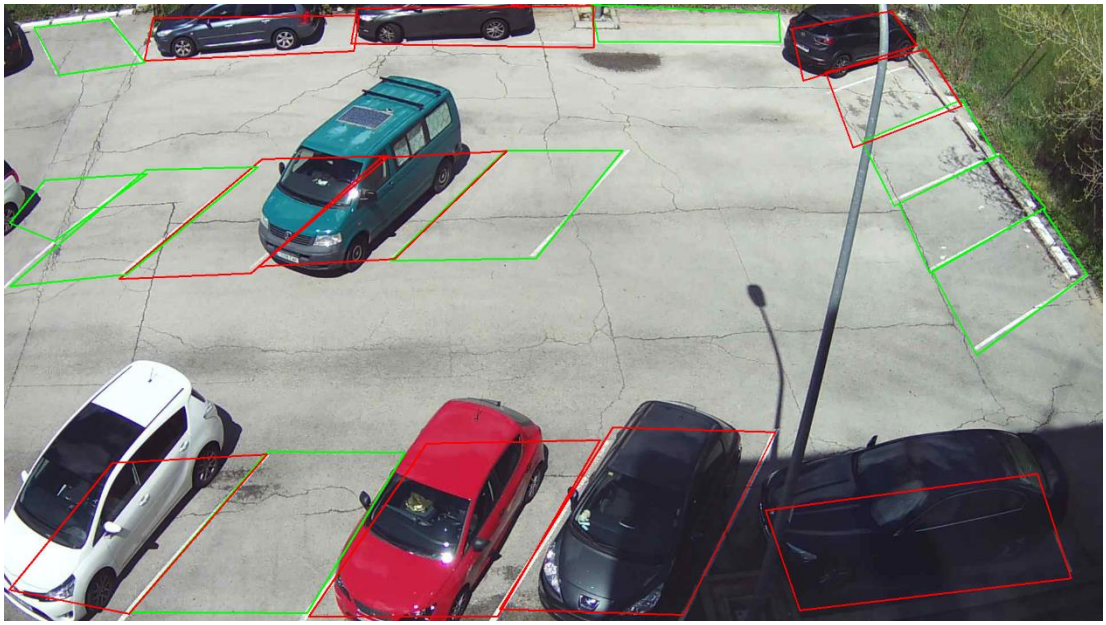

Figure 4.7 Model's light variation strength

4.2.4 Adaptability to different scenarios

Due to the aim of this project is to obtain a parking independent model, the fitting of the model to multiple environments, camera locations, etc. is needed to deploy it in real production. As it can be deducted from previous figures, the parking camera resolution varies significantly, as well as the real environment of the parking. Achieving the performance the model has reached in such different scenarios indicates it is prepared for deployment in an increased number of unlike parkings in future implementations. Scenarios as different as the following pictures depict have been predicted with our final model in order to illustrate the model's adaptation:

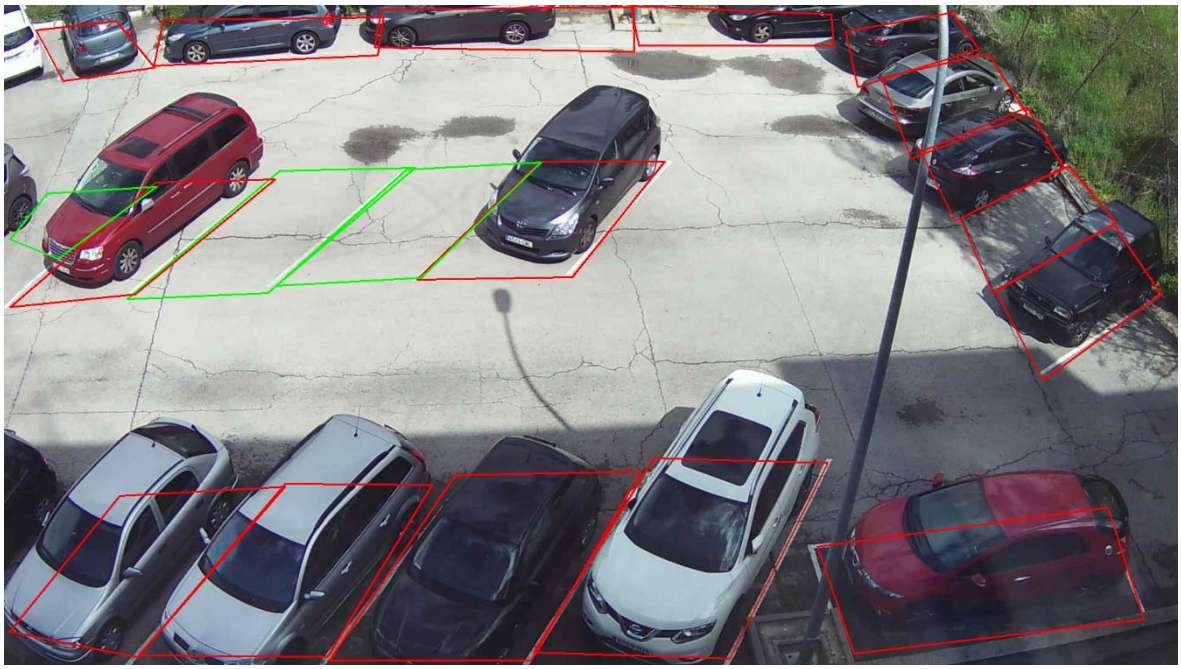

Figure 4.9 C2 parking adaptation

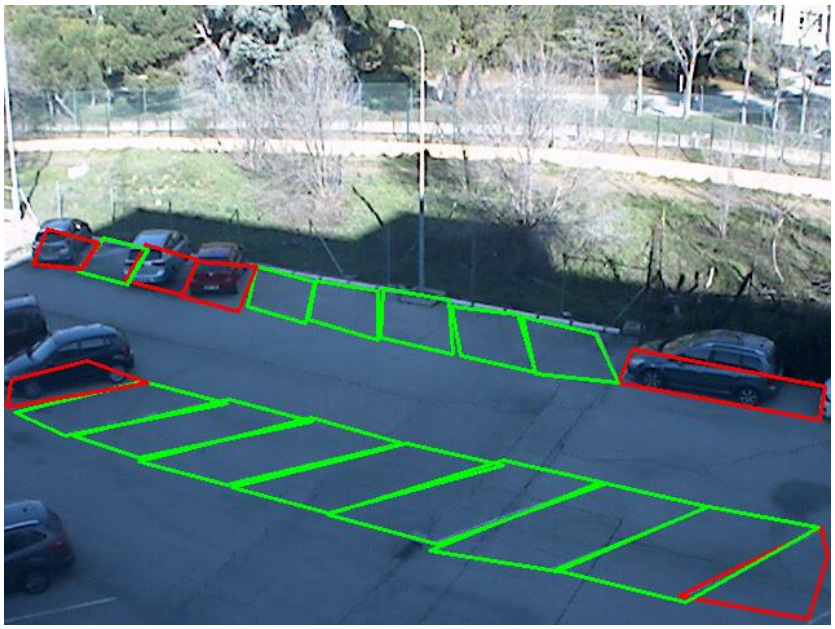

Figure 4.10 Model low light detection

Figure 4.11 U parking detection

4.3 STATE OF THE ART COMPARISONS

In this section, the state-of-the art methods mentioned in section 2.3 are compared with our algorithm in the succeeding table.

Algorithms	Decentralized parking [8]	PKLot [9]	Proposed
Test Accuracy (%)	99,3	99,3	99,522

5 CONCLUSIONS AND FUTURE WORK

In the following section we will detail the conclusions extracted from the project, as well as the future lines of work and implementations that could be included to the project.

5.1 CONCLUSIONS

In this project we have developed a Deep neural network model based on Convolutional Neural Networks able to detect with a significant accuracy the occupancy levels of different real parkings. Written in Python and based on Keras API, the model has been tested in multiple situations and has been compared with other state of the art methods. The model creation, as well as its training process have been analyzed and detailed in order to represent the development followed.

Thanks to the automatization process, an extended study of the influence of parameters such as image input size, training examples and weather conditions could have been studied. The difference between the manual procedure and this automatic process has been crucial for the fast implementation of the algorithms.

The neural network model overcomes slightly the accuracy of state of the art methods. This indicates that the development has been successful.

Moreover, the high accuracy levels achieved on different test datasets represent the advanced state of the development. This could indicate the model is ready to be tested and implemented with commercialization purposes.

The influence of parking lot's sizes has been one of the most determinant parameters in the training process. Establishing a size of 256x256 pixels has increased significantly the performance of the model without a remarkable increase of the model's prediction time. Due to the fact that the measure is not made every second, we can prioritize the accuracy over time performance.

The transfer learning processes applied to the individual model in order to achieve a generic parking model has supposed a turning point to obtain a model able to measure the occupancy on non-trained parkings.

In the succeeding sections, the achieved goals, as well as the future lines of work of the project.

5.2 ACHIEVED GOALS

The following list describe the achieved features of this project.

Developing a deep neural network model

As the main goal of the project, the development of the model has been accomplished from its generation to its training and implementation on real world situations. The elaboration of an algorithm based on a growing technology such as neural network set a perfect starting point for future lines of work.

Transfer learning

To obtain a model whose accuracy does not rely on the parking location, the retraining procedure implicit on the transfer learning process described in this project have been fundamental.

Image transformations for datasets

At the initial point, the dataset was composed by an elevated number of complete parking images. The storage of parking lot's coordinates, as well as the geometric transformations applied automatically for each parking place, have been two processes for the automatization procedure described in the following point.

Automatic dataset creation

The dataset generation from raw images and a complex label structure into an automatically created size-adapted dataset is one of the most difficult achieved goals of the project.

5.3 FUTURE WORK

The possible improvements that could be done to the project would be described in the following section.

Parking lot's automatic detection

As one of the most significant modifications of the project, the automatic detection of parking lots would reduce the labour of the parking lot segmentation to a non-human intervention process, which would permit an even faster implementation of the neural network model on different parkings whose lots could be divided easily.

Indoor parkings

The neural network has been trained for multiple outdoor parkings. Due to the fact that indoor parkings have not been the aim of this project, a future line of work could be the application of the transfer learning procedure to indoor datasets in order to implement the model in these conditions and study its performance versus other implementations with the same goal.

6 BIBLIOGRAPHY

[1] University of Tartu, “Digital Image Processing,” 1 September 2014. [Online]. Available: https://sisu.ut.ee/imageprocessing. [Accessed 23 April 2018].

[2] NVIDIA, “NVIDIA Developer OpenCV Section,” 2018. [Online]. Available: https://developer.nvidia.com/opencv. [Accessed 1 May 2018].

[3] Packt, “Packthub - OpenCV fundamentals,” 12 June 2013. [Online]. Available: https://hub.packtpub.com/quick-start-opencv-fundamentals/. [Accessed 4 May 2018].

[4] OpenCV, “OpenCV CUDA Platform,” 2018. [Online]. Available: https://opencv.org/platforms/cuda.html. [Accessed 4 May 2018].

[5] L. Hardesty, “MIT News, Explained Neural Netwoks,” 14 April 2017. [Online]. Available: http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. [Accessed May 6 2018].

[6] deeplearning4j.org, “Introduction to Deep Neural Networks,” 2017. [Online]. Available: https://deeplearning4j.org/neuralnet-overview. [Accessed 7 May 2018].

[7] S. University, “CS231n Convolutional Neural Networks for Visual Recognition,” 5 March 2015. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed 8 May 2018].

[8] A. Giuseppe, F. Carrara, F. Falchi, C. Gennaro, C. Meghini and C. Vairo, "Deep learning for decentralized parking lot occupancy detection," in *Expert Systems with Applications,Volume 72,ISSN 0957-4174*, 2017, pp. Pages 327-334.

[9] P. R. d. A. L. S. O. A. S. B. E. J. S. and . A. L. K. , "PKLot – A robust dataset for parking lot classification," in *Expert Systems with Applications, Volume 42, ISSN 0957-4174*, 2015, pp. Pages 4937-4949.

[10] Keras, “Keras Documentation,” 15 May 2015. [Online]. Available: https://keras.io/. [Accessed May 16 2018].

[11] J. M. Kusterer, "Hierarchical Data Format," NASA, 12 April 2012. [Online]. Available: https://eosweb.larc.nasa.gov/HBDOCS/hdf.html. [Accessed 22 May 2018].

[12] I. S. ,. G. E. H. Alex Krizhevsky, "ImageNet classification with deep convolutional neural networks," in *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 2012, pp. 1097-1105.

[13] K. H. X. Z. S. R. and J. S. , Deep Residual Learning for Image Recognition, 2015.

[14] J. B. Diederik P. Kingma, "Adam: A Method for Stochastic Optimization," in *arXiv:1412.6980*, 2014.

[15] R. Prabhu, "Understanding of Convolutional Neural Network (CNN)—Deep Learning," 4 March 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Accessed 10 May 2018].

[16] Refactored, 2018. [Online]. Available: https://refactored.ai/. [Accessed 10 May 2018].

7 APPENDIX A: ETHICAL, ECONOMICAL, SOCIAL AND ENVIRONMENTAL ASPECTS

7.1 INTRODUCTION

Because of the elevated number of vehicles in cities, parking is one of the most distinguised problems associated with it. In centrical areas, drivers take several minutes to find a parking place. This produces a consumption of resources such as time and fuel for the driver along with the emision of polluting gases. For this reasons, this project emerges as an effective solution for drivers, which will reduce the time employed to park their cars, as well as a it represents an ecologic measure, able to diminish the pollution in cities.

In the following sections we will be describing the main impacts involved in this project.

7.2 DESCRIPTION OF IMPACTS

The main impacts affected by this project are explained in the subsequent list, divided by its type.

Economical

As mentioned in the previous section, the solution will reduce the fuel consumed by drivers for parking purposes. Due to the fact that the large number of vehicles in our initial city of application, Madrid, the economical impact of our project will be considerable, highlighting as one of the core purposes.

Social

The decrease in time spent on parking will augment the fluity of traffic in the affected areas. Traffic is one of the most important problems in cities, which are the principal places to implement the solution, as it could be deducted from the recent traffic restrictions applied in Madrid.

Environmental

The time reduction in parking situations will decrease the contaminant emission produced by vehicles. As mentioned before, the elevated number of cars and motorbikes in big cities will multiply this positive impact, reducing the ecological footprint generated by vehicles.

7.3 ANALYSIS OF MAIN IMPACTS

From the main aspects influenced by our project, we are going to analyse the reduction of emissions.

Due to the high contamination's levels on cities like Madrid, where the pollution limits have been surpassed for the 8th consecutive year, this impact is fundamental. As contamintation reduction compose one of the features of our project, this perspective leads our solution to be in a good position for implementation financed by public organisms. This stablish a solid perspective regarding economic and administrative issues. Moreover, governments from other countries affected by this type of problems could find our solution interesting, which will lead in a significant reduction of the ecological footprint created by cars.

7.4 CONCLUSIONS

From the previous sections, we can extract the following conclusions:

- The reduction of the ecological footprint is the main impact of the project. The decrease of emmisions produced by car due to the time reduced in parkings are the most significant aspect regarding ecological issues.

- In economical aspects, the reduction of fuel consumed by car for parking is a positive impact for the users of our implementation.

- The increase of the fluidity of traffic generated by our implementation in cities reduces social problems caused by traffic, such as traffic jams in roads with a single line for each way, very common in cities.

8 APPENDIX B: ECONOMIC BUDGET

Table 3 Economic bugdet illustrates the detailed economic budget of our project.

	hours	Price/hour	TOTAL
Labour cost (direct costs)	300	15 €	**4.500 €**

Material resources costs (direct costs)	Purchase price	Use in months	Depreciation in years	TOTAL
PC and software	1.500,00 €	6	5	150,00 €
Cameras	500,00 €	6	5	50,00 €
Electronical components	30,00 €	6	1	15,00 €
TOTAL				**215,00 €**

General expendures (indirect costs)	15%	on DC	**707,25 €**
Industrial benefit	6%	on DC and IC	**325,34 €**

SUBTOTAL		**5.747,59 €**
VAT	21%	**1.206,99 €**
TOTAL		**6.954,58 €**

Table 3 Economic bugdet