

https://dspace.org/

Automated parking space detection

Nyambal, Julien Cedric

2018

https://hdl.handle.net/10539/25627

Nyambal, Julien Cedric, (2018) Automated parking space detection, University of the Witwatersrand, Johannesburg, https://hdl.handle.net/10539/25627.

Downloaded from a DSpace repository. https://dspace.org/



AUTOMATED PARKING SPACE DETECTION

Julien Cedric Nyambal

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, in fulfillment of the requirements for the degree of Master of Science

Supervised by Dr. Richard Klein

Johannesburg, 2018

ABSTRACT

Parking space management is a problem that most big cities encounter. Without parking space management strategies, the traffic can become anarchic. Compared to physical sensors around the parking lot, a camera monitoring it can send images to be processed for vacancy detection. This dissertation implements a system to automatically detect and classify spaces (vacant or occupied) in images of a parking lot. Detection is done using the Region based Convolutional Neural Networks (RCNN). It reduces the amount of time that would otherwise be spent manually mapping out a parking lot. After the spaces are detected, they are classified as either vacant or occupied. It is accomplished using the Histograms of Oriented Gradients (HOG) with the Linear and Radial Basis Function (RBF) Support Vector Machines (SVM), Convolutional Neural Networks (CNN) and a Hybrid approach. The classifiers are trained, tested and validated using data collected for this research. We compared the results of the Hybrid classifier against CNN and SVMs. The Hybrid classifier performed better than all the other ones with an accuracy of 89.36% and a precision of 82.54%, which is the best score obtained from all the other classifiers used. Novel contributions of this work include the new labeled database, the use of the RCNN for bay detection, and the classification of bays using the hybrid CNN and SVM.

DECLARATION

I, Julien Nyambal, hereby declare the contents of this dissertation to be my own work. This report is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Julien Nyambal
May 31, 2018

ACKNOWLEDGEMENTS

I would like first to thank my supervisor, Dr. Richard Klein, for the support and motivation he gave me during my research. His guidance played a crucial role in the success of my work. Dr. Klein first inspired me to pursue this Master's degree, and also inspired me to know more about the field as much as I can. His knowledge of the domain pushed me to always strive for the best.

A significant factor in my research was the precious advice of Professor Turgay Çelik. As lecturer, he taught me Machine Learning and Computer Vision, and as advisor, he opened my mind to incredible ideas related to research.

To my parents Simon and Marie-Claire, close and extended family for the support and motivation they gave me in my work during good and bad times.

Finally, I would like to thank CSIR for their financial support for this Master's degree through the Scholarship Award: CSIR Inter-Programme Bursary Award under the ICT Meraka.

Contents

Abstract i

Declaration ii

Acknowledgements iii

Table of Contents iv
Contents . . . iv
List of Figures . . . vii
List of Tables . . . ix

Publications xi

1 Introduction 1

2 Background 3
2.1 Convolutional Neural Networks . . . 3
2.1.1 Convolutional Layer . . . 4
2.1.2 Non-Linearity/Activation Layer: Rectification Linear Unit (ReLU) . . . 6
2.1.3 Pooling Layer . . . 9
2.1.4 Dropouts . . . 10
2.1.5 Fully Connected Layer . . . 10
2.1.6 Classification . . . 10
2.1.7 Training . . . 11
2.1.8 Popular Convolutional Neural Networks . . . 12
2.1.9 Special Case: Region-based Convolutional Neural Network . . . 14
2.2 Support Vector Machines . . . 16
2.2.1 Theory . . . 16
2.2.2 Margin . . . 16
2.2.3 Linear Separability . . . 16
2.2.4 Kernels . . . 18
2.3 Feature Engineering . . . 20
2.3.1 Histogram of Oriented Gradients . . . 20
2.3.2 Convolutional Neural Networks for Feature Extraction . . . 23
2.4 Conclusion . . . 23

3 Related Work 24
3.1 Sensors Based Detection . . . 24
3.2 Video Based Methods . . . 25
3.2.1 Traditional Image Processing . . . 25
3.2.2 Deep Learning - Convolutional Neural Networks . . . 27

3.2.3 Conclusion . . . 28

4 Dataset Design **29**
4.1 Collection . . . 29
4.2 Data Preprocessing and Labelling . . . 31
4.2.1 Preprocessing for Region-based Convolutional Neural Networks (RCNN) . . . 31
4.2.2 Data Labeling for RCNN . . . 32
4.2.3 Preprocessing for Support Vector Machines (SVM) & Convolutional Neural Networks (CNN) . . . 34
4.3 Conclusion . . . 35

5 Automated Bay Detection **37**
5.1 Region based Convolutional Neural Networks . . . 37
5.2 Architecture . . . 37
5.3 Training & Validation . . . 38
5.4 Testing . . . 39
5.5 Analysis . . . 40
5.6 Conclusion . . . 41

6 Bay Classification **43**
6.1 Convolutional Neural Networks . . . 43
6.1.1 Architecture . . . 43
6.1.2 Training & Validation . . . 44
6.1.3 Results . . . 45
6.2 Support Vector Machines . . . 46
6.2.1 Preprocessing & Features Extraction . . . 47
6.2.2 Histogram of Oriented Gradients . . . 47
6.2.3 Train & Validation . . . 48
6.2.4 Results . . . 48
6.3 Convolutional Neural Networks as a Feature Extractor for Support Vector Machines . . . 50
6.3.1 Architecture . . . 50
6.3.2 Preprocessing & Features Extraction . . . 50
6.3.3 Training & Validation . . . 51
6.3.4 Results . . . 52
6.4 Conclusion . . . 53

7 Analysis **55**

8 A Parking Space Detection System **57**
8.1 The camera . . . 57
8.2 The Server . . . 57
8.2.1 Overview of the System . . . 57
8.2.2 Detailed View: The whole process . . . 58
8.2.3 Detailed View: The classification Unit . . . 59
8.3 Visual Results of the Classifier on Real Images . . . 59
8.4 Conclusion . . . 61

9 Conclusion & Future Work **62**
9.1 Conclusion . . . 62
9.2 Contributions . . . 63
9.3 Future Work . . . 63

A Pseudo Code **64**

B Datasets **67**

References **74**

List of Figures

2.1 Convolution Neural Networks Process . . . 3
2.2 Convolution Filter Matrix . . . 4
2.3 Different Convolution Filters applied on a image . . . 6
2.4 The filter (b) convolves on the matrix (a) to produce (c): The Convolution operation . . . 7
2.5 CNNs Filters . . . 8
2.6 Some Convolutional Neural Networks Activation Functions and Their Derivatives . . . 9
2.7 Left line: Max-pooling — Right line: Average-pooling . . . 9
2.8 Convolutional Neural Networks and Fully Connected Layers . . . 10
2.9 Simulation of computation of Classification Loss: Mean Squared Error vs Cross-Entropy 12
2.10 LeNet5, [LeCun *et al.* 1998]. . . . 13
2.11 AlexNet, [Krizhevsky *et al.* 2012]. . . . 13
2.12 Object Detection Process with RCNN, [Girshick *et al.* 2014]. . . . 14
2.13 RCNN Selective Search Process . . . 15
2.14 RCNN Selective Search Process . . . 15
2.15 Support Vector Machines, [OpenCV 2017] . . . 16
2.16 Linear Separability of Some Data Points . . . 17
2.17 SVM Kernels, [ScikitLearn 2017] . . . 19
2.18 Gradients Computed from Histograms of Oriented Gradients algorithm. . . . 20
2.19 Histograms of Oriented Gradients: Gradients Computation . . . 22

3.1 Physical sensors around parking spots . . . 24
3.2 Background subtraction method, [del Postigo *et al.* 2015] . . . 26
3.3 Multi-class SVM, [Wu *et al.* 2007] . . . 26
3.4 Example of Convolutional Neural Networks, [kdnuggets 2016] . . . 27

4.1 Different periods of the day for data collection. . . . 30
4.2 Correction of radial distortion . . . 31
4.3 Mean Normalization . . . 32
4.4 Sample of dataset for RCNN . . . 33
4.5 Data Labeling for RCNN . . . 33
4.6 Instances dataset for classification for SVM and CNN . . . 35
4.7 Execution time of processing the mean and the standard deviation of an image . . . 35

5.1 ZF Net [Zeiler and Fergus 2013] . . . 37
5.2 Training loss of the RCNN Detection Unit . . . 39
5.3 Difference of image quality between Train/Validation and deployement images . . . 40
5.4 Detection of the System on a parking space . . . 41
5.5 Individual detected spots . . . 41

6.1 Example of ambiguous vacant parking . . . 45
6.2 Confusion Matrix and ROC curve of custom VGG16 CNN classifier . . . 46
6.3 Activation on HOG critical points . . . 47

6.4 Metrics of classifier SVM with different Kernels 49
6.5 Metrics of classifier CNN → SVM with different Kernels 53

7.1 Comparison of all the used algorithms 55

8.1 An Automated Parking Space System 58
8.2 Detailed view of the system 59
8.3 Classification Unit 60
8.4 Detection visuals 60
8.5 Classification visuals 60

B.1 Occupied Parking Spaces 68
B.2 Vacant Parking Spaces 69

List of Tables

2.1 LeNet5 vs AlexNet, [Li 2017] . . . 13

4.1 Dynamics of the Chamber of Mines parking (Wits) . . . 29
4.2 Data Collection Summary Training/Validation/Testing: Detection . . . 30
4.3 Data Collection Summary Training/Validation/Testing: Classification . . . 30

5.1 RCNN Detection Training Configurations . . . 38
5.2 Mean Average Precision (mAP) RCNN Detection Unit . . . 40

6.1 Custom VGG-16: Modifications . . . 44
6.2 Data Distribution for Training/Validation/Testing . . . 45
6.3 CNN Classification Training Configurations . . . 45
6.4 CNN Classification Training Configurations . . . 46
6.5 HOG Configuration . . . 48
6.6 Data Set Settings . . . 48
6.7 SVM Performance metric . . . 49
6.8 Custom VGG-16: Training . . . 51
6.9 CNN $\rightarrow$ SVM parameters . . . 52
6.10 Data Set Settings . . . 52
6.11 CNN $\rightarrow$ SVM Performance metric . . . 52

7.1 Comparison of this research results with the literature . . . 56
7.2 Overall Results . . . 56

Nomenclature

AUC Area Under the Curve

$AUROC$ Area Under an Receiver Operating Characteristic curve

$CNN \rightarrow SVM$ Convolutional Neural Networks as a feature extractor for SVM

CNN Convolutional Neural Networks

DoG Difference of Gaussians

HOG Histogram of Oriented Gradients

$MNIST$ Modified National Institute of Standards and Technology Database for handwritten digits recognition

$One\ Hot\ Encoding$ Process that transforms categorical features in to a vector to be used in a Machine Learning algorithm. For example, let L = {Occupied, Vacant} the set of labels of the Machine Learning algorithm of choice. The One Hot Encoded version of L becomes: L= {[1,0], [0,1]}, where the position of 1 that encodes the label.

RBF Radial Basis Function

$RCNN$ Region based Convolutional Neural Network

SGD Stochastic Gradient Descent

$SIFT$ Scale-Invariant Feature Transform

SVM Support Vector Machines

PUBLICATIONS

Portions of this research have been presented at:

- The Eighth Cross-Faculty Postgraduate Symposium at the University of the Witwatersrand, Johannesburg, South Africa (2017)
- [Nyambal and Klein 2017] J. Nyambal and R. Klein. Automated parking space detection using convolutional neural networks. In 2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech), pages 16. IEEE, Nov 2017
- Deep Learning Indaba, Johannesburg, South Africa (2017)

Chapter 1

Introduction

Getting an available parking spot is a problem faced by many car owners. It can quickly create congestions in the parking lot, especially at peak hours, as drivers are stuck not knowing where to park their car. Sometimes, and most of the time, there are vacant parking spots, but the drivers do not have any information about them. It could be that either the free parking space is far from them, or some cars or objects hide it. In some cases, parking lots are managed by people who might not have the entire view of the next available parking space. Sometimes the driver has to check for a vacant space by circling in the parking lot, another driver will come, and many losses arise: fuel, time and temper.

Urban planning does not follow the quick growth of population dynamics. It implies that newly bought vehicles between two urban planning implementations, might not be accommodated in all the existing parking facilities. It leads to bad management of the space by the drivers. This research aims to use of the existing video-monitored parking areas to manage them efficiently. According to Shoup [2006], in Cape Town where the average population in 2006 was 3,239,768 inhabitants, people spend on average 12 minutes to find a space to park their car [www.westerncape.gov.za 2006].

This work explores some techniques and proposes a solution to the parking space detection problem. One significant step towards informing the driver about the status of the spot is to classify it as vacant or occupied. Image classification is an active area of research that gives the ability to a system to interpret the content of an image accordingly. One significant advantage of using image classification techniques is the cost of the operation which is relatively cheaper than using physical sensors, as discussed in Section 3.1.

Image classification requires many parameters for the system to build patterns that will be useful for the distinction between the different categories object to classify. Feature engineering is at the heart of classification. A feature defines a specific pattern in the environment of interest in voice, text or image data. A set of features or feature vectors represent an object to classify in the language understood by the classifier. The image classifier then collects all the feature vectors, to distinguish between all the classes involved to infer later the class of an object presented to it.

Looking for the best features to classify a parking spot is the key to success in this problem. Tschentscher *et al.* [2015] used as feature extractors the Histogram of Oriented Gradients (HOG) and the color histograms. Finally, the Local Binary Patterns (LBP) and the Local Phase Quantization (LPQ) also correctly describes the pattern generated by the spots. Those features are usually called handcrafted features, which are tedious to craft due to the number of parameters involved. Support Vector Machines provide a fast classification technique that allowed Tschentscher *et al.* [2015] and Wu *et al.* [2007] to design some solutions to classify the spots.

Handcrafted features and features selection are critical and can be used with the SVM. Another way of getting features is to use a deep learning approach, with CNNs for instance, where they are

no longer handcrafted as they are part of the learning process itself, to understand what the features should be. Although those handcrafted features might be more potent than the ones generated by the CNN, as they are engineered primarily for a purpose, those generated by the CNN tend to produce a lower error-rate. Amato *et al.* [2016] and Valipour *et al.* [2016] designed some CNN based systems for classifying parking spaces and obtained better results by some state-of-the-art methods. A hybrid approach that uses automatically learned features from a deep learning technique might produce better results with other classification methods.

Beyond classifying a parking space as vacant or occupied, this work develops an approach of detecting potential spots, using the Region Based Convolutional Neural Networks, and later classifying each part. After detecting and classifying potential bays, different classification methods are compared by efficacy and efficiency to ensure the best experience of drivers when requesting a parking spot using the system in real time.

The remainder of this dissertation is organized as follows organized. Chapter 2 provides a technical background regarding the techniques used in the research. That chapter discusses essential concepts: Convolutional Neural Networks (CNN), Support Vector Machines (SVM) and feature engineering. Those three concepts are important to understand the rest of the document.

Chapter 3 provides the related work from the literature that focuses on the different approaches used for the parking space detection problem. Sensor-based approaches discussed in the first part of the chapter shows the motivation of the research of using video cameras since physical sensors on the ground are accurate but cumbersome. The second part of the chapter discusses the video-based methods using traditional image processing and modern Machine Learning techniques.

Towards implementing the system, Chapter 4, 5 and 6 respectively discuss the dataset design, the detection of individual parking spots in an image and finally the classification methods used after each spot is detected.

Chapter 7 provides an evaluation of the methods developed in the preceding chapters. After that evaluation, Chapter 8 summarizes the results of this research into a system that informs the user about vacancies of a parking lot.

Chapter 9 concludes the dissertation and recommends some future work and possible extension of the research project.

Chapter 2

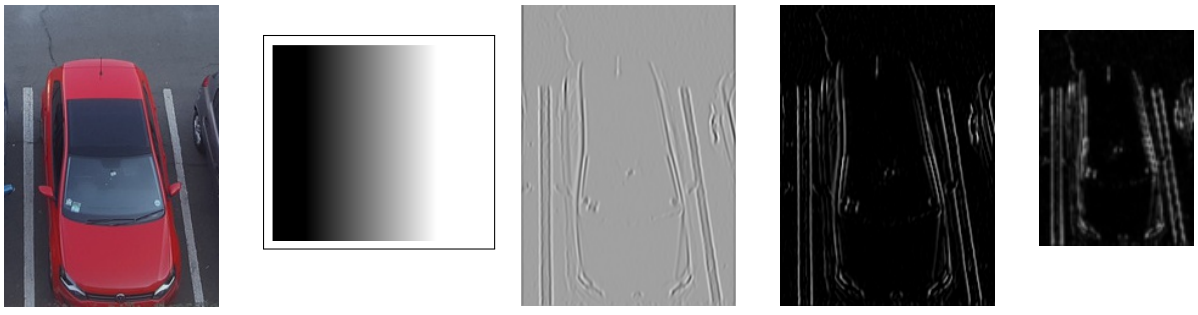
Background

This chapter introduces Machine Learning, image processing and other mathematical concepts that form the foundation of this work.

The first part of this chapter discusses Convolutional Neural Networks for classification, followed by the Region based Neural Networks for localization for detection of a parking, Support Vector Machines and the feature engineering are exposed in this Chapter.

2.1 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a technique of machine learning used for pattern and object recognition. They are similar to the human neural networks built with synapses (weights) and neurons, pooling and convolution layers. To recognize objects, features need to be detected and extracted. Engineering the optimal features can be difficult. CNNs learn extracted features from the training set using their multiple layers. In the case of images, CNNs learn to extract features from the pixels of the input image. CNNs are computationally expensive at the training phase especially when the dataset contains many classes with many objects to classify. Fortunately, these computations lend themselves to GPU parallelization. LeCun *et al.* [1998] built the successful CNN LeNet-5, for handwritten digits recognition using the MNIST (Modified National Institute of Standards and Technology) database with an accuracy of 99.2%.



(a) (b) (c) (d) (e)

Figure 2.1: Convolution Neural Networks Process: (a) Original image. (b) Convolution kernel. (c) Convolution result. (d) ReLU result. (e) MaxPooling result

.

CNNs are made of multiple layers that perform some computation on their input. For example, a

network usually contains convolution, activation or non-linearity layer and subsampling layer followed by a number of fully connected layers with relevant activations.

Figure 2.1 shows the convolution process from the input image up to the first iteration of the sub-sampled image.

The sub-sampled image is usually smaller than the input image, as illustrated in Figure 2.1d. The image is then unrolled into a 1D feature vector that is used as the input into subsequent fully connected layers. The following sections give insight into the different Convolutional Neural Network layers.

2.1.1 Convolutional Layer

The convolution layer performs a mathematical operation called the convolution defines by the general formula:

$$C(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx \tag{2.1}$$

where f and g are input functions, C is the result of the convolution. The convolution is a linear function since:

$$(h * (w + k))(t) = (h * w)(t) + (h * k)(t) \tag{2.2}$$

Given an image A of size $I \times J$, and filter F of size $M \times N$, when A is convolved with F, the value of pixel x_{ij} of A is the weighted sum of the pixels' intensities of in the neighbouring area of A. The filter F is often called weight matrix. Based on the expansion of Equation 2.1, A convolved with F produced a new image with pixel intensities defined by:

$$x_{ij} = \sum_{m=-M}^{M} \sum_{n=-N}^{N} (a_{i-m,j-n} \times f_{m-n}) \tag{2.3}$$

where $i \in \{0,\ldots,I\}$ and $j \in \{0,\ldots,J\}$ represent the position of the pixel in the image, and $m \in \{-M,\ldots,M\}$ and $n \in \{-N,\ldots,N\}$ index over the elements in the filter. a and f are the intensities of the image A and the filter F at the relevant positions [Prince 2012].

Equation 2.3 solely relies on the size of F which should have the form $2k$+1, where $k \in \mathbb{N}^+$. From Equation 2.3, the domain of M and N includes negatives values since the middle column and row has index 0. Whatever is on the left of 0 is negative, and positive otherwise. The gray column and row in Figure 2.2 correspond to the coordinates of each entry of the filter F, which is the on the white background including the red cell.

	-1	0	1
-1	-1	0	1
0	-1	0	1
1	-1	0	1

Figure 2.2: 3×3 Convolution Filter Matrix.

In general, filters are not bound to the previous constraints, since there exist rectangular filters.

The convolution operation is illustrated in Figure 2.4. The original matrix in Figure 2.4a is convolved with the filter Figure 2.4b to produce the matrix in Figure 2.4c. The values in the resulting image are computed from Equation 2.3. For illustration purposes, only the pixel values are in blue in the original matrix is calculated for the convolution. The cell with the bold value corresponds to the center pixel (127) where the filter performs the convolution, on that pixel and on the 8 pixels surrounding it. The resulting image contains a green area that corresponds to the result of the convolution in

that specific area of the original image. The bold number (240) is the final value of the pixel after the convolution. Let us focus on both bold numbers and see how 127 is transformed to 240.

Let A the original matrix with $a_{i,j}$ a pixel value of A and F the filter with $f_{m,n}$ a weight value.By Equation 2.3, the resulting pixel at position $x_{1,1}$:

$$\begin{aligned} x_{1,1} &= a_{1-(-1),1-(-1)} \times f_{-1,-1} + a_{1-(-1),1-0} \times f_{-1,0} + a_{1-(-1),1-1} \times f_{-1,1} + \\ &\quad a_{1-(0),1-(-1)} \times f_{0,-1} + a_{1-(0),1-0} \times f_{0,0} + a_{1-(0),0-1} \times f_{0,1} + \\ &\quad a_{1-1,1-(-1)} \times f_{1,-1} + a_{1-1,1-0} \times f_{1,0} + a_{1-1,1-1} \times f_{1,1} \\ x_{1,1} &= a_{2,2} \times f_{-1,-1} + a_{2,1} \times f_{-1,0} + a_{2,0} \times f_{-1,1} + \\ &\quad a_{1,2} \times f_{0,-1} + a_{1,1} \times f_{0,0} + a_{1,0} \times f_{0,1} + \\ &\quad a_{0,2} \times f_{1,-1} + a_{0,1} \times f_{1,0} + a_{0,0} \times f_{1,1} + \end{aligned} \tag{2.4}$$

By replacing $a_{i,j}$ and $f_{m,n}$ by their values coming from Figure 2.4:

$$\begin{aligned} x_{1,1} &= 255 \times (-1) + 127 \times 0 + 240 \times 1 + \\ &\quad 127 \times (-1) + 127 \times 0 + 127 \times 1 + \\ &\quad 0 \times -1 + 0 \times 0 + 255 \times 1 \\ &= 240 \end{aligned} \tag{2.5}$$

Provided the above illustration, there exist many filters that are used for the convolution which are, but not limited to the Gaussian, Sobel, Laplacian Filters and Robert Edge Cross Detector. Figure 2.3 the Gaussian and Robert Filters.

CNNs are built of many convolution layers which contain a number n of $r \times r$ filters. Those layers convolve with each filter with the input image to produce n feature maps, as illustrated in Figure 2.5. The stride sets the number of pixels to skip between placing the convolution filter in the input image. For example, given an image I and a filter F, F convolves with I starting by the top-left pixel of I, sliding to the right. The step taken by F from one pixel to the next is the stride. It aims to reduce the size of I while keeping the relevant information for the next processing.

The convolution as defined earlier will run into some problems processing the entries located at the edges of the image, since the center entry of the filter should be on top of each pixel. Therefore its surrounding entries will hit nothing on pixel $x_{0,0}$ of image I for instance. To overcome this issue, one can use the zero padding as illustrated in Figure 2.5. The original image is of shape 5×5×3. The image is convolved with two filters of shape 3×3×3×2 (2 weight matrices of shape 3×3×2). In that instance, the filter cannot process the pixel at the very first location (top left corner). The zero padding allows a layer of zero's outside the defined region of the image for the filter to go through all the pixel of the image. The zero padding also helps to decide what to do when the filter goes over the edge of the image. To do so, the input image is padded with a v layers of zeros around the image, so that the result of the convolution by the filter F does not decrease too much the spatial dimension of the output image. For CNNs design, those values are calculated with the following formulas:

$$z - pad = \frac{F_s - 1}{2}, \tag{2.6}$$

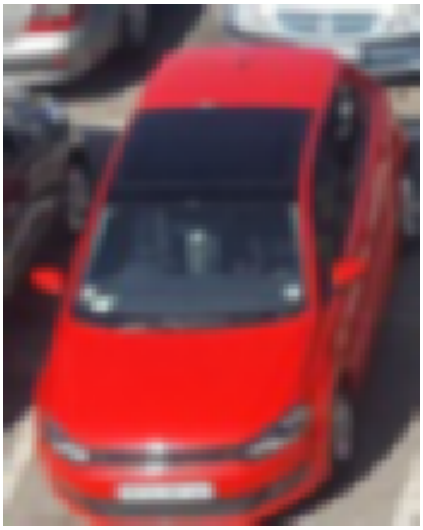
where F_s is the filter size of the image I, $z - pad$ is the size of the padding around I before the convolution by the filter F. The output size of the resulting image of a convolution is also important when designing very deep CNNs. That size is expressed as follows:

$$output = \frac{I_s - F_s + 2 * pad}{stride} + 1, \tag{2.7}$$

where I_s is the size of the input image, F_s is the filter size, pad is the size of the padding around the input image and $stride$ is the stride used by the filter [Deshpande 2017].



(a) Original Image



(b) Gaussian Filter on image (a)



(c) Roberts Filter on image (a)

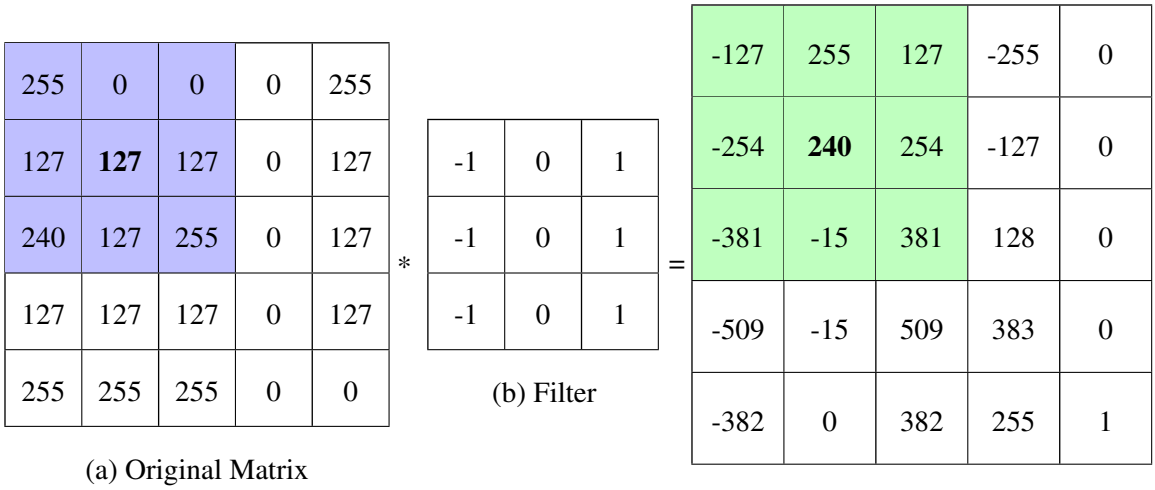
Figure 2.3: Different Convolution Filters applied on a image

.

The feature maps represent the activation of all the features detected from the image through the convolution layers. After the convolution layer is applied to the input image, the feature maps block is a $z \times k \times n$ where $z \times k$ is the size of a resulting feature map from a filter, n is the number of filters used for the convolution.

2.1.2 Non-Linearity/Activation Layer: Rectification Linear Unit (ReLU)

Convolution layers perform convolution operations on input images, which is a linear as illustrated in Equation 2.2. If a network is made of convolution layers only, the resulting classifier will act like one big linear system. It will not adapt to more complex and non-linear data points, therefore will not be able to detect all the different features from an input image. To generate a classifier that can adapt to non-linear input data, activation functions need to bring non-linearity to the result of the convolution layers. An activation function fires when its input is above a threshold value. Those activation functions should also be differentiable so that the computation of the error through backpropagation allows the



(a) Original Matrix

(b) Filter

(c) Resulting Matrix

Figure 2.4: The filter (b) convolves on the matrix (a) to produce (c): The Convolution operation

gradient descent to get to the optimal values of the parameters – at least localy.

There exists many activation functions, but the most successful are the following functions: Sigmoid, Tanh, and ReLU. Each of those functions receive as input the linear combination of all the weights (W_i) of the previous neurons or the resulting feature map after convolution added to a bias (b) vector as follow:

$$\theta = \sum_{i=1}^{n} W_i \cdot X_i + \mathbf{b} \tag{2.8}$$

The Sigmoid and the Tanh are good activation functions from their structure in terms of their range and derivative. They respectively have a range of (0,1) and (-1,1). One good property of those functions is their derivatives. They can be expressed in terms of the original function as follows: Let $S(\theta)$ and $T(\theta)$ be the Sigmoid and Tanh functions respectively.

$$\begin{aligned} S(\theta) &= \frac{1}{1+e^{-\theta}} \\ T(\theta) &= \frac{e^{\theta}-e^{-\theta}}{e^{\theta}+e^{-\theta}} \end{aligned} \tag{2.9}$$

From Equation 2.9, the following derivatives can be produced:

$$\begin{aligned} S'(\theta) &= (\frac{1}{1+e^{-\theta}})(1-\frac{1}{1+e^{-\theta}}) \\ &= S(\theta)(1-S(\theta)) \end{aligned} \tag{2.10}$$

and

$$T'(\theta) = 1 - tanh^2(\theta) \tag{2.11}$$

Equation 2.10 and 2.11 show that the computation of the derivative of those functions during the back-propagation is more efficient since they are expressed in terms of the original function value, which avoids the need for numerical approximations. It also assists with speed, as the function value is already calculated in the forward pass through the network.

Unfortunately for deeper networks, those functions both suffer from vanishing gradient and saturation of neurons during training. The vanishing gradient is caused primarily because of the range of the derivative of each function which is very small. Whatever the size of the input is, it will get

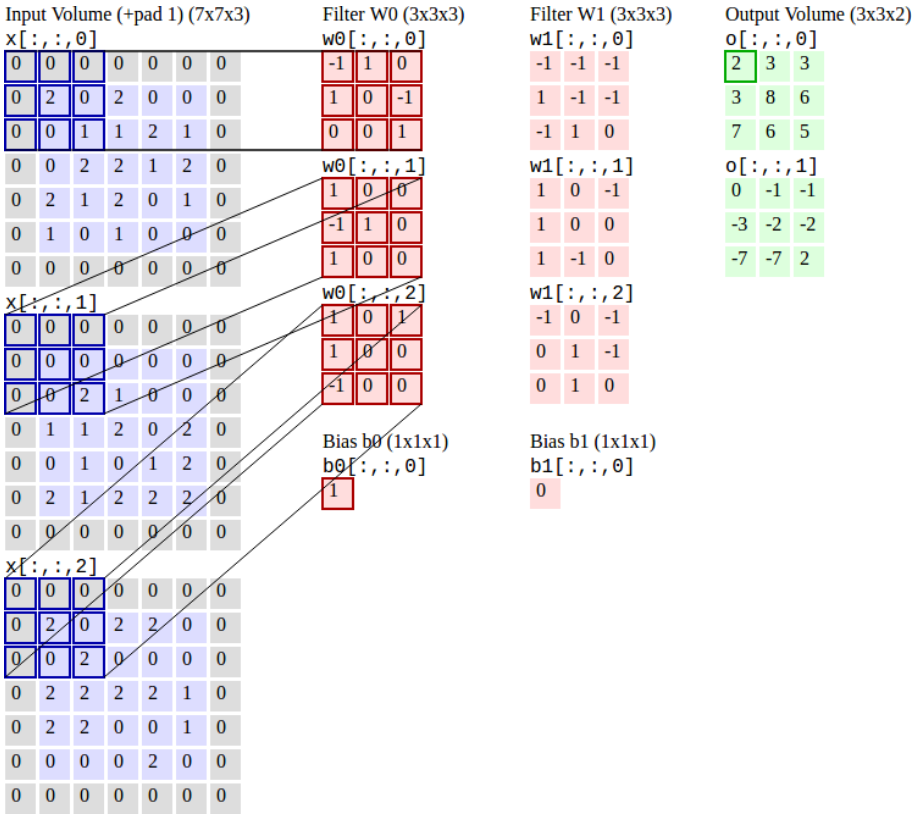


Figure 2.5: CNNs Filters in action. The filters x 2 (pink) are applied to the input image (blue) that has 3 channels. Since 2 filters have been used, 2 feature maps are produced, [DL4J 2017].

squeezed to either (0,1) for the Sigmoid and (-1,1) for the Tanh function. This means that the layer on top of the network might not learn anything from the earlier layers since they do not receive any signal. The saturation of neurons usually occurs where a neuron or a set of neurons is over-confident about the response of the processed input, or if the initialization if the weights are too high. In both cases, the activation will return values close to the most significant boundary of the range which is 1. Introducing any of those two functions in a CNN might lead to bad results.

The ReLU function is less expensive in computation than the others, that is the reason why it is used in many CNN architectures.

The output of the convolution layer may contain some negative values in some feature maps, which the ReLU unit can normalize. The output of this layer is produced by the function:

$$f(\theta) = max(0, \theta), \tag{2.12}$$

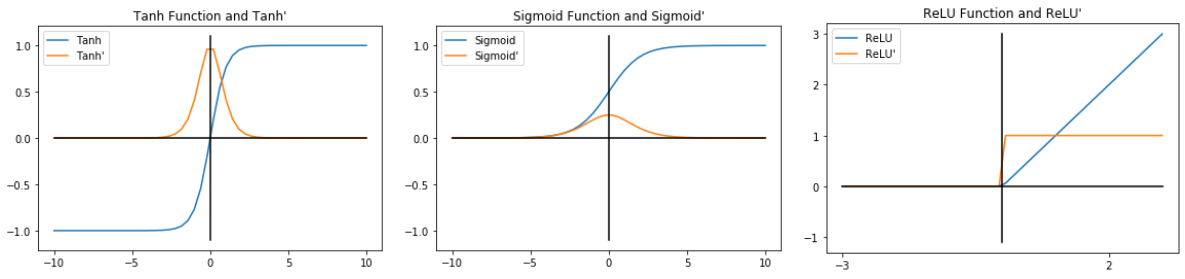
and its derivative is:

$$f(\theta) = \begin{cases} 1 & \text{if } \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

where θ is a pixel from the feature map produced by the convolution layer.

The ReLU layer is based on the non-linear Equation 2.12. As shown in Figure 2.6c, when a pixel value is less or equal to zero, then the output of the ReLU function is zero. Otherwise, the value of that pixel is returned. Compared to the Sigmoid and Tanh non-linear functions, the ReLU does not produce either the vanishing gradient or the saturation of neurons.

The ReLU operation also introduces non-linearity to the CNNs since the convolution operation is



(a) (b) (c)

Figure 2.6: Some Convolutional Neural Networks Activation Functions and Their Derivatives

linear. The output of the ReLU layer is a matrix having the same size as the feature maps produced by the convolution layer.

2.1.3 Pooling Layer

After the ReLU layer, the image should be downsized to extract the most valuable information as they (the pooling layers) provide some translation invariance and control overfitting. The kernel used is a matrix $n \times n$. The pooling layer or down-sampling layer reduces the size of the output produced by the ReLU layer. There are two types of pooling:

- **max-pooling:** the pooling layer kernel returns the maximum value from all the pixels covered by the kernel,
- **average-pooling:** instead of returning the maximum value from all the pixels covered by the kernel, the average of the pixels covered by the kernel is returned.

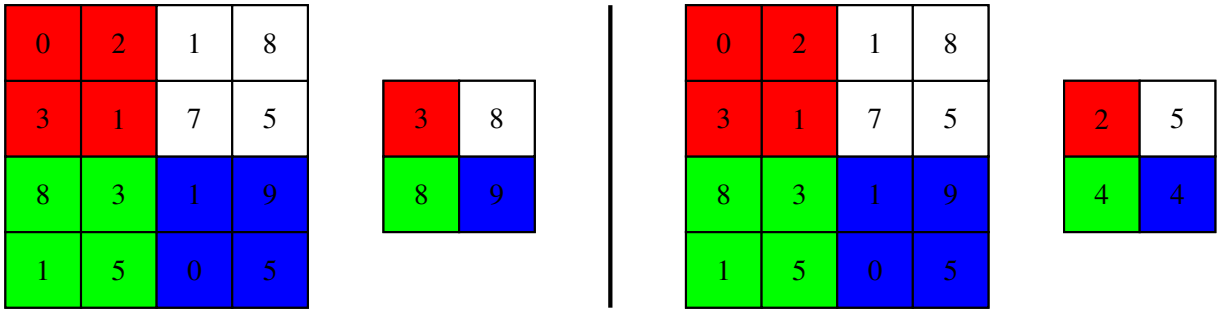


Figure 2.7: Left line: Max-pooling — Right line: Average-pooling

One of the primary purposes of the pooling layer is to reduce the number of parameters to send forward, which means less computational power required deeper in the network. Given the output of the ReLU, the pooling (Max-pooling) returns a new image containing information about the presence of a feature a feature. The location of the feature is useless at this level. Figure 2.7 demonstrates the pooling layer in action. The setting is made of a kernel of size 2 x 2, and the stride is 2. The left-hand side shows the result of the max-pooling. Similarly, the right-hand side shows the result of the average-pooling. That pooling method returns the ceiling of the average of the pixel value convolved with the kernel.

2.1.4 Dropouts

Dropout layers are essential in the network to help avoid overfitting and co-adaptation of neurons. The role of this layer is to turn off, randomly some neurons to avoid the network overfitting the data during and to stop neurons co-adapting during training. The proportion of neurons to be deactivated is set beforehand and is usually around 50The dropout option is active only during training and deactivated during testing when the neuron outputs are scaled instead.

2.1.5 Fully Connected Layer

Following the previous layers, there are fully connected layers where all neurons are connected to all the inputs as done a typical artificial neural network. At the end of that layer, the classification happens using a classification function, the popular being the softmax activation. The number of neurons at the output is the number of classes on which the network is trained. Each output neuron produces a value, where the values produced by the neurons that are normalized sum to 1 using a softmax approach, the highest amongst all get activated, and the correct class is returned as final output. While training, the classification needs to be corrected using the backpropagation, since the convolutional neural networks are a supervised learning process.

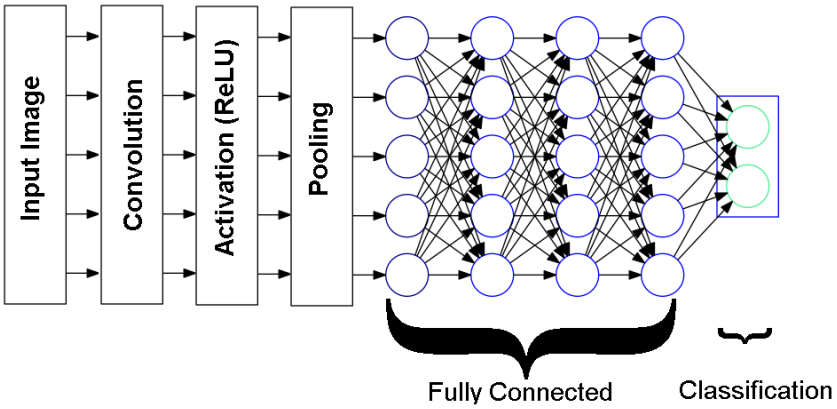


Figure 2.8: Convolutional Neural Networks and Fully Connected Layers

As in a typical Artificial Neural Network, each neuron has an activation function that gets triggered based on its input. There are several activation functions, but the most used one is the Rectification Linear Unit [Hahnloser *et al.* 2000].

Figure 2.8 shows the Fully Connected Layers from a Convolutional Neural Networks configuration. The most important part is the Fully Connected layers where the Artificial Neural Networks learn all the features coming from the Convolutional Neural Networks, up to the very last layer which is the classification one [Bishop 2006].

2.1.6 Classification

Convolutional neural networks are mostly used for classification problems. Let's assume that the classifier learns how to distinguish between two classes, the Softmax function produces a probability vector define by the probabilistic formula:

$$\hat{y} = P(y_i = k|x_i, \theta) = \frac{e^{\theta_k^T x_i}}{\sum_{k=1}^{K} e^{\theta_k^T x_i}} \tag{2.13}$$

where $P(y_i = k|x_i, \theta)$ is the probability assigned to the true label y_i given the item x_i, parametrized by θ and X is the dataset such that $x_i \in X$, and k iterates over the two classes, which belong to K. In this work, y is one-hot encoded, and $\hat{y}$ result from the CNN classification function which is the Softmax from Equation 2.13, which is a vector of size $D \times 1$, where D is number of classes. The highest score returned in the vector $\hat{y}$ determines the class of the item x_i. To check the rightness of the classification, the loss function needs to compute the error rate update the parameter if needed.

2.1.7 Training

The previous layers contribute to the training process that initially sets and updates all the weights of the CNN so that they get optimized later on for a specific classification task. Backpropagation is responsible for computing the weights updating of the network. The weights include the convolutional filters and the fully connected synapses.

The CNN training process needs a lot of data. Although, the amount of data required to train a CNN is an active area of research. At the beginning of this work, the training was made of only 1000 images. Later on, that amount increased with the data collected. Given that data, there are four steps towards adequately training a network.

The forward pass consists of processing an input image from the training set on the whole network until the very last layer, which is the classification layer. All the weights are randomly initialized, therefore convolution filters and fully connected weights fire randomly given the image. At this stage, the classification is expected to be erroneous, since the parameters are not optimized.

That leads us to the loss function $J(\Theta)$. That function measures the classification error. The network is fed with labelled data, which means that during the training, the resulting class given the input image can be compared to the original class for accuracy. There are many loss functions exist for this purpose, but the most used are the Mean-Squared Error (MSE) and Cross-Entropy loss which are defined as follows:

$$\text{MSE}(y, \hat{y}) = J(\Theta) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.14}$$

$$\text{Cross-Entropy}(y, \hat{y}) = J(\Theta) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{2.15}$$

where Equation 2.14 and Equation 2.15 are respectively the Mean-Squared Error (MSE) and the Cross-Entropy loss functions. N is the training set size.

For this process, it is now common to use label the data using the one-hot encoding approach. It creates, for each data item a vector of the size of the number of classes, and assigns the value 1 to the index corresponding to the class of the item in the dataset. For example, let three classes to classify: Dog, Bird, Cat. Instead of assigning a random code to each class, one-hot encoding works by giving each class a specific position in the 1D vector:

[Dog Bird Cat]: specific order of the vector.

A sample of a one-hot encoded data looks like:
[1 0 0] = Dog,
[0 1 0] = Bird,
[0 0 1] = Cat.

Now let's assume that the Softmax from Equation 2.13 produces the following vector for the class Dog: [0.6, 0.1, 0.3]. The MSE and the Cross-Entropy produce respectively 0.26 and 0.22 as loss values. Although this is one case out of many, the cross-entropy emphasizes more on the correct class

compared to the MSE that has misclassification scores including in its result. To illustrate the previous claim, Figure 2.9 simulates multiple computations of classification losses using both Mean Squared Error and Cross-Entropy loss functions. The x-axis represents the decision-score of the classifier for the first class. From Equation 2.15, that corresponds to $\hat{y}$ only given the one-hot encoding that assigns 1 to y. For Equation 2.14, $\hat{y}$ and y are assign all the indices of the classification and the one-hot encoded label vectors, which means that the Mean Squared Error also has misclassification score in its error result. In Figure 2.9, from decision-score [0, 0.558], the MSE has a lower loss than the Cross-Entropy. That side of the decision-score means misclassification. For decision-score (0.558, 1], the Cross-Entropy produces better losses.

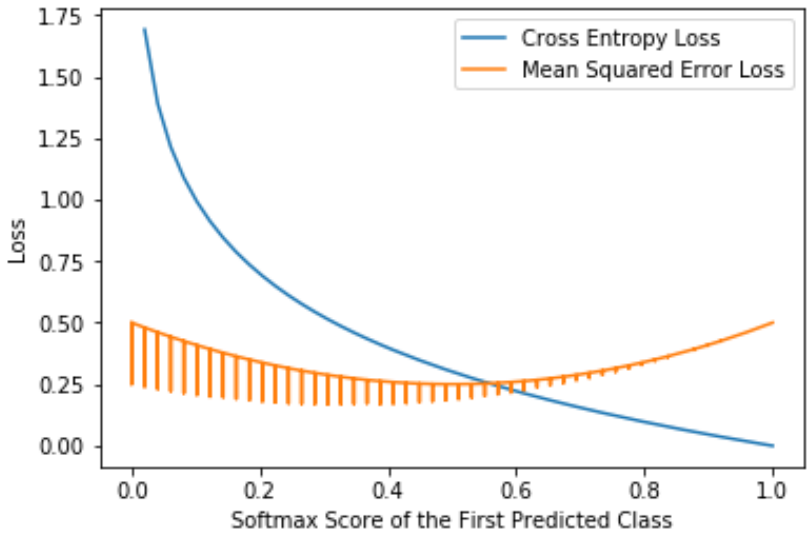


Figure 2.9: Simulation of computation of Classification Loss: Mean Squared Error vs Cross-Entropy

After the computation of the loss function, in case of a bad score (if the current score did not reach the target), the backward pass or backpropagation is run through the network to adjust the weights that are contributing to the high value of the loss. Every neuron is connected to another one with a synapse or weight. The weight set is $\mathbf{\Theta} = \theta_{d,s}^{(a)}$, where a is the layer number, d is the index of the connected neuron and s is the index of the current neuron.

The backpropagation aims to adjust $\mathbf{\Theta}$, set of all weight parameters so that the loss generated during training gets minimized. The weight update is the process of computing the derivative of the loss function with respect to all the weights parameters of the network. It is usually performed using an optimizer, the Stochastic Gradient Descent (SGD) for example. The gradient descent is the widely used optimizer for adjusting the weights defined by the closed form expression:

$$\theta_{d,s}^{(a)} \leftarrow \theta_{d,s}^{(a)} - \alpha \frac{\partial J(\Theta)}{\partial \theta_{d,s}^{(a)}} \tag{2.16}$$

where α is a learning rate, which is initially chosen to be 0.01. It was then updated during training according to the learning rate policies (fixed, exponential or step decay). Many other optimizers are also used besides the Stochastic Gradient Descent defined above: Adaptive Gradient, Adam, or Nesterov's gradient.

2.1.8 Popular Convolutional Neural Networks

Convolutional neural networks have been implemented by LeCun *et al.* [1998] for handwritten digits recognition. Figure 2.10 shows one of the very first successful convolutional neural networks, LeNet5

that was successfully used for bank check recognition systems in the United States.

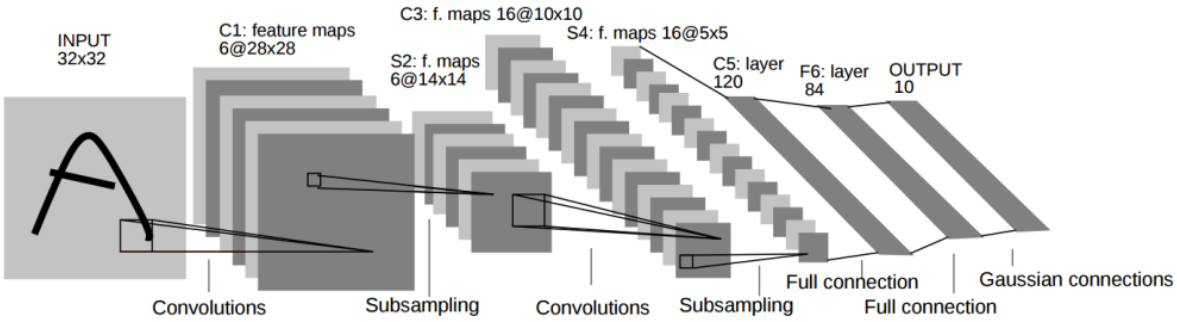


Figure 2.10: LeNet5, [LeCun *et al.* 1998].

LeNet5 is a relatively small network that has few convolution layers. The input is a 28×28 grayscale image. The number of classes is ten corresponding to the ten digits from 0 to 9.

More recently, convolutional neural networks performed as well as the state-of-the-art algorithms for image classification. AlexNet, for example, won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge). It takes 256×256×3 images as input and has 1000 classes. While LeNet5 can run on CPU, AlexNet required for training two GTX 580 3GB GPUs. From Figure 2.11, the difference in architecture is to be noted compared to the architecture of the LeNet network. The two different networks classify various categories of objects and AlexNet is inspired by LeNet5. Table 2.1 shows the comparison between AlexNet and LeNet networks.

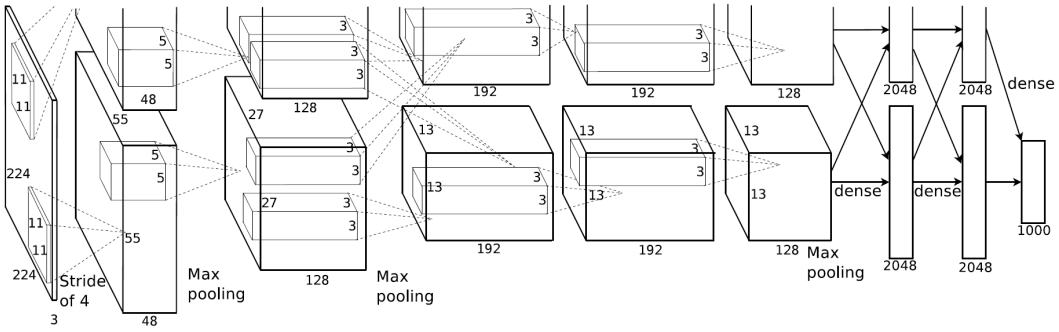


Figure 2.11: AlexNet, [Krizhevsky *et al.* 2012].

CNN Architectures	LeNet5	AlexNet
Classification task	digits	objects
Number of classes	10	1000
Image size	28x28x1 (grayscale)	256x256x3(RGB)
Training set (Samples)	60000	1.2 million
Units (Neurons)	8084	658000
Parameters	60000	60 million
Connections	34400	652 million
Total operations	412 billion	200 quadrillion

Table 2.1: LeNet5 vs AlexNet, [Li 2017]

2.1.9 Special Case: Region-based Convolutional Neural Network

Normal CNNs classify an object from an image containing it. However, they cannot locate or scale the classified object. The Region-based Convolutional Neural Networks (RCNNs) overcome this issue by generating region proposals based on image segmentation and selective search processes, then classifies those regions based on the training classes.

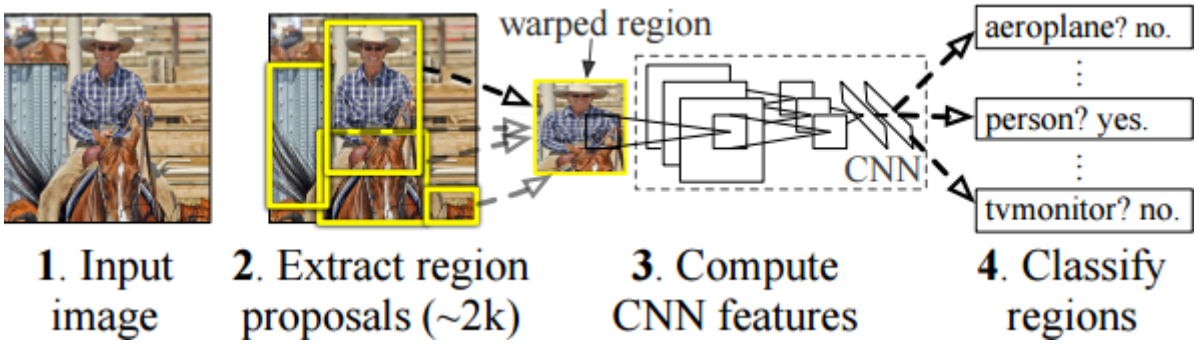


Figure 2.12: Object Detection Process with RCNN, [Girshick *et al.* 2014].

As illustrated in Figure 2.12, given the input image, RCNNs produce region proposals from a search algorithm, the selective search, to classify the selected regions within an image from the training classes. The search can return a thousand regions based on the selective search algorithm.

RCNNs primarily rely on image segmentation [Felzenszwalb and Huttenlocher 2004]. An image can be taken as a connected graph of a set of vertices $v_i \in V$ and edges $(v_i, v_j) \in E$. The segmentation gives each edge a weight w which is the strength of the neighboring vertices connecting that edge. For the case of an image, vertices are pixels. Therefore, based on the weight w, the image can be segmented into regions. Each region shares similar properties (pixel intensity). Given an input image Figure 2.13a, the result of image segmentation produces the image in Figure 2.13b.

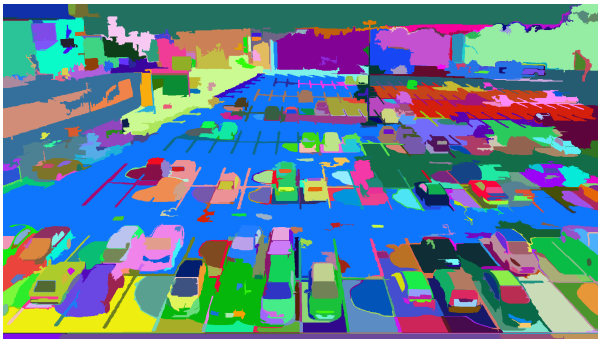
The selective search uses the results of the image segmentation to compute the presence of multiple objects in an image regardless of the scale of those objects [Uijlings *et al.* 2013]. The original algorithm uses the SVM to train a classifier based on images containing objects, and the labels are ground truth of the location of the object. The segmentation returns candidate areas that represent objects from the image. RCNNs then use the selective search algorithm to use those candidate areas for object detection. Within an image, an object can have more than one candidate box, as illustrated in Figure 2.13c. Since one box is required, the candidate box with the highest detection probability with a ground-truth image is the box selected. The detection score is computed using a standard CNN, per candidate area. Ideally, one candidate area holds, one object, but because that object can be captured more than once, the highest CNN score will define the right candidate area of the object of interest. The background is the default class of the RCNNs so that only the classes of interest are returned, and the computation time for the classification gets reduced since the background is discarded.

Figure 2.13c represents the result of the selective search algorithm just after training 1 iteration, and Figure 2.14a is the result of the RCNN after multiple iterations. Based on Figure 2.13b, the cars and other objects in the image are quickly detected, due to their shape. The ground does not create boundaries like any other objects on the images from Figure 2.13a. This is where the region based comes into play to focus on a specific part of the labelled region (empty parking bay), and get features out of it. Chapter 5 digs deeper in to learning and predicting the presence of empty parking bays using RCNNs.

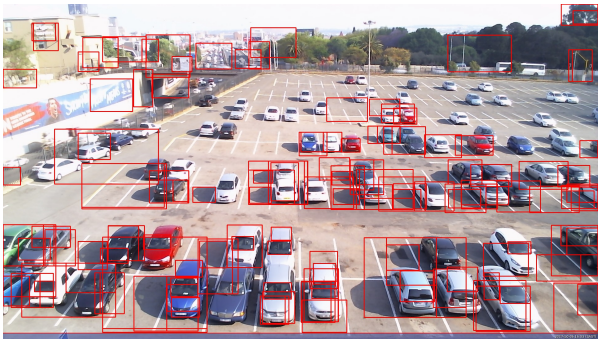
Given all the candidate regions produced by the selective search from RCNNs, each of them undergoes a typical CNN classification process as shown from Section 2.1.1 to Section 2.1.5 to return a



(a) Original Image

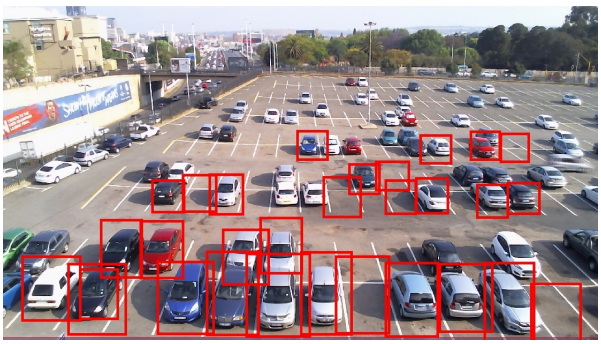


(b) Segmented Image



(c) Selective Search Based on Segmentation with Few Training Iterations

Figure 2.13: RCNN Selective Search Process



(a) Selective Search Based on Segmentation with Few Training Iterations

Figure 2.14: RCNN Selective Search Process

class per region, Figure 2.12.4.

2.2 Support Vector Machines

2.2.1 Theory

Support Vector Machines (SVM) are a supervised machine learning technique that is based on a linear decision boundary or hyperplane to separate data into different classes. They were formally built for binary classification, [Cortes and Vapnik 1995] but have since been extended to multi-class classification, [Knerr *et al.* 1990]. SVMs are a category of linear classification that draws a boundary line between the different classes of the data, using the value produced by a linear combination function which is the dot product of the feature vectors and the weight vector parameters that are learned during training.

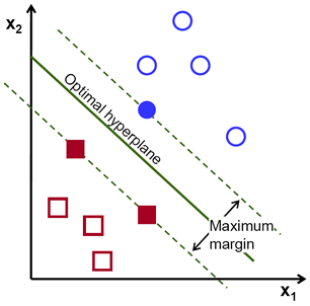


Figure 2.15: Support Vector Machines, [OpenCV 2017]

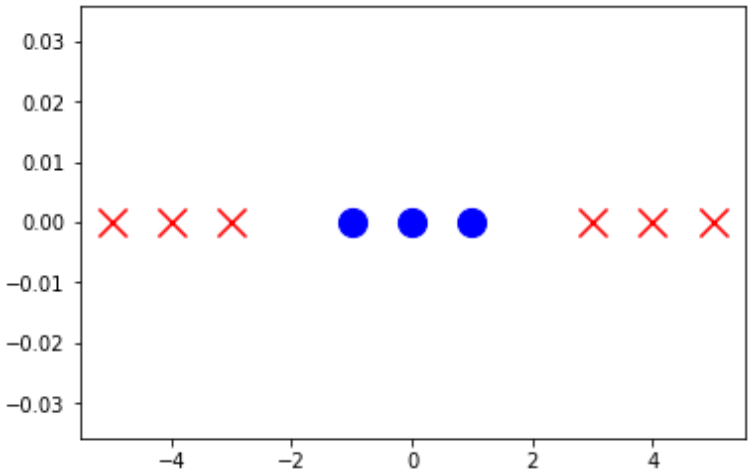
2.2.2 The Margin

The input data of the SVMs are multi-dimensional feature vectors coming from feature extraction algorithms. For example, an image can be transformed into feature vectors by using the Histograms of Oriented Gradients. The SVMs map those vectors into a high-dimensional space, then find the maximum margin distance between the lines (or hyperplanes) passing through the support vectors. As illustrated in Figure 2.15, the support vectors are the points that are the closest to the optimal hyperplane generated by the SVMs algorithm, the full red squares (2) and the full blue circle (1). The optimal margin lines are at an equal distance to the optimal hyperplane plane. In a case of a linearly separable data, the SVMs aim to maximize the margin distance around the hyperplane: this is a linear decision boundary. In the case of overlapping data vectors, soft margin allows those vectors in the margin with a very low weight in the function defining the optimal hyperplane.

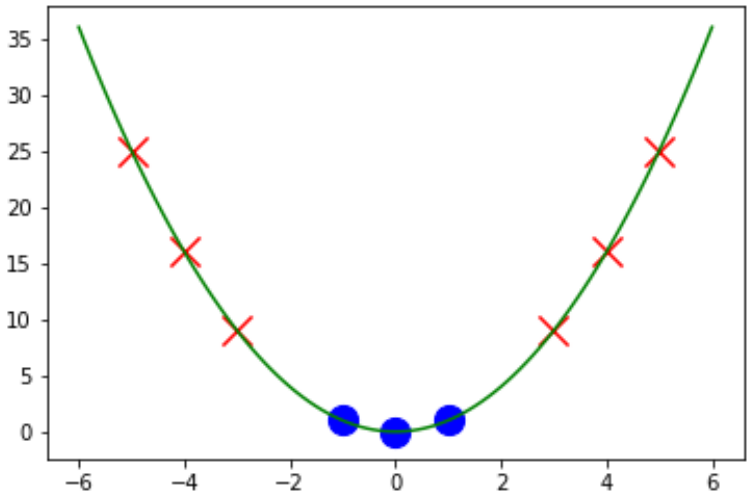
2.2.3 Linear Separability

Sometimes the data (1D) are not linearly separable, as illustrated in Figure 2.16a, since no straight line can adequately classify those points. If we expand those points with some non-linear transformations of the features, these points may become linearly separable as shown in Figure 2.16b. That black line shows that the 1D initial data embedded (non-linearly separable) in a 2D setting is now linearly separable.

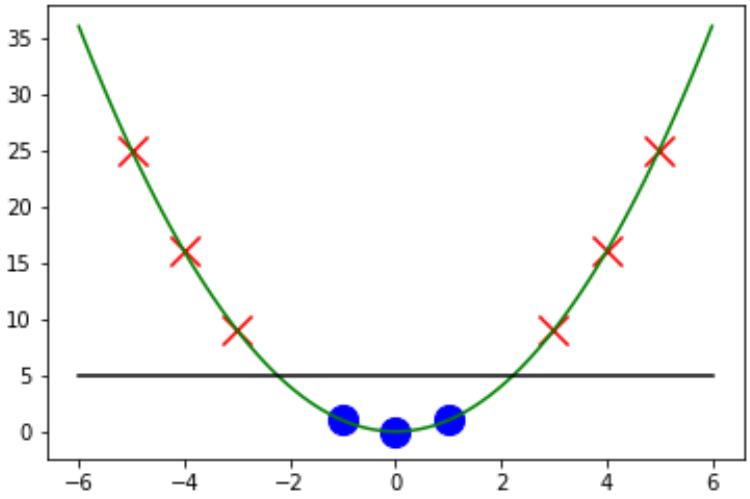
However, for higher dimensional vectors, the feature vector becomes longer, the computations are more expensive, finding a hyperplane is harder because it is in a higher dimensional space. The kernel function allows us to do the necessary mapping to reduce the computational complexity.



(a) Non Linearly Separable Data points



(b) Embedding of 1D Data into 2D setting



(c) Linearly Separable Data points

Figure 2.16: Linear Separability of Some Data Points

2.2.4 Kernels

A kernel is a mathematical function that takes as input some feature vectors in a given dimension space and efficiently maps it into a higher dimensional space. When the number of features is high, and the vectors are not linearly separable, trying to explicitly transform the dataset from the current dimensional space to a higher dimensional space is expensive in terms of computation and memory.

To illustrate this, let us use the Radial Basis Function Kernel to embed the vector $\mathbf{x}$ = [x_1,x_2] into a higher dimensional space. In the weight space there is $\mathbf{w}$ =

That produces the following result:

$$\begin{aligned} \text{Let } \mathbf{x}, \mathbf{w}, K(\mathbf{x}, \mathbf{w}) &= exp(-\lambda||\mathbf{x} - \mathbf{w}||^2) \\ &= exp(-\lambda(\mathbf{x} - \mathbf{w}) \cdot (\mathbf{x} - \mathbf{w})) \\ &= exp(-\lambda((\mathbf{x} \cdot \mathbf{x}) - (\mathbf{x} \cdot \ \mathbf{w}) - (\mathbf{w} \cdot \mathbf{x}) + (\mathbf{w} \cdot \mathbf{w})) \\ &= exp(-\lambda(||\mathbf{x}||^2 - 2(\mathbf{x} \cdot \ \mathbf{w}) + ||\mathbf{w}||^2)) \end{aligned} \tag{2.17}$$

The previous equation implies that the new feature vector of $\mathbf{x}$ in higher dimensional space is:

$$\Phi x = [e^{-\lambda||\mathbf{x}||^2}, e^{-\lambda 2(\mathbf{x} \cdot \ \mathbf{w})}, e^{-\lambda||\mathbf{w}||^2}] \tag{2.18}$$

The result of Equation 2.17 for embedding in higher dimensional space produced a feature vector of dimensionality 3 for an initial vector of dimensionality 2. This implies that the kernel done in a higher dimensional space is expensive. The kernel trick solves the problem by applying the dot product in a lower dimensional space for the same results.

2.2.4.1 The Kernel Trick

From feature vectors transformed to a higher dimensional space, the kernel computes the dot product of all the points in that new space which is more expensive. The kernel trick computes the final dot product using the lower dimensional vectors. By definition, the kernel is derived as follows:

Let $\mathbf{x} \in \mathbf{X}$ be a data point and $\mathbf{K} : X \times X \to \mathbb{R}$ is a kernel function. We choose $\mathbf{K}$ such that:

$$\exists \Phi : \mathbf{X} \to \mathbb{R} | \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j}), \tag{2.19}$$

where $\mathbf{X}$ contains all the data points. In this way, we can efficiently compute the kernel value as the dot product of two simpler functions. Since the calculation of the dot product of all the features vectors in a giant data space is not done explicitly, that is why it is called kernel trick.

In that new space, hopefully, the vectors are linearly separable, and the SVMs can be applied for classification. For example, a polynomial of high degree has d separate terms that need to be calculated. Whereas, in Equation 2.21, individual polynomial terms calculations sums up to computing the dot product, then raising the result to the power d.

2.2.4.2 Types of Kernel function

In the different cases of kernels, X is the set of data points, feature vectors of the dataset. $\mathbf{x_i}, \mathbf{x_j}$, belong to X and K is the kernel used to map the points of X to a higher dimensional space.

- **Linear Kernel**: Appropriate when the data is linearly separable. Its expression derives from Equation 2.19:

$$\forall \mathbf{x_i}, \mathbf{x_j} \in X, \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j}, \tag{2.20}$$

which is the representation of a straight line. As shown in Figure 2.17a, the data is linearly separable.

- **Polynomial Kernel**: Given a non-linearly separable data, the polynomial kernel performs some combinations with all the feature vectors to hopefully get a hyperplane to separate the data in a higher dimensional space. As illustrated in Figure 2.17b, the data is not linearly separable, and in that instance, a quadratic function separated well the data for classification. The polynomial kernel is expressed as follows:

$$\forall \mathbf{x_i}, \mathbf{x_j} \in X, \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + c)^d, c \in \mathbb{R}^+, d \in \mathbb{N}^+ \tag{2.21}$$

where c is the trade-off between the maximum margin separator and the classification error. The linear kernel is a special case of the polynomial kernel with d=1 and c=1.

- **Radial Basis Function Kernel**: The Radial Basis Function or RBF or Gaussian kernel is a one-dimensional probability density function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(- \frac{(x-\mu)^2}{2\sigma^2} \right), \tag{2.22}$$

where μ is the mean of the distribution, x is a sample point of the dataset.

Similarly to the polynomial kernel, the RBF classifies non-linearly separable data. In this particular case, the data might be severely scattered in a low dimensional space. The kernel mapping will project all the data points to the new n-dimensional space where the different classes are correctly clustered for better classification. Figure 2.17c illustrates the RBF kernel.

The RBF function comes from the exponential part of Equation 2.22 and is expressed as follows:

$$\forall \mathbf{x_i}, \mathbf{x_j} \in X, \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = exp\left(- \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\sigma^2} \right). \tag{2.23}$$

Equation 2.23 is reduced to the well-known form:

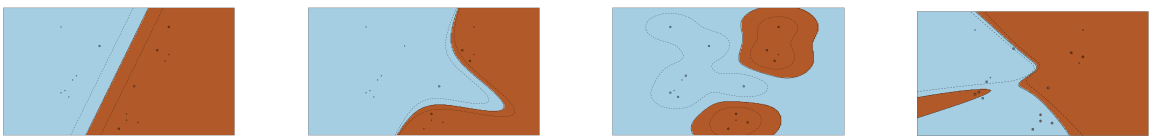
$$\forall \mathbf{x_i}, \mathbf{x_j} \in X, \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = exp(-\lambda||\mathbf{x_i} - \mathbf{x_j}||^2), \tag{2.24}$$

where $\lambda = \frac{1}{2\sigma^2}$. This is the result of the kernel trick applied on x_j and x_i, which is less expensive than the computation of the kernel done in Equation 2.17.

- **Sigmoid Kernel**: The Sigmoid kernel is a linear kernel embedded in a $\tanh$ function which makes it a non-linear function. It can be expressed as follows:

$$\forall \mathbf{x_i}, \mathbf{x_j} \in X, \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\alpha \mathbf{x_i} \cdot \mathbf{x_j} + \mathbf{c}), \tag{2.25}$$

where α and c are respectively the slope and the y-intersect of the line $\alpha \mathbf{x_i} \cdot \mathbf{x_j} + \mathbf{c}$. The Sigmoid kernel is illustrated in Figure 2.25.



(a) Linear Kernel (b) Polynomial Kernel (c) Radial Basis Function Kernel (d) Sigmoid Kernel

Figure 2.17: SVM Kernels, [ScikitLearn 2017]

2.3 Feature Engineering

In machine learning, features are referred as inputs of a potential classifier. A set of features represents the description of an entity. For example, to represent a car in an image, some patterns can be extracted from the image to represent that car based on edges, corners, the distribution of the colors.

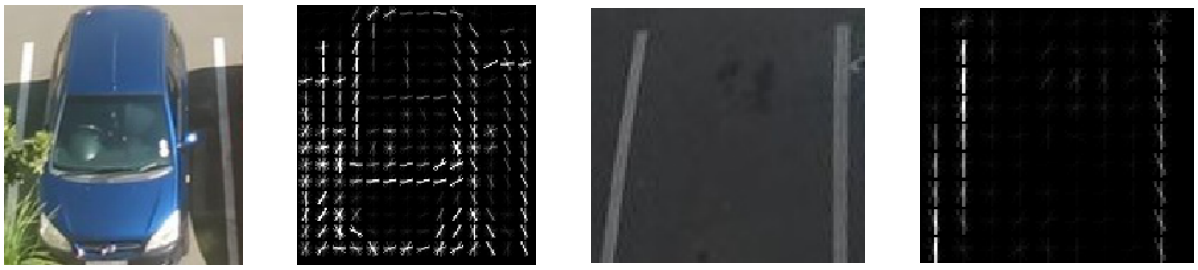
In statistics, features for a potential model(classifier) can be easily removed if not needed for the classifier to be trained faster. That process is called feature selection. For instance, to predict the class rate of a student in a class, given all his/her registration records and marks, the first name, the address, phone number and the email address can be easily removed from the initial set of features that represent a student. In computer vision, features are generated from the pixels of an image. A pixel alone cannot represent anything from an image, but a set of pixels, given their location and intensity can give an identity of an object in an image. The process of generating useful mathematical descriptors of images using the raw pixel information is called feature extraction.

There exist many techniques for extracting features from an image. The Histograms of Oriented Gradients (HOG) and the Convolutional Neural Networks (as features extractors) became very notorious in many application because of the properties they have given the accuracy of the features they describe.

2.3.1 Histogram of Oriented Gradients

Histogram of oriented gradients was initially designed to extract features for human detection, in particular, and object detection in general [Dalal and Triggs 2005]. This computer vision technique computes the orientation of the gradient of an input image to extract important information, mostly shapes like edge directions and corners. It generates a feature vector based on those captured orientation gradients and can be fed to a classifier like Artificial Neural Networks or SVM. The success of the HOG quickly grew and is used for many other applications, for instance, face recognition.

Figure 2.18 is an intuition of the Histogram of Oriented Gradients (HOG) algorithm. The gradients are computed based on the different shapes found in the image. For instance, Figure 2.18b, represent the gradients from the edges and corners from Figure 2.18a. Similar remarks for the vacant parking bay from Figure 2.18c and its corresponding HOG representation. Those gradients, when correctly tuned, give a correct representation of the object to detect.



(a) Original Image (b) Gradient Vectors (c) Original Image (d) Gradient Vectors

Figure 2.18: Gradients Computed from Histograms of Oriented Gradients algorithm.

To perform the histogram of oriented gradients on an image, one needs to follow some steps:

2.3.1.1 Gradient Computation

To get the Histograms of Oriented Gradients, the gradients are calculated from the image, by convolving a kernel with the image the vertical and horizontal directions. The Sobel filter (kernel), which fires on the x-axis (vertically) with the matrix S_x and on the y-axis (horizontally) with the matrix S_y, produces the best performance. The gradient can be calculated on a single channel image (grayscale) or on each channel of the image. S_x and S_y are defined as follows:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.26}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.27}$$

From both gradients intensities on x-axis and on y-axis, the magnitude (G) and angle of the gradient on each pixel is calculated as follows:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.28}$$

$$angle = arctan\left(\frac{G_x}{G_y}\right) \tag{2.29}$$

where:

G_x = Image * S_x $\leftarrow$ (Sobel filter on x-axis)

G_y = Image * S_y $\leftarrow$ (Sobel filter on y-axis)

Image = of dimensions $w \times h \times c$, $c \in \{1,3\}$

***** = Convolution operator as described in Section 2.1.1

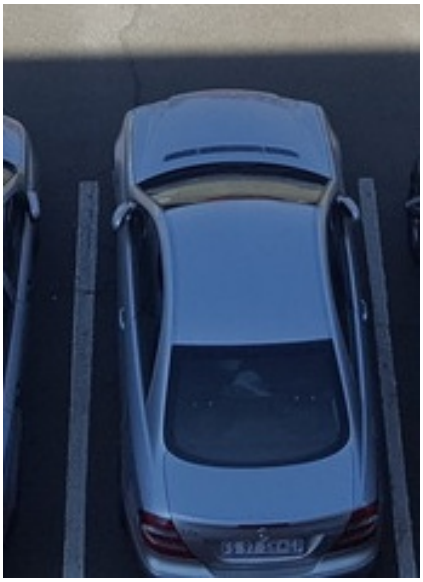
Figure 2.19 presents the gradient computation of the Histogram of Oriented Gradients. Figure 2.19a contains the original image that has edges. The computation of the gradients on x and y-axes produces high responses on edges respectively Figure 2.19b (horizontally) and Figure 2.19c (vertically). Figure 2.19d represents the maximum magnitude generated by the all the filters on the original image, produced by the Equation 2.28.

2.3.1.2 Orientation Binning

The image is divided into cells that are made up of pixels. Weighted by its gradient magnitude each pixel votes for the gradient orientation of the cell. In this way, each cell has a histogram of the orientations built from the pixels within the cell. These orientations are quantized into a small number of bins. The orientations range from 0°-180° for unsigned gradients and from 0°-360° for signed gradients.

2.3.1.3 Block Normalization

The HOG descriptor primarily captures shape information and should be invariant to changes in lighting. To accomplish this, neighboring cells are grouped into blocks which are normalized using the methods below. Normalizing over many blocks enhances the robustness of the HOG algorithm to change of lighting, that has a negative effect on the edge orientations and shape detection. The HOG uses many normalization methods:



(a) Original Image



(b) Vertical Edges (G_x)



(c) Horizontal Edges (G_y)



(d) Gradient Magnitude G

Figure 2.19: Histograms of Oriented Gradients: Gradients Computation

- L2-norm: $\frac{\overrightarrow{v}}{\sqrt{\|\overrightarrow{v}\|_2^2+\epsilon}}$,
- L2-Hys: which is a L2-norm where the values $\overrightarrow{v}$ are limited to 0.2 and renormalized based on [Lowe 2004],
- L1-norm: $\frac{\overrightarrow{v}}{\|\overrightarrow{v}\|_1+\epsilon}$
- L1-sqrt: $\sqrt{\frac{\overrightarrow{v}}{\|\overrightarrow{v}\|_1+\epsilon}}$.

where $\overrightarrow{v}$ be an unnormalized vector, $\|\overrightarrow{v}\|_k$, k is the normalization type which in the set $\{1,2\}$ and ϵ a small value for regularisation and numerical stability.

2.3.2 Convolutional Neural Networks for Feature Extraction

From Section 2.1.5, the Convolutional Neural Networks before the fully connected layers run many preliminary steps to produce features that will be the input to the fully connected layers. The layers up until the fully connected layers, are primarily concerned with extracting features to be used by the fully connected layers. After the last convolution or pool layer, the 2D feature maps can be unrolled into a 1D feature vector. Its associated label (from the original imput image to the network), goes along with that vector.

In general, CNNs learn filters to optimize the classification accuracy. Therefore those filters are expected to capture the critical part of the information. The benefit is that these features are learned from the dataset, rather than being generic feature extractors such as HOG. Other machine learning techniques need feature extraction: Artificial Neural Networks (ANN), SVM. It makes sense to try those techniques with the features extracted from the CNN.

A similar approach employed by Niu and Suen [2012] where they converted the extracted features from CNN and did the classification using the SVM, for recognizing handwritten digits in the MNIST dataset. This research is the first to tackle the automatic parking detection using the hybrid approach proposed in this research. Chapter 6 explores this idea in depth.

2.4 Conclusion

This chapter gave an insight of the techniques required to understand the rest of the document. The CNN is a very powerful algorithm both for feature extraction and classification in a sense that many units (layers) learning a pattern from the image to later be able to classify with accuracy the subsequent images. CNNs are made to classify an image that contains one object in it only. They require a lot of computational power, and for better performance, a GPU hardware acceleration is required. An extension of the CNN in the research is the Region based CNN that, instead of classifying one object per image, it can use the selective search algorithm to get multiple instances (activations) of that object from the same image.

Since the SVM relies on features, the HOG is prevalent in many detection and classification applications. However, there are many of those like the bag-of-words, SIFT and DoG. The most important feature extractor in the case of this research is the HOG which is computed by calculating the gradients in cells and normalizing over blocks. This returns a set a vector per image that can be used to describe the image.

CNNs learn to extract features during their training phase. Traditionally these features are passed into subsequent fully connected layers but can be used as features passed to other machine learning algorithms.

This chapter considered relevant mathematical and machine learning background. The next chapter considers related work especially targeting the parking space detection and classification problem.

Chapter 3

Related Work

This chapter presents and discusses the main techniques from literature used for parking space analysis. Section 3.1 describes some algorithms that are sensor-based and in Section 3.2 presents the vision-based techniques.

3.1 Sensors Based Detection

Hardware sensors are very accurate when set properly. There are two categories of sensors: intrusive sensors systems or pavement embedded systems, and the non-intrusive sensor systems or overhead technologies, [Tschentscher *et al.* 2015].

Intrusive sensors systems are installed from under the ground, and the sensor itself is plugged into a hole on the road by the parking space as shown in Figure 3.1a. They are directly connected to the server via cabling. This type of sensor is usually installed when the parking is being newly built. The installation of that type of sensors requires effort, time, energy, money, and maintenance. Non-intrusive sensor systems are quite easy to install, as they are installed on the surface of the region of interest. Non-intrusive sensors are usually wireless devices that are installed either on top of the parking spot (overhead sensor) or just on the ground as shown in Figure 3.1b.



(a) Intrusive sensors, [eucars 2017]



(b) Non-intrusive sensors, [eucars 2017]

Figure 3.1: Physical sensors around parking spots

Lee *et al.* [2008] implemented an intrusive sensor-based system to detect the cars going in and out of the parking lot. From there, the number of available spots in the parking lot can be deduced from the traffic. The detection is based on how close the vehicle is from the magnetic sensor using a modified version of the min-max algorithm. A distance is returned at the end of the algorithm, and it

gets compared to a distance threshold value to detect whether the car is entering or leaving the parking lot.

Most researchers implement non-intrusive sensor systems as they are cheaper to install. Lee *et al.* [2008] implemented a wireless sensor network to get the result generated by a magnetometer sensor placed at the entrance of the parking lot to count the number of vehicles entering the parking against the number of vehicles that are already parked. Idris *et al.* [2009] also developed a wireless sensor network system, but used ultrasonic sensors placed on the floor of the parking spot, monitored by an interactive system that involves the user. The user selects the desired parking spot on a screen at the entrance of the parking spot, and after confirmation of the availability of the spot, the A* algorithm is then computed to provide the shortest path from the user to the designated parking spot. The ultrasonic sensor is used to validate the occupation of the parking spot by the user.

The use of sensors requires additional apparatus and planning if it must be implemented in a place where their use was not planned initially. However, many universities, shopping malls are already equipped with video cameras, which are mounted at some parking lots. The following section gives details on the automatic parking space detection based on video cameras.

3.2 Video Based Methods

3.2.1 Traditional Image Processing

Video cameras are relatively cheap in installation and maintenance, compared to the sensors above. They are usually already installed for surveillance of parking. Many solutions have been found to automatically detect empty parking space at a place monitored by a video camera. Camera calibration has to be performed to get the intrinsic parameters of the camera [Heikkila and Silven 1997], so that we get approximately the mapping of the 3D coordinates provided by the camera with some noise, and the 2D coordinates processed by the system with less noise (less radial distortion for example).

3.2.1.1 Background Subtraction

The background of an image can reveal much information about the dynamics of the object in it. The statistical analysis of the parking space and the environment that the camera faces can be used to detect moving or stopped cars based on the non-stationary Gaussian distribution of a group of pixels in a background subtracted image [Choeychuen 2013]. The threshold value, calibrated according to the current environment adjusts the system not to get triggered on non-vehicles. The weather and the change of lighting can severely affect the performance of the background subtraction. For example, the motion of the clouds on windscreens or the rain. Automatic thresholding is a technique that is used to adjust the threshold value automatically according to the situation faced by the algorithm.

More recently, del Postigo *et al.* [2015] used background subtraction with a single camera to detect and track vehicles, therefore infer the status of each spot from the parking area. To avoid catching unnecessary motion produced by the background subtraction algorithm, they have applied the Mixture of Gaussians (MoG) to and classify the non-cars objects as background. After defining the background model from the parking area, the moving, static and previously moving vehicles are recorded on a transience map. The technique uses pixel values of the grayscale image to infer the state of the object belonging to that set of pixels as illustrated in Figure 3.2.

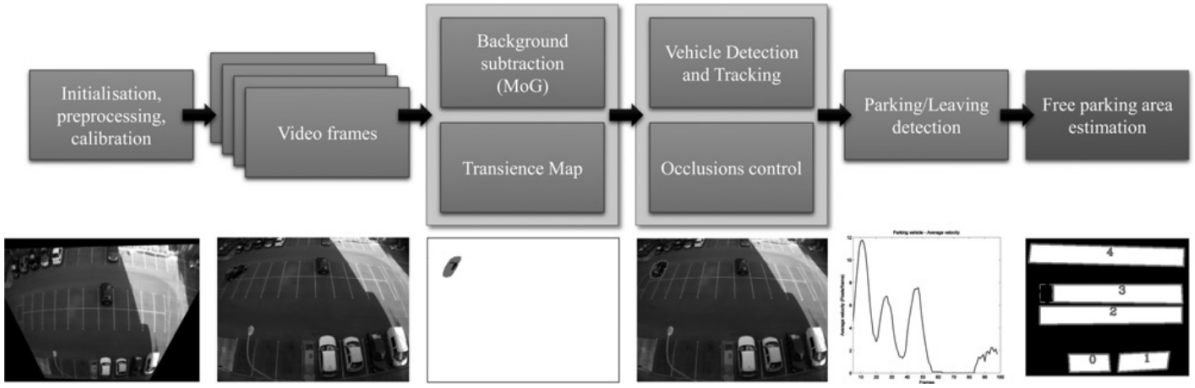


Figure 3.2: Background subtraction method, [del Postigo *et al.* 2015]

3.2.1.2 Support Vector Machines

Cars have some patterns when it comes to edges. Using the edges of cars, Banerjee *et al.* [2011] used the Prewitt edge detection operator to recognize a car in a specific parking spot. They tested their algorithm in a controlled environment (lighting and other natural effects that can alter the image remain constant).Wu *et al.* [2007] proposed a new method for detecting available parking spaces using both multi-class Support Vector Machines (SVMs) and a Markov Random Field (MRF) framework. Figure 3.3 shows the preprocessing of the input image before training. Figure 3.3a is the original image. Each parking row is segmented from the input image, Figure 3.3b, and then by using perspective correction in each row followed by the patch generation, each patch is labeled for training. As illustrated in Figure 3.3d, the SVMs have 8 different classes. After extracting the features from the patches of parking spots generated by the preprocessing, SVMs classifiers are applied to each patch to infer the status of the parking space.

Wu *et al.* [2007] optimized their results using the MRF to correct the conflict generated by one parking spot shared by 2 patches. They have reached an accuracy of 93.52%. Tschentscher *et al.* [2015] also used an SVM, but they have used the Difference-of-Gaussian (DoG) and the color histograms as features for their SVM classifier. After the experiments, they achieved 92.12% accuracy using the color histograms (RGB) and the SVM, and then they achieved 94.13% accuracy using the DoG and the SVM. Later, de Almeida *et al.* [2015] used the Local Binary Pattern (LBP) and the Local Phase Quantization (LPQ) as features, with the SVM classifier. They have reached an accuracy of 99%.

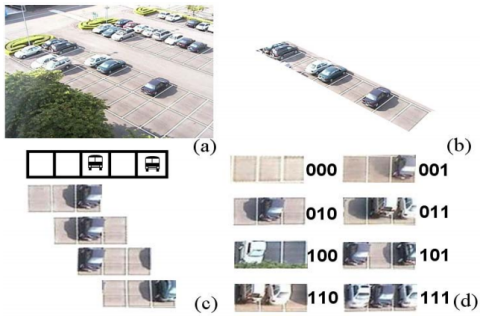


Figure 3.3: Multi-class SVM, [Wu *et al.* 2007]

SVM classifier was mostly used by researchers to tackle the parking space management problem because of its performances and precision in many previous applications. However, all the above techniques were car-driven recognition. Car-driven recognition is mostly based on getting a pattern on the texture or features of the car. Prewitt edge detection operation [Banerjee *et al.* 2011] will focus more on finding the sharp edges that correspond to the ones on a car, and the DoG, color histograms, LBP and the LPQ are all getting the texture of cars to infer its presence [Tschentscher *et al.* 2015; de Almeida *et al.* 2015]. Some researchers noticed that the parking space management problem could also be solved by investigating the space or the area instead of on focusing on the car that is in a parking spot. The background subtraction has been used to get the moving cars in a scenario against the static

cars. Background subtraction output can be very noisy as it can detect moves of non-targeted objects or change of lighting. Choeychuen [2013] proposed an automatic thresholding technique only to activate the groups of pixels that correspond to a car using the adaptive background model method. They aimed to create a map of the parking lot using the histograms of spatial features, their experiments have shown satisfactory results but in an environment where light, temperature and other natural influence were constant. Choeychuen [2013], del Postigo *et al.* [2015] used the background subtraction with the Mixture of Gaussians to get the estimate of the parking lot after creating a transience map analysis to detect stationary, moving and previously moving vehicles in a non-delimited parking lot.

3.2.2 Deep Learning - Convolutional Neural Networks

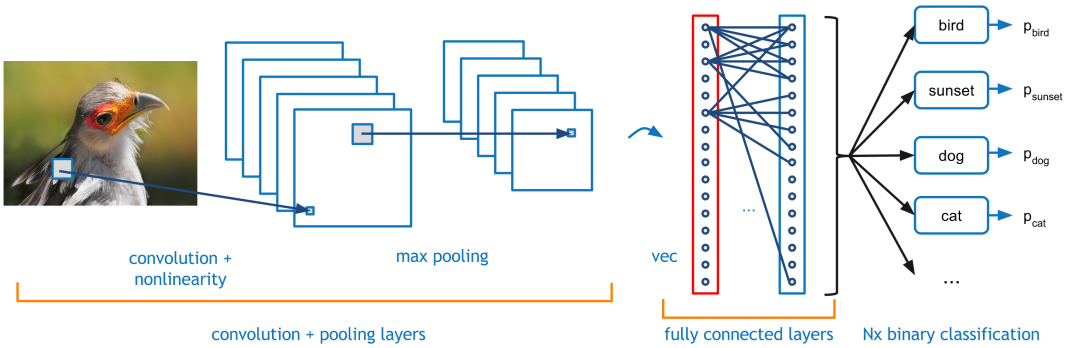


Figure 3.4: Example of Convolutional Neural Networks, [kdnuggets 2016]

Traditional image processing techniques were proved to be successful in experimental and in real life situation. The problem with the traditional image processing techniques, however, is usually the feature extraction. It can be difficult to engineer features for a car model since there are many types of cars and it will be hard to have a generalized model. From this point of view, complex features can be learned through the network [Amato *et al.* 2016]. CNN architectures like AlexNet [Krizhevsky *et al.* 2012] or LeNet [LeCun *et al.* 1998] have successfully classified many trained objects (AlexNet), and handwritten digits (LeNet5). The only downfall of CNN is that CNNs are computationally expensive during training, but fast in inference. For the CNN to perform at its best, a large dataset containing all possible situations of what is to be classified is required. de Almeida *et al.* [2015] provided during their research a larger dataset for the parking space management researchers. Those networks are now used for classification purposes, and recently many authors have used CNN to classify parking spots as presented in Section 2.1.9.

Valipour *et al.* [2016] implemented their solution around the CNN VGGNet-F [Chatfield *et al.* 2014] on which they have modified the last hidden layer to accommodate the two classes involved in the classification: occupied or empty stall. To inform users about the status of the parking lot, a mobile application will receive feeds from the server that performs the classification and returns the result to the user. They have reached an accuracy of 99.99%. Amato *et al.* [2017] used a modified version of the AlexNet architecture [Krizhevsky *et al.* 2012]. To make it run on a Raspberry Pi (R.Pi), they drastically reduced the number of convolutional layers to allow all the computations to happen in the R.Pi's, which is very restricted in terms of resources, ram capacity, and disk memory. There was a slight loss of accuracy between the networks they modified and their own. Their modified CNN was already trained (weights are already set), the R.Pi's performs the classification which is a single forward pass through the network. Their approach is not different from Valipour *et al.* [2016], but instead of creating a server that will process the frames from the parking lot and dispatch it back to the clients, they decided to create their smart cameras that are each made of a Raspberry Pi and a Pi camera. The Raspberry Pi will perform all the computations on-board and will return the results to the

server. They are avoiding having a single point of failure. The downfall is the implementation of their custom camera that makes the implementation more expensive.

3.2.3 Conclusion

This chapter gave an insight of the work done by the previous researchers on the parking space management. The sensor-based approach is more intuitive since it involves hardware that can without delay send and return signals coming to it. There are two types of sensors: intrusive (on-ground) and non-intrusive sensors (overhead). Both of those types cost money in for maintenance and installation.

The video-based approach relies on some cameras that monitor the parking lot. The main input here is the set of pixels of each frame captured by the camera. The background subtraction, as used by del Postigo *et al.* [2015]; Choeychuen [2013] in general computes the change in the pixel value that is between 0 and 1 to infer the vacancy of a parking spot.

Machine learning became a very active field of research. Using SVM associated with some feature extractors, like color [Wu *et al.* 2007], the spots could be classified into groups of 3 spots.

Also in machine learning, the CNN revolutionized the field of object recognition in particular and computer vision in general. First implemented by LeCun *et al.* [1998] with multiple versions of LeNet, that algorithm can use the mathematical convolution, and more other techniques to get the most robust features from a set of images. Many researchers have been and are still using the CNNs with relatively good accuracy.

This research, based on the study of the literature, brings a contribution to the parking space management, by using the RCNN, and both the CNNs and SVMs in the same system to respectively detect RCNN and classify parking spaces CNNs and SVMs.

That contribution was achieved through many steps. The initial step consisted of locating the experimental parking area required to collect the data. Chapter 4 shows how data was collected and preprocessed to be used by the step sequence steps for features extraction and classification.

Chapter 5 discusses how Region based Convolutional Neural Networks or RCNN are used in the project to detect individual parking spots from an image.

Chapter 6 brings a comparison between known classifications techniques, which are the Support Vector Machines and the Convolutional Neural Networks, with an emphasis on the features captured by the Histograms of Oriented Gradients and the CNN as features extractors.

Chapter 4

Dataset Design

4.1 Collection

This work is also based on a physical data collection. Physical data collection means that all the images are produced from the project. For better learning during training, the GoPro Hero 4 and a Samsung Galaxy S6 cameras produce the best results with its minimum specifications which are 5 megapixels for the GoPro and 12 megapixels. The GoPro can stand still for a long period at a single position, and the Samsung Camera was just used to increase the source of inputs. Otherwise, the shooting process is identical. For testing, the Logitech C525 - HD Webcam was used.

Prior placing the camera on a building or a window, the study of the area showed the need for understanding the dynamics of the parking lot. We visually observed a pattern on how the spots are filled in terms of time of the day, from Monday to Friday (This applies for the parking space used for the research which is a university area):

Period of the day	Time	State of the parking
Morning 1	5:00 - 7:00	Filling
Morning 2	7:01 - 12:40	Fully packed
Afternoon 1	12:41 - 14:00	Getting empty
Afternoon 2	14:01 - 16:00	Fully packed
Afternoon 3	16:00 - ...	Getting empty

Table 4.1: Dynamics of the Chamber of Mines parking (Wits)

From the Table 4.1 above, there are some parameters to take into consideration to collect data for the project. If the collection happens in the red area from the same table, not much can be gained from the environment other than a fully packed parking. That situation will contribute to the data since it contains a full parking, but those states last for 2 hours or more as shown from the column 'Time' in Table 4.1 for the red time slot.

Figure 4.1a and 4.1b show respectively an instance of blue and red times. The collection for this project was taking place usually in the blue area, in the red areas and the green areas for around 30 to 45 minutes, every day, taking into account the lighting, all from Table 4.1. The collection process ran for 3 months, including weekends, to get empty spots as, during the week, the spaces will mostly be busy as seen in Figure 4.1c. The GoPro was set to capture 1 images per minute which means that every day, around 90 to 135 images were shot, given a combination of rounds, for instance, Morning1 + Morning2 + Afternoon3.



(a) Blue area (b) Red area (c) Collection on a Saturday

Figure 4.1: Different periods of the day for data collection.

For testing data collection, the Logitech C525 - HD Webcam was placed precisely where the GoPro was during training collection. The differences between the two events reside in the camera specifications, and time (date) of the collection. The Logitech webcam is a low-quality cheap camera compared to the GoPro which has a higher quality. The testing data were collected 8 months after the training data. Section B shows the data collected from both the GoPro and the Logitech Webcam.

Table 4.2 gives a summary of the data collection for the detection unit. The collection for this unit takes a whole image as input similar to Figure 4.1b. Per image, some bays are labelled as occupied or vacant. We have to note that the labelled bays are not done through time. Each bay in every image was manually and individually labelled. This process could not be automated. There is a large difference in proportion between the classes in each set. The aim is to have a detection unit that has more accuracy in detecting an occupied parking space because it is acceptable for a user to drive pass a vacant bay, but not to directed to an occupied bay. That increases the chance of occupancy decision. Less vacant parking spaces with noise in them, build up a more realistic cluster of that class.

	Classes			
Sets	**Bays Labelled as Occupied**	**Bays Labelled as Vacant**	**Number of Whole Images**	**Number of Labelled Bays**
Train	30450	16194	770	46644
Test	640	2495	49	3135
Total	31090	18689	819	49779

Table 4.2: Data Collection Summary Training/Validation/Testing: Detection

Table 4.3 gives a summary of the data collection for the classification unit. The Train and Validation data were both taken from the GoPro and the Samsung Galaxy S6 camera. This to ensure that as many details are extracted from the images to enhance the strength of each collected feature. The images were generated by cropping bays from the collected images. More details about the process in Section 4.2.3.

	Classes			
Sets	**Occupied**	**Vacant**	**Total**	**Camera**
Train	6000	6000	12000	GoPro & S Galaxy S6
Validation	1500	1500	3000	GoPro & S Galaxy S6
Test	2500	2500	5000	Logitech Webcam

Table 4.3: Data Collection Summary Training/Validation/Testing: Classification

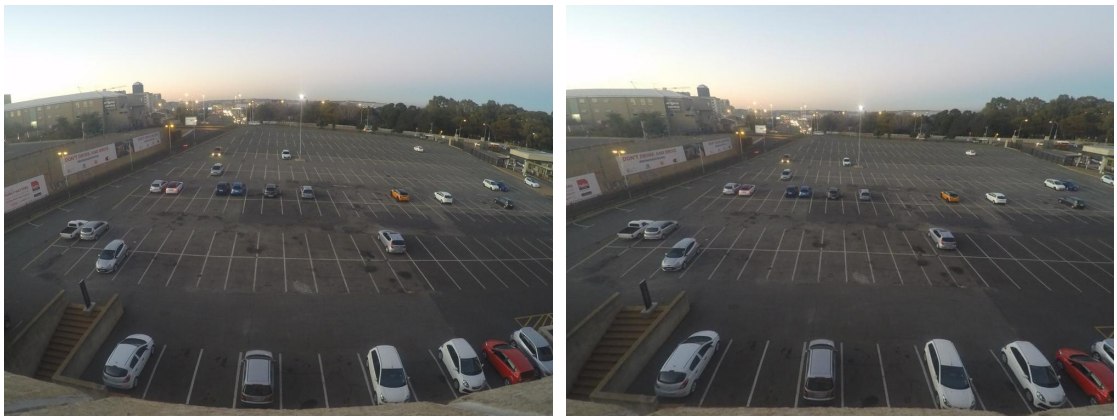
The camera shoots from the top of a building (Chamber of Mines, University of the Witwatersrand), at around 20-30 meters from ground level. The lines of the parking spots are parallel to the

central axis of the camera.

4.2 Data Preprocessing and Labelling

This section exposes the methods employed to prepossess the images for the classification.

Before any preprocessing on the images coming from either the GoPro or the Samsung camera, the image should be corrected from distortion (GoPro) and size (GoPro and Samsung). All the raw images are organized in such a way that they are grouped per date in separate folders. In other words, one super-folder holds all the sub-folders containing the images.



(a) Image with Barrel radial distortion (b) Image with radial distortion corrected

Figure 4.2: Correction of radial distortion

The wide angle setting of the GoPro camera produces a distortion called barrel distortion or fisheye as illustrated in Figure 4.2a. ImageMagick which is a Linux package corrects the distortion given the intrinsic values of the camera 4.2b. For both camera and using the same Linux package, the size of all the images was reduced to 1000 x 1000 pixels for consistency. That process is hardware hungry, the Hydra cluster at the University of the Witwatersrand was used to run both the distortion removal and the resizing of all the images.

4.2.1 Preprocessing for Region-based Convolutional Neural Networks (RCNN)

This step aims to have a set with a zero mean and unit variance. That keeps the information of the images valid and reduces the size of all the images for faster training. Given an image like Figure 4.1b, performing a normalization technique, e.g., mean normalization, will destroy the information required for the detection, given all the channels, as many colour distributions are present on the full image.

$$x_i \leftarrow \frac{x_i - \mu_{image}}{\sigma_{image}} \tag{4.1}$$

where x_i, μ and σ are respectively the pixel value of the image, the mean and the standard deviation produced by all the pixel values of the image.

To support the previous statement, Equation 4.1 shows that to each pixel of the image, if only the mean is subtracted (ignoring the division by the standard deviation), regions with pixel values less than the mean turn gray as illustrated in Figure 4.3b, where the equation applies to Figure 4.3a, which is a 3-channel image. Those gray regions alone could be significant as they mostly represent occupied spots.

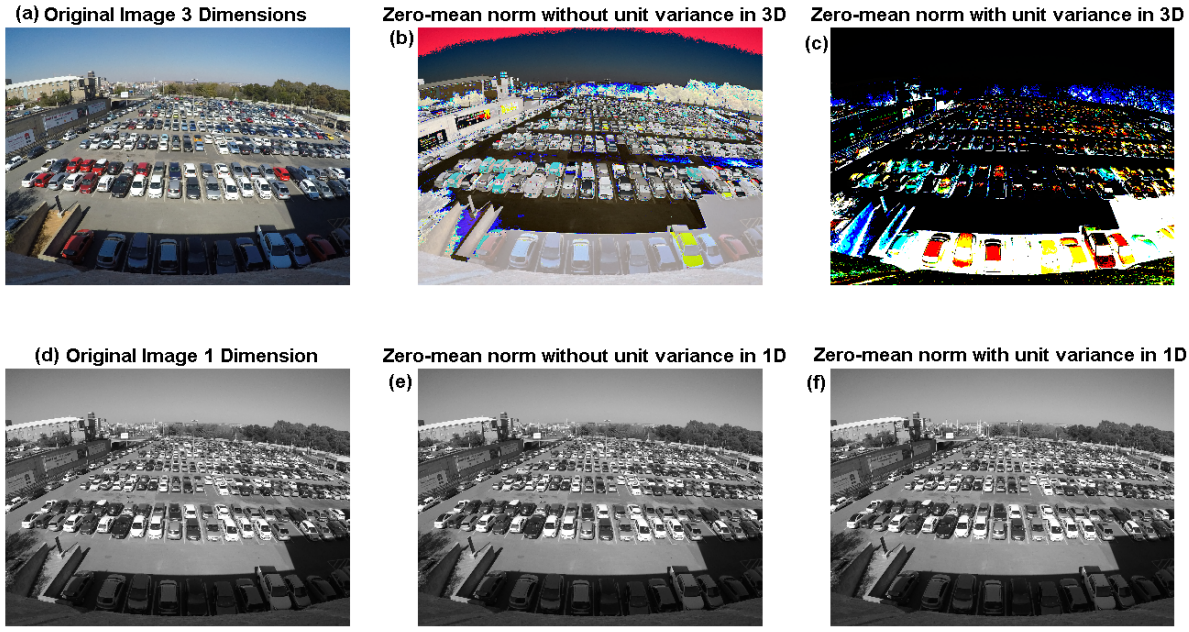


Figure 4.3: Mean Normalization

Unfortunately, the shadow at the bottom of Figure 4.3b will incorrectly lead the model into believing that it is a single region.

Applying the full Equation 4.1 to the image on Figure 4.3a produces Figure 4.3c which exhibits the same properties as Figure 4.3b where the activations appear on the regions of interest and also on other places, which is not ideal for the image segmentation.

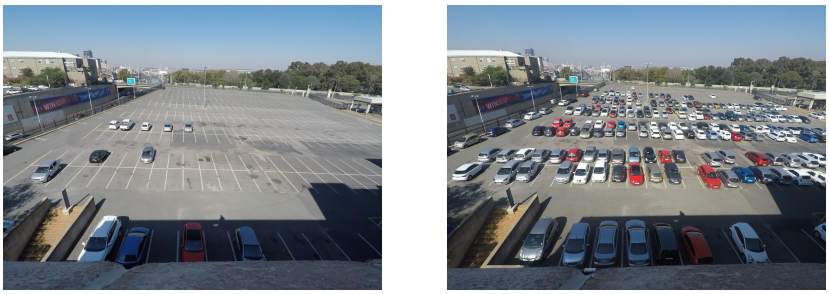
In the same logic, for a grayscale image (1-channel image) like Figure 4.3d, applying Equation 4.1 with and without dividing by the standard deviation returns the same image as Figure 4.3d as illustrated in Figure 4.3e and Figure 4.3f. There is a difference in the pixel values as they are centered at 0 and have a standard deviation which is 1.

For the detection phase, Figure 4.3a is therefore selected for the RCNN process since a 3-channel for this research offers a better as colour intensity on all channels without distorting too much the image for the selective.

After getting the images from the camera, they should be transformed into data usable by the RCNN network. As opposed to the other types of Convolutional Neural Networks, the RCNN takes a whole picture as input containing one or more classes during training as explained in Section 2.1.9.

4.2.2 Data Labeling for RCNN

After the data collection and preprocessing, the labeling takes place since we are building a supervised learning type of system. The data labeling requires manual selection of the region of interest by drawing a rectangle around the spot, then using an XML file per image to record all the possible classes, with the XML generated using LabelMe [TzuTaLin 2016], which follows the ImageNet labeling standards [Deng *et al.* 2009]. That format is used as input to the RCNN. Each file is the ground-truth of all spots appearing for its corresponding image. Figure 4.4 displays 2 instances of the same parking lot on different days: Figure 4.4a is a day in the weekend and Figure 4.4b is a normal business day. Both figures explicitly contain the classes required for the system. From those images, it is easier to get as much as information from both classes as possible.



(a) Almost empty parking lot (b) Full parking lot

Figure 4.4: Sample of dataset for RCNN

The data labeling for the RCNN is organized into 3 folders as illustrated in Figure 4.5:

- **Images (Dark Blue):** Contains all the collected images from the parking lot. Those are raw and preprocessed in terms of distortion and size of the image. Each of them is associated with the index or name, which is also a part of the labeling,

- **Annotations (Red):** Contains the corresponding xml files of all the image files in the folder **Images**. Those xml files contain the coordinates of each box corresponding to a parking spot in the file having the same name as that xml file.

- **ImageSets (Green):** Contains 2 text files, train.txt and test.txt. Those files contain the image file names from the folder **Images**.

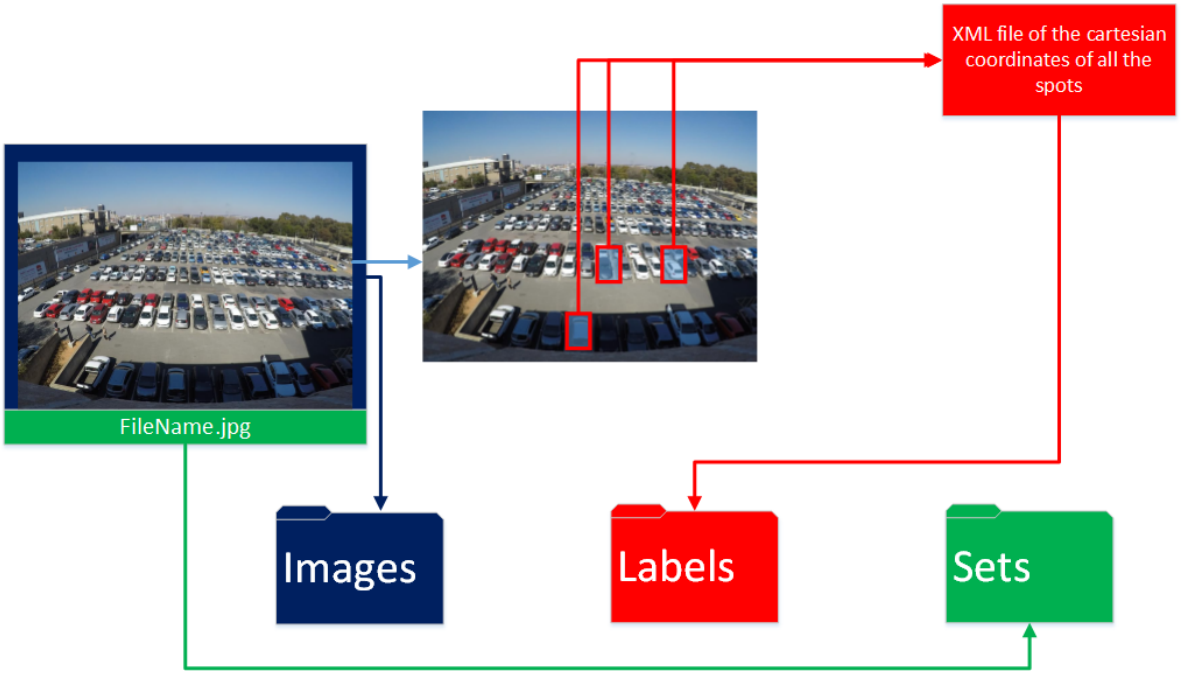


Figure 4.5: Data Labeling for RCNN

4.2.3 Preprocessing for Support Vector Machines (SVM) & Convolutional Neural Networks (CNN)

Both SVM and CNN require as in input an image containing one object that corresponds to a class. In the case of those two, image processing techniques are very useful as it helps to generate crucial information about the image of the object to classify.

Before processing those images, they need to be individually extracted from the main images of the parking lot. Based on the previous sections, around 2000 images of parking spots have been collected and organized in a folder named after their day of collection, which implies many individual spots to withdraw from those images. To get individual spots from all those images quickly, the images, their corresponding XML files produced as the result of the data labelling developed in Section 4.2.2, and a script generate all the spots identified from the XML files, as illustrated in the Algorithm 2. That algorithm explains the image cropping process, originally written in Python. Each XML file contains a critical section of the image which looks like the following lines:

```
<bndbox>
<xmin>55</xmin>
<ymin>510</ymin>
<xmax>246</xmax>
<ymax>747</ymax>
</bndbox>
```

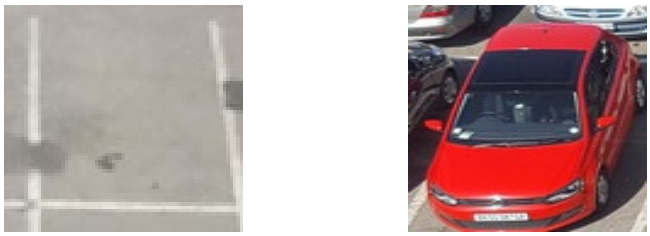
The Parking Spot Cropping Algorithm searches in each xml file for those 4 values between $<$ $xmin$ $>$ and $<$ $/xmax$ $>$, similarly for y, where those correspond to the Cartesian coordinates of a manually located spot in an image. Each XML file is processed first before its corresponding image. The layout of all the file puts the XML first and the image after. The order is automatic given that the XML file and the image have the same name just with the difference in the file extension that set them apart. Figure 4.6 shows an instance of what the cropping algorithm from every image.

After that, the individual spots are separated into 2 folders: Occupied and Vacant. They each contain around 10000+ crops of spots, which are used to generate the Train, and Validation folders with the ratio of 60% and 40%, respectively. While the images are of the same parking spaces, the images in those folders are independent of each other in that the lighting, time of day, and quality of the camera differ substantially. In every set, images from all the time slot from Table 4.1 are present. As a reminder, all the images used for training, come only from the GoPro and Samsung Galaxy cameras. All the deploy set images were collected months after the collection of the training data, and they come from the Logitech Webcam, which has much lower quality compared to the previous cameras. The mean normalization suggested earlier with Equation 4.1 applies but, the mean of the Train set is evaluated and returned first, and then subtracted from the Validation, Test and Train set. This step is essential as it avoids overfitting during training.

To classify bays using the SVM or the CNN, normalizing is a very important step to take. In this research, there was an increase in performance when normalizing the images using the Algorithm 1, on all the channels. The aim is to get images having a unit variance and zero mean. Technically lines 7-9, and line 14 are computed using the OpenCV libraries (CV2.MEANSTDDEV) for the computation of the mean and the standard deviation over Numpy (numpy.std & numpy.mean). The results are not different, but the speed of execution for the number of images applied to the algorithm is better using OpenCV. Figure 4.7 supports the previous statement by showing how for each image, identified by their ID (x-axis), the OpenCV libraries outperform the Numpy libraries. We have to note that OpenCV computes at once the mean and standard deviation as compared to Numpy where they are 2 different functions.

In terms of processing within each channel, since a snippet contains either an occupied or vacant

spot, normalizing in all channels made a little difference in performance compared to the grayscale image. In other words, RGB images are better than grayscale images for classification.



(a) Vacant Spot (b) Occupied Spot

Figure 4.6: Instances dataset for classification for SVM and CNN

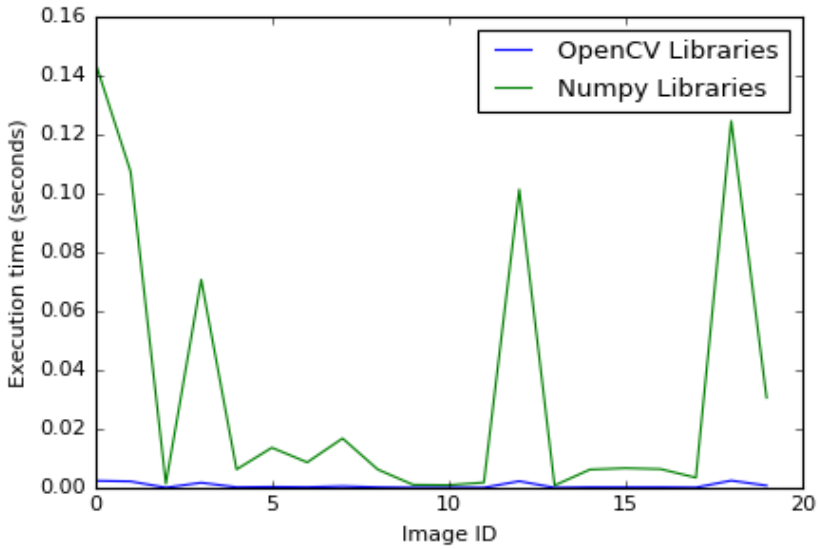


Figure 4.7: Execution time of processing the mean and the standard deviation of an image

4.3 Conclusion

This chapter described and data collection and preprocessing of the images of parking lots, both for the detection and classification. Data preprocessing, especially mean normalization is very important in image processing in general. However, in Section 4.2.1, mean normalization is not required since there were factors after the normalization that would make the learning useless. For instance, Figure 4.3c shows how the activation is very random on throughout the image which is not very ideal when feeding the classifier. After taking the images as they are (no normalization), they are labelled and sorted for the detection training.

Section 4.2.3 discussed the data preprocessing for the CNN and SVM. They have similar properties in terms of input data for the algorithm. Only the crops of the parking bays were used, since both SVM and CNN receive images containing one object each, at a time during training. Each crop is mean normalized, then labeled. One finding here is about the library to use during the mean normalization. From Equation 4.1, the standard deviation can be omitted as it does not impact on the response generated by the image after subtracting the mean from it. Figure 4.7 shows that OpenCV libraries are faster at processing the mean and standard deviation compared to the Numpy libraries. Although one could implement it from scratch, performance will be an issue because of lack of proper optimization.

This data is now ready for training, validation, and testing by the various algorithms involved in this research project.

Chapter 5

Automated Bay Detection

5.1 Region based Convolutional Neural Networks

This section is one of the main of this dissertation. The system can recognize parking spots using machine learning, Support Vector Machines, and Convolutional Neural Networks. According to the literature, RCNNs have not been applied to the parking space management problem for bay detection. While Chapter 3 identified the work using CNNs and SVM manually on calibrated parking lots, this appears to be the first attempt at an automatic definition of bays using RCNNs.

For the detection phase, the RCNN shows good performances in scanning and locating parking spaces. The following lines explain the process under the automatic detection of the spot.

5.2 Architecture

Section 2.1.9 gives an overview of a typical RCNN. The detection applied in this section uses the ZF Net architecture [Zeiler and Fergus 2013]. There are plenty of other RCNN architectures, but ZF Net is selected mainly for its size. It is not too deep for the task assigned to it, since only 2 classes are to be recognized, and the images are also relatively small. Success implies that the number of features can be extracted given a small network. The other choices are the VGG-{16,19} [Simonyan and Zisserman 2014] which are massive networks to work with, given only 2 classes of small images.

Another constraint is the amount of item that the GPU can handle at a time during a single detection. All of those elements contribute to the choice of the ZF Net as architecture for the detection segment of the system.

Figure 5.1 illustrates the ZF Net architecture as designed by its authors Zeiler and Fergus [2013].

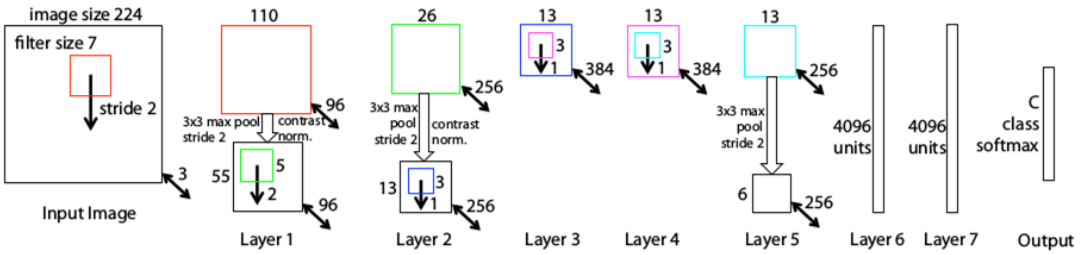


Figure 5.1: ZF Net [Zeiler and Fergus 2013]

The network as a whole remains with its 8 layers, but the very last layer is equipped with a softmax classification unit with 2 gates (2 classes) which are: occupied or vacant.

5.3 Training & Validation

Figure 4.5 exhibits the top view of the data labeling for the detection phase of the project. In reality, an input to the RCNN architecture is an image and its corresponding xml file. For this phase, the dataset results from both labeling the images and keeping their names organized into Training and Validation set files. This work extends the multi-object detection of Ren *et al.* [2015] using 2 classes only. The python implementation of their work, py-faster-rcnn package, contributed a lot to training the detection portion of this research project. Before using it, one needs to correctly configure Caffe [Jia *et al.* 2014], which is the engine processing the heavy loads during training.

Training an RCNN requires some computational power. The most important part of the hardware for this work is the GPU which is a Nvidia GeForce GTX 1080 with 8 GB of graphic memory with cuDNN, 16 GB of RAM and all running with an Intel i7 processor. Although the configuration of the workstation is of standard class, training still needs to be done with caution, not to overload the graphic memory.

The training itself requires multiple modifications of the hyperparameters of the core of py-faster-rcnn to suit the situation. After they are done, the training & validation phase can start. In an attempt of training an RCNN from scratch, many issues rose: convergence of the cost function values was not happening since it was very unstable at a high rate, even when using premature stopping at an acceptable error, the accuracy was terrible. That is why the weights of a pre-trained network, ZF Net, worked better than a fresh one since that network was pre-trained on more than 100000+ images which means that the activation neurons of the network are more prominent to fire the matching pattern of the project's objects.

In terms of RCNN training configuration:

RCNN Training Configuration	
Batch Size	5
Momentum	0.9
Learning Rate Policy	Step
Base Learning Rate	0.001
Weight Decay	0.0005
Network	ZF Net
Number of iterations	200000
Solver Type	Stochastic Gradient Descent

Table 5.1: RCNN Detection Training Configurations

A short explanation of the parameters shown in the above table:

- Batch size: number of training items that will propagate through the network at one iteration. It constitutes a fraction of the dataset.
- Momentum: Coefficient between 0 and 1 that reduces the massive variation of the gradient descent during training.
- Learning Rate Policy: Rule applies to the learning rate to modify its value to increase better learning.
- Base Learning Rate: Determines the initial learning rate chosen before learning.

- Weight Decay: Number between 0 and 1 that controls the growth of the weights but penalizing them when they become too large.
- Network: Convolutional Neural Network used for training.
- Number of iterations: Number of times the training dataset does a forward and backward pass.
- Solver type: Type of gradient descent used to calculate the backpropagation.

Given that configuration, the training for 200000 iterations took precisely 17 hours and 42 minutes, using the dedicated deep learning node of the School of Computer Science and Applied Mathematics at the University of the Witwatersrand. Usually, the training process of a standard CNN will return automatically the accuracy produced by the validation set at every epoch. For the RCNN, a number of hyperparameters, Figure 5.2, actually determine the strength of the classifier being trained. From the same figure, we can put more emphasis on the $train_loss$ which is more informative about the accuracy of the classifier. The other values are not very important for the analysis required in this research as they are losses linked to the processing done by the selective search and intermediate classification of the RCNN, within the network.

At the end of the training process, the loss ($train_loss$) converges just around 0.3 which is not very bad (usually around the order of 0.05 from the literature) for a relatively small dataset.

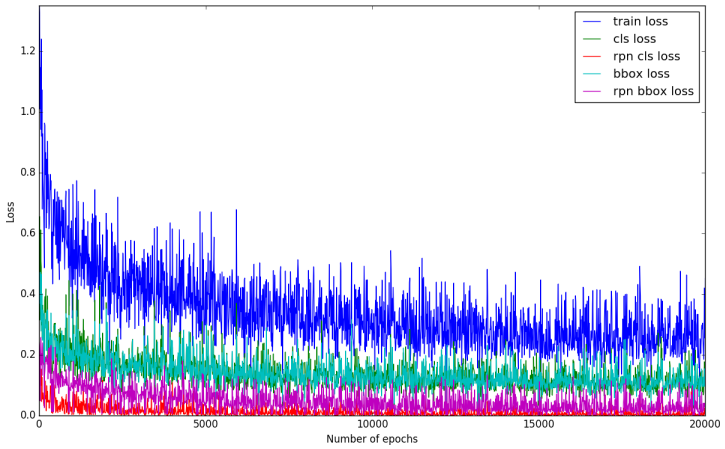


Figure 5.2: Training loss of the RCNN Detection Unit

5.4 Testing

This phase is also done as "indoors" as in the training & validation phase. Another set, which is different from the 2 previous ones, is tested against the newly built RCNN model to evaluate its performance on images that have not been used for the previous phase.

Table 5.2 presents the mAP (mean Average Precision) of the network. As it displays, the mAP for the Vacant class is much lower than the Occupied class. Table 4.2 shows that the classes are not balanced in favor of the Occupied class. From the data collection configuration, this result was predictable, but more factors increase mAP of the Occupied class. The RCNN is mainly based on image segmentation, which generates subsets of the image-graph. Subsets are mostly occupied and vacant parking spaces, with their ground-truth for training. During testing, the shadow on the ground generates a subset similar to an occupied space. Therefore, many parking spaces are misclassified as occupied.

Compared to the results Ren *et al.* [2015] generated, which are between 60% and 80%, ours are quite high, especially for the Occupied class. The vacant class falls within the range produced by Ren *et al.* [2015]. This might be because of the number of classes involved in the research project which is 2, against 1000, and the number of training sample with is around 2000 against beyond 100000+ images for Ren *et al.* [2015].

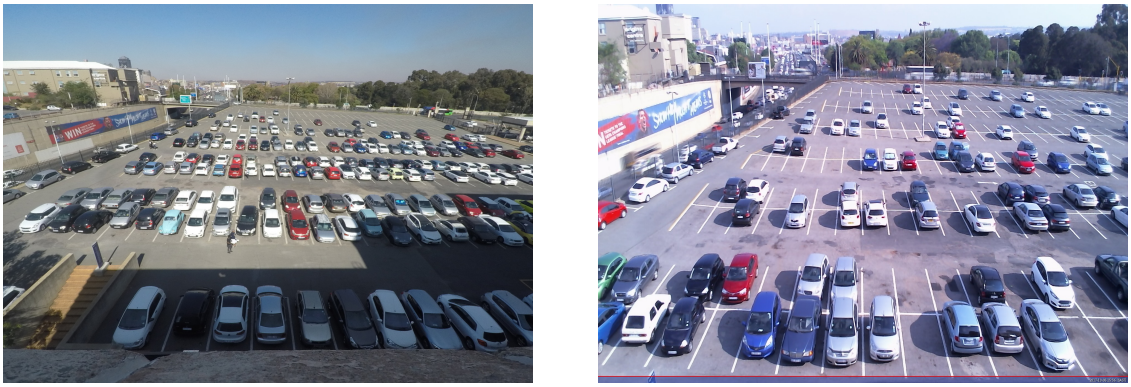
Vacant	Occupied
79.90%	90.65%

Table 5.2: Mean Average Precision (mAP) RCNN Detection Unit

5.5 Analysis

Section 5.4 shows that in the testing set, the occupied spots are more accurately detected than the vacant ones. That is in theory, and giving that the testing images are just the ones coming from the same source (GoPro Camera) but are appearing neither in the training nor the validation set. Parking spaces are less likely to be equipped with GoPro cameras around from monitoring. During this work, a simulation of the normal parking camera shows that this detection model can still perform as if the image was coming from a GoPro camera. Figure 5.3 represents the 2 types of image qualities involved in the research. Figure 5.3a is a sample of the training set and Figure 5.3b is a video frame coming from the deployment camera, which is a Logitech webcam c525. There is a difference concerning image quality, that might heavily compromise the detected, in theory. The system looks for a gray ground and some lines to imply the presence of parking spots. However, in practice, some spots are missed mainly due to many reasons like:

- The absence of sunlight on the spots,
- Occlusion,
- The shadow of a vehicle on a spot.



(a) Image coming from GoPro (12Mp)

(b) Image coming from Logitech Webcam (8Mp)

Figure 5.3: Difference of image quality between Train/Validation and deployement images

The detection module runs every 10 minutes to allow the non-detected spots to be considered. Although, due to the multiples factors stated earlier, some detected spots might disappear on the next round of detection. It implies that the training data might not have enough data regarding occlusions, shadow-on-spot, or the data labeling did not enforce on that particular detail. Figure 5.4 is a general explanation of the previous statement. That figure shows the detection under different periods of the

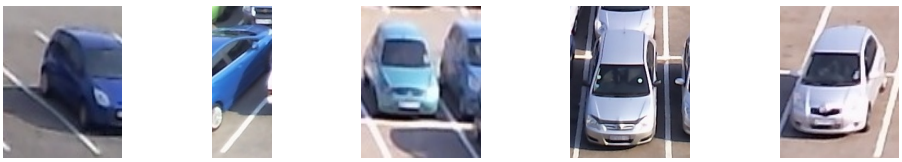


(a) Overcast (b) Sunny (c) Overcast (d) Cloudy

Figure 5.4: Detection of the System on a parking space

day. The best detection rate occurs when the sky is not cloudy, Figure 5.4a,b,d. Those three instances detect the most spots of the image, that is ideal for the full coverage of the parking when the second phase will take place to classify every spot. During cloudy weather, the detection range is narrowed, reduced to the spots closer to the camera, Figure 5.4d. Again, multiple detections help in increasing the system's range of action. To better understand what is individually detected, let's zoom in the region of interest cropped from the images captured by the camera, like the previous figure for instance. The detection unit is not perfect, many irregularities and ambiguities appear on some detected spots. Figure 5.5 explicitly shows some results from the detection unit where :

- (a) is the representation of two detected spots, one occupied and one vacant, they are aligned horizontally. This type of detection will turn out to be very ambiguous for the classifier since 2 classes are present in the same image,
- (b) is the part of a parked vehicle. That is still a valid spot since the line is present, and as soon as the vehicle vacates the spot, only one bay will show to the classifier,
- (c) is the same situation as (a), but this time the two spots are aligned vertically. Since there were no cars parked on the front bay, the detector grabbed both spots,
- (d) is the correct capture of a parking spot to be sent to the classifier. This snippet will be fully valid whether the spot is vacant or not,
- (e) is almost similar to (d) but, in the absence of the car, this will again the situation explained in (c).



(a) (b) (c) (d) (e)

Figure 5.5: Individual detected spots

5.6 Conclusion

The result of the process described above produces an RCNN caffemodel which is the detection unit. That model equipped with its network configuration detects with some error the spots in a given parking space. The phase described in this chapter allows the system not to rely on human intervention for detecting individual parking spaces on its own. This unit alone can do the detection/classification. The major problem with this unit doing both works is that image size and the number of elements to detect. The analysis of the result as discussed in Section 5.5 shows that this unit is not perfect. It detects more occupied spots than vacant ones, as expected from Table 5.2. The position of the camera as well

impacts on the result produced since the higher the camera, the better the spots will be. In this case, the camera is quite low in height, therefore, some detected spots will mostly contain additional ones. The multi-object detection scans the whole image, therefore doing it on many images as in a video will be very cumbersome. To reduce the amount of work required to classify each spot, it can be classified individually and then report to the whole image. To do so, classification unit needs to be a separate component from the detection one.

The detection unit can automatically detect bays, especially when they are initially occupied, with 90.65% accuracy. Figure 5.4 illustrates the dependence of this unit on external factor like the weather, lighting changes, which leads to missed detected spots between detections. To overcome that, previous bays initially detected are saved in a data structure or a file. At the next detection, the newly detected bays can be appended to the set of already detected ones. If a bay is already recorded (coordinates), then that new entry of that detected bay is discarded since it is already present.

Chapter 6

Bay Classification

Once the detection unit operates and computes the location of the spots, they need to be classified. The CNNs and SVM are trending because of its classification tasks according to the literature. In this research, the second part of the system consists of classifying a detected spot as described above and classifying it as vacant or occupied. This section presents the different classification processes attempted to build an automated parking space detection system. The following lines discuss the architecture or design, the training-testing in the validation procedures and the deployment of respectively the CNN, SVM and CNN $\rightarrow$ SVM.

6.1 Convolutional Neural Networks

Let us start the Convolutional Neural Networks as a classifier for the detected spots. This part of the classification uses and extends Section 2.1. Each CNN of the literature is trained to perform many classification tasks with more than 500 classes for some of them. For example, AlexNet was trained on 1000 classes [Krizhevsky *et al.* 2012]. That requires the CNN to be sophisticated enough to learn most of the features present in the input image for better learning.

The following lines present the different steps to produce a CNN classifier for the system. The VGG-16 was the best network found for the system.

6.1.1 Architecture

The VGG-16 is a big network that was trained with 1000 different classes. This research focuses on 2 classes only, so the last layer (classification) of 1000 unit is replaced with a classification layer of 2 units. The input layer of the VGG-16 is an image of shape 224x224x3, which is a color image. Again in this work, the images coming from the detection unit are quite small, 80x80x3, but they are resized to 100x100x3 for the network. The network has not been trained from scratch. Table 6.1 illustrates the custom VGG-16 built for the classification phase of the system. The weights of all the network up to the first fully connected layer were kept intact. Because, the VGG-16 is already trained to recognize edges, corners, and some complex shapes, there is no need to train that section. The green layer represents the first fully connected layer of the original VGG-16. It will not be retrained. The red layers (fully connected) represent the added custom fully connected layers to have some more domain-specific features for this work. They will be trained from scratch.

Input (**100** x **100** x 3)
Convolution 3D - 64
Convolution 3D - 64
MaxPlooling
Convolution 3D - 64
Convolution 3D - 64
MaxPlooling
Convolution 3D - 128
Convolution 3D - 128
Convolution 3D - 128
MaxPlooling
Convolution 3D - 512
Convolution 3D - 512
Convolution 3D - 512
MaxPlooling
Fully Connected - 4096
Fully Connected - 2018
Fully Connected - 1024
Fully Connected - 2
Softmax

Table 6.1: Custom VGG-16: The bold parts correspond to the changes made to the original network.

6.1.2 Training & Validation

Compared to the RCNN where an input image contains multiple instances of many objects to classify, the training and validation of the classification unit consist of a set of images of the same category grouped in a folder, which constitutes a class. All operations are run on the same configuration as in the RCNN in Section 5.3.

Let us start with the data. From Section 4.2.3, which deals with data labeling and preprocessing, the primary images of the parking lot generated more than 10000 snippets of parking bays, all classes involved. Table 6.2 describes the number of snippets per set for training and shows that the classes are balanced. Each set is built independently from the others to avoid any leak to go through. This implies that no snippet from any set will look like one in the other set.

One of the most critical steps in this research is the data selection. In Machine Learning in general, and in computer vision in particular, quality of the data affects the accuracy of the classifier. The data contains many features that are useful for some or just not necessary. During our investigation, images are just heavy in terms of their size, for the learning. Visually they represent a class, but after processing, they mean nothing to the learning algorithm. To answer those questions, the mean normalization is an approach to the answer. In this case images of parking spots are either a filled bay, or an empty one, both with white lines around them. The image means on all channels from the training set returns an average amount of each pixel of an input image to fit into either class. In other words, a consistent activation should fire from one class all the other for later learning and prediction. Once the data is ready, it can be fed into the pre-trained network for learning. Table 6.3 shows the configuration of the custom VGG-16 network used for the training of the classification unit.

We experimented many configurations for the training phase using a “Grid-Search-like” method by looping in all the solver types stated in Table 6.3 for the best results in terms of accuracy.

	Classes		
Set	**Occupied**	**Vacant**	**Total**
Train	6000	6000	12000
Validation	1500	1500	3000
Test	2500	2500	5000

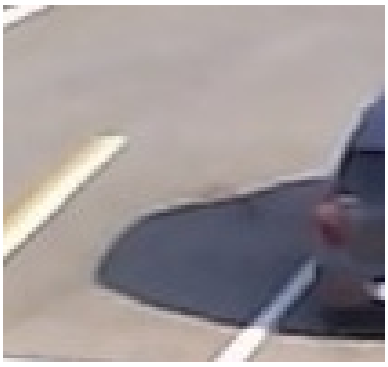
Table 6.2: Data Distribution for Training/Validation/Testing

CNN Training Configuration	
Batch Size	100
Momentum	0.9
Learning Rate Policy	Step
Base Learning Rate	10^{-3}
Weight Decay	10^{-6}
Number of iterations	500
Network	VGG-16
Input Size	100 x 100 x 3
Solver Type	SGD, Adam, Nadam
Loss	Categorical Cross-Entropy

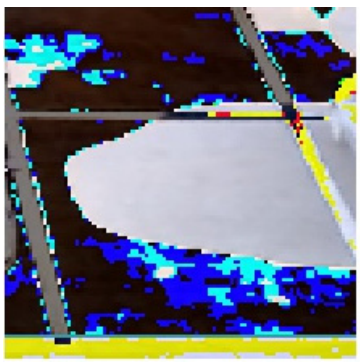
Table 6.3: CNN Classification Training Configurations

6.1.3 Results

The CNNs are the new classification trend today given the positive results they produce in literature. In this project, amongst all the other networks, VGG-16 present a simple architecture compare to the more complex one used in AlexNet. Some changes at the top of the original VGG-16 allowed to classify between 2 classes instead of 1000.



(a) Original image



(b) Image mean subtracted

Figure 6.1: Example of ambiguous vacant parking

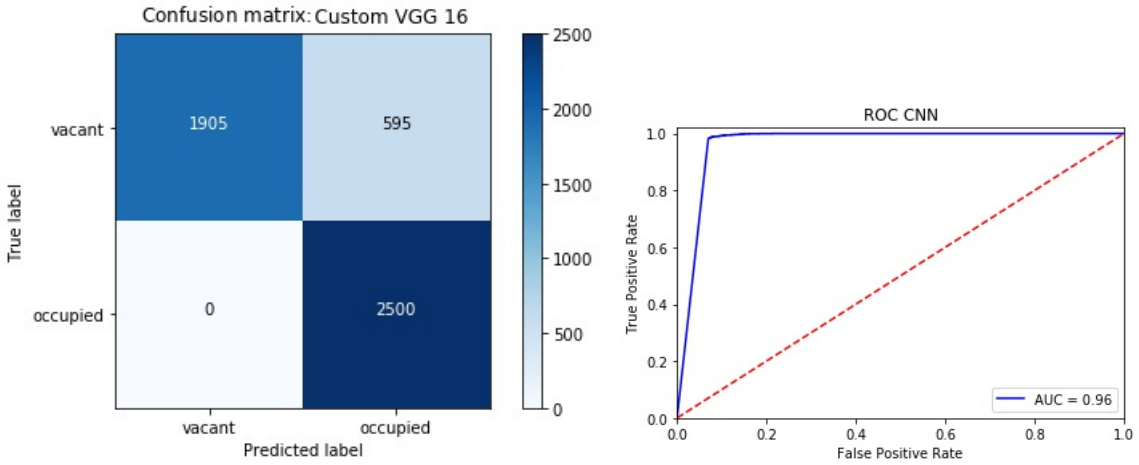
After training, validation, and testing, we evaluated the classifier to calculate the confidence index during deployment time. Figure 6.2a shows the confusion matrix generated by the trained classifier on real-time images. One critical remark is that it can theoretically and unambiguously classify an occupied spot. Some vacant spots are classified as occupied, due to some images that are similar to Figure 6.1a where the shadow is confused to a car. As we can see, Figure 6.1b which is an instance of the situation of the previous one. Before the classification works, the original image is mean subtracted. That reveals that gray area from Figure 6.1b that tells that there is a significant change in that particular area between that image and the mean image. This implies that, since that figure contains a considerable

change of pixel at one area, this implies that a car might be present in the image.

Performance Metric		
Accuracy	88.10%	
Precision	80.78%	
Cohen's Kappa	0.76	
F1-Score	0.86	vacant
	0.89	occupied

Table 6.4: CNN Classification Training Configurations

On average, the classifier performs well given all the external parameters, like the weather, and the multiple cameras. Table 6.4 contains all the evaluation metrics of the built classifier. The precision and Cohen's Kappa values describe the performance of a suitable classifier with an acceptable error rate. The accuracy as well is reasonable given that this classifier is better than Wu *et al.* [2007] who got 83.57% for example when using the SVM. Regarding the individual performance per class, both of them return an excellent score during classification at deployment time. Finally, Figure 6.2b shows an Area Under the Curve which shows that the classifier is confident at 88% in taking the right decision taken during the classification.



(a) Confusion matrix CNN: Custom VGG16

(b) ROC curve CNN: Custom VGG16

Figure 6.2: Confusion Matrix and ROC curve of custom VGG16 CNN classifier

6.2 Support Vector Machines

This section extends Section 2.2 with details on the implementation of the systems using SVM. The previous lines show that CNN's are good with some state-of-the-art algorithms for classification tasks due to their "automatic" feature extraction technique with the stack of layers. Although the optimal configuration of all the hyperparameters is still an active area of research, most of the time some tweaks on the default parameters produce acceptable results. Whereas SVM's also rely on many important parameters that are crucial for the classification such as feature extraction and kernel selection. Good extracted features for the SVM might produce better results than the CNN's.

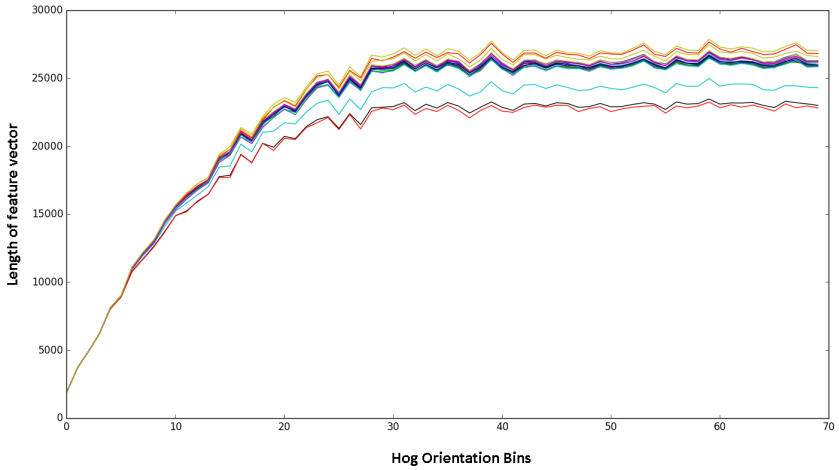


Figure 6.3: Activation on HOG critical points

6.2.1 Preprocessing & Features Extraction

For all classification procedures, whether it is about CNNs or SVMs, all the images are preprocessed based on Section 4.2.3. For the features extraction, many techniques are available. In this work, we first explored the Histograms of Oriented Gradients (HOG) which is an extension of Section 2.3.1. Further down, by curiosity, what could be the effect/response of using features built by the CNN as input to an SVM classifier?

6.2.2 Histogram of Oriented Gradients

HOG is one of many techniques that is very useful when it comes to features extraction from images. In this work, although the default values permitted some good features to be extracted, Figure 6.3 shows effect of the orientation bins on a given image to return orientation gradients that are turned into exploitable feature vectors. This Section defined the best HOG hyperparameters for the best classification accuracy of the SVM. From Figure 6.3, some observations can be made:

- $\theta < 20$ orientation bins produce the best feature vector size for our image,
- $\theta = 20$ orientation bins produce an acceptable feature vector size for our image. The figure above shows that, the length of the feature vector for this specific orientation bin is twice larger than lower ones.
- $\theta > 20$ produces the similar amount of features for the same image as $\theta = 20$, which means that feature vectors cannot be selected from this region.

From the previous points, one would then select for the orientation bin $\theta = 20$, for a 50 x 50 x 1 image. The trade-off here is the length of the feature vector produced by the specific orientation gradient. It increases dramatically when the orientation bin increases. In the case of $\theta \geq 20$, the length of the feature vector will not be optimal for the classification as it will increase the training time. An acceptable value for θ could be such that:

$$\begin{aligned} \theta &\mapsto f(\theta) \\ (8, 10) &\rightarrow (6000, 13000) \end{aligned} \tag{6.1}$$

where the domain here is the number of orientation bin, and the range is the activation on the turning point in the image which is correlated to the size of the feature vector. Table 6.5 shows the full configuration for the project's HOG.

HOG Configuration	
Image size	50 x 50 x 1
Bins	9
Pixels per cell	2
Cells per block	5
Block Normalization	L2
Feature Vector Length	6500

Table 6.5: HOG Configuration

6.2.3 Train & Validation

The most important part of training is the set of features that are used for gaining the most from the image to pass it to the kernel. Table 6.6 shows the configuration for the training of the SVM. We used the library Scikit-learn [ScikitLearn 2017] to train, validate and test the classification_SVM unit.

	Occupied	Vacant
Train	6000	6000
Test	2500	2500
Validation	1500	1500

Table 6.6: Data Set Settings

6.2.4 Results

Given the configuration stated in Table 6.6, the performance of the resultant classifier after training is summarized in Table 6.7. We recall that the deployment data comes from a camera with parameters different from the one used for the collection{testing and validation}. The graph below shows that the SVM associated with the HOG performs well in real life (both for the linear and RBF kernels), although it runs best with a cloud-free or sunny sky. When it is not the case, the accuracy drops dramatically due to the misclassification generated by the effect of the clouds on the windscreen or, the color distribution ration between ground/car or ground only in a specific spot becomes similar, which implies that an occupied spot can be identified as a vacant one.

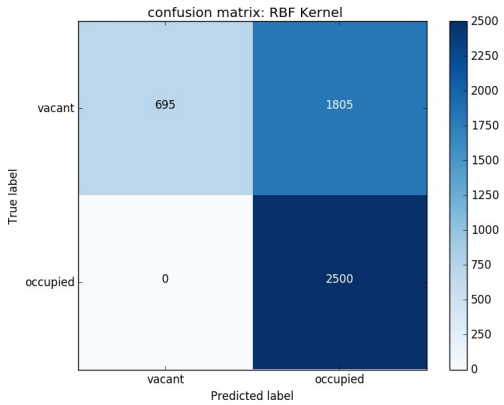
As stated earlier, the table above tells us about the training accuracies of the SVM using two different kernels. At the end of this work, we want to compare the accuracy of the system trained from high-quality images to a real-life scenario that takes as input images coming from a cheaper low-quality camera. The confusion matrix from Figure 6.4 shows that the linear kernel has an error rate inferior to the RBF kernel. Although both of them perform very poorly on the new images coming from the webcam, with a better classification rate on the linear classifier, the reason being the strong activation of the HOG features on the shadow reflecting on the ground like in Figure 6.1b. By increasing the number of orientation bins, which increases the strength of the HOG, the result enhances the activation of false positives. The AUROC (Area Under the ROC) shows that the classifier strictly relies on chance when it comes to classifying vacant space in either case. Figure 6.4c and Figure 6.4d explain the previous sentence and justifies it with the AUC scores which are 0.64 and 0.68 for the RBF and the Linear kernel respectively. Those are generally poor scores according to Darwin [2015]. The value of the Cohen's Kappa confirms the previous statement since it falls within the fair agreement according to McHugh [2012]. In other words, the Cohen's Kappa returns a “relatively strong” value only and only due to the substantial agreement between the predicted and actual value of the occupied spots. In either case

of the Kernel, all the occupied parking spaces are correctly classified in real-time – i.e. images are classified faster than they arrive.

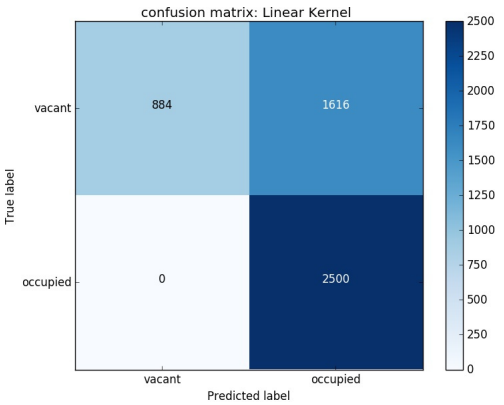
Finally, to bring more details to the previous paragraph, the F1-Score justifies the weak performance in overall of the classifier. From Table 6.7, in the F1-Score section, from either case, vacant spots have a score which is at least the third of the occupied ones. That explains the overall low accuracy of the classifier.

Performance Metric			
Kernel	RBF	Linear	
Train Accuracy	84.37	84.57%	
Deployment Accuracy	63.90%	67.68%	
Precision	58.07%	60.74%	
Cohen's Kappa	0.27	0.35	
F1-Score	0.44	0.52	vacant
	0.73	0.76	occupied

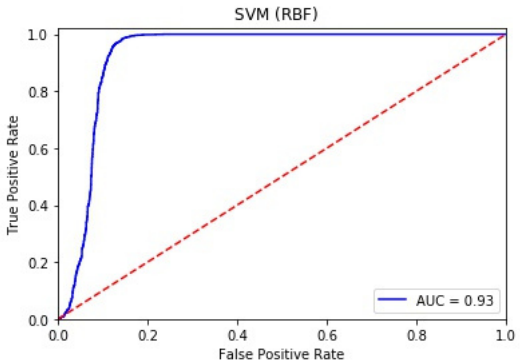
Table 6.7: SVM Performance metric



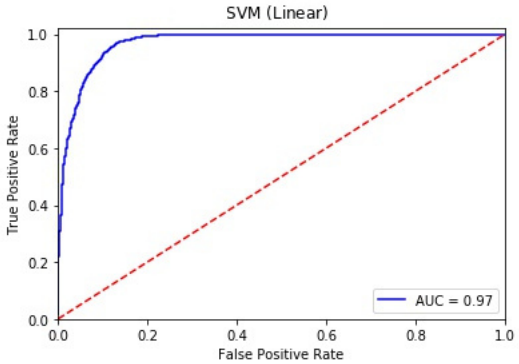
(a) Confusion matrix SVM RBF kernel



(b) Confusion matrix SVM Linear Kernel



(c) AUC of the RBF Kernel



(d) AUC of the Linear Kernel

Figure 6.4: Metrics of classifier SVM with different Kernels

From Table 6.7, the SVM kernels perform very well for images coming from the same source (camera). It is important here to note that the accuracy of the Linear kernel is higher than the RBF kernel. Theoritically the RBF kernel should be able to do anything that the Linear kernel can do, and more. The difficulty is that is can be harder to tune and it can take longer to train. This is an intersting observation, as the RBF is more powerful as it is a superset of the Linear kernel. This might indicate

that the data is quite linearly separable, so the Linear kernel find the hyperplane quicker than the RBF. It means that RBF could be tuned better or needs more training and that should be investigated further.

The deployment camera added some more details on the incoming images, which increases dramatically the number of false positives, especially for the vacant spots identified as occupied because of the shadow created by the adjacent car in the parking spot next to it. This is much worse than CNN. Compared to the SVM, the CNN generalize better to unseen datasets.

The next section experiments the strength of the CNN with the classification unit of the SVM calculated by the kernels.

6.3 Convolutional Neural Networks as a Feature Extractor for Support Vector Machines

This hybrid approach is motivated by the fact that both algorithms have some advantages and disadvantages. The features extracted for the SVM can come from various feature extractors, for instance, the HOG used in the earlier sections. The HOG needs to be tuned well for it to be at it best. Unfortunately, it is specific to a situation, for instance, in Section 6.2.4, the HOG returned more than 80% training accuracy and produced in deployment 65% in average at deployment. This is because the occupied spots were correctly classified with an F1-Score of 0.75 on average. From their structural architecture, the CNN extract features from the dataset as part of training, then classify them as described in Section 2.1.

6.3.1 Architecture

As stated earlier, this hybrid method is an attempt that aims to bring a contribution to the parking space management problem using computer vision. So far, monotonic methods have been used including the SVM and CNN.

The proposed system is based on the power of the feature extraction algorithm of the CNN which lies on the internal architecture of the CNN described in Section 2.1. To the top of the CNN is plugged either of the two kernels from Section 6.2 which are the RBF and Linear Kernel.

6.3.2 Preprocessing & Features Extraction

This technique is slightly different from the previous ones where the original image is broken down into features using one function. In this case, where the features are coming exclusively from the CNN, the images should not be shuffled. If they are, each image should keep its label or class close, like a tuple: $\{image.jpg, class_name\}$. Given the images and their corresponding label, they get split into two parts: the images in their full shape and the labels. From this time, the images should not be shuffled, otherwise, the order between the images and the labels will just get lost. The dimensionality of the tensor holding the images is $D = [X, m, m, c]$ where X is the number of images in the tensor, m is the size of one image (which is usually a scare matrix) and the number of channels of each image represented by c.

The image is preprocessed as shown in Section 4.2.3 to obtain images (data points) such that the dataset X has a unit standard deviation and the mean centered at 0 for all the images to be normalized.

After the image normalization, the features need to be extracted using the fully trained CNN. The network architecture of that CNN is identical to the one in Section 6.1, with some little changes. That

network contains all the layers for training, loss and gradient update computation. All the top layers including the softmax layers are thrown away, leaving only the very first fully connected layer, as shown in Figure 6.8. The following image is the simple representation of the feature extraction unit to train the hybrid system:

The input is the whole data A is of dimensionality D, which complies appropriately with the input of the CNN used. X does one forward pass, a prediction on all the images, which returns a matrix of dimensionality $D_{reduced} = [X, 4096]$, where X is the number of images in the matrix, that correspond to the number of images of the tensor used as the input and 4096 corresponds to the number of features per data point.

At this stage, the extracted features barely have a sense to a human, that is the reason why from earlier the images and their respective label should be in the same other such that each point from $D_{reduced}$ corresponds to its correct label as it was with D before the extraction.

After all the process, the features and their label are stored in different variables for the training of the SVM classifier. Again, the kernel used here will be the RBF and the Linear.

Input (**100** x **100** x 3)
Convolution 3D - 64
Convolution 3D - 64
MaxPlooling
Convolution 3D - 64
Convolution 3D - 64
MaxPlooling
Convolution 3D - 128
Convolution 3D - 128
Convolution 3D - 128
MaxPlooling
Convolution 3D - 512
Convolution 3D - 512
Convolution 3D - 512
MaxPlooling
Fully Connected - 4096
Fully Connected - 2018
Fully Connected - 1024
Fully Connected - 2
Softmax

Table 6.8: Custom VGG-16: The red units correspond to the layers removed to extract features from the convolutions, and the green layer corresponds to the layer where the features are actually produced.

6.3.3 Training & Validation

This hybrid approach requires some time and setting to ensure quality training, testing, and validation. Since this classification unit is made of two different algorithms, they need to be trained differently, which is an approach similar to the one used by Niu and Suen [2012].

The CNN unit is trained as from Section 6.1.2. Since the CNN has its own prediction unit (Softmax), it has to be removed, including all the fully connected layers except the first one. The CNN block is a set of optimized parameters of the trained CNN. The features comes from the CNN as described in Section 6.3.2.

The classifier training happens in the SVM block. This research uses the same data for all the algorithm used to avoid biased results. Therefore, as a reminder, Table 6.10 represent the data used in this section to train the hybrid unit. The optimized features are then fed into the classifier with the parameters from Table 6.9.

Kernel	RBF	Linear
Input (Feature size)	4096 x 1	4096 x 1
C	1	1
Loss	—	Square Hinge
Gamma	auto	—
Iterations	1000	1000

Table 6.9: CNN → SVM parameters

	Occupied	Vacant
Train	6000	6000
Test	2500	2500
Validation	1500	1500

Table 6.10: Data Set Settings

6.3.4 Results

The training/validation phase produced some interesting results. After all the iterations, the hybrid model is saved disk for the testing. The testing data is different from the training and validation data as it was taken with different camera type, and it has been generated from the detection unit.

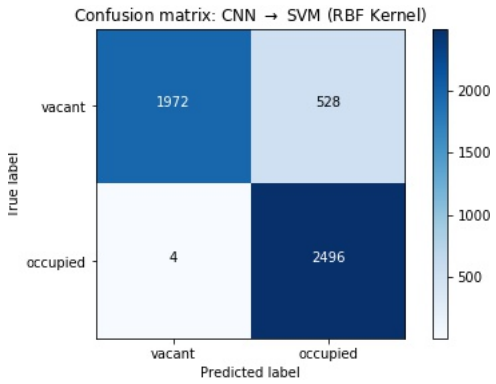
Performance Metric			
Kernel	RBF	Linear	
Train Accuracy	97.97%	98.67%	
Deployment Accuracy	89.36%	86.60%	
Precision	82.54%	80.63%	
Cohen's Kappa	0.79	0.69	
F1-Score	0.88	0.84	vacant
	0.90	0.86	occupied

Table 6.11: CNN → SVM Performance metric

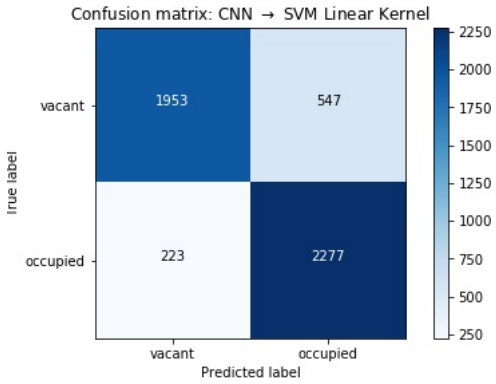
For the hybrid classifier, Table 6.11 shows that the RBF kernel is better than the Linear one. It might show that the CNNs probably have not pulled out the same linearly separable features that the HOG have.

The results are quite impressive and surprising because of the strength of the RBF kernel in the hybrid unit. The previous table also shows that, despite the superiority of the RBF kernel, both perform well in all the measurements used to evaluate them.

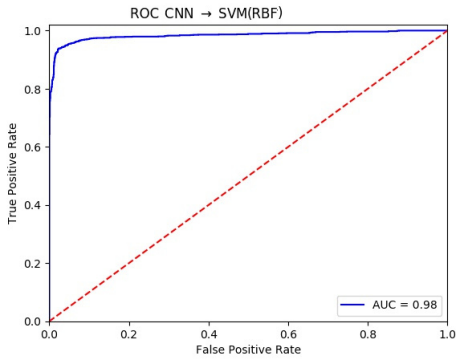
Figure 6.5 details the numbers obtained from Table 6.11. Figure 6.5a shows that the classifier with the RBF kernel misclassified four occupied spots as vacant and 526 vacant spots as occupied, which is acceptable given the change of weather and the noise present in most of the images sent to the classifier. As seen before, the vacant spots, especially in the afternoon, producing a shadow that stays on the next spot either on its left or right depending on the position of the sun. Again increasing the training set might update the values of the parameters for this problem. Its corresponding Area Under the ROC



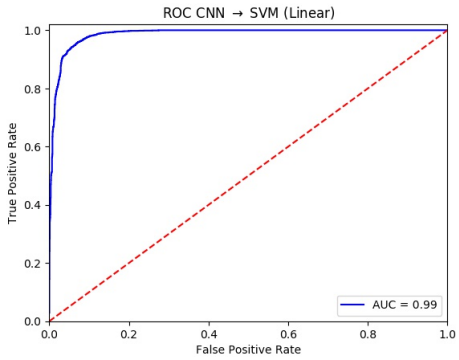
(a) Confusion matrix CNN → SVM RBF kernel



(b) Confusion matrix CNN → SVM Linear Kernel



(c) AUC of the CNN with the RBF Kernel



(d) AUC of the CNN with the Linear Kernel

Figure 6.5: Metrics of classifier CNN → SVM with different Kernels

curve, Figure 6.5c, shows that the classifier relies less on chance to predict, with an AUROC score of 0.89, which is acceptable.

Figure 6.5b shows the confusion matrix of the CNN → SVM with the Linear kernel. This configuration produces performances less acceptable than the previous one with the RBF kernel, but they have a similarity in terms of misclassification. Both configurations have a high rate of error when it comes to classifying vacant parking spots, which is 21%. The reasons being the ones described in the previous paragraph.

6.4 Conclusion

This chapter described in depth the classification process used in this research project. We have to note that all algorithms used in this chapter were trained on the same set of data. The CNN as a classifier is a modified version of the VGG-16. Instead of 1000 classes, 2 classes were implemented at the top of the network to accommodate the number of classes related to the project. The CNN classifier was trained and validated using one set, then tested using a different dataset. As result, the CNN as classifier returns acceptable scores as shown in Table 6.3. The AUROC curve shows that it does not rely on change most of the time to run classifications.

The SVM requires some features to run correctly. The HOG was investigated to get the best configuration for gain in terms of time and performance, because the more features generated, the greater the computation time becomes. To have the best the SVM, the kernels were experimented separately: RBF and Linear. They have different properties as the linear kernel is a linear function

in the space, whereas the RBF kernel is a more complex function with a higher order polynomial. Figure 6.4 summarized the performance of the SVM using it different kernels. They performed poorly compared to the other algorithms. Table 6.7 shows that they barely obtained 60% for precision and 70% accuracy.

Lastly, the CNN as feature extractor is the heart of this work. Regarding the CNN section of this classifier, the model of the previously trained CNN classifier was used, but the only difference was the number of layers that decreased. All the fully connected and classification layer were ejected to give the place to the SVM. For this step, given the SVM model, the dataset that was fed into the CNN was in order, in such a way that each produced vector corresponds to a label since those vectors mean nothing to a human eye. The SVM used both the CNN features and their corresponding labels for training/validation/testing. The results were surprising as shown in Figure 6.5 and in Table 6.11.

Chapter 7

Analysis

Throughout this work, many results were produced to find the best classifier to detect and classify parking spaces. Figure 7.1 summarizes all the results done in this research project using the deploy data. As a reminder, the deploy data comes from a low-quality camera. It was collected 8 months after the beginning of the research and has not been used for training or hyperparameters selection. This dataset is entirely foreign to the training and validation datasets. From Table 7.1, most researchers from the literature used the CNN or the SVM to solve problems with good accuracies. All those authors used the previously written algorithm except for Niu and Suen [2012], who experimented its hybrid CNN→SVM with the MNIST dataset.

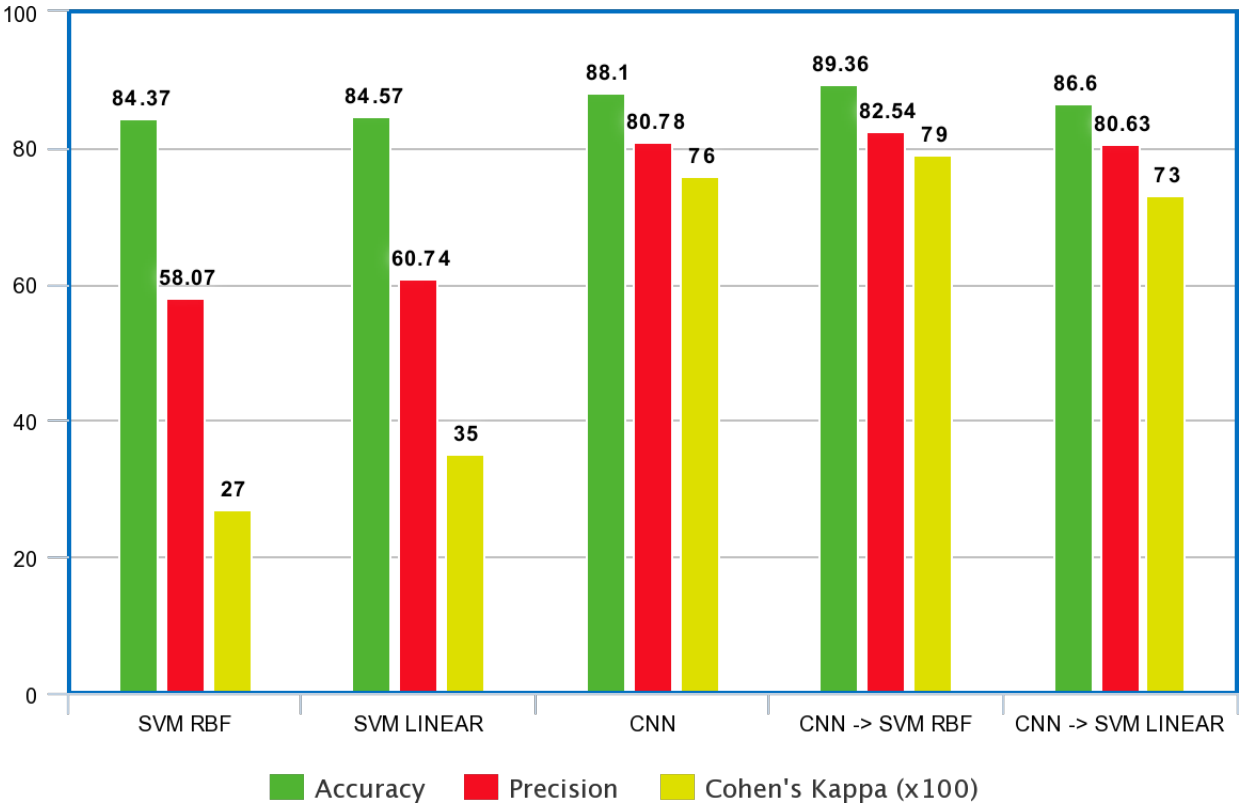


Figure 7.1: Comparison of all the used algorithms

From Figure 7.1, the accuracy of all the used algorithms are above 80%, that is mostly due to the high success rate of the occupied parking space over the vacant spaces. The features used for the SVMs are made of HOG. Those HOG features could not well define the status of a parking space. This

justifies the low score produced by the SVM RBF and SVM Linear in terms of accuracy and Cohen's Kappa. Despite the high accuracy returned by those two algorithms, the values they have scored for the precision, 60% in average, and Cohen's Kappa, 0.31 in average, do not make them very good classifier for this research that aims to get a high success rate when classifying the spots.

References	Method	Best Accuracy
Wu *et al.* [2007]	SVM	94.5%
Niu and Suen [2012]	hybrid CNN SVM	94.40%
Amato *et al.* [2016]	CNN	86.61%
This work	CNN → SVM	**89.36**%

Table 7.1: Comparison of this research results with the literature

Concerning the other algorithms, the traditional CNN, the CNN → SVM {RBF & Linear }, they produced better results than all the SVM variants. Most researchers prefer the CNN because of the selection of the relevant features, compared to the HOG, where those need to be hand-crafted. They can return good results as in Wu *et al.* [2007]'s work. In this project, all CNN related classifiers returned an accuracy above 85% and a precision above 80%. Therefore the CNNs are suitable for feature descriptor for the problem since they accurately identify the status of the parking space examined.

Throughout this research and all the experiments performed, the RBF kernel has always scored a value less than the Linear kernel. However, when using the CNN → SVM, the RBF kernel produced from far better results compared to the Linear one.

Regarding speed, the CNN → SVM RBF is the classifier to use since it has the best properties compared to all the others tested algorithms. Given that the SVM is at the top of the network, it has a faster computation time compared to the traditional Fully Connected layers added to the SGD unit [Szarvas *et al.* 2005]. The speed is also justified by the CNN layers that are already optimized for the problem, which makes the forward pass process perform faster fast.

Performance Metric						
	CNN	SVM	SVM	CNN → SVM	CNN → SVM	
Kernel		RBF	Linear	RBF	Linear	
Train Accuracy	98%	84.37%	84.57%	97.97%	98.67%	
Deployment Accuracy	88.10%	63.90%	67.68%	89.36%	86.60%	
Precision	80.78%	58.07%	60.74%	82.54%	80.63%	
Cohen's Kappa	0.76	0.27	0.35	0.79	0.69	
F1-Score	0.86	0.44	0.52	0.88	0.84	vacant
	0.89	0.73	0.76	0.90	0.86	occupied

Table 7.2: Overall Results

Finally, Table 7.2 is a general representation of all the algorithms used in this work. It is an extension of Figure 7.1. This table shows that the SVM RBF with the Histogram of Oriented Gradients for classifying parking bays was the worst of the list regarding the overall performance. The CNN as feature extractor for the SVM RBF produces the best results from all those comparisons which make the hybrid system a better approach for the classification of the parking bays.

Chapter 8

A Parking Space Detection System

The previous chapters contribute to the technical part of the system. The data flow is seamless to the user. The major components of the system are the camera and the processing computer that will be called server.

8.1 The camera

This work uses a low-resolution camera, like a webcam, during the deployment phase, to simulate a real-time capturing with an IP camera, mounted on a wall or a stand facing the parking space. As in Section 4.1, the camera is mounted on a wall or a dedicated stand, facing the parking lot, in such a way the axis of the camera is parallel to the parallel lines of a spot.

In the case of an IP camera, the connection to the server happens instantly since those are equipped with network utilities like the ability to connect to a network and transfer images directly without a computer. The research's camera is just a webcam that does not have any network utilities, so an additional computer (a laptop) helps the camera to send the images to the server, as it is a bridge between the two components (the camera and the server).

8.2 The Server

8.2.1 Overview of the System

The server in this project is the computer in charge of the detection, classification and messaging to the user processes. The computer linked to the webcam sends an instruction (script) to the camera, to shoot every minute. After the camera captures a real-time image from the parking, it is saved temporarily in the laptop connected to it in a stack style, such that the last image that comes, will get processed, leaving behind the already existing images, since they have been processed. The queue process is ensured just by the file naming. Later on, another script in the computer connected to the webcam fetches the first image in the stack, which is, in other words, the last image recorded and sends it through the network to the server.

The server runs the detection unit once every minute, this because multiple reasons like the change in weather, or a passing car hiding some white lines on the ground. As seen in Section 5.5, the weather modifies the way the system works. A sunny or cloud-free sky will return more parking space more

than an overcast one. It increases the chances of getting the maximum number of available spots in a parking.

The detection is done automatically as opposed to the manual detection done by Nyambal and Klein [2017], where coordinates are entered on the image to create a map of the parking space. After the detection phase on the server, each spot is individually classified. Basic color codes are assigned to the state of a spot where green means vacant and red means occupied.

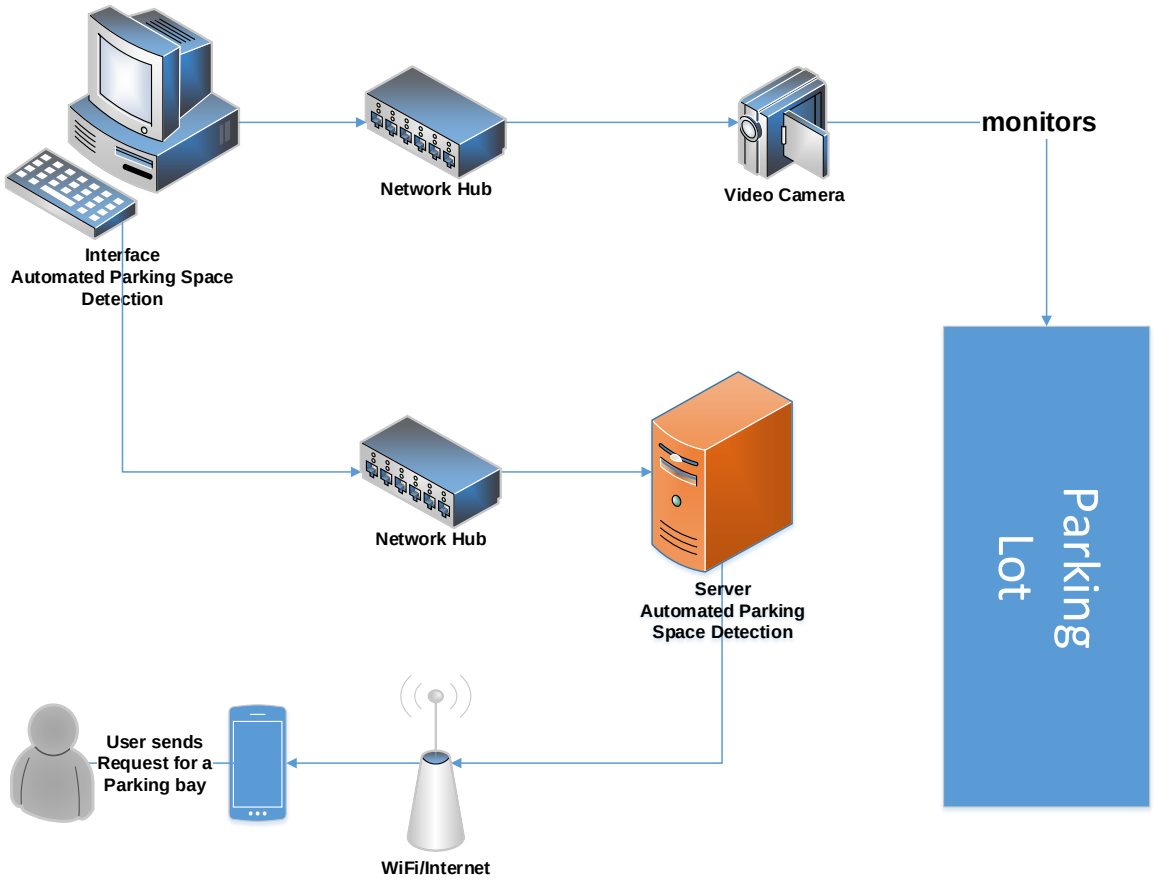


Figure 8.1: Describes how the video frames containing the images the parking lot leave the camera to the computer that has the Automated Parking Space Detection system. After the images data have processed by the system, the result, which is a JSON file, is sent to the server. The server will then send the JSON file to the android phone when requested by the user.

8.2.2 Detailed View: The whole process

This section provides with a closer view of what happens mostly in the orange system in the data flow Figure 8.1. That figure shows in detail what exactly is happening internally in the system. The different colored arrows are explained below:

- **Red**: Will happen only every minute so that the system does not get overloaded. The detection unit reports to the system for the classification unit to take over on the subsequent images. This setup allows the system to know about the location of the spots given the images sent by the camera to the system.
- **Green**: The classification updates the system about the status of each detected spot. This is a

repetitive process since each parking space is classified individually.

- **Gold**: The spots are already detected and classified. The System at the first iteration sends the very first frame from the camera to the monitor. Later on, the system updates that first frame with the detected spots coming from the classification unit. The monitor is an endpoint and does not affect the system if it is withdrawn. Potential users wanting to park their car are the target endpoints of the system, that will automatically bring them to the closest available parking bay.
- **Dark Blue**: The system receives the frames of the parking lot from the camera for further processing. One frame comes once every minute.
- **Dotted Orange**: This represents the trigger from the system to start the whole process which is to start shooting the parking space with the camera. It is done with a repetitive control structure and will die only when manually stopped. It happens once.

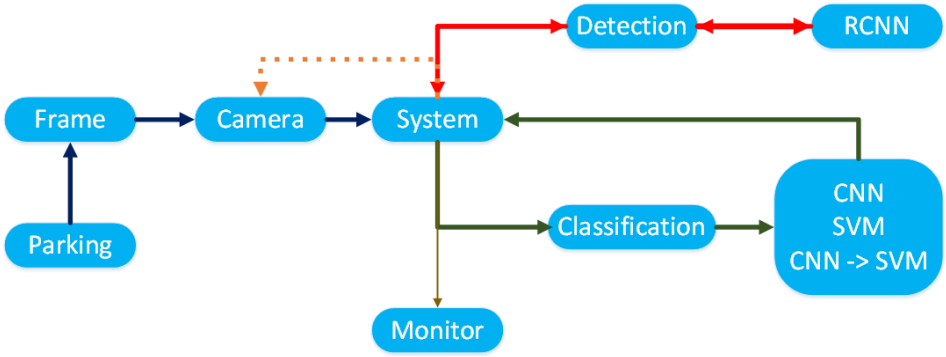


Figure 8.2: Detailed view of the system

8.2.3 Detailed View: The classification Unit

Zooming in the whole system described earlier brings us to the classification unit from Figure 8.2. That is the heart of the system. Figure 8.3 shows the working of the classification part of the system on an image of a parking lot. In this section, we assume that the image already went to the detection unit. Therefore the system has the locations of all the detected parking spots. Each phase is described below:

- **Purple**: From the original image returned by the camera, a detected spot is cropped to produce just an image containing only either a vacant or occupied spot.
- **Dark Green**: The cropped image of the spot becomes the input of the classification unit.
- **Red**: The system did not take into account the SVM or the normal CNN. It consists only of the hybrid classifier: CNN $\rightarrow$ SVM. It extracts the features from the CNN to return a 1D vector, associated with the corresponding labels to feed the SVM with either the RBF or the Linear kernel.
- **Green**: The result of the classification unit produces a color corresponding to the decision: green and red corresponding respectively to vacant and occupied.

8.3 Visual Results of the Classifier on Real Images

Now let's see the system as a black box. After trying all those algorithms on real-life video coming from the webcam of the research some visual results could be captured to show the efficiency of all

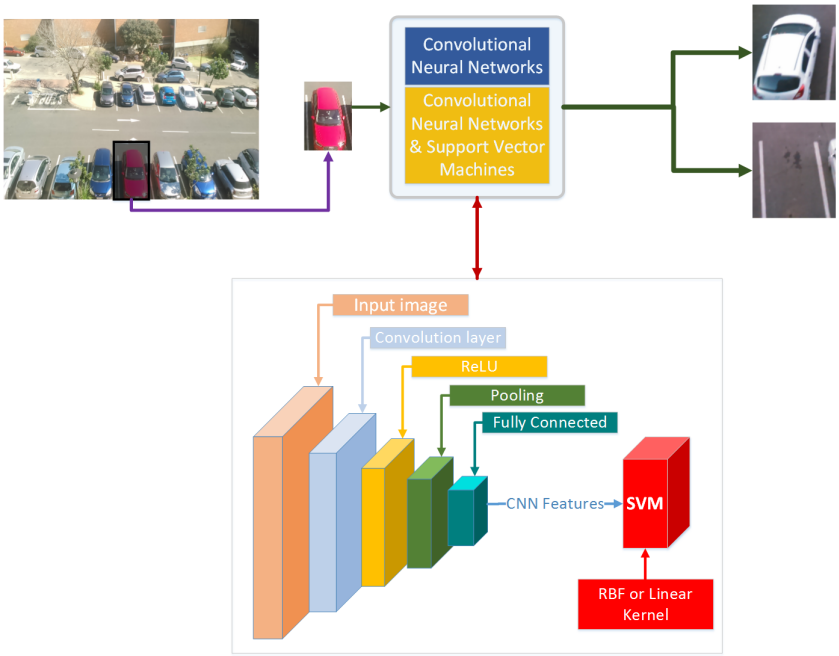
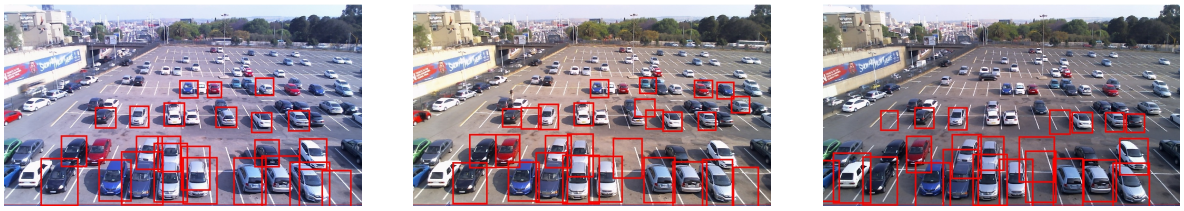


Figure 8.3: Classification Unit

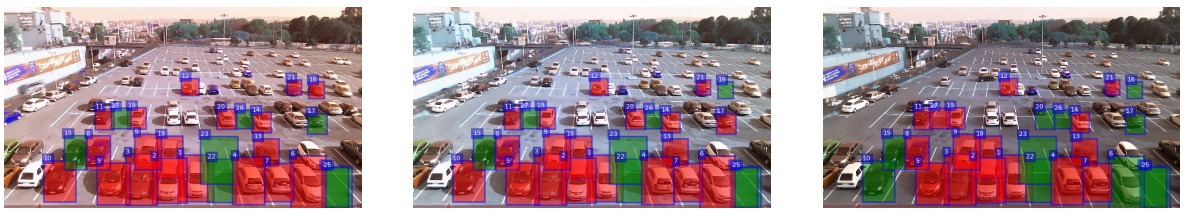
those algorithms, although the results are very similar no matter what algorithm is being used. Figure 8.4 and Figure 8.5 show respectively the detection and classification unit in action.



(a) (b) (c)

Figure 8.4: Detection visuals

Figure 8.4 represents the parking lot with many spots detected at a different time of the day. In Figure 8.4a and Figure 8.4b, on the first row from the bottom, they have the same number of detected spots, but on the second row, some more spots are detected but at different positions. In Figure 8.4c, because of the absence of the sunlight, fewer spots are detected. After that, the classification unit takes over based on the location of all the possible spots detected as shown in the previous figure.



(a) (b) (c)

Figure 8.5: Classification visuals

Figure 8.5 represents the target parking lot with all the locations loaded and already classified.

In Figure 8.5a, in average, the classifier performed well, on the second row from the bottom, and to the left, an occupied space is classified as vacant, as well as on the third row to the right. Since the classification happens every minute, those errors are corrected depending on the external conditions to the classifier (weather, people, lighting changes) using temporal information, or multiple vacant classifications can be performed before reporting. The classification runs on a static location file that holds the location of all the spots, that is why the classification on all the images in Figure 8.5 apply on some areas. After 10 mins, the setup will change as the detection unit will catch more locations.

8.4 Conclusion

This chapter provided the output generated by the system to detect and classify parking spots. Although the system runs, there is still some misclassification that are the direct interpretations of the confusion matrices drawn earlier. In real-time, the error generated is tolerable given the fact that the classifier has one minute give a new decision, and the precision and accuracy of that decision depends on the classifier used.

On the user-side, there are tolerance levels that can be raised from the detection unit. Vacant bays that are actually occupied might make users angrier because after moving to that bay finally finds it occupied. Moreover, the occupied bays that are actually marked vacant are more tolerable situations. Thresholds can be set on the levels of confidence that the system needs to minimize user frustration.

Chapter 9

Conclusion & Future Work

9.1 Conclusion

Parking space management is a problem that affects all parts of the modern society. According to Bhana [2017], the University of the Witwatersrand lacks parking spaces for students and staff, because of multiple reasons, one being the reallocation of a parking space to a different area. Another one common reason is the lack of management that can help drivers to park their car, by going ahead with the car knowing that they will get a space in a monitored parking lot. In this research project, we attempted to address this issue by using monitoring cameras to train and deploy a hybrid classifier which is built from CNNs for features extraction and SVMs for classification.

Solutions to this issue moved gradually from physical devices installed around the parking spaces itself, to more sophisticated ideas like machine learning and computer vision. In Chapter 3, we saw the idea to study the pixels of an image to withdraw valuable information from it (features), increased chances of getting the perfect vision system to locate. Then, classify a parking space, and assist the driver to park her/his car if the spot is free, otherwise, inform the driver accordingly to save time and petrol.

Valuable information from an image is the critical element to approach this issue using computer vision. Chapter 4, we have shown the process of getting the information from the spot from data collection to preprocessing. Getting the correct data is crucial for the whole process. Therefore, we used GoPro camera to collect the image of an active parking lot, and due to the intrinsic parameters of the camera, the images needed to be corrected. Mean normalization is essential to reduce the variance between the pixel values while bringing their mean around zero. We saw that this step does not always enhance the processing of an image in the presence of more than one object or noise in it. However, when a single image is present in an image (crop of a parking spot), mean normalization is very important to enhance the presence (or absence) of the object of interest in the image.

Chapter 5 addresses the issue of automatically locating a spot from an image containing a parking space. This task is mostly innovative in the way that the map of the parking lot needs to be designed and after that classify each mapped spot. Although this eases the lengthy process of manually mapping all the spots in the image, it is not perfect. Many spots are missed during the detection due to various factors like weather, that generates a sharp change of the color distribution on the ground (spot white lines included) or on the car. Despite the imperfection of the detection unit, we went forward to classify them. Potential solutions to increase the confidence of the detection unit consist of including temporal information like the Kalman filter to collect bays positions over time. That will reduce the blinking effect of the detection unit when not confident about a parking bay.

In this dissertation, we compared the performances of five different techniques mostly based on

features generation which are the SVM RBF, SVM Linear, CNN, CNN → SVM RBF and CNN → SVM Linear. The HOG was the primary feature extraction for the SVM since it has been widely used in the literature. We noticed that the HOG required some correct tuning to return valuable feature for the SVM to process them. For the SVM RBF and SVM Linear with our HOG configuration returned very low scores, especially 59% in average for both classes (occupied and vacant), which cannot be acceptable for a real-time system due to a high level of uncertainty. The custom VGG-16 CNNs returned a precision of 80.76%. Since the SVM are built around a function (kernel), we attempted to throw the features generated by the custom CNN to the SVM RBF and SVM Linear and we obtained a precision of 81.54% in average for both classes, which in this case is an exciting finding for the future of the CNN as feature extractor. The result of that hybrid system outperformed all the other methods experimented.

9.2 Contributions

The main contributions of this work:

- RCNN for automatic bays detection: From the literature, the lot was manually mapped and calibrated before classifying the bays. Using RCNN which is based on image segmentation a,d selection search, bays are automatically detected in a parking lot,
- CNN as feature extractor for SVM: The CNN detect and extract features as part of training. That step avoids the selection of the wrong or less accurate set of features for the classifier. For the automated parking space detection problem, this is the first attempt at using SVM to classify features from CNN,
- Dataset: The dataset used in this work was collected as part of the research. The available dataset from de Almeida *et al.* [2015] for parking space did not help due to the distance of the camera to the ground, the angle of the bays, the texture of the ground. This dataset was initially designed, built and used in Nyambal and Klein [2017]. That dataset is correctly labelled and contains full images of parking lots, and crops of corresponding bays.

9.3 Future Work

Compared to the attempts from the literature, this project presents an autonomous system capable of detecting and classifying parking spots. Some improvements to the current system might be:

- Detection of more spots: Figure 5.4 clearly shows that many spots were missed, but, next to each of those, a spot is discovered. Improving the system will involve either bringing in more data for training or getting other processing techniques might increase the detection rate, therefore, obtaining a more realistic map of the parking lot,
- This project focused more on the automatic detection of the spots and the most appropriate classification technique for a reliable real-time application. Dispatching the updated map of the parking lot to a server, that will direct the driver to the next available spot.
- This research is based on images with the lines of the parking bays parallel to the axis of the camera. More images of diffrernt parking lots, in different rotations of the camera will be used to address rotation invariance.
- Compare the performance of the CNN-SVM hybrid against the well-trained CNN (through hyper-parameter training).

Appendix A

Pseudo Code

Algorithm 1 Mean Normalization

Input: train set, test set, validation set, Train_mean
Output: Normalized train set, test set, validation set

$X_{test} \leftarrow$ All images from test set
$X_{val} \leftarrow$ All images from validation set
X_{train}, X_{test}, X_{val} = RESIZE(X_{train}, X_{test}, X_{val}, (100,100)) ▷ All images with size 100 x 100
$X_{train} \leftarrow$ All images from train set
μ_{train}, σ_{train}= 0
procedure COMPUTEMEANSET(X) ▷ X is a set of images
    $mean$, $summ$ = 0
    for $i \leftarrow$ 0 to $length(X$ - 1) **do**
        $summ$ = $summ$ + $\sum(X[i])$ ▷ Sum of all pixel values of image X[i]
    end for
    $mean$ = $summ$ / LENGTH(X)
    return $mean$
end procedure
$\mu_{train} \leftarrow$ COMPUTEMEANSET(X_TRAIN)
$\sigma_{train} \leftarrow \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu_{train})^2}{N-1}}$. ▷ Where N is the total number of pixel intensities of the image
for $i \leftarrow$ 0 to $length(X_{test}$ - 1) **do**
    $X_{test}[i] \leftarrow (X_{test}[i]$ - $\mu_{train})$ / σ_{train}
end for
for $i \leftarrow$ 0 to $length(X_{val}$ - 1) **do**
    $X_{val}[i] \leftarrow (X_{val}[i]$ - $\mu_{train})$ / σ_{train}
end for
for $i \leftarrow$ 0 to $length(X_{train}$ - 1) **do**
    $X_{train}[i] \leftarrow (X_{train}[i]$ - $\mu_{train})$ / σ_{train}
end for
return X_{train}, X_{test}, X_{val}

Algorithm 2 Parking Spots Cropping

Input: Image_full_parking.jpg, Image_full_parking.xml
Output: spot_n.jpg ▷ n spot number in Image_full_parking.jpg

procedure GETCOORDINATESFROMXMLFILE($PathToXmlFile$)
    X_min = []
    X_max = []
    Y_min = []
    Y_max = [] ▷ Lists of cartesian coordinates of all parking spots, all of type integer-16
    for $line \in$ all lines in $PathToXmlFile$ **do**
        if "$< xmin >$" $\in line$ **then**
            Get the number between "$< xmin >$" and "$< /xmin >$" and append it to X_min
        end if
        if "$< xmax >$" $\in line$ **then**
            Get the number between "$< xmax >$" and "$< /xmax >$" and append it to X_max
        end if
        if "$< ymin >$" $\in line$ **then**
            Get the number between "$< ymin >$" and "$< /ymin >$" and append it to Y_min
        end if
        if "$< ymax >$" $\in line$ **then**
            Get the number between "$< ymax >$" and "$< /ymax >$" and append it to Y_max
        end if
    end for
    return X_min, X_max ,Y_min, Y_max
end procedure
$data_path \leftarrow$ Path to all data collection directories
Create Directory $Results$: Directory that will receive all the snippets
xml_file = newTextFile ▷ Placeholder for the path of the xml file of each image
$image_file$ = newTextFile ▷ Placeholder for the path of each image
x_min = []
x_max = []
y_min = []
y_max = []
for $folder_n$ *in* $data_path$ **do**
    for $file_n$ *in* $folder_n$ **do**
        if $file_n$ contains "$.xml$" **then**
            $xml_file \leftarrow$ Path to $file_n$
            x_min, x_max, y_min, y_max = GETCOORDINATESFROMXMLFILE(xml_file)
        end if
        if $file_n$ contains "$.jpg$" **then**
            $image_file \leftarrow$ Path to $file_n$
            $image$ = Read $image_file$
            $image_copy$ = Copy $image$
            for $i \leftarrow 0$ to $lenght(x_min$ - 1) **do**
                $crop$ = $image_copy[y_min[i]{:}y_max[i],x_min[i]{:}x_max[i]]$
                Save $crop$ on disk in $Results$ directory
            end for
        end if
    end for
end for

Algorithm 3 Appending Newly Bays To Detection Set

Input: Image
Output: Updated_bays_locations

procedure DETECTION($Image$)
    As defined in Chapter 5
    return bays_locations
end procedure
procedure UPDATE_DETECTION_SET(Image, Set_bays_locations)
    if Set_bays_locations is empty **then**
        Set_bays_locations = DETECTION(Image)
    else
        Add Set to $Updated_bays_locations$
    end if
    return Set_bays_locations
end procedure
Image = From camera (Deployment)
Set_bays_locations = {} ▷ Empty set
Updated_bays_locations = {} ▷ Empty set
Updated_bays_locations = UPDATE_DETECTION_SET($Image$, $Set_bays_locations$)

Appendix B

Datasets



Figure B.1: Occupied Parking Spaces: The 2 first rows are the pictures coming from the webcam at the deployment phase. All the edges are blurry: white lines, vehicle's lines. The 2 last rows are the pictures coming from the GoPro Camera for the training/testing/validation phase. All the lines are sharp, wince that camera has a higher definition.

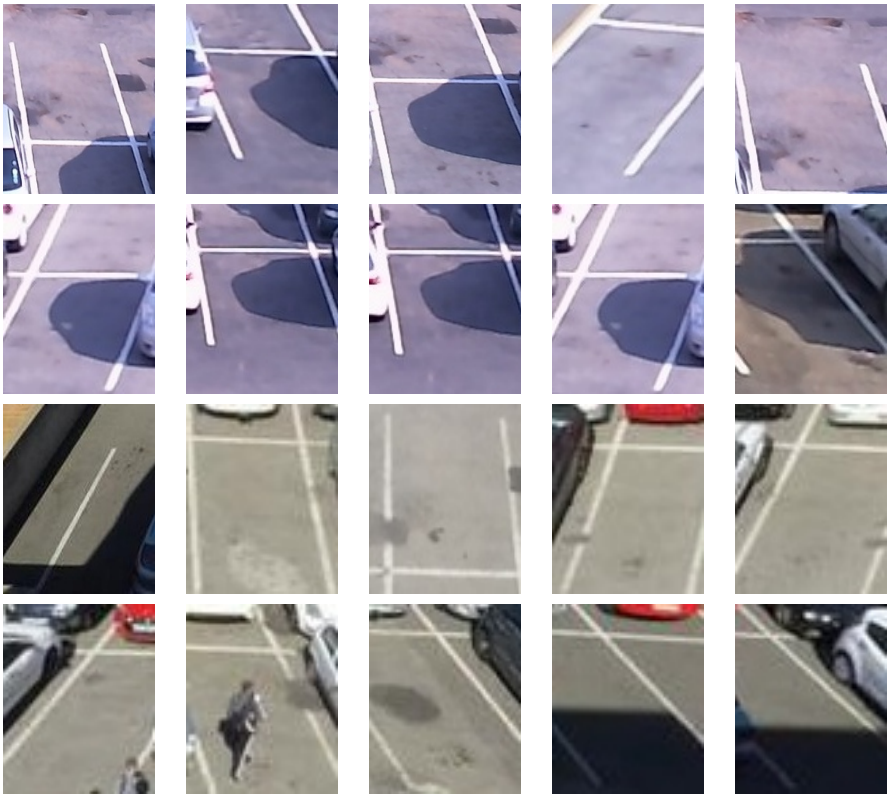


Figure B.2: Vacant Parking Spaces: The 2 first rows are the pictures coming from the webcam at the deployment phase. All the edges are blurry: white lines. The 2 last rows are the pictures coming from the GoPro Camera for the training/testing/validation phase. All the lines are sharper, wince that camera has a higher definition.

References

[Amato *et al.* 2016] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo. Car parking occupancy detection using smart camera networks and deep learning. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 1212–1217, June 2016.

[Amato *et al.* 2017] Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, Carlo Meghini, and Claudio Vairo. Deep learning for decentralized parking lot occupancy detection. *Expert Systems with Applications*, 72:327 – 334, 2017.

[Assist 2017] Park Assist. *Parking Guidance Systems from Park Assist*. https://www.parkassist.com/, 2017. (Accessed on 03/06/2017).

[Banerjee *et al.* 2011] S. Banerjee, P. Choudekar, and M. K. Muju. Real time car parking system using image processing. In *2011 3rd International Conference on Electronics Computer Technology*, volume 2, pages 99–103, April 2011.

[Bhana 2017] Aarti Bhana. Hunting for parking brings no joy for students. *Wits Vuvuzela*, page 2, 2017.

[Bishop 2006] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Canny 1986] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[Canziani *et al.* 2016] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.

[Chatfield *et al.* 2014] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[Choeychuen 2013] K. Choeychuen. Automatic parking lot mapping for available parking space detection. In *2013 5th International Conference on Knowledge and Smart Technology (KST)*, pages 117–121, Jan 2013.

[Chollet 2015] François Chollet. *Keras*. https://github.com/fchollet/keras, 2015. (Accessed on 04/04/2017).

[computerstationco 2017] computerstationco. *OverHead sensors*. http://computerstationco.net/let/images/imagedata/pms_img3.jpg/, 2017. (Accessed on 04/03/2017).

[Cortes and Vapnik 1995] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[Dalal and Triggs 2005] 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)*, volume 1, pages 886–893 vol. 1, June 2005.

[Darwin 2015] Darwin. *The Area Under an ROC Curve*, 2015.

[Das *et al.* 2014] Deepjoy Das, Dr Saharia, et al. Implementation and performance evaluation of background subtraction algorithms. *arXiv preprint arXiv:1405.1815*, 2014.

[de Almeida *et al.* 2015] Paulo R.L. de Almeida, Luiz S. Oliveira, Alceu S. Britto Jr., Eunelson J. Silva Jr., and Alessandro L. Koerich. {PKLot} a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937 – 4949, 2015.

[DeepDetect 2015] DeepDetect. *Training an image classifier service*. http://www.deepdetect.com/tutorials/train-imagenet/, June 2015. (Accessed on 05/17/2016).

[del Postigo *et al.* 2015] C. G. del Postigo, J. Torres, and J. M. Menndez. Vacant parking area estimation through background subtraction and transience map analysis. *IET Intelligent Transport Systems*, 9(9):835–841, 2015.

[Deng *et al.* 2009] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[Deshpande 2017] Adit Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks*. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/, June 2017. (Accessed on 06/03/2017).

[DL4J 2017] DL4J. *Convolutional Networks*. https://deeplearning4j.org/convolutionalnets.html/, 2017. (Accessed on 03/29/2017).

[eucars 2017] eucars. *Ground sensors*. https://www.eurocarparks.com/custom/uploads/2009/12/sensor_parking.jpg/, 2017. (Accessed on 04/03/2017).

[Felzenszwalb and Huttenlocher 2004] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, Sep 2004.

[Girshick *et al.* 2014] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[Girshick *et al.* 2016] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):142–158, Jan 2016.

[Hahnloser *et al.* 2000] Richard H. R. Hahnloser, R. Sarpeshkar, Misha Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405 6789:947–51, 2000.

[Heikkila and Silven 1997] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1106–1112, Jun 1997.

[Hough 1962] Paul Hough. *Method and means for recognizing complex patterns*, December 18 1962. US Patent 3,069,654.

[Idris *et al.* 2009] MYI Idris, EM Tamil, NM Noor, Z Razak, and KW Fong. Parking guidance system utilizing wireless sensor network and ultrasonic sensor. *Information Technology Journal*, 8(2):138–146, 2009.

[Jia *et al.* 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[kdnuggets 2016] kdnuggets. *conv-net2.png (PNG Image, 2762 944 pixels).* https://flickrcode.files.wordpress.com/2014/10/conv-net2.png/, 2016. (Accessed on 03/04/2017).

[Knerr *et al.* 1990] S. Knerr, L. Personnaz, and G. Dreyfus. *Single-layer learning revisited: a stepwise procedure for building and training a neural network*, pages 41–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[LeCun *et al.* 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Lee *et al.* 2006] Yun-Seok Lee, Han-Suh Koo, and Chang-Sung Jeong. A straight line detection using principal component analysis. *Pattern Recognition Letters*, 27(14):1744 – 1754, 2006.

[Lee *et al.* 2008] S. Lee, D. Yoon, and A. Ghosh. Intelligent parking lot application using wireless sensor networks. In *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on*, pages 48–57, May 2008.

[Li 2017] Fuxin Li. *Convolutional Networks*. http://classes.engr.oregonstate.edu/eecs/winter2017/cs519-006/Slides/6_CNN.pdf/, 2017. (Accessed on 04/01/2017).

[Lowe 2004] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[McHugh 2012] Mary L. McHugh. Interrater reliability: the kappa statistic. In *Biochemia medica*, 2012.

[Mishkin *et al.* 2016] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the imagenet. *CoRR*, abs/1606.02228, 2016.

[Niu and Suen 2012] Xiao-Xiao Niu and Ching Y. Suen. A novel hybrid cnnsvm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318 – 1325, 2012.

[Nvidia 2016] Nvidia. *NVIDIA DIGITS*. https://developer.nvidia.com/digits/, 2016. (Accessed on 10/16/2016).

[Nyambal and Klein 2017] J. Nyambal and R. Klein. Automated parking space detection using convolutional neural networks. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, pages 1–6. IEEE, Nov 2017.

[OpenCV 2017] OpenCV. *Introduction to Support Vector Machines OpenCV 2.4.13.2 documentation*. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html, May 2017. (Accessed on 05/15/2017).

[Prince 2012] Simon J. D. Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.

[Reed *et al.* 2014] Scott E. Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *CoRR*, abs/1412.6596, 2014.

[Ren *et al.* 2015] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[ScikitLearn 2017] ScikitLearn. *SVM-Kernels*. http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html, May 2017. (Accessed on 05/21/2017).

[Shoup 2006] Donald C Shoup. Cruising for parking. *Transport Policy*, 13(6):479–486, 2006.

[Simonyan and Zisserman 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[Smith and Topin 2016] Leslie N. Smith and Nicholay Topin. Deep convolutional neural network design patterns. *CoRR*, abs/1611.00847, 2016.

[Supercircuits 2012] Supercircuits. *BLK-HDPTZ12 Security Camera Parkng Lot Surveillance Video*. https://www.youtube.com/watch?v=U7HRKjlXK-Y&index=3&list=LLFyrxWYsZ_k519ZXx-sGuqg&t=22s&spfreload=5/, 2012. (Accessed on 03/19/2017).

[Szarvas *et al.* 2005] M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata. Pedestrian detection with convolutional neural networks. In *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pages 224–229, June 2005.

[Tschentscher *et al.* 2015] M. Tschentscher, C. Koch, M. Knig, J. Salmen, and M. Schlipsing. Scalable real-time parking lot classification: An evaluation of image features and supervised learning algorithms. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015.

[TzuTaLin 2016] TzuTaLin. *tzutalin/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images*. https://github.com/tzutalin/labelImg/, 2016. (Accessed on 09/30/2016).

[Uijlings *et al.* 2013] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[Valipour *et al.* 2016] S. Valipour, M. Siam, E. Stroulia, and M. Jagersand. Parking-stall vacancy indicator system, based on deep convolutional neural networks. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 655–660, Dec 2016.

[Vondrick *et al.* 2013] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 101(1):184–204, 2013.

[Wu *et al.* 2007] Q. Wu, C. Huang, S. y. Wang, W. c. Chiu, and T. Chen. Robust parking space detection considering inter-space correlation. In *2007 IEEE International Conference on Multimedia and Expo*, pages 659–662, July 2007.

[www.westerncape.gov.za 2006] www.westerncape.gov.za. *SOCIO ECONOMIC PROFILE: CITY OF CAPE TOWN 2006*. https://www.westerncape.gov.za/text/2007/1/city_of_cape_town_se_profile_optimised.pdf/, 2006. (Accessed on 03/19/2017).

[Zeiler and Fergus 2013] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.