Universidade do Minho
Escola de Engenharia
Departamento de Informática

Helder Manuel Pereira Novais

Community Based Repository for Georeferenced Traffic Signs

February 2018

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Helder Manuel Pereira Novais

Community Based Repository for Georeferenced Traffic Signs

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
António José Borba Ramires Fernandes

February 2018

Anexo 3

DECLARAÇÃO

Nome

Hélder Manuel Pereira Novais

Endereço electrónico: helder_m_p_novais@hotmail.com Telefone: 916509034 / ____

Número do Bilhete de Identidade: 14383883

Título dissertação ☒/tese ☐

Community Based Respository for Georeferenced Traffic Signs

Orientador(es):

António José Borba Ramires Fernandes

Ano de conclusão: 2017

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado Integrado em Engenharia Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 24/10/2017

Assinatura: Helder Manuel Pereira Novais

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor António Ramires Fernandes for his patient guidance and valuable suggestions and critiques throughout the development of this dissertation.

I would like to thank Alexandra Ferreira who supported me and accompanied me closer during these difficult months.

I would also like to express my gratitude to all my friends, who heard my complaints when things did not go as planned and provided advice whenever they could.

Finally, I wish to thank my parents for their invaluable support, encouragement and faith during my studies.

ABSTRACT

In traffic environments, road signs have a key role to control, warn, and command or prohibit the driver of certain actions. Traffic sign maintenance is essential to prevent negative events. In order for these traffic signs to play the role they were designed for, periodic on-site inspections are essential and followed out to determine if signs are in good condition and visible, both during the day and night. However, periodic inspections are time and cost consuming.

Another issue is related to the drivers' awareness to the traffic signs on the road. Many factors, both internal and external to the driver, may potentially contribute to him missing a sign. Given the purpose of this dissertation, we will focus primarily on the external factors such as the sign being damaged or occluded, or distractions caused by the many gadgets inside the vehicle. Due to all these extraneous influences, a traffic sign recognition system may help the driver to respect these signs and increase significantly their safety, as well as the others around them.

Some high-end vehicles already have such a warning system, at least for danger signs. However, drivers with these vehicles represent a small fraction of the total driving force. This dissertation aims at bringing such a system to a much broader audience.

Smartphones are one of the most used devices by society today, mostly due to the many functionalities they provide in day to day life and their relative accessible monetary value. The increased computational power and cameras' quality improvement of these devices over the years make them good candidates to support the access to this kind of technology to all. In other words, smartphones of this day and age have the necessary resources to be used as instruments for sign recognition.

Hence, we propose a dual purpose community based approach. On the one hand, each driver can use his mobile device to detect, recognize and geolocate traffic signs, contributing to the traffic sign central repository. Detection is performed using Cascade Classifiers, while a Convolutional Neural Network supports the recognition phase. The repository, based on the information received from the clients, can be used to provide sign status reports and to enable more direct and timely inspection instead of relying on prescheduled global inspections. On the other hand, drivers would have access to the database of traffic signs, therefore being able to receive real-time notifications regarding traffic signs such as speed limit signs, school proximity, or road construction signs. Hence, allowing the system to perform its function even if the recognition phase is not active when used in a low computational power device.

RESUMO

Em ambientes rodoviários, os sinais de trânsito têm um papel fulcral para controlar, avisar e ordenar ou proibir o condutor de realizar certas ações. É essencial a manutenção dos sinais de trânsito para prevenir acontecimentos negativos. Para que os sinais de trânsit desempenhem a sua função, as inspeções rodoviárias periódicas são essenciais para determinar se os sinais estão em bom estado e visíveis, quer de dia, quer de noite. No entanto, as inspeções são bastante dispendiosas.

Outro problema está associado à consciência dos condutores em relação aos sinais de trânsito nas estradas. Muitos fatores, quer sejam eles internos ou externos, podem contribuir para um condutor não reparar num sinal, tal como a obstrução ou danificação do mesmo ou mesmo distração causado pelos *gadgets* dentro do veículo. Devido a todas estes influências, um sistema de reconhecimento de sinais de trânsito pode ajudar o condutor a respeitar estes sinais e aumentar significativamente a sua segurança e dos restantes em seu redor.

Alguns veículos de alta gama já possuem este tipo de sistemas de alerta, pelo menos para sinais de trânsito. Porém, esses veículos representam uma pequena fração da força motriz total. Esta dissertação visa levar esse sistema a um público muito mais amplo.

Os *smartphones* são um dos dispositivos mais usados pela sociedade nos dias de hoje, muito devido às funcionalidades que disponibilizam no dia-a-dia e seu valor monetário relativamente acessível. O aumento do poder computacional e melhoramento da qualidade da câmara destes dispositivos ao longo dos anos, fazem destes bons candidatos para suportar o acesso deste tipo de tecnologia a todos. Por outras palavras, os *smartphones* deste de hoje possuem os recursos necessários para serem usados como instrumentos para o reconhecimento de sinais.

Portanto, propomos uma abordagem baseada na comunidade com um duplo propósito. Por um lado, cada condutor pode usar o seu dispositivo móvel para detetar, reconhecer e georeferenciar sinais de trânsito, contribuindo para um repositório centralizado. A deteção é realizada através de *Cascade Classifiers*, enquanto que uma Rede Neuronal Convolucional trata da fase de reconhecimento. O repositório, baseado na informação recebida por parte dos clientes pode ser usada para fornecer relatórios acerca do estado dos sinais, de modo a possibilitar inspeções mais diretas e atempadas em vez de inspeções globais pré-agendadas. Por outro lado, os condutores teriam acesso a uma base de dados de sinais de trânsito e portanto, permitindo criar notificações em tempo real sobre sinais de trânsito tais como sinais de limite de velocidade, proximidade de escolas ou sinais de construção de estradas.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1 INTRODUCTION

Road traffic is an interaction between drivers, vehicles and pedestrians. One of the most important elements in road infrastructures are traffic signs. They control the traffic flow, inform about the directions and distances, warn about the road condition, prohibit/mandate the driver of doing certain actions and guide the right of way. Thus, it is really important to maintain them in good conditions, to improve the prevention of negative events.

One of the most efficient instruments to detect issues on road infrastructures is road inspection (Mccarthy et al., 2010). This assessment consists of periodical on-site evaluation carried out by trained safety expert teams and result in a formal report which is delivered to the relevant road authority. The most important part of the inspection relies on traffic control elements maintenance like traffic signs and road surface markings (Cardoso et al., 2007). It implies the comparison between the current road conditions against the stored reference state. Unfortunately, road safety inspections are very expensive both regarding the time and expertise required.

Several approaches for automating (at least partly) the inspection procedure have been developed ((S. R. Madeira and Santos, 2005) and (S. Maldonado-Bascon and Acevedo-Rodriguez, 2008)). The process is based on synchronizing the GPS coordinates with the image acquired by a camera used solely for this purpose. However, these automated procedures require to have a vehicle equipped with at least one of these cameras to capture the road images and a GPS receiver to georeference the road elements, like traffic signs, and inertial sensors or odometers to increase the system accuracy. To achieve a full inspection under these circumstances, it requires a large amount of time because it is performed by one car only.

Furthermore, depending on the interval at which the road inspections are carried out, a sign may find itself in poor visible conditions for a long period of time. Signs which are not clearly visible may be easily missed by the driver, potentially being a safety issue. Additionally, with all the gadgets in recent cars it is not hard to miss important signs. To improve safety it is important to increase the awareness of the driver.

Due to that purpose, many traffic signs detectors and classifiers systems have been developed in the last ten years (A. Broggi and Porta, 2007). Most traffic sign recogition (TSR) systems have a detection phase and a classification phase. There are several techniques that can be applied for traffic sign recognition: based on colour, shape, among others (Nguwi and Kouzani, 2006). The TSR systems have become a crucial part of advanced driver assistant systems (ADAS). Normally the main purposes are the recognition of speed limits and no overtaking signs. In this way, these systems can warn the driver if the top speed is being exceeded or if an overtaking is being done incorrectly due to a continuous line which forbids the car to cross it. However, not all vehicles have the possibility or capability to incorporate these kind of systems in their internal computational systems.

All these problems lead us to arrange a way to reconcile the peoples' safety and reducing the road inspection costs, making this assessment available to all.

Nowadays, mobile devices like smartphones or tablets are available to everyone since they have become increasingly powerful and monetarily accessible over the years. They have integrated various types of sensors such as accelerometer, gyroscope, GPS and Wi-Fi connection. It is important to emphasize that the quality of the cameras have been improving as well. This makes these devices very good candidates to achieve these goals.

1.1 OBJECTIVES

With this dissertation it is intended to explore the following topics:

- Reliably detect, recognize and georeference traffic signs;
- Be able to resolve recognition failures due to environment lighting variations and deformed signs;
- Use the community to contribute with georeferencing data to a central database;
- Use the data stored in the database to warn beforehand the driver of signs in his surrounding area;
- Evaluate the viability of this type of technology in mobile devices.

Furthermore we also could extend the framework to:

- Use the data stored in the database to elaborate reports for road traffic sign maintenance;
- Use the data stored to create a map of traffic signs using web mapping services (e.g. Google Maps).

The main goal is to create a community based framework to deal with both these problems which consists of a server side repository and a client mobile application.

On the one hand, we have a client mobile application that can recognize traffic signs and display warnings to the driver. Signs recognised by the clients' application are sent to a server, maintaining a central repository, which in turn delivers a partial copy of the repository signs to the clients.

The server, based on the reports received by the clients, can infer the signs' condition, i.e., when a sign is in good condition, or damaged and/or occluded. Therefore, it has the potential to avoid global traffic sign inspections and timely direct inspection teams to signs which are potentially damaged or occluded.

On the other hand, since clients receive the data of previously recognised signs from the repository, they do not rely only on the effective recognition of signs to be able to display warnings to the user. This makes this framework suitable to a wide range of devices. High-end devices, computationally capable of performing the recognition tasks, will be able to contribute to the central repository, while less performing devices can still benefit from the traffic sign repository for issuing alerts to the driver.

1.2 DISSERTATION STRUCTURE

This document will be divided in 5 chapters:

- Chapter 1 describes the motivation behind the choice of this theme and the objectives of this work;
- In Chapter 2 some reviews will be presented from previous studies related to detection and classification methods, artificial neural networks, server communication technologies and web mapping services;
- Chapter 3 provides an overview of the proposed client-server architecture;
- The implementation details of the proposed framework and tests describing the real usage of the client application, results from the sign recognition and detection phase, and georeferencing accuracy are reported in Chapter 4;
- Conclusions and what it is proposed to do as future work are shown in Chapter 5.

2

STATE OF THE ART AND RELATED TECHNOLOGIES

This chapter covers some of the technologies and methods that will be explored and implemented in Chapters 3 and 4.

2.1 COMPUTER VISION

Computer vision aims to build systems capable of performing some human visual system tasks or surpass them in same cases (Huang, 1996). Many vision tasks are related to the extration of 3D environment information from 2D devices, like video cameras, and then processing the information in order to detect or recognize a specific entity.

In the next sections some computer vision techniques will be presented that can be used to detect traffic signs.

2.1.1 *Colour-based Detection*

Colour-based detection is a technique widely used due to the fact that the colours of traffic signs are normally easy to distinguish from the surrounding environment ((Tran, 2013) and (Laguía, 2016)). Hence, the number of colours in use and their usage significantly influence the implementation and performance of the algorithm.

However, in captured images of real world scenes, it is not always possible to obtain required coloured regions by applying threshold directly to RGB colour space, due to external factors, like illumination conditions. To overcome the problem of colours presented in different modes, this factor should be separated from colour information. Hereupon, the best strategy is to work on different colour spaces. In De la Escalera et al. (2003), HSI (Hue, Saturation and Intensity) space was chosen for colour analysis. Hue is the angle, specified such that red is at zero, green at 120 degrees, and blue at 240 degrees. Hue thus represents what humans implicitly understand as colour. It is the particular wavelength frequency (HSI). On the other hand, saturation refers to the dominance of hue in the colour. In other words, saturation is the purity of the colour. Intensity is the brightness of light present. When light is at its fullest intensity, colours will become bright. To avoid lighting conditions problems, only Hue and Saturation components are used.

Other studies conducted a more exhaustive colour study. For example, Y. Aoyagi (1996) noted that colour sign segmentation cannot be done, because the Hue component changes with the deteorioration of the signs caused by conditions like distance, weather, and sign age. Concluding, the segmentation stage is not absolutely reliable in perfectly detecting the sign pixels.

There are other problems from colour-filtering such as noise that leads to many false positive pixels. These pixels could be further filtered out using a simple noise filter, but usually a more sophisticated technique involving shape detection is used to separate real signs from large chunks of false positive objects (Tran, 2013).

2.1.2 *Shape-based Detection*

Just like signs have specific colours, they also have very well defined shapes that can be searched for. The most common signs have the following shapes:

- circular for prohibitive or mandatory signs.
- triangular if the sign indicates danger; if the triangle is inverted is a yield sign.
- retangular are used for recommendation or information signs.
- the octagonal shape is only used for the stop sign.

Shape based methods ignore the colour in favor of the characteristic shape of signs. Shape detector developers argue that colour-detection methods are outdated since all colour detectors also use shape information for further filtering, and are unreliable due to changes in lighting and sign wear (Mogelmose et al., 2012).

Several approaches for shape-based detection of traffic signs have been developed, being the approximation of the Hough transform approach the most common (Brkic, 2010). Other approaches like Gavrila (1999) performs simple template matching from a database in a image. The Loy and Barnes (2004) method is similar to the Hough transform. It uses the symmetric nature of the shapes, together with the pattern of edge orientations exhibited by equiangular polygons with a known number of sides, to establish possible shape centroid locations in the image.

Although they appear to be a more robust way of detection, they suffer from object occlusion, rotation or distortion. Hence, the sign shape can look different and not all detectors can handle that.

2.1.3 *Feature-based Detection*

Colour and shape-based detection using heuristic algorithm are relatively cheap operations. However, if traffic signs are damaged, occluded by trees or bushes or poorly lit, detection may become an issue. To overcome this problem, a more generic sign detection based on features could be used. The feature concept is very general and the choice of feature in a particular computer vision system may be highly dependent on the specific problem at hand.

2.1.3.1 *Haar Features*

There are many methods for extracting the most significant features from images to perform traffic sign recognition. One of the earliest feature based detection methods was proposed by Paul Viola and Michael Jones (Viola and Jones, 2001) where they used Haar wavelets features. These features are similar to convolution kernels which are used to detect the presence of a specific feature in the input image such as edges, corners, line ends, or spots. Each feature consists of two or more rectangular regions enclosed in a template. A simple rectangular feature can be defined as the difference of the sum of pixels of areas inside the rectangle, which can be at any position and scale within the original image (Wikipedia, 2016a). The feature value f of a Haar feature which has k rectangles is obtained by Equation 1:

$$f = \sum_{i=1}^{k} w^{(i)}.\mu^{(i)} \tag{1}$$

where $\mu^{(i)}$ is the mean intensity of the pixels in a image enclosed by the ith rectangle (Pavani et al., 2010). Henceforth, we will refer to the quantity μ as the rectangle mean. In equation 1, $w^{(i)}$ is the weight assigned to the ith rectangle. The sum of all weights must be equal to zero. So the values assigned to the rectangles in Figure 1(a) must be -1 and 1. The main problem is the number of all possible parameters.

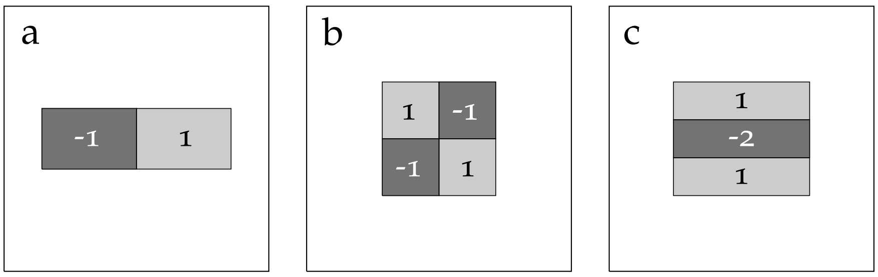


Figure 1.: Haar Features

There are only 2 two-rectangle features, 2 three-rectangle features (Figure 1(b)) and 1 four-rectangle feature (Figure 1(c)). The two-rectangle features can be define as a 1x2 pixel block or 2x1 if it is 90 degrees rotated. The same for the three-rectangle feature, but with a pixel block of 1x3 or 3x1. The four-rectangle feature has a pixel block of 2x2 so the result is the same even if rotated. So accounting to all positions, scales and types of features, we end up calculating over 162000 features for a 24x24 window. Although each feature can be computed quite efficiently, calculating the complete set is extremely expensive. Besides, most of the features are irrelevant to detect a certain object in the image. The best strategy is selecting the best features and this can be made with AdaBoost (Schapire, 2013).

For each feature, it finds the best threshold that will separate the positive or negative examples. Naturally there will be errors. So, the features that can detect more than half the cases are selected.

As simple as this can seem, it is not. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the process continues recursively until the required accuracy is achieved or required number of features are found (Open Source Computer Vision). The individual features are called weak classifiers. The linear combination of all of these classifiers constructs a strong classifier. According to the paper, the final setup had around 6000 features which is a huge reduction from the initial over 162000. However, applying 6000 features is still inefficient.

For this purpose, the Cascade of Classifiers was introduced in order to distribute all features in groups for different stages of classifiers (early stages have less features and are faster than subsequent ones). A window is discarded if it fails in any stage. It is considered an object region only if it passes in all stages.

2.1.3.2 *Local Binary Patterns Features*

Another type of feature is called Local Binary Patterns (LBP) (Huang et al., 2011) . Is a non-parametric operator which summarizes the local special structure of an image. To create LBP features first it is necessary to split the examined window into multiple cells(e.g 16x16 pixels for each cell). For each cell, binary comparisons of pixel intensities between the center pixel and its neighbors are used, from the top-left pixel in clockwise manner (Figure 2).

If a neighbor has a lower intensity than the center pixel it is converted to zero, otherwise to one. The output will be an eight-digit binary number that corresponds to a 256 decimal value. Then, a histogram is computed, over the cell, of the frequency of each value. Histograms may or may not be normalized. Finally, the histograms of all cells are concatenated and can be used as a feature vector for the entire image.

Also LBP features are integers in contrast to Haar features that are floats (OpenCV User Guide). Therefore, training and detection are several times faster in contrast to Haar fea-

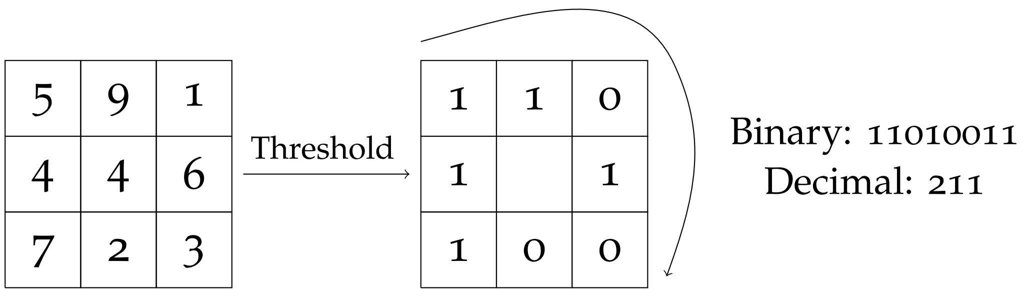


Figure 2.: Basic LBP Operator

tures. In terms of detection accuracy, it depends on training. It is possible to train a LBP-based classifier that will provide almost the same quality as Haar-based one.

One limitation of the basic LBP operator is that its small 3x3 neighborhood cannot capture dominant features with large scale structures. To overcome this limitation the LBP operator was later generalized to use neighborhoods of different sizes. The Multi-scale Block Local Binary Patterns (MB-LBP) (Liao et al., 2007) is a solution that uses sub-region average gray-values for comparison instead of individual pixels. The whole filter is composed by 9 blocks, where the size of the operator is a parameter. Figure 3 represents an operator of size 9x9. It has several advantages compared to LBP: it is more robust, provides a more complete image representation, by capturing the micro and macro information from a sub-region and it can be computed very efficiently using Integral Images (Viola and Jones, 2001).

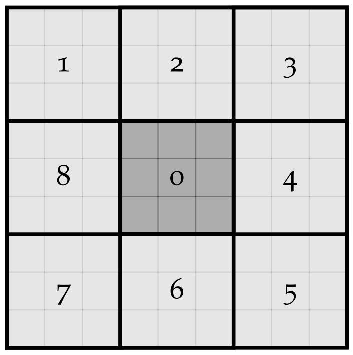


Figure 3.: MB-LBP Operator

Another extension to the original LBP uses so called uniform patterns. An LBP pattern is called uniform if it contains at most two bitwise transitions from 0 to 1 or vice versa when the corresponding bit string is considered circular. For example, 00000000 has no transition and 00011100 has two transitions. Ojala et al. (2002) noticed that in their experiments with texture images, uniform patterns account for around than 90% of all patterns when using a 3x3 neighborhood. Using uniform patterns, the length of the feature vector for a single cell reduces from 256 to 59, if we are working with the same neighborhood. This improves the overall performance. However, the basic definition of uniform LBP patterns cannot be used for MB-LBP. This is due to the fact that same properties of circular continuities in 3x3 patterns are not valid when the size of LBP block becomes larger. This leads to redundant information and must be eliminated in order to produce efficient classifiers. Once more the solution is using the AdaBoost algorithm to select the best features. Experiments show that MB-LBP significantly outperforms other LBP based method in terms of accuracy (Liao et al., 2007), being a more robust method, although it brings a little computational overhead over the original LBP operator.

2.2 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are processing devices (algorithms or actual hardware) modeled after the neuronal structure of the mammalian cerebral cortex, but on a much smaller scale. Many types of artificial neural networks exists today. They are configured for a specific intent, such as pattern recognition or data classification. So they are an alternative as well as a complement for image classification. Before discussing the basis of a neural network in detail, it is essential to understand some basic concepts that define and describe them.

2.2.1 *Artificial Neurons*

An artificial neuron works similary to a biological neuron. The neuron receives inputs from the other neurons (in case of the input neurons, the data comes from the environment).

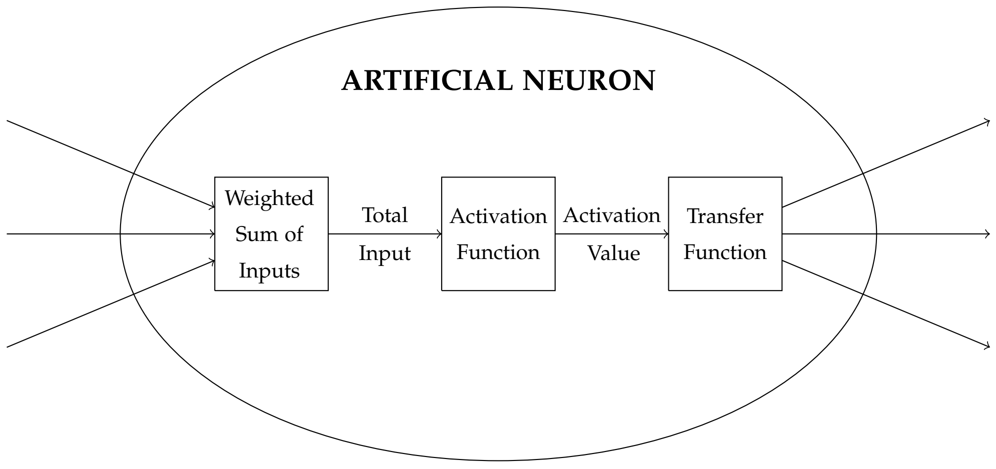


Figure 4.: Artificial Neuron Representation

Once in the neuron, the inputs are weighted and combined into a single value in the box labeled weighted sum of inputs (Figure 4). Normally the inputs are simply weighted and added together. The result is the total input, which is transformed by another function called the activation function.

The activation function dictates what the neuron is supposed to do with the signals after the weights have had their effect. Depending on the complexity of the models, the activation function may or not take into account the previous state in conjunction of the weighted sum of the neurons' input. In most artificial neural networks the activation function is deterministic, but may be stochastic in more complex networks (Russell, 1993). The activation value is then passed through the transfer function.

The transfer function defines how the activation value leaves the neuron and is distributed to the others (or to the outside environment for output neurons). In some cases, the transfer function works like a threshold value. If the activation value is greater than a given threshold, then the neuron will output a one. Otherwise, the resultant value will be zero.

Most artificial neural networks use the transfer function as a saturation function where a certain value reaches a maximum level has no further effect on the output of the neuron. The most used saturation functions are the sigmoid function and the hyperbolic tangent function. Both functions are continuous, and their values asymptotically approach a high and low value, with a smooth transition in between. The sigmoid transfer functions' output ranges from zero to one (Figure 5). The main difference between the sigmoid and the hyperbolic tangent function is that the second ranges from negative one to positive one. In fact, the sigmoid and hyperbolic tangent functions are linked by Equation 2:

$$\tanh(x) = 2.sigmoid(2x) - 1 \tag{2}$$

The sigmoid tranfer function is normally used in networks which purpose is classification, while the hyperbolic tangent function is used for prediction.

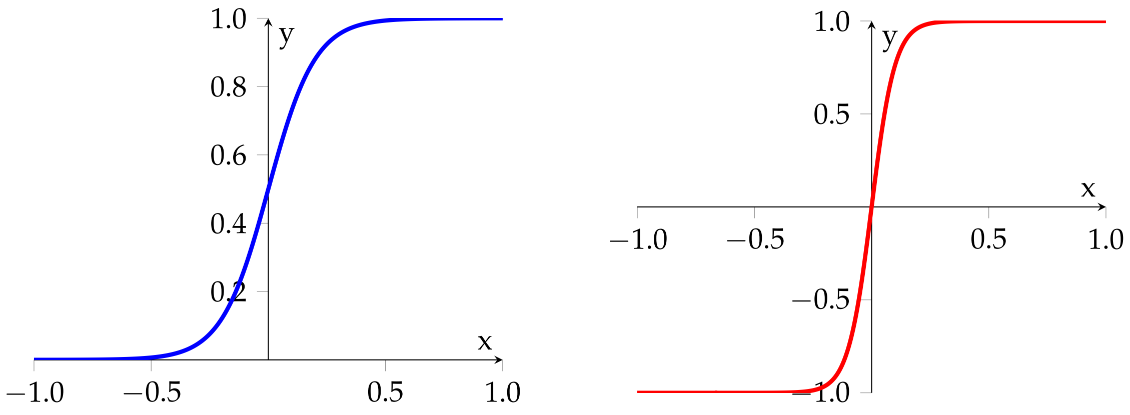


Figure 5.: Sigmoid (Left) and Hyperbolic Tangent (Right) functions

2.2.2 *Layers*

A neural network consists of groups of neurons arranged in defined structural units known as layers. A layer of neurons is a group of neurons that share a functional feature. There are three possible types of layers in a network, each one with a particular purpose. The first layer is called input layer. The neurons in this layer have the job of receiving data from the environment, like data files or any other transmitting devices. On the opposite side of the network the neurons send back the data transformed to the user in a form predefined by the setup of the network. This layer is the output layer. In between these two layers lies one or more hidden layers, depending on the architecture and complexity of the network as well as the computing power of the device. These layers are called hidden because the user can not see their inputs and outputs because they connect only to other neurons. The neural network can be fully connected, which means each hidden layer and each output layer is connected to every unit in the layers either side.

2.2.3 *Network Architecture*

There are different types of neural networks but they are generally classified in two types of architectures: feed-forward or feedback.

In feed-forward networks, the neurons do not communicate between them if they reside on the same layer. This is an advantage because the networks can be computed much faster because there are no delays due to the interaction between the neurons until they achieve a steady state (Beale and Jackson, 1990). Also, the information flows only in one direction.

Feedback networks on the contrary can have signals travelling in both directions using loops. The neurons are allowed to communicate with any other neuron, including themselves. The input data is continuously computed until the neurons settle into a state of equilibrium. Feedback networks are good at reconstructing facts from incomplete and error filled inputs (Russell, 1993).

2.2.3.1 *Backpropagation*

The term backpropagation refers to a type of learning algorithm for adjusting the weights in a multiple layer feed-forward network (Russell, 1993). Is a training method for artificial neural networks proposed by Rumelhart and Hinton (1986) to solve the credit-assignment problem.

Once the input has been propagated across the entire network to the output layer, the output is then compared to the desired output, using a loss function which quantifies the inaccuracy of predictions, and an error value is calculated for each of the neurons in the output layer by subtracting the output value for the targeted one.

The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which represents its contribution to the original output (Wikipedia, 2016d). These errors are then used to update the weights in order to minimize the loss function.

The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that different neurons learn to recognize different characteristics of the total input space. After training, when an input pattern is incomplete, neurons in hidden layers of the network will determine if the input contains a pattern that is similar to a feature that each neuron have learned to recognize during the training.

2.2.4 *Convolutional Neural Networks*

While classical artificial neural networks perform well for simple classification problems, they have several drawbacks when it comes to real-world image based applications (Bouchain, 2006). This is due to the full connectivity between the neurons. For example, for a 32x32x3 (32 for width, 32 for height and 3 for the RGB colour channels) image, it generates 3072 connections per single neuron in a first hidden layer of the network. This does not scale well with higher resolution images. In this kind of network architecture, the topology of the input data is not taken in account treating all the pixels on the exactly same way. Therefore, full connectivity of neurons is not a good option for image recognition, as the huge number of parameters quickly leads to overfitting. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data (Wikipedia, 2016b).

A Convolutional Neural Network is a type of feed-forward neural network in which the connection between the neurons is inspired by the organization of the animal visual cortex, particularly from cats (Hubel and Wiesel, 1959). The animal visual cortex is the most powerful visual processing system in existence, so it is natural trying to recreate its behavior (LeNet). This model solves the problem of the classical neural networks by exploiting the strong spatially local correlation present in natural images (Wikipedia, 2016c). These networks take advantage of the fact that input consists of images. So, unlike ordinary neural networks, the neurons are arranged in three dimensions: height, width and depth (the depth refers to the third dimension of the activation volume, not the depth of a full neural network, which indicates the number of layers in a network) (Li and Karpathy, 2015). There are many models that can be found in the literature, namely the Neocognitron (Fukushima, 1980), HMAX (Serre et al., 2007) and LeNet-5 (LeCun et al., 1998).

The architecture of a Convolutional Neural Network are commonly formed by three layer types: convolutional layer, pooling layer and fully-connected layer (Li and Karpathy, 2015).

2.2.4.1 *Convolutional Layer*

The convolutional layer is the core of a Convolutional Neural Network that does most of the computational work.

The layers' parameters consist of a set of learnable filters (similar to kernels) which have a small spatiality called receptive field (Figure 6), but extend through the full depth of the input volume. The receptive field of the neuron can be seen as the filter size. For example, a simple filter may have a size of 5x5x3, which represents the height, width and depth, respectively. If the input volume size is 32x32x3 then each neuron in the convolutional layer will have 5*5*3 = 75 weights. So, we can confirm that the connectivity is local in space (5x5), but full along the input depth(3).

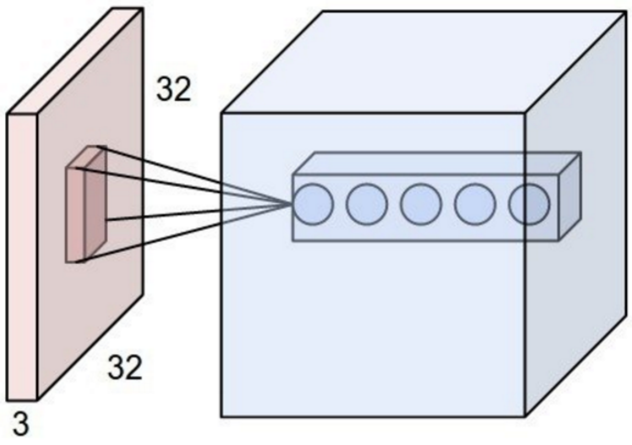


Figure 6.: Receptive Field. *Source:* Li and Karpathy (2015)

During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input, producing a 2-dimensional activation map that gives the responses of that filter at every spatial position. As a result, the network will learn filters that activate when they detect some type of visual feature such an edge. The set of activation maps is stacked along the depth dimension and produce the output volume.

The output volume size is controlled by three parameters: the depth, stride and zero-padding.

1. The depth corresponds to the number of filters we would like to use, each one looking for different features in the input. The set of neurons that are searching in the same region is referred as depth column.

2. The stride controls how the depth columns around the spatial dimensions(width and height) are allocated. If the stride is one then every pixel will have a depth column. This leads to overlapping receptive fields and large output volumes. Hence, the larger the stride, the smaller the output volume will be and the receptive fields will overlap less.

3. The last parameter zero-padding controls the output volume spatial size by padding the borders of the input volume with zeros. In some cases is preferred to preserve the spatial size of the input volume so the input and output width and height are the same.

Another property in convolutional layers is the parameter sharing scheme that controls the number of parameters. It relies on a reasonable assumption: if one feature is useful to compute at some spatial position, then it should be useful to compute at a different position. In other words, denoting a single 2-dimensional slice of depth as a depth slice, we constrain the neurons in each depth slice to use the same weights. In the case of a volume of size 32x32x96 it has 96 depth slices, each one of size 32x32. With a convolutional layer of 5x5x3 we have a total of 96*5*5*3 = 7200 weights instead of calculating 32*32*96*5*5*3 that generates more than 7 million weights. Therefore, as all neurons in a single depth slice are using the same weight vector, then the forward pass can be computed as a convolution of the neurons' weights with the input volume. This is why it is common to refer to the sets of weights as a filter(or a kernel), that is convolved with the input (Li and Karpathy, 2015).

2.2.4.2 *Pooling Layer*

Another important concept is pooling which performs a form of non-linear down-sampling. There are two ways to do pooling: max pooling and average pooling (Hijazi et al., 2015), max pooling being the most common. This layer is normally inserted in-between successive convolutional layers. It divides each depth slice into a set of rectangles and for each sub-region discovers the maximum value. The most usual form is a pooling layer with filters of size 2x2 and a stride of 2 applied to each slice, thus discarding 75% of the activations. (Figure 7). Hence, reducing the spatial size of the input, the computational work is also reduced, controlling the overfitting as well. Although these changes, the depth dimension remains unaffected.

However, more recent studies disagree about the use of pooling layers. Instead they use repeated convolutional layers only (Springenberg et al., 2014). To reduce the size of the representation they suggest using larger stride in the convolutional layer.

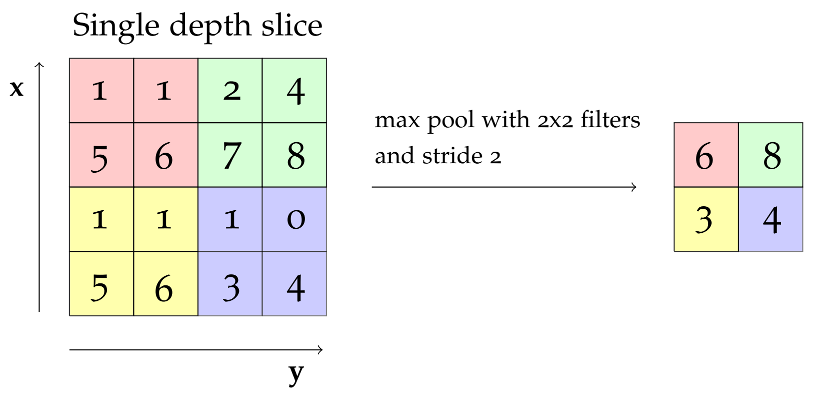


Figure 7.: Max-polling with a 2x2 filter

2.2.4.3 *Fully-connected Layer*

The fully-connected layer, after several convolutional and max-pooling layers, is responsible for the high-level reasoning in the neural network. It is often used as the last layer in a Convolutional Neural Network. The neurons in the this layer have full connection to all activations in the previous layer, like in regular neural networks.

2.3 CLIENT-SERVER COMMUNICATION TECHNOLOGIES

In the client-server computing paradigm, one or more clients and one or more servers, along with the underlying operating system and communication systems, form a composite system allowing distributed computation, analysis, and presentation (Sinha, 1992).

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems (Leach et al., 1999). The HTTP protocol is a request/response protocol based on the client/server where entities act like HTTP clients, and the Web server acts as a server.

The HTTP client sends a request to the server in the form of a request method and The HTTP server responds with a status line, including a success or error code and possible some body content.

There are abstraction architectures of this protocol in order to facilitate its use.

2.3.1 *Representational State Transfer (REST)*

Representational State Transfer (REST) is a software architectural style for designing stateless network applications which usually runs over standardized HTTP protocol. REST is a lightweight alternative of Remote Procedure Calls (RPC) and Web Services such as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) standards.

REST was originally introduced and defined in Roy Fielding PhD thesis in 2000 at UC Irvine (Fielding, 2000).

Most of the modern web-services provide REST API for HTTP-based communication. Another alternative would be to use SOAP technology for receiving data from the smartphone application or vice-versa.

SOAP supports additional security protocols that REST does not. This is required for banking transactions or any similar operation where security failures could be catastrophic (RVS). However, in most cases this kind of security is not the main concern.

Developers prefer to use more efficient and easier communication protocols to their applications. And this is where REST wins over SOAP. The REST APIs and the documentation are also much simpler to understand. Another disadvantage of SOAP is that only permits XML as data format while REST handles multiple formats, being JavaScript Object Notation (JSON) usually the best since its parsing is much faster than XML. Additionally, researches claim that REST-based implementation is proved to be more efficient in network bandwidth and latency than SOAP-based services (Mulligan et al., 2009). This is due to the fact that REST reads can be cached, while SOAP reads are not. This provides a better performance and scalability.

The following are the key principles of the REST architecture (BIR):

- Every resource is identified by an URI (Uniform Resource Identifier). URIs are a unified concept for IDs.
- Link things together. The links are used to refer to identifiable resources wherever possible. A client should be able to move the application from one state to the next by following a link.
- Use standard methods. For clients to be able to interact with your resources, they should implement default application protocol (HTTP) correctly, by making use of the standard methods GET, PUT, POST, DELETE.
- Resources with multiple representations. The representation of the resource can be transferred between the server and the client in multiple formats. For example, the client could retrieve data in text format XML, JSON or any other format that both parties agree upon.
- REST has a stateless communication. Since the server state is only resource-related, it should not have to store any sort of communication state for any of the clients. This improves the scalability because the server can quickly free the resources.

The CRUD operations (Create, Read, Update, Delete) are supported for the resources and are equivalent to standard HTTP methods described in Table 1. Most operations are

HTTP method	CRUD	Safe	Idempotent
GET	read	yes	yes
POST	create	no	no
PUT	update/create	no	yes
DELETE	delete	no	yes

Table 1.: HTTP methods in a typical RESTful API

idempotent which means that if the same operation is executed several times, it results in the same state. The POST method however in not idempotent. It is usually used for creating new identities on a certain resource. The DELETE method is classified as idempotent according to the HTTP specification but when the service actually deletes the resource, the next call will not find the resource to delete it and return a 404 (Not Found).

Concluding, the goal of the REST architecture is to provide a more scalable system in order to support a great number of components and simultaneous interactions with clients. REST also provides general interface, therefore, it supports interoperability. In addition to these, the architecture supports independent deployment of components as well as intermediary components to reduce latency, enforce security and encapsulate legacy systems (Peter Somogyi and Szalay-Beko, 2014).

2.4 WEB MAPPING SYSTEMS

Web Mapping is the process of using maps obtained by spatial and geographical data from a geographical information system (GIS) database. Web mapping is much more than just a web cartography, it is seen as both service and consumer activity.

According to Kraak (2001), web maps can function as an interface or index to additional information in a way that facilitates an up-to-date, dynamic, and interactive presentation of geodata to many more users at a minimal cost.

Many Web mapping systems (WMS) have been developed and continually upgraded and have become standard ways of sharing geographic information on the Web. Most WMS provide APIs to display maps in which users have the ability to zoom and pan interactively. These APIs also provide functionality to add layers of geographic information to maps. Two of these systems will be described in the following sections.

2.4.1 *Google Maps*

Google maps is a web mapping service developed by Google. It offers satellite imagery, street maps, real-time traffic conditions and route planning for travelling by foot, car or bicycle (Wikipedia, 2016e). Google Maps provides a top-down view where the images are taken from satellites or from aircrafts to obtain high-resolution imagery if the cities. Google Street View provides panoramic views of stitched images from positions along many streets in the world.

Google Maps has an API, which is a free service, to allow developers to integrate Google Maps into their websites. According to Built With, there are over four million websites using the Google Maps API, making it the most used web based application development API.

2.4.1.1 *Google Map Marker*

Google Map Marker is a map editing service with the purpose of making possible the contribution of any user by fixing incorrect driving directions, adding biking trails, or adding a missing building or road, in order to increase the quality of the maps in regions where it is difficult to extract the geographic information. Unfortunately, in Portugal and Spain the service in unavailable. In November 2016, it was announced that Google Map Maker will be retired in March 2017 and merged with Google Maps itself (Google Map Maker).

2.4.2 *OpenStreetMap*

Similarly to Google Map Maker, another free collaborative tool to edit maps is called OpenStreetMap. When contributing to OpenStreetMap, the server renders map tiles on-the-fly, allowing to see the changes within minutes. This technical property is not available when contributing to Google Maps.

Google Maps also restricts the access of raw map data, in order to maintain a commercial advantage to expose downstream products and services generated from the raw map data (OpenStreetMap), while OpenStreetMap releases maps in their most raw form for free. This is hugely powerful since it offers to the developers a wide range of services and experiments which would simply not be possible without access to raw map data.

One important and useful feature is filtering map parts to extract specific information. This can be done with the Overpass API that receives a user query and gives back the dataset to the correspondent query. It is highly optimized for retrieving large quantities of data in a few minutes. This is an interesting tool to analyse only a portion of a map, done in a way that Google never will.

3

GLOBAL FRAMEWORK ARCHITECTURE

In the last years many solutions to increase road safety have been developed. High-end car brands have already integrated traffic sign recognition systems in their advanced driver assistant system with that purpose. However, they are too basic, being restricted normally to detect and recognize speed limit signs only, and most of the population does not have the possibility to afford these kind of vehicles. Another issue in these systems is the wrong sign classification resulting in an incorrect alert.

On the other hand we have the road inspection procedure, which consists on periodical on-site evaluation carried out by trained safety expert teams, resulting in a formal report which is delivered to the relevant road authority to detect issues on road infrastructures. The most important part of the inspection relies on traffic control elements maintenance like traffic signs and road surface markings. Unfortunately, the road safety inspection is very expensive and requires much time in terms of expertise. Furthermore, as mentioned before, road inspection schedule may create situations where a damaged sign is not fixed for a long period of time.

Thus, all these problems leads us to find a solution affordable to all, reconciling the people safety, by increasing the reliability of the algorithm, reducing the road inspection costs, and allow for more timely and targeted inspections.

3.1 PROPOSED APPROACH - SOLUTION

To achieve the above goals in an efficient and rapid manner this dissertation tries to answer two complementary problems:

- building and maintaining a traffic sign repository through community based contributions;
- improve the driver safety by notifying the presence of some relevant signs.

The proposed approach to the first problem is to build a smartphone application that uses the device camera to capture frames of the street road ahead, and then processes these frames to detect if and which traffic signs are present in the image. Based on the GPS location of the car, the orientation of the camera, and the dimensions of the traffic sign on the image, the sign location can be computed. Once a traffic sign is recognised, it is sent to a repository together with its georeference. This way, it is possible to build a repository over time. Later, this information can be used to elaborate reports for road traffic sign maintenance and generate a global map of all georeferenced signs. The whole process is illustrated in Figure 8.

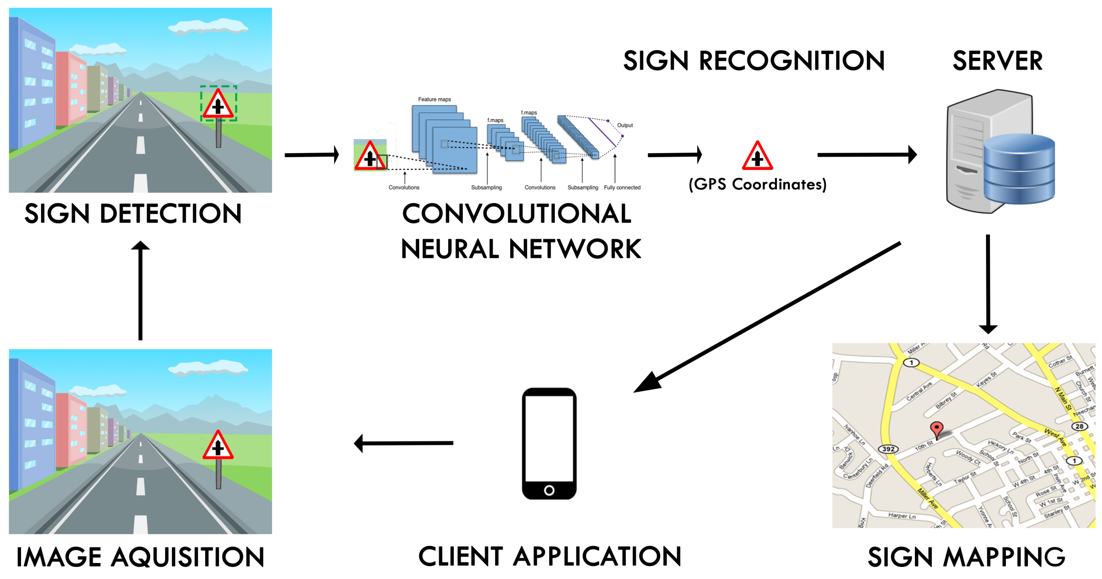


Figure 8.: System Architecture Scheme

The community based approach, if adopted by a significant number of drivers, has several advantages when compared to an approach where there is a single unit collecting data, or at least there is only a single unit assigned to an area.

For instance, temporary signs will enter the repository when the client application starts reporting them. Once a period of time has passed without any report they will be removed from the repository. Damaged signs will likely have lower recognition rates when compared to well maintained signs, hence the recognition rate can be a hint that a sign requires maintenance. This information can be used by road inspection teams to provided timely and direct intervention.

Last but not least, computer vision techniques are not 100% accurate, and if a single unit is doing a single passage over a street there is a possibility that some signs will not be recognised. While the same is true for our approach, even if a number of client applications fail to recognize a sign, due to lighting conditions for example or excessive speed for the system to work properly, other drivers will pass through the same signs sooner or later and eventually the sign will be added to the repository.

The second problem, improve the driver safety by notifying the presence of some relevant signs, uses both the data collected on real time and the data from the repository. The real time data will be useful for areas where there is no data yet on the repository and for recent temporary signs. On the other hand the real time data will be confronted against the repository data. This has several benefits:

1. if a sign is recognised and it is not in the repository then it is added to the repository;
2. if a sign is in the repository and its confidence level is lower than a defined threshold value then the sign is probably requiring maintenance;
3. if a sign is in the repository and it is not recognised anymore by any client application, then the sign has either been removed, or it has completely occluded (for instance due to tree foliage appearing in the Spring), or it is requiring urgent maintenance;
4. provides a constant flow of data regarding the detection/recognition algorithms accuracy which can be used to improve them.

Having traffic sign information available will allow to alert the driver, for instance in the case of speed limits violation (using the phones sensors).

Potentially, this system could be extended to detect road pavement issues, and report the need to repaint the lane guides. It can also incorporate other information such as fixed speed radar locations, and other information provided by the traffic police itself.

3.2 CLIENT SIDE

The client is responsible for the collection of traffic sign information which is then displayed and for setting off warning alerts when certain conditions are met. The main advantage of using this framework, from a user point of view, is the ability to have an application that can warn the driver of specific situations and help him anticipate adverse events.

When the application starts, it verifies if the car is in motion or not. If it is moving, it is constantly performing two tasks: identifying traffic signs revealed in the frames captured by the devices' camera and checking the local repository to see if it contains signs that match the vehicle motion direction and location. Otherwise, when not in motion, the system stays idle, saving resources.

The client application architecture has five main modules (Figure 9): the detection module, the classification module, the georeferencing module, the client-server communication module and the interface module. These modules will all be explained and described in detail in the following sections of this chapter.

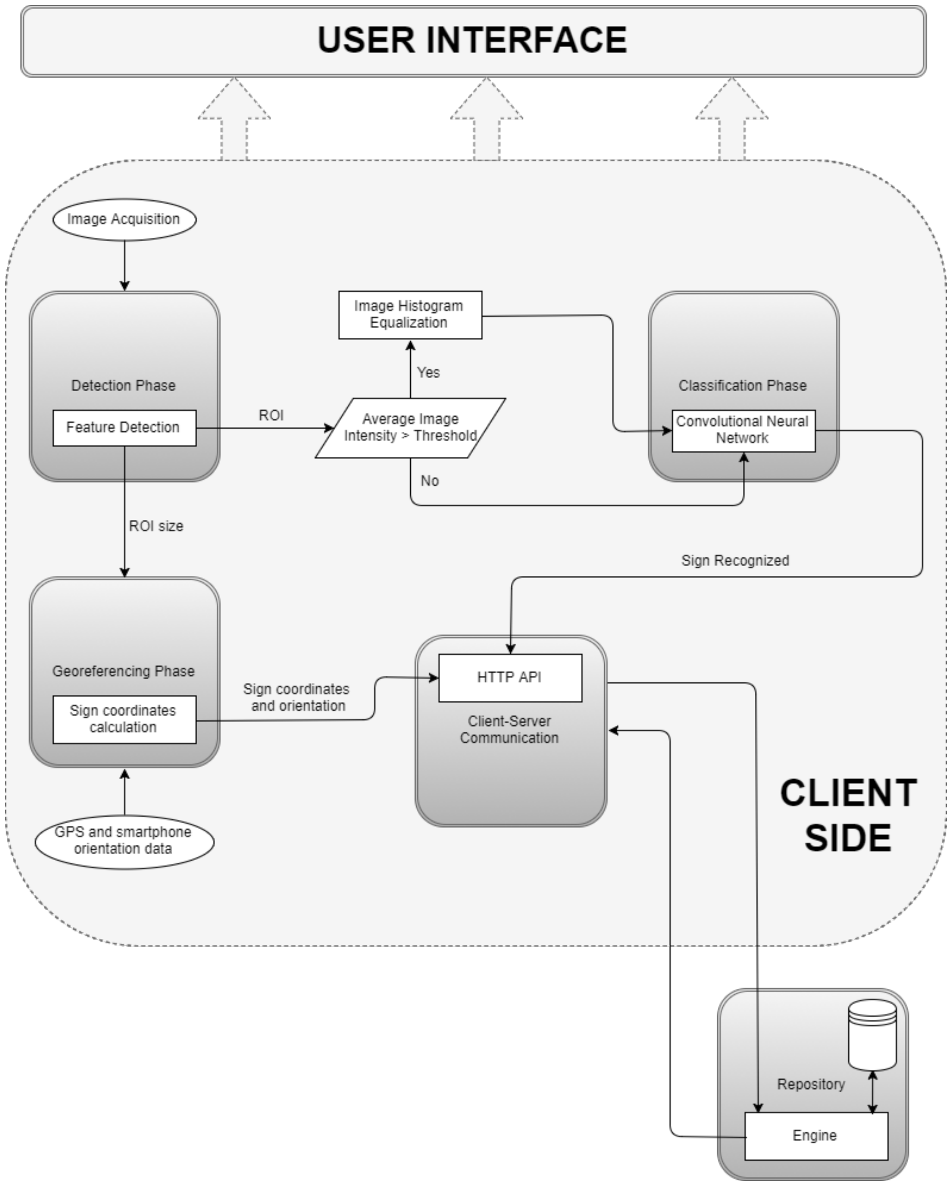

USER INTERFACE
Image Acquisition
Detection Phase
Feature Detection
ROI
Image Histogram Equalization
Yes
Average Image Intensity > Threshold
No
Classification Phase
Convolutional Neural Network
ROI size
Sign Recognized
Georeferencing Phase
Sign coordinates calculation
Sign coordinates and orientation
HTTP API
Client-Server Communication
GPS and smartphone orientation data
CLIENT SIDE
Repository
Engine

Figure 9.: Client side architecture

3.2.1 *Detection Module*

In Section 2.1, it was referred that colour and shape detection methods were not very reliable since discolouration, occlusion or vandalism could adversely affect the detection phase by creating many false positives or negatives.

Therefore, in this work, detection is performed based on features. The application utilises a feature of OpenCV library called OpenCV Training Cascade. The version of OpenCV used was 2.4.9 and provides AdaBoost training using Haar, LBP or HOG (Histograms of

Oriented Gradients) features. Due to the low computational power of the mobile devices, LBP was selected because it only uses binary comparisons between neighbouring pixels. Both training and detection with LBP are several times faster than to the other two methods. According to OpenCV User Guide, OpenCV uses the Multi-scale Block Local Binary Patterns (MB-LBP).

If a Cascade Classifier detects any object that resembles a sign in the full frame, it will create a polygon (normally a rectangle) around the region of interest (ROI). The colour of the detected region may vary a lot due to different illumination conditions. So before sending the ROI to the classification module, if the region is too dark or too bright, based on a threshold value, a preprocessing is performed: the ROI undergoes an histogram equalisation , adjusting the intensity of the image pixels. If the average pixel intensity of the ROI is in between the given thresholds, the raw image is sent directly to the classification module, without any preprocessing.

3.2.1.1 *Training*

To train a Cascade Classifier with AdaBoost, it is needed to provide one set of positive and one set of negative samples. The positive samples contains only objects to be detect, in this case traffic signs. Negatives samples, in the other hand, should not contain any instances of the object. They are used as background images.

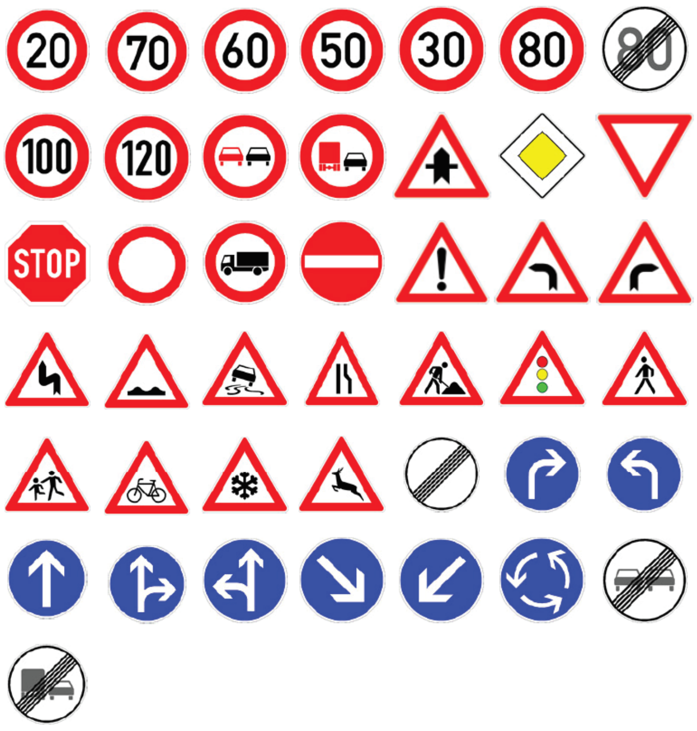


Figure 10.: German Traffic Sign Recognition Benchmark dataset. *Source:* (Hijazi et al., 2015)

The German Traffic Sign Recognition Benchmark (GTSRB) (Hijazi et al., 2015) dataset (Figure 10) provides 39,209 training images for 43 distinct classes of signs. Image resolution in pixels ranges from 15x15 to 222x193. Each class contains a considerable number of traffic sign images with different levels of brightness, with random objects occluding the sign, different perspective angles, blur and noise. Therefore, it is a good source to create the positive samples, improving the detection in uncontrolled environments.

The OpenCV funtion to train the Cascade Classifier is `opencv_traincascade`. This function uses only grayscale images as input while training. If the positive or negative samples are in RGB, OpenCV converts them automatically to grayscale. So, the colour of each sign template can not be used to classify it specifically. So, the features are used to distinguish signs from non-signs through shape and their particular properties. Therefore, the classifier can be seen as a robust sign-shape detector. This may lead to problems like detecting objects that have the same shape and are not signs.

3.2.2 *Classification Module*

Due to fact that Cascade Classifiers only accept grayscale images, the colour properties necessary for specific identification of the sign are lost and we only can determinate the type of sign in question (i.e triangular, circular, or octagonal). That leads us to search for a solution to the problem.

As refered in Section 2.2, neural networks, in particular Convolutional Neural Networks, are quite good when it comes to image recognition, due to its architecture. Therefore, a modular deep learning library, called TFLearn was used. It is built on top of TensorFlow, an open-source software library for building and training neural networks to detect and decipher patterns and correlations. It provides a higher-level API to facilitate and speed-up the experiments. Thus, besides the higher abstraction level, the main reason for choosing this library is the possibility of taking advantages of all TensorFlows' available features due to its full compatibility with it.

Similarly to Cascade Classifiers, the neural network requires training before being usable. The GTSRB dataset was the same used for training the network, coinciding with the number of different classes trained corresponds to the number of the possible signs. There is also a test dataset with 12630 images, to verify the neural network' accuracy. The ROI detected in the detection module is scaled to a 32x32 pixel resolution before entering as input to the CNN in the classification phase, which handles the sign recognition. The result should be one of the 43 different output possibilities.

3.2.2.1 *Network Arquitecture*

Initially the network construction followed a rather simple topology, with only one convolutional layer with 32 3x3 filters, followed by a pooling layer and finally a fully connected layer.

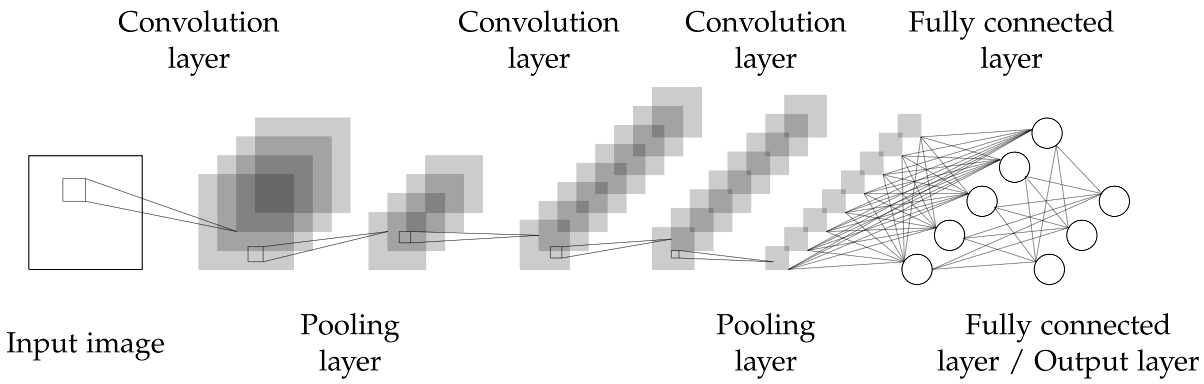


Figure 11.: Network architecture

Later, the network layers were increased having three convolutional layers, one with 32 3x3 filters and two stacked with 64 3x3 filters. In between them there are two max-pooling layers before reaching the last fully connected layer (Figure 11). Another topology created is similar to the previous one described, being the main difference the extra max-pooling layer between the last two convolutional layers (instead of being stacked). Regarding this topology, two different filter sizes were used (a 3x3 and a 5x5 filter, respectively) to verify the impact of this parameter on the accuracy.

These three networks where tested to verify if there is a large discrepancy in the results and choose the best network in terms of accuracy with the lowest complexity possible. The selected neural network was the second described. The set of tests and results to support this selection can be consulted in Section 4.2.1.

3.2.3 *Georeferencing Module*

Since the smartphone is inside the vehicle, when the sign is detected and recognised, his GPS position will be the vehicles' position. To provide the highest accurate location Fused Location Provider API was used. It analyses GPS, mobile and Wi-Fi network location data. Also, this provider uses different device sensors to define if a user is walking, riding a bicycle, driving a car or just standing in order to adjust the frequency of location updates. So it does not try to update your location as frequently if the accelerometer indicates that you are not moving, which has the side benefit of saving battery.

As stated in Krsák and Toth (2011), to determine and to calculate the approximate position of the recognised sign we can apply the Equations 3 and 4:

$$sx = x + p.cx.\cos\beta - cy.\sin\beta \tag{3}$$

$$sy = y + p.cx.\sin\beta + cy.\cos\beta \tag{4}$$

where sx and sy represents the calculated longitude and latitude of the recognised traffic sign. The values of variables x, y are obtained from a GPS device and represent the current longitude, latitude, respectively. p symbolizes the position of the sign in the full frame captured by the smartphones' camera. If the sign is located on the left half of the image, then the p value is +1, else is -1.

The c_x and c_y values are used to compute the shift of position in longitude and latitude. The c_x refers to the distance from the car to the side of the road and has been set at 0.00005 in decimal degrees based on experimental results. The c_y is obtained based on the sign size in pixels, calculating the distance of the camera to the plane, perpendicular to the device view direction, where the sign can be found.

To determine the distance c_y from the camera to the sign, required for Equations 3 and 4, we can use the distance Equation 5,

$$c_y = (W * F)/P \tag{5}$$

where P is the ROI width in pixels sent from the detection module, W is the real width of the sign and F is the focal length. The focal length calculation is shown in Section 4.1.3. This is an approximation of the distance from the device to the plane, perpendicular to the view direction, where the sign can be found.

Dimensions(cm)	Small	Normal	Large
Width	60.0	70.0 90.0	115.0

Table 2.: Portuguese sign dimensions

The IMT reports the size of the Portuguese traffic signs as presented in Table 2. The specific size used depends on the classification of the road, with 70 centimeters being used for urban roads. At this stage we are focused only on urban areas so we'll assume all signs have this dimension.

After the distance is determined, we need to convert to decimal degrees to use in Equations 3 and 4. Hence, we have to convert this distance in meters to both a shift in latitude and longitude. In the equator both latitude and longitude represent approximately the same shift, so we convert our distances to degrees considering this referential, i.e., we divide both c_x and c_y by 111319.44, the distance corresponding to one degree at the equator.

Then we apply a longitude correction factor to Equation 3 using the inverse cosine of the latitude, as in Equation 6.

$$s_x = x + \frac{p.c_x.\sin\beta - c_y.\cos\beta}{cos(y)} \tag{6}$$

The β value represents the azimuth which indicates the angle in radians between the devices' current compass direction and magnetic north. His value can thus range from 0 radians, if the top edge of the device faces magnetic north, and if the top edge faces south, the azimuth is π radians. Similarly, if the top edge faces east, the azimuth is $\pi/2$ radians and if the top edge faces west, the azimuth is $3\pi/2$ radians.

3.2.4 *Client-Server Communication Module*

All the data collected by the previous modules need to be sent to the repository. In Section 2.3, are presented some protocols to send or receiving data via HTTP-based communication, namely the REST API.

Retrofit is a type-safe REST client for Android (or just Java) developed by Square. It provides a powerful framework for authenticating and interacting with APIs and sending network requests with OkHttp. The library fetches JSON or XML data from the RESTful web service and once the response is received, it will be parsed as a Plain Old Java Object (POJO), which should be specified for the object in the response. Custom JSON parsers and the Gson utility library are supported for deserialization and parsing. Retrofit works with REST API using the Java interface implementation.

In addition of sending information, this module can receive data from the server too, in order to be able to warn the driver beforehand of signs in the surrounding area.

3.2.5 *Interface Module*

To enable and facilitate the use of the framework by the client, a graphical user interface was built. It was designed to display all the important information in the main screen, increasing the drivers' awareness. The application was called CADAS (Community Advanced Driver Assistance System) and it has one button in the main menu (Figure 12).

Before starting the application, the smartphone must be placed in a landscape position, with the camera facing the road and the device screen to the driver. When clicked, it will initialise all four the modules described before, downloading also a partial copy of the repository and stored all information in the device for further usage and verify if the vehicle is in motion, displaying the current car speed in the top part of the screen (Figure 13). If the car is stopped, then the application stays in a dormant state, without processing any of the frames.

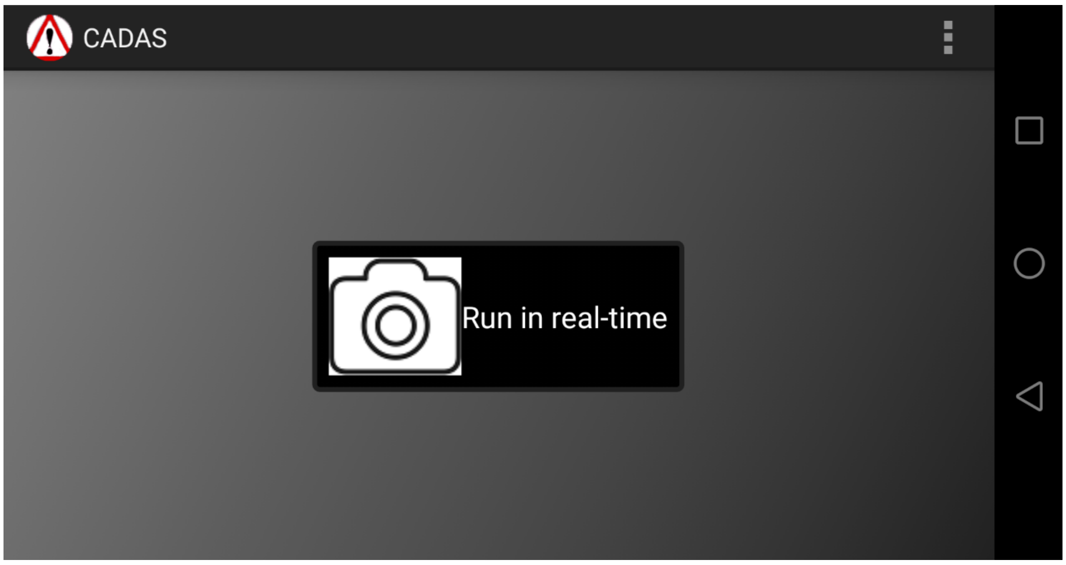


Figure 12.: Application Main Menu

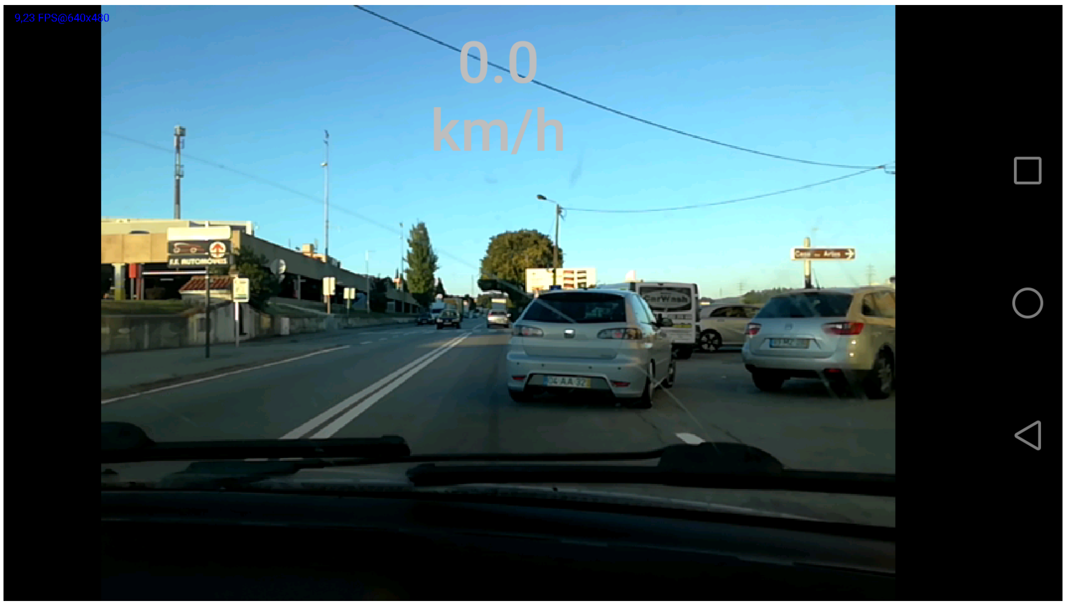


Figure 13.: Stopped car

As soon as the car starts moving, the speedometer starts to show the car' speed and the frames captured are filtered in order to detect the presence of traffic signs in the road ahead. Figure 14 shows that the vehicle is at a speed of 47 kilometers per hour.

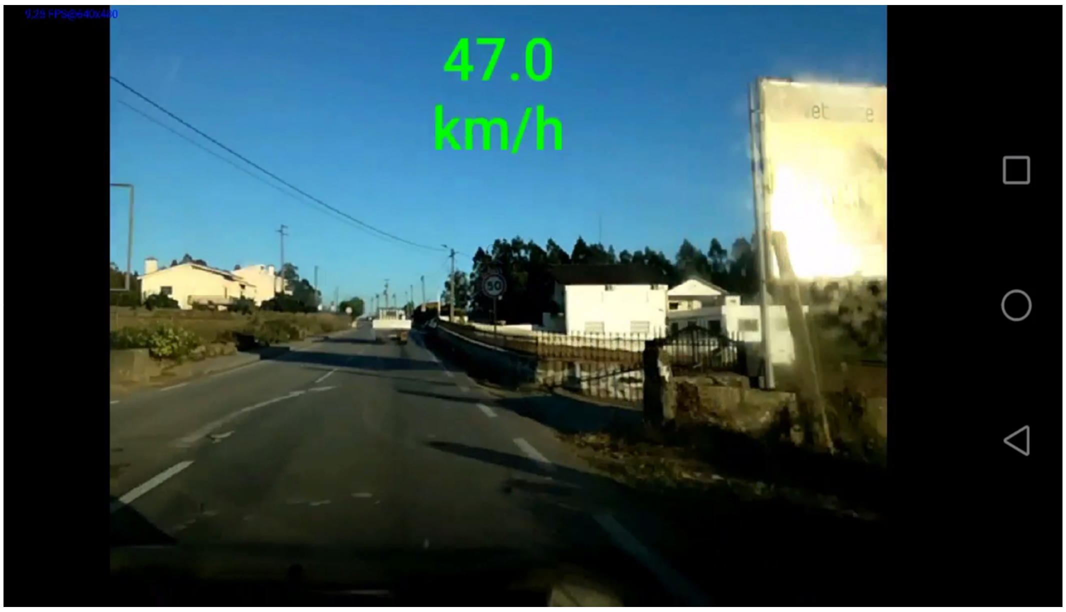


Figure 14.: Driving example

When a sign appears in the road and it is captured by the device camera, is subsequently processed and displayed on the smartphone screen alerting the driver. Figure 15 represents the detection of a 50 speed limit sign.

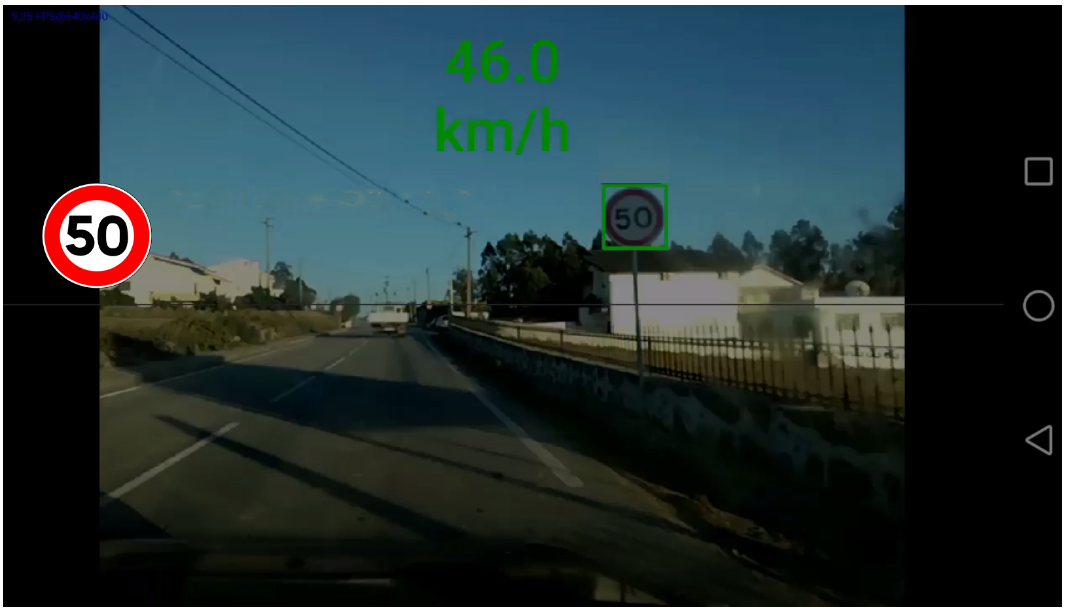


Figure 15.: Detection example

Since the vehicle speed does not exceeds the speed limit, the speedometer stays with a green color. Otherwise, if the car exceeds the limit, the speedometer changes the color to red, warning the driver (Figure 16).

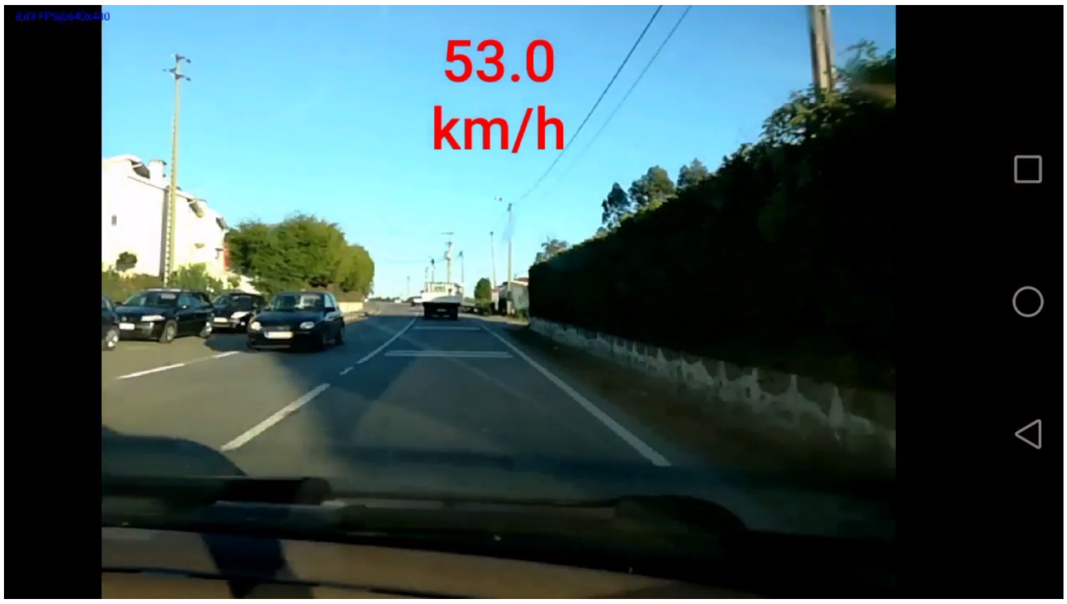


Figure 16.: Overspeeding example

For instance, if the driver is approaching a location where the local copy of the repository has a speed limit sign and the driver is exceeding that limit, which can be detected based on GPS data, the application should notify the driver.

If the sign is in the local repository then the warning can be displayed in advance, otherwise it will be displayed as soon as the sign is recognised.

The same reasoning can be applied to school proximity signs, or signs indicating road maintenance.

3.3 SERVER SIDE

The server side of the framework is responsible for the maintenance of the repository, the elaboration of reports of potentially damaged or obstructed signs, as well as providing clients with the stored traffic signs data so that they can have their own local copy of the repository.

Since we are building a community based platform, it is necessary to respond efficiently to requests of a large volume of clients. For that purpose, Node.js was chosen for the server side. Node.js uses an asynchronous event-driven and non-blocking model with one only thread that listens for events and then triggers a callback function when one of those events is detected. The requests can be executed in parallel, unlike blocking languages that only execute a new request after the previous request has been completed. This improves the application performance and scalability. To create a robust HTTP API easily and quickly the Express framework was used too.

After the request is handled by the server, the data is stored in a MySQL database. The schema has two tables: one for the signs that were first recognised and another table for signs that were recognised subsequently in the same coordinates, but with different labels (Figure 17).

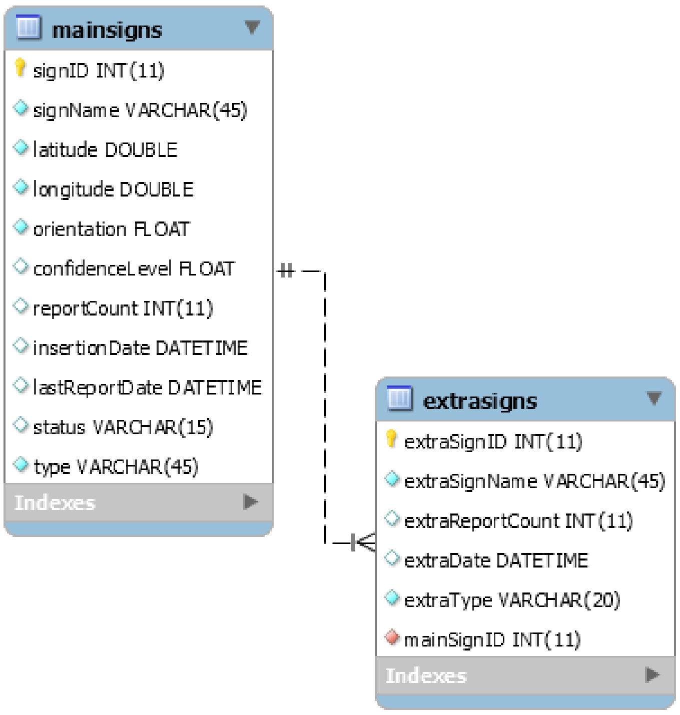


Figure 17.: Repository database logical model

This table has a foreign key that is associated with the ID of the original sign in the primary table. The need to use two tables is to not replace immediately the current sign by the new sign if it is not the same. Storing different sign occurrences in a secondary table may help to realise if it was a recognition failure or if the sign is truly damaged.

If we have two signs in the same coordinates, although they are two different signs, one will be inserted in the main table and the other in the secondary one, because it assumes that it was a different occurrence of a traffic sign already existing in that location. The solution is to differentiate by the sign type. Thus, if the sign has different types, even if they have the same GPS coordinates, both will be inserted in the main table.

The repository contains the following information for each sign:

- sign ID
- sign name
- GPS location

- orientation vector
- confidence level
- report count
- date of insertion
- date of last report
- status
- sign type

The sign name and GPS location are self explanatory. The orientation vector allows to associate a sign with a particular street or road direction. The confidence level is used mainly to elaborate reports and determine the status of the signs. The date of insertion refers to the first entry of the sign, while the data of last report inform when it was last detected. The status tells us if the sign is still active, allowing the repository to include not only the present signs but also an history of sign placement.

The confidence level allows us to determine if a sign should be inspected or even mark it as inactive. When reports arrive from clients the confidence level is updated as a consequence, triggering alarms that can be used for sign inspection. Only signs with a confidence level above a defined threshold are delivered to the clients.

3.3.1 *Handling Reports from Clients*

When the server receives a report from a client with the positive identification of a new traffic sign, it verifies if the sign is already in the database by comparing the sign coordinates of the new entry with all already stored. If the coordinates do not match any previous record then the sign is stored with a predefined initial confidence level. Otherwise, it will compare if the sign recognised is the same as before. If the sign is equal, it will increase the confidence level of the respective sign and updates the date of last report as well. Furthermore, it will average the coordinate values, thus reducing the error. If not, the confidence level is decreased and the sign is stored in the secondary table. If the confidence level decreases below a threshold level, the sign status will change, notifying that the sign may need to be repaired. Figure 18 shows this process.

The reports that arrive to the server may be false positives. But will in principle never get above the required confidence level to get distributed to clients and in a given period of time, if there are no more records of that sign, the repository will automatically delete that data.

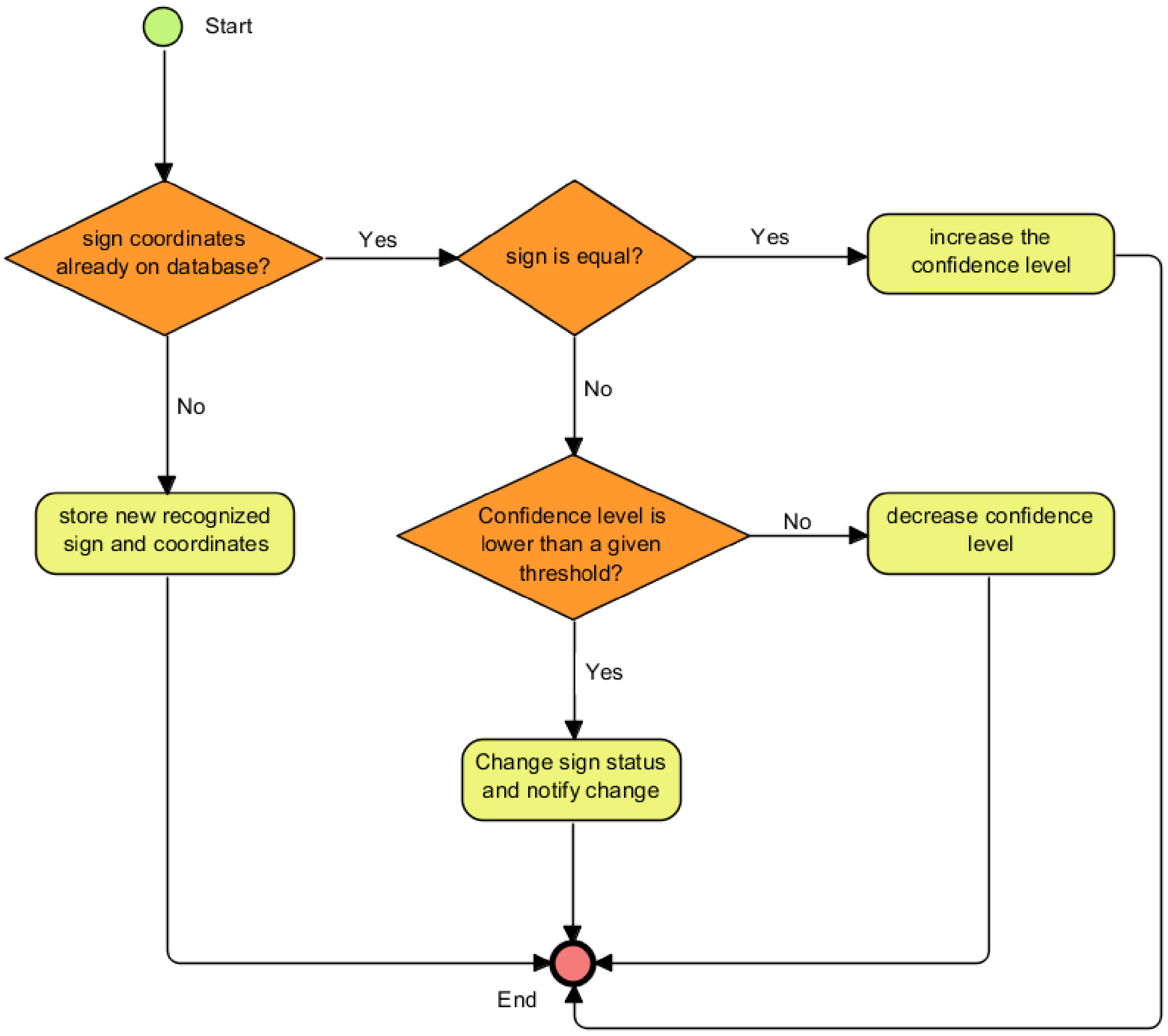

Start
sign coordinates already on database?
Yes
sign is equal?
Yes
increase the confidence level
No
No
store new recognized sign and coordinates
Confidence level is lower than a given threshold?
No
decrease confidence level
Yes
Change sign status and notify change
End

Figure 18.: Repository process

4 DEVELOPMENT AND EXPERIMENTS

In this chapter it will be presented the implementation of the whole architecture and how each module was built as described in chapter 3. Tests and results will be shown too.

4.1 IMPLEMENTATION

The application was based on Lê Quangs' traffic sign recognition system that can be found in `https://github.com/quangpropk/TrafficSignsDetection`. It provides a real-time traffic sign detection algorithm able to find the signs even in adverse environment conditions. The images are acquired directly from the device camera using OpenCV native camera module. This corresponds to the first stage of the complete process (sign detection). All the remaining modules were built from scratch which will be detailed in the following sections.

4.1.1 *Detection*

The application has already two different Cascade Classifier files: one for prohibitory signs and other for danger signs. Further Cascades were created using the templates from the GTSRB dataset for the remaining sign types: stop, end of prohibition, mandatory and yield signs (Figure 19).

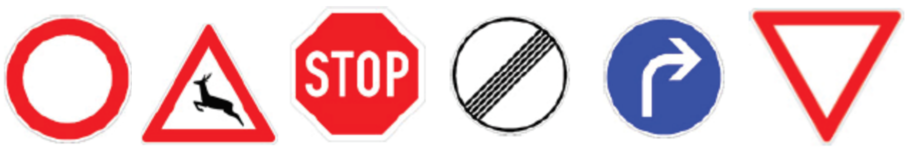


Figure 19.: Sign category examples

4.1.1.1 *Training*

OpenCV requires the positive training images to be of the same aspect ratio. So it is necessary to resize the images before sending them as input to train the Cascade Classifier. An easy way to achieve that is using two ImageMagick commands to resize, crop and center them into the appropriate size (Listing 4.1).

```
mogrify -resize 24x24 *.ppm
mogrify -gravity center -crop 24x24+0+0 *.ppm
```

Listing 4.1: Resize and cropping images

Thereupon, a file should be produced with the list of all the positive images in the format `[filename] 1 0 0 [image width] [image height]` because the images contain only one object. A file for the list of negative images should also be created. To create both files we could use the Unix command lines in Listing 4.2.

```
find . -name '*.ppm' -exec echo \{\} 1 0 0 24 24 \; > positives.dat
find . -name '*.ppm' > negatives .dat
```

Listing 4.2: Create files containing the list of positives and negatives images

The positive samples file should be packed into a .vec file with the `opencv_createsamples` command (Listing 4.3).

```
opencv_createsamples -info positives.dat -vec positives.vec -w 24 -h 24 -num
   700
```

Listing 4.3: Produce dataset of positive samples in a format supported by `opencv_traincascade` command

The -w and -h parameters must be exactly the same used in the mogrify commands executed earlier.

Finally, the training can be done with the `opencv_traincascade` command (Listing 4.4) and uses the follow parameters:

- `-data <cascade_dir_name>` defines the folder where the cascade will be stored as well each stage checkpoint until the training ends.
- `-vec <vec_file_name>` indicates the vector with the positive samples.
- `-bg <background_file_name>` indicates the file with the path of all negative samples.
- `-numPos <number_of_positive_samples>` defines the number of traffic sign templates.
- `-numNeg <number_of_negative_samples>` indicates the number of negative samples.
- `-numStages <number_of_stages>` defines the number of cascade stages to be trained.
- `-featureType <{HAAR(default), LBP, HOG}>` indicates the type of feature in use for feature extraction. For this application it will be used the LBP.

- `-w <sampleWidth>` indicates the uniform width of positive samples.
- `-h <sampleHeight>` indicates the uniform height of positive samples.

```
opencv_traincascade -data newCascade -vec positives.vec -bg negatives.dat -
    numPos 700 -numNeg 1000 -numStages 20 -featureType LBP -w 24 -h 24
```

Listing 4.4: Train Cascade Classifier using LBP features

While training is in process, the `opencv_traincascade` command will save each stage independently and assembles them in a XML file in the end. If the training is interrupted, the last checkpoint is loaded and the training process restarts from that point and not from the beginning all over again. When the training is completed, the cascade file is created as XML file as well and is ready to be used for detection.

4.1.1.2 *Optimizations*

The original application was running at a resolution of 1440x1080. Due to the large amount of pixels to process, the average frames per seconds was 5. Androids' cameras have a wide range of resolutions available from 320x240 to 1920x1080. At a resolution of 640x480 was found an equilibrium between quality and processing time of each frame, running at 10 frames per second.

Since every frame is processed, there is often unnecessary use of computational resources of the mobile device when the car is stationary, as it may not detect or detect a sign several times. One way to control this is to use smartphone sensors to verify if the car is moving. The device GPS receiver is used to verify if the car is in motion or not with the method `getSpeed()`, from the `Location` data class, which returns the vehicle speed. If the speed is zero then the frame processing is paused until the vehicle starts moving again.

The application has six distinct Cascade Classifiers, one for each category, according to sign markings and shape. This brings a performance issue since most smartphones will not be able to sustain an appropriate frame rate if running all six Cascades every frame. Our goal is to maintain a frame rate at least 8 frames per second. This corresponds to a frame every 2.08 meters considering a speed of 60 kilometers per hour, or 3.125 meters at 90 kilometers per hour, which is more than suitable for urban environments taking into consideration that a sign can be detected at a much larger distance. To attain this, each Cascade is executed on an independent thread. However, running six threads at once is quite demanding for the mobile CPU, not achieving the desired frame rate. Therefore, it was decided to divide the work between alternate frames: we will run three of the six Cascades in the first frame and in the next frame the remaining ones are executed.

One last change was loading into the device memory all Cascade Classifiers at once instead of reading from the XML files in every iteration. Thanks to these changes, the application runs in a more fluid way.

4.1.1.3 *Usage*

First, the Cascade XML files created as previously specified must be loaded into the mobile device memory. Next, to detect a sign from a grayscaled image, OpenCV provides the `detectMultiScale` method for the `CascadeClassifier` class. The routine has the parameters described in Table 3.

Parameter	Meaning	Value
const Mat image	The input image in grayscale	
vector<Rect> objects	Vector containing the boundaries of detected sign candidates	
double scaleFactor	Specify how much the image size is reduced at each image scale	1.2
int minNeighbors	The number of neighbours each candidate rectangle should have	3
int flags	Not used	0
Size minSize	Minimum possible object size	32x32
Size maxSize	Maximum possible object size	

Table 3.: DetectMultiScale method parameters

The minimum object resolution was set to 32x32 pixels because it matches with the input images size of the neural network that will be described in Section 4.1.2. The rectangle around the ROI is defined with the OpenCV drawing function `rectangle` (Listing 4.5). It receives the full frame acquired by the device camera in RGB, the top-left and the bottom-right points of the ROI, the rectangle color (in this is case is green) and the rectangle lines thickness.

```
Core.rectangle(mRgba,facesArray[i].tl(), facesArray[i].br(), FACE_RECT_COLOR,
    2);
```

Listing 4.5: Generate rectangle for the ROI

As refered in Section 3.2.1, if the ROI is exposed to large variations in illumination, it suffers an histogram equalisation. First we need to convert the ROI from RGB to YCrCb color space. Then we need to split all the three channels, i.e Y, Cr and Cb and we equalise the Y channel. After the equalisation is complete, we merge again the three channels and convert once more to the RGB channel. The final result is the color image equalised.

4.1.2 *Classification*

The convolutional network topologies described in Section 3.2.2 were tested using the TFLearn API. Before training the network, it was necessary to choose which activation function best suited to penalize the deviation between the predicted and true labels. Normally is used in the last layer of the network (fully-connected).

In Section 2.2.1 two functions were presented : the sigmoid and the hyperbolic tangent functions. Although the hyperbolic tangent can be more accurate, it is very expensive in terms of computational work.

TFLearn possess enumerous activation functions, particularly those already mentioned. The softmax function is a function similar to sigmoid, but for multi-classes. Basically, calculates the sum of all probabilities of each class, which means that the sum of all outputs is one. The sigmoid function only outputs a single value, independent of all other values.

The ReLU function is also used afterwards the convolutional layers because increases the nonlinear properties of the decision function, since computing linear operations have been made during the process, without affecting the receptive fields (Wikipedia, 2016c). It is also much faster than other functions named before.

4.1.2.1 *Training*

For the network training, the full german traffic signs dataset was used. The first step was building a HDF5 dataset. HDF5 is a unique technology that makes possible the management of extremely large and complex data collections (HDF5). TFLearn API provides a method called `build_hdf5_image_dataset()` to create the required dataset, its parameters being presented in Table 4.

Parameter	**Meaning**	**Value**
str target_path	Defines the path of root folder	
tuple image_shape	Resize the input images	(32,32)
str output_path	Defines the output path for the hdf5 dataset	
str mode	Defines the data source mode	'folder'
bool categorical_labels	If True, labels are converted to binary vectors	True
bool normalize	If True, normalize all pictures, dividing the pixels by 255	True

Table 4.: build_hdf5_image_dataset method parameters

Executing the method (Listing 4.6), it takes several minutes to create the dataset.

```
build_hdf5_image_dataset("Images/", image_shape=(32, 32), mode='folder',
    output_path='dataset.h5', categorical_labels=True, normalize=True)
```

Listing 4.6: Build HDF5 image dataset

Once completed, it is not necessary to execut it again to train with other network topologies, unless the size parameter changes. Next, the dataset is loaded, storing the size of the input images and the number of different traffic sign classes in the X and Y variables, respectively (Listing 4.7).

```
import h5py
h5f = h5py.File('dataset.h5', 'r')
X = h5f['X']
Y = h5f['Y']
```

Listing 4.7: Load dataset

The network topologies described in Section 3.2.2.1 were used to initialise network training. TFLearn provides "layers" that represent an abstract set of operations to make building neural networks more convenient (Listing 4.8).

```
network = conv_2d(network, 32, 3, activation='relu')
network = max_pool_2d(network, 2)
network = conv_2d(network, 64, 3, activation='relu')
network = conv_2d(network, 64, 3, activation='relu')
network = max_pool_2d(network, 2)
network = fully_connected(network, 512, activation='relu')
network = fully_connected(network, CLASSES, activation='softmax', name="out")
network = regression(network, optimizer='adam',
                     loss='categorical_crossentropy',
                     learning_rate=0.001)
```

Listing 4.8: Network layers

Once the network model has been built, the training can be initialized using the `DNN` model class.

Besides the X and Y variables used to feed the training model, it was defined the number of epochs (or cycles), the `batch_size` that defines the number of training examples in one forward pass, the `snapshot_shot_step` which will create a training checkpoint every *N* steps, the `snapshot_epoch` that saves a model checkpoint at each epoch if `True`, the `run_id` provides a name for the run. Finally, the `validation_set` uses a percentage of the input data for validation, in order to minimise the overfitting (Listing 4.9).

```
model = DNN(network, checkpoint_path='model_signs.ckpt', max_checkpoints=1)

model.fit(X, Y, n_epoch=150, shuffle=True,
        show_metric=True, batch_size=64, snapshot_step=False,
        snapshot_epoch=False, run_id='network_id', validation_set=0.1)
```

Listing 4.9: Network training

Finally, the network model is saved in a file with any extension (Listing 4.10).

```
model.save('signs_neural_network.tflearn')
```

Listing 4.10: Save the network model

During the training, we analyzed the accuracy according to the number of epochs. The three network topologies were trained with 150 epochs each. The CIFAR-10 network was the most accurate (Figure 20).

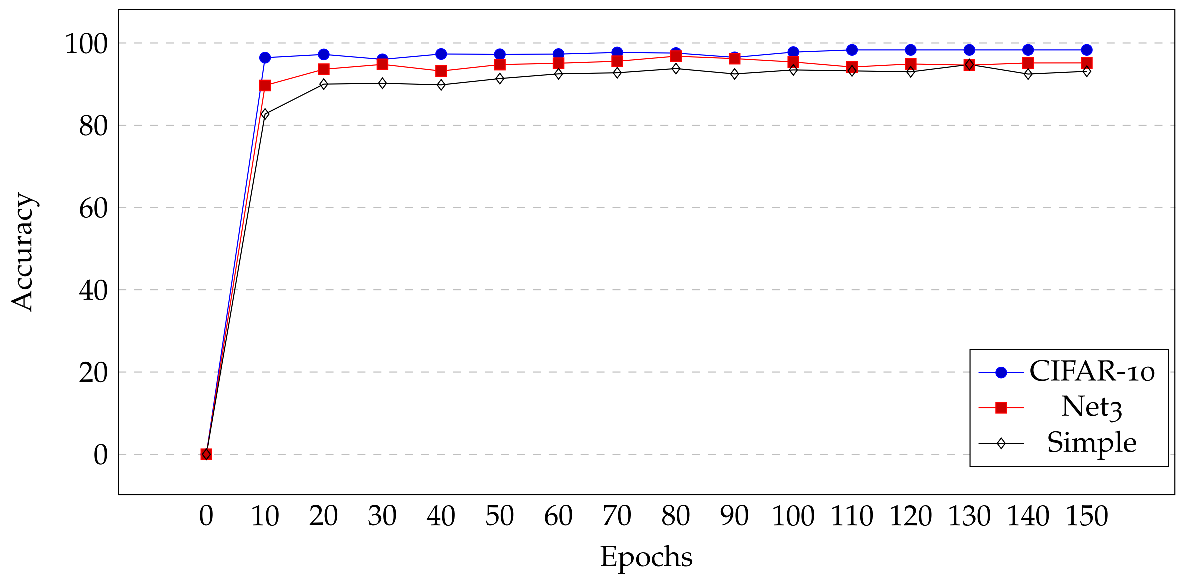


Figure 20.: Neural network training progress

4.1.2.2 *Freezing Neural Network*

On TensorFlow, weights usually are not stored inside the file format during training. Instead, they are held in separate checkpoint files, and there are variable operations in the neural network that load the latest values when they are initialized. Since we need to use the neural network on a smartphone device, it is not very convenient to have separate files. For that purpose we can use the `freeze_graph.py` script that takes a graph definition and a set of checkpoints and freezes them together into a single file.

What this does is loading the neural network model, pull in the values for all the variables from the latest checkpoint file, and then replace each variable operation, i.e "Add", "MatMul", or "Conv2D", with a constant operation that has the numerical data for the weights stored in its attributes. It then strips away all the extraneous nodes that are not used for forward inference, and saves out the resulting model into an output file.

4.1.2.3 *Usage*

To run the neural network in the smartphone we have to use the Java Native Interface (JNI) to call the functions because the core of the TensorFlow is written in C++.

To initialize our classifier, we must use the function `initializeTensorFlow()`. It receives as parameter the freezed model of the CNN, the number of classes, the input image size, the image mean and a label file. The label file contains the name of all 43 distinct traffic signs and it was created with the means of matching the neural network result output with the correspondent traffic sign label.

Next, the `classifyImageBmp()` receives the ROI as input to classify it. The result of the operation is a list of all probabilities for each class. Later, we select the result with the higher probability.

4.1.3 *Georeferencing*

The first thing we need is to know is the GPS location of the device. As described in Section 3.2.3 the Fused Location Provider API was used. We must declare a `LocationRequest` that is a data object to request a quality of service for location updates.

As we want the highest accuracy location possible, we must create a location request with `setPriority()` set to `PRIORITY_HIGH_ACCURACY`, `setInterval()` to 500 milliseconds and `setSmallestDisplacement()` to one meter. This would be appropriate for mapping applications that are showing the location in real-time.

To compute the shift from the device camera to the sign, we need the focal length of the camera, which can be obtained with the Android API. The result obtained was 515 pixels for a Huawei P9 Lite. To double check this value we also performed a test with the help of VIZARIO.Cam application and a chessboard pattern. Initially a calibration matrix is computed. The calibration matrix has five parameters: (f_x, f_y) are the focal lengths and (u_0, v_0) which are the optical centers expressed in pixels coordinates. γ represents the skew coefficient between the x and the y axis, and is often 0 (Equation 7).

$$K = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

After 50 photos of a chessboard pattern taken from different angles, the calibration matrix in Equation 8 was acquired providing a focal length of approximately 521 pixels. Hence, the results provided by the API seem accurate enough for our purpose, at least for this device.

$$K = \begin{bmatrix} 520.9933 & 0 & 319.9882 \\ 0 & 521.2108 & 243.8441 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Finally, to calculate the device orientation we must use the hardware sensors. One alternative was using the accelerometer and magnetic field sensors. Basically, we need to register the sensors `TYPE_ACCELEROMETER` and `TYPE_MAGNETIC_FIELD` and get the measured data. Unfortunately, if the device suffers any linear acceleration or if there are is magnetic interference the measured values are getting noisy. The usage of the gyroscope jointly with the accelerometer and magnetic field can dramatically improve the measurement since it has a very good response time.

The sensor `TYPE_ROTATION_VECTOR` is the combination of all sensors for measuring the devices' orientation device. It uses the accelerometer, gyroscope and magnetometer if they are available. It needs to initially orient itself and then eliminate the drift that comes with the gyroscope over time.

To increase the orientation accuracy, we considered an average of the last five orientation values.

4.1.4 *Client-Server Communication*

To make the HTTP requests using the Retrofit REST client, we need to describe the API endpoints URL that we want to interact with Listing 4.11.

```java
public interface API {
    @GET("signs/list")
    Call<ArrayList<Sign>> getSigns(@Query("latitude") String latitude,
        @Query("longitude") String longitude);

    @POST("signs/add")
    Call<ResponseBody> uploadSigns(@Body SignsData data);
}
```

Listing 4.11: Retrofit HTTP API as a Java Interface

It defines the API and the methods `getSigns()` and `uploadSigns()` to request the list of signs in a certain radius and upload a list of collected signs, respectively. The `@GET` annotation declares that this request uses the HTTP GET method, while the `@POST` annotation uses the HTTP POST method. The code snippet also illustrates the usage of Retrofit's `@Query` parameter functionality. In the `getSigns()` method, the `{latitude}` and `{longitude}` queries will be attached to the URL endpoint, e.g. `?latitude=40.56&longitude=-8.39`, when calling the method. In the `uploadSigns()` method, the `@Body` parameter is replaced by the Java object containing all the collected signs by the user.

The `ServiceGenerator` class (Listing 4.12) uses `Retrofit` builder to create a new REST client with the given API base url (`BASE_URL`). For example, the traffic signs repository base url is located at `https://traffic-signs-repository.herokuapp.com`.

Retrofit can be configured to use a specific converter. This converter handles the data (de)serialization. There are several converters already available for various serialization formats. The converter used was Gson that will map the defined object to JSON format and it will finally send as the request's body to your defined server.

```java
public class ServiceGenerator {
    private static String BASE_URL =
        "https://traffic-signs-repository.herokuapp.com";

    @NonNull
    static Retrofit getRetrofit() {

        return new Retrofit.Builder()
                .baseUrl(AppConfig.BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
    }
}
```

Listing 4.12: ServiceGenerator class

Besides performance, other aspects such as mobile data should be taken into account given possible Wi-Fi network access restrictions. Everytime a sign is detected and recognised, the information regarding the sign is sent to the server. Therefore, the devices' own internet data will be necessary to perform this operation. Although only a few bytes of mobile data are required for one operation, several traffic signs tend to appear in a short period of time, which leads to an exaggerated consumption of data and may even overload the server. In addition, when the system receives a partial copy of the existing data stored in the repository to alert the driver of possible signs ahead on the road, large data consumptions may occur.

To avoid overloading requests to the server and save mobile data, it was decided to periodically send the set of captured signs. In order to accomplish this, an AsyncTask was used to upload the signs every ten minutes. The signs were converted to a Java object `SignsData` (Listing 4.13) that is an ArrayList of SignData object (Listing 4.14), which has the necessary attributes for each sign.

```java
public class SignsData {
    private ArrayList<SignData> signsdata;
}
```

Listing 4.13: SignsData Java object

```java
public class SignData {
    @SerializedName("name")
    private String signName;
    @SerializedName("latitude")
    private String latitude;
    @SerializedName("longitude")
    private String longitude;
    @SerializedName("orientation")
    private String orientation;
    @SerializedName("type")
    private String type;
    ...
}
```

Listing 4.14: SignData Java object

The `CreateRequests` class (Listing 4.15) is responsible for the operations execution.

```java
public class CreateRequests {
    public void uploadSignsSet(SignsData coords) {
        API getResponse = ServiceGenerator.getRetrofit().create(API.class);
        Call<ResponseBody> call = getResponse.uploadSigns(coords);
        call.enqueue(new Callback<ArrayList<Sign>>() {
            @Override
            public void onResponse(Call<ArrayList<Sign>> call,
                Response<ArrayList<Sign>> response) {
                if (response.isSuccessful()) {
                    ...
                } else {
                    ...
                }
            }
            @Override
            public void onFailure(Call<ArrayList<Sign>> call, Throwable t) {
                ...
            }
        });
    }
}
```

Listing 4.15: CreateRequests class

It contains two methods: the `getSigns()` that will fetch all the stored signs in the repository within a radius of 100 kilometers and the `uploadSignsSet()` that will upload a set of collected signs. Taking the example of the latter method, first we create a usable client from the annotated interface for API requests (Listing 4.11).

The resulting client will be able to execute the network requests. After that we make an asynchronous request, which implies to implement a callback with its two callback methods: `onResponse` and `onFailure`. When calling the asynchronous `uploadSignsSet()` method, with a `SignsData` object that contains all the signs to be sent to the server as a parameter, we have to implement a new callback and define what should be done once the request finishes.

4.1.5 *Repository*

When the server receives the clients' data, which contains a set of recognised signs along with its georeference, it will also execute a SQL query to select all signs in a defined radius, being the coordinates of the first detected sign the central point of the circle. It is quite hard to be accurate because of the Earth's irregular shape. If we model the Earth as a sphere the shortest path between two coordinates is on a large circle and we can compute the distance using the Haversine formula or the Spherical Law of Cosines.

We have tested the difference between the distance accuracies and the computational performance of both equations. In terms of execution time, the Haversine formula is slightly slower than the Spherical Law of Cosines by 18%. In the case of the distance differences, we can see in Figure 21 that for distances less than about 0.5 meters, the two formulas diverge. Above 0.5 meters they tend to agree. Since the radius is in the order of the kilometers, the difference is not significant.

Although the Spherical Law of Cosines seems to be a good alternative, this operation must be performed quite often, so it is not very appropriate because it is slow to compute. Assuming that the world is pretty flat over distances of a few kilometers, we could estimate the great circle with a straight line in the latitude-longitude space will produce minimal error, hence a lot faster to calculate.

This way we could use the Euclidean distance. A minor complication is the fact that the length of a degree of longitude depends on the latitude. As we said before, a degree of longitude spans 111319.44 meters on the Equator. Adjusting for this is easy: multiply the longitude by the cosine of the latitude. Then we can take the Euclidean distance between the two points, and multiply by the length of a degree (equation 9):

$$distance = 111319.44 * \sqrt{(lat2 - lat1)^2 + (\cos(lat1) * (lng2 - lng1))^2} \tag{9}$$

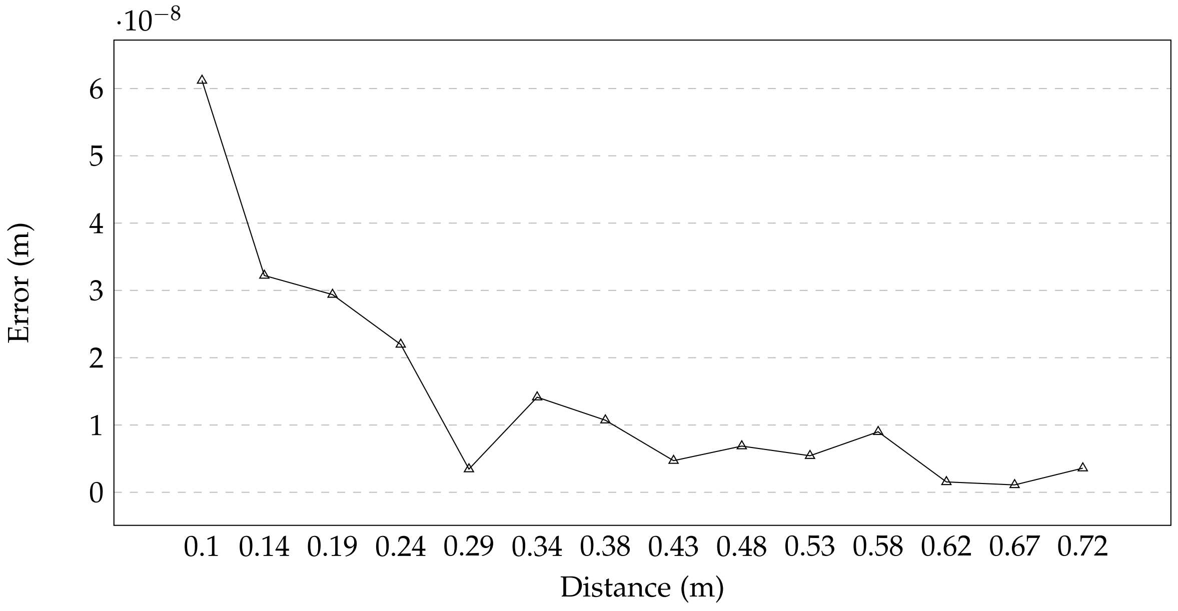


Figure 21.: Distance differences between Haversine formula and Spherical Law of Cosines

After some small tests, the Euclidean distance proved to be over seven times faster than the Spherical Law of Cosines and for distances up to 50 kilometers is of 56 meters at the most, which is not very significant. Hence, the formula used was the Euclidean distance.

The objective is to compare the current data with a partial copy of the full database. The first phase is to compare the coordinates of each current sign with all the coordinates of the partial copy. If the signs' coordinates are identical, this means that the sign has already been detected by another client. As we know, the computed signs' coordinates from various clients are not always the same even if the sign is the same. There is always a small variance of a few meters. To overcome this problem, we can verify if the current coordinates are inside of a small radius around the previous location. Since the Euclidean distance error is five centimetres for distances up to ten meters, we can use in a similar way as for the selection of the partial copy of the repository.

So, if the sign coordinate is at a distance of up to ten meters and the sign name is the same, it will average the coordinates, thus decreasing the location error. Besides that, it will also increase the confidence level, report count and updates the date of last report. If the sign is different in the same coordinates, then a new sign will be created and it will be inserted in the secondary table of the database. The only parameter that will be modified in the main table in this case is the confidence level that will decrease. If it will decrease below a defined threshold value, the sign status parameter will change from "OK", that is its default value, to "DAMAGED" (Listing 4.16).

```
UPDATE signsTable set confidenceLevel = IF(confidenceLevel > 0,
    confidenceLevel - 0.01, confidenceLevel), status = IF(confidenceLevel <
    0.25, 'DAMAGED', status) WHERE signID = ? "
```

Listing 4.16: Update sign query

If there is no occurrence of this sign, it means that it is not in the repository. So, a new sign is inserted into the main table with default parameters.

4.1.5.1 *Optimizations*

Initially, each processed sign was sent individually into the database. This does not scale very well, because we will have to make a request to the database for each sign. If a client sends a hundred signals at once, it can become problematic and may lead to the system failure due to lack of resources. The ideal would be to send all processed data at once.

`INSERT` statements that use `VALUES` syntax can insert multiple rows (Listing 4.17). To do this, we must create nested array of objects in which each of these contains the sign data. The question mark is the placeholder for the processed data to be stored.

```
INSERT INTO signsTable (latitude, longitude, signName, orientation,
    confidenceLevel, reportCount, insertionDate, lastReportDate, status, type)
     VALUES ?
```

Listing 4.17: Bulk Insert query example

However, we can only execute a bulk update if the rows are updated with the same values. As we need to update multiple rows with distinct values it is not possible.

Another optimization made was regarding the search of the traffic signs sent to the repository, as well as the comparison with the ones already pre-existent. In other words, as an attempt to reduce the number of comparisons and unnecessary computational work, when a sign is found during the search in the partial copy of the repository, the search stops once there is an occurrence of the same traffic sign data in the existing database.

4.1.6 *Notifications*

After the client device receives a partial copy of the repository signs, they will be stored in a SQLite local database. This database, in turn, is accessed periodically through an AsyncTask to verify if a sign is near by from the driver's location. For this, it is necessary to know the GPS position of the vehicle and its orientation. Since the orientation is in radians, it can vary from 0 to 6.28 approximately. So, regarding the vehicle orientation, we can select from the database the traffic signs in a range of radians, e.g signs between 2 and 2.99 radians.

To verify the distance of the current vehicle location and the signs in the database we can use a strategy similar to that used in the repository to check for traffic signs in a given radius, i.e use the Euclidean distance. Since SQLite does not suport trigonometric functions, we can not use the cosine of the latitude correction in the equation. So the results are not so accurate. For a distance of 500 meters the error is in the order of 165 meters.

4.2 TESTS AND RESULTS

In here we report on the tests performed on the client mobile application to evaluate the feasibility of our proposal regarding the smartphone performance. We also test the some CNN topologies variants, some options we made during training, as well as the georeference accuracy.

4.2.1 *Topologies Testing*

Different topologies with different filter sizes were tested using the GTSRB test dataset. The dataset contains 12630 images, approximately one third of the training dataset size. The results can be consulted in Table 5.

Topology	Filter size	Epochs	Accuracy
Simple	3	150	88.12%
CIFAR-10	3	150	94.24%
Net3	3	150	92.34%
Net3	5	150	90.98%

Table 5.: Neural network accuracies

As we can see, the most accurate topology is the CIFAR-10. This is due to its two stacked convolutional layers that can develop more complex features of the input volume before the destructive pooling operation. The least accurate was the simple topology, as expected. A single Convolutional Neural Network is not capable of extracting all the important features from the input images while training. Hence, the classification is not as accurate.

Comparing both Net3 topologies with different filter sizes we see an accuracy decrease if using a 5x5 rather than a 3x3 filter.

Intuitively, stacking convolutional layers with tiny filters as opposed to having one convolutional layer with big filters allows us to express more powerful features of the input, and with fewer parameters.

4.2.2 *CIFAR-10 network testing*

Since the CIFAR-10 network topology obtained the highest accuracy rate compared to the others topologies, it was chosen to proceed with the remaining tests. Before testing the application on a vehicle, the CNN was tested on a desktop to determine its accuracy. In addition to the already tested german database, where it achieve success recognition rate of 94.24%, another database was used.

The Belgian Traffic Sign Dataset (BTSD) (Timofte et al., 2009) has 1850 similar traffic signs that were tested in our CNN achieving an overall success rate of 92.32%.

The neural networks suffered from similar problems, during the course of the tests, in both databases. Some of the cases in which the CNN was not able to classify the signs from the german database were due to very low resolution images, large illumination variations, vegetation occlusion or vandalism (Figure 22).

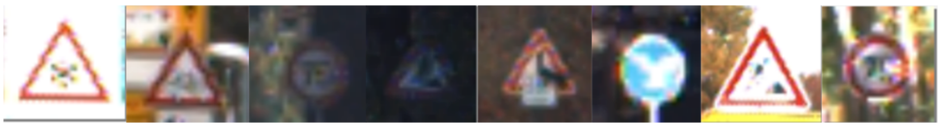


Figure 22.: GTSRB incorrect recognition examples

In the case of the belgium database, Figure 23 shows some examples of rotated and discolored signs that are difficult to identify, even for the human eye. There are also examples where the signs are only partly visible in the image.

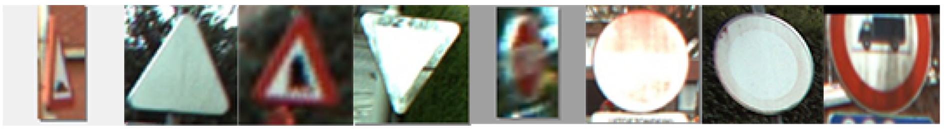


Figure 23.: BTSD incorrect recognition examples

The results for both database test sets are presented in Table 6 showing the number of correct recognitions and the recognition rate. The small discrepancy between both recognition rates is due to the slight difference between the German and Belgian signs.

Database	Total signs	Correctly Classified	Recognition Rate
GTSRB	12630	11902	94.24%
BTSD	1850	1708	92.32%

Table 6.: GTSRB and BTSD recognition testing results

4.2.3 *Client Testing*

The device used for testing was a Huawei P9 Lite which has a octa-core processor ARM-Cortex 53, 4 cores with 1709 MHz and another 4 with 2016 MHz of clock speed. Although being a powerful mobile device, it runs on a RISC architecture, where benchmarks have proven that is much slower than an Intel Core i5 or i7 in terms of CPU performance. Using the Geekbench 4 benchmark, it was possible to determine that the mobile device CPU was 3 to 5 times slower than x64 desktop processors (Figure 24).

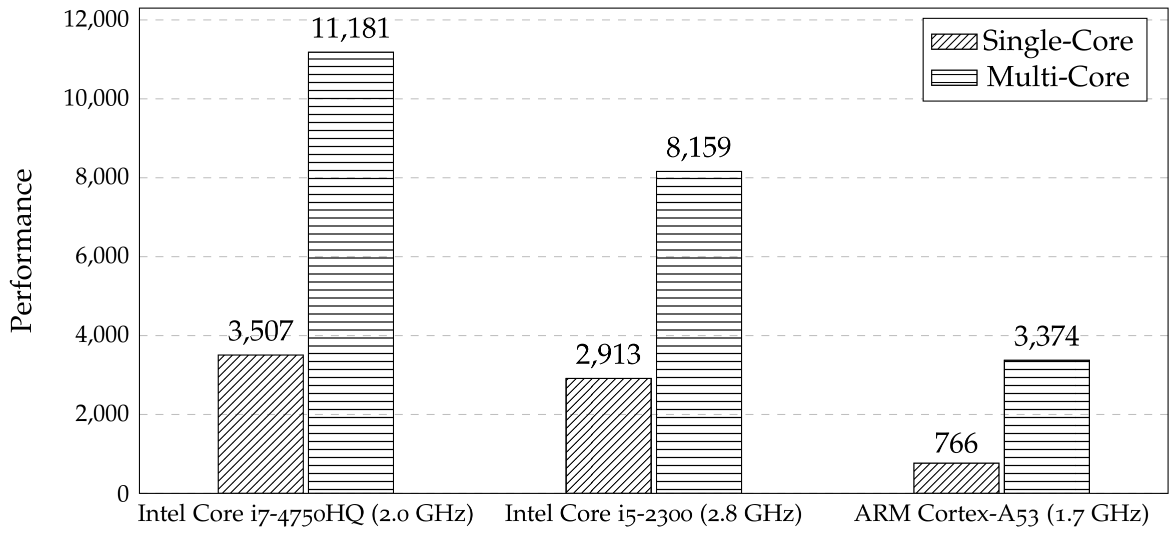


Figure 24.: Benchmark of CISC vs RISC processors

Nevertheless, the application achieved an acceptable frame rate of approximately 10 frames per second at a camera resolution set at 640x480.

We performed some live tests driving a car and recording the number of signs detected and identified, as well as their GPS locations. The average speed for this drive was 60 km/h. As mentioned in Section 3.2, the classification of traffic signs is a two phase process. First we detect regions of interest (ROI) in the full frame captured by the devices' camera where potentially we can have a traffic sign, and then we use a CNN applied to the ROI to recognize the sign. The system behaved well when driving in day time, even with lighting variations.

Table 7 reports on the detection phase, detailing the number of hits, misses and false positives for each Cascade Classifier. Were detected 138 signs from a total of 145 that was trained for. The detection rate is over 95%.

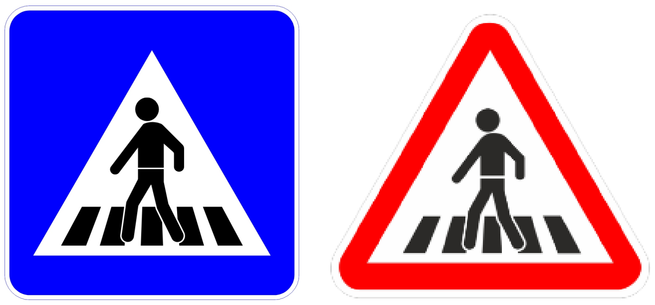
One probable cause for the misses occurred was the possible motion blur of the acquired image and hence missing important information.

The case of false positives was due to the existence of objects with shapes similar to traffic signs, but not only. In the sign presented in Figure 25a, the system identified it as a danger

Cascade Type	Hits	Misses	False positives
Prohibition	52	2	3
Danger	57	2	2
Stop	5	-	1
End of prohibition	4	-	-
Mandatory	15	2	3
Yield	5	1	-

Table 7.: Detection results

sign. The most similar danger sign is shown in the Figure 25b. This is due to the inner triangle of the informative sign and since the Cascade Classifier works in a similar way as a shape detector, it is natural to have this kind of mistakes. The CNN was able to recognise these signs successfully.



(a) Pedestrian crossing informative sign (b) Pedestrian crossing danger sign

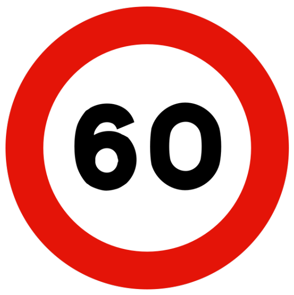
Figure 25.: Informative and danger sign comparison

Regarding the second phase, sign recognition, the CNN was able to recognize successfully 121 of the 138 traffic signs successfully detected on the first phase (Table 8). Combining both reports we get a successful identification of 121 out of 145 signs, i.e. a 83.44% success rate.

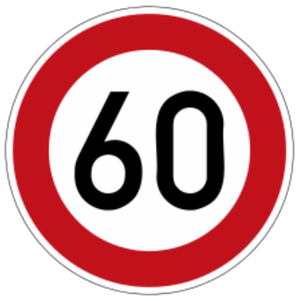
Total signs	Detection Rate	Recognition Rate	Global Rate
145	95.51%	87.68%	83.44%

Table 8.: Recognition results

Some of recognition problems registered were due to sign similarities. Figure 26a represents the portuguese 60 speed limit and Figure 26b represents the german 60 speed limit, which was used for the CNN training. The two templates differ in the way the number 6 is represented. When the CNN is trying to classify the sign, the closest result it gets is the 80 speed limit sign (Figure 27), since the number 8 and 6 are similar in terms of shape.



(a) Portuguese 60 speed limit sign



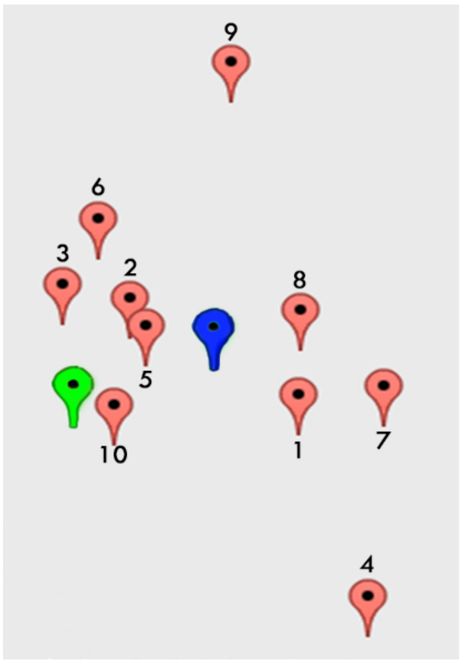
(b) German 60 speed limit sign

Figure 26.: Speed limit templates comparison

Figure 27.: German 80 speed limit sign

A short georeferencing accuracy test was performed doing 10 passes with a vehicle in a short circuit capturing the same traffic sign. On the right side of Figure 28, the red markers represent the recorded positions of the vehicle on each lap. On the left side of the image are the calculated sign locations relative to when they were detected and recognised by the device. The computed sign location differs from 0.68 to 5.4 meters to the real sign position (green marker on the right side of Figure 28). Performing an average of all ten coordinates we achieved a distance of 2.18 meters (blue marker).

The main cause of the lack of accuracy is the devices' GPS. The system can provide the location and altitude, but it has inherent error sources that have to be taken into account when a device receives the GPS signals from the satellites in orbit. The main GPS error source is due to inaccurate time-keeping by the devices' clock. These little time discrepancies can create problems in terms of the exact location of a given object. Trees and buildings also have a great influence in GPS accuracy. Hence, the car position varies a lot. Another factor may be magnetic interference from the vehicle or other devices which causes drift in smartphone orientation.



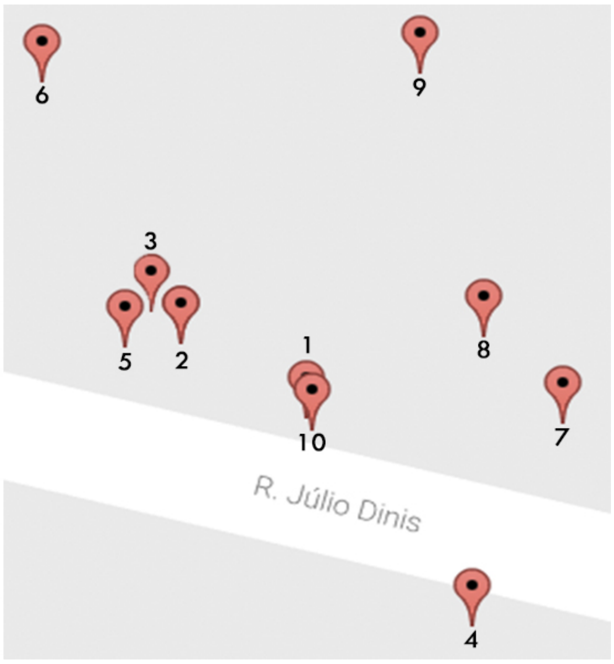


Figure 28.: Left: Sign computed locations (pink), mean value (blue) and sign real location (green); Right: Vehicle GPS location

5 CONCLUSION

As proposed in the beginning of this dissertation, our purpose was to create a client-server framework for a community based traffic sign repository. In order to do so, several steps were taken.

Regarding the client side, a mobile application was developed with the goal of detecting, recognising and georeferencing traffic signs. There are many approaches for traffic sign detection using computer vision techniques, After a careful study of the strong and weak points for each of these approaches, we opted for a feature based detection.

In terms of traffic sign recognition, convolutional neural networks were found to be highly efficient and accurate.

For the last phase of this process, the georeferencing phase, two equations based on the Krsák and Toth (2011) paper were used and produced positive results in terms of location accuracy.

We have shown that a current medium range smartphone can perform as a client for this framework which means that in the near future every smartphone will have the necessary computing power to be used as a traffic sign report contributor client in this framework.

Regarding the server side, this is where the central repository is located. The main advantage of a community based approach is the constant update of the traffic signs' state, which is possible with the contribution of each client. In other words, a vast number of people using it will promote a faster update of the traffic signs' state and more signs will be covered. This creates the possibility to alert entities responsible for road maintenance of potentially damaged or occluded traffic signs, granting a more timely and directed inspection.

5.1 PROSPECT FOR FUTURE WORK

As future work we would like to use the framework itself to build an always increasing database of images that can be used for further training of the detection and recognition system increasing its accuracy.

We also would like to take advantage of the Open-StreetMaps API to be able to determine the size of the traffic signs based on the road classification, provide street names and speed limits, amongst other possibilities.

Another useful extension would be to detect road marks. Road marks are an important safety feature that also requires periodic inspection. Extending this framework would enable to assert the status of these marks avoiding on site inspection.

Performance wise, we could use the mobile GPU to process the detection and recognition of traffic signs to verify if there are any significant advantages in comparison to the current CPU approach.

The neural network is only able to recognize 43 different traffic signs for now. There are other databases, for exemple the Belgium Traffic Sign database that can extend the number of classes, expanding the capabilities of the actual neural network.

BIBLIOGRAPHY

A brief introduction to rest. URL: https://www.infoq.com/articles/rest-introduction. Accessed: 2016-11-20.

The hsi color space. URL: https://pt.scribd.com/doc/16617255/HSI-Color-Space. Accessed: 2016-11-04.

Rest vs soap, the difference between soap and rest. URL: http://spf13.com/post/soap-vs-rest. Accessed: 2016-11-20.

P. Medici A. Broggi, P. Cerri and P. P. Porta. Real time road signs recognition. *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 981–986, 2007.

Russell Beale and Tom Jackson. *Neural Computing-an introduction*. CRC Press, 1990.

David Bouchain. Character recognition using convolutional neural networks. *Institute for Neural Information Processing*, 2007, 2006.

Karla Brkic. An overview of traffic sign detection methods. *Department of Electronics, Microelectronics, Computer and Intelligent Systems Faculty of Electrical Engineering and Computing Unska*, 3:10000, 2010.

Built With. Websites using google maps. URL: https://trends.builtwith.com/websitelist/Google-Maps. Accessed: 2017-01-12.

Joao L Cardoso, Christian Stefan, Rune Elvik, and Michael Srensen. Road safety inspection-best practice guidelines and implementation steps. Technical report, Technical report, deliverable D5 of the EU FP6 project RIPCORD, ISEREST, 2007.

Arturo De la Escalera, J Ma Armingol, and Mario Mata. Traffic sign recognition and analysis for intelligent vehicles. *Image and vision computing*, 21(3):247–258, 2003.

Express. Express: Node.js web application framework. URL: https://expressjs.com/. Accessed: 2017-08-18.

Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4): 193–202, 1980.

Dariu M Gavrila. Traffic sign recognition revisited. In *Mustererkennung 1999*, pages 86–93. Springer, 1999.

Geekbench 4. Processor benchmarks. URL: https://browser.primatelabs.com/processor-benchmarks. Accessed: 2017-01-08.

Google Map Maker. Google map maker graduates to google maps. URL: https://productforums.google.com/forum/#!topic/map-maker/UEN0wxhj6Rs. Accessed: 2017-01-12.

HDF5. What is hdf5? URL: https://support.hdfgroup.org/HDF5/whatishdf5.html. Accessed: 2017-01-09.

Samer Hijazi, Rishi Kumar, and Chris Rowen. Using convolutional neural networks for image recognition. 2015.

Di Huang, Caifeng Shan, Mohsen Ardabilian, Yunhong Wang, and Liming Chen. Local Binary Patterns and Its Application to Facial Image Analysis: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(4):1–17, March 2011. doi: 10.1109/TSMCC.2011.2118750. URL http://liris.cnrs.fr/publis/?id=5004.

TS Huang. Computer vision: Evolution and promise. *Imaging science and technology, Evolution and promise*, pages 13–20, 1996.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

IMT. Sinalização vertical - características. URL: http://www.imtt.pt/sites/IMTT/Portugues/InfraestruturasRodoviarias/InovacaoNormalizacao/Divulgao%20Tcnica/SinalizacaoVerticalCaracteristicas.pdf. Accessed: 2016-12-26.

Menno-Jan Kraak. Settings and needs for web cartography. *Web cartography: Developments and prospects*, pages 1–7, 2001.

Emil Krsák and Stefan Toth. Traffic sign recognition and localization for databases of traffic signs. *Acta Electrotechnica et Informatica*, 11(4):31, 2011.

Carlos Javier Fernández Laguía. *A study of wireless digital posts and traffic signs using smartphones*. 2016.

Paul J Leach, Tim Berners-Lee, Jeffrey C Mogul, Larry Masinter, Roy T Fielding, and James Gettys. Hypertext transfer protocol–http/1.1. 1999.

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.

LeNet. Convolutional neural networks (lenet). URL: http://deeplearning.net/tutorial/lenet.html. Accessed: 2016-12-13.

Fei-Fei Li and Andrej Karpathy. Convolutional neural networks for visual recognition, 2015.

Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z. Li. *Learning Multi-scale Block Local Binary Patterns for Face Recognition*, pages 828–837. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74549-5. doi: 10.1007/978-3-540-74549-5_87. URL http://dx.doi.org/10.1007/978-3-540-74549-5_87.

Gareth Loy and Nick Barnes. Fast shape-based road sign detection for a driver assistance system. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 70–75. IEEE, 2004.

T Mccarthy, N Maynooth, C Mcelhinney, C Cahalane, and P Kumar. Initial results from european road safety inspection (eursi) mobile mapping project. *Proceedings of ISPRS CRIMT, Newcastle*, 2010.

Andreas Mogelmose, Mohan M Trivedi, and Thomas B Moeslund. Traffic sign detection and analysis: Recent studies and emerging trends. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, 2012.

Gavin Mulligan, Denis Gra, et al. A comparison of soap and rest implementations of a service based interaction independence middleware framework. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 1423–1432. IEEE, 2009.

Y. Nguwi and A. Kouzani. A study on automatic recognition of road signs. *Procs. IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6, 2006.

Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, July 2002. ISSN 0162-8828. doi: 10.1109/TPAMI.2002.1017623. URL http://dx.doi.org/10.1109/TPAMI.2002.1017623.

OkHttp. Okhttp: An http http/2 client for android and java applications. URL: http://square.github.io/okhttp/. Accessed: 2017-09-18.

Open Source Computer Vision. Face detection using haar cascades. URL: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html. Accessed: 2016-11-13.

OpenCV User Guide. Cascade classifier training. URL: http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html. Accessed: 2016-11-14.

OpenStreetMap. Comparision google services. URL: http://wiki.openstreetmap.org/wiki/Comparision_Google_services_-_OSM. Accessed: 2017-01-12.

Overpass API. Overpass api. URL: http://wiki.openstreetmap.org/wiki/Overpass_API. Accessed: 2017-01-12.

Sri-Kaushik Pavani, David Delgado, and Alejandro F Frangi. Haar-like features with optimally weighted rectangles for rapid object detection. *Pattern Recognition*, 43(1):160–172, 2010.

Tamás Kozsik Peter Somogyi, Jukka Nurminen and Máté Szalay-Beko. *Analysis of Server-Smartphone Application Communication Patterns*. 2014.

Retrofit. Retrofit: A type-safe http client for android and java. URL: http://square.github.io/retrofit/. Accessed: 2017-09-18.

Ronald J Rumelhart, David E Williams and Geoffrey E Hinton. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Bradley Steven Russell. *A comparison of neural network and regression models for Navy retention modeling*. PhD thesis, Monterey, California. Naval Postgraduate School, 1993.

P. Siegmann H. GomezMoreno S. Maldonado-Bascon, S. Lafuente-Arroyo and F. Acevedo-Rodriguez. Traffic sign recognition system for inventory purposes. *Proc. of IV, Eindhoven*, page 590–595, 2008.

A. M. Sousa J. F. Sobral S. R. Madeira, L. C. Bastos and L. P. Santos. Automatic traffic signs inventory using a mobile mapping system for gis applications. *International Conference and Exhibition on Geographic Information*, 2005.

Robert E. Schapire. *Explaining AdaBoost*, pages 37–52. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-41136-6. doi: 10.1007/978-3-642-41136-6_5. URL http://dx.doi.org/10.1007/978-3-642-41136-6_5.

Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):411–426, 2007.

Alok Sinha. Client-server computing. *Communications of the ACM*, 35(7):77–98, 1992.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Square. Square open source. URL: https://square.github.io/. Accessed: 2017-09-18.

TensorFlow. Tensorflow. URL: https://www.tensorflow.org/. Accessed: 2016-12-28.

TFLearn. Tflearn: Deep learning library featuring a higher-level api for tensorflow. URL: http://tflearn.org/. Accessed: 2016-12-28.

Radu Timofte, Karel Zimmermann, and Luc Van Gool. Multi-view traffic sign detection, recognition, and 3d localisation. In *Applications of Computer Vision (WACV), 2009 Workshop on*, pages 1–8. IEEE, 2009.

Hai S. Tran. *Traffic Sign Recognition system on Android devices*. Massey University, 2013.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages 511–518. IEEE, 2001.

VIZARIO.Cam. Vizario.cam. URL: https://www.vizar.io/vizariocam/. Accessed: 2017-06-18.

Wikipedia. Haar-like features. https://en.wikipedia.org/wiki/Haar-like_features, 2016a. Accessed: 2016-12-23.

Wikipedia. Overfitting. URL: https://en.wikipedia.org/wiki/Overfitting, 2016b. Accessed: 2016-12-13.

Wikipedia. Convolutional neural network. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network, 2016c. Accessed: 2016-12-13.

Wikipedia. Backpropagation. URL: https://en.wikipedia.org/wiki/Backpropagation, 2016d. Accessed: 2017-01-02.

Wikipedia. Google maps. URL: https://en.wikipedia.org/wiki/Google_Maps, 2016e. Accessed: 2017-01-03.

T. Asakura Y. Aoyagi. *A study on traffic sign recognition in scene image using genetic algorithms and neural networks*. 22nd International Conference on Industrial Electronics, Control, and Instrumentation, 1996. IEEE.

A

SCIENTIFIC PAPER

The award for best paper at the 24th Portuguese Meeting of Computer Graphics and Interaction that toke place between 12 and 13 October 2017 at the Azurém Campus of the University of Minho, in Guimarães was awarded to the researchers Hélder Novais and António Ramires Fernandes that authored the paper entitled "Community Based Repository for Georeferenced Traffic Signs".

Community Based Repository for Georeferenced Traffic Signs

Helder Novais
Departamento de Informática
Universidade do Minho, Portugal
a64378@alunos.uminho.pt

António Ramires Fernandes
Centro Algoritmi
Universidade do Minho, Portugal
arf@di.uminho.pt

***Abstract*—Traffic sign maintenance requires periodic on-site inspection to determine if signs are in good conditions and visible, both day and night. However, periodic inspections are time and cost consuming. Another issue is related to the drivers awareness to the traffic signs on the road. Many factors may potentially contribute to a driver missing a sign, such as the sign being damaged or occluded, or distraction caused by the many gadgets inside the vehicle.**

We propose a dual purpose community based approach. On the one hand, each driver can use his mobile device to detect, recognize and geolocate traffic signs, contributing to the traffic sign central repository. Detection is performed using cascade classifiers, while a convolutional neural network support the recognition phase. The repository, based on the information received from the clients, can be used to provide reports about sign status, preventing the need for global inspections and providing the information required for more direct and timely inspections. On the other hand, the drivers would have access to the database of traffic signs therefore being able to receive real-time notifications regarding traffic signs such as speed limit signs, school proximity, or road construction signs.

***Keywords*—community-based approach, computer vision, deep learning, traffic sign maintenance, trafic sign recognition**

I. Introduction

Road traffic is an interaction between vehicles and pedestrians. One of the most important elements in road infrastructures are traffic signs. They control traffic flow, inform about directions and distances, warn about road condition, prohibit/mandate the driver to do certain actions and guide the right of way.

One of the most efficient instruments to detect issues on road infrastructures is road inspection [1]. The inspection procedure implies the assessment of the current road conditions against the stored reference state. These consist of periodical on-site evaluation carried out by trained safety expert teams and results in a formal report which is delivered to the relevant road authority. The most important part of the inspection relies on traffic control elements maintenance like traffic signs and road surface markings [2]. Unfortunately, road safety inspection is very expensive both regarding the time and expertise required. Furthermore, depending on the interval at which they are carried out, a sign may find itself in poor visible conditions for a long period of time.

978-1-5386-2080-9/17/$31.00 ©2017 IEEE

Signs which are not clearly visible may be easily missed by the driver, potentially being a safety issue. Furthermore, with all the gadgets in recent cars it is not hard to miss important signs. To improve safety it is important to increase the awareness of the driver.

We propose a community based approach to deal with both these problems which consists of a server side repository and a client mobile application.

On the one hand, we have a client mobile application that can recognize traffic signs and display warnings to the driver. Signs recognized by the client application are sent to a server maintaining a central repository, which in turn feeds the clients with the sign repository. Nowadays, capable smartphones are widely available as they have become increasingly powerful and monetarily accessible over the years. They integrate various types of sensors such as accelerometer, magnetometer, gyroscope, GPS and Wi-Fi connection, as well as a significant increase in camera quality. This makes these devices very good candidates to achieve this goal.

The server, based on the reports received by the clients, can infer the sign's condition, i.e., when a sign is in good condition, or damaged and/or occluded. Therefore, it has the potential to avoid global traffic sign inspections and timely direct inspection teams to signs which are potentially damaged or occluded.

On the other hand, since clients are fed with a database of previously recognized signs, they do not rely only on the effective recognition of signs to be able to display warnings to the user. This makes this framework suitable to a wide range of smartphones. High end devices, computationally capable of performing the recognition tasks, will be able to contribute to the central repository, while less performing devices can still benefit from the traffic sign repository for issuing alerts to the driver.

The remainder of this document is structured as follows: section II describes previous work on traffic sign recognition and geoinformation repositories; section III provides an overview of the proposed client-server architecture; section IV describes the mobile client application in detail, including the traffic sign identification, and its communication with the server, whereas section V describes the server side, including the repository structure, its update policies and report elaboration. Tests describing the real usage of the client application, results from the sign recognition and detection phase, and georeferencing accuracy are reported on section VI. The paper concludes in section VII.

II. Related Work

This section reports on previous work both on traffic sign recognition, as well as georeferenced inventories (section II-C).

Most traffic sign recognition (TRS) systems have a detection phase (section II-A) and a recognition phase (section II-B). First the image is acquired by a camera where it starts to search for regions of interest (ROI), i.e., areas that resemble traffic signs. Once a ROI is detected, it goes to the recognition phase where it will be classified.

A. Traffic Sign Detection

There are several techniques that can be applied for traffic sign recognition. One of the most common technique is based on colour, since traffic signs colours are normally easy to distinguish from the surrounding environment [3]. Due to external factors, like lighting conditions is not always possible to apply a threshold directly to RGB. Escalera and Mata [4] chose HSI space for colour analysis.

Other studies conducted a more exhaustive colour study. For example, Aoyagy and Asakura [5] noted that colour sign segmentation is not a good solution because the Hue component changes with the deterioration of the signs caused by conditions like weather, and sign age. Concluding, the colour based segmentation stage is likely to cause issues when detecting more weather exposed and older signs.

Just like signs have specific colours, they also have well defined shapes that can be searched for. Several approaches for shape-based detection of traffic signs have been developed as well, being the approximation of the Hough transform approach the most common [6] [7]. Other approaches performs simple template matching from a database in a image [8]. However, they suffer from object occlusion, rotation or distortion. Hence, the sign shape can look different and not all detectors can handle that.

To overcome these problems, a more generic sign detection based on features could be used. One of the earliest feature based detection methods was proposed by Paul Viola and Michael Jones [9] where they used Haar wavelets features. These features are similar to convolution kernels which are used to detect the presence of a specific feature in the input image, such as edges, corners, line ends, or spots. Timofte et al. [10] use the Viola-Jones detector to detect six different sign classes.

Another type of feature detector is called Local Binary Patterns (LBP) [11]. Training and detection is several times faster in contrast to Haar features because it performs all calculations using integers instead of floats [12]. In terms of detection accuracy, it is possible to train a LBP-based classifier that will provide almost the same quality as a Haar-based one. Nevertheless, the basic LBP operator has some limitations such as the inability of capturing dominant features with large scale structures due to its small neighborhood. The Multi-scale Block Local Binary Patterns (MB-LBP) [13] is a more generic solution since it allows to use neighborhoods of different sizes. Experiments show that MB-LBP significantly outperforms other LBP based method in terms of accuracy [13], being a more robust method, although it brings a little computational overhead over the original LBP operator.

B. Traffic Sign Recognition

Similarly to the detection methods, the recognition phase consists in training a classifier to learn certain features in order to predict the correct sign class.

Maldonado-Bascon et al. [14] use a Support Vector Machine (SVM) to an SVM is trained to classify the shape of a detected traffic sign. Timofte el al. [10] system uses six SVM classifiers, each one for a different traffic sign subclasses (triangle-up, triangle-down, circle-blue, circle-red, rectangle and diamond) for the different candidate traffic signs. They work on the RGB colour channels normalized by the intensity variance.

Another type of classifier frequently encountered in traffic signs recognition literature is an artificial neural network. In particular, multilayer perceptron (MLP) and convolutional neural network (CNN). The main difference between these two is that a CNN has convolutional and pooling layers. The convolutional layer is used for feature extraction to detect local patterns, like edges. The pooling layer is used to reduce the spatial size of the representation, controlling the network overfitting. Credi et al. [15] trained a CNN for traffic sign recognition using a combination of greyscale and the original RGB raw images reaching an classification accuracy of 96.9% on the German Traffic Sign Recognition Benchmark (GTSRB) [16]. Sermanet and LeCun [17] achieved 98.97% with a multi-scale convolutional network architecture. Ciresan et al. [18] introduced a multi-column CNN and achieved a correct recognition rate of 99.46%. These last two methods surpassed the human recognition rate stated at 98.84% [19].

C. Geoinformation inventories

Normally these detection and recognition methods are used for visual data acquisition with help of a specialized equipped vehicle. Besides the requirement of at least one camera to capture the road images it is common to use of a differential GPS receiver to be able to georeference the sign. In addition, to increase the system accuracy, inertial sensors, odometer or infrared cameras may also be used [20] [21].

Arnoul et al. [21] describes an early inventory system called AUTOCAT, which enables automatic detection and georeference of traffic signs. The road is simultaneously observed with two cameras: a normal camera and an infrared camera. A sign is detected by combining infrared readings and color information. When a sign is detected, it is photographed using a high resolution digital camera. As GPS technology was not publicly available at the time, the 3D location of the sign is reconstructed using camera motion information provided by an inertial odometer and Kalman filtering. The photo and the 3D location are then stored in the inventory.

Several approaches for automating, at least partly, the inspection procedure were developed later [20] [22] [23]. However they require to have a vehicle equipped with particular hardware to travel on all roads to achieve a full inspection, which requires a large amount of time and respective man power.

The system described in Madeira et al. [20] relies on video data acquired by two cameras on a van, a GPS and an inertial navigation device to obtain the instantaneous position and orientation of the vehicle. Absolute geographic coordinates of the detected signs are then obtained using triangulation. Chigorin and Konushin [24] also use triangulation to obtain the signs' position in world coordinates.

Kršák and Toth [25] use two analytical expressions to calculate the approximate coordinates of the traffic sign. The vehicles position obtained from the GPS receiver is shifted by a constant value in latitude and longitude because signs are usually located on the left or right side of the road.

III. Global Framework Architecture

The framework has two components: a client mobile application (section IV), and a server side traffic sign repository (section V). On the server side, the repository will contain all the traffic sign information gathered by the mobile clients. Mobile clients perform image acquisition, traffic sign detection, recognition and georeferencing (section IV-A). When traffic a sign is detected it is matched against the information on the mobile local copy of the repository. Based on the result of this match the client will send information to the repository to:

- report on a new sign not present in the local copy of the repository;
- confirm a sign location if present in the local copy of the repository;
- report a failure to recognize a sign which is present in the local copy of the repository.

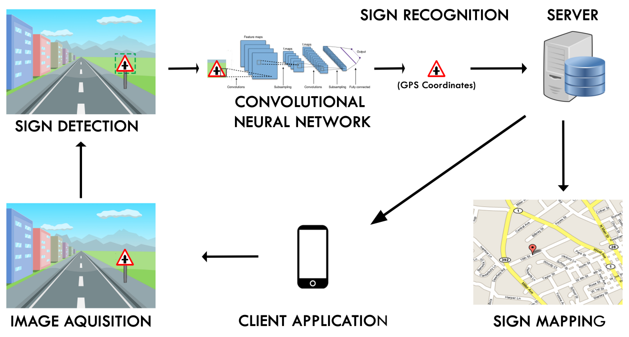


Fig. 1: System architecture

The server side repository in turn feeds new signs to the clients. These communications can be performed immediately for clients who allow the usage of mobile data, or deferred to when Wi-Fi is available. The overall process is depicted in Figure 1.

Besides acting as a traffic sign information collector, the client side is also in charge of notifying the driver to the presence of some relevant signs in some circumstances (section IV-B).

On the server side the information gathered by the clients is processed (section V-B) and stored in the repository (section V-A). This information can later be used to elaborate reports for road traffic sign maintenance and generate a global map of all georeferenced signs (section V-C).

IV. Client Side

The client side is responsible for both collecting traffic sign information and displaying warnings to the driver when certain conditions are met. For low end devices only the later will be performed. The main advantage of using this framework under these circumstances is the ability to have an application that can warn the driver of specific situations as described in section IV-B.

When the car is in motion the client is constantly performing two tasks: identifying traffic signs of images acquired by the camera, and checking the local repository to see if it contains signs that match the car motion direction and location.

When the application identifies a sign which is already in the local repository it reports a confirmation to the server.

If a sign present in the local repository is not detected or recognized, and the appropriate cascade is being used (see section IV-A), the client also reports to the server. These situations can mean that the sign recognition requires further training to become more accurate, or that the sign may be damaged or partially occluded by vegetation for instance.

Both situations are relevant for this framework, therefore, it is essential to report this situation to the server. The reported information should contain the ROI of the detected sign, the classification result from the CNN, the GPS location and the motion direction of the car. The motion direction is relevant to establish in which side of the road the sign is located. When no sign is detected the ROI is void, hence, it should send the full image. This would allow to later determine on the server side if it is a detection issue or a damaged/occluded sign.

If the application identifies a sign which is not on the local repository this may be due to a false positive sign identification, or to a new sign which needs to be added to the repository. In both cases the associated information gets posted to the repository. False positives may imply that the detection or recognition phases require further training. Nevertheless, the server is able to cope with false positives without having "ghost" signs sent to the clients, see section V-B.

A. *Traffic sign detection, recognition and georeferencing*

For sign detection MB-LBP features were used, as implemented in the OpenCV Training Cascade library [12], since it is several times faster in terms of performance than Haar or HOG features, and performance is a relevant issue in smartphones.

The application has six distinct cascade classifiers, one for each category, according to sign markings and shape.

The categories we are currently considering are: prohibition, danger, stop, end of prohibition, mandatory and yield signs. Figure 2 presents an example of each category in the order defined.

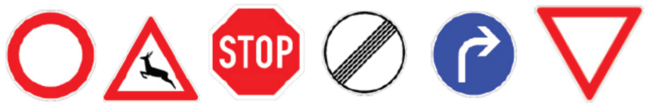


Fig. 2: Sign category example

This brings a performance issue since most smartphones will not be able to sustain an appropriate frame rate if running all six cascades every frame. Our goal is to maintain a frame rate at least 8 frames per second. This corresponds to a frame every 2.08 meters considering a speed of 60 km/h, or 3.125 meters at 90 km/h, which is more than suitable for urban environments taking into consideration that a sign can be detected at a much larger distance. To attain this, each cascade is executed on an independent thread.

However, running six threads at once is quite demanding for the mobile CPU, not achieving the desired frame rate. Therefore, it was decided to divide the work between alternate frames: we will run three of the six cascades in the first frame and in the next frame the remaining ones are executed.

The recognition stage is handled by a CNN using a modular deep learning library called TFlearn built on top of TensorFlow. The proposed network receives as input an 32x32 image, corresponding to the ROI, and has 3 convolutional layers, one with 32 3x3 filters and two stacked with 64 3x3 filters. In between there are two max-pooling layers before reaching the last fully connected layer (Figure 3).

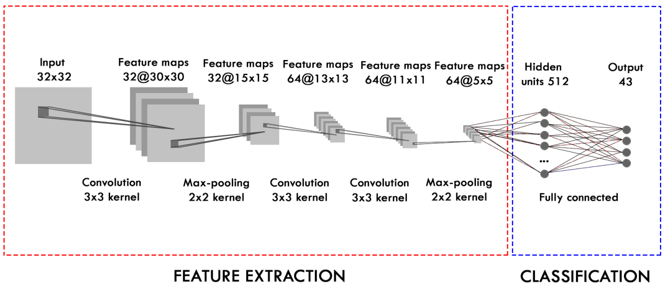


Fig. 3: CNN structure

Regarding the recognition phase, first the ROI is scaled to 32x32 (in accordance with the input of the recognition phase). Signs detected on the full frame may vary a lot due to different illumination conditions, which may cause difficulties in this stage. So if the image is too dark or too bright, based on a threshold value, a preprocessing is performed: the ROI undergoes an histogram equalization in YCrCb color space, adjusting the intensity of the image pixels. An example of the effect of this equalization is shown in Figure 4. If the average pixel intensity of the ROI is in between the given thresholds, the raw image is sent directly to the recognition stage, without any preprocessing.

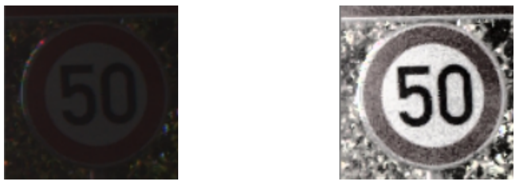


Fig. 4: Left: Original image from the GTSRB; Right: Histogram equalized image

The cascade classifiers and the CNN require training before being usable. The GTSRB dataset was used for this purpose. It provides 39,209 training images for 43 distinct classes of signs. Each class contains a considerable number of traffic sign images with different levels of brightness, with random objects partially occluding the sign, different perspective angles, blur and noise, being suitable for sign classification even in uncontrolled environments such as outdoor environments.

Figure 5 illustrate the detection and recognition procedure.

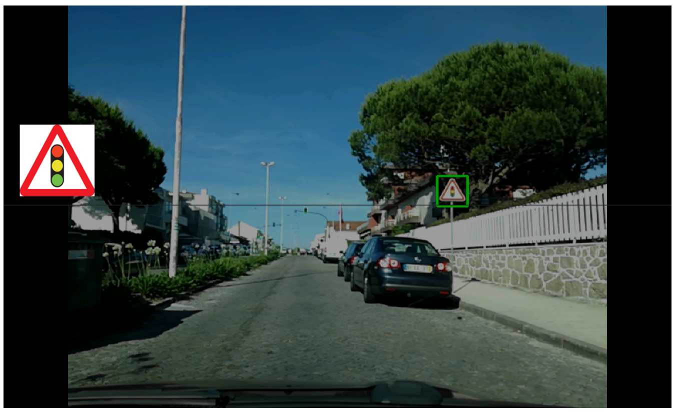


Fig. 5: Sign detection example

The next step is sign geolocation. In [25], equations 1 and 2 are used to determine the approximate position of the recognized sign.

$$s_x = x + p.c_x.\sin\beta - c_y.\cos\beta \tag{1}$$

$$s_y = y + p.c_x.\cos\beta + c_y.\sin\beta \tag{2}$$

where s_x and s_y represents the calculated longitude and latitude of the recognized traffic sign. The values of variables x, y are obtained from a GPS device and represent the current vehicle longitude and latitude, respectively. To provide the highest accurate location Google's Fused Location Provider API was used. It analyses GPS, cellular and Wi-Fi network location data. Also, this provider uses different device sensors to define if a user is walking, riding a bicycle, driving a car or just standing in order to adjust the frequency of location updates. So it does not try to update your location as frequently if the accelerometer indicates that you are not moving, which has the side benefit of saving battery.

Variable p symbolizes the position of the sign in the frame captured by the smartphone's camera. If the sign is located on the left half of the image, then p is +1, else is -1.

Value β represents the azimuth which indicates the angle in radians between the device's current compass direction and magnetic north. To calculate the orientation data the hardware sensors were used.

Values c_x and c_y are used to compute the shift of position in longitude and latitude. The c_x refers to the distance from the car to the side of the road and has been set at 0.00005 in decimal degrees by based on experimental results. The c_y is obtained based on the sign size in pixels, calculating the distance of the camera to the plane, perpendicular to the device view direction, where the sign can be found.

To perform this latter computation we need the focal length of the camera, which can be obtained with the Android API. The result obtained was 515 pixels for a Huawei P9 Lite.

To determine the distance c_y from the camera to the sign, required for equations 1 and 2, we can use the distance equation 3.

$$c_y = (W * F)/P \tag{3}$$

Where P is the sign width in pixels, W is the real width of the sign and F is the focal length that was acquired before (in pixels). This results in an approximation of the distance from the device to the plane, perpendicular to the view direction, where the sign can be found.

The IMT [26] reports the size of the Portuguese traffic signs as presented in Table I.

Dimensions (cm)	**Small**	**Normal**	**Large**
Width	60.0	70.0 - 90.0	115.0

TABLE I: Portuguese sign dimensions

The specific size used depends on the classification of the road, with 70 centimeters being used for urban roads. At this stage we are focused only on urban areas so we will assume all signs have this dimension.

After the distance is determined, we need to convert to decimal degrees to use in equations 1 and 2. Hence, we have to convert this distance in meters to both a shift in latitude and longitude. In the equator both latitude and longitude represent approximately the same shift, so we convert our distances to degrees considering this referential, i.e., we divide both c_x and c_y by 111319.44, the distance corresponding to one degree at the equator.

Then we apply a longitude correction factor to equation 1 using the inverse cosine of the latitude, as in equation 4.

$$s_x = x + \frac{p.c_x.\sin\beta - c_y.\cos\beta}{cos(y)} \tag{4}$$

Taking the example of Figure 5, we will present an example of the sign georeferencing and the actual location obtained.

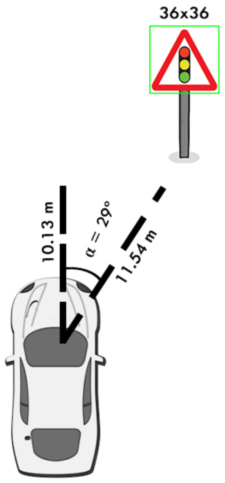


Fig. 6: Sign georeferencing illustration

The size of the detected ROI was 36x36 pixels, and using equation 3 provides a distance of 10.13 meters. Based on both legs we can compute the hypotenuse which represents the distance from the device to the sign. Knowing both the hypotenuse and one of the legs we get the angle $\alpha = 29$ degrees, corresponding to a distance of 11.54 meters as shown in Figure 6. With help of Google Street View we can see that the marking is at a distance of 2.22 meters (Figure 7).

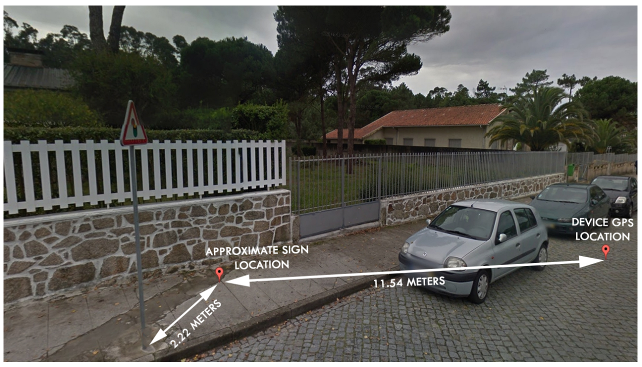


Fig. 7: Sign location

B. Driver Notification

Signs can be configured for driver notification when some conditions are met. For instance, if the driver is approaching a location where the local copy of the repository has a speed limit sign and the driver is exceeding that limit, which can be detected based on GPS data, the application should notify the driver.

If the sign is in the local repository then the warning can be displayed in advance, otherwise it will be displayed as soon as the sign is recognized.

The same reasoning can be applied to school proximity signs, or signs indicating road maintenance, in which case the warning can be displayed always regardless of the speed of the vehicle.

The client can be configured for this purpose via a configuration file that associates a sign category to a condition, where the condition can be a speed threshold, or empty (in which case it is always true). The main advantage of having a local repository is the ability to notify the driver in advance. This can be achieved by also storing a time value, which can then be translated to a distance since the current speed is known.

V. Server Side

A. Repository database

The server side of the framework is responsible for the maintenance of the repository, the elaboration of reports of potentially damaged or obstructed signs, as well as feeding clients so that they can have their own local copy of the repository.

The repository contains the following information for each sign:

- sign id
- gps location
- motion vector
- confidence level
- report count
- date of insertion
- status

The first two items are self explanatory. The motion vector allows to associate a sign with a particular street or road direction. The confidence level is used mainly to elaborate reports and determine the status of the signs. The status tells us if the sign is still active, allowing the repository to include not only the present signs but also an history of sign placement.

The GPS location is also an issue that has to be tackled since the smartphone hardware does not have the accuracy of the GPS sensors used in other approaches such as [20] [23]. Our approach is to store a running average of the reported locations, thereby potentially decreasing over time the location error, see section VI for a short test regarding this issue.

The confidence level allows us to determine if a sign should be inspected or even mark it as inactive. When reports arrive from the clients the confidence level is updated as a consequence, triggering alarms that can be used for sign inspection. Only signs with a the confidence level above a defined threshold are fed to the clients.

B. Handling reports from clients

When the server receives a report from a client with the positive identification of a new traffic sign, the sign is added to the repository with a predefined initial confidence level. As each report of the same nature for the same sign arrives its confidence level is increased. When the confidence level reaches a set level then a new sign alarm is triggered and the sign starts to be fed to the clients.

If a report arrives notifying the server that the client was unable to recognize a particular sign then its confidence level is decreased.

For signs that are truly damaged of significantly occluded it is expected that these reports keep arriving eventually triggering an alarm for low confidence level. Also reports from false positives will in principle never get above the required level of confidence to get distributed to clients.

For temporary signs, such as road construction signs, the confidence level can be updated more aggressively to ensure that on the one hand the sign is provided to the clients in useful time, and on the other hand that the sign does not remain available to clients long after it is removed from site.

C. Report elaboration and mapping the signs

The ability to determine which signs are potentially damaged of severely occluded based on the confidence level is one of the most relevant features of this framework. Based on these reports the traffic sign inspections can be more directed and performed timely.

Furthermore, global traffic sign periodic inspections will no longer be necessary. If a significant user base is achieved then the framework will be able to determine with a large degree of confidence when and where inspections are required.

Another feature of interest is the ability to populate a map, such as Google maps or Open Street maps with traffic signs based on the stored GPS coordinates. Based on this mapping the reports can indicate street names and road identification, or even contain the map itself, in addition to the GPS coordinates making it more easily readable by road inspectors.

VI. Tests and results

In here we report on the tests performed on the client to evaluate the feasibility of our proposal regarding the smartphone performance. We also test the CNN regarding some options we made regarding training, as well as the georeference accuracy.

A. CNN testing

Before testing the application on a vehicle, the CNN was tested on a desktop to determine its accuracy. Two distinct databases were used: The first was the GTSRB testing database with 12630 images where over 94% success recognition rate was achieved.

Some of the cases in which the CNN was not able to classify the signs were due to very low resolution images, large illumination variations, vegetation occlusion or vandalism (Figure 8).

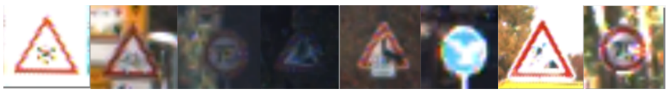


Fig. 8: GTSRB incorrect recognition examples

The Belgian Traffic Sign Dataset [10] (BTSD) has also been used for further testing. 1850 similar traffic signs templates were tested in our CNN achieving an overall success rate of 92.32%. When we got an incorrect recognition the CNN classification was of similar traffic signs. Furthermore, the misclassified signs had identical problems as the GTSRB database.

Fig. 9: BTSD incorrect recognition examples

Figure 9 shows some examples of rotate and discolored signs that are difficult to identify, even for the human eye.

The results for both database test sets are presented in Table II showing the number of correct recognitions and the recognition rate.

Database	Total traffic signs	Signs correctly classified	Recognition rate
GTSRB	12630	11902	94.24%
BTSD	1850	1708	92.32%

TABLE II: GTSRB and BTSD recognition testing results

B. *Client testing*

The device used for testing was a Huawei P9 Lite which has an octa-core processor ARM-Cortex 53, 4 cores with 1709MHz and another 4 with 2016MHz of clock speed. The camera resolution was set at 640x480 and we we're able to achieve approximately 10 frames per second.

We performed some live tests driving a car and recording the number of signs detected and identified, as well as their GPS locations. The average speed for this drive was 60 km/h.

As mentioned in section IV-A, the identification of traffic signs is a two phase process. First we detect regions of interest (ROI) where potentially we can have a traffic sign, and then we use a CNN applied to the ROI to recognize the sign.

Cascade type	Hits	Misses	False positives
Prohibition	52	2	3
Danger	57	2	2
Stop	5	-	1
End of prohibition	4	-	-
Mandatory	15	2	3
Yield	5	1	-

TABLE III: Detection results

Table III reports on the detection phase, detailing the number of hits, misses and false positives for each cascade classifier. The detection rate is over 95%. Most of the misses occurred when having multiple signs on a single image (see Figure 10).

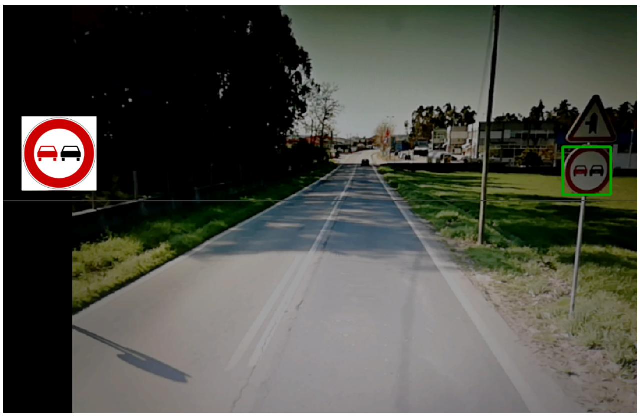


Fig. 10: Example of the detection phase missing a sign when multiple signs are present

Regarding the second phase, sign recognition, the CNN was able to recognize successfully 121 of the 138 traffic signs successfully detected on the first phase (Table IV).

Detection Rate	Recognition rate
95.51%	87.68%

TABLE IV: Recognition results

Combining both reports we get a successful identification of 121 out of 145 signs, i.e. a 83.44% success rate.

A short georeferencing accuracy test was performed doing 10 passes with a vehicle capturing the same traffic sign. In the left image from Figure 11 every red marker represents the recorded position of the vehicle on each lap.

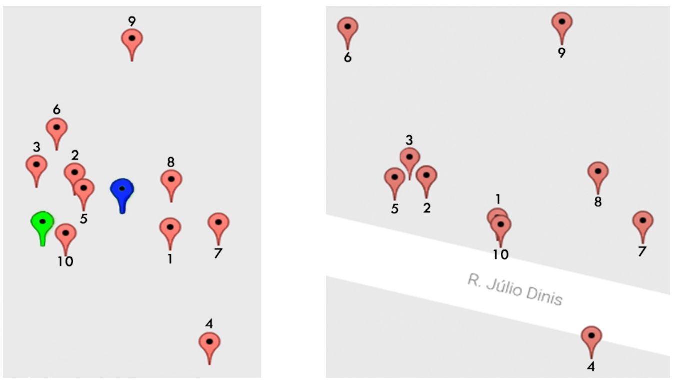


Fig. 11: Left: Sign computed locations (pink), mean value (blue) and sign real location (green); Right: Vehicle GPS location

Due to the low accuracy of the smartphone's GPS the car position varies a lot. Another factor may be magnetic interference from the vehicle or other devices which causes drift in smartphone's orientation. All this makes it difficult to calculate the sign's exact position. The computed sign location differs from 0.68 to 5.4 meters to the real sign position (green marker on the right image from Figure 11). Doing the average of all ten coordinates we achieved a distance of 2.18 meters (blue marker).

VII. Conclusion

We proposed a client-server framework for a community based traffic sign repository creation. The main advantage of a community based approach is that the server side, where the central repository is kept, is able to keep track of potential damaged or occluded signs reporting these situations allowing for more directed inspections and eliminating the need for general periodic inspections.

The community based approach also allows the prompt and automatic insertion of new signs and in particular the maintenance of temporary signs such as those for road maintenance.

We have shown that a current medium range smartphone can perform as a client for this framework which means that in the near future every smartphone will have the necessary computing power to be used as a traffic sign report contributor client in this framework.

As future work we would like to use the framework itself to build an always increasing database of images that can be used for further training of the detection and recognition system increasing its accuracy.

We also would like to take advantage of the OpenStreetMaps API to be able to determine the size of the traffic signs based on the road classification, provide street names and speed limits, amongst other possibilities.

Another useful extension would be to detect road marks. Road marks are an important safety feature that also requires periodic inspection. Extending this framework would enable to assert the status of these marks avoiding on site inspection.

Acknowledgments

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013.

References

[1] T. Mccarthy, N. Maynooth, C. Mcelhinney, C. Cahalane, and P. Kumar, "Initial results from european road safety inspection (eursi) mobile mapping project," *Proceedings of ISPRS CRIMT, Newcastle*, 2010.

[2] J. L. Cardoso, C. Stefan, R. Elvik, and M. Srensen, "Road safety inspection-best practice guidelines and implementation steps," Technical report, deliverable D5 of the EU FP6 project RIPCORD, ISEREST, Tech. Rep., 2007.

[3] P. M. A. Broggi, P. Cerri and P. P. Porta, "Real time road signs recognition," *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 981–986, 2007.

[4] J. A. A. de la Escalera and M. Mata, *Traffic sign recognition and analysis for intelligent vehicles*. Image and Vision Computing, 2003.

[5] T. A. Y. Aoyagi, *A study on traffic sign recognition in scene image using genetic algorithms and neural networks*. 22nd International Conference on Industrial Electronics, Control, and Instrumentation, 1996, iEEE.

[6] K. Brkic, "An overview of traffic sign detection methods," *Department of Electronics, Microelectronics, Computer and Intelligent Systems Faculty of Electrical Engineering and Computing Unska*, vol. 3, p. 10000, 2010.

[7] G. Loy and N. Barnes, "Fast shape-based road sign detection for a driver assistance system," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2004, pp. 70–75.

[8] D. M. Gavrila, "Traffic sign recognition revisited," in *Mustererkennung 1999*. Springer, 1999, pp. 86–93.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001.

[10] R. Timofte, K. Zimmermann, and L. Van Gool, "Multi-view traffic sign detection, recognition, and 3d localisation," in *Applications of Computer Vision (WACV), 2009 Workshop on*. IEEE, 2009, pp. 1–8.

[11] D. Huang, C. Shan, M. Ardabilian, Y. Wang, and L. Chen, "Local binary patterns and its application to facial image analysis: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 4, pp. 1–17, Mar. 2011. [Online]. Available: http://liris.cnrs.fr/publis/?id=5004

[12] OpenCV User Guide, "Cascade classifier training," URL: http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html, accessed: 2016-11-14.

[13] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li, *Learning Multi-scale Block Local Binary Patterns for Face Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 828–837. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74549-5_87

[14] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. López-Ferreras, "Road-sign detection and recognition based on support vector machines," *IEEE transactions on intelligent transportation systems*, vol. 8, no. 2, pp. 264–278, 2007.

[15] J. Credi, "Traffic sign classification with deep convolutional neural networks," Master's thesis, Chalmers University of Technology, 2016.

[16] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, no. 0, pp. –, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608012000457

[17] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 2011, pp. 2809–2813.

[18] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, vol. 32, pp. 333–338, 2012.

[19] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[20] A. M. S. J. F. S. S. R. Madeira, L. C. Bastos and L. P. Santos, "Automatic traffic signs inventory using a mobile mapping system for gis applications," *International Conference and Exhibition on Geographic Information*, 2005.

[21] P. Arnoul, M. Viala, J. Guerin, and M. Mergy, "Traffic signs localisation for highways inventory from a video camera on board a moving collection van," in *Intelligent Vehicles Symposium, 1996., Proceedings of the 1996 IEEE*. IEEE, 1996, pp. 141–146.

[22] P. S. H. G. S. Maldonado-Bascon, S. Lafuente-Arroyo and F. Acevedo-Rodriguez, "Traffic sign recognition system for inventory purposes," *Proc. of IV, Eindhoven*, p. 590–595, 2008.

[23] S. Šegvic, K. Brkić, Z. Kalafatić, V. Stanisavljević, M. Ševrović, D. Budimir, and I. Dadić, "A computer vision assisted geoinformation inventory for traffic infrastructure," in *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*. IEEE, 2010, pp. 66–73.

[24] A. Chigorin and A. Konushin, "A system for large-scale automatic traffic sign recognition and mapping," *CMRT13–City Models, Roads and Traffic*, vol. 2013, pp. 13–17, 2013.

[25] E. Krsák and S. Toth, "Traffic sign recognition and localization for databases of traffic signs," *Acta Electrotechnica et Informatica*, vol. 11, no. 4, p. 31, 2011.

[26] IMT, "Sinalização vertical - características," URL:http://www.imt-ip.pt/sites/IMTT/Portugues/InfraestruturasRodoviarias/InovacaoNormalizacao/Divulgao\%20Tcnica/SinalizacaoVerticalCaracteristicas.pdf, accessed: 2017-05-26.