DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Android-based camera-based traffic sign detection in driving simulator data with OpenCV

Tobias Bauer

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Android-based camera-based traffic sign detection in driving simulator data with OpenCV

Android-basierte Kamera-gestützte Verkehrschilder-Erkennung in Fahrsimulator-Daten mit OpenCV

Author:	Tobias Bauer
Supervisor:	Prof. Dr. Uwe Baumgarten
Advisor:	Sebastian Eckl, M.Sc.
Submission Date:	15.02.2018

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.02.2018 Tobias Bauer

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Section

1.1.1 Subsection

2 Motivation

//Noch in Bearbeitung In present times advanced driver assistance systems, also called ADAS, are common features, even for middle classed driving vehicles. Fully autonomous cars are on the rise and probably only a matter of years before entering series production and running on public streets. These systems usually cover a certain kind of traffic sign recognition. Right now older or not fully equipped cars often lack these kind of extras, which induces a certain absence of relative safeness over newer cars. As it is not really economic to retrofit these features professionally, the question arises, whether you could add some of them to your vehicle without too much of circumstances, for example with a low cost device, most drivers already have, like an Android Smartphone mounted to the mobile phone bracket. In particular, because these devices have all the hardware, you need to implement this kind of features on, a camera, processors and a display to output the results. So it would actually make sense to have an application, that assists the driver in following the speed limit, when steering a vehicle.
Being open source is one of the major benefits of Android over other operating systems for Handheld devices, meaning the code is publicly visible and primarily it is free of charges. There are also other gratuitous components beneficial to implement such functionalities, like the OpenCV-library for computer vision and other open-source frameworks for machine learning, like Tensorflow or Caffe(2). So the question comes to mind, how well this low-cost kind of solution actually scores.

2.1 KIA4SM

Cooperative Intelligent Transport Systems (C-ITS) is a current research field, aiming to connect vehicles, mobile devices, traffic and transportation infrastructure[8]. The KIA4SM project wants to achieve cooperative behavior between independently designed and running systems participating in traffic. Therefore it is not inevitable to change each participating system to make communication possible, rather it is desired to generate a homogeneous platform for heterogeneous devices

3 Related Work

In this section i am going to inspect projects from third parties, who also did work related to recognizing traffic signs, explaining them and trying to point out which methods have been used and why.

3.1 D

3.1.1 Subsection

4 Artificial Neural Networks

Subsequent section gives a brief introduction into Artificial Neural Networks, covering general informations, the components of them and different types of Neural Networks

4.1 Fundamentals

4.1.1 General

Artificial Neural Networks, or simply Neural Networks, are certain kind of computing systems that are inspired by the functionality of Biological Neural Networks, for Example the brains of mammals'. ANNs are import representatives or techniques for modern Artificial Intelligence, although the idea of NNs is not as new as the recent hype over them, seen in the last couple years. Actually, the beginning of ANNs point back to the end of the 1940s, when D.O. Hebb created Hypothesis on neural plasticity [11] and these theories where actually being executed on digital calculators by Farley an Clark in 1954 [9]. Today's networks are basically trained by feeding data and telling them what the corresponding output should look like. So what has changed over the last couple of years, increasing the importance of NNs, is the massive amount of new data and a general increase in computing power[moore], these two major advantages over earlier years, may have been contributed to the new success of Artificial Neural Networks.

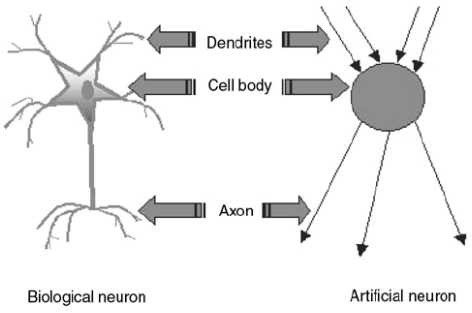

Figure 4.1: Biological and Artificial Neuron [4]

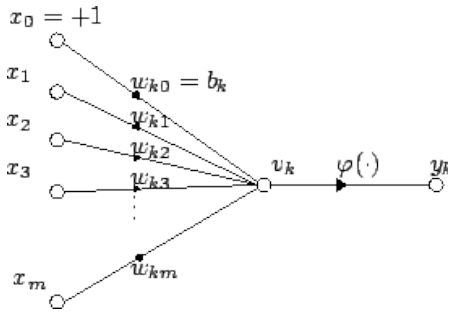

Figure 4.2: Single Artificial Neuron [1]

4.1.2 Neurons

An artificial Neuron is a mathematic function, aimed to imitate biological Neurons. As already mentioned, ANNs have several things in common with Biological Neural Networks, in this Analogy, an Artificial Neuron is a simplified abstraction of a Biological Neuron, both have a cell body and incoming respectively outgoing connections, called Dendrites and Axons [12] (Figure: 4.1). Every arriving signal over the Dendrite is first multiplied with it's corresponding weight, then added together. Be noted that x_0 is no input signal from a different Neuron, but rather is a constant input of +1 and is called the bias input.
After totalizing the input values, our signal is channeled through an activation- or transfer function (Figure 4.2). There are several possibilities for the transformation function to choose from, which aims to handle when the Neuron produces output and when it remains silent. Another reason would be to keep the values between certain boundaries, usually it is desirable to stay in range [0;1]. A Common transfer function would be the Sigmoid function (4.3) although there are diverse other reasonable choices like the hyperbolic tangent function. Catching up with above mentioned bias, it's purpose is to shift the activation function horizontally, the corresponding weight determines how much the function is being shifted. Coming back to previous analogy, the Sum-Operator in series connection to the Activation function form the artificial cell body.
After forming an output signal y_k (4.2), the Axon can fork into one or multiple branches, serving one or multiple Neurons as it's or their input dendrites, after they have been weighted by their corresponding new weights.

$$v_k = \sum_{i=0}^{m} x_i w_{ki} \tag{4.1}$$

$$y_k = \varphi(v_k) = \varphi(\sum_{i=0}^{m} x_i w_{ki}) \tag{4.2}$$

$$S(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}} \tag{4.3}$$

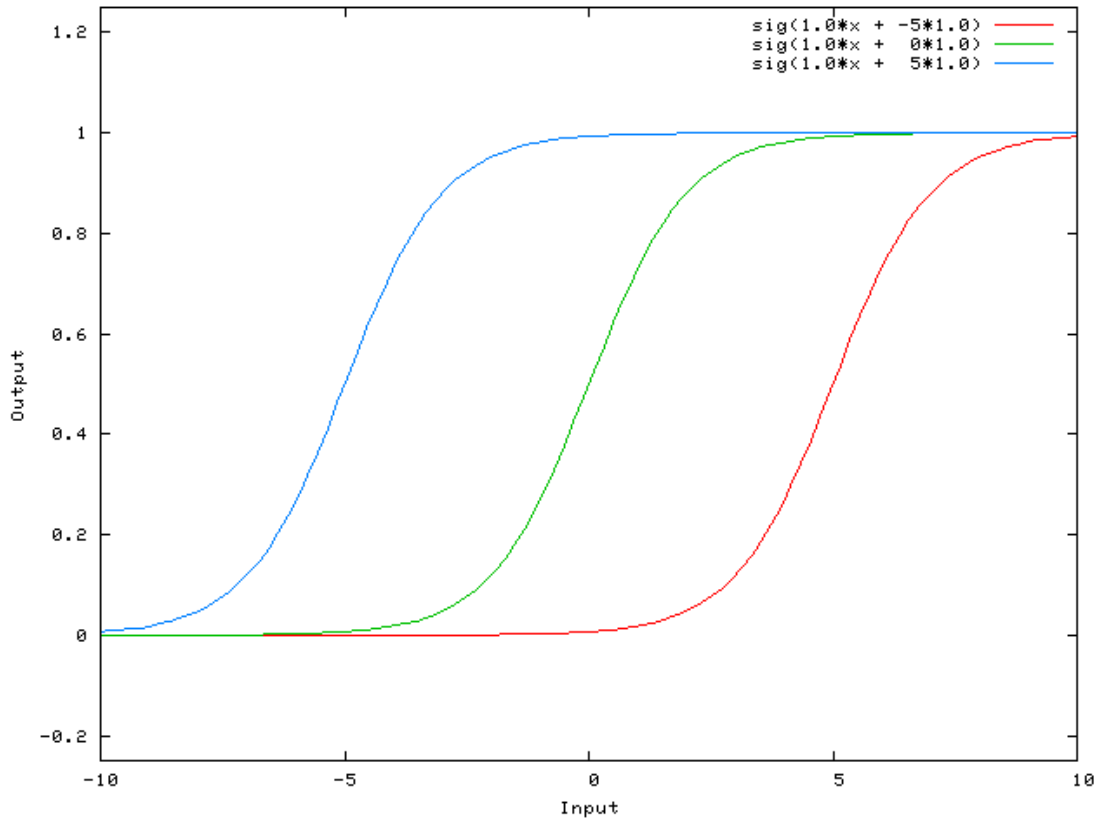

Figure 4.3: Sigmoid function with bias weight: 5 (blue), 0 (green), -5 (red)[2]

4.1.3 From Neurons To Networks

Feeding a signal or data through a Neuron and transporting resulting output further to the next Neurons basically forms a Network. Now as the major purpose of Artificial Neural Networks are to process any sort of data given to the ANN and produce some sort of output. Consequently a model needs to have two different kinds of Interfaces for importing and exporting data.
The input part comprises a so called Input Layer, containing several Neurons that can be fed with data from outside of the network. Respectively there is the other end of the ANN, that produces a result after the calculation, performed by our net. Every Neurons outputting data outside of the net is called output Neuron, these Neurons together form the Output Layer. The remaining Neurons between input and output are hidden Neurons, they can be grouped in one or more hidden layers, depending on the architecture, normally each Neuron of a layer l_n has a connection to every Neuron of

it's subsequent layer l_{n+1}.

4.2 Common Types

There are a lot of different types of Neural Networks, but digging deeper into the matter would definitely go beyond the scope of this discussion, so i am going to limit this section to describe more or less shallowly only the two main types of ANNs [12]. The type of architecture, which I used in my project is called Feed Forward Neural Network, in this approach data is only fed in one direction, meaning from the Input Layer successively to the Hidden Layer(s) and at some point to the Output Layer, forming no cycles or loops.
In contrast to this Recurrent Neural Networks are slightly more complex than FFNNs, because they add the directed cycles component, therefor these networks have internal memory to a certain extent, that is capable of forming dynamic temporal behavior.

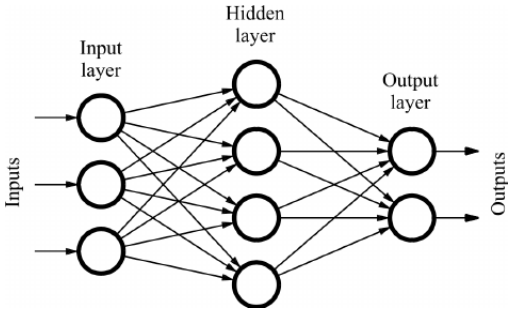

Figure 4.4: Feed Forward Neural Network [10]

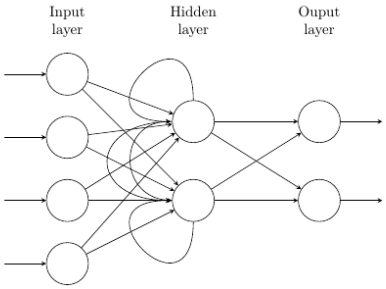

Figure 4.5: Recurrent Neural Network [15]

5 Concept

The task of looking at a picture and telling what is in it, is quite trivial to humans and would usually not pose a challenging task to perform, whereas it is not that simple to computers. In order to classify images, or more exactly the objects comprised, machines ordinarily need a certain amount of preprocessing to extract characteristic Features. As described in chapter 4 it is common practice to partition the many small steps of the process in two bigger, more generalized, steps of Feature Extraction and Classification. Our targeted devices are going to be any smartphones running Android as operating system, basically the app on the device should be able to handle live input from the camera, detecting and classifying in real time. In the following subsections I am going to break down in detail, what methods are used and how they work.

Komponentendiagramm

Figure 5.1: Original image[3]

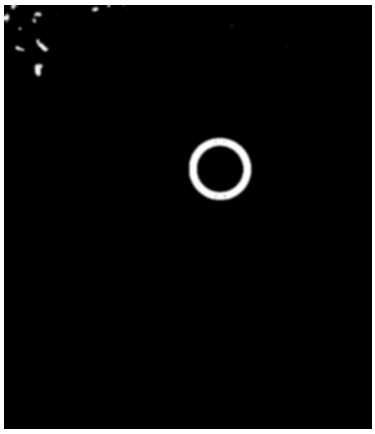

Figure 5.2: Filtered Image

Figure 5.3: Detected image

5.1 Detection

Before we pass data through to the Classification phase, it is a reasonable step in our Program, to specify a Region Of Interest, with regard to saving some processing power. In order to specify ROIs, I took advantage of the red circle around European Speed signs, the main steps of this part are to filter every red pixel out of the original image (Figure: 5.2), subsequently an algorithm checks the filtered image for circles (Figure: 5.3). Filtering red pixels from a picture is not that big of a sorcery, so i am going to skip that part for now and come back later to this in the implementation section. The real magic happens when you try to search an image for circles, therefore a common method is the Circle Hough Transform.

5.1.1 Circle Hough Transform

The Circle Hough Transform is a special case of the Hough transform, which determines lines in an image through Edge Detection Algorithms [13]. There are two slightly different Circle Hough Transforms, the first is only able to find circles of fixed radial size, the second is generally capable of finding Circles of any size. At first I am just going to start with the fixed radius form of the algorithm, subsequently we are going to extend that approach to a more universal algorithm.

Fixed Radius

First of all our image is being Converted to a binary image (Figure: 5.4), further it is being processed via edge detection to determine every visible edge contained in the photo (Figure: 5.5). After this, the actual core of the Algorithm is being executed. Therefore an Accumulator Matrix is initialized with zeros. As the radius of the circle is considered constant, every ring in the picture can solely be described by two variables (x and y coordinate), therefore the helping Accumulator Matrix is 2-Dimensional. Above-mentioned Matrix can be pictured as a grid over the original image, where single pixels can be combined to bigger baskets. Now every basket or pixel of a detected edge (Figure: 5.5) becomes the center of a new circle, with given fixed radius. Now each pixel that got "cut" by the circle gets incremented or up-voted in corresponding Accumulator Matrix. (Figure 5.6) Visualizes the voting steps, by displaying the green, red and blue circle as single examples of voting. The baskets or pixels with the highest votes can be considered centers of circles 5.7).

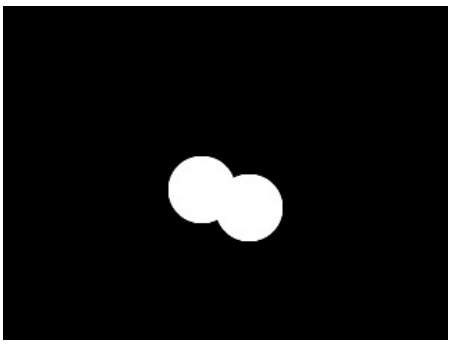

Figure 5.4: Binary image [17]

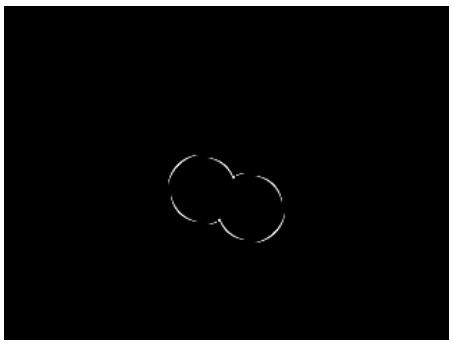

Figure 5.5: Edge detected [18]

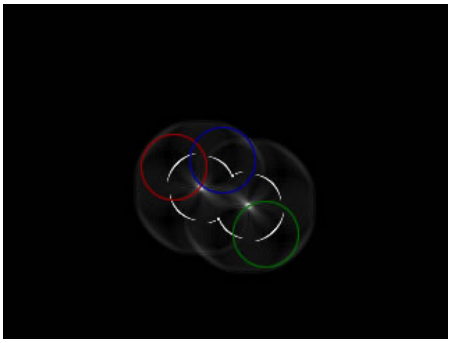

Figure 5.6: Voting process visualized[7]

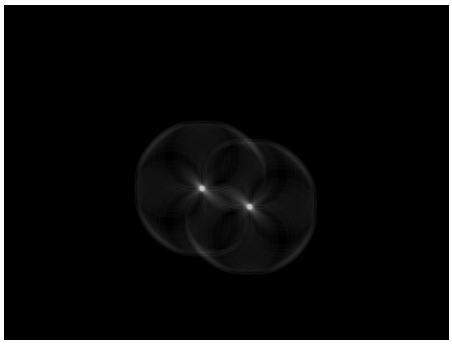

Figure 5.7: Voted image [16]

Variable Radius

A circle on a two-dimensional plane can be described by following formula[10]:

$$(x-a)^2 + (y-b)^2 = r^2$$

, meaning the circle can be represented by the parameters (a, b, r), where "a" and "b" are the 2-Dimensional coordinate values representing the center of the circle, whereas the variable "r" provides the radius of given ring.
The preprocessing (Figure: 5.4 and Figure: 5.5) is identical to the fixed-radius Circle Hough Transform, but what makes the difference now is, our Accumulation Matrix gains another Dimension, from having variable radii, therefore it is 3-Dimensional now. Usually you specify a range for the radiuses to make things a little easier. Now the voting happens exactly as it would with algorithm described before, but the difference now is that it has to be done with every radius in the before determined range. The center of the potential circles are now the ones with the most voted radii in all radius-dimensions. The desired radius of a point is the radius with the highest vote.

5.2 Classification

In my project, I decided the scope of signs should be from 10 to 90 km/h, to reduce the amount of input data, I had to provide, this range could be extended easily, by adding output Neurons and broaden the data set classes, used to train the Artificial Neural Network.

Preceding stage delivers an image of fixed size, that needs to be categorized into one out of 10 groups, these are Speed limits "10km/h", "20km/h",..., "90km/h" and "no sign". To gain these information about the sub-image of the original image, I decided to apply a Feed Forward Deep Neural Network. Said Neural Network is fed each individual pixel, provided by the ROI to its input neurons, the trained Model then outputs a prediction on what class the input picture might belong to.

5.2.1 Model

When coming up with determining my own model for classification purposes, it was relatively trivial to define the number of output neurons: They just had to be the number of different signs, plus an additional neuron for signalizing, given input image can not be classified as a sign, this leaves us with 10 neurons in our output layer. Generally the more neurons your model has, the more calculation operations have to be performed, when calculating output.
As the signs, I specialized on, were all circular, it made sense, to accept a square image as input. Therefore the number of neurons, included by the input layer had to be a square number, as they represent the pixels fed by the image. Comparable datasets/-classifiers, like NIST or MNIST[14] use 20x20 respectively 28x28 pixels. These datasets are established ones, that are used to train classifiers for recognizing handwritten digits. With regard to the more or less limited processing power of Android smartphones, I took over the 20x20 format, demanding 400 neurons for the input layer.
A NN is referred to be "deep", as soon as it contains more than one hidden layer on the inside. Since i wanted to create a deep Neural Network with the limitations of running it in real time on an Android device, I chose only two input layers. The amount of neurons in each layer was once again chosen with regard to the performance claims and set to 100 each.
What needs to be said about the simplified visualization of my model(Figure:5.8) is, not every connection between the neurons is present in the figure. Obviously every arrow to the next layer is symbolic for a connection from said neuron to every neuron from the subsequent layer, and of course i was not able to represent every node in each layer.

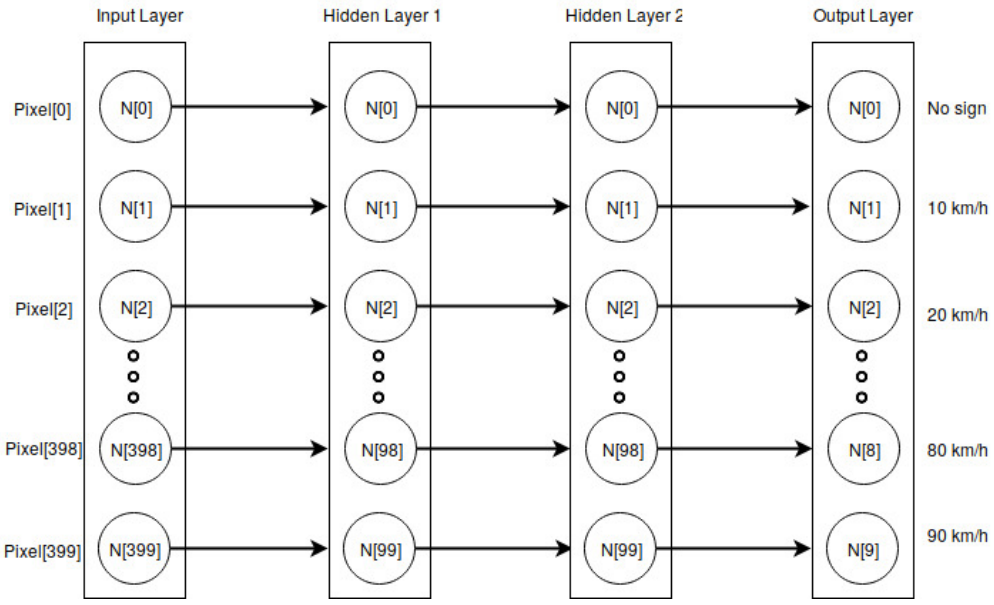

Figure 5.8: Simplified representation of the Neural Network Model

6 Integration into existing App

The App I had to build upon, already had some features related to the open source car driving simulator "Speed Dreams", but before we go deeper into theses functionalities, I am going to start with the present structure and Appearance. First of all, a dark theme has been chosen for the design of the App. As can be seen in Figure 6.1, the main screen contains different tiles for each task. It is possible to choose the source of intended input, therefore the user decides between loading an existing image (Figure 6.2), or hitting the camera icon to call a life-video feed of the back-facing camera. Depending on which input source has been chosen, the functionalities are applied either one single time or multiple times per second, both ways the outputs are printed onto the original images. Both features, that are named "Vehicle Detection" and "Lane Detection" are pretty much self explaining, but it should be mentioned that both methods are very likely trained by "Speed Dreams" data, meaning that the Lane detection is good for detecting lanes in the game, but might not be that brilliant in detecting real lanes, furthermore the vehicle detection recognizes only the back of the red car in the virtual rally course.

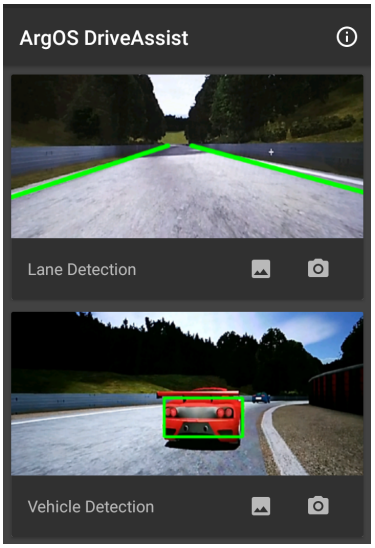

Figure 6.1: Main screen

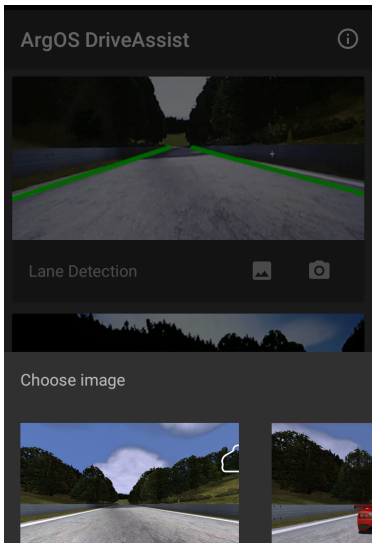

Figure 6.2: Load Image

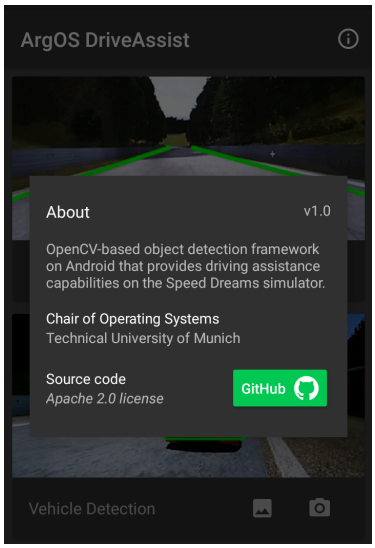

Figure 6.3: About

7 Implementation

Subsequent chapter is going to give a detailed view on how the methods described in section 5, were implemented. Moreover possible problems regarding the process are listed. As i took the android-application described in passage 6 as a basis, there is no reasonable alternative to further developing the app with Android Studio.

7.1 Native Development Kit

The Android NDK is a toolset that lets you implement parts of your app in native code, using languages such as C and C++. For certain types of apps, this can help you reuse code libraries written in those languages [5]. Moreover employing the Native Development Kit to your Java code can increase the performance of the application significantly [19]. I anticipated using the Native Kit might have a bigger workload, than Android's classic SDK, but I thought this to be alright, if the performance was superiorly compared to the approach written in Java.
First part, dealing with the detection of Speed Signs, doing image processing with OpenCV, was relatively simple in C++, compared to the mammoth task of building Tensorflow from source, loading a model and integrating integrating everything into the actual app. It actually was so time consuming, I did not think I could manage to make it work on schedule. A lack of supporting features like Code Completion in Android Studio, poor documentation from Tensorflow (at least for average developers) paired with proportionally sparse discussion board entries like "Stack Overflow", that often referenced different versions, led me to the turning point, where I gave up using the NDK for the SDK. In total, this excursion cost me about one to two weeks of my time, reserved for implementation.

7.2 Extracting Speed Sign Candidates

I implemented the algorithm in three languages: C++, Java and Python, every language serving different purposes. C++ and Java versions where necessary to operate for the application, first one running the Native approach, the second for the "standard" version. Last program written in Python was utilized for extracting data from images

collected from Google image search, running on my PC, in order to get training data to feed my Neural Network model (more about this topic later on). Nevertheless, apart from syntactical differences, every program basically does the same operations. For the sake of simplicity I am only going to present the python variation, as it is probably the easiest to read.
After reading the image, changing the colourspace from RGB/BGR to HSL/HSV is an optional, but reasonable step, because in the RGB/BGR space, all three Values of red, green and blue are correlated when it comes to changes in brightness [6]. Unlike that, HSL/HSV holds lightness or the value of brightness in a separate variable, which can lead to better results with respect to filtering colours under difficult lighting conditions. In the following we are going to stick to the HSV space, even though both work almost analogical, they differentiate each other from the arrangement of brightness.
Nevertheless filtering red pixels from our preferred colourspace works slightly different than just excerpting the Red channel of an image, as you would using RGB:

```
hsv_image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
lower_red = cv2.inRange(hsv_image, (0, 100, 100), (10, 255, 255))
upper_red = cv2.inRange(hsv_image, (160, 100, 100), (179, 255, 255))
red_combined = cv2.addWeighted(lower_red, 1.0, upper_red, 1.0, 0.0)
red_combined = cv2.GaussianBlur(red_combined, (9, 9), 2, 2)
```

Figure 7.1: Python code for extracting red pixels

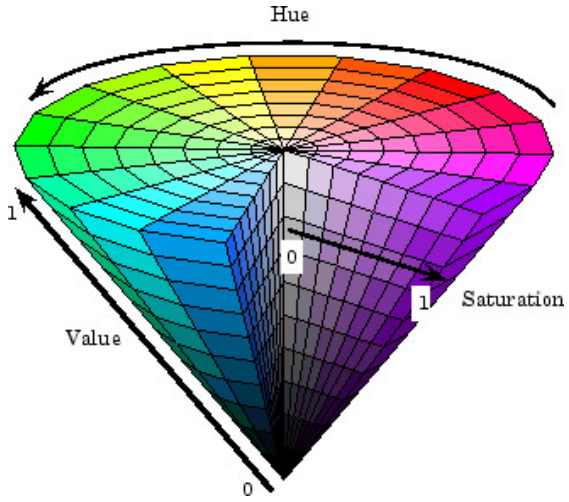

Figure 7.2: HSV colourspace

7.3 Neural Net Classification

7.3.1 Building Neural Network

8 Evaluation

8.1 Section

8.1.1 Subsection

List of Figures

List of Tables

Bibliography

[1] URL: https://en.wikipedia.org/wiki/Artificial_neuron#/media/File:Artificial_neuron.png.

[2] URL: https://i.stack.imgur.com/t2mC3.png.

[3] URL: https://upload.wikimedia.org/wikipedia/commons/1/1f/Fu%C3%9Fg%C3%A4nger_10_kmh.jpg.

[4] *Analogy-between-artificial-neuron-and-biological-neuron*. URL: https://www.researchgate.net/profile/Carmen_Galan2/publication/7947079/figure/fig1/AS:277631451844608@1443203904065/Fig-1-Analogy-between-artificial-neuron-and-biological-neuron.png.

[5] *Android NDK info*. URL: https://developer.android.com/ndk/index.html.

[6] H. Cheng, X. Jiang, Y. Sun, and J. Wang. "Color image segmentation: advances and prospects." In: *Pattern Recognition* 34.12 (2001), pp. 2259–2281. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(00)00149-7.

[7] *Circle Hough Transform explained*. URL: http://www.aishack.in/static/img/tut/circlehough_explanation.jpg.

[8] S. Eckl, D. Krefft, and U. Baumgarten. *KIA4SM - Cooperative Integration Architecture for Future Smart Mobility Solutions*. Apr. 2015.

[9] B. Farley and W. Clark. "Simulation of self-organizing systems by digital computer." In: *Transactions of the IRE Professional Group on Information Theory* 4.4 (Sept. 1954), pp. 76–84. ISSN: 2168-2690. DOI: 10.1109/TIT.1954.1057468.

[10] *Feed Forward Neural Network*. URL: http://www.aman.site/img/in-post/DeepLearning_neural_network/a-feed-forward-neural-network.png.

[11] D. O. Hebb. *The Organization of Behavior*. 1949.

[12] A. K. Jain, J. Mao, and K. M. Mohiuddin. "Artificial neural networks: a tutorial." In: *Computer* 29.3 (Mar. 1996), pp. 31–44. ISSN: 0018-9162. DOI: 10.1109/2.485891.

[13] D. J. Kerbyson and T. J. Atherton. *Circle detection using Hough transform filters*. July 1995. DOI: 10.1049/cp:19950683.

[14] *MNIST dataset*. URL: http://yann.lecun.com/exdb/mnist/.

[15] *Recurrent Neural Network*. URL: https://i.stack.imgur.com/KmrmP.png.

[16] *Result of Circle Hough Transorm*. URL: http://www.aishack.in/static/img/tut/hough_circle.jpg.

[17] *Sample circles for describing Circle Hough Transform*. URL: http://www.aishack.in/static/img/tut/circles.gif.

[18] *Sample edge detetction for describing Circle Hough Transform*. URL: http://www.aishack.in/static/img/tut/circle_edges.jpg.

[19] K.-C. Son and J.-Y. Lee. "The method of android application speed up by using NDK." In: *2011 3rd International Conference on Awareness Science and Technology (iCAST)*. Sept. 2011, pp. 382–385. DOI: 10.1109/ICAwST.2011.6163104.