NTNU
Det skapende universitet

Automatisk nummerskiltgjenkjenning for mobile enheter

Peter Køste Ringset

Master i datateknologi

Innlevert:	juni 2015
Hovedveileder:	Asbjørn Thomassen, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

ABSTRACT

This thesis will explore systems and current publications on automatic license plate recognition, and propose a new system that is designed for a mobile device implementation. Details of an actual implementation are also presented. Many automatic license plate recognition systems are made for stationary cameras, and may often fall short for a mobile device. The mobility of a device, such as a smart phone, gives some new constraints to consider. Among these are processing power, memory usage, angle of view, and distance to vehicle. Automatic license plate recognition is usually split into three significant parts, license plate detection or location, character segmentation, and character recognition. The proposed system uses a trained cascade classifier, with adaptive boosting classifiers, using the multi-scale block local binary pattern feature for license plate detection. Character segmentation is done using statistical methods, separating characters from background elements and noise. Lastly, character recognition is done using a multi-class support vector machine, based on feature vectors describing each character. The results for the presented implementation shows that automatic license plate recognition is feasible for a mobile device implementation. License plate detection and character recognition is done with an accuracy of 99 %, while character segmentation is done with an accuracy of 85 %. Combined average processing time for all three stages is 50 ms, which also makes the system feasible for live video. Moreover, character recognition is done at under 0.5 ms for each license plate.

SAMMENDRAG (NORWEGIAN ABSTRACT)

Denne masteroppgaven skal utforske systemer og nylige publikasjoner for automatisk nummerskiltgjenkjenning, og foreslå et nytt system som er designet for en mobil enhet. Detaljer rundt en implementasjon er også presentert. Mange slike systemer er laget for stasjonære kamera, og vil ofte ikke kunne fungere godt for en mobil enhet. En mobil enhet, f.eks. en smarttelefon, gir noen nye utfordringer å ta høyde for. Blant disse er begrenset prosessorkraft, begrenset minne, vinkel til kamera, og avstand til kjøretøy. Automatisk nummerskiltgjenkjenning deles ofte i tre deler; nummerskiltlokasjon, bokstavsegmentering og bokstavgjenkjenning. Det foreslåtte systemet bruker en trenet kaskadeklassifikator, med adaptiv boosting-klassifikatorer, basert på multi-scale block local binary pattern kjennetegn for nummerskiltlokasjon. Bokstavsegmentering gjøres ved statistiske metoder som separerer bokstaver fra bakgrunnselementer og støy. Bokstavgjenkjenning løses ved hjelp av en egenskapsvektorbasert støttevektormaskin, hvor egenskapsvektorene beskriver kjennetegn ved hver bokstav. Resultater for det foreslåtte systemet viser at automatisk nummerskiltgjenkjenning er mulig for en mobil enhet. Nummerskiltlokasjon og bokstavgjenkjenning gjøres med nøyaktighet rundt 99 %, mens bokstavgjenkjenning gjøres med nøyaktighet rundt 85 %. Total gjennomsnittlig prosesseringstid for systemet er 50 ms, noe som gjør at metoden også kan brukes på levende bilder. Det må også nevnes at bokstavgjenkjenning gjøres på under 0,5 ms per skilt.

It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years.

— John Von Neumann, circa 1949

ACKNOWLEDGMENTS

This masters thesis is done as a part of the Computer science masters program at the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU). The work described in this thesis is based on an idea from WTW AS. A masters thesis has also been completed in the spring of 2014 by Abraham Straume Bah, focusing on the same idea. Our work has benefited greatly from the work done by Bah, and we wish to extend our gratitude to him.

The project has been supervised by Asbjørn Thomassen, who has provided valuable advice on research methodologies and academic writing, and has provided valuable feedback during the process.

Additionally, I would like to thank some of the people closest to me; Siri, for warm words of encouragement and invaluable assistance at home during the course of this semester, and my father, for all his assistance during my time at NTNU.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

ADABOOST	adaptive boosting
AI	artificial intelligence
ALPR	automatic license plate reconition
ANN	artificial neural network
CCA	connected component analysis
G-DCD	global direction contributivity density
GPU	graphics processing unit
HSV	hue-saturation-value
L-DCD	local direction contributivity density
LBP	local binary pattern
LP	license plate
MB-LBP	multi-scale block local binary pattern
MLP	multilayered perceptron
OCR	optical character recognition
PNN	probabilistic neural network
RBF	radial basis function

RGB	red-green-blue
SCW	sliding concentric windows
SVM	support vector machine

Part I

AUTOMATIC LICENSE PLATE RECOGNITION

This thesis begins with an introduction to the subject of Automatic license plate recognition. A lot of related work exists on the subject, the core papers that support the work done in the thesis will be presented. Some detail covering background theory needed for the proposed method and the related research will be also be presented.

1 INTRODUCTION

1.1 MOTIVATION

Automatically recognizing and reading license plates is a task that has become more and more popular in recent years. The three most significant problems of the task are **1)** finding the location of the license plate (LP) within an image or a video stream, **2)** segmentation of the characters, and **3)** the recognition of the characters within the segmented regions, also known as optical character recognition (OCR). Anagnostopoulos et al. (2008) offers an extensive survey of the current systems as of 2008 that have been researched and developed. Another survey by Patel et al. (2013) covers some more recent publications. Many systems offer good results, where the success rate with test data often is claimed at 93% or better. Good results often come at the expense of processing time or generality. Likewise, the solutions that offer better computational complexity or generality often cannot produce results as good. As stated by Anagnostopoulos et al. (2008), research that exists on automatic license plate reconition (ALPR) is not being tested uniformly. This means that the test data may contain varying degrees of difficult test cases, and can lead us to wonder how these systems actually compare. Computational complexity will traditionally not vary significantly with the test data, but the success rates and generality of the ALPR-systems may. One of the motivating factors behind this thesis is therefore to evaluate different methods for the ALPR task, to see if we can draw any conclusions as to how the methods work in terms of both accuracy and efficiency. We note that accuracy addresses how *accurate* a method detects and recognizes LP text, while *efficiency* denotes how fast it is possible to complete the task. Accuracy is most often measured in percent, efficiency most often in seconds or milliseconds. Additionally we note that the words positive and negative are used to denote contexts where an LP is either present (positive) or not present (negative).

ALPR can be useful in many different contexts, some of them are automatic toll booth systems, traffic management systems, and parking fee systems. A specific use case is found in a parking fee system provided by a local company, WTW AS. The parking fee system works by having an end user downloading and using a mobile application, in which they enter the LP number of their car, and use the application to pay the parking fee for a given time period. Since the car does not have any visual form of inspection the parking fee inspectors have to use a mobile device that checks the parking fee system's database for each car they want to inspect. The parking inspector enters the LP text manually into the mobile device, and that task can be quite time consuming. Even if one LP can be manually entered in a manner of seconds, the total time spent entering LP numbers for an entire day can be very large. This provides another motivating factor for the system; to provide parking inspectors with a tool that lightens the load during their work day, and provide them with a tool that improves efficiency on the tasks that take large parts of the work day. We hypothesize that this task is possible to solve with current mobile devices, especially since mobile devices has seen a significant increase in processing power and battery capacity over recent years. We also hypothesize that using a mobile device will in some ways make the ALPR harder. The camera used in ALPR is often stationary, and this results in having images that has small variations in distance and viewing angle to the LP, and sometimes in illumination. With a mobile device one cannot make any such assumptions. A handheld device can capture images from any possible angle and distance, and can also capture images outdoors where there is a large variation in lighting conditions. ALPR systems are most often required to work on still images; but a more desirable solution for a mobile device would be with live video. By using live video we can envision a system where the user would only have to point the device's camera towards an LP, and the recognition would be done automatically and instantaneously. A future version of such a system could also include mounting a device onto a moving vehicle, and be able to do ALPR by simply driving past parked vehicles.

This thesis is intended as a continuation of work done in a masters thesis by Bah (2014). The masters thesis resulted in a proposed system, that was never completely implemented. The first part of the ALPR task (locating the LP) was never solved sufficiently, and subsequent parts of the system could never be tested. We will therefore try to build upon the research by Bah while evaluating the current research on the field, and in our work towards a working prototype.

1.2 RESEARCH GOALS

The research goals for this project, derived from the motivations are as follows:

A. Determine if ALPR is feasible for a mobile device, with regards to efficiency and accuracy. If possible, also investigate the possibility for a general purpose system, that is not dependent of the LP's design.

B. Find which methods are feasible for live video processing, with regards to efficiency and accuracy. It is also desirable to uncover if there is a tradeoff between accuracy and efficiency, and find a possible balance between the two.

1.3 CONTENTS OF THE THESIS

The thesis is split into three parts. Part i gives an introduction to the ALPR task, related background theory and related work. Part ii will focus on the creation of a working system, a prototype, and provide and discuss results. Lastly, part iii contains the appendices.

2 BACKGROUND THEORY

2.1 IMAGE PROCESSING METHODS

Many of the following methods shows how an image may be processed, most often as a matrix. A matrix I is used to represent the image, where $I(i, j)$ accesses the pixel value at row i and column j. The image coordinate system has its origin at the north-west corner of the image; row indices increases southward and column indices increases eastward. We will mainly deal with grayscale images in this chapter. Most of the mentioned methods are either designed for grayscale images or binary images. The expansion from grayscale to color is in most cases trivial, where applicable.

A technique called window scanning is often used in image processing. Window scanning allows for using an area of an image to analyze or process, rather than using just a pixel's value. The main principle is to use a small window (usually significantly smaller than the image), and position the window over the image, so that the window creates a mask over the image. The pixels contained within the window mask are considered, while pixels that fall outside are disregarded. This technique is present in many different methods, some of them are convolution, sliding concentric windows (SCW) and Haar-like features. Usually, the window starts in the north-west corner of the image, and is moved eastward. When it reaches the east edge of the image it is shifted southward one increment, and starts again at the west end of the image. The increment by which the window is moved can vary, but usually it is 1 pixel in both directions.

Three different color-models are used in this thesis, grayscale, red-green-blue (RGB), and hue-saturation-value (HSV). The grayscale color model represents an image with no colors, only different shades of gray. Pixels are usually represented with an 8-bit unsigned integer, giving 256 possible values. The RGB color model divides an image into three channels where each of them resemble a grayscale image. Where a grayscale image at it's brightest intensity is white, a channel in an RGB image will represent it's color's value at maximum intensity. The RGB color model is additive, meaning that a pixel is made brighter by increasing channel values. The HSV model is conceptually different than the RGB and grayscale model. Hue represents an angle in a color-wheel, saturation represents how much color there is, and value represents the lightness or darkness of the color. The color wheel is 360° where the angle is mapped to the wavelength of the color. This results in having red at 0°, green at 120°, and blue at 280°. Lowering saturation will produce more washed out colors, and a saturation of 0 will make the colors grayscale.

2.1.1 *Convolution*

Convolution is a mathematical operation usually performed on two functions. The convolution between functions f and g and is notated $(f * g)$. The result of a convolution is a third function, that represents the response found by projecting the response of the first function onto the response of the second function. Formally, it is notated as

$$(f * g)(t) = \int_0^t f(\tau) \cdot g(t - \tau) d\tau$$

where t may for instance represent time and f and g may represent two signals (Kreyszig (2006)). The Fourier transform can be used to solve the expression analytically. In the case of signals, a Fourier transform will take a function with time as its dimension and transform it so that frequency becomes its transformed dimension.

$$\mathscr{F}(f(t)) = \hat{f}(x) = \int_{-\infty}^{\infty} f(t) \cdot e^{-2\pi itx}$$

When transformed to the frequency dimension, a convolution becomes a simple multiplication.

$$(f * g)(t) = \mathscr{F}^{-1}\left(\mathscr{F}(f(t)) \cdot \mathscr{F}(g(t))\right)$$

In the case of image processing, we can view an image as a discrete two-dimensional signal, and this allows us to bring the convolution operation to images. In a two-dimensional discrete space, the two continuous functions are replaced with two two-dimensional matrices, and the formula becomes a little different (Gonzalez et al. (2009))

$$F(x, y) * G(x, y) = \sum_{n_1} \sum_{n_2} F(n_1, n_2) \cdot G(x - n_1, y - n_2)$$

In image processing the image will usually be one of the matrices, while a kernel, or a processing element, is the other. The kernel is defined from the operation one wants to perform on the image. Algorithm 2.1 shows the details of image convolution. The algorithm takes a grayscale image and a kernel as input and produces the convolved image. The kernel is most often square and must be of odd dimension. The basic flow of the algorithm is that the center of the

Algorithm 2.1 Image convolution with a kernel.

```
$I$ is an image with $n_1$ rows, $m_1$ columns.
$k$ is a kernel of size $n_2$ rows and $n_2$ columns.
$C$ is the result of the convolution.
1:  for $i = 0$ to $n_1$ do
2:    for $j = 0$ to $m_1$ do
3:      $o_x = i - \lfloor \frac{n_2}{2} \rfloor$
4:      $o_y = j - \lfloor \frac{n_2}{2} \rfloor$
5:      $a = 0$
6:      for $x = 0$ to $n_2$ do
7:        for $y = 0$ to $n_2$ do
8:          $a = a + \big(I(o_x + x, o_y + y) \cdot k(x, y)\big)$
9:        end for
10:     end for
11:     $C(i, j) = a$
12:   end for
13: end for
```

kernel is positioned directly over each pixel, and each element in the kernel is multiplied with the corresponding pixel in the image. The sum of these multiplications is the pixel value in the resulting convolution. This is done for all pixels. Note that since the center of the kernel is supposed to be aligned with an image pixel, we will have some trouble with the edges of the image. By for example placing the kernel at position $(0, 0)$, a part of the kernel will be outside the bounds of the image. A simple solution to this is to add additional pixels along the edges of the image. The kernel will still only process the original pixels, but the additional pixels will help the kernel cover an equal number of pixels throughout the convolution.

As with most matrix transforms, convolution also has an identity transform matrix. This is a zero-valued matrix with arbitrary size, and value 1 at the center

$$k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Convolution can often produce pixel values that are outside the defined range for pixel values, and therefore a normalization of the image may be required after convolution.

Gaussian blur

Convolution offers many important image operations, Gaussian blur being one of them (Gonzalez et al. (2009)). A Gaussian blur kernel is based on the Gaussian function

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Since a kernel matrix must be discrete, the Gaussian blur kernel will be an approximation of the Gaussian function

$$k = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.1}$$

Note that the sum of the elements in the kernel is 1. The Gaussian blur will make the value of each pixel a weighted average of its own value and the pixels surrounding it. The size of the kernel will determine the strength of the blur, a larger kernel will produce a more washed out image. The Gaussian blur is often used to remove unwanted noise or artifacts in an image before performing further processing.

Sobel kernel

The Sobel kernel is used for finding edges in an image (Gonzalez et al. (2009)). The kernel is an approximation of the gradient function of an image

$$\nabla f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y}$$

and will produce an image that has marked out vertical or horizontal edge characteristics. The Sobel kernel is approximated as

$$k_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, k_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.2}$$

where k_v is the vertical kernel and k_h is the horizontal kernel. An example of using the vertical Sobel kernel is shown in figure 2.1.

2.1.2 *Binary images*

When analyzing images, a binary representation of the image may be useful. A binary image is formally expressed as a matrix

$$I(i,j) \in \{0,1\}$$

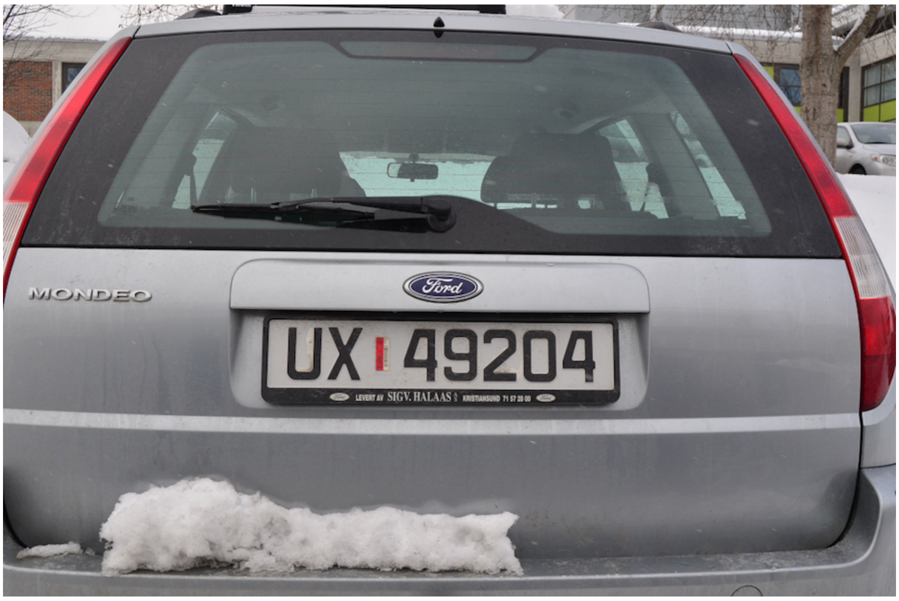
where each element in the matrix is either 0 or 1. The process of converting an image to a binary image, most often referred to as segmentation or binarization, consists of determining which of the pixel values should be 0 and which should be 1. In its simplest form, a histogram of the image—a counting of the number of occurrences of each pixel value—can be used to find a suitable threshold, so that every value under the threshold is set to 0, and every value over the threshold is set to 1. The key problem with thresholding methods is selecting the correct threshold.

Otsu's method

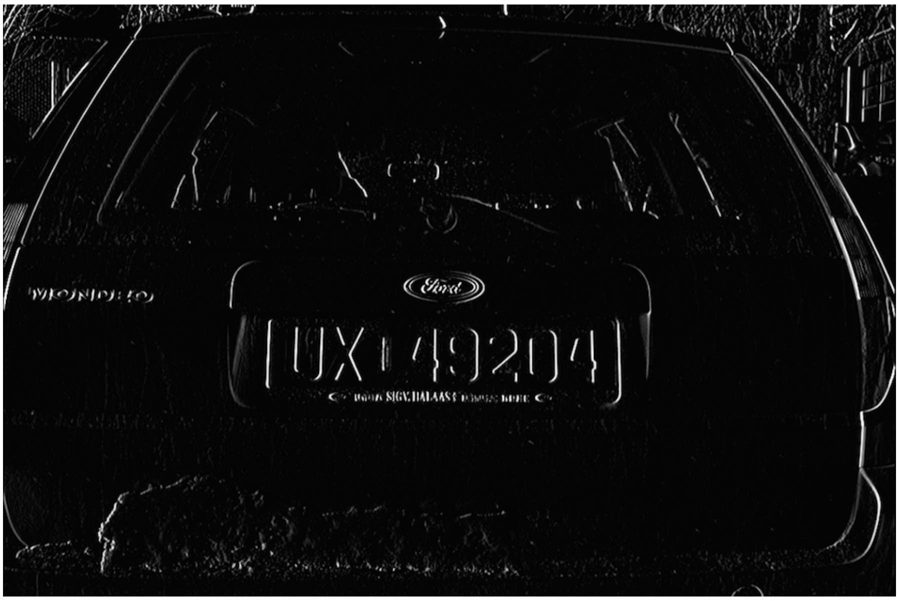
Otsu's method (Gonzalez et al. (2009)) is a clustering-based thresholding method, that is based on the assumption that the image has two classes of pixels; foreground and background. This is called a bimodal image, meaning that the histogram presents two obvious peaks. The method is based on analyzing the histogram, and finding the separation between foreground and background that minimizes the variance within each class. An example of a bimodal histogram is shown in figure 2.2.

The formula for the method is

$$\arg\min_t \sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$



(a) The original image.



(b) Image after convolving with vertical Sobel kernel.

Figure 2.1: Sobel kernel edge image example.

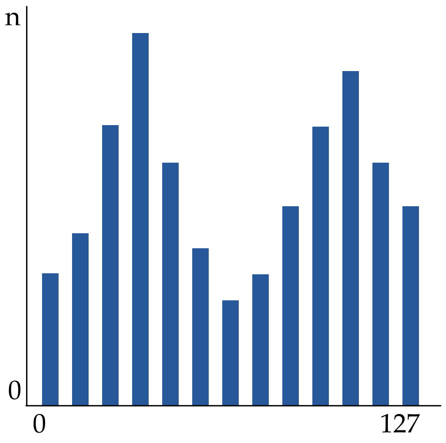


Figure 2.2: Histogram example.

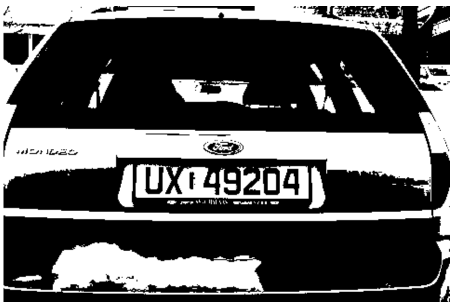


Figure 2.3: An example of using Otsu's method. The original image is the same as in figure 2.1.

where ω represents weights that denote the number of pixels on either side of the threshold. The goal is to find the threshold t that minimizes the expression. Minimizing the variance within each class is equivalent to maximizing the variance between the classes, and the expression becomes

$$\arg\max_t \sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) \tag{2.3}$$

This can be rewritten as

$$\arg\max_t \sigma_b^2(t) = \omega_1(t)\omega_2(t)\left[\mu_1(t) - \mu_2(t)\right]^2$$

An example of using Otsu's method is shown in figure 2.3.

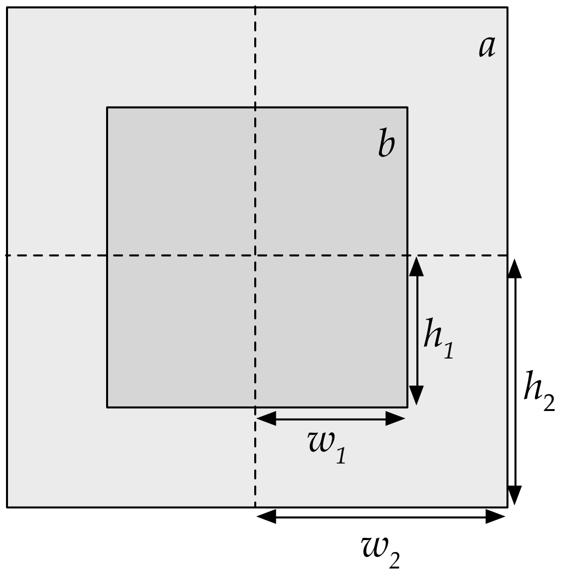


Figure 2.4: Sliding concentric windows example, adapted from Anagnostopoulos et al. (2006)

The computational complexity of Otsu's method is about $O(L^4)$, where L is the total number of intensity levels for a pixel. There are however several available improvements.

Sliding concentric windows

Sliding concentric windows (SCW) is a method developed by Anagnostopoulos et al. (2005). The method is based on using two differently sized windows where the center pixel of the first window is aligned with the center pixel of the second window, i.e. concentric. The center of the concentric windows scan the image, and compute statistical measures for the pixels overlapped by each of the windows. Figure 2.4 shows how the two windows are positioned over each other; note that the windows does not necessarily have to be of the same aspect ratio.

A statistical measure m (mean value or standard deviation) is calculated for both windows, a and b, and the relation between them is used to determine if the current pixel is foreground or background. If $\frac{m_b}{m_a} > t$ the pixel is marked as foreground, where m_a and m_b is the statistical measures of windows a and b, and t is an empirically found threshold. The pixels not marked as foreground are assumed as background. The computational complexity of SCW is $O\left(n \cdot m \cdot \left(w_1 h_1 + w_2 h_2\right)\right)$ where n and m is the size of the image and w_1, h_1, w_2, h_2 is the size of the two windows.

Connected component analysis

Connected component analysis (CCA) is an algorithm for labeling areas of objects in a binary image (Gonzalez et al. (2009)). There may be several areas in a binary image that are regarded as objects, and they have the property of having pixel values of 1, i.e. foreground, and being connected by either 4-connectivity or 8-connectivity. 4-connectivity means that pixels positioned north, south, west, or east are connected. 8-connectivity includes all of the 4-connectivity pixels and the four diagonally positioned pixels, that is, the 8 pixels that surround a pixel. In CCA however, we do not consider all of the pixels in the connectivity due to the iterative nature of the algorithm. The algorithm starts at position $(0, 0)$, and iterates through each element in each row. This means that the only pixels that needs to be considered are the two pixels that are placed diagonally north, and the pixel directly north and directly west of the current pixel, totaling 4 pixels. The complete procedure is shown in algorithm 2.2.

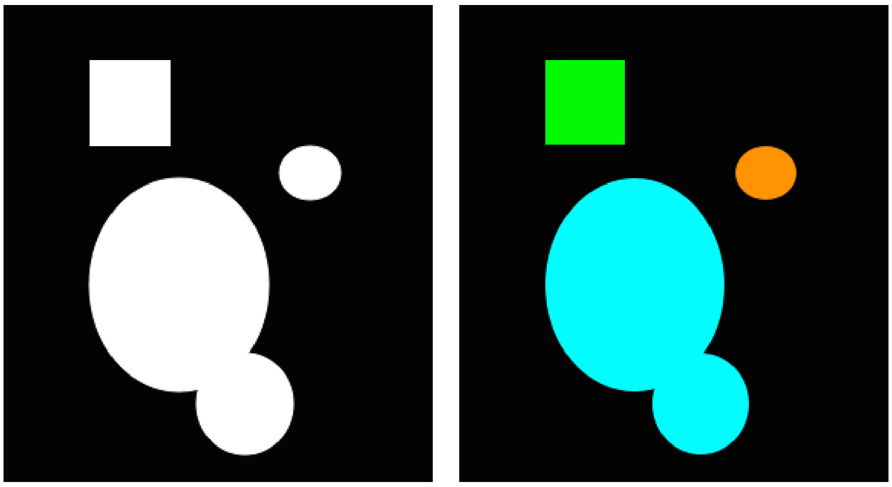
Figure 2.5 shows how CCA will perform on an example image. The colors in the image illustrate the different labels given to each component. The computational complexity of CCA is $O(n \cdot m)$ where n and m is the size of the image.

2.1.3 *Trichromatic imaging and color difference*

Trichromatic imaging theory is based on the fact that the human eye samples light in three different color-bands; red, green and blue. The theory is explained in Yang et al. (2012). When evaluating a pixel, the notion of difference can become important, and trichromatic imaging allows us to define a difference color that expresses the difference between two colors. Per trichromatic imaging theory a color can be defined from its red, green, and blue components, and gives the possibility to use these components when evaluating the difference. Suppose a target color $T = \{T_R, T_G, T_B\}$ and a back-

Algorithm 2.2 Connected component analysis for 8-connected pixels, adapted from Bah (2014).

```
1:  for each pixel $p$ in image $I$ do {iterate by row and column}
2:    $l = 0$ {the current label}
3:    if $p$ is foreground then
4:      if one and only one neighbor $n$ is foreground then
5:        assign $p$ same label as $n$
6:      else if a set $N$ of neighbors are foreground then
7:        assign $p$ same label as one of the pixels in $N$
8:        mark all labels in $N$ as an equivalence class
9:      else
10:       increment $l$
11:       assign $p$ the label $l$
12:     end if
13:   end if
14: end for
15: give each equivalence class a unique label
16: for each pixel $p$ in $I$ do
17:   assign $p$'s equivalence class' label
18: end for
```



(a) The original image. (b) Connected component analysis performed, and labels assigned to each component.

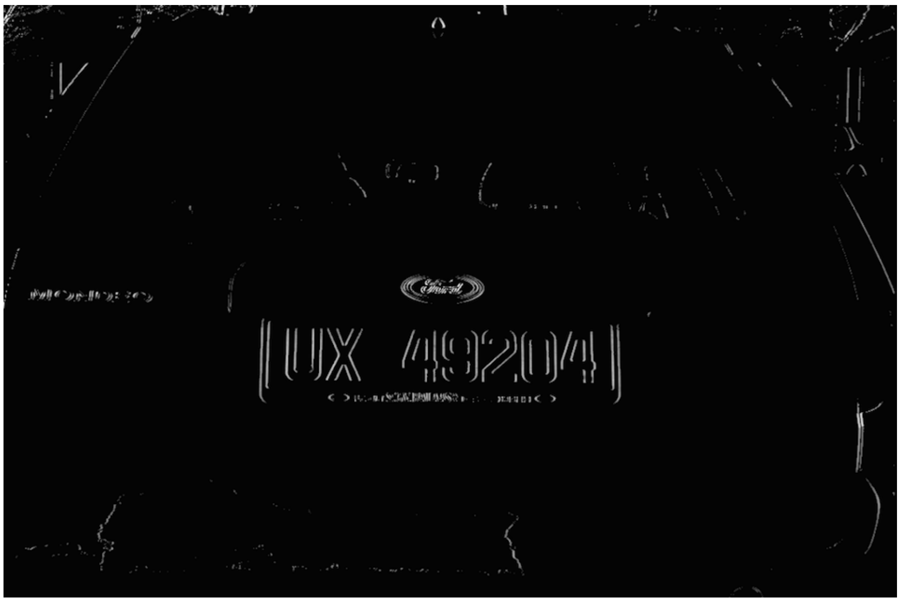
Figure 2.5: Connected component analysis example.

ground color $L = \{L_R, L_G, L_B\}$, then we can express the difference color $D_{TL} = \{R_{TL}, G_{TL}, B_{TL}\}$ by

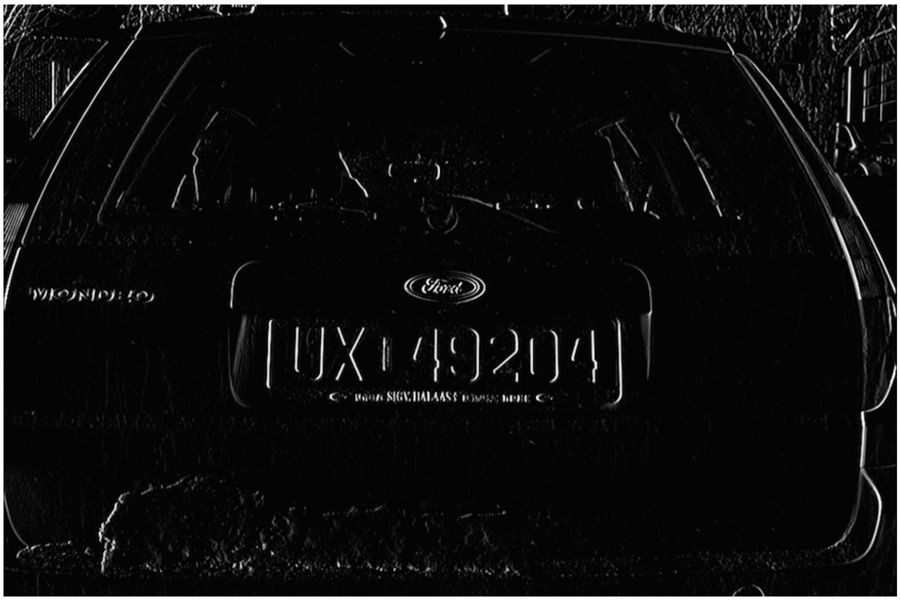
$$\begin{aligned} R_{TL} &= |T_R - L_R| \\ G_{TL} &= |T_G - L_G| \\ B_{TL} &= |T_B - L_B| \end{aligned} \tag{2.4}$$

The difference color can be used as a measure of how an image changes in a given direction by computing delta pixel values. Two pixels at locations (i, j) and $(i, j+3)$ can give some information as to how the transition between the two locations can be analyzed. Yang et al. (2012) explains how this can be used to compute an edge image of LPs where combinations of character color and plate color are known a priori.

The set of a priori color combinations produce a set of a priori difference colors, Ω. When evaluating an image the difference color for each pixel is calculated. This difference color is compared to all elements in Ω. If the difference color is deemed to be similar enough to at least one difference color in Ω, the corresponding pixel is marked as foreground. An example where the plate color is known to be white, W, and the character color is know to be green, G, gives a desired difference color $D_{WG} = |\{1,1,1\} - \{0,1,0\}| = \{1,0,1\}$. A row-wise iteration of a candidate image then finds each pixel's difference color by comparing with the pixel located three columns to the east, and checks if the difference color is similar to the desired difference color D_{WG}. We will not go into the notion of similarity here, other than to note that the HSV and the RGB color models are used. This offers a much more fine grained form of edge detection for images with good color conditions, as it is able to significantly narrow down the number of found edges. An example where the a priori color combinations is defined as black-white and green-black is shown in figure 2.6. As seen from the figure, the image produced by color difference edge detection contains far less noise than the Sobel image, and there is significantly less edge information that is not pertinent to the LP. Note that virtually all of the edge information that pertains to the LP is intact, and is more clearly defined.



(a) The color difference edge image.



(b) Sobel edge image.

Figure 2.6: Color difference edge image example.

2.1.4 *Haar-like features*

Haar-like features were first introduced by Viola and Jones (2001), and were originally created for the task of face recognition. The features are based on simple calculations within windows in an image. An example of a Haar-like feature is to divide a window into two parts, and checking if the sum of the pixel values in the one half is greater than the sum of pixels in the other half by some amount. Haar-like features are boolean valued, either true or false. Each feature will be very simple, but by combining a very large number of them, a more sophisticated type of recognition can be produced. To speed up the calculation of Haar-like features, integral images can be used. An integral image is an image where each pixel is the sum of itself and every preceding pixel. By preceding we mean pixels that are positioned north, north-west, or west of the pixel. Formally, an integral image can be expressed as

$$I_{\int}(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} I(k,l)$$

Since Haar-like features are based on comparing sums of pixels, the integral image can offer a substantial performance boost. The sum of pixels in a window with position (i,j) and size $w \times h$ can be expressed as

$$\sum_{k=x}^{i+w} \sum_{l=y}^{j+h} I(k,l) \Leftrightarrow I_{\int}(i+w, j+h) - I_{\int}(i+w, j) - I(i, j+h) + I(i,j)$$

The use of the integral image makes it possible to calculate each Haar-like feature in constant time, regardless of size. Figure 2.7 shows four examples of Haar-like features. In each example, the blue area within the window represents the area to be compared with the white area within the window.

2.1.5 *Multi-scale block local binary patterns*

Multi-scale block local binary pattern (MB-LBP) is a representation technique that divides a region into blocks. MB-LBP (Liao et al. (2007)) is an extension of local binary pattern (LBP). LBP is a feature that is concerned with a local 8-connected neighborhood of a

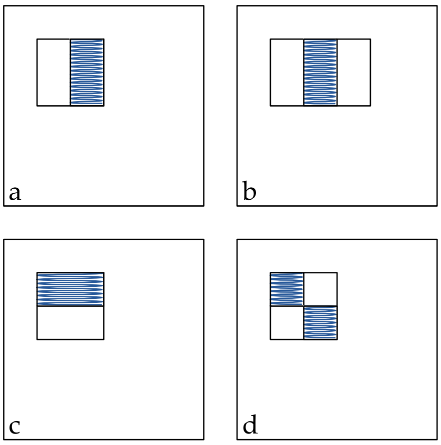
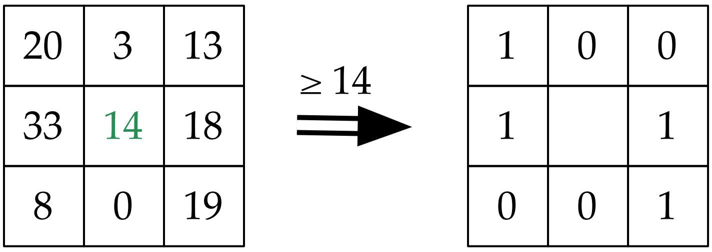


Figure 2.7: Haar-like features, adapted from Viola and Jones (2001).



The corresponding binary pattern of the neighborhood is 10011001_2 or 153 in decimal.

Figure 2.8: Local binary pattern example.

given pixel, and produces a signature for that pixel by comparing the pixel value to the neighbors. Each neighbor corresponds to a bit in a byte number, and the bit is hot if the corresponding neighbor's pixel value is greater than or equal to the given pixel's value. Figure 2.8 shows how the LBP signature of a pixel and it's neighborhood may be calculated.

MB-LBP uses the same principle over larger areas by averaging pixel values that fall within each block. An MB-LBP may have an arbitrary size, it is however a good practice to use sizes that are multiples of 3 in order to get the same number of pixels within each block. The use of an integral image will make the calculation of an MB-LBP possible in constant time. MB-LBPs are more robust than regular LBPs, since it is able to generalize larger regions. As image quality increases, a local 3×3 neighborhood will be increasingly insufficient to capture any patterns that may be present. Although the method allows for arbitrarily large areas, there will be some upper bound to the accuracy of the feature as it's size increases. Additionally, even though the method allows for large areas, it still possible to use all the local LBP features if that is desirable.

Training a classifier using MB-LBP is also regarded as more time efficient than Haar-like features, since a single feature has fewer degrees of freedom. While a Haar-like feature has one degree of freedom in each of it's size axes plus one degree per separation line, the MB-LBP has only two degrees of freedom. This dramatically reduces the feature space, and provides a much faster training process.

2.1.6 *Projection into vectors*

Projection is a wide term, in this context we use the term as a form of aggregation. Given that one has found a region of interest within an image, it is sometimes necessary to analyze some statistics about the region before determining if this is the region we are searching for. A projection vector can then be a useful tool, since it describes some key characteristics about the region. We mainly use vertical and horizontal projection vectors, but other orientations are also possible (Gonzalez et al. (2009)). A horizontal projection vector is a vector where each element in the vector is the sum of the pixel values in each of the rows in the region. Similarly, a vertical projection vector's elements are the sums of the columns in the region. Formally, the vertical and horizontal projections of an image I are

$$\begin{aligned} p_v &= \mathbf{1} \cdot I \\ p_h &= I \cdot \mathbf{1}^\mathsf{T} \end{aligned}$$

Figure 2.9 shows how vertical and horizontal projection vectors represents the image. As seen from the figure, the vertical vector can be a good estimation of where characters start and end, and the horizontal vector can estimate the baseline and cap height of the characters.

2.2 VIDEO PROCESSING

As stated by research goal B, we want to investigate if and how an ALPR system could be created by using live video. Live video is usually at least 25 frames per second. In this report we will mostly focus

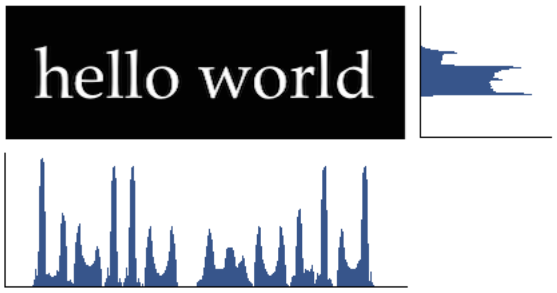


Figure 2.9: Vertical and horizontal projection vectors. The graph to the right of the image represents the horizontal projection, while the graph below the image represents the vertical projection.

on processing time with regards to live video. We intend to analyze each frame separately, and treat each frame in the same manner as a still image would be treated. This implies that the three stages of ALPR are still the same. It could be argued that there is some lost potential here, there is some information in the temporal dimension of the video stream that is not being utilized. For instance, a method of tracking objects (like an LP) across frames would have an easier task of locating the object in the next frame if there is some prior knowledge from previous frames on the object's most likely location and movement. Properties like these are considered to be outside the scope of this thesis, and will be disregarded. We do however acknowledge that the information found in the temporal dimension of a video stream is cause for further work.

The devices we will consider for our system has a normal frames per second rate of 30 (discussed more in section 4.1). This gives a maximum processing time of $\frac{1\text{s}}{30\text{fps}} = 33.33$ ms/frame. A system aiming to process every single frame of a video stream should however have some extra time in between frames, so a processing time of roughly 25 ms should be the maximum. Fortunately, there is another approach that may lessen the tight time constraint. By only analyzing every other or every third frame the system can give the user an impression of being live, while in reality being slightly behind. We regard this as an acceptable compromise.

2.3 ARTIFICIAL INTELLIGENCE METHODS

All of the presented artificial intelligence (AI) methods are classifiers, and are based on learning. Object recognition in images is a difficult task, and in most cases it is impossible to solve analytically. Learning-based methods will therefore provide the necessary adaption to the task, and most often it can solve the task satisfactory.

2.3.1 *Artificial neural networks*

An artificial neural network (ANN) is a graph, composed in several layers (Mitchell (1997)). Each layer contains a number of nodes, or neurons, and is connected to the next layer via edges, or synapses. Each edge has an associated weight which scales the data sent to the receiving neuron. The architecture of ANNs are inspired from biological neural networks, which model a central nervous system in a brain. Each neuron in an ANN has an activation function, which determines if the inputs to the neuron should cause it to send something to the neurons itself is connected to. Generally, an ANN has at least two layers, an input layer and an output layer. The input layer is where the neurons are given data to process, and the output layer is where the processed data is delivered. Between the input and output layers there may be several hidden layers. Most ANNs have at least one hidden layer.

An ANN is usually trained with the feed-forward back-propagation algorithm. The algorithm feeds labeled training examples to the network, and calculates an error of the classification the network produces. The error is fed backwards into the network, allowing the network to update the weights and activation functions of the network so that the error is minimized. This can be done by using gradient descent.

The activation function in the neurons can be of many different types, some of them are step functions, linear combinations and non-linear functions. A step function has a binary output

$$f(x) = \begin{cases} 1 & x > t \\ 0 & x \leq t \end{cases}$$

and can be useful in the output layer when a binary answer may be needed. A linear combination is a weighted sum of the inputs

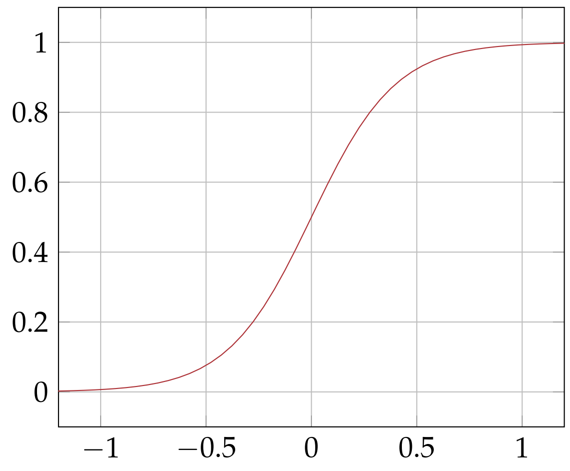


Figure 2.10: The Sigmoid function, here with $k = 5$.

to a neuron plus a bias value that is obtained through training. The Sigmoid function is an example of a nonlinear function that resembles the step function. The Sigmoid can model a range of uncertainty, which is impossible with the step function. In addition, its derivative is easy to calculate, which makes it easy to use when training the network with back-propagation and gradient descent. The function can be expressed as

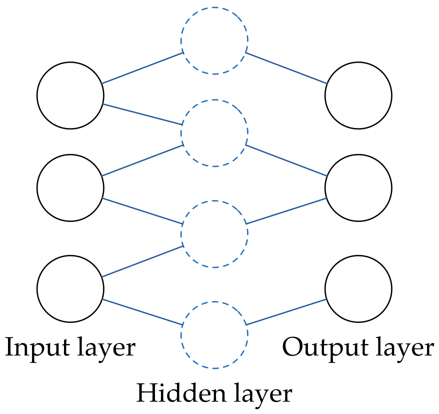
$$f(x) = \frac{1}{1 + e^{-kx}}$$

where k can be used to tune the slope.

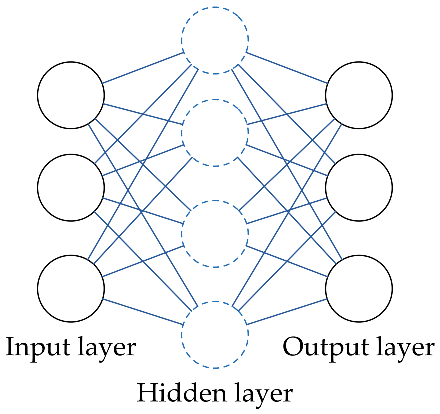
A multilayered perceptron (MLP) is a special case of an ANN, where each layer is fully connected to the next layer (Mitchell (1997)). The architecture of the networks makes it a good candidate for data that is not linearly separable, like the XOR function. We can tailor an ANN to a specific structure when we have some notion of how certain inputs correlate with certain classifications. An MLP can be better if there is a large amount of uncertainty around the relation between inputs and classification.

2.3.2 *Support vector machines*

Support vector machines (SVMs) are binary classifiers. The method is mathematically inspired, and is based on finding support vectors that separates a hyperplane optimally (Russell and Norvig (2003)).



(a) Regular artificial neural network.



(b) Multilayered perceptron.

Figure 2.11: Artificial neural networks.

The hyperplane represents the data space, and the separation gives the classification of new examples. Figure 2.12 shows how the data space can be optimally separated, so that the support vectors (the thin green lines) has a maximal distance from each other. More formally, the goal is to optimize the expression

$$\arg\max_{\alpha} \sum_{j} \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k)$$

where α represents the support vectors, $y_i \in \{-1, +1\}$ is the classification and x_i is a training example. Normally, the SVM is a linear classifier, meaning that a classification is found by inspecting a linear combination of an examples' properties. The function, or kernel, used by the SVM can be exchanged with a nonlinear function, like the Gaussian radial basis function (RBF)

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \tag{2.5}$$

A position paper by Bennett and Campbell (2000) details some of the advantages and shortcomings of SVMs. An SVM will not have any problems with local minima, as the method is guaranteed to converge on a global minima if it is given a sufficiently good training set. The results from an SVM are easier to reproduce, and SVMs are more independent of algorithm choice than other methods. It is also able to handle fairly large datasets, it's computational complexity is $O(D^2)$.

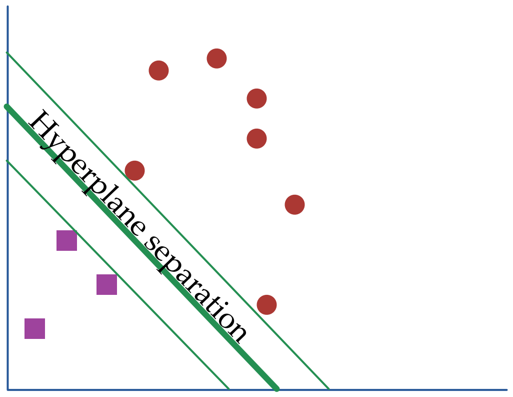


Figure 2.12: Support vector machine.

However, an SVM may in many cases not outperform a classifier that is hand-tuned for a specific domain; an SVM is not able to incorporate domain knowledge like other methods can. Additionally, it does not solve the feature selection problem. Using optimal feature sets can have a large impact on a method's performance, and selection of the most information-bearing features must be found separately before training an SVM. SVMs are inherently only capable of binary classification, but it is possible to construct either a single multi-class kernel for an SVM, or training an ensemble of SVMs (Hsu and Lin (2002)). One of the possible strategies is called *one against one* classification, which for k classes constructs $k(k-1)/2$ classifiers where each classifier is trained to distinguish between two classes. A voting among the classifiers is then the method for finding the correct classification. There are several other possibilities for solving the multi-class SVM problem, but this one is shown to be competitive, and has the advantage of producing output that allows for inspecting the confidence in a classification.

2.3.3 *Cascade classifiers and adaptive boosting*

Cascade classifiers is an ensemble learning method, and was first introduced by Viola and Jones (2001). It is based on concatenating a series of classifiers, where each classifier may be a weak classifier. The cascade is usually a binary classifier, having a positive and a negative classification. Only examples classified as positive are passed to the next classifier; an example is immediately discarded if it is classified as negative. Figure 2.13 shows how the control flow is executed. The nature of the cascade classifier means that the first classifiers in the cascade may be relatively simple or weak classifiers, since they will function as a filtering mechanism. Here, the term *weak* refers to the classifier's ability to classify examples; a classifier that slightly outperforms random guessing (around 50% correctness) is said to be weak, while a classifier that is correct most of the time is said to be strong. The early classifiers often produces a large number of false positives, that are discarded by later classifiers. This allows for a very fast classification of examples that are obviously negative, giving the method more time to focus on the hard examples.

The training of a cascade is done in a similar manner. A minimum hit rate and a maximum false alarm rate is defined for the training process. The minimum hit rate describes how many of the *positive* training examples that are required to be correctly classified, while the maximum false alarm rate describes how many of the classifications we can accept to be false positives. Usually the minimum hit rate is somewhere between 99.0 % and 99.9 %, and the maximum false alarm rate around 50 %. Stages are trained in turn, and is fed with the examples that were classified as positive by the previous stage. Each stage can have multiple classifiers, the stage in total is required to have a correctness rate for the positive examples that is greater than or equal to the minimum hit rate. Classifiers are added until the rate of false positives among the positive examples drops below the maximum false alarm rate. Usually this will result in having few classifiers in the first stages, and increasing numbers towards the end of the cascade. Each stage in the cascade reduces the rate of false positives, but it also decreases the true positive rate accordingly. The overall hit rate will be the minimum hit rate for each stage to the power of the number of stages, likewise for the false alarm rate. With an example of 99.5 % minimum hit rate, 50 % maximum false alarm rate, and 15 stages, the resulting cascade has an expected accuracy of $(99.5\%)^{15} = 92.76\%$ correct classification with a false alarm rate of $(50\%)^{15} = 0,003\%$ being false positives.

In some cases the classifiers in the cascade may be AdaBoost classifiers, which also is an ensemble learning method (Russell and Norvig (2003)). The AdaBoost algorithm is a variation of the traditional boosting algorithm. The method is based on having multiple weak classifiers, where each of the weak classifiers are assigned weights to account for their abilities. During training, the classifiers that are able to correctly classify hard training examples, i.e. training examples that the preceding classifiers failed to classify correctly, will be rewarded by a weight increase. This means that classifiers that seem to do correct generalizations in the domain are favored.

Viola and Jones (2001) describes using a cascade classifier with AdaBoost classifiers based on Haar-like features to do face detection. A modified version of the AdaBoost algorithm is used, the main variation being that each AdaBoost classifier will only rely on one feature. The steps are described in algorithm 2.3. Zhang et al. (2007) proposes an expansion of this work by using MB-LBP features instead of Haar-like features. Each of the MB-LBP features are used to create a multi-branch tree of classifiers with 256 branches, one branch for each possible value of the feature. Given a training set $(x_1, y_1), \dots, (x_n, y_n)$ where x_i is an image, x_i^k is the MB-LBP value of feature k with image x_i, and $y \in \{1, -1\}$, the classifier for feature k can be defined as

$$h(x_i) = \begin{cases} a_0^k & \text{if } x_i^k = 0 \\ \vdots & \\ a_j^k & \text{if } x_i^k = j \\ \vdots & \\ a_{255}^k & \text{if } x_i^k = 255 \end{cases}$$

where x_k is the k-th training example. The variable a_j is defined as

$$a_j^k = \frac{\sum_i w_i y_i \delta(x_i^k = j)}{\sum_i w_i \delta(x_i^k = j)}$$

where w_i is the associated weight of the instance, y_i is the correct classification, and $\delta(a = b)$ is a function that returns 1 if $a \equiv b$.

In both cases the possible feature space for each feature type is way to large to process, and so an AdaBoost algorithm is used in each stage to find a subset of the possible features.

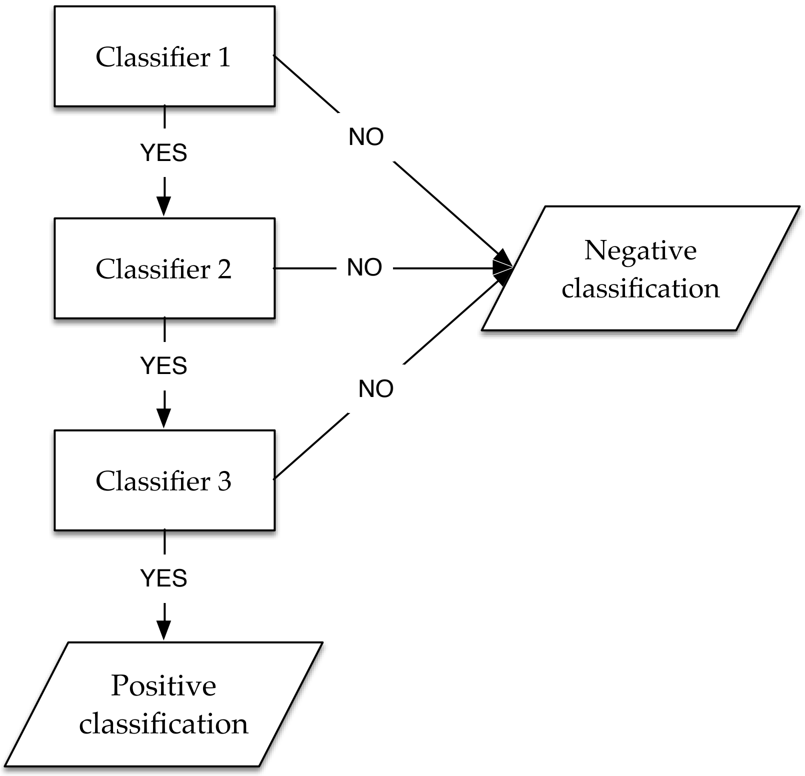


Figure 2.13: Cascade classifier control flow.

Algorithm 2.3 Cascade classifier AdaBoost algorithm, adapted from Viola and Jones (2001).

```
Training examples are $(x_1, y_1), \ldots, (x_n, y_n)$.
Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where $m$
and $l$ are the number of negative and positive training examples
respectively.
1: for $t = 1, \ldots, T$ do
2:     Normalize the weights,
           $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
       so that $w$ is a probability distribution.
3:     For each feature $j$, train a classifier $h_j$ which is restricted to
       using a single feature. The error is evaluated with respect to $w_t$,
       $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
4:     Choose the classifier, $h_t$, with the lowest error $\epsilon_t$.
5:     Update the weights
           $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
       where
           $$e_i = \begin{cases} 0 & h_i(x_i) \equiv y_i \\ 1 & \text{otherwise} \end{cases}$$
           $$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$$
6: end for
7: The final strong classifier is
           $$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(t) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
   where $\alpha_t = \log \frac{1}{\beta_t}$
```

3 RELATED WORK

This chapter will cover publications and papers related to ALPR. The chapter is divided in the same manner as the ALPR task, and the stages will be covered in order.

3.1 LICENSE PLATE LOCATION

The task of license plate location is the first task that needs to be solved in an ALPR system. One of the challenges of locating an LP is that the scene may be very complex, so that locating possible positions in the image, so-called regions of interest, is often a required tradeoff. Lighting conditions will typically vary greatly. The size, angle, and rotation of the LP within the scene may also be subject to many variations, especially considering that the target system is intended to run on a handheld device. In this section we will describe the approaches that have contributed to or inspired our ALPR system.

3.1.1 *Trichromatic imaging and color difference*

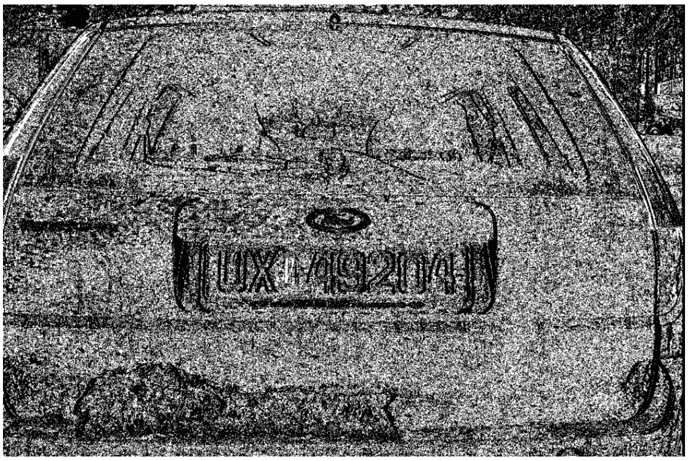
We know from several different papers that vertical edge information can be a key factor in locating an LP in an image (Abolghasemi and Ahmadyfard (2009); Kocer and Cevik (2011); Quan et al. (2009); Thome et al. (2011); Yang et al. (2012); Zheng et al. (2005, 2013)). The LP region will typically present as a distinct horizontal rectangle of dense edge information, allowing it to be a subject of recognition. The literature presents many different approaches as to how this region may be detected and located, most often including some sort of filtering mechanism to extract horizontal rectangle-shaped regions of dense edge information.

Trichromatic imaging is based on the theory that an image can be represented using the three color bands red, green and blue. Yang et al. describes how a difference color can be used to compare pixels, and how this comparison can help produce a color sensitive gradient edge image. Equation 2.4 on page 17 shows how the difference color is calculated, and how the difference color metric is used to produce a color-sensitive gradient image. Each pixel in an image is evaluated against the pixel that lies three increments to the east. A requirement of the method is to have a priori knowledge of the possible combinations of plate color and character color, so that the difference color for each distinct pair may be precomputed. Each pair of colors will constitute a rule, that the evaluated pixels in an image must fulfill to be marked as foreground. The rules are based on the relation between the red, green and blue components of the difference color, as well as defining acceptable ranges for the hue and saturation values. A rule representing the difference color for the white-red combination, $D_{TL} = \{R_{TL} = 0, G_{TL} = 255, B_{TL} = 255\}$, $h = \pi$, $s = 0$, where h is hue and s is saturation, can be

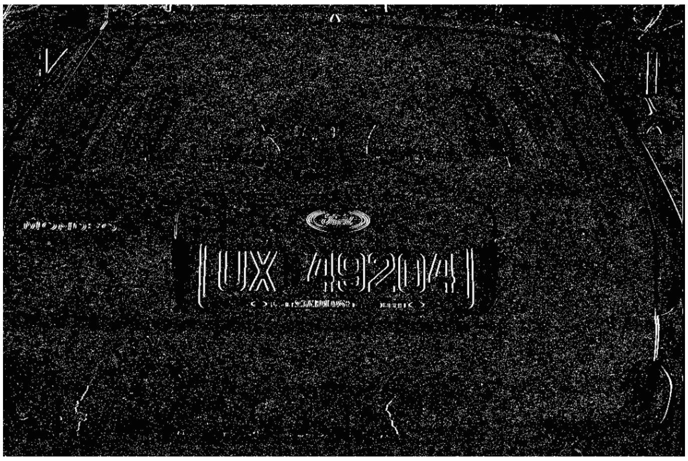
$$\begin{cases} \frac{G-R}{G} \geq 0.5 \\ \frac{B-R}{B} \geq 0.5 \\ \frac{5\pi}{6} \leq h \leq \frac{7\pi}{6} \end{cases}$$

The intensities of the RGB components of D_{TL} produces the first two clauses of the rule, the high intensity components G and B must have much larger values than the low intensity component R. The hue produces the third clause, the hue of the difference color defines an acceptable range for the evaluated pixel's hue. Note that each a priori difference color may correspond to several combinations of colors. The above example with the white-red difference color identical to the difference color for cyan-black.

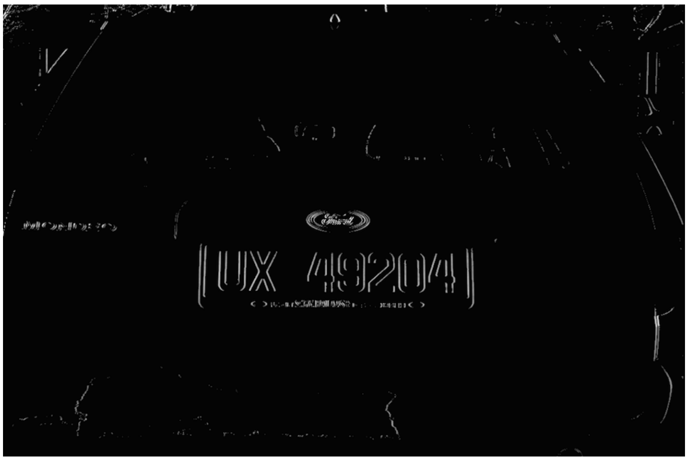
The algorithm for producing the edge image outlined by Yang et al. does however need some modifications to be able to produce edge images that are useable. The formulas provided for calculating hue and saturation does not produce the same results as a system implementation provided in the OpenCV framework. Secondly, the algorithm states that pixels that satisfy all of the given rules (6 in total) should be marked as foreground. By effectively binarizing the image at this stage a significant amount of noise is introduced to the image. By adjusting the HSV color calculation, and using the value from HSV as the pixel value we get an edge image with significantly less noise. Figure 3.1 show the results of these corrections.



(a) Edge image using the paper's equations for calculating HSV.



(b) Edge image using system framework calculation of HSV.



(c) Edge image using system framework calculation of HSV, and using v from HSV as pixel value.

Figure 3.1: The variations in implementation detail of color difference edge image.

The edge image now contains very little noise, and most of the edge information in the image is associated with an LP region. There is still however a need for some filtering before trying to find a rectangle that can be wrapped around a horizontally rectangular region. The image is row-scanned, and rows that contain more than n_1 edge regions where the inter-spacing is less than a threshold d_1 are labelled. n_1 is given as the minimum number of characters allowed in an LP, and d_1 is some empirically found value. After the row-scanning is done the labelled edges close to each other are fitted for a rectangle that tightly wraps the edge regions. Edges that are contained within a wrapping rectangle with a given aspect ratio are kept and marked as a region of interest, while those that fall outside are discarded.

Yang et al. tested the method on two test sets, one containing 600 (600$\times$330 pixels) different LP types from 104 different countries and regions, and the other containing 784 Chinese (768$\times$576 pixels) LPs exclusively. The success rates on the two test sets were 95.3 % and 93.1 %, respectively. The processing was performed on a Pentium 4, 2.5 GHz with 1 GB RAM, and is reported to execute in 54 ms on average for the first test and 63 ms on average for the second test set. A variation of the method was also tested, where a first pass using only two difference color rules was used on the Chinese test set. The full set of difference color rules was only used if the first pass was not able to find a region of interest in the image. The success rate of this variation was 94.1 %, and executed in 59 ms on average.

One of the things that make this paper particularly interesting is the method's ability to work on many different plate types. We also observe that the set of rules does not appear to have a very adverse effect on the execution time or the accuracy. The method can not be claimed to be color invariant, but the ability to handle a vast collection of different layouts is nonetheless very impressive. During our own implementation and testing we did however encounter some problems with the methods ability to detect edges in images with dirty LPs, or washed out colors. This will be presented and discussed further in chapter 6.

3.1.2 *Cascade classifier using Haar-like features and MB-LBP features*

Zheng et al. (2013) proposes an LP detection based on a cascade classifier, much like the cascade classifier by Viola and Jones (2001). The cascade used by Zheng et al. has some key differences; the first two classifiers in the cascade are not AdaBoost classifiers based on Haar-like features. Another important difference is that the Viola and Jones-cascade uses greyscale images, while the classifier proposed by Zheng et al. (2013) uses the vertical edge image representation for both training and classification. The first classifier in the proposed cascade calculates the vertical edge density and the second classifier calculates the edge density variance. The original cascade classifier is intended as a general purpose object detection algorithm, and using edge information in the general case would not be justified since there would be no guarantee that the object would present any significant edge features. However, in the case of ALPR we recognize that using the vertical edge information may lead to a higher detection rate, given the visual layout of an LP and it's tendency to contain edge-dense areas.

Another system using a cascade classifier is proposed by Bah (2014). This cascade classifier uses MB-LBP features instead of Haar-like features, and is based on the work described by Zhang et al. (2007). The use of MB-LBP features complicates the base classifier used in the AdaBoost classifiers. Each MB-LBP feature maps to a value, but the value does not bear any information in itself. The number will simply act as an index into an error function, which estimates how good a feature performs over the entire training set, and it is this function that makes the feature into a classifier. Unfortunately, Bah had some problems constructing a classifier that would produce reasonable results, and the work was never taken any further. Our initial tests with the OpenCV framework does however provide us with positive results. With standard settings for the classifier, using the same data set, we have produced a classifier that works as expected for the LP location problem. We therefore attribute the problems described by Bah to factors not related to the technique, but possibly to parameter settings or implementation details.

Zheng et al. (2013) reports having a detection rate of 96.4% over 169 LP regions. The paper does not give a specific execution time for LP detection, but reports that the entire ALPR task is executed in 100 ms or less on a Pentium 2.8 GHz. It is also pointed out that the algorithm has room for optimizations, so that the execution time may be reduced. Zhang et al. (2007) reports using a cascade classifier algorithm similar to the original algorithm by Viola and Jones (2001), but with some improvements in the performance metrics. The MB-LBP based cascade reportedly needs fewer features to perform at a similar accuracy level, and considerably less execution time to perform at a similar accuracy level. We do however note that the work by Zhang et al. is only tested on a face-detection test set.

3.1.3 *Sliding concentric windows*

Anagnostopoulos et al. (2006) proposes the use of SCW for detecting LP regions. SCW is performed on the image, and the pixels marked as foreground are used as a mask on the original greyscale image. The resulting image is then binarized using Sauvolas method, which is a local thresholding method. SCW is based on local neighborhood statistics, and so it stands to reason that the thresholding also is done with a local method. One of the key effects is that illumination variations within an image will not have converse effects on the marked areas from SCW. The resulting image is analyzed with CCA. Each component in the image is then tested for aspect ratio, orientation and Euler number. The orientation is calculated from the axis of least second moments, which is an estimate of a line that minimizes the squared distance to all points in the object. The Euler number represents the number of *holes* in the object, i.e. closed curve areas of background located within the outer borders of the object. The aspect ratio is required to be > 2 and < 6, the orientation $< 35°$ and the Euler number > 3. All components that pass these tests are regarded as regions of interest. A second pass of the algorithm is executed with the inverted image if no regions of interest are found during the first pass.

The method is tested on 5 test sets, totaling 1334 images. 96.5 % of all LPs are correctly detected, and the average execution time is 111 ms on a Pentium 4, 3.0 GHz with 512 MB RAM.

One remark about the method is that there is some sort of assumption on the size of the LP. SCW is performed with given window-sizes, and so there will be a lower bound and an upper bound for the sizes it is able to detect. SCW operates by comparing mean value or standard deviation of pixel values in the two windows. The relation between the statistics for the windows determine that the pixel at the concentric point should be marked as foreground if the relation is bigger than some defined threshold. The threshold is reported to be found by trial and error, and is another point for concern. A threshold like this will most often not work well if set to a specified value, it is in fact reported that the user controls the threshold while operating the system. To be able to support different sizes for LPs it would be possible to do several analyses of an image with differently sized SCW windows. It is however important to note that the time complexity of the algorithm is quadratically proportional to the size of the windows, and executing many passes of the algorithm with different window sizes will have significant effects on the execution time.

3.2 CHARACTER SEGMENTATION

The next step after identifying some region(s) of interest is character segmentation. Character segmentation aims to identify and extract character regions within a region, or possibly discard a region based on not finding suitable character segments. This stage of the ALPR task is conceptually different from the other two stages in that an AI method for segmenting characters is not widely used or accepted as a fruitful course of action. Most methods base the character segmentation on a form of binarization method, and some utilize additional statistical means to estimate character boundaries.

3.2.1 *Projection boundaries and blob detection*

Zheng et al. (2013) describes a method using projections, binarization and CCA to segment characters. A color correction step is done first to correct for dark foreground on light background or vice versa. Equally spaced horizontal lines are placed over the image, each line counting the number of pixels above a certain intensity value along the line. The image is inverted if the averaged number of counted pixels along each line lies above a given threshold. This results in having a grayscale representation of each region of interest where the character color (i.e. white) can be assumed to be lighter than the plate color (i.e. black). The algorithm proceeds by using the vertical edge image of the region to create a horizontal projection vector. The vector is used to estimate the baseline and the cap height of the characters, and crops the region accordingly. The region is then binarized using Otsu's method, and a vertical projection vector is created. The vertical projection estimates character regions, and each character region is analyzed with CCA. Each of the components is validated with a set of rules and actions:

- If the upper boundary is different from the *overall upper boundary*, the component is discarded[1].
- If the inverse aspect ratio $\left(\frac{1}{\text{aspect ratio}}\right)$ is > 0.8 or < 0.3, the component is discarded.
- If the area of the component is more than one sixth of the whole image size, it needs further segmentation[2].
- If any two neighbor segments' widths are smaller than 66.66 % of the average character width, the segments are joined together. However, if the joined segment has a width 33.33 % greater than the average character width, the joining is not done after all.

The segments still present after the above set of rules and actions is performed is accepted as character segments, and passed to the character recognition stage.

The approach was tested on a test set of 587 LP images with 3502 characters, where all of the characters were correctly segmented. The method did however also find some false positives, giving a total accuracy of 98.82 %. As noted in the previous section, this paper does not account for time spent on each stage in ALPR, but states that the entire ALPR task is executed in less than 100 ms, using a Pentium 2.8 GHz processor.

3.2.2 *Sliding concentric windows and statistical methods*

Anagnostopoulos et al. (2006) describes using SCW for character segmentation. The region of interest is resized to a predefined size (228×75 pixels) and then SCW is run on the resized region. During this step, the windows are sized to match the expected aspect ratio of each character segment, so that the windows will more easily capture the statistical variations that occur between the foreground

1 *Overall upper bound* is not defined in the paper.

2 What *further segmentation* entails is not described in the paper, but we imagine that this may have to do with changing the threshold found by Otsu's method for binarization

and the background. The resulting image is analyzed with CCA, and components that have a sufficient height (> 32 pixels) and are oriented upright ($> 75°$) are kept. A vertical and a horizontal projection vector is then calculated from the final components. The standard deviation of each of the vectors then define the start and end points of each character segment along their respective axes.

The method was tested on a Pentium IV, 3.0 GHz with 512 MB RAM, and correctly segmented 89.1 % of 1334 LPs with an average processing time of 37 ms. The success rate also includes the character recognition stage, and so it is expected that the true success rate is at least as high.

3.2.3 *Tilt correction*

None of the above mentioned methods try to correct any tilt or skew in the regions of interest. During our initial tests we found that most images present some tilt, and correcting this can have a positive effect on character recognition. It could be argued that the character recognition stage should handle this. Unfortunately, the information needed to perform a tilt correction is lost when character regions are extracted from a region of interest, and so it is prudent to take this into account during character segmentation.

Quan et al. (2009) proposes using an improved Hough transform (Gonzalez et al. (2009)), which is used before segmentation to correct tilt. The region of interest is analyzed both horizontally and vertically to find slopes for each of the axes, and right align them before proceeding with a character segmentation algorithm. It is important to note that this method addresses shear and not orientation. This enables the method to correct regions that are skewed because the image was captured from an angle and not straight ahead. Another method is proposed by Bah (2014), where baseline estimation is the foundation of the method. The baseline is found by linear regression on each character segments' lowest point. The baseline becomes the bottom line in a rotated rectangle that should enclose each character segment, and a vertical projection vector is created for this rotated rectangle. The projection vector is used to estimate character start and end points, their heights are already given from the enclosing rotated rectangle.

3.3 CHARACTER RECOGNITION

Character recognition, often referred to as OCR, is the final stage in ALPR. One of the main advantages of this stage is that the research of OCR is not limited to ALPR. A vast corpus of literature exists on the subject, relating both to the most popular task of digitizing books and other printed material, but also from natural scenes. In this section we will present methods that relate to ALPR, and methods that show promise for an actual implementation.

3.3.1 *Tesseract OCR*

Tesseract OCR is an OCR engine that is maintained by Google (Google Inc. (2015)). The engine is regarded as one of the leading OCR tools available, and is used in many different contexts. It's first step is to perform CCA and organize components into lines of text. Each line is separated into words based on the components' interspacing, and each word is passed to a two-pass recognition process. The first pass attempts feature extraction and character recognition on each word, and words that are deemed as satisfactory are passed as training data to an adaptive classifier. The adaptive classifier is used for character recognition lower down on the same page. The second pass is executed after the whole page has been processed once, utilizing the fully trained adaptive classifier.

Zheng et al. (2013) describes using the Tesseract software with very good results. They do however note that some modification has been made to the engine, but not how the modifications are done. The reported success rate is 100 % for all correctly segmented character regions.

It is also important to note that Tesseract is trainable, and that it is possible to define dictionaries that describe the possible language one seeks to do OCR in. From the online documentation (Google Inc. (2015)) we also note that the engine is usually trained for certain fonts, allowing it to achieve a higher accuracy rate.

3.3.2 *Feature extraction for Support Vector Machines*

Using some extracted features instead of the actual character segment is an often used technique in OCR (Capar and Gokmen (2006); Kocer and Cevik (2011); Quan et al. (2009); Smith (2007); Thome et al. (2011); Wen et al. (2011)). The intension is that a vector of features may contain more discriminating information than the image will, and so it is possibly easier to base the training and classification on the vector of features. There are many different types of feature one can extract, some of the most interesting are described by Wen et al. (2011). Four different types of feature vectors are described:

G-DCD FEATURE The global direction contributivity density (G-DCD) feature scans the image from left to right. Whenever a transition from background to foreground is detected it calculates the angle of the boundary, and the angles are stored into a vector. The vector is divided into parts, where the sum of each part appended to the original vector constitute the feature vector. This feature vector can provide information about how a character's shape is distributed over the image.

L-DCD FEATURE The local direction contributivity density (L-DCD) feature is similar to the G-DCD feature. A character segment is divided into sub-regions, and a single-valued G-DCD calculation is performed for each sub-region. Each sub-region's value constitute the feature vector.

CONTOUR FEATURE The character region is scanned along equally spaced scan lines, horizontally and vertically. For each scan line, the distance to the *k*-th occurence of a transition from background to foreground is recorded. The distances constitutes the elements of the feature vector. This feature can give us information about how the character is laid out in the region, and for increasing *k* it can give information about how complex shapes are laid out.

PENETRATED FEATURE The number of transitions from background to foreground are recorded along equally spaced scan lines, horizontally and vertically. The number of transitions for each scan line constitutes the elements of the feature vector. The penetrated feature is a simplification of the more recognized projection vector.

While some of these feature may not be very easy to grasp, the results presented by Wen et al. are more easily interpreted. The different types of features was tested on numerals and Chinese characters using an SVM with an RBF kernel. For Chinese characters the best performance was found using the contour feature with an accuracy of 98.79 %. The numerals achieved the best performance with the contour feature with an accuracy of 98.41 %. The contour feature is reported to have an average processing time of 5 ms per character. We also note that the fastest feature, the penetrated feature, has an average processing time of 1 ms and an accuracy of 97.55 % on english characters.

3.3.3 *Artificial neural networks*

ANN is another popular approach in OCR. Different systems vary between using feature vectors and the character region as input to the network.

Anagnostopoulos et al. (2006) describes using a probabilistic neural network (PNN) for OCR. PNNs are a special case of the more known MLP, where each layer of the network is fully connected with the next layer. PNNs are usually has one hidden layer, where each neuron represent one training example, and has a probability distribution of how a sample correlates with the the training example. This means that the method is quite memory intensive compared to other ANN flavors. The speed is also surpassed by other types of networks, since the network can become quite large and require very many computations per classification. Still, the method described by Anagnostopoulos et al. achieves an 89.1 % accuracy, with an average processing time of 128 ms. As in the previous section, this accuracy also represents the success in character segmentation.

Kocer and Cevik (2011) describes using the more traditional MLP networks, one for letters and one for numerals. Letters and numerals are separated in order to avoid uncertainty around characters that have similar features, such as O and 0, Z and 2, I and 1, and B and 8. Each character is described with a feature vector that represents the characters' average absolute deviation metric, a metric that is described with the equation

$$V = \frac{1}{N}\left(\sum_{N} |f(x,y) - m|\right)$$

where $f(x,y)$ is the pixel value at location (x,y), m is the mean value of the pixels and N is the number of pixels. The images were divided into blocks, the average absolute deviation for each block would be an element in the image's feature vector. Having letter images divided into 6×8 blocks, and numeral divided into 7×7 blocks, the resulting feature vectors has 48 and 49 elements respectively. The networks are reported to be three-layer networks, having 1 hidden layer. The number of nodes in each layer is not reported, but we surmise that the input layer has as many nodes as the feature vector, and the output layer as many nodes as the number of letters and numerals. With a test set of 347 letters and 1022 numerals, the method was able to correctly classify 344 letters and 1000 numerals (98.17 %). No processing time is given, but a general knowledge of ANNs leads us to believe that classification of characters can be done with satisfactory speed.

Part II

THE PROTOTYPE

The second part of this thesis is focused on the work done and it's results. We will begin by presenting some key aspects of the chosen platform and the frameworks and libraries that have been used to create the prototype. Moreover, we will also give an overview of the system model to offer insight into which algorithms and techniques are used. Finally, we will present our tests and results, and discuss the results.

4 MOBILE PLATFORM: IOS

The work by Bah (2014), which this thesis builds upon, was originally intended for the Android platform. The plan to implement some software for the Android platform was however abandoned due to time constraints and lack of experience with the platform. The author of this thesis is an experienced iOS developer, and we believe that this will aid greatly in the implementation of a prototype system for the iOS platform. When choosing between available mobile device platforms we have only considered the Android and the iOS platform. The Windows Phone platform could be viable in the future, but is as of yet not mature enough and has a far too low market share. Recent data[1] shows that the Windows Phone market share is 3.5 % worldwide. From a practical standpoint this makes the platform unattractive, especially with regards to realizing the system in a real-world application.

Aside from the advantage of having experience, there are some other justifications for choosing the iOS platform. Mobile device operating systems are traditionally updated regularly (see table A.3 in appendix A for more details). Since hardware specifications are improved regularly, it naturally also entails that the supporting software increases equally in complexity and performance. Each new version of the mobile device operating systems often releases new and complex functionality, and supporting many different versions quickly becomes a complicated task. Recent data[2] shows that only 3.3 % of the user mass for the Android platform is on it's latest major operating system version, version 5. If we include the previous version (4.4) the number rises to 44.2 %. In comparison, data from Apple[3] shows that 78 % of it's user mass is on the latest iOS version, version 8. Including the previous version of iOS raises the percentage to 98 %. This, in combination with the advantage of experience with

1 `http://bgr.com/2014/05/28/ios-vs-android-vs-windows-phone/`, accessed 2015-04-05.

2 `https://developer.android.com/about/dashboards/index.html`, accessed 2015-04-05.

3 `https://developer.apple.com/support/appstore/`, accessed 2015-04-05.

the platform, leads us to choose iOS as the target platform for our implementation.

4.1 DEVICES

The iOS platform was originally launched in 2007 with it's first device, the first model of the iPhone. It has since been expanded to include more Apple devices, such as the iPod touch and the iPad. iOS is originally just the name of the operating system used by the devices, but as both devices and functionality has been added over the last years it has become a family of products. The most prominent of them is the original product, the iPhone. At the time of writing, the iOS platform is at it's eight major version (version 8.2), a new major version is released once every year. Simultaneously with releasing a new iOS version, some new devices are often also released. The trend so far has been that a new iPhone model is released every other year, and years in between releases updated products with improved hardware specifications. An example of this is the fifth model of the phone, the iPhone 5 which was released 2012, and the iPhone 5s which was released in 2013.

The Apple iOS devices have seen a significant increase in technical capabilities since the first version in 2007. The processor has moved from 32-bit to 64-bit, and from single core to multi-core. RAM has been increased by a factor of 8 or 16, and the camera from 2 MP to 8 MP. The technical specifications of the first iPhone and the currently latest iPhone and iPad models are shown in table 4.1. A complete table with detail for all models can be found in appendix A. There are also some characteristics not covered in the table, like RAM type. The first model featured LPDDR DRAM, while the latest models features LPDDR3 DRAM, which provides significantly faster memory access. The processor core has also been significantly upgraded, from a single core 412 MHz processor to a multi-core 1.4 GHz 64-bit processor, and the architecture of the processor core is quite different. The camera has also seen some significant improvements, not only in the number of pixels covered. The optics in the camera has been improved in almost each model. As stated in chapter 1, the increase in hardware capabilities leads us to believe that the time is right for exploring the ALPR task on a mobile device.

MODEL	YEAR	PROCESSOR	RAM
iPhone	2007	412 MHz, 32-bit	128 MB
iPhone 6	2014	2x1.4 GHz, 64-bit	1 GB
iPad Air 2	2014	3x1.5 GHz, 64-bit	2 GB

MODEL	GPU	PHOTOS	VIDEO	BATTERY
iPhone	103 MHz	2 MP f/2.8	n/a	1400 mAh
iPhone 6	4x450 MHz	8 MP 1.5µ f/2.2	1080p 30-240 fps	1810 mAh
iPad Air 2	8x450 MHz	8 MP	1080p 30-240 fps	7340 mAh

Table 4.1: iOS device specifications.

As seen from table 4.1, the iPad Air 2 is significantly more powerful than the iPhone 6. The battery capacity is also superior, but it must be taken into account that the iPad has a much larger screen, and will use more power than the iPhone. The size of the device is also something that must be taken into account. The ALPR system should be able to run on a mobile device that could be carried by a person that would want to inspect LPs in a public space. Table 4.2 shows their heights, widths, depths and weights. The iPad Air is substantially larger and heavier than the iPhone 6, but the increased processing power may be necessary to provide satisfactory results from the ALPR system. The decision on whether to use the iPhone 6 or the iPad Air 2 will be made at a later time; the architecture of the platform allows for a system that is interchangeable between iPhones and iPads. It is therefore prudent to wait until after some testing has been performed before making a final decision.

4.2 OPERATING SYSTEM, SDK, AND LANGUAGE

The iOS platform's main programming language is Objective C. Objective C is a superset of C, and is object oriented and has a dynamic

MODEL	HEIGHT	WIDTH	DEPTH	WEIGHT
iPhone 6	138.1 mm	67.0 mm	6.9 mm	129 g
iPad Air 2	240 mm	168.5 mm	6.1 mm	444 g

Table 4.2: Weight and size of iOS devices.

runtime. The main compiler for Objective C on iOS is Clang backed by LLVM. Being a superset of C, Objective C also supports regular C code, and has also capabilities for embedding C++ code. Even though the C++ syntax has some major differences from Objective C, a combined language called Objective C++ exists to bridge the gap between the two languages. There are some limitations to what functionality is available in Objective C++, it does not include the full power of each language. Rather, it functions as a means to stitch together different parts of an application.

The technologies available in the iOS SDK is packaged as frameworks (Apple Inc. (2014)). Additionally, the operating system is based on a layered structure, where each layer provides another level of abstraction from hardware. Each of the frameworks reside in one of the four layers, an overview of the layers are shown in figure 4.1. The top layer contains all of the frameworks related to user interfaces and most software pattern related methods, while the lower levels provide interfaces to hardware like the camera, bluetooth antenna, and even processor scheduling. Code pertaining to each layer may communicate with lower layers as needed. The most important framework, the Foundation framework, is available through all layers, and among other things defines the most basic data types. The most important layer in an ALPR context is the media layer. The media layer contains graphics, audio and video frameworks, and allows us to interface with all of the relevant hardware on the device. Among the graphics-related frameworks we have Metal and OpenGL ES. OpenGL ES is a subset of the traditional OpenGL made for embedded systems. Metal is a similar type of framework, but is based on Apple's own technologies, and is claimed to be much more efficient. Both frameworks give us the possibility of dispatching code to the graphics processing unit (GPU) of the device. Unfortunately, using third-party libraries for

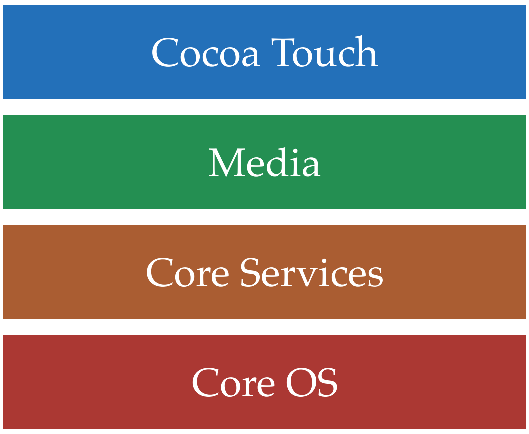


Figure 4.1: Architecture of the iOS operating system.

Metal and OpenGL is not possible without major adjustments to the source code.

For our ALPR system, the most important frameworks will be UIKit, Grand Central Dispatch, and AV Foundation. UIKit is a framework for user interface components and classes that emphasize the Model-view-controller pattern and the delegate-pattern. Grand Central Dispatch gives us fine-grained multi-threading, and provides ease-of-use functions for dispatching operations on multiple threads. AV Foundation is the framework that gives us access to the device's camera, and is used to set up a video capture and configure the camera's settings.

4.3 EXTERNAL LIBRARIES AND FRAMEWORKS

To ease the development of the system prototype we have used two prominent third party libraries. Both are available as open-source code, and can be found on the web (Itseez (2014); Google Inc. (2015)). When using third party libraries the notion of frameworks becomes especially useful, as it enables us to include a whole library as a single package, that is compiled and ready to be linked into the application we are building. There are many different package managers available, we have used the CocoaPods package manager since both these libraries are available there. It also allows for specifying specific versions of the third party software, and handles any dependencies to any other third party libraries.

4.3.1 *OpenCV*

OpenCV is a large library created for various computer vision applications (Itseez (2014)). The library is maintained by a company named Itseez, and has over 2500 optimized implementations of various computer vision and machine learning algorithms. Many of the standard image processing techniques like binarization, Hough transform, and convolution are implemented, as well as more complicated methods like object detection, ANNs, and SVMs.

4.3.2 *Tesseract OCR*

The Tesseract OCR (Smith (2007); Google Inc. (2015)) software was originally started as a PhD research project in 1984 at HP, and was developed there until 1994. The software was not released until 2005, when it was made available as an open source project. Google then decided to continue it's development, and has maintained the software ever since. The software was initially intended for use on scanners, but has since been adopted for more advanced use. Some of the steps used in the algorithm was not released as open source, and as an effect the Tesseract OCR engine assumes that it's input images is already binarized as black text on white background. Tesseract is also written in C++, and is regarded as one of the top OCR engines today. Trained data is provided with the software, but it is also possible to train Tesseract to a specific domain. On the tests referenced in Smith (2007), the recognition rate lies at approximately 98%.

5 SYSTEM MODEL

The prototype created for this thesis is implemented for iOS 8, and runs on both iPhone and iPad models. Both the OpenCV library and the Tesseract OCR library has been used in developing this prototype. Each stage of the ALPR task has some defined inputs and outputs, so that interchanging implementations for a given stage can be done without altering the rest of the system. Additionally, ALPR calculations are done asynchronously and on dedicated threads. The main thread of an iOS application will take care of all user interface-related tasks, and so it is a good practice to keep heavy calculations away from this thread. Another motivation for asynchronous dedicated threads is to be better able to measure efficiency. Lastly, while processing a video stream it is desirable to process as many frames as possible. By using a dedicated thread it is easy to see if the thread is still processing some previous frame when a new frame arrives from the camera. This allows for discarding frames that cannot be processed in real time.

The data set used by the model contains a training set of 2250 positive images and 4354 negative images, and a test set of 546 images of Norwegian LPs. Additionally, a set for training OCR was created from the training set images, totaling 5162 segmented images of characters.

5.1 LICENSE PLATE DETECTION

The first stage in the ALPR task is LP detection. Two different approaches has been selected for this task, one based on a cascade classifier and the other based on searching a color difference edge image from trichromatic imaging.

5.1.1 *Cascade Classifier using features*

OpenCV has an implementation of the cascade classifier described by Viola and Jones (2001) using Haar-like features. MB-LBP features are also included, the implementation is based on Liao et al. (2007). There are some small differences between the training implementation in OpenCV and the implementations described by Viola and Jones and Liao et al.. Both papers advocate for using a specific version of the AdaBoost algorithm, while the OpenCV implementation allows the user to select between four different variations. Moreover, some parameters allows for slightly different classifiers, like tree depth. Tree depth denotes if each AdaBoost classifier can be a decision tree or a stump, which is effectively one classifier. It is also possible to limit the maximum number of classifiers allowed in each stage. The original cascade classifier training algorithm by Viola and Jones is set to terminate when the entire cascade reaches a minimum defined hit rate, or a maximum defined false alarm rate. The hit rate denotes the rate of true positive classifications to all positive training examples. Similarly, the false alarm rate denotes the rate of false positive classifications to all negative training examples. The OpenCV training implementation differs in defining the number of stages in the cascade explicitly, while the Viola and Jones version adds as many stages that are necessary to reach globally specified minimum hit rate and maximum false alarm rate.

Since training is a very time-consuming effort — training a 15-stage cascade classifier takes at least two weeks on a normal modern personal computer — a computer provided by NTNU was used for training the different classifiers. This machine has 32 Intel Xeon 2.6 GHz processors, and has 128 GB RAM. Using such a machine reduces training time from weeks to days, normal training time for a Haar-like feature based cascade classifiers is around two days.

The OpenCV library also includes an application for generating new positive training images. This is done by rotating the image around the x, y, and z-axes, and by applying illumination variations and distortions. From our training set of 2250 images we have trained several different classifiers with different numbers of positive training examples and different numbers of stages to investigate the effects on accuracy. Given the nature of the algorithm when using a trained classifier for detection, we expect that an increased number of stages will negatively affect efficiency. Table 5.1 shows the different variations in number of positive training examples and number of stages used.

# POS. EX.	FEATURE TYPE	# STAGES
5000	Haar-like	12
5000	Haar-like	15
5000	MB-LBP	12
5000	MB-LBP	15
10000	Haar-like	12
10000	Haar-like	15
10000	MB-LBP	12
10000	MB-LBP	15

All classifiers were trained with the same number of negative examples, 4354. Additionally, all classifiers were trained with the same minimum hit rate of 99.5 % and maximum false alarm rate of 50 %.

Table 5.1: Trained cascade classifiers.

5.1.2 *Color difference edge image and filtering*

Many different ALPR systems utilize edge information in candidate images to locate an LP. Due to the nature of the characters present in the LP, the region normally presents as a rectangle of dense edge information. By using a color difference edge image the noise in the image can be considerably reduced, and finding regions of interest may be an easier task. Yang et al. (2012) presents an algorithm for calculating the color difference edge image and subsequently filtering and searching the resulting image for regions of interest. Filtering is done by smoothing the image with an appropriately sized kernel, followed by a binarization. The procedure for searching is shown in algorithm 5.1.

Our implementation of this technique has some slight modifications. Line 4 in algorithm 5.1 states that the rectangle should be segmented out of the original image; our variation does not segment out the region and the image is left the same. This is done to avoid prematurely extracting components from the image that may in following row analyses turns out to be adjacent to additional components, i.e. part of a larger region of interest. Additionally, the rectangles that are marked as regions of interest may be very close to each other, some of

Algorithm 5.1 Locating regions of interest within a color difference edge image, adapted from Yang et al. (2012).

```
$I$ is a candidate image possibly containing an LP.
$n_1$ is the minimal number of LP characters.
$d_1$ is the maximal interspacing between LP characters.
$(a \times b)$ ranges from the minimal to the maximal LP size.
1: for every row $R$ in $I$ do
2:   if $R$ traverses $\geq n_1$ adjacent connected components and inter-
     spacing $\leq d_1$ then
3:     if the adjacent connected components can be enclosed by a
       rectangle $(a \times b)$ then
4:       segment the rectangle out of the image
5:       mark the rectangle as a region of interest
6:     end if
7:   end if
8: end for
```

them even overlapping. Another modification is therefore to merge overlapping or touching regions of interest. The entire algorithm is shown in algorithm 5.2, and an example result is shown in figure 5.1.

5.2 CHARACTER SEGMENTATION

Two different implementations have been explored for character segmentation, one inspired by the method described by Zheng et al. (2013), and another inspired by the SCW-method of Anagnostopoulos et al. (2006).

5.2.1 *Statistical character segmentation*

Statistical character segmentation is done by purely analyzing statistical characteristics for a region of interest. The algorithm is closely based on the character segmentation described by Zheng et al. (2013). The basic flow of the algorithm is to use a horizontal projection vector for a Sobel edge image to estimate character baseline and cap height to crop away the top and bottom of the image. Further, a binarized image (using Otsu's method) creates a vertical projection vector that is used to estimate character regions. Each region is analyzed with

Algorithm 5.2 Modified finding location of regions of interest within a color difference edge image, modified from Zheng et al. (2013).

```
  $I$ is a candidate image possibly containing an LP.
  $n_1$ is the minimal number of LP characters.
  $d_1$ is the maximal interspacing between LP characters.
  $(a \times b)$ ranges from the minimal to the maximal LP size.
 1: will contain marked regions
 2: for every row $R$ in $I$ do
 3:     $m$ is the number of connected components traversed by $R$.
 4:     if $m \geq n_1$ then
 5:         $C \leftarrow \emptyset$ is an empty set representing the current region
 6:         for $k = 1$ to $m$ do
 7:             $c_k$ is the $k$-th connected component traversed by $R$
 8:             if distance from $c_k$ to $c_{k-1} \leq d_1$ and $k < (m-1)$ then
 9:                 $C \leftarrow C \cup c_i$
10:             else
11:                 if $|C| \geq n_1$ and $C$ can be enclosed by a rectangle $(a \times b)$
                    then
12:                     Find the smallest $(a \times b)$ enclosing all $c \in C$.
13:                     $\Omega \leftarrow \Omega \cup (a \times b)$
14:                 end if
15:                 $C \leftarrow \emptyset$
16:             end if
17:         end for
18:     end if
19: end for
20: Project every rectangle in $\Omega$ onto an image $P$ that has the same
    size as $I$
21: Find all contours $\Theta$ in $P$
22: Mark rectangles enclosing contours in $\Theta$ as regions of interest
```

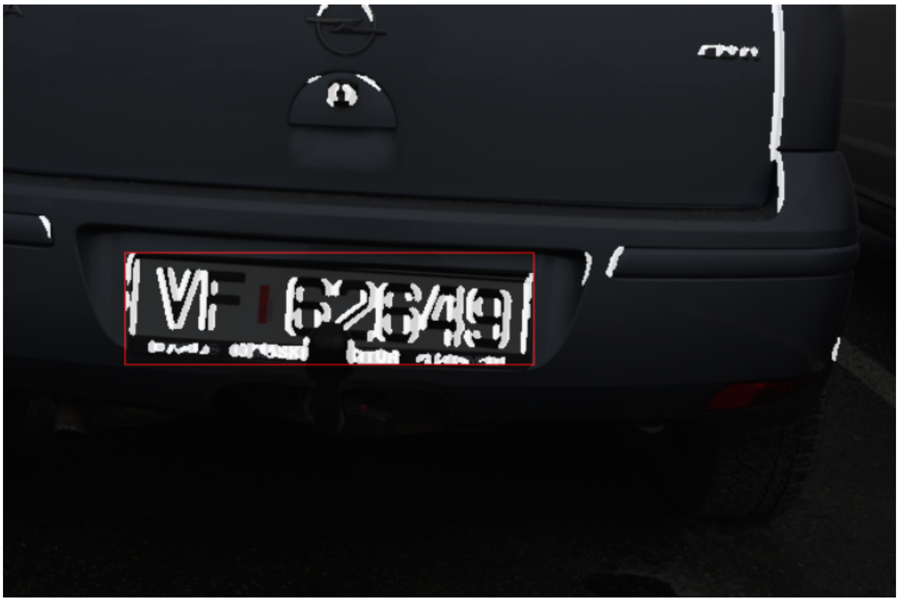


Figure 5.1: License plate location using color difference edge image.

CCA, and the connected components are filtered with the following rules and actions:

- If the upper boundary is different from the *overall upper boundary*, the component is discarded.
- If the inverse aspect ratio $\left(\frac{1}{\text{aspect ratio}}\right)$ is > 0.8 or < 0.3, the component is discarded.
- If the area of the component is more than one sixth of the whole image size, it needs further segmentation.
- If any two neighbor segments' widths are smaller than 66.66 % of the average character width, the segments are joined together. However, if the joined segment has a width 33.33 % greater than the average character width, the joining is not done after all.

The original algorithm has some ambiguities regarding the filtering rules. One of the rules states that a component needs further segmentation if it's size exceeds one sixth of the image's size, but there is not given any account of what this actually entails. A possible solution is to increase the threshold found when using Otsu's method for binarizing the image, and run CCA and the filtering rules again for this particular region. Another unclear aspect is how the upper boundary of the components are compared in the first rule. The wording used suggests that the upper boundaries must be within a certain range, but there is no mention of the components' height or the lower boundary of the components. Moreover, the estimation of character baseline and cap height is not accounted for. The use of a horizontal projection vector is discussed, but how it's values are used is not mentioned. The example projections shown in the paper are free from noise, giving a good estimation of the characters, but there is no mention of possible noise that may be present.

Our algorithm addresses some of these issues. We propose using a non-zero threshold for estimating character baseline and cap height. Additionally, when filtering connected components, some different rules are used. The rule regarding the upper bound of the characters use mean values and standard deviation to identify outliers, and we also include the same evaluation for the lower bounds of the components. A new metric regarding fill percent is also introduced, putting a constraint on how many pixels a component may occupy within it's bounding rectangle. The edge image of the region is also considered, requiring that the component traverses some edge in the corresponding edge image.

After filtering is done we propose adding a second phase of the algorithm, to enhance accuracy in this stage of the ALPR task, as well as the next. The objective of this phase is to correct rotated regions, so that components that does not conform to the baseline and cap height of the other segments may be removed. Having right aligned character segments will also have a positive effect on the character recognition stage. If all characters have the same alignment there will be less room for variation within each character class, and it should be an easier task to discriminate between them. Some related methods argue that correcting orientation is an expensive task, since it will require resampling of the region. Some initial tests does however reveal that the operation can be executed efficiently. We found that correcting orientation takes 2.3 ms on average using an iPhone 6. In light of this information we find that correcting orientation is an operation that can be included. Given the nature of the algorithm it is also natural to consider orientation correction at this particular point in the process. Earlier in the process the image is still populated with potentially many non-character segments, and trying to estimate a correction angle would involve a substantial amount of noise. Estimating orientation correction at a later point in the process would also prove difficult, as the character regions' spatial information does not follow into the next ALPR stage. Orientation correction is done by estimating a rotation angle using least squares estimation of the segments' center points. The image is rotated so that the estimated rotation angle is level. It is prudent to repeat some of the earlier steps in the algorithm after the rotation, since the rotation will involve resampling the image. We propose performing CCA again to find components that are potential character segments, and use the filtering rules for upper and lower bounds, and fill percent to remove those that are most likely non-character segments. The entirety of the method is shown in algorithm 5.3.

5.2.2 *Sliding concentric windows*

Anagnostopoulos et al. (2006) describes in an algorithm for using SCW for character segmentation. Our implementation is inspired by this method, but has been extended to account for remaining non-character elements. The original algorithm is described in algorithm 5.4. SCW is run on a region, and each component produced from SCW must meet some criteria to be kept in the image. Then, a statistical analysis involving standard deviation and mean value of the horizontal and vertical projections estimate bounding rectangles of the components that are left in the image.

Our modified algorithm, algorithm 5.5, follows many of the same steps. The component criteria check is modified to check the area and height of each components bounding rectangle. Then, a orientation correction is performed to account for rotated regions. The original statistical analysis to find bounding rectangles is kept, but the extracted components are subject to an additional round of filtering. Here, their aspect ratio and fill-percent are checked to ensure that as few as possible false positives are produced.

5.3 CHARACTER RECOGNITION

The character recognition stage is implemented with three different variations, ANN classifier, SVM classifier, and Tesseract OCR. An example of training data characters is shown in figure 5.2.

Algorithm 5.3 Statistical character segmentation, modified from Zheng et al. (2013).

```
    $I$ is a region of interest, possibly containing an LP.
    $\alpha, \beta, \gamma, \zeta$ are empirically found constants.
 1: $I_\nabla \leftarrow$ Sobel edge image of $I$.
 2: $\mathbf{h}_\nabla \leftarrow$ horizontal projection of $I_\nabla$.
 3: $y_1, y_2 \leftarrow$ the start and the end of the longest consecutive range where elements in $\mathbf{h}_\nabla > \alpha$.
 4: Remove row numbers $< y_1$ and $> y_2$ from $I$.
 5: $I_b \leftarrow I$ binarized with Otsu's method.
 6: $\mathbf{v}_{I_b} \leftarrow$ vertical projection of $I_b$.
 7: $R \leftarrow I_b$ split into regions using start and end indices of non-zero consecutive ranges in $\mathbf{v}_{I_b}$.
 8: $C \leftarrow$ components found from CCA on all regions $\in R$.
 9: $\mu_u, \mu_l, \sigma_u, \sigma_l \leftarrow$ upper and lower bound mean and standard deviation values for all $c \in C$.
10: for each component $c_k \in C$ do
11:     if $|\texttt{upperBound}(c_k) - \mu_u| > 2\sigma_u$ or $|\texttt{lowerBound}(c_k) - \mu_l| > 2\sigma_l$ then
12:         remove $c_k$ from $I_b$.
13:     else if $\texttt{aspectRatio}(c_k) > 0.9$ or $\texttt{aspectRatio}(c_k) < 0.2$ then
14:         remove $c_k$ from $I_b$.
15:     else if $\texttt{area}(c_k) > \frac{1}{6}\texttt{area}(I)$ or $\texttt{area}(c_k) < \gamma$ or $\texttt{fillPercent}(c_k) > \zeta$ then
16:         remove $c_k$from $I_b$.
17:     else if $c_k$ does not traverse any edge in $I_\nabla$ then
18:         remove $c_k$ from $I_b$.
19:     end if
20: end for
21: $\phi \leftarrow$ estimated baseline angle found using least squares estimation on components left in $I_b$.
22: $I_b \leftarrow I_b$ rotated with angle $-\phi$.
23: $C \leftarrow$ components found from CCA on $I_b$.
24: $\mu_u \leftarrow$ upper bound mean value for all $c \in C$.
25: $\sigma_u \leftarrow$ upper bound standard deviation for all $c \in C$.
26: for each component $c_k \in C$ do
27:     if $|\texttt{upperBound}(c_k) - \mu_u| > 2\sigma_u$ or $|\texttt{lowerBound}(c_k) - \mu_l| > 2\sigma_l$ or $\texttt{fillPercent}(c_k) > \zeta_1$ or $\texttt{fillPercent}(c_k) < \zeta_2$ then
28:         remove $c_k$ from $C$.
29:     end if
30: end for
31: Create character segments from components in $C$
```

Algorithm 5.4 Character segmentation by sliding concentric windows, adapted from Anagnostopoulos et al. (2006).

```
$I_1$ is a region of interest, possibly containing an LP, resized to 228×75 pixels.
 1: for each pixel $p_k$ in $I_1$ do
 2:     run SCW with inner window size 5×11, outer window size 9×21, statistical measure standard deviation, and threshold 0.7.
 3: end for
 4: $I_2 \leftarrow$ the resulting image of SCW.
 5: $I_3 \leftarrow$ the inverse image of $I_2$.
 6: $C \leftarrow$ components found from CCA on $I_3$.
 7: $I_4 \leftarrow$an empty image with the same size as $I_3$.
 8: for each component $c_k \in C$ do
 9:     if orientation$(c_k) > 75°$ and height$(c_k) > 32$ then
10:         copy $c_k$ into $I_4$.
11:     end if
12: end for
13: $\mathbf{v}_{I_4} \leftarrow$ vertical projection of $I_4$.
14: $\mathbf{h}_{I_4} \leftarrow$ horizontal projection of $I_4$.
15: $C \leftarrow$ components in $I_4$ that fall within the bounds of standard deviation of $\mathbf{v}_{I_4}$ and $\mathbf{h}_{I_4}$.
16: Create character segments from components in $C$.
```

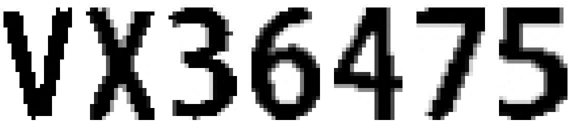


Figure 5.2: Character recognition training data example.

Algorithm 5.5 Character segmentation by sliding concentric windows, modified from Anagnostopoulos et al. (2006).

```
$I_1$ is a region of interest, possibly containing an LP, resized to 228×75 pixels.
$\alpha, \beta$ are empirically found constants.
1: for each pixel $p_k$ in $I_1$ do
2:     run SCW with inner window size 3×3, outer window size 7×7, statistical measure standard deviation, and threshold 0.95.
3: end for
4: $I_2 \leftarrow$ the resulting image of SCW.
5: $I_3 \leftarrow$ the inverse image of $I_2$.
6: $C \leftarrow$ components found from CCA on $I_3$.
7: $I_4 \leftarrow$an empty image with the same size as $I_3$.
8: for each component $c_k \in C$ do
9:     if $\texttt{area}(c_k) < \frac{1}{5}\texttt{area}(I_3)$ and $\texttt{height}(c_k) < \alpha$ then
10:        copy $c_k$ into $I_4$.
11:    end if
12: end for
13: $\phi \leftarrow$ estimated baseline angle found using least squares estimation on components in $I_4$.
14: $I_5 \leftarrow I_4$ rotated by angle $-\phi$.
15: $C \leftarrow$ components found from CCA on $I_5$.
16: $\mathbf{v}_{I_5} \leftarrow$ vertical projection of $I_5$.
17: $\mathbf{h}_{I_5} \leftarrow$ horizontal projection of $I_5$.
18: $C \leftarrow$ components in $I_5$ that fall within the bounds of standard deviation of $\mathbf{v}_{I_5}$ and $\mathbf{h}_{I_5}$.
19: for each component $c_k \in I_5$ do
20:     if $\texttt{aspectRatio}(c_k) > 1$ or $\texttt{aspectRatio}(c_k) < 0.25$ or $\texttt{fillPercent}(c_k) < \beta_1$ or $\texttt{fillPercent}(c_k) > \beta_2$ then
21:        remove $c_k$ from $C$.
22:    end if
23: end for
24: Create character segments from components in $C$.
```

5.3.1 *Artificial neural network*

The artificial neural network is implemented using OpenCV, and is trained with a training set of 5162 segmented images of characters. Training and classification is done using three-feature vectors:

1. The first feature is a horizontal projection of the character region, resized to 38 pixels.
2. The second feature is a vertical projection of the character region, resized to 27 pixels.
3. The third feature is the entire image laid out as a vector, resized to 10×14 pixels.

The entire feature vector is $38 + 27 + 10 \cdot 14 = 205$ elements. The topology of the network is chosen as a direct result of the size of the feature vector and the number of classes. The network is an MLP, and has three layers. The input layer has 205 nodes, while the hidden layer and the output layer has 30 nodes, one for each possible class. Note that the set of 30 classes is complete, some letters are not part of the Norwegian LP format. The network is trained using the back-propagation algorithm (Russell and Norvig (2003)). A symmetrical sigmoid function

$$f(x) = \beta \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{5.1}$$

is used as the activation function in the network, the function is shown in figure 5.3. Initial tests shows that the network performs best with the parameters α and β set to 1.

5.3.2 *Support vector machine*

An SVM is implemented using OpenCV, and is trained with the same training set as the ANN. Feature vectors are also computed in the same manner. The SVM uses a linear kernel function, and is configured for soft margins, a property of C-support vector machines (Chang and Lin (2011)). Soft margins means that adjustments of support vectors will allow training examples to reside on the wrong side of a support vector, but is given a penalty score (multiple of C) for doing so. Using soft margins can help steer the classifier away from

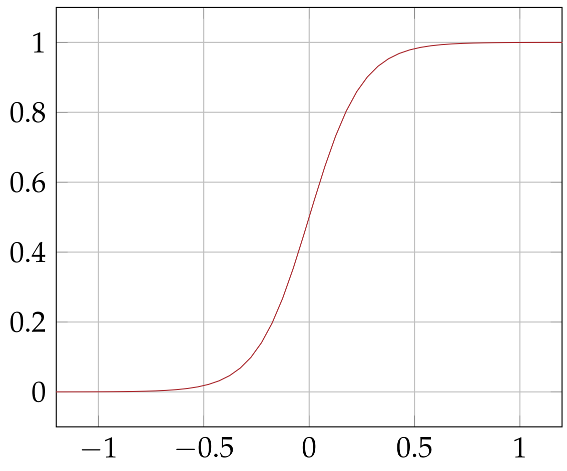


Figure 5.3: Symmetrical sigmoid function (equation 5.1), here with $\beta = 1, \alpha = 8$

overfitting a training set. The classifier is trained with the *one-against-one* strategy, which trains $k(k-1)/2$ classifiers for k classes, where each classifier is trained to distinguish between two given classes.

5.3.3 *Tesseract OCR*

Tesseract OCR can be configured for many different use-case scenarios of OCR, in this case it is configured for single character recognition. Tesseract can be trained manually, or it can be used with some training data that is available from the website (Google Inc. (2015)), which is used here. Another training data file, from the open source project OpenALPR by New Designs Unlimited, LLC (2015), is also used. The latter is trained for European LPs, while the former is trained for general OCR on characters in the English alphabet.

6 RESULTS AND EVALUATION

This chapter contains the results for each of the three stages in the ALPR task. Each stage has two different implementations, as discussed in chapter 5. Additionally, the results will be interpreted and evaluated. To evaluate the ALPR system we use two test sets. The first test set contains 546 annotated images, and is used on the first two stages of the ALPR task. The tests are carried out for three of the most common image sizes on the iPhone and iPad models: 640×480 pixels, 1280×720 pixels, and 1920×1080 pixels. The second test set contains 5162 character segments, and is used in cross-validation to train and test the third stage of the ALPR task.

6.1 LICENSE PLATE DETECTION

License plate detection is done with two different techniques, a cascade classifier and a filtering method based on a color difference edge image. We evaluate the accuracy of the methods as the number of true positive detections, measured as the opposite of the number of false negative detections. In order to further assess the quality of each method we also evaluate the number of false positives detected in the test set. False positives are recorded whenever a detected rectangle overlaps less than 75 % of the correct rectangle for an LP. It could be argued that false positives should count towards the classifiers accuracy. In this case we leave the false positives out of the accuracy calculation, because a false positive could be discarded at a later stage. Moreover, we deem it better to falsely detect an LP, and displaying a bogus LP to a user, rather than missing an LP. If a system misses too many LPs it is decidedly not useable, while a system with some false positives is still useable. Even so, our aim is to minimize false positives, and a method with too many false positives must be discarded.

6.1.1 *Cascade classifier*

Since the cascade classifier is trained, it is largely dependent on it's training data and training method. The cascade classifier method has been tested with Haar-like features and MB-LBP-features, and varying degrees of training examples. The original publication by Viola and Jones (2001) suggests that the classifier needs a fairly large training set — 9832 positive examples — whereas the ALPR system by Zheng et al. (2013) uses only 305 positive examples for training. The number of stages in a classifier is also an important property. While fewer stages leads to faster training and classification, adding stages can help filter out false positives. There is however a risk of overfitting for the training set if the number of stages is set too high. Each classifier can be adjusted with a filtering parameter, which decides how many suspected true positive rectangles are needed in a region to produce a detection. The filtering is not a linear process, there is a nonlinear calculation that pertains to the rectangles' center of mass, and interspacing. Consequently it is very hard to analytically find a best filtering. Instead we have used a method inspired by the gradient descent algorithm to find the filtering that minimizes the sum of false negatives and false positives, while having the fewest number of false negatives. Figure 6.1 shows how the false negative and false positive numbers decline and increase with the filtering parameter. The marked out intersection shows the minimum sum of false negatives and false positives. Table 6.1 shows the results for the different combinations of positive examples and feature types. The abbreviations FP and FN stands for false positive and false negative.

The best results for each training set size in table 5.1 are marked out as bold. We observe that for all image sizes the 15-stage MB-LBP classifier has the highest accuracy. Additionally, the classifier trained with 10000 training examples has the lowest number of false positives for every image size. The test results also shows that the MB-LBP feature significantly outperforms the Haar-like feature for any given combination of image size, number of stages, and number of positive training examples. The number of false negatives produced by MB-LBP features are significantly lower than that of the Haar-like features, and the number of false positives are also lower in almost every case.

# POS. EX.	# STAGES	HAAR-LIKE	MB-LBP
5000	12	FN: 30, FP: 40	FN: 16, FP: 13
		94.51 %	97.07 %
	15	FN: 14, FP: 14	**FN: 8, FP: 8**
		97.44 %	**98.53 %**
10000	12	FN: 37, FP: 32	FN: 11, FP: 22
		93.22 %	97.99 %
	15	FN: 16, FP: 16	**FN: 7, FP: 3**
		97.07 %	**98.72 %**

(a) Accuracy for 480×640 pixel test images.

# POS. EX.	# STAGES	HAAR-LIKE	MB-LBP
5000	12	FN: 58, FP: 34	FN: 16, FP: 17
		89.38 %	97.07 %
	15	FN: 29, FP: 11	**FN: 7, FP: 14**
		94.69 %	**98.72 %**
10000	12	FN: 46, FP: 32	FN: 31, FP: 26
		91.58 %	94.32 %
	15	FN: 28, FP: 19	**FN: 14, FP: 13**
		94.87 %	**97.44 %**

(b) Accuracy for 1280×720 pixel test images.

# POS. EX.	# STAGES	HAAR-LIKE	MB-LBP
5000	12	FN: 66, FP: 83	FN: 21, FP: 44
		87.91 %	96.15 %
	15	FN: 42, FP: 24	**FN: 13, FP: 17**
		92.31 %	**97.62 %**
10000	12	FN: 63, FP: 77	FN: 39, FP: 43
		88.46 %	92.86 %
	15	FN: 45, FP: 28	**FN: 15, FP: 8**
		91.76 %	**97.25 %**

(c) Accuracy for 1920×1080 pixel test images.

Table 6.1: Cascade classifier accuracy results for license plate detection.

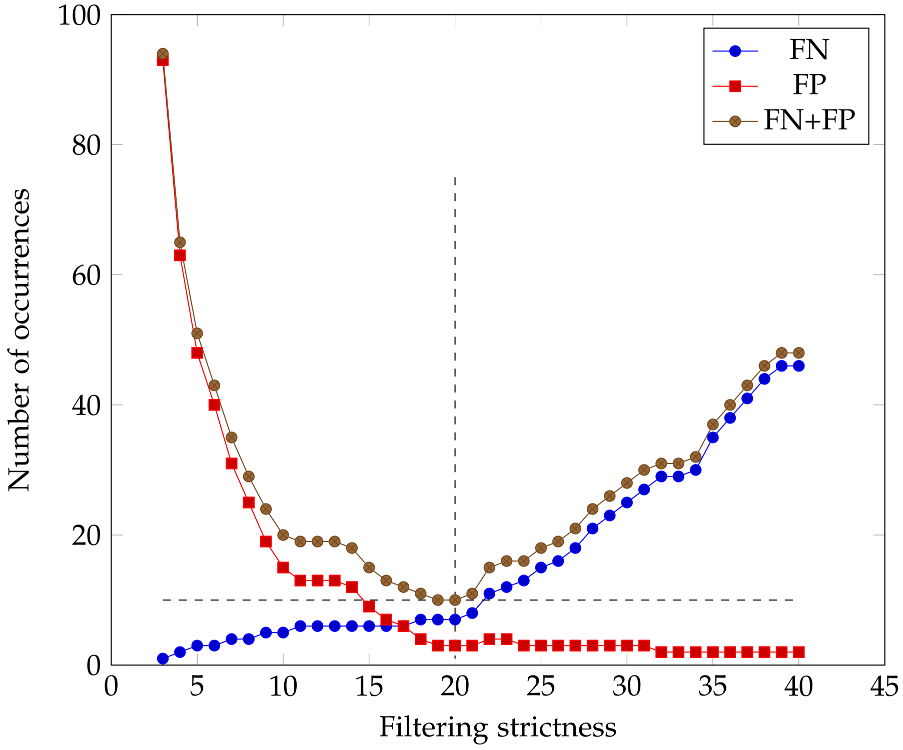


Figure 6.1: Estimation of best filtering parameter for cascade classifier

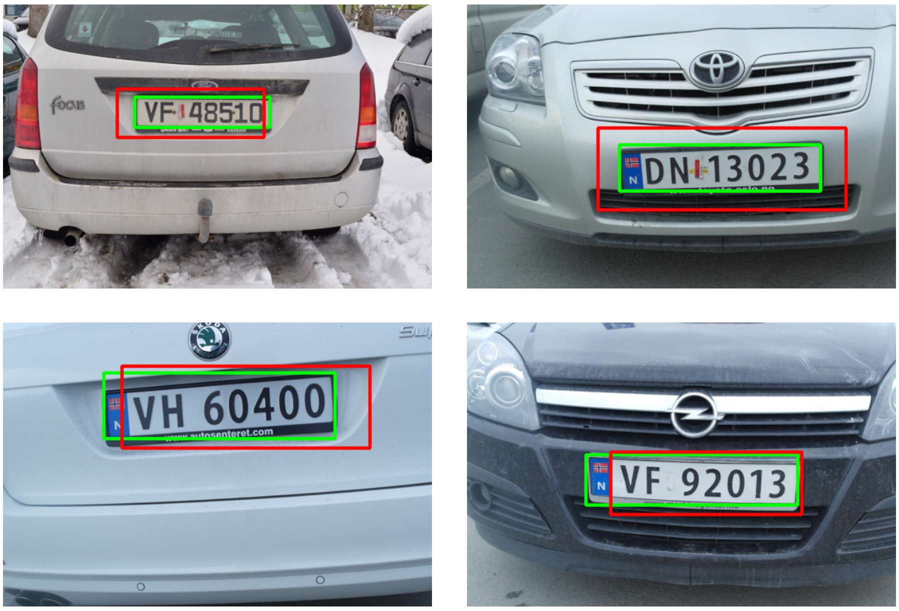


Figure 6.2: Examples of cascade classifier detections. The green rectangles represent the correct rectangle for the LP, the red rectangles represents the detected rectangle.

6.1.2 *Color difference edge image and filtering*

The color difference edge image filtering method is implemented as described in algorithm 5.2. The parameters n_1, d_1 and $(a \times b)$ are estimated using the same gradient descent inspired technique mentioned in the previous section. Although the parameters initially have a strong connection to reality, initial tests reveal that setting them to real-world values does not provide a decent LP location. d_1 is also dependent on the expected size of the LP, so that varying the parameter is necessary to be able to test for different image sizes. The results from the method are shown in table 6.2.

We observe that the accuracies are significantly lower than all of the cascade classifiers, and that the number of false positives are alarmingly high. With these high false positive rates, about one in every five detections are false positives. We speculate that one of the reasons for these high numbers of false positives is the filtering algorithm applied to the color difference edge image. While the edge image produced by the method contains significantly less noise than a Sobel edge image, the method in itself is still not able to correctly

IMAGE SIZE	ACCURACY
640×480	FN: 123, FP 154
	77.47 %
1280×720	FN: 112, FP: 106
	79.49 %
1920×1080	FN: 64, FP: 339
	88.27 %

Table 6.2: Color difference edge image filtering accuracy results for license plate detection.

filter out non-LP elements, and correctly identify regions of LP elements. Figure 6.3 shows an example of a false positive detection. The white regions marked out in the image are the elements that are left after filtering is done, and the red rectangles are the detected regions. The front lights of the car in the image creates elements that are not filtered out by the algorithm, and their semi-parallel nature produces a detection. Another problem with the method arises when an LP is dirty, or the image has washed out colors. Since the method is dependent on quantizing color difference, some LPs are not detected when the contrast between characters and plate is too low.

Given these results, it is clear that the cascade classifier is the best alternative for the LP detection stage.

6.1.3 *Efficiency*

The efficiency, i.e. the average processing time, is measured as the average time needed to detect rectangles for every example in the test set. There is possibly some overhead processing included in the given times, but still, it gives us a clear indication of how the different classifiers compare. In a real-world application it is natural to account for some overhead, since the application would in any case need to do some work that is not strictly pertinent to the ALPR task. Table 6.3 shows the average processing times for each of the cascade classifiers, tested on an iPhone 6. The table shows that the image size has a significant effect on the processing time, as expected. The average processing time scales almost linearly given the number of pixels in

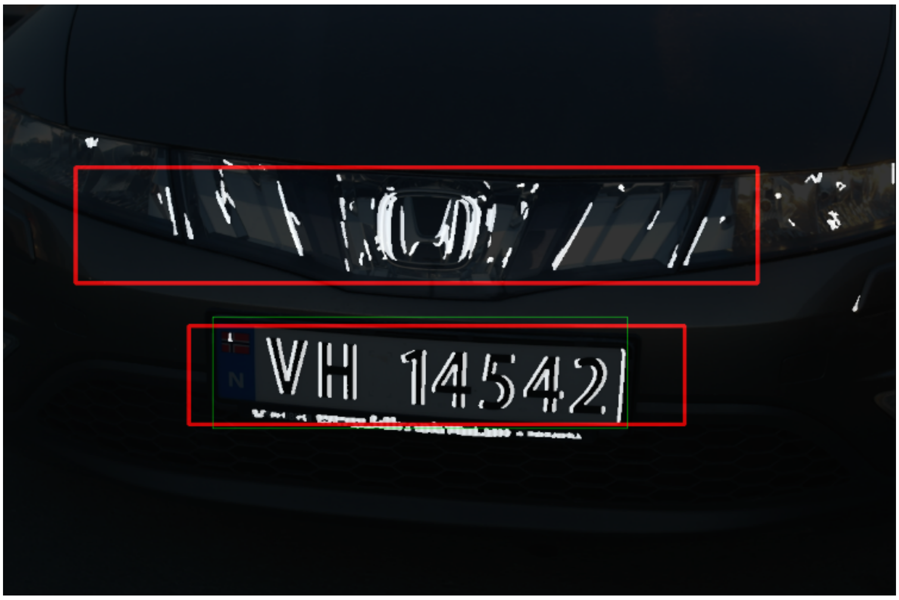


Figure 6.3: Example of false positive detection by Color difference edge image filtering.

an image. By comparison, the classifiers use about a third of the processing time when tested on a notebook computer (2.5 GHz Intel i7, 16 GB RAM). The color difference edge image filtering method has already been discarded due to it's low accuracy, but it is worth mentioning that it also shows efficiency 3 to 4 times slower than the cascade classifier.

6.2 CHARACTER SEGMENTATION

The character segmentation stage has two different implementations. One is solely based on statistical analysis of the region (algorithm 5.3), while the other is based on SCW followed by a statistical analysis (algorithm 5.5). In this stage we still analyze false positives and false negatives, to get a clearer picture of how the different methods work for the test set. Unlike the previous stage, false positives and false negatives are both viewed as errors. Two different metrics are used to evaluate accuracy, character accuracy and LP accuracy. Character accuracy denotes how many of the total number of characters in the test set that are successfully segmented, while LP accuracy denote how many of the LPs in the test set have all their characters success-

# POS. EX.	# STAGES	HAAR-LIKE	MB-LBP
5000	12	46.15 ms	33.53 ms
	15	44.55 ms	31.36 ms
10000	12	42.90 ms	34.96
	15	43.69 ms	32.89 ms

(a) Efficiency for 640×480 pixel images.

# POS. EX.	# STAGES	HAAR-LIKE	MB-LBP
5000	12	164.94 ms	111.83 ms
	15	157.13 ms	108.58 ms
10000	12	148.46 ms	116.06 ms
	15	152.95	111.08 ms

(b) Efficiency for 1280×720 pixel images.

# POS. EX.	# STAGES	HAAR-LIKE	MB-LBP
5000	12	419.95 ms	292.97 ms
	15	401.42 ms	276.69 ms
10000	12	371.14 ms	308.59 ms
	15	383.61 ms	293.54 ms

(c) Efficiency for 1920×1080 pixel images.

Table 6.3: License plate location efficiency results.

IMAGE SIZE.	FN	FP	CHAR. ACC.	LP ACC.
640×480	224	43	93.01 %	83.88 %
1280×720	212	33	93.59 %	86.08 %
1920×1080	192	38	93.98 %	87.00 %

(a) Statistical character segmentation.

IMAGE SIZE.	FN	FP	CHAR. ACC.	LP ACC.
640×480	289	97	89.90 %	64.65 %
1280×720	101	147	93.51 %	71.43 %
1920×1080	106	133	93.74 %	72.89 %

(b) Sliding concentric windows.

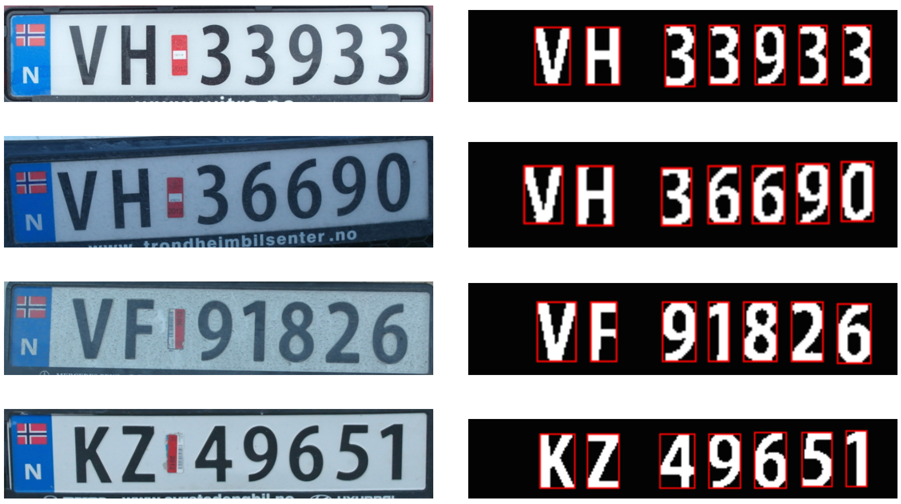
Table 6.4: Character segmentation accuracy results.

fully segmented with no false positives. The main difference between these two metrics is that the character accuracy will enlighten a general notion of segmentation capabilities, while the LP accuracy gives more information about how well these methods will work for the ALPR task. The results from the tests are shown in table 6.4.

We observe that the two different methods have their faults in different ways. The statistical character segmentation's errors largely comes from false negatives. In comparison, the SCW method has a less uneven distribution of false negatives and false positives. The character accuracy for the SCW presents as slightly better than the statistical character segmentation. However, by looking at the LP accuracy it is clear that the statistical character segmentation is far better suited for the ALPR task. The average processing times for each of the methods, shown in table 6.5, shows that the methods also differ significantly in efficiency. The processing times were tested on an iPhone 6. The SCW method is not very affected by the image size, since the region is resized to a given format before the algorithm is run. The poor accuracy of the SCW method makes it clear that the statistical character segmentation is superior to the SCW approach. Even so, we remark that the LP accuracy leaves something to be desired.

IMAGE SIZE.	STAT. REG. SEG.	SCW
640×480	11.92 ms	15.67 ms
1280×720	19.39 ms	15.99 ms
1920×1080	24.02 ms	16.10 ms

Table 6.5: Character segmentation efficiency results.


(a) Correct segmentations

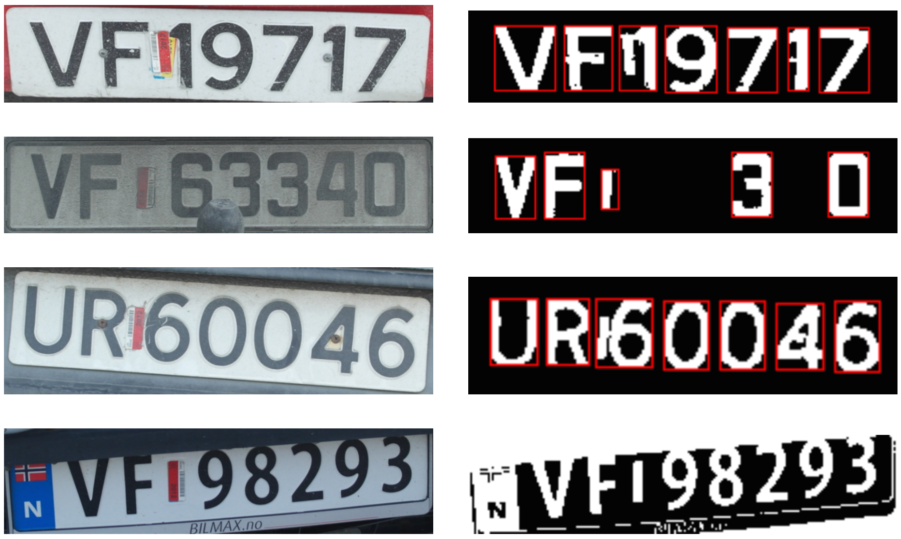

(b) Incorrect segmentations

Figure 6.4: Examples of character segmentations. The red rectangles represent the outline of the segmented character regions.

METHOD	ACCURACY	EFFICIENCY
MLP	98.67 %	0.08 ms /char.
SVM	99.56 %	0.16 ms /char.

(a) 5-fold cross-validation.

METHOD	ACCURACY	EFFICIENCY
MLP	98.79 %	0.09 ms /char
SVM	99.60 %	0.16 ms /char.

(b) 10-fold cross-validation.

METHOD	ACCURACY	EFFICIENCY
Tesseract eng.	90.36 %	5.01 ms /char.
Tesseract EU	92.93 %	3.14 ms /char.

(c) Tesseract

Table 6.6: Character recognition accuracy and efficiency results.

6.3 CHARACTER RECOGNITION

Character recognition has two different implementations, an MLP network, and a multi-class SVM. To evaluate the implementations, and form a baseline for wanted accuracy and efficiency, we also include the Tesseract OCR engine. Since the two implementations require training, we use *k*-fold cross-validation to train and test. *k*-fold cross-validation splits the data set into *k* subsets, and each subset is in turn used for testing while all other subsets are used for training. This allows for using the entire data set for both training and testing, while still keeping training and testing data completely separated in each fold. Tesseract is used with it's provided training data, allowing it to consume the entire data set as test data. Table 6.6 shows the accuracy and efficiency of the two implemented methods and two different Tesseract versions. All of the tests were executed on an iPhone 6. The English version of Tesseract is used with it's provided training data, while the European version of Tesseract is used with training data produced by New Designs Unlimited, LLC (2015).

We observe that both the implemented classifiers are significantly more accurate than Tesseract, and that efficiency is also significantly faster. The European version of Tesseract is much more suited for the Norwegian LPs, both in terms of accuracy and efficiency. There is however such a large performance gap to the SVM and the MLP classifiers that Tesseract can easily be discarded. The SVM implementation offers a higher accuracy than the MLP, but the efficiency is about half for both the 5-fold and the 10-fold tests. For an average Norwegian license plate with 7 characters the difference between the two methods is likely less than 0.6 ms, which is insignificant if we compare to the expected efficiency of the other stages. We choose to use the SVM for character recognition because of it's higher accuracy. Figure 6.5 shows examples of character recognition done in the test set, note the similarities between the letter D and the numeral 0 in some of the examples.

6.4 PUTTING IT ALL TOGETHER

To test the complete ALPR system we have created a final test that uses the three above selected implementations. Each image in the test set is tested, and whenever a stage is successful the result is passed to the next stage. This means that stages 2 and 3 may not be tested with the complete test set. We expect to see results that are fairly similar to those presented in the previous sections, although there are some variations in the methods. The character segmentation stage was in the previous section tested with the correct frame for the given LP, but in this instance the segmentation stage will be run with the frames that are detected by the first stage. Further, in the previous sections we did not regard false positives as errors in the LP detection stage, because a false positive could be discarded at a later stage. Here the false positives will be discarded if the character segmentation produces too few segments, and will be recorded as a failed detection otherwise. Additionally, since the test set contains only Norwegian LPs, we have used two separate SVMs to do character recognition. The first 2 characters in a Norwegian LP are always letters, and the following 4 or 5 are always numerals. This allows for training two different classifiers, that can have a reduced set of classes to classify. The main motivation for doing this is to avoid problems with ambiguous characters, such as O-0, B-8, and I-1. As discussed in section 5.3.2, the classifier is trained with the *one-against-one* strategy, which produces an ensemble of $k(k-1)/2$ classifiers. Reducing the

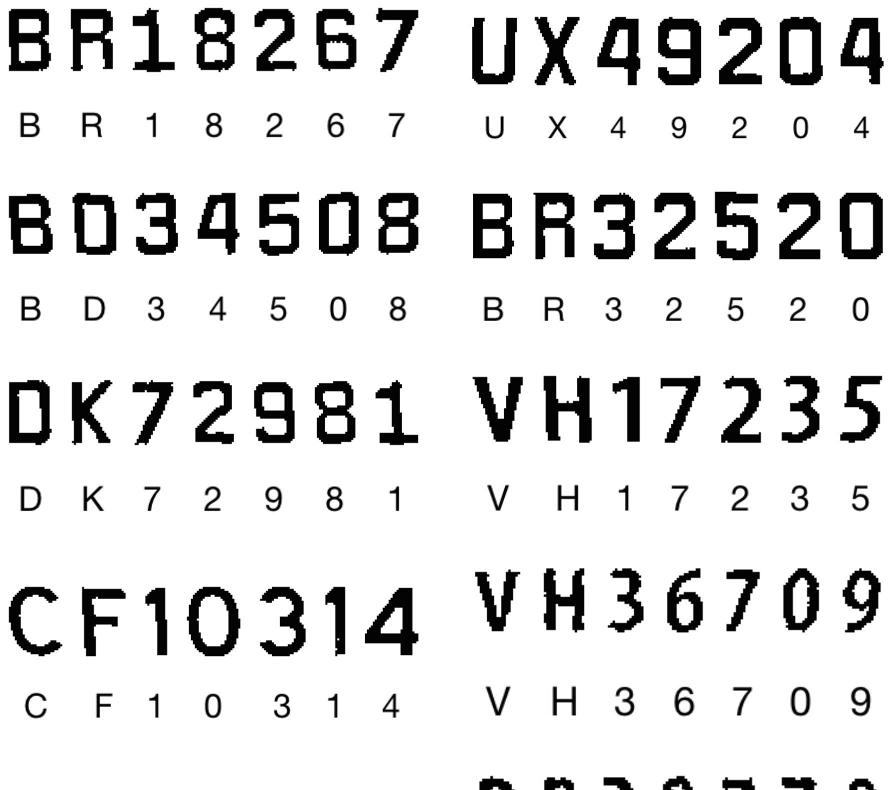


Figure 6.5: Examples of character recognition.

METHOD	ACCURACY	EFFICIENCY
LP detection	98.90 %	31.82 ms
Character segmentation	85.37 %	18.33 ms
Character recognition	99.13 % (95.88 %)	0.44 ms (1.05 ms)
Total	**83.70 %**	**50.59 ms**

(a) Results for 640×480

METHOD	ACCURACY	EFFICIENCY
LP detection	99.09 %	109.09 ms
Character segmentation	84.32 %	24.33 ms
Character recognition	99.12 % (96.28 %)	0.45 ms (0.98 ms)
Total	**82.97 %**	**133.87 ms**

(b) Results for 1280×720

METHOD	ACCURACY	EFFICIENCY
LP detection	98.18 %	266.56 ms
Character segmentation	84.26 %	25.97 ms
Character recognition	99.34 % (96.92 %)	0.45 ms (1.19 ms)
Total	**82.78 %**	**292.98 ms**

(c) Results for 1920×1080

Table 6.7: Complete ALPR system results.

number of classes will have a positive impact on the classifiers processing time, since the processing time is quadratically dependent on the number of classes.

The results for the complete tests are shown in table 6.7. Note that the parenthesized accuracy and efficiency for character recognition are the values produced by a single SVM for all character classes.

We observe that both accuracy and efficiency of the dual-SVM is far better than the single SVM version (parenthesized in table 6.7). Further we observe that all of the other efficiency results and accuracy results cohere with the results from the previous sections. Image size does not seem to have a significant effect on accuracy in any of the

METHOD	SIZE	ACCURACY	EFFICIENCY
Yang et al. (2012)	600×330	95.3 %	54 ms
Zheng et al. (2013)	640×480	96.4 %	<< 100 ms
Anagnostopoulos et al. (2006)	1024×768	96.5 %	111 ms
Proposed system	640×480	98.90 %	31.82 ms

Table 6.8: Performance of license plate detection in the literature.

three stages, and so we pragmatically select the smallest image size as a reference result since it's efficiency is by far the best.

6.5 EVALUATION

Tables 6.8, 6.9, and 6.10 shows how the proposed system compares to the methods in the reviewed literature. A general observation is that the proposed system is considerably more efficient than any of the other methods. This may be due to recent advances in hardware. On the other hand, the processing times given for the proposed system are all from testing on an iPhone 6, which may be comparable with older workstation computers with regards to processing power.

The method by Zheng et al. (2013) is the most similar method from the literature to the proposed system's LP detection stage. Zheng et al. use a cascade classifier with 6 stages. The first 2 stages uses features based on edge image statistics, while the 4 last stages are based on Haar-like features. With these 6 stages, they are able to achieve a good accuracy, and a reasonable efficiency. The proposed system is based on the same cascade classifier algorithm, but uses 15 stages, and is based on MB-LBP features. Our tests reveal that the MB-LBP features can more efficiently and accurately discriminate LP regions than the Haar-like feature are able to. This leads us to believe that the accuracy achieved by Zheng et al. has a large contribution from the first two edge statistics stages. Another detail that separates this method from the proposed system is the use of edge images. Zheng et al. uses Sobel edge images for both training and classification, while the proposed system uses normal greyscale images. Initial tests were done to investigate the promise of using edge images, but were quickly discarded due to alarmingly high numbers of false positive detections. Additionally, we found that using only 6 stages for both

METHOD	SIZE	ACCURACY	EFFICIENCY
Zheng et al. (2013)	640×480	98.82 %	<< 100 ms
Anagnostopoulos et al. (2006)	1024×768	89.1 %	37 ms
Proposed system	640×480	85.37 %	18.33 ms

Table 6.9: Performance of character segmentation in the literature.

MB-LBP and Haar-like features was infeasible due to the overwhelming number of false negative and false positive detections. The tests done by Zheng et al. does not detail any specific processing times for each of the stages in the ALPR task, but it is noted that the entire task is completed in less than 100 ms. Given this, we believe that the proposed system is at least as good as the method by Zheng et al. (2013) with regards to efficiency, and is decidedly more accurate.

The color difference edge image filtering method of Yang et al. (2012) produces very different results from the implementation explored in this system. Our tests revealed an accuracy of 88.27 % for a modified implementation, while the tests by Yang et al. shows an accuracy of 95.3 %. There is a notable discrepancy here, which is not well understood. The modifications made to our implementation strictly improved the results of the method, but still it is well away from achieving the same accuracy claimed by Yang et al.. Moreover, we note that there are some issues with the method, detailed in section 3.1.1, that if not corrected would prevent the method from any form of accurate detection. We suspect that there must be some aspects of the method not detailed in the publication that are essential to achieving the claimed accuracy.

The second stage of the ALPR task, character segmentation, is the stage where the results of the proposed system differs most from the literature. Both implementations are based on corresponding methods from the literature. The statistical character segmentation by Zheng et al. (2013) claims an accuracy of 98.82 %, while our modified implementation achieves 85.37 % at it's best. The SCW method of Anagnostopoulos et al. (2006) achieves an accuracy of 89.1 %, while our modified algorithm achieves 72.89 % at it's best. These accuracy rates reflects the number of correctly segmented LPs, and not the number of correctly segmented characters. The character accuracy for both methods are considerably better, 91.86 % and 93.74 % respec-

METHOD	ACCURACY	EFFICIENCY
Wen et al. (2011)	98.41 %	5 ms /char.
Anagnostopoulos et al. (2006)	89.1 %	18.3 ms /char.
Kocer and Cevik (2011)	98.17 %	n/a
Proposed system	99.13 %	0.44 ms /LP

Table 6.10: Performance of character recognition in the literature.

tively. The LP accuracy for both methods are negatively influenced by an unfortunate distribution of either false negatives or false positives. Ideally a method would either segment either all or no characters correctly, but in this instance the incorrect segmentations are spread over many different images, leading to a lower LP accuracy. This indicates that the method could use improvement in correctly identifying true positives and true negatives.

The third and last stage of the ALPR task is character recognition. The proposed SVM method achieves very convincing results, with the best accuracy at 99.13 %. The corresponding efficiency is 0.44 ms per LP. There are some variations in efficiency in the tests. Changing between one classifier for all classes, or two classifiers that each cover numerals or letters has a profound effect on the accuracy. Obviously, separating numerals and letters is contingent on having prior knowledge of the LP format, and is not always possible.

The most comparable method in the literature is the SVM classifiers by Wen et al. (2011). Their results shows a best accuracy at 98.41 %, and a corresponding efficiency of 5 ms per character. The best efficiency reported is another SVM variation, with 1 ms per character and an accuracy of 97.55 %. The MLP implementation by Kocer and Cevik (2011) shows an accuracy of 98.17 %, but the efficiency is not accounted for. The PNN by Anagnostopoulos et al. (2006) is far surpassed by both methods. We observe that the results of the proposed system is comparable, and also competitive to the results by Wen et al.. The accuracy of the proposed method is slightly higher, and the efficiency is magnitudes faster.

7 DISCUSSION AND CONCLUSION

7.1 DISCUSSION

The two research goals, A and B, stated in chapter 1 will be presented and discussed below.

A. Determine if ALPR is feasible for a mobile device, with regards to efficiency and accuracy. ...

The first part of research goal A is specifically concerned with a mobile device's capabilities to run an ALPR system. ALPR can possibly involve many computationally intensive operations, and it is not given that a mobile device like the iPhone may be sufficient to host such a system. The results in chapter 6 shows that all three parts of the ALPR task can be solved on a mobile device. The ALPR system detailed in this report has been implemented and tested for the recent iPhone mobile devices. There is a caveat with the character segmentation accuracy. Both LP detection and character recognition shows accuracy around 99 %, while character segmentation has it's highest isolated accuracy at 87 % (table 6.4). There is more than one way of measuring accuracy for this stage. Here, only completely correctly segmented plates is recorded as correct, and all other instances as incorrect. A more forgiving way of measuring accuracy is to measure how many percent of the characters that were correctly segmented. While this gives a good indication of a method's capabilities, it falls short in a more practical setting. An LP is composed of parts, i.e. characters, but in the real world an LP is an atomic entity which is indivisible. As a consequence, the accuracy of the character segmentation stage must be held to this constraint in order to assess the success of the method. The three stages in the ALPR task is executed consecutively, and each stage is dependent on the results from the previous. This entails that the final accuracy of the system is the product of the accuracy of each of the stages. The character segmentation method shows that it is able to extract characters in most settings, but is often challenged by dynamic illumination conditions, and dirty or partially covered LP characters. LP detection and character recognition appears as much more robust, and show much greater capability of handling noise. There are of course some artifacts with these stages as well, but in a much smaller sense than the character segmentation. The results in chapter 6 shows that ALPR is feasible for a mobile device, but it also shows that character segmentation is a stage that needs more work in order to give satisfactory accuracy results. We strongly believe that the shortcomings of the character segmentation stage is not a consequence of the mobile platform, but rather a problem with the method itself. The combined efficiency of the ALPR system is around 50 ms, which gives the ability to process up to 20 images per second.

A. *(continued) ...* If possible, also investigate the possibility for a general purpose system, that is not dependent of the LP's design.

The second part of research goal A is concerned with the generality of the ALPR system. Some systems are based on template matching, like Quan et al. (2009). Typically, a template matching method may achieve a very high accuracy, since the format of the LP and some distinguishing marks are known a priori. This does however put constraints on the possible types of LPs that can be detected and recognized by the method. Moreover, such a method may be prone to errors when given images that show partially covered, or rotated or skewed LPs. The second part of research goal A addresses this, with a purpose to find methods that are not geared towards a specific LP design. The proposed ALPR system has been designed with this in mind. There are however some remarks regarding this. The LP detection stage consist of a cascade classifier, that is trained with images of LPs. The set of training images is constructed from a smaller set of original images, where the original images are rotated around their x-, y-, and z-axes, and different types of distortion are applied to produce more training examples. This is done to avoid overfitting the data set, but also to generalize the appearance of an LP. We will however note that the entirety of the test set is made up of Norwegian LPs, and so we cannot speak to the generality of the method without having performed any tests. We propose this as a point for further work. Another consideration is that the complete test of the ALPR system, detailed in section 6.4, made one assumption regarding the LP format. The assumption was that Norwegian LPs always contains

2 letters followed by 4 or 5 numerals. This was done in part to remove ambiguity from the classification for numerals and characters that appear similar. Examples of such pairs of letters and numerals are O-0, B-8, and I-1. Another reason to split numerals from characters is that the SVM classifier is actually an ensemble classifier, whose classification time is quadratically dependent on the number of classes. By removing the 10 numeral classes from the general classifier it's classification time is dramatically improved, and as we observe from the results, classification of an entire LP's characters can be done in about 0.5 ms! We must note that this assumption regarding the distribution of letters and numerals goes against the notion of generality. However, the results from both 5-fold and 10-fold cross validation for the SVM (table 6.6) shows that the classifier can generalize well over the data set, and shows very good results also here.

B. Find which methods are feasible for live video processing, with regards to efficiency and accuracy. ...

Research goal B states that we wish to uncover methods that are feasible for video processing. As it turns out, the methods that shows the highest accuracy are also the methods that show the best efficiency. This means that there are no conflicts of interest between still image processing and live video processing. As discussed in section 2.2, we have not considered video processing beyond the scope of processing a video stream frame by frame. The cascade classifier is well suited for live video processing, because of it's discriminative nature. The cascade uses very little time to discard parts of an image, or a video frame image, that obviously does not contain an LP, while using more processing power on parts that are less obvious. Greediness of this kind means that negative images will be processed even more efficiently than positive images, allowing the system to save resources. In comparison, the color difference edge image filtering method would in any case need to do a full processing of every image before being able to discard a negative image. Disregarding accuracy, the cascade classifier is still much better suited for live video processing than any other method for LP detection investigated in this thesis. The character segmentation stage is not entirely as dynamic as the cascade classifier, but still has two possible efficiency gains in the algorithm for negative regions. The algorithm performs two different filtering passes over the possible character regions. This means that the algorithm can terminate if there are too few possible character regions left after the first filtering pass. The algorithm starts with a character height estimation, which also provides a point for early termination if the estimated character height proves to be too short for any realistic hope of character recognition. The final stage, character recognition, does not provide any points of early termination for negative classifications. The classifier will try to classify all characters regardless of the probability of previous characters being correctly classified. Since the character recognition classifier is an ensemble classifier, it does provide a probability distribution for each test example over all possible classes. It would be possible to use this distribution to determine if the current character has a very low probability of being correctly classified. This in turn would mean that the total probability of correctly recognizing all characters in the LP is equally compromised. However, our experience with the probability distribution produced by the ensemble classifier indicates that some characters will in most cases have a low probability for the correct classification, while still being the peak of the distribution. We therefore deem it prudent to view the probability distribution as more indicative of peaks, rather than an absolute distribution.

B. *(continued)* ... It is also desirable to uncover if there is a tradeoff between accuracy and efficiency, and find a possible balance between the two.

The second part of research goal B is concerned with the possibility of having to sacrifice accuracy for efficiency, or vice versa. The results in chapter 6 shows that there is no such tradeoff to be considered. The most apparent tradeoff we have observed is between both efficiency and accuracy, and generality. The character recognition results for the complete ALPR classifier shows that accuracy and efficiency is improved when assuming a certain sequence of letters or numerals in the LP format. The core of the problem stems from the pairs of similar numerals and letters. Distinguishing between these pairs can be particularly difficult for one of the older Norwegian LP fonts. The choice of using such prior knowledge can limit a system's ability to work across different types of LP formats. On the other hand, very many countries have a predetermined format.

As a final thought regarding the accuracy of the system, we would like to point out that the character segmentation stage has no form of AI. All the methods in the reviewed literature are based on statistical methods and image processing techniques. A possible explanation for the low accuracy for this system could be that this problem is not easily solved by conventional statistical methods. Some form of intelligent learning could possibly help improve these filtering techniques; in the very least, a form of evolutionary algorithm could be one way to find the most discriminative features of non-character elements. We would like to propose this as a point for further work, exploring the possibility of using an AI inspired technique do improve the reliability and accuracy of this stage.

7.2 CONCLUSION

In this thesis we have shown how an ALPR system may be constructed for a mobile device, specifically the family of iOS devices. The main concerns for a mobile device implementation is the limited processing power of the device, and it's ability to capture images from many different angles and distances. The choice of methods therefore becomes a vital part of the process. The corpus of literature on the ALPR subject shows that there are many different possibilities for implementation, although few of them are designed for a mobile device implementation. Our work has consisted of uncovering, assessing, and improving existing methods.

The final implementation shows that ALPR is feasible for a mobile device. The ALPR task is split into three parts, LP detection, character segmentation, and character recognition. LP detection is done with a cascade classifier, inspired by the work of Viola and Jones (2001), and is tasked with finding the location of an LP in an image. The original cascade classifier is based on Haar-like features. We have found that MB-LBP features are both more efficient and accurate for LP detection. Character segmentation is performed on detected LP regions. Character segmentation should identify and extract characters from the region. This is done by binarizing the region, and filtering out non-character elements with statistical methods. The final stage, character recognition, is tasked with determining which characters the character segmentation stage has extracted from the region.

Both LP detection and character recognition are shown to have accuracy around 99 %, which is competitive against most other methods in the literature, especially considering that the system is constructed for recognition from arbitrary angles and distances. However, character segmentation has an accuracy around 85 %, which is too low for the entire system to be competitive against other methods in the literature. The combined accuracy of the system is 83.70 %. The average processing time of an image containing an LP is around 50 ms (32 + 18 + 0.5), allowing the system to process up to 20 images per second. We also must remark that the final stage, character recognition, has an average efficiency of under 0.5 ms. Given this, the system is capable of both still image processing, as well as some live video processing. The nature of the LP detection stage cascade classifier allows for more efficient processing of negative images, meaning that a negative image may be processed faster than a positive image. Character segmentation and character recognition should in any case not be necessary for negative images.

The character segmentation stage is the least accurate stage in the system, and we propose further work, especially concerning learning based methods and evolutionary methods, to improve the accuracy of this stage. Additionally, we propose that the generality of an MB-LBP-based cascade classifier should be explored, to uncover the potential of the method. As discussed in the previous section, the use of a priori knowledge regarding the format of the LP can increase both efficiency and accuracy for character recognition. We do however note that a decision on whether or not to assume a certain LP format can be made at a later time.

BIBLIOGRAPHY

Vahid Abolghasemi and Alireza Ahmadyfard. An edge-based color-aided method for license plate detection. *Image and Vision Computing*, 27(8):1134 – 1142, 2009. (Cited on page 31.)

C.-N.E. Anagnostopoulos, I.E. Anagnostopoulos, G. Tsekouras, G. Kouzas, V. Loumos, and E. Kayafas. Using sliding concentric windows for license plate segmentation and processing. In *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, pages 337–342, Nov 2005. (Cited on page 14.)

C.-N.E. Anagnostopoulos, I.E. Anagnostopoulos, V. Loumos, and E. Kayafas. A license plate-recognition algorithm for intelligent transportation system applications. *Intelligent Transportation Systems, IEEE Transactions on*, 7(3):377–392, Sept 2006. (Cited on pages 14, 36, 38, 42, 56, 60, 62, 63, 81, 82, and 83.)

C.-N.E. Anagnostopoulos, I.E. Anagnostopoulos, I.D. Psoroulas, V. Loumos, and E. Kayafas. License plate recognition from still images and video sequences: A survey. *Intelligent Transportation Systems, IEEE Transactions on*, 9(3):377–391, Sept 2008. (Cited on page 3.)

Apple Inc. iOS Technology Overview, 2014. URL `https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html`. Last checked: 2015-04-21. (Cited on page 50.)

Abraham Straume Bah. Fast and reliable recognition of car license plates. Master's thesis, Norwegian University of Science and Technology (NTNU), 2014. (Cited on pages 4, 16, 35, 39, and 47.)

Kristin P. Bennett and Colin Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explor. Newsl.*, 2(2):1–13, December 2000. (Cited on page 25.)

A. Capar and M. Gokmen. Concurrent segmentation and recognition with shape-driven fast marching methods. In *Pattern Recognition,* *2006. ICPR 2006. 18th International Conference on*, volume 1, pages 155–158, 2006. (Cited on page 40.)

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011. (Cited on page 64.)

Rafael C Gonzalez, Richard E Woods, and Steven L Eddins. *Digital image processing using MATLAB*. Gatesmark Publishing, 2 edition, 2009. (Cited on pages 8, 10, 11, 15, 21, and 39.)

Google Inc. Tesseract OCR, 2015. URL `https://code.google.com/p/tesseract-ocr/`. Last checked: 2015-04-21. (Cited on pages 40, 51, 52, and 65.)

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, Mar 2002. (Cited on page 26.)

Itseez. OpenCV Web page, 2014. URL `http://opencv.org`. Last checked: 2015-04-21. (Cited on pages 51 and 52.)

H. Erdinc Kocer and K. Kursat Cevik. Artificial neural networks based vehicle license plate recognition. *Procedia Computer Science*, 3(0):1033 – 1037, 2011. (Cited on pages 31, 40, 42, and 83.)

Erwin Kreyszig. *Advanced engineering mathematics*. John Wiley and sons Inc, 9 edition, 2006. (Cited on page 8.)

Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning multi-scale block local binary patterns for face recognition. In Seong-Whan Lee and StanZ. Li, editors, *Advances in Biometrics*, volume 4642 of *Lecture Notes in Computer Science*, pages 828–837. Springer Berlin Heidelberg, 2007. (Cited on pages 19 and 54.)

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. (Cited on pages 23 and 24.)

New Designs Unlimited, LLC. OpenALPR web page, 2015. URL `http://www.openalpr.com`. Last checked: 2015-04-21. (Cited on pages 65 and 77.)

Chirag Patel, Dipti Shah, and Atul Patel. Automatic number plate recognition system (anpr): A survey. *International Journal of Computer Applications*, 69(9):21–33, May 2013. (Cited on page 3.)

Jin Quan, Quan Shuhai, Shi Ying, and Xue Zhihua. A fast license plate segmentation and recognition method based on the modified template matching. In *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on*, pages 1–6, Oct 2009. (Cited on pages 31, 39, 41, and 86.)

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3 edition, 2003. (Cited on pages 24, 27, and 64.)

Ray Smith. An overview of the Tesseract OCR engine. In *ICDAR*, volume 7, pages 629–633, 2007. (Cited on pages 41 and 52.)

Nicolas Thome, Antoine Vacavant, Lionel Robinault, and Serge Miguet. A cognitive and video-based approach for multinational license plate recognition. *Mach. Vis. Appl.*, 22:389–407, 2011. (Cited on pages 31 and 41.)

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*, volume 1, pages I–511–I–518 vol.1, 2001. (Cited on pages 19, 20, 26, 28, 30, 34, 35, 53, 54, 68, and 89.)

Ying Wen, Yue Lu, Jingqi Yan, Zhenyu Zhou, K.M. von Deneen, and Pengfei Shi. An algorithm for license plate recognition applied to intelligent transportation system. *Intelligent Transportation Systems, IEEE Transactions on*, 12(3):830–845, Sept 2011. (Cited on pages 41 and 83.)

Xing Yang, Xiao-Li Hao, and Gang Zhao. License plate location based on trichromatic imaging and color-discrete characteristic. *Optik - International Journal for Light and Electron Optics*, 123(16):1486 – 1491, 2012. (Cited on pages 15, 17, 31, 32, 34, 55, 56, 81, and 82.)

Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and Stan Z. Li. Face detection based on multi-block lbp representation. In Seong-Whan Lee and Stan Z. Li, editors, *Advances in Biometrics*, volume 4642 of *Lecture Notes in Computer Science*, pages 11–18. Springer Berlin Heidelberg, 2007. (Cited on pages 28, 35, and 36.)

Danian Zheng, Yannan Zhao, and Jiaxin Wang. An efficient method of license plate location. *Pattern Recognition Letters*, 26(15):2431 – 2438, 2005. (Cited on page 31.)

Lihong Zheng, Xiangjian He, Bijan Samali, and Laurence T. Yang. An algorithm for accuracy enhancement of license plate recognition. *Journal of Computer and System Sciences*, 79(2):245 – 255, 2013. (Cited on pages 31, 34, 35, 37, 40, 56, 57, 61, 68, 81, and 82.)

Part III

APPENDIX

A IOS DEVICES DATA

The data in the following tables can be found at `http://en.wikipedia.org/wiki/List_of_iOS_devices`, last accessed 2015-03-31.

MODEL	YEAR	PROCESSOR	RAM	GPU	CAMERA	VIDEO	BATTERY
iPhone	2007	412 MHz, 32-bit	128 MB	103 MHz	2 MP f/2.8	n/a	1400 mAh
iPhone 3G	2008	412 MHz, 32-bit	128 MB	103 MHz	2 MP f/2.8	n/a	1150 mAh
iPhone 3GS	2009	600 MHz, 32-bit	256 MB	150 MHz	3 MP f/2.8	480p 30 fps	1219 mAh
iPhone 4	2010	800 MHz, 32-bit	512 MB	200 MHz	5 MP f/2.8	720p 30 fps	1420 mAh
iPhone 4S	2011	2x800 MHz, 32-bit	512 MB	2x200 MHz	8 MP f/2.4	1080p 30 fps	1432 mAh
iPhone 5	2012	2x1.3 GHz, 32-bit	1 GB	3x266 MHz	8 MP 1.4µ f/2.4	1080p 30 fps	1440 mAh
iPhone 5C	2013	2x1.3 GHz, 32-bit	1 GB	3x266 MHz	8 MP 1.4µ f/2.4	1080p 30 fps	1510 mAh
iPhone 5S	2013	2x1.3 GHz, 64-bit	1 GB	4x450 MHz	8 MP 1.5µ f/2.2	1080p 30 fps, 720p 120 fps	1560 mAh
iPhone 6	2014	2x1.4 GHz, 64-bit	1 GB	4x450 MHz	8 MP 1.5µ f/2.2	1080p 30-240 fps	1810 mAh
iPhone 6 plus	2014	2x1.4 GHz, 64-bit	1 GB	4x450 MHz	8 MP 1.5µ f/2.2	1080p 30-240 fps	2915 mAh

Table A.1: iPhone hardware specifications

MODEL	YEAR	PROCESSOR	RAM	GPU	CAMERA	VIDEO	BATTERY
iPad	2010	800 MHz, 32-bit	512 MB	200 MHz	n/a	n/a	6613 mAh
iPad 2	2011	2x800 MHz, 32-bit	512 MB	2x200 MHz	0.7 MP	720p, 30 fps	6579 mAh
iPad Mini	2012	2x1.0 GHz, 32-bit	512 MB	2x250 MHz	5 MP	1080p, 30 fps	4440 mAh
iPad 3	2012	2x1.0 GHz, 32-bit	1 GB	3x266 MHz	5 MP	1080p, 30 fps	11487 mAh
iPad 4	2012	2x1.4 GHz, 32-bit	1 GB	4x533 MHz	5 MP	1080p, 30 fps	11560 mAh
iPad Mini 2	2013	2x1.3 GHz, 64-bit	1 GB	4x450 MHz	5 MP	1080p, 30 fps	6471 mAh
iPad Air	2013	2x1.4 GHz, 64-bit	1 GB	4x450 MHz	5 MP	1080p, 30 fps	8827 mAh
iPad Mini 3	2014	2x1.3 GHz, 64-bit	1 GB	4x450 MHz	5 MP	1080p, 30 fps	6471 mAh
iPad Air 2	2014	3x1.5 GHz, 64-bit	2 GB	8x450 MHz	8 MP	1080p, 30 fps	7340 mAh

Table A.2: iPad hardware specifications

OS VERSION	RELEASE DATE	DEVICES RELEASED WITH VERSION
iPhone OS 1	June 29, 2007	iPhone
iPhone OS 2	July 11, 2008	iPhone 3G
iPhone OS 3	June 17, 2009	iPhone 3GS, iPad
iOS 4	June 21, 2010	iPhone 4, iPad 2
iOS 5	October 12, 2011	iPhone 4s, iPad 3
iOS 6	September 19, 2012	iPhone 5, iPad 4, iPad Mini
iOS 7	September 18, 2013	iPhone 5s, iPad Air, iPad Mini 2
iOS 8	September 17, 2014	iPhone 6, iPhone 6 plus, iPad Air 2, iPad Mini 3

Table A.3: iOS platform release history

B SOFTWARE AND TOOLS USED

Software

XCODE Xcode, version 6.2, IDE used for development.

COCOAPODS CocoaPods, version 0.36.0.beta.2, iOS package manager software.

OPENCV Third party library for image processing. The library is written in C/C++ and has been adopted to other languages through wrapper classes.

TESSERACT OCR Tesseract OCR is an OCR library. Tesseract can be used for general OCR on a page of text, or can be trained for more specific use cases.

Hardware

APPLE MACBOOK PRO Apple MacBook Pro 15-inch notebook computer, 2,5 GHz Intel Core i7, 16 GB 1600 MHz DDR3

UBUNTU SERVER Ubuntu server, 32 cores Intel Xeon 2.6 GHz, 128 GB RAM, provided by NTNU.

APPLE IPHONE 6 iPhone 6 smart phone, running iOS 8.2, see appendix A for technical specifications.

Data

DATA SET FROM TRONDHEIM, NORWAY Data set of license plates, 2888 images of various cars with focus on their license plates, provided by WTW AS.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

`http://code.google.com/p/classicthesis/`

Final Version as of June 9, 2015 (`classicthesis` version 1.0).