# Character Recognition in LEDs and LCDs

**A dissertation submitted to The University of Manchester for the degree of**

**Master of Science**

**in the Faculty of Engineering and Physical Sciences**

**2015**


## Fadi Khader Issa Tannous

## School of Computer Science

# List of Contents

**Total Word Count: 21900**

# List of Figures

# List of Tables

# List of Equations:

# Abstract

In this day and age, the majority of household appliances feature LED and LCD digital displays. Digital displays inherently enhance the interaction between man and machine. They provide a graphical user interface that exploits the human's ability to perceive light and notably deduce meaning out of shapes, symbols and characters. Nevertheless, the prevalence of digital displays in electronics and household appliances poses a significant barrier to the blind and visually impaired.

This Dissertation proposes a quasi-real-time android application that tackles the challenge of interpreting the contents of LED and LCD digital displays. This proposal aims to provide assistance for the visually-impaired via their smart phone. The application accepts a wide range of display environments as it addresses the core visual challenges of digitizing digital display content, from glare and specular reflections to high contrast bleeding and oversaturation. On the other hand, this project recognizes the prevalence of seven-segment displays in household appliances as it employs classification techniques tailored to the seven-segment encoding scheme. The application is split into three components, segmentation, classification and the user interface. First and foremost, this project mainly focuses on the segmentation component as it appreciates the significance of accurate image segmentation and its contribution to overcome the aforementioned visual challenges. The classification component integrates an ensemble of classifiers. Each classifier is fed a unique image capture and a majority vote between all classifiers is assembled for accurate classification. Furthermore, the application incorporates native Android speech recognition and Text-To-Speech libraries together in an auditory user interface that facilitates the operation for visually impaired users. The application is initially trained to recognize basic English alphanumeric characters and a small set of basic symbols. Each component is subject to individual evaluation based on a set of pre-defined labeled training data where each component's ability is exercised separately. The application is implemented in Java; using the Android Studio software development kit, and C++; integrating the OpenCV computer vision library.

# Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

1. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

2. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

3. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the dissertation, for example graphs and tables ("Reproductions"), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

4. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/display.aspx?DocID=487 ), in any relevant Dissertation restriction declarations deposited in the University Library, The University Library's regulations (see http://www.manchester.ac.uk/library/aboutus/regulations ) and in The University's Guidance for the Presentation of Dissertations.

# Acknowledgements

I would like to thank my supervisor, Dr. Tim Morris, for his guidance throughout this project. Without your continuous support, motivation and supervision, this project would never be a reality.

I would also like to thank my parents and sisters for their undivided moral and financial support.

# 1 Introduction

Amidst the information age, one should not be surprised that the majority of households rely heavily on electric appliances; ranging from entertainment equipment e.g. TVs, speakers, projectors and media hubs, to the basic necessities e.g. washing machines, refrigerators, dish washers and microwave ovens. The growth in the utilization of such appliances has been evident throughout the past 50 years, however, very little research examines the actual rate of growth. Nevertheless, the British Department of Energy and Climate Change (DECC) reported a growth of 1.7% per year for the total amount of electricity consumption by household domestic appliances between the years 1970 and 2013 [1]. Moreover, the DECC report accounts for an average of 50% improvement in energy efficiency of household appliances between 1990 and 2013. The growth of electricity consumption by household appliances in conjunction with the improvements in energy efficiency indicates at least a linear growth rate in the utilization of household appliances. Evidently, as these electronic appliances progress in terms of efficiency, design and human-interaction they tend to become increasingly dependent on LED and LCD digital displays. Despite the various benefits that digital displays offer in the field of human-machine interaction, digital displays often hinder the operation of electronic appliances for visually impaired users.

## 1.1 OCR Challenges

Most digital displays share the common seven-segment character set encoding, and feature high-contrast panels. Despite the similarity between them, the majority of Optical Character Recognition (OCR) systems currently available are baffled when faced with the task of reading digital displays. Indisputably, some of the most popular and accurate OCR systems on the market today will suffer from lower accuracy when dealing with digital displays. Some even state it clearly in the manual e.g. The KNFB Reader Mobile [2]. Realistically, most OCR systems are designed to address the challenge of digitizing text on printed documents , however, are not equipped to tackle the challenges that digital displays give rise to. To begin with, the OCR system must be designed to efficiently segment each possible character on the digital display and ignore all the background clutter. The prevalence of monochrome digital displays in conjunction with low-contrast panels poses the main challenge for accurate segmentation. Monochrome displays and colored displays where the background

contains the same color as the characters, render most color-based segmentation techniques useless. Moreover, techniques that exploit spatial proximity in conjunction with color information often yield the best results. Nevertheless, the majority of these techniques are computationally intensive and infeasible for smartphone implementations. In addition, the OCR system must also consider the numerous effects that the digital display's outer surface (usually plain glass) has on the captured image. The visibility of digital displays can be easily impaired by light reflections that are a direct consequence of the glass outer surface e.g. glare (as illustrated in figure 1.1 below). Light reflections can be specular or diffuse, depending on the roughness of the outer surface. Consequently, the visibility is somewhat dependent on the lighting conditions, the reflectiveness and roughness of the outer display surface and the location/angle of the camera relative to the display surface normal. Moreover, LED displays usually feature very high-contrast panels, which lead to obscure fuzzy blobs when captured on camera. This is a consequence that is more commonly referred to as 'high-contrast bleed' as shown in figure 1.2 below.


*Figure 1.1: Example of Specular Reflections/Glare*


*Figure 1.2:* Top*: High-contrast Bleed* ***Versus*** Bottom*: Clear Image*

## 1.2 Motivation

The World Health Organization (WHO) categorizes four levels of visual functionality. Level 1 describes normal vision, level 2 describes moderate visual impairment, level 3 describes severe visual impairment and level 4 describes complete blindness. Moreover, the WHO estimates that 285 million people, about 4% of the Human population, are visually impaired: 39 million are classified under level 4 and 246 million are classified under 'low vision' (levels 2 and 3 combined) [3]. More to the point, in a report that studies the economic impact of partial sight blindness in the UK prepared for The Royal

National Institute of Blind People by Access Economics Pty Limited [4]; it is estimated that the percentage of visually impaired males and females will grow from 1.7% and 2.8% to 3.8% and 5.7% respectively between 2010 and 2050. The sheer size of the visually impaired community combined with the expectation of growth, highlights the significance of addressing the formerly acknowledged challenges. That is the challenges of facilitating the operation and use of technology, specifically household appliances for the visually impaired.

Relying on electronics (smartphones or camera equipped computers) to facilitate the operation of other electronics (household appliances) seems counter-intuitive. One can easily argue about the inefficiency of acquiring digital information through the visual medium against the use of more modern techniques; such as equipping household appliances with NFC, Near Field Communication, chips that would enable household appliances to efficiently communicate with smartphones. Nevertheless, between the current capabilities of the average smartphone and the excellent natural spatial localization abilities of the visually impaired, the concept of digitizing the content of (inherently digital) electronic appliances seems more feasible. Various studies analyzed and evaluated the abilities of the visually impaired in spatial localization. These studies essentially evaluated the use of other heightened senses (smell, touch, taste and hearing) for spatial localization; an ability that sight mostly controlled. The majority of these studies suggest that the visually impaired are generally excellent at spatial localization [5] [6] [7], especially when the visually impaired user is accustomed to his/her surroundings i.e. his/her home. Simultaneously, a study on helping the visually impaired users properly aim a camera emphasizes the positive attitude of blind or visually impaired users when employing assisted photography and its appeal to their community [8].

## 1.3 Main Objectives

This project has two main objectives:

1. The implementation of a quasi-real-time android application that permits visually disabled users to accurately read digital displays, particularly displays commonly found in household appliances.
2. The independent evaluation of all major components of the application over a comprehensive set of pre-defined data that describes the targeted environment.

## 1.4 General Development Approach and Methodology

This application consists of three integral components that were developed, implemented and evaluated independently. The components are meant to work in succession, where the segmentation component partitions the image into sets of pixels, the classification component classifies each set of pixels into alphanumeric characters or symbols, and finally the user interface analyzes all characters and reports the outcome.

In general, the development of all components was mainly concerned with achieving high accuracies, comparable to state-of-the-art systems, with acceptable run-time. Nevertheless, throughout the initial stages of development of every component, the only concern was accuracy. In the context of multi-core smartphones, two run-time optimization approaches stand out: using more efficient implementation of algorithms and using run-time oriented data structures. Usually when one opts for a more efficient implementation of an algorithm, there is a trade-off between accuracy and run-time. For example, the final histogram implementation embedded in this application employs city block distance metric instead of the standard Euclidean distance metric , to calculate the difference between colors, as it achieves substantially faster run-time but suffers from slightly less accurate distance measurements. On the other hand, when one opts for run-time oriented data structures it usually involves a trade-off between memory usage and access times. Back to the point, by focusing solely on accuracy, the initial development process revealed the performance upper bound. Consequently, throughout the latter

development stage of each component, every run-time optimization decision was independently assessed based on two features:

$$\Delta \mathbf{A} = \frac{new\_accuracy}{upper\_bound\_accuracy} \qquad \Delta \mathbf{T} = \frac{upper\_bound\_time - new\_time}{upper\_bound\_time}$$

*Equation 1: Accuracy Change (%)* *Equation 2: Run-time Change (%)*

The algorithms for segmentation and classification components were initially developed in MATLAB R2014a. MATLAB supports a wide range of libraries and algorithms for image processing, which renders it the perfect environment for initial development. More to the point, the second stage of development was mainly concerned with porting the algorithms from MATLAB to native java; as native java provides a much faster interface to test the algorithms than Android Studio. The porting from MATLAB to native java mainly revolved around translating pure MATLAB script to java, implementing functions that are not supported in java and embedding external libraries.

First and foremost, the OpenCV library or Open Source Computer Vision. OpenCV is a C++ library of algorithms mainly focused at real-time computer vision. Essentially, two ways exist to linking the OpenCV library to an android project: static binding or dynamic linking. Dynamic linking is more efficient, as it requires that the android operating system installs a separate OpenCV manager application that will be linked dynamically to any new application (that utilizes OpenCV). On the other hand, static binding involves incorporating the required parts of the OpenCV library into the Android Application Package or APK. Static binding is the clear choice as it facilitates an easier installation experience (no requirement for a separate OpenCV manager installation) and this project only utilizes a very small portion of the OpenCV library. Details about linking OpenCV to an android project are available in Appendix C. More to the point, this application also incorporates LIBSVM v1.04. LIBSVM is an open source machine learning library, written in C++, which implements the Sequential Minimal Optimization algorithm for Kernalized Support Vector Machines. This project also exploits the native Android Text-To-Speech and Speech Recognition libraries.

On the other hand, the client-server classification model update protocol is built on HTTP, JSON and java. The server is a java-applet running on Apache Tomcat7. Tomcat is an open-source web server developed by Apache Software Foundation. Tomcat7 is

running on a personal virtual server, provided by Ocean Digital. The virtual server boasts one i7 core clocked at 3 GHz, 512MB Ram, and a 20 GB SSD.

All testing and evaluation were implemented on Android Studio and ran on a Galaxy S4 I9505; the I9505 features a Quad-core 1.9 GHz Krait 300, 2 GB LPDDR3 RAM and runs Android OS v5.0.1. Refer to Appendix A for a comprehensive overview of all experiments, tests, and evaluations performed in this project and information about their respective implementations in the attached files.

The system architecture diagram illustrated in figure 1.3 below describes the overall implementation.

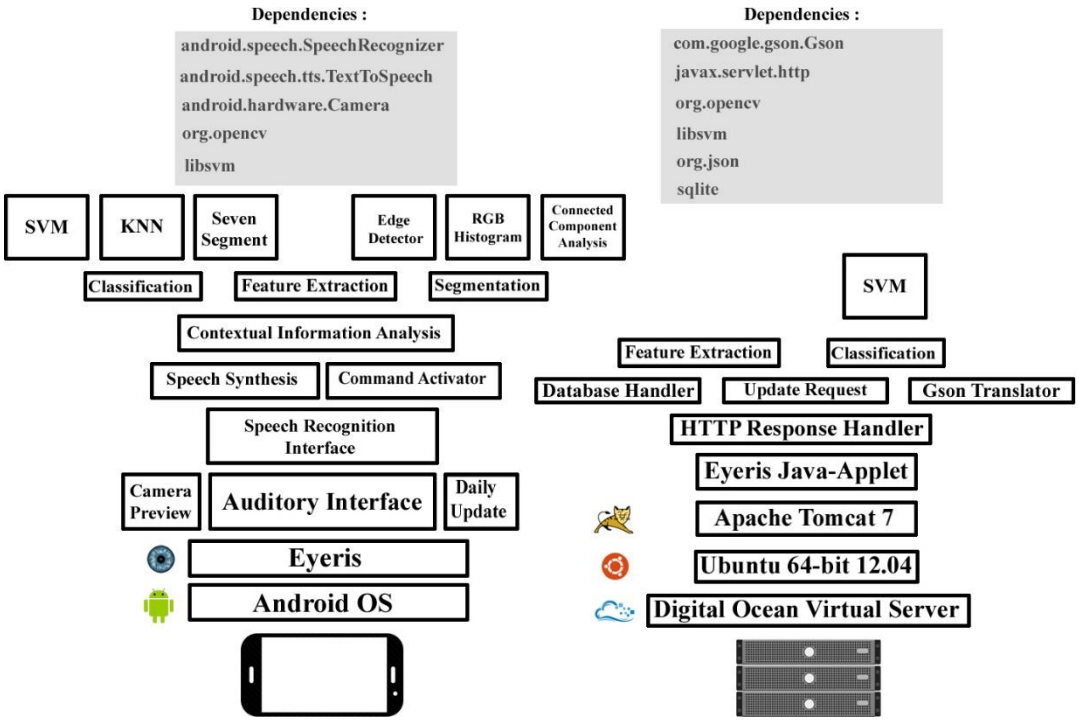



*Figure 1.3: System Architecture*

## 1.5 Contributions

The project contributions are four-fold:

1. A fast and accurate segmentation algorithm that employs edge-based detection in conjunction with color-histograms to tackle the visual complications/challenges that are a consequence of the targeted environment.
2. A classification ensemble that incorporates three unique classifiers, SVM, KNN, and a Seven-Segment density classifier, each fed a unique capture of the same environment.

3. A contextual information analysis framework that attempts to understand the true meaning behind the characters by exploiting positional contextual information, classification confidences and recognition history.
4. A simplistic http client-server protocol, where the java-based server acquires unrecognized image data from clients and awaits manual labeling. The server essentially re-trains its own classification models after labeling and then offers automatic updates to all online clients.

## 1.6 Overview of Dissertation

The remainder of the dissertation is structured as follows: chapter 2 describes a brief overview of the history of assisting the less abled, a comprehensive overview of state-of-the-art OCR-related approaches and basic knowledge of segmentation, classification, feature selection and connected component analysis. Then chapter 3 highlights the design decisions and methodology that led to the final implementation. Chapter 4 describes the final implementation in the context of an android application. Whereas, chapter 5 contains the details about the evaluation procedure for all the components of the application and discusses the evaluation results. Finally, chapter 6 concludes the report, emphasizing results, deductions and observations; while also proposing possible future improvements.

# 2 Literature Review and Background

The utilization of computers for the assistance of the physically or mentally disabled is far from new. The first instances of this concept began to surface in the mid-1980s, where a study tackled the challenge of using microcomputers to enhance the reading skills of children with learning disabilities [9]. The study proposes that computers can be programmed to deliver effective training in simple decoding skills (reading). Moreover, as the industrial form of computers (mainframes, and microcomputers) evolved into commercial and personal computers throughout the late 1980s and 1990s; the research of using computers for assisting the less abled grew exponentially.

## 2.1 Assisting the disabled

A large variety of systems that primarily tackle the assistance of the disabled are currently available. Dedicated reading devices that can read text off of printed documents are the most common and accurate. These dedicated devices are available in forms of portable devices and full-on heavy standalone systems [10]; where the latter usually achieves the best results. Moreover, a variety of dedicated devices that address learning disabilities, mobility impairment and environmental control are also commonly available today [11]. Nevertheless, platform independent software solutions are also available. The majority of these systems incorporate elements, popular in the field of artificial intelligence, that naturally tackle the challenges of mimicking the human senses and intellect; such as optical character recognition, neural networks, speech recognition and robotics.

## 2.2 OCR-Related Approaches

Throughout the recent boom in the evolution of computing performance, a wide variety of OCR-based systems began surfacing. The vast majority of these systems were fundamentally designed to tackle the digitization of printed documents across a small set of standard fonts and a small range of font sizes. These types of OCR-systems are widely available today across various platforms from smartphones and personal computers to dedicated OCR devices and websites [12]. Even though these systems are of state-of-the-art design and often yield extremely accurate results i.e. 95-100%, they

tend to fail miserably when faced with the challenge of digitizing digital displays, handwritten documents, and even unorganized printed documents. Despite the fact that some studies addressed the challenge of reading/digitizing text that can appear in various scales, fonts, colors and orientation [13]; none of them were able to achieve accuracies remotely close to OCR-systems that read printed documents. Consequently, the need for custom designed OCR-systems that tackle the task of digitizing a specific environment becomes apparent.

Systems that tackle the assistance of the visually impaired through the use of OCR are also widely available today. Unfortunately, the vast majority of these systems do not appreciate the complexities and challenges of addressing different contextual environments and often tend to yield inconsistent accuracies across diverse environments. Nevertheless, a small portion of these systems do recognize the significance of tailoring the system to address a specific environment and often make various assumptions that facilitate the operation of the system; given that the environment is consistent. To begin with, a study by the Department of Electrical and Electronics Engineering, The Kavery Engineering College, India, describes a portable camera-based system designed to assist the visually impaired in reading product labels from hand-held objects [14]. The recent study, published in April 2014, enables the users to recognize a wide variety of products based on the labeling. The proposed system recognizes the significance of addressing the various environments that the product will be in. The system ingeniously separates the handheld object that it's aiming to recognize and the environmental background; it achieves this accurate segmentation by instructing the users to shake/tremble the product in their hand. Another common related approach is the Optical Character Recognition of seven-segment display. Seven-segment displays are a form of electronic display devices that fundamentally encode characters, usually numerical characters, into sets of seven segments as shown in figure 2.1.

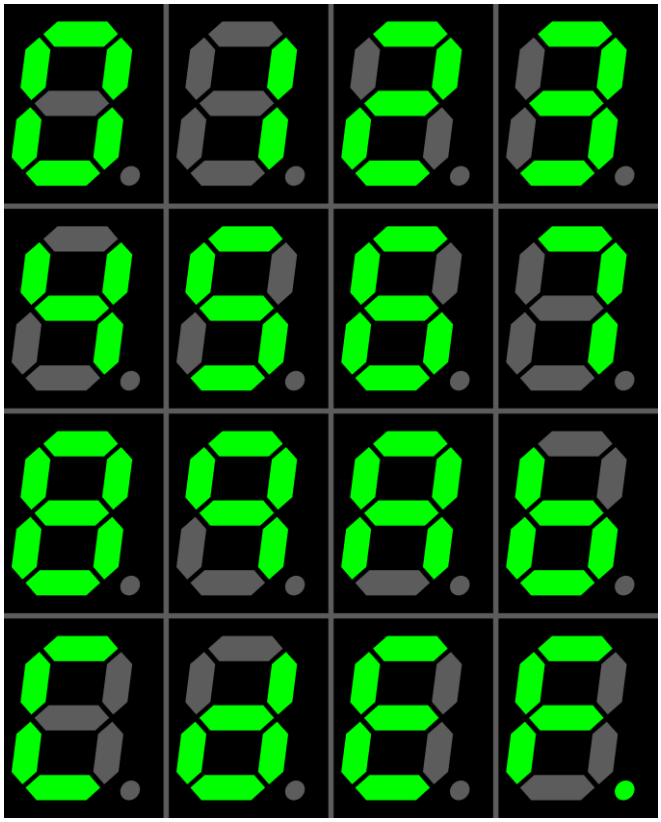


*Figure 2.1: 0-9 and a-f Encoded on a Seven-Segment Display*

The use of seven-segment displays is very common in household appliances, seven-segment Optical Character Recognition (SSOCR) has been briefly studied [15]. SSOCR is rendered trivial by the inherent simplicity of seven-segment encoding and the exploitation of decision trees can lead to very efficient implementations.

In the context of OCR-based smartphone applications, three unique implementations standout. The KNFB Reader [16] is available on the Symbian Operating System platform and iOS; the KNFB reader features state-of-the-art OCR algorithms and achieves accurate results given the correct environment. On the other hand, Text Detective [17] is available on Android OS and iOS simultaneously; Text Detective exploits multiple frames of input rather than just one image alongside a speech synthesis component that reads the output aloud. Last but not least, Prizmo [18] is available on iOS and OSX; Prizmo features support for 11 different languages and is optimized for recognizing text from books. While, all three applications are positively reviewed and undeniably able to recognize printed text on white background accurately; all three of

them tend to suffer from lower accuracy and prolonged run-time when faced with handwritten text or digital displays.

## 2.3 Segmentation and Classification

The challenges of digitizing the content of digital displays lie in developing efficient segmentation and classification implementations that tackle the inherent complexities of our specific target environment (digital displays prevalent in household appliances). As detailed in the introduction, the segmentation approach must be able to separate the wanted characters from all the unwanted clutter and also be able to cope with the complexities of the environment/surroundings. On the other hand, the classification approach must be able to incorporate contextual/historical information to speed-up the process and derive appropriate meanings. On top of that, both components; segmentation and classification, must be able to run within a feasible amount of time. The main focus of this report will be the development of suitable segmentation and classification implementations.

Segmentation is a crucial component in most OCR systems and a wide variety of approaches have been developed and rigorously studied. To begin with, the process of segmentation fundamentally attempts to separate pixels in an image into segments depending on visual resemblance. Visual resemblance can be characterized by two key descriptors; color difference and spatial proximity. Consequently, in general two types of segmentation exist; segmentation based solely on color difference and segmentation based on color difference and spatial proximity between pixels. The latter approach is obviously more computationally heavy and most implementations are not feasible on current smartphone platforms. Color-only based approaches can be essentially perceived as a Color histogram, a representation of the distribution of colors in an image. Color-based approaches or histograms fundamentally differ in the approach taken to calculate the color difference between two pixels and how much difference is tolerated to classify two different colors as one. Common histogram implementations exploit biological information about the human eye to calculate an appropriate tolerance for color differences and calculate color differences using a specific color scheme (RGB, HSV, HSL, CMYK) [19]. On the other hand, approaches based on color and spatial

proximity are commonly referred to as blob detectors; blob detectors fundamentally scan regions of images and form blobs out of similarly colored pixels that lie within certain spatial limitations [20]. The most popular and accurate segmentation techniques exploit multiple color schemes and differential geometry (for faster spatial proximity calculations) [21]. The majority of blob detection algorithms are custom tailored to fit the needs of a specific environment; as the algorithm aims to exploit the specific spatial/geometrical characteristics present in a given environment. Although some general blob detection algorithms are available that can be adapted to a specific environment; these algorithms tend to focus on segmentation accuracy and are usually very computationally demanding. One other segmentation approach that is significant in the context of OCR is edge detection. Edge detection is a mathematical approach that aims to identify points on the edge of significant objects in an image. The approach exploits the relative change in image brightness and primarily assumes that a sharp change in image brightness indicates an edge. Unfortunately, edge detectors usually tend to fail in coping with noise, especially in forms of glare and light reflections.

On the other hand, classification is a very crucial component in all OCR systems and an enormous amount of research tackles the efficiency of its possible implementations. Four unique classification methods stand out in the context of OCR [22]. Support Vector Machines, commonly referred to as SVMs, are supervised learning models that analyze data and recognize patterns. SVMs are considered to be one of the most powerful classification techniques and can efficiently accomplish non-linear classification using the kernel trick [23]. Artificial Neural Networks, commonly referred to as ANN, are statistical learning algorithms inspired by the biological neural networks, particularly the human brain. ANNs are used to estimate functions that rely on huge sets of input; ANNs have been commonly used to solve machine learning tasks including character and speech recognition. ANNs are known to be computationally intensive but are known to produce accurate results and in some cases provide a more flexible learning algorithm than most (works with Supervised or Unsupervised learning). K-Nearest Neighbor, commonly referred to as KNN, is a non-parametric method used generally for classification and fundamentally classifies the data based on its relationship among its k nearest neighbors. KNN is a supervised learning algorithm that excels in the

classification of data, however, it has been known to be quite computationally intensive. Last but not least, Decision Trees or decision tree learning is a method that uses decision trees as a predictive model. Decision Trees are known to be very computationally efficient when the set of classes that the input can be classified into (labels) is small. However, Decision Trees tend to inherently over-fit the model to the training data and can lead to unnecessarily computationally intensive models when the number of labels is large.

## 2.4 Feature Selection

Feature selection is the process of defining a set of relevant features that are used for constructing a classification model. Essentially, the set of features are selected to maximize the classifier's ability to distinguish between different classes of data. The choice of feature selection techniques has a direct and significant influence on classification accuracy. In general, one aims to select a minimalistic set of features that perfectly distinguishes all classes.

The field of feature selection has been extensively studied over the past couple of decades. Consequently, an enormous set of open-source feature selection techniques exist. The most prevalent techniques available today are inspired by the scale-invariant feature transform or SIFT descriptor. A descriptor is fundamentally a feature detection technique, where the descriptor first attempts to detect point of interest in an image and then calculates a set of features to describe each interest point. SURF or Speeded Up Robust Features is the most widely adopted descriptor in the context of object recognition or classification. SURF is inspired by SIFT and claims to achieve better robustness against different image transformation while also running several times faster [24]. Fortunately, OpenCV supports a wide range of descriptors including SURF, SIFT, FREAK, BRIEF, BRISK, ORB, OPPONENT_FREAK, OPPONENT_BRIEF, OPPONET_BRISK and OPPONENT_ORB. Unfortunately, SURF and SIFT descriptors are patented and require a license for use in OpenCV. Figure 2.2 below illustrates the average run-time of all available OpenCV descriptors.

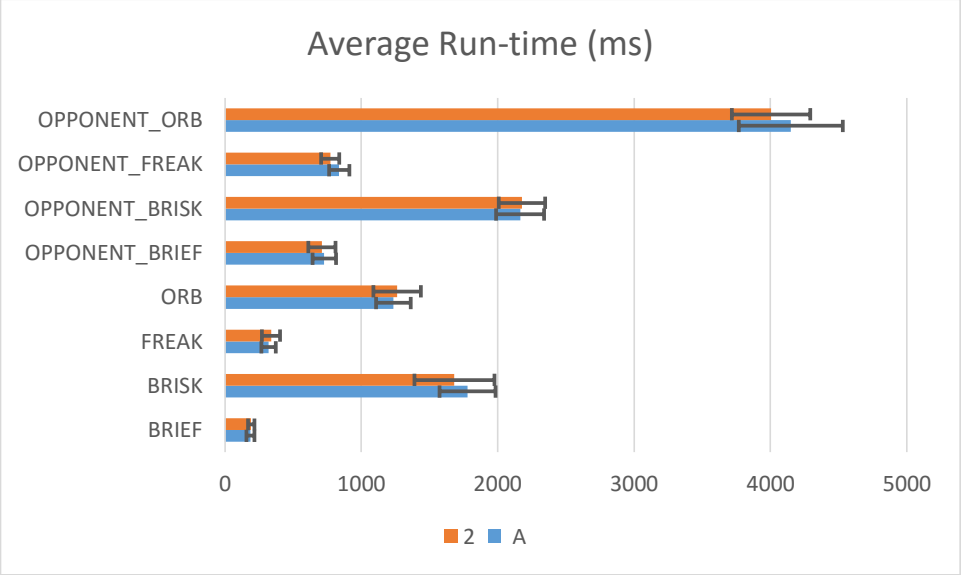



*Figure 2.2: OpenCV Descriptors: Average Run-time in milliseconds (blue for A.jpeg results and red for 2.jpeg)*

The performance results published above are the outcome of a simple OpenCV experiment that surveyed all supported descriptors. The descriptors were tested on two different 512x512 binary images, 2.jpeg and A.jpeg portrayed in figures 2.3 and 2.4 below. Each test was ran 5 times on a Galaxy S4 (I9505). The complete experiment and its results are available in the attached files, refer to Appendix A for details.


*Figure 2.3: 2.jpeg*

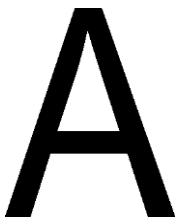



*Figure 2.4: A.jpeg*

On the other hand, feature extraction can be perceived as an alternative approach to feature selection. Feature extraction involves transforming high-dimensional data to a lower dimension. Simply put, feature selection attempts to find a subset of the original features while feature extraction attempts to reduce the dimensionality of the original features. The most common dimensionality reduction technique is referred to as Principal Component Analysis or PCA. PCA is a linear dimensionality reduction technique that performs a mapping of the data to a lower-dimensional space by maximizing the variance of the data in the low-dimensional representation [25]. Various other transformation techniques have been heavily employed in feature extraction, including Discrete Cosine Transform (DCT), Fast Fourier Transform (FFT) and

Discrete Fourier Transform (DFT). These transformations essentially convert data into elementary frequency components [26]. Discrete transformations usually complement feature extraction and provide suitable feature descriptors for a variety of data sets, including object and character recognition. These transformations can also be utilized to create language independent character recognition systems [27]. Figure 2.5 below describes the basic operation of discrete transformations. On the other hand, Figures 2.6 and 2.7 below illustrate the run-time requirements of the aforementioned discrete transformations and PCA on android, respectively.

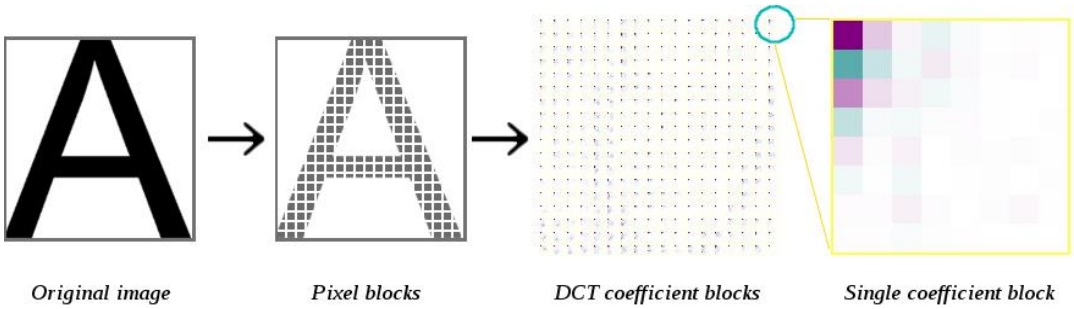



*Figure 2.5: Overview of DCT Operation*

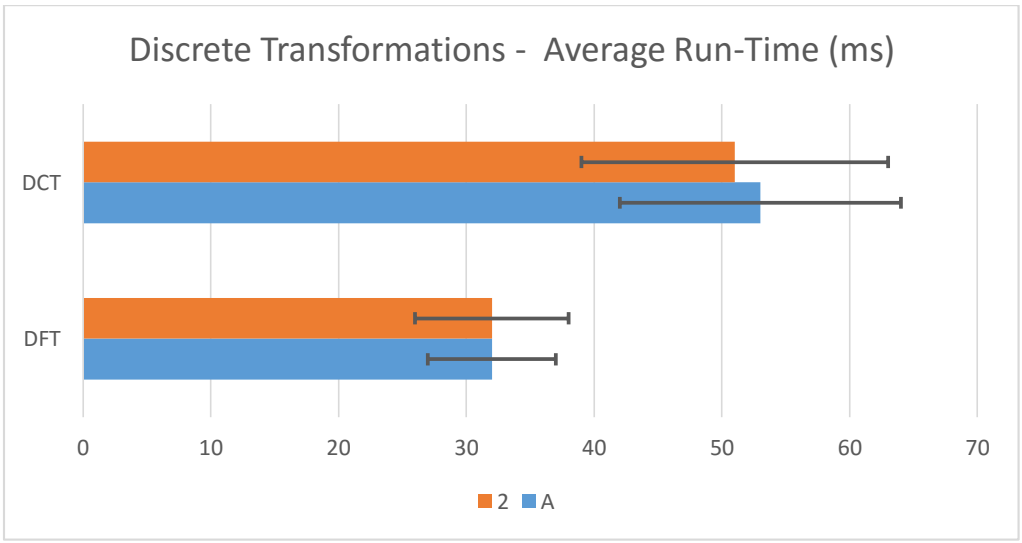



*Figure 2.6: Discrete Transformations: Android Average Run-time in milliseconds (blue for A.jpeg results and red for 2.jpeg)*

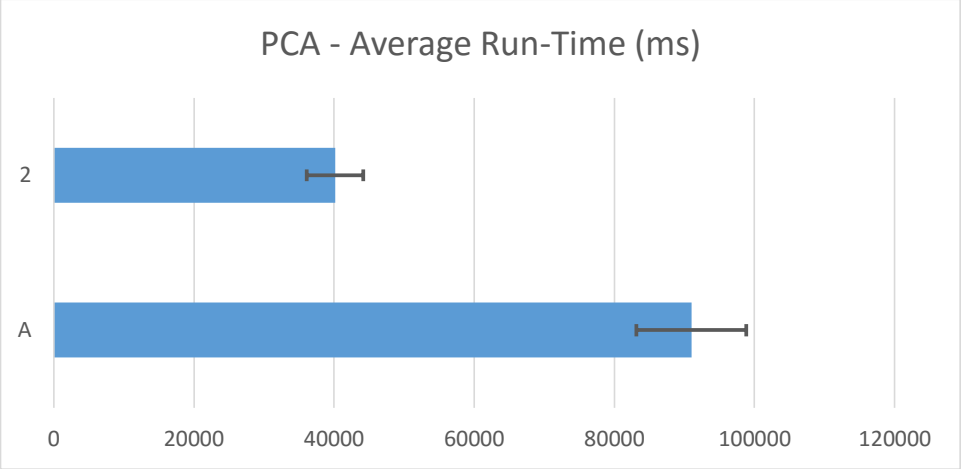



*Figure 2.7: Principal Component Analysis: Android Average Run-time in milliseconds*

The results above describe the average run-time requirements of each algorithm on the same images illustrated in figures 2.3 and 2.4, which was implemented on android and tested on the same Galaxy S4. Although discrete transformations and PCA can facilitate the feature selection/extraction process, they are infeasible for a quasi-real-time implementation. Nevertheless, the most accurate character recognition systems available today employ discriminating features that describe patterns prevalent in the specific language that the system is aimed at. Numerous studies inspect feature selection or extraction based on geometric features dominant in a certain language [28] [29], facilitating more accurate OCR systems. In comparison, the final feature selection component implemented in this application requires 0.9 millisecond per character; that is ~205x faster than the fastest OpenCV descriptor (BRIEF), ~35x faster than DFT and ~55x faster than DCT.

## 2.5 Connected Component Analysis

Connected component analysis is an algorithm inspired by graph theory, where an image is analyzed and its pixels are grouped into components based on pixel connectivity [30]. Pixel connectivity is defined by neighboring pixels that share similar intensity values. The neighboring pixels can either include all 8 pixels surrounding a given pixel in 2D (8-connecetivity) or just 4 pixels (4-connectivity) as illustrated in figure 2.8. Connected component analysis is commonly known as connected component labeling or blob extraction and is usually used in computer vision

applications to detect connected regions in binary digital images. Various implementations of connected component analysis exist, each implementation attempts to maximize efficiency from a different perspective. One-pass, two-pass and multi-pass algorithms were developed mainly due to significant attempts in parallelizing the algorithm; a pass is one iteration over all pixels in the image. The two-pass algorithm is by far the most prevalent implementation available, which is a consequence of the algorithm's simplicity and efficiency. As implied by the name, the two-pass algorithm makes two passes over the image: the first pass to assign temporary pixel labels (recording similarity or connectivity) and the second pass to replace each temporary label by the smallest label of its equivalence class [31]. The following figure 2.9 graphically illustrates the operation of the two-pass algorithm using 4-connectivity and 8-connectivity.

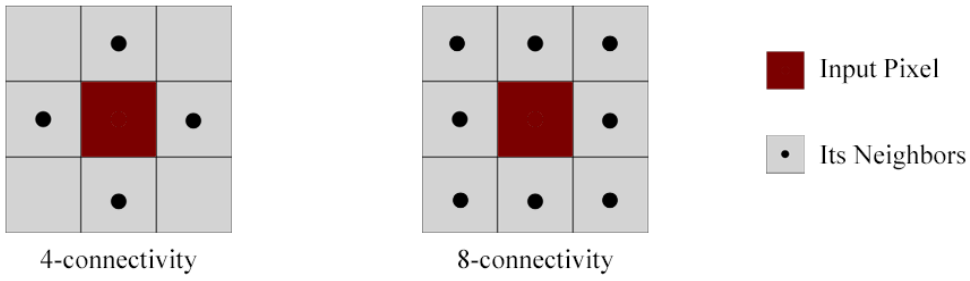



*Figure 2.8: 4-connectivity versus 8-connectivity*

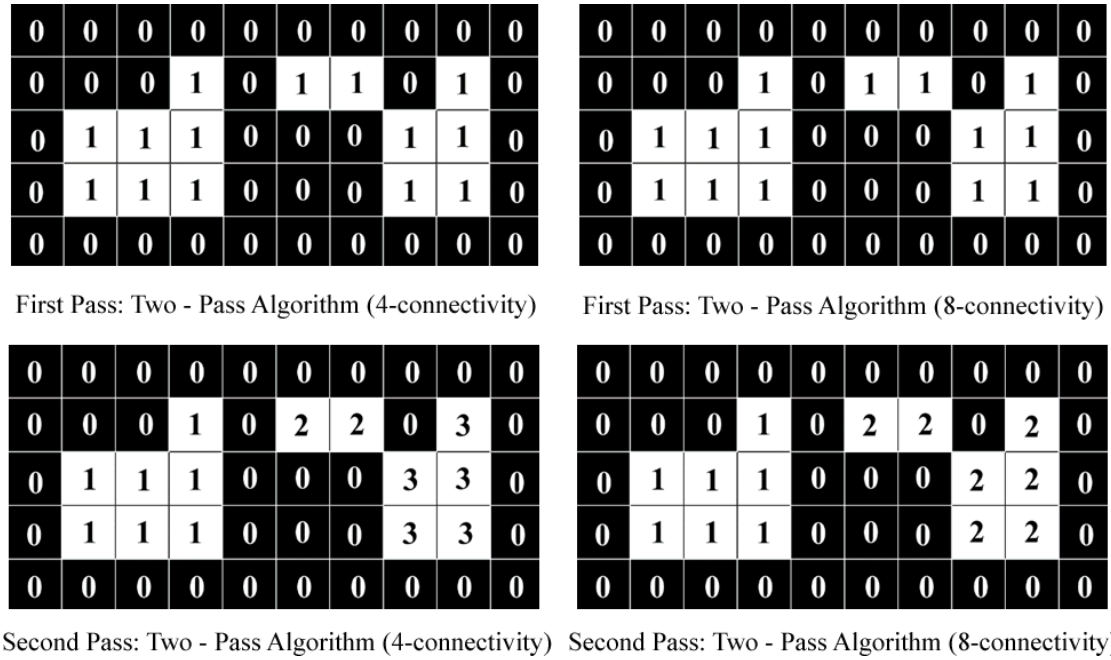



*Figure 2.9: Two-pass Algorithm (4-connectvity versus 8-connectivity)*

# 3 Design

This project fundamentally revolves around developing an Android application that is capable of digitizing the content of digital displays, specifically displays commonly found in household appliances. The application will distinguish itself from all related approaches / competition by primarily focusing on developing Segmentation and Classification algorithms that inherently and efficiently address the complexities of digital displays. Moreover, the application aims to incorporate sound recognition alongside speech synthesis to provide an auditory interface that facilitates the operation for the targeted users. Consequently, the design components behind this project are principally split into three main categories: Segmentation, Classification and auditory Interface.

## 3.1 Segmentation

Precise segmentation is undeniably the most crucial component required to achieve accurate recognition of digital displays. In essence, segmentation can be perceived as the process of figuring out which sets of pixels present in an image are likely to represent characters. To begin with, we must define the characteristics of characters commonly found in our targeted environment. In the context of household appliance displays, characters generally belong to the English alphanumeric set and occasionally symbolic characters are featured in specific types of appliances; i.e. ovens will usually feature temperature related symbols while washing machines might feature fabric related symbols. Characters present on a household appliance display, whether alphanumeric or symbolic, usually have very consistent spatial/geometrical characteristics and appear in one high-contrast color. In this context, a word is a set of neighboring characters that together describe one piece of information as illustrated in the figure 3.1.


*Figure 3.1: word 1 describes time while word 2 describes the temperature*

Characters present in the same word are perfectly aligned next to each other where the upper and lower bound also match perfectly. Moreover, the orientation is also consistent between characters present in the same word. In the case of multi-colored display, characters present in the same word appear in one consistent color. More to the point, the seven-segment character encoding is very prevalent in household appliance displays and it would definitely benefit the segmentation process to exploit the characteristics of the seven-segment encoding. Throughout the following section, various segmentation algorithms are proposed; each algorithm aims to uniquely exploit the aforementioned common characteristics of characters present in our targeted environment. The algorithms are evaluated in terms of segmentation accuracy and run-time efficiency. The evaluation is performed using a set of 36 labeled images, taken personally with a 13 Megapixel Camera and scaled down to 1280x720 pixels (~921 Kilo pixel). The evaluation image set is illustrated in Appendix B.

### 3.1.1 Histograms

Histograms in essence are representations of colors present in an image. Generally, Histograms are simple to calculate and are independent of the orientation of image content. However, Histograms inherently have numerous limitations for the task of segmentation for character recognition. To begin with, Histograms with a large number of color bins entail very long run-times; due to the large amount of color comparison computations. Moreover, Histograms inherently only tackle color information and do not consider any spatial proximity information. This leads to segmentation failure, in cases of monochrome displays, as the characters are segmented with the background

noise that could have been easily ignored using spatial proximity information. To our purposes, the algorithm is split into three main components: the iterative procedure, the chosen color model and the calculation of an appropriate color difference tolerance. To begin with, we propose a very common and efficient Histogram algorithm [32] that is naturally independent of the color model and the color difference tolerance. Pseudocode for the algorithm is available in Appendix D.

Fundamentally, the chosen color model and the color difference tolerance have the most significant effect on the accuracy of the segmentation and the run-time. In essence, the Histogram is attempting to separate sets of pixels in an image based on visual resemblance and visual resemblance naturally stems from the human brain's ability to perceive light. An enormous amount of research went into describing color schemes that can model human vision and perception. The most commonly used color schemes are RBG, HSV and CMYK. To begin with, RGB is an additive color model that essentially mixes varying intensities of Red, Green and Blue to model many colors. The RGB model is the most common today, as it attempts to simulate the three types of color receptors found in the human eyes. HSV or Hue-Saturation-Value is the most common cylindrical-coordinate representation of colors. The HSV color model is well known because it allows the differentiation of colors based on brightness (perceived luminance). Finally, CMYK is a subtractive color model where the intensities of Cyan, Magenta, Yellow and Key (Black) are used to describe a color. These three color schemes have been widely used in the context of computer vision and OCR. All three provide a unique modeling perception of human vision; consequently, each model enables a different approach to calculating color difference. The use of multiple color models concurrently is known to produce a more accurate presentation of visual resemblance. However, the conversion between different color models can be computationally intensive. More to the point, the use of different types of color models concurrently is often used to allow the segmentation process to cope with glare, light reflections and all the complexities that digital displays give rise to. Fundamentally, the use of perceived luminance and saturation from the HSV model in conjunction with the basic RGB model has been known to enable color difference calculations that ignore specular reflections [33]. Given that the majority of digital displays in household

appliances feature flat-surface glass panels, we can safely assume that the majority of light reflections are specular.

The use of Gaussian filters is another common approach in handling the effects of light reflections. Gaussian filters perform a weighted average of surrounding pixels based on a Gaussian distribution and are used to remove Gaussian noise [34]. Fortunately, light reflections always tend to appear as a Gaussian distribution; however, the use of Gaussian filters is infeasible for a realistic quasi-real-time implementation.

One histogram implementation that stands out is the "Just Noticeable Difference' or JND histogram [32]. The JND histogram attempts to offer a color difference calculation and a specified tolerance such that each color bin is visually dissimilar. The JND histogram defines visual similarity based on biological information regarding the operation of human vision. The whole model relies on the fact that the human retina features three types of color cones sensitive to Red, Green and Blue; where Green cones are the most sensitive, blue cones are moderately sensitive and red cones are the least sensitive. Consequently, the JND models the RGB color scheme into 24x28x26 bins for R, G, and B respectively. Moreover, color difference is calculated using the standard Euclidean distance measurement; as illustrated below.

$$\Delta r^2 + \Delta g^2 + \Delta b^2 = d^2$$

*Equation 3: Euclidean Distance*

Where d corresponds to the color difference, Δr is the red color difference, Δg is the green color difference and Δb is the blue color difference. On the other hand, City Block Distance is an alternative metric that calculates the distance between two points in the xy-plane, as the distance in x plus the distance in y (similar to the way one moves in a city). The use of City Block Distance for calculating color difference is illustrated below:

$$\Delta r + \Delta g + \Delta b = d$$

*Equation 4: City Block Distance*

The use of the City Block Distance metric will enable a significant speed-up but will lead to less accurate results.

### 3.1.2 Histogram and Windowing

The use of windowing on histograms enables the exploitation of some spatial information without rendering the approach computationally infeasible. Basically, instead of segmenting pixels in an image based solely on color; this approach segments pixels for each window in the given image based solely on color. The set of windows can be based on a random set of pre-defined rectangles that feature height/width ratios, orientations and positions that are likely to separate words or sets of characters from the background. The approach in no way guarantees accurate results; however, it is a very computationally efficient approach into exploiting spatial proximity information and tends to yield accurate results in most cases. The accuracy of the approach relies on the consistency of different digital displays that incline to display information in the middle, with a certain size (usually not smaller then display width * 0.05 and not bigger than display width *0.5), and a height-to-width ratio between 0.2 – 1. Unfortunately, this windowing approach requires accurate camera positioning and it is very likely that a small portion of household appliances will not fit the windowing description.

In order to remove the accurate camera positioning requirement, we must investigate a dynamic windowing approach that calculates a set of windows depending on the features of the image itself. This approach requires more computations; consequently it is crucial to implement the dynamic calculations efficiently. We experimented with one approach, where the set of windows are calculated based on the relative change in the density of pixels horizontally and vertically. This approach assumes that when a horizontal or vertical line encounters a character, it will perceive a significant relative change in pixel intensity. The pixel intensity calculations are relatively expensive; optimistically, not as expensive as exploiting all spatial proximity information i.e. blob detector. Both windowing approaches were evaluated using the most efficient histogram implementation.

### 3.1.3 Blob Detectors

Blob Detectors have been extensively studied throughout the past decade [35] and a wide variety of unique implementations are available. Blob detectors aim to exploit both color difference information and spatial/geometrical information. Nevertheless, the development of a blob detection approach relies heavily on the characteristics of the imaging environment and even though an enormous set of implementations are already

available; the vast majority of them focus on accuracy and consequently require infeasible computational requirements. Therefore, two different blob detection approaches were developed with the main focus on computational efficiency. To begin with, given the current evolution of multi-core processors, specifically for ARM architectures; ARM currently holds the undisputed lead in smartphone processors, a multi-threaded approach seemed appropriate. The first approach is primarily a random blob detector; the algorithm randomly picks pixels and for each pixel it initiates 8 threads where each thread explores a specified region around the original pixel. The algorithm retains a Boolean array that points out pixels that belong to a segment; such that each pixel can only belong to one segment. The algorithm essentially splits the area around a specified pixel into 8 fields; where each thread iterates through one field and adds pixels of similar color to the current segment as illustrated in figure 3.2 below.

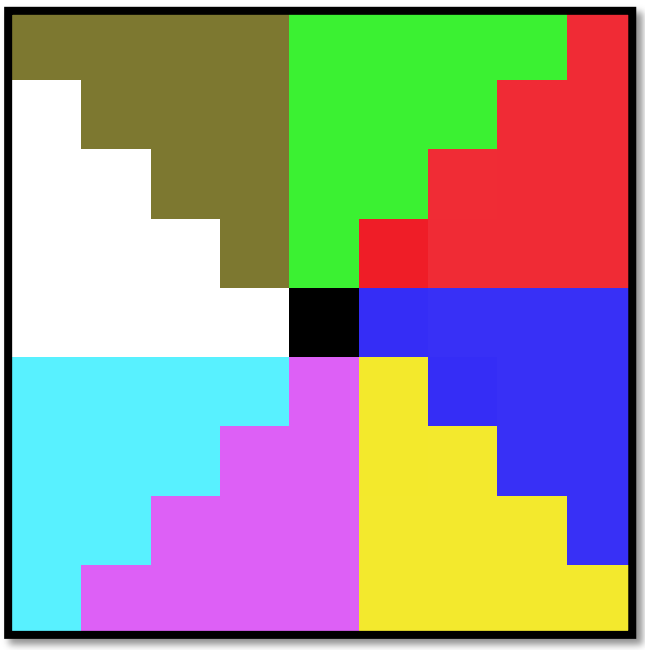


*Figure 3.2: Multi-threaded Blob Detector Illustration*

The black pixel in the middle is the randomly selected pixel and each color represents the area that each thread will explore. Each thread works its way from the center to the image borders and a thread terminates when it encounters an entire line (horizontal or vertical depending on the area orientation) with no similar pixels. The second blob detection approach discards the multi-threaded approach and implements a recursive nearest neighbor technique similar to KNNs. This approach relies on an iterative

selection of pixels, and for each selected pixel the algorithm considers all 8 surrounding pixels and for any of these pixels that match the original, does the same recursively. The algorithm is strictly serial; nevertheless, the recursive nature of the approach enables the operating system to exploit inherent parallelism. The parallelism stems from the fact that each calculation for a given pixel is independent of the result of the calculation for all other pixels. This approach offers a more coarse grained multi-threaded approach; such that ratio of thread work time to thread initiation time increases. Nevertheless, this approach suffers from redundant calculations as the calculations are not made aware whether a pixel belongs a specific segment; such that a pixel can belong to more than one segment. The exploitation of a Boolean array that represents pixels that belong to a specific segment would overcome the redundancy challenge but would render the approach unparallelizable; as the memory constraint limits the operating system's ability to exploit inherent parallelism.

### 3.1.4 Edge Detector and RGB histogram

As previously stated, edge detection is a mathematical approach that aims to identify points on the edge of significant objects in an image. The Canny edge detector is a popular edge detection algorithm. Its popularity is a consequence of the algorithm's performance, the Canny edge detector is known to produce reasonably accurate results with very little run-time requirements. For example, the OpenCV Canny edge detector requires 80 milliseconds to process a 512x512 image on a Galaxy S4. Consequently, we propose a final segmentation approach that would exploit both color difference and spatial information. The color difference information is obtained by using an RGB histogram and spatial information is obtained by using the Canny edge detector. Essentially, this approach performs a connected component analysis on the set of edges detected and splits each connected component. Then the algorithm exploits the RGB histogram information to calculate the background color and foreground color for each connected component. Finally, the algorithm splits the content of the display into words

by identifying characters with similar background and foreground colors. This algorithm distinguishes itself from all others by efficiently exploiting spatial information. The efficiency stems from the inherent speed of the Canny edge detector, as it provides a relatively fast approach to address spatial information.

### 3.1.5 Geometric Matching and Splitting

An independent component of segmentation, is the geometric matching and splitting of words. This component primarily aims to exploit the consistent geometrical characteristics of our targeted environment; excluding color and spatial proximity information. As mentioned before, the general characteristics involve the geometrical aligning of the upper and lower bounds of characters in the same word, the consistent orientation of all characters (not only ones in the same word) and the standards of the seven-segment encoding scheme. This component is independent of the color and spatial-based segmentation component and it fundamentally aims to complement it. Essentially, after the first segmentation component completes its operation; this component iterates through all the calculated segments and attempts to geometrically match segments together based on the proximity of the upper and lower bounds. If a segment matches the bounds of another, the algorithms joins both segments into one and attempts to split the new segment into its basic characters. The splitting is based on spatial proximity and a seven-segment density scheme that assesses the probability of it being a seven-segment encoded display according to the density of pixels in each of the seven-segments (hypothetical seven-segments). More to the point, if two segments are matched and are then split perfectly into multiple characters; only then are they considered to be potential meaningful characters. This approach enables feeding entire words into the classification component, consequently facilitating the process of deriving meaning out of characters. This component is applied after the initial segmentation component (one that exploits color and spatial properties) and is included into the evaluation of all segmentation approaches.

### 3.1.6 Evaluation

| Approach | Mean Run-time | S.D Run-time | Mean Accuracy | S.D Accuracy |
|---|---|---|---|---|
| RGB Histogram | 1604 | 219 | 99.7 | 1.1 |
| HSV Histogram | 2637 | 457 | 99.8 | 1.1 |
| CMYK Histogram | 2183 | 311 | 95.9 | 17.5 |
| HSV-RGB Histogram | 3150 | 713 | 99.7 | 1.1 |
| JND Histogram | 1810 | 266 | 99.7 | 1.1 |
| Static Windowing JND Histogram | 2020 | 585 | 99.7 | 1.1 |
| Dynamic Windowing JND Histogram | 2353 | 462 | 99.7 | 1.1 |
| 8-threaded Blob Detector | 5158 | 1423 | 99.8 | 1.1 |
| Nearest Neighbor Blob Detector | 1462 | 122 | 99.8 | 1.1 |
| Canny Edge RGB Histogram | 743 | 41 | 98.3 | 1.8 |

*Table 3.1: Segmentation Evaluation Results – Euclidean*

Although all the developed segmentation algorithms were designed with consideration to run-time requirements, the main objective was achieving higher accuracy rates. The initial development process insured that any design decision that created a trade-off between improved accuracy and improved run-time would favour improved accuracy. More to the point, all the aforementioned segmentation algorithms must utilize some kind of colour distance metric. On one hand, exploiting the Euclidean distance metric would result in improved accuracy results on the expense of run-time. While on the other hand, exploiting the city block distance metric would result in improved run-time requirements on the expense of accuracy. Initially favouring accuracy, all Evaluation results illustrated in table 3.1 describe the performance of the algorithms when exploiting the Euclidean distance metric. Nevertheless, the following table 3.2 describes the performance of the algorithms when exploiting the City Block distance metric and

describes the influence of this run-time optimization decision on every algorithm based on the following two features:

$$\Delta\mathbf{A} = \frac{new\_accuracy}{upper\_bound\_accuracy} \qquad \Delta\mathbf{T} = \frac{upper\_bound\_time - new\_time}{upper\_bound\_time}$$

*Equation 5: Accuracy Change (%)* *Equation 6: Run-time Change (%)*

| Approach | Mean Run-time | S.D Run-time | Mean Accuracy | S.D Accuracy | **ΔA** | **ΔT** |
|---|---|---|---|---|---|---|
| RGB Histogram | 1355 | 164 | 96.2 | 1.0 | 0.96 | 0.16 |
| HSV Histogram | 2025 | 380 | 97.1 | 1.0 | 0.97 | 0.23 |
| CMYK Histogram | 1741 | 275 | 92.4 | 13.5 | 0.96 | 0.20 |
| HSV-RGB Histogram | 2635 | 433 | 96.2 | 1.0 | 0.96 | 0.16 |
| JND Histogram | 1443 | 197 | 96.2 | 1.0 | 0.96 | 0.20 |
| Static Windowing JND Histogram | 1503 | 309 | 96.2 | 1.0 | 0.96 | 0.26 |
| Dynamic Windowing JND Histogram | 1741 | 216 | 96.2 | 1.0 | 0.96 | 0.26 |
| 8-threaded Blob Detector | 4163 | 1117 | 97.1 | 1.0 | 0.97 | 0.19 |
| Nearest Neighbor Blob Detector | 1236 | 103 | 97.1 | 1.0 | 0.97 | 0.15 |
| Canny Edge RGB Histogram | 631 | 45 | 95.8 | 3.2 | 0.97 | 0.15 |

*Table 3.2: Segmentation Evaluation Results – City Block*

The evaluation results shown in table 3.1 and table 3.2 above are a result of 20 runs for each segmentation approach; the majority of these implementations are stochastic. Stochastic algorithms exploit random probability distributions and usually yield slightly different results. The results above describe the mean accuracy and mean run-time over 20 runs; where each run runs a specific segmentation algorithm on the 36 images available in the evaluation set (refer to Appendix B). All images were captured using 1280x720 resolution; however, the segmentation is performed on 256x144 resized images. These evaluation results are merely meant to highlight the feasibility of using a given segmentation algorithm and illustrate the reliability and consistency of each

algorithm. All evaluations were implemented on Android Studio and ran on Galaxy S4 I9505. The complete experiment and its results are available in the attached files, refer to Appendix A for more details.

## 3.2 Classification

Accurate classification is another crucial component in the context of this project. The accuracy of the classification component on a given segment of pixels is inherently dependent on the segmentation process. The precision of the segmentation component has a significant effect on the accuracy of classification and consequently has the most influence on the operation of the application as a whole. Nevertheless, given the run-time and accuracy limitations posed by the various possible segmentation methods; it is necessary that the classification component operates with the best possible accuracy and attempt to incorporate mechanisms that reduce the dependency on segmentation precision.

### 3.2.1 Classification Model

Classification, in the context of machine learning, is the challenge of identifying to which of a set of classes a new observation belongs. Classification is an example of supervised learning; where the learning algorithm exploits a training data-set of correctly labeled observations as illustrated in figure 3.3.

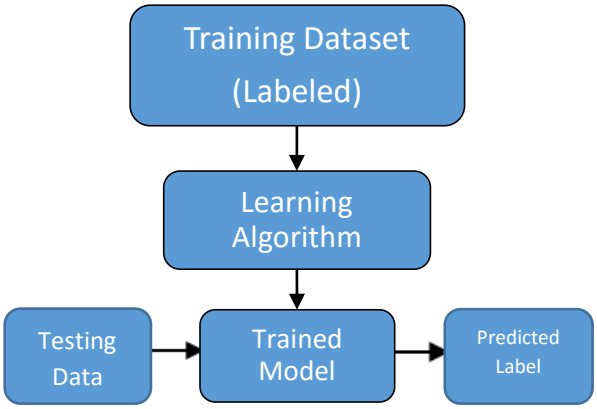



*Figure 3.3: Standard Classification Model*

To begin with, the classification component requires a labeled training dataset that contains all possible classes that characters in our environment might belong to. A simple approach would exploit manually produced labeled images of the English alphanumeric set including the seven-segment encoding scheme and a pre-defined list

of fonts commonly used in digital displays. More to the point, the exploitation of the aforementioned classification model enables the preliminary training of a fully functioning classification model on a dedicated server and simply including the static trained model to the application; this approach is favorable given the limited capabilities of current smartphone platforms. Nevertheless, a static classification model would not be able to incorporate new classes into the model and will not be able to dynamically exploit testing data to update the classification model. In the context of our environment, there is no complete set of possible classes that characters in our environment can belong to; while the English alphanumeric set covers a huge portion of the characters the application will encounter, the set of symbols that will be encountered is endless. Therefore, a more dynamic model that aims to unify all the benefits of a static model and enables the dynamic updating (online-learning) of the model itself is proposed. The concept is simple, the proposed model is essentially a static model that is initially trained using a set of images that describe the alphanumeric set over different fonts and images that describe a limited set of common symbols. As the client application encounters new testing data it will report to an online server that aims to update the classification model daily using new data encountered from various clients. This client-server approach also enables the use of common image search engines, especially ones that specialize in symbols, to create a constantly and dynamically updated classification model. The only downside is the client's requirement to retrieve the updated model as frequently as possible.

### 3.2.2 Classification Ensemble

In an effort to push the classification accuracy to the limits a classification ensemble is proposed. A classification ensemble, illustrated in figure 3.4 below, involves training multiple classification models such that it predicts data labels by issuing a majority vote between all individual classification models. An ensemble of classifiers aims to overcome the mistakes that individual models can make by aggregating multiple classifiers. The diversity of classification models in an ensemble is encouraged to a certain degree; there exists a trade-off where we don't want all models to be identical but we don't want them to be different just for the sake of it (possibly sacrificing unique

individual performance). Moreover, the concept of feeding each classification model in an ensemble different images (of the same environment) usually yields more accurate results; as it decreases the chance that two classification models make the same mistake and mistakes made by only one classifier are more likely to be overruled by majority voting. Last but not least, classification ensembles require little to none parameter tuning and tend to work exceptionally well "off-the-shelf" [36]. Unfortunately, classification ensembles entail increased storage capacity and increased computations.

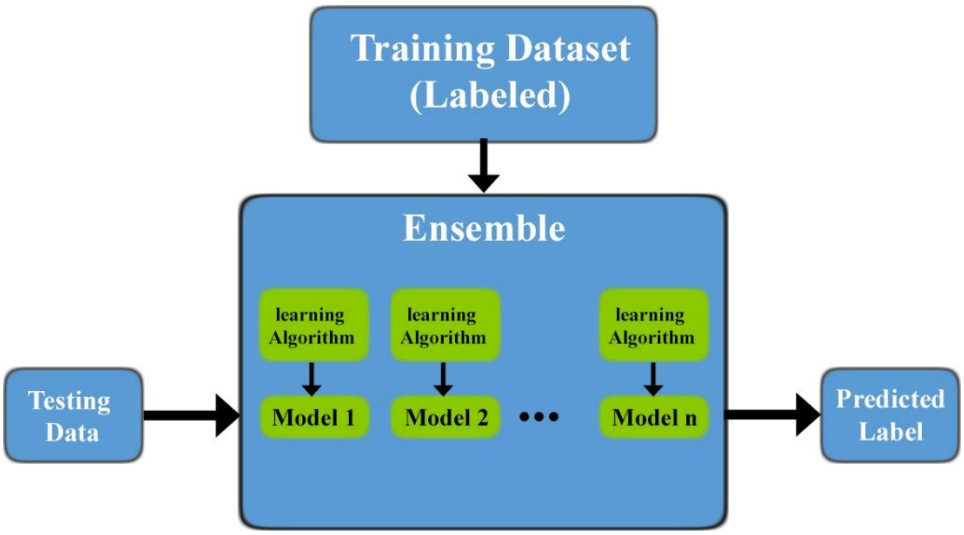



*Figure 3.4: Classification Ensemble Model*

A classifier or classification technique is an approach to building classification models from an input dataset. The most common examples include decision trees, neural networks, support vector machines and naïve Bayes classifiers. Each one of these techniques employs unique learning algorithms to identify a classification model that fits the relationship between the feature training dataset and the class labels. In the context of this application, the aim is to develop multiple unique classifiers that achieve the best accuracy while executing within an acceptable time limitation.

To being with, Support Vector Machines are considered to be one of the most powerful classification techniques available today. The success of SVMs is three-fold. Firstly SVMs employ a regularization parameter which emphasizes the issues of over-fitting and facilitates avoiding them. Secondly SVMs exploit the kernel trick to efficiently perform non-linear classification and to enable incorporating problem-specific expert

knowledge by engineering the kernel itself. Finally SVMs are defined by convex optimization problems, in which any local minimum must be a global minimum, for which many efficient algorithms exists i.e. SMO [37]. More to the point, SVMs' only disadvantage lies in the determination of parameters: including parameters for regularization, the choice of kernel and the kernel's parameters. Moreover, SVMs are inherently two-class classifiers. Two unique strategies that tackle multi-class SVM classification exist. The most common of them is "one-versus-all" or OvA, which involves training a single (two-class) classifier per class where training samples of that class are positive samples (represent one class) and all other training samples are negative (represent the other class). OVA classifiers make decisions by applying all classifiers to the new test sample, predicting the label which corresponds to the classifier with the highest confidence score. The other approach is "one-versus-one" or OvO, which involves training $\frac{K(K-1)}{2}$ binary classifiers for a k-class problem where each classifier can distinguish between two different classes. The only apparent difference between these two approaches is that the OvA strategy requires base classifiers that produce confidence scores rather than just a class label. Although a linear two class SVM model is relatively trivial to implement, a full blown SVM implementation that supports multiple kernels and multi-class classification is beyond the scope of this project. Fortunately, LIBSVM is an open-source SVM library that implements the sequential minimal optimization algorithm for Kernalized SVMs, supporting OvO multi-class classification, cross validation, probability estimates, various kernels and weighted SVMs. Furthermore, LIBSVM already provides an open-source java wrapper for its library.

As previously mentioned, K-Nearest Neighbor, commonly referred to as KNN, is a non-parametric method used generally for classification and is known to be competitively accurate and quite computationally intensive. Consequently, we propose a minimalistic native Java KNN implementation. The procedure of KNN classification is simple, given a dataset of training features and the features of a new testing data, the algorithm calculates the distance between each training feature and the new test features and then

returns the most common label of the k-th nearest neighbors (in terms of distance between features). Pseudocode in figure 3.5 below illustrates the basic KNN procedure.

```
KNN(training_model, test_features, k)
{
    distances = array[training_model.length]
    for(i = 0; i < training_model.length; i++)
        distances [i] = calculate_distance(training_model[i], test_features)
    sort(distances)
    labels = emptyList()
    for(i = 0; i < k; i++)
        labels.add(labelOf(distances[i]))
    return mostCommon(labels)
}
```

*Figure 3.5: KNN Pseudocode*

As evident in the code above, the KNN algorithm is very minimalistic and straight to the point, with no room for modification. It is also clear that the majority of the calculations or run-time is spent on calculating the distances between features. A simple run-time analysis revealed that around 99 percent of the run-time is spent on distance calculations (using Euclidean Distance). Therefore in an effort to reduce the computational requirements, we propose another experiment that compares KNN's performance when using Euclidean Distance versus City Block Distance. Figures 3.6, 3.7, and 3.8 below illustrate the results. Both experiments and their respective results are available in the attached files, refer to Appendix A for more details

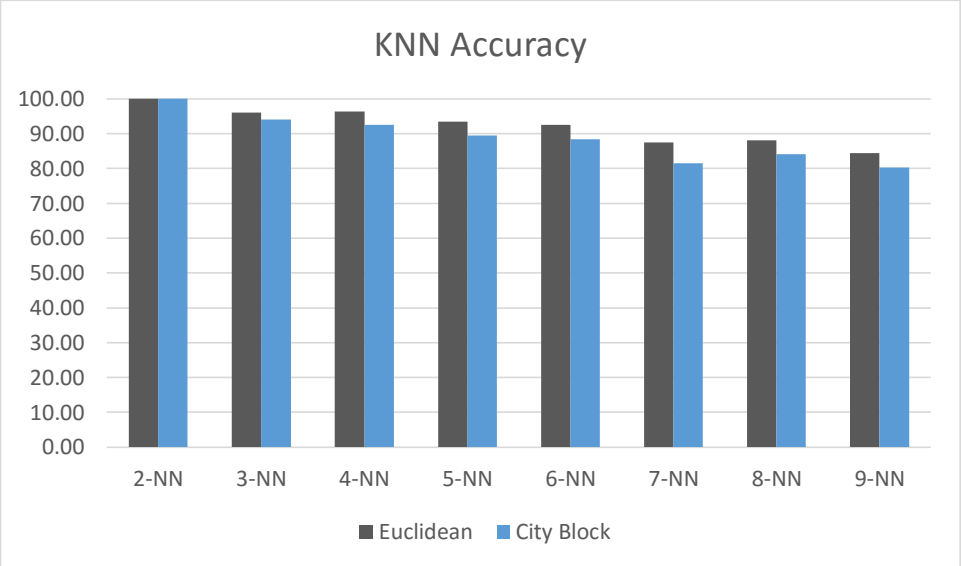



*Figure 3.6: KNN Accuracy: Euclidean Vs City Block*
*(blue for city block results and black for Euclidean)*

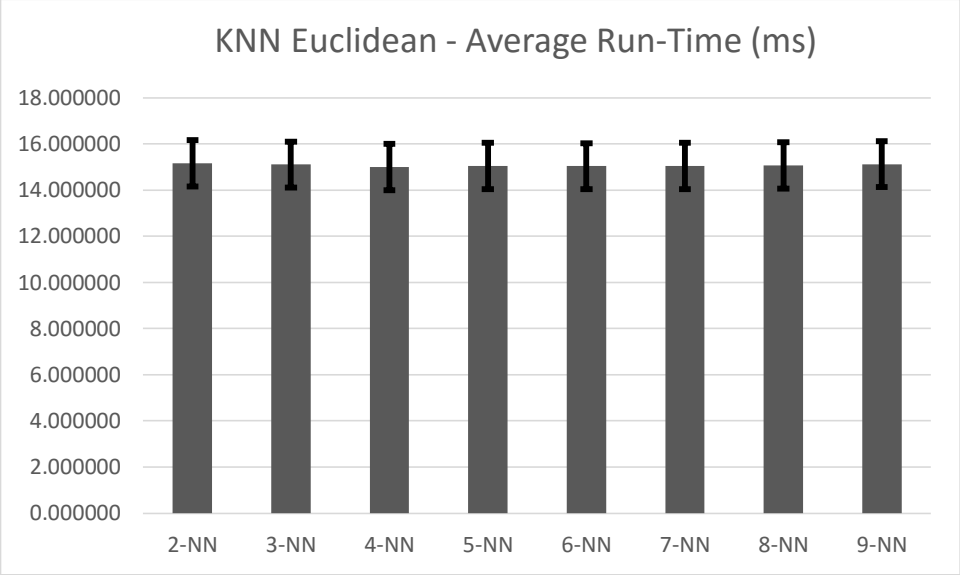



*Figure 3.7: KNN Average Run-time: Euclidean*

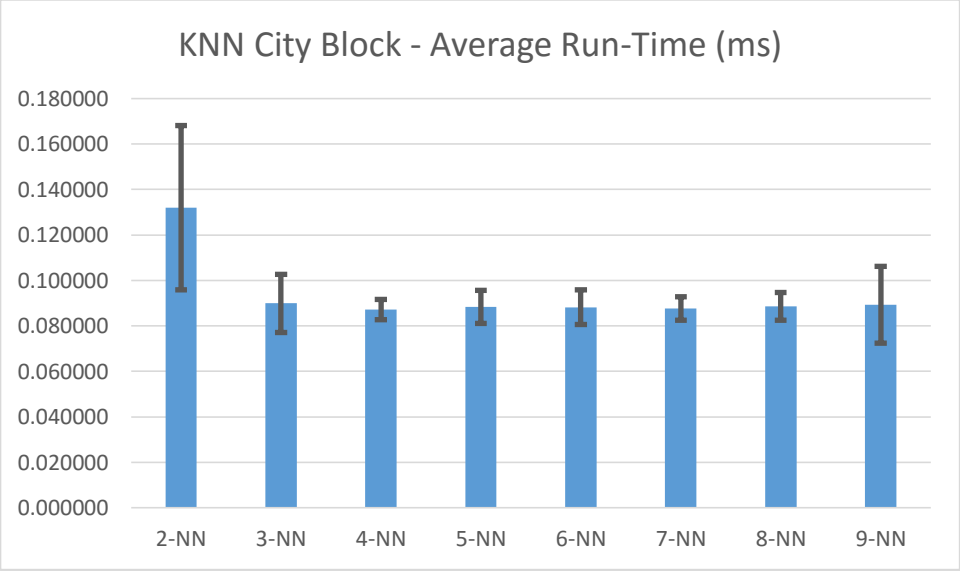



*Figure 3.8: KNN Average Run-time: City Block*

As expected, the evaluation results prove that the City Block metric is able to achieve faster execution times (~100x) and suffers from slightly reduced accuracy (~1-3%). These experiments were implemented in native Java and executed on a pc, rather than a smartphone. This experiment is merely meant to demonstrate the influence the distance metric poses on the algorithm's performance rather than the actual performance of this algorithm (clearly it executes slower on a smartphone platform).

Finally, we propose a custom classifier that aims to identify Seven-Segment characters. As implied by the name, Seven-Segment displays feature seven elements that can be lit in different combinations to represent different characters. Seven-Segment displays are nearly uniform, usually the seven segments are elongated hexagons organised in an

oblique or slanted arrangement. Unfortunately, some variations do exist. The numbers 6, 7 and 9 may be represented in two different approaches, with or without a 'tail', as illustrated in figure 3.9 below. Moreover, in some cases the segments are not oblique and are represented by trapezoids and rectangles instead of hexagons. Sixteen-segment displays which are capable of encoding the entire alphanumeric set do exist, even when combined with fourteen-segment displays, they are still significantly less prevalent than seven-segment displays and have been mainly replaced with a dox matrix alternative. These display formats are described in figure 3.11. More to the point, given that Seven-Segment displays feature seven elements where each element can possess one of two states, we can calculate the number of all possible states as $2^7$ which equals 128 states. All 128 possible states of the Seven-Segment encoding are illustrated in Figure 3.10 below.

*Figure 3.9: Seven-Segment with or without a 'tail'*

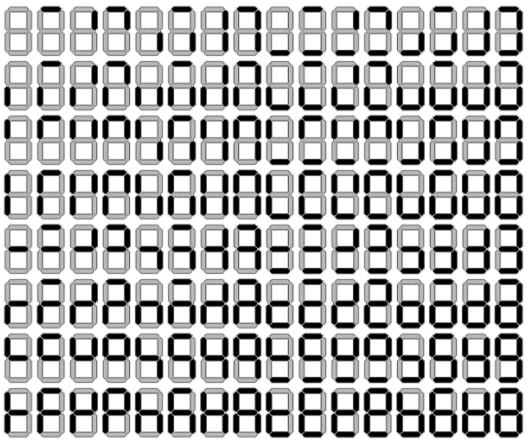


*Figure 3.10: All 128 states of the Seven-Segment format*

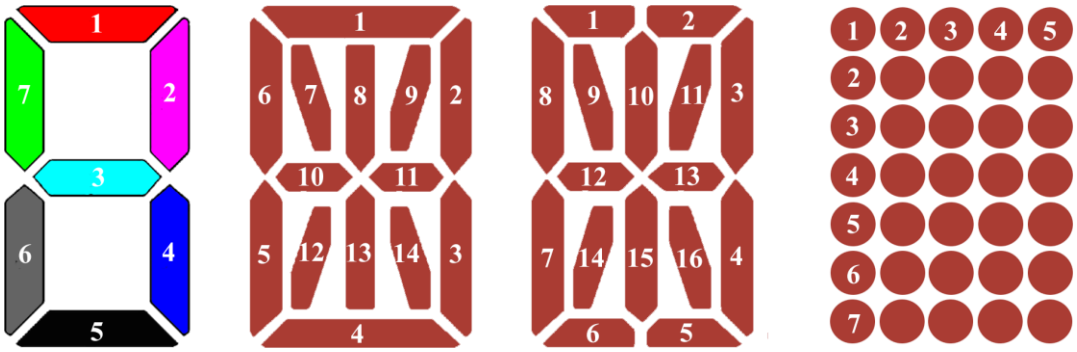



*Figure 3.11: Seven-Segment, Fourteen-Segment, Sixteen Segment and 5x7 dot matrix formats respectively*

The proposed Seven-Segment classifier exploits hardcoded information that describes all 128 possible states, for each state a boolean array describes whether each of the seven segments is lit or not. The classifier operates in three consecutive steps. First it employs seven-segment mapping to count the number of pixels in the input image that lie on each segment; the mapping is essentially an image of the same size as the input, where every

pixel value holds the identity of the segment that encapsulates that pixel. Simply put, the seven-segment mapping resembles the seven-segment image in Figure 3.11, where every red pixel has the value of one and every purple pixel has the value of two and so on. The second step calculates the density of pixels in each of the seven-segments, this is achieved by dividing the numbers counted in the first step by the total number of pixels in each segment. Any segment with a pixel density bigger than or equal 0.5 is considered lit. The third and final step matches the lit segments with the (hardcoded) possible states. Pseudocode in figure 3.12 below illustrates the Seven-Segment classification procedure.

```
SS_Classifier(input_image)
{
    hardcoded_states = {{"0", {T,T,F,T,T,T,T}, {"1", {F,T,F,T,F,F,F}}…}
    pixel_counter = array[7]
    for(x = 0; x < input_image.width;x++)
        for(y = 0; y < input_image.height;y++)
            if(input_image[x][y] == 255)
                pixel_counter[SSMapping[x][y])]++
    lit_segments = boolean[7]
    for(i = 0; i < 7; i++)
        lit_segments[i] = (pixel_counter[i] / SS_Segment_Size(i) ) >= 0.5
    options = emptyList()
    for(i = 0; i < hardcoded_states.length;i++)
        if(BooleanMatch(lit_segments, hardcoded_states[i][1])
            options.add(hardcoded_states[i][0]);
    return options
}
```

*Figure 3.12: Seven-Segment Classifier Pseudocode*

### 3.2.3 Segmentation Dependency

Last but not least, an exploratory approach is proposed to address the classification's dependency on segmentation. As a consequence of glare, light reflections and background clutter the segmentation component will likely yield imprecise separation of wanted segments. Figure 3.13 bellow illustrates the possible precision issues in segmentation that are direct consequences of glare.

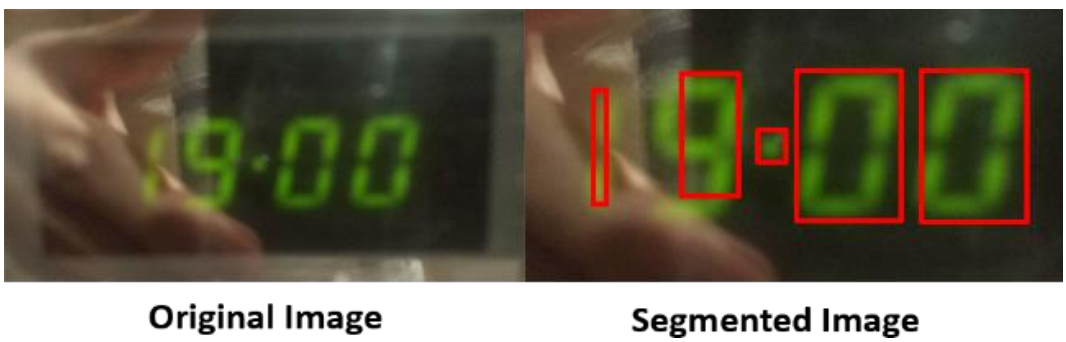



*Figure 3.13: Segmentation Precision Issues*

The original image portrayed in figure 3.13 is taken from the segmentation evaluation image set; while the segmented image is the result of Canny edge RGB histogram approach (which achieved the best performance in terms of accuracy and run-time feasibility). Although the segmentation somewhat accurately recognized the required characters, the segments are not very precise; especially the segmented 9 character. As mentioned previously, this is a direct consequence of light reflections; specifically the reflection of the image taker's hand. Moreover, this imprecision in segmentation is likely to influence the classification process; as that 9 can be easily misclassified into a 3. Consequently, we propose an exploratory approach that fundamentally takes each segment of pixels and explores the probability of neighbouring pixels being a part of the segment itself. The approach explores a dynamic amount of neighbouring pixels that depends on the size of the segment itself; essentially, it attempts to explore pixels that lie within a window that is 110% the size of the original segment and lies on the same centre point. This approach has shown significant enhancement of the segmentation precision; as illustrated in figure 3.14 below.

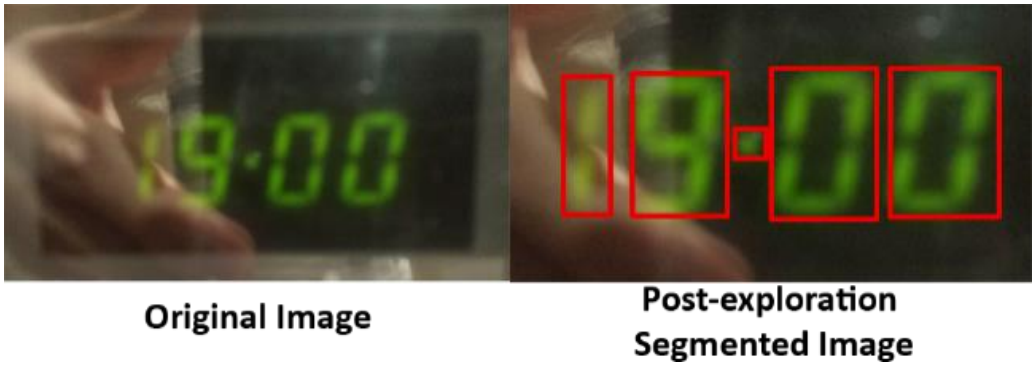



*Figure 3.14: Results of Exploratory Approach*

## 3.3 Auditory Interface

Throughout the evolution of the smart phone, the majority of the leading mobile operating systems started to integrate native system support for voice recognition and synthesis. The integration of these features commenced in the year 2009 [38]; when Android OS v1.6 introduced Text-To-Speech (TTS). Moreover, as these features evolved over time, the performance of these features improved exponentially up to a point where voice recognition and synthesis became a market standard. Today, a vast range of speech synthesizers with support to numerous languages and dialects are available. Furthermore, extremely accurate speech recognition algorithms that dynamically adapt to one's voice are available [39].

### 3.3.1 Speech Recognition

In the context of this project, the use of speech recognition will be limited to a small set of commands. Three basic commands essential in operating the application are ‘Detect’, ‘Help’, and ‘Exit’; where the ‘Detect’ command instructs the application to read the content of the targeted display, whereas the ‘Help’ command prompts the application to explain its interface, and finally the ‘Exit’ command simply instructs the application to terminate. More to the point, the speech recognition component is required to run continuously (non-stop), without online (internet) requirements, achieve accurate recognition rates and run as a light background thread without affecting the performance of the main thread. Fortunately, Android already provides support for offline continuous speech recognition and the ability to run it as an asynchronous background thread. Consequently, we conduct a simple experiment to test the accuracy of speech recognition. The experiment simply involves a blank android application that only initiates the native offline speech recognition interface. The experiment revolves around testing the accuracy of the native library on the three chosen commands. The results are illustrated in table 3.3 below.

| Command | ‘Detect’ | ‘Help’ | ‘Exit’ |
|---|---|---|---|
| Accuracy | 14/25 | 18/25 | 21/25 |

*Table 3.3: Native Android Speech Recognition Experiment*

Granted this trivial experiment only tests my personal voice and does not consider any variation in significant factors including background noise, accent, and distance from microphone. The experiment is merely meant to illustrate that even the most sophisticated speech recognition engines available do not achieve perfect results. Taking a hint from Microsoft’s Xbox, we propose a simple modification that aims to boost the accuracy of the interface. The modification entails that a fixed word is added to the beginning of all commands, similar to how the Xbox operates. We chose the fixed word ‘Eyeris’, the proposed name of the application, such that the commands are now ‘Eyeris Detect, ‘Eyeris Help’ and ‘Eyeris Exit’. Therefore, we re-conduct the same

experiment to analyze differences in accuracy. The results are illustrated in table 3.4 below.

| Command | ‘Eyeris Detect’ | ‘Eyeris Help’ | ‘Eyeris Exit’ |
|---|---|---|---|
| Accuracy | 22/25 | 24/25 | 24/25 |

*Table 3.4: Second Native Android Speech Recognition Experiment*

The results clearly show a significant improvement in recognition accuracy. We re-iterate that these experiments are merely meant to highlight the accuracy of the native Android speech recognition interface and clarify the basic design choices made throughout the development of this component. A more comprehensive evaluation of the final speech recognition component is available in chapter 5.

### 3.3.2 Speech Synthesis

The implementation of a speech synthesizer is well beyond the scope of this project. Fortunately, the diversity of natively available state-of-the-art speech synthesizer on the Android platform enables a fast and simple approach to implementing the sound-based interface. Fortunately, the only requirement for the speech synthesizer component is the ability to run as a separate/background thread; which android natively supports since the release of Android API Level 3 [40]. Moreover, the majority of tech-savvy or technically literate visually impaired smartphone users that utilize speech synthesis on a daily basis adopt custom speech synthesis interfaces. These interfaces exploit the native-android speech synthesizer and provide custom interfaces that control the pitch, the speech rate, the speech volume, and the preferred language/dialect. Consequently, the application will only provide support for direct speech volume control via the dedicated volume buttons in an effort to reduce the complexity of operating this application when custom speech synthesis interfaces are present. However, in cases where the user does not utilize custom interfaces, the speech synthesis interface can be controlled from the native Android settings application.

The utilization of the native Android speech synthesis interface concurrently with continuous speech recognition interface poses a significant challenge. The challenge stems from the use of continuous speech recognition, as the speech recognition engine is bound to encounter the synthesized speech. Two approaches exist to tackling this challenge, the first approach proposes an alternating framework where one

asynchronous thread controls both speech recognition and speech synthesis components. Consequently, the framework will ensure that only one component is active at a time. The other approach involves incorporating one other functionality to the speech recognition engine, the ability to recognize that the detected speech is synthesized. The ability to recognize whether speech is synthesized or not is a well addressed challenge in the context of speech recognition. Nevertheless, given that we control the list of words that the speech synthesize can express, a simple matching between the speech recognition and the list provides a relatively accurate and simple approach to tackling this challenge. This challenge and its consequences on the final implementation are more comprehensively detailed in chapter 4.1.5.

### 3.3.3 Contextual Information Analysis

The user interface fundamentally addresses the challenges of machine interaction with the visually impaired. The interface will primarily be in charge of three components: speech recognition, speech synthesis and contextual information analysis. More to the point, the interface receives a set of lines, each line is a spatially organized set of words; a word describes a set of characters that as a whole portray one piece of information. The interface will then attempt to derive meaning out of the words, relying on historical recognition data, the set of symbols found in the same line and classification confidences. Deriving meaning out of characters or words is usually referred to as contextual information analysis or contextual content analysis.

In order to derive meaning out of words, first we must define the set of all possible word categories. By iterating through an enormous selection of household appliances, the set of possible word categories can be determined. The identified categories include time, date, temperature, number and text. Given the preceding five word categories, the classification confidences for each character and the types of characters present in a word, we propose a confidence-based word categorization procedure. The procedure essentially handles a line at a time, where it iterates through all words in that line and assigns every word a category and a confidence. The procedure initially categories words by adopting the highest confidence character options and employing simple elimination techniques; i.e. Words categorized as ‘Time’ must contain at least one digit, can only contain “AM” or “PM” alphabetic characters and must abide to the standard time formatting (hours 0 – 23, minutes 0-59, seconds 0-59), whereas words categorized

as ‘Temperature’ must contain at least one digit and one alphabetic character where the alphabetic character must be either an ‘f’ or a ‘c’ or a ‘k’ for Fahrenheit, Celsius and Kelvin respectively. All elimination techniques are basically format-based assumptions and are more comprehensively detailed in Appendix E. Initially, the confidence value assigned to each word is the sum of the classification confidences for each chosen letter in the word. Throughout the Initial categorization process, if any word can be parsed to two or more categories then the word is parsed to the category that is more common in that line. Once all words in the line are assigned a category and a confidence, the algorithm reiterates through all words in that line in order of highest confidence to lowest and attempts to re-categorize each word based on higher confidence categories present in the same line. One must emphasize that the classification component returns a list of options and their respective confidences for each character in a word. Consequently, in the final stage of the procedure, the category of the word can only change if higher confidence categories exist in the same line, if the word can be parsed to a higher confidence category based on character options with lower confidence and if the new confidence value, calculated as the sum of classification confidences for the chosen characters, is within an acceptable range from its original confidence value. Simply put, this method initially attempts to categorize words based on their most probable classification options and then considers whether any word can be re-categorized based on its surroundings and other probable classification options. Refer to figure F.1 in Appendix F for a visual illustration of the operation of contextual information analysis on a given example.

# 4 Implementation

This chapter revolves around the final implementation of the core system. The system architecture adopts a client-server model, consisting of two parts, the client systems and the server systems. The adopted model essentially defines the basic distribution of the system and the inherent relationship between both parts; where the client application always initiates the connection to the server, while the server process is always on hold anticipating requests from any client. Consequently, this chapter is split into two sections. The first section describes the implementation of the client system as an Android application. Whereas, the second section describes the implementation of the server system as a Java servlet.

## 4.1 Client

In this context, the client is implemented as an Android Application. The application is designed specifically to tackle character recognition in LEDs and LCDs. LEDs and LCDs are the most dominant types of digital displays present in household appliances. Fundamentally, this application addresses character recognition in digital displays to facilitate the operation of digital display-equipped household appliances for the visually impaired. More to the point, smartphones provide the perfect environment for this project. The majority of active smartphones feature high resolution cameras perfect for computer vision, high definition microphones and speakers which enable the operation of a user-friendly auditory interface, state of the art speech recognition and synthesis libraries that facilitate the development of the auditory interface and multi-core processors capable of handling the load.

Google currently supports over one billion active monthly Android users [41], is one of many reasons why this application was built for the Android OS. Other reasons include the fact that Android was one of the first mobile operating system to launch support for speech recognition and synthesis in 2009 [38] and according to the latest surveys Android holds the undisputed lead in mobile operating system market share at 78% as of the first quarter of 2015 [42]. As previously mentioned, the core contributions of this project revolve around the design and development of segmentation and classification techniques that are implemented in this application. Consequently, as expected, the main focus of the implementation is the client system. Although the development of the client system was initially scattered across MATLAB, native java and Android Studio, the final application is implemented in java solely on Android Studio. Moreover, the final android application is dependent on two distinct libraries OpenCV and LIBSVM.

OpenCV provides variety of useful algorithms that facilitate computer vision calculations relevant in this context and LIBSVM provides a comprehensive and efficient SMO implementation of Support Vector Machines. The application consists of three integral components: segmentation, classification and the auditory interface. The class diagram portrayed in figure 4.1 below illustrates the overall structure of the client system.

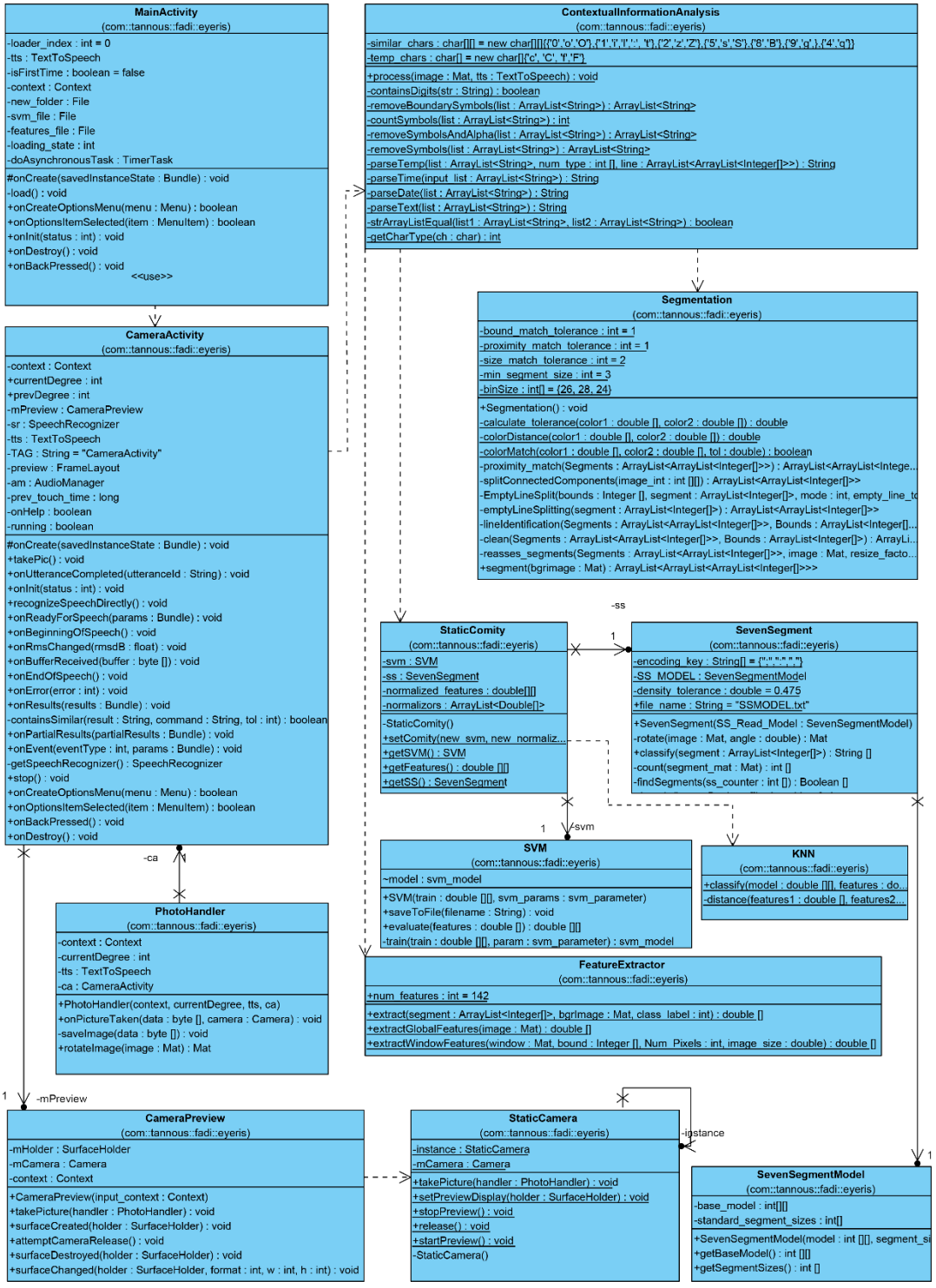



*Figure 4.1: Client Class Diagram*

This section is in term split into six sub-sections. The first section comprehensively describes the implementation environment of the client system. The second section reveals the implementation details of the segmentation component. The third section illustrates the pre-classification methods implemented for feature selection and normalization. Then the fourth section reveals the implementation details of the classification component. The fifth section describes the implementation of the auditory interface and the utilization of native Android libraries for speech recognition and synthesis. The sixth section illustrates the implementation of the dynamic classification model update protocol. The seventh and final section describes significant miscellaneous methods implemented in this application.

### 4.1.1 Implementation Environment

This subsection provides a comprehensive overview of the Android implementation environment, aiming to offer all the details required to recreate the same environment in Android Studio. To begin with, the application was implemented on Android Studio Version 1.1.0. Next we must define the basic parameters of the application, including minimum SDK version, target SDK version, and compile SDK version; set to 19, 21, and 21 respectively. Finally, this application is dependent on two libraries, OpenCV and LIBSVM. Properly importing these libraries is crucial for this implementation. LIBSVM can be imported easily by following the standard approach provided by Android [43] for setting up a Library module. On the other hand, OpenCV is written in C++ and the library's java wrapper must be properly linked to the implementation environment. Two approaches exist that address linking the OpenCV wrapper to Android: Static binding or Dynamic linking. Details about both approaches are available in Appendix C.

### 4.1.2 Segmentation

The segmentation component is perhaps the most significant contribution in the perspective of this project. The component's significance is a direct consequence of the core challenges associated with digital displays. When captured on cameras, digital displays commonly exhibit various visual irregularities: including glare, light reflections (specular or diffuse), oversaturation and high contrast bleeding. These visual irregularities are the reason that most state-of-the-art character recognition systems fail

when tackling digital displays. Consequently, addressing these irregularities is the core objective of this project.

In the context of character recognition, two integral components stand out: segmentation and classification. Segmentation is responsible for splitting the input image into sets of pixels, where each set of pixels describes one character, and discarding all background pixels. Whereas, the classification component is responsible for identifying the character or symbol that every set of pixels represent. The previously mentioned visual irregularities directly affect the segmentation component's ability to accurately identify unique characters. This is a consequence of the visual irregularities' direct influence on pixel values, specifically pixels that represent characters of interest. Simply put, humans identify unique character segments in an image based on the color consistency in the pixels that represent that character and any change in the color of some of these pixels directly affects the ability to identify unique characters. The segmentation component in essence attempts to mimic the human's ability to recognize characters from background. However, the average human possesses an enormous amount of experience from real-life situations that allow him/her to identify visual irregularities and adjust their perception of the image accordingly. On the other hand, machines are much less capable of tackling visual irregularities, as any change in pixel values within a character would inherently lead the segmentation component to consider further segmentation of that character. The classification component receives a set of organized pixels for each character, regardless of color, and attempts to classify it. Consequently, the classification component is not directly affected by any visual irregularities. Nevertheless, the classification component is naturally dependent on the segmentation accuracy and in term indirectly affected by any visual irregularities. Therefore, the segmentation component must tackle these challenges head on.

Over the course of this project, eleven unique segmentation algorithms were designed and evaluated. Each algorithm was evaluated on a set of 36 images that represent our targeted environment. In terms of performance, the Canny edge based RGB histogram was able to achieve the best results, accomplishing ~96% average segmentation accuracy within an average of 631 milliseconds. As implied by the name, this approach relies on the RGB histogram to provide appropriate color difference information and exploits a Canny edge detector to quickly obtain spatial information. Briefly, the algorithm operates in three consecutive steps: the first step revolves around splitting

connected components obtained by the Canny edge detector, the second step employ the RGB histogram to calculate the background and foreground color for each connected component and finally, the last step partitions the content of the input image by identifying connected components with similar background and foreground colors. This approach operates under the assumption that characters in the same word exhibit the same color and lie on the same background. The algorithm's efficiency stems from inherent speed of the Canny edge detector.

Although, 631 milliseconds can be considered too lengthy for a quasi-real-time implementation; effectively tackling visual irregularities is more significant to this project. Nevertheless, we consider two run-time optimizations, other than exploiting the city block distance metric, and evaluate their influence independently. As previously stated, any run-time optimization decision is evaluated based on two features: accuracy and run-time. Both optimizations revolve around the chosen resolution for processing input images. As previously mentioned, initial testing on MATLAB revealed that 256x144 is the optimal resolution for processing. The first proposed optimization intends to lower the processing resolution for the entire segmentation component to 128x72 (by half). This optimization yielded an average of 300 milliseconds run-time and an average of 74% accuracy with negligible standard deviations. This optimization led to ~52% decrease in run-time while imposing ~22% decrease in accuracy. This optimization is not included in the final implementation. Nevertheless, it inspired the following optimization.

Even though the Canny edge detector provides a relatively fast approach to exploiting spatial information, it still is the most computationally intensive calculation performed in this algorithm. Consequently, this proposed optimization intends to lower the processing resolution, just for the Canny edge detector, and then resize the result to match the RGB histogram dimensions. Therefore, we evaluate lowering the processing resolution to 128x72 (by half) for the Canny edge detector processing segment. This optimized version yielded an average of 550 milliseconds run-time and an average of 95% accuracy with negligible standard deviations. This optimization led to ~13% decrease in run-time while imposing ~1% decrease in accuracy. This optimization is included in the final implementation.

The final implementation of the segmentation algorithm is briefly illustrated in figure 4.2 below.

```
Segementation(input_image_256x144)
{
    color_information = Histrogram.RGB(input_image_256x144)
    Imgproc.resize(input_image_256x144, resized_image, new Size(128, 72))
    Imgproc.Canny(resized_image, canny_image)
    Imgproc.resize(canny_image, canny_image, new Size(128, 72))
    connected_components = SplitConnectedComponents(canny_image)
    BG_FG_colors = calculate_bg_fg_colors(connected_components, color_information)
    lines = identifySimilarlyColored(connected_components, BG_FG_colors)
    return lines
}
```

*Figure 4.2: Final Segmentation Implementation Pseudocode*

The Pseudocode illustrated above describes the overall structure of the code. Full implementation details are included in the attached files, refer to Appendix A.

### 4.1.3 Preprocessing: Feature Selection and Normalization

In the field of machine learning, feature selection and extraction are preprocessing procedures responsible for identifying the set of relevant features used in the construction of the classification model. On the other hand, feature normalization is a preprocessing procedure used to standardize the range of independent features in a dataset [44]. Feature normalization is not a required preprocessing procedure; however, it does have a direct influence on the classification objective functions. Essentially, classifiers attempt to classify a data sample by calculating the difference between the features that represent that data sample and the features of all data samples in the classification model. The difference is commonly calculated in terms of Euclidean distance. Consequently, if the features selected for classification are not normalized then some features will have a broader range of values than others which leads to disproportional feature contributions to the calculated distance. Basically, feature normalization ensures that all features have a proportional contribution to classification.

Unfortunately, the majority of common feature selection and extraction techniques relevant to the field of character recognition are computationally infeasible for a smartphone implementation. Nevertheless, some of the most accurate state-of-the-art character recognition systems employ discriminating features that describe patterns prevalent in the language that the system is aimed at. This approach only works in language specific character recognition systems. In this context, the targeted language is a combination of the English alphanumeric set and the set of symbols commonly

incorporated in household appliances. The vast majority of relevant studies explore discriminating features present in handwritten characters rather than digital or printed ones. This is a consequence of the consistency of printed characters which enables the use of detail oriented features rather than adopting a generalistic approach. One must emphasize that in this context we are tackling digital characters rather than handwritten ones.

The most common features used for English printed character recognition are usually calculated from image moments. An Image moment is a weighted average of the image pixels' intensities. Usually functions of these moments are used to calculate useful image properties. Properties of an image that can be calculated by the image moments include area, centroid, orientation information, eccentricity, and Euler number. The area is calculated in terms of total intensity to quantify the extent of the two-dimensional object or character. The centroid expresses the geometric center of the 2D character in terms of the average position of the intensities in the character. The orientation information is usually expressed as the angle of rotation of the character. Whereas the eccentricity refers to the smallest eclipse that fits the entire character skeleton. Finally, the Euler number is the number of separate objects minus the number of holes in the character. The utilization of image moments is motivated by their ability to achieve orientation and scale invariance.

Other significant feature selection techniques include windowed analysis and boundary analysis. Windowed analysis refers to the process of dividing an image into windows of equal size and analyzing the content of each window separately; usually the number and normalized length of lines present in each window is used. Whereas boundary analysis refers to a simple approach that is known to work well with printed characters; the approach relies on calculating boundary points for every row and every column present in the image. Unfortunately, calculating the boundary points for every row and column is computationally intensive; nevertheless, this approach enables a simple trade-off between accuracy and computational intensity by simply skipping some rows and columns.

Initially, we propose three distinct sets of features motivated by geometric consistencies present in English characters.

**Set 1: Image Moments Features**

1. Euler Number.
2. Eccentricity.
3. Angle of rotation/orientation.
4. Normalized Area.
5. Centroid.

**Set 2: 2x2 Window Features**

1. Number of horizontal lines.
2. Number of vertical lines.
3. Number of right diagonal lines.
4. Number of left diagonal lines.
5. Normalized length of horizontal lines.
6. Normalized length of vertical lines.
7. Normalized length of right diagonal lines.
8. Normalized length of left diagonal lines.
9. Normalized area of window.

**Set 3: Boundary Features**

Two boundary points for all odd rows and columns.

Every combination of these three sets is evaluated in terms of classification accuracy and feature selection run-time requirements. The evaluation is performed by calculating classification accuracy on a linear SVM; the linear SVM model is built using English alphanumeric characters across five common fonts. The fonts include Gill Sans, Calibri, Arial, Cambria, and Twentieth Century; the same set of fonts used to build the final implementation of the KNN and SVM classifiers. The following figures illustrate the performance evaluation results for every combination of feature sets.

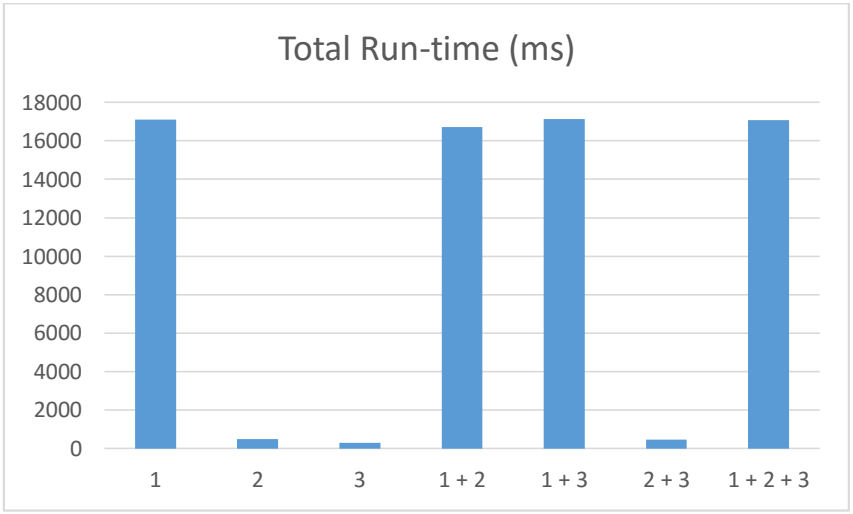



*Figure 4.3: Un-normalized Feature Set: Run-time Evaluation*

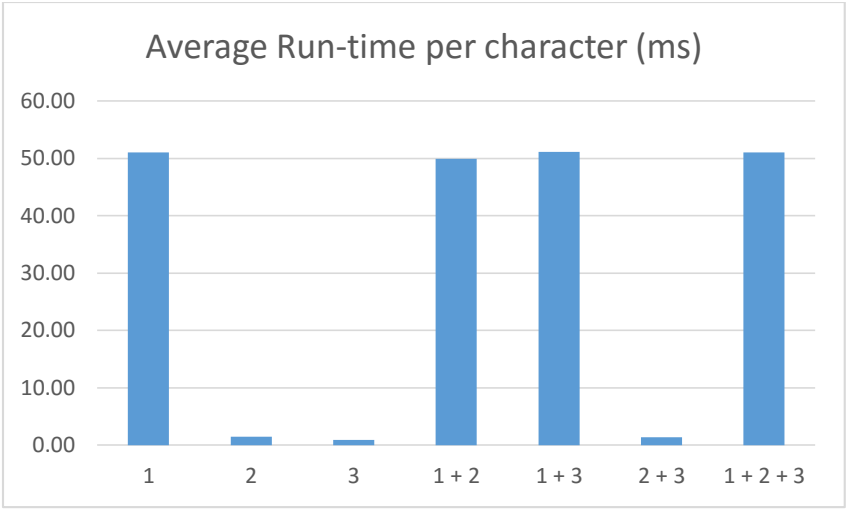



*Figure 4.4: Un-normalized Feature Set: Average Run-time (per character) Evaluation*

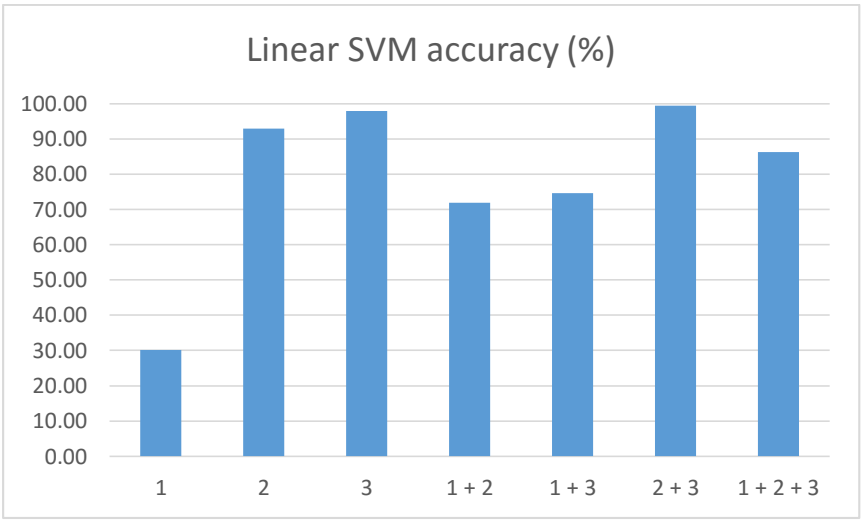



*Figure 4.5: Un-normalized Feature Set: Classification Accuracy Evaluation on Linear SVM*

The preceding figures use the numbers 1, 2 and 3 to refer to the aforementioned feature sets (image moments, window and boundary features, respectively).The preceding results clearly show that the image moments feature set is very computationally intensive. The image moments feature sets exhibits low accuracy because these features are constructed to enable scale and orientation invariance. The final implementation of the application exploits only the boundary and window feature sets and requires around 0.9 milliseconds of processing per character.

Feature normalization is relatively simple to implement. Basically, the algorithm iterates over the same feature in every sample in the dataset and calculates the mean and standard deviation. The algorithm then reiterates in the same fashion and

standardizes the value of every feature to $\frac{\text{feature\_value} - \text{mean}}{standard\_deviation}$. The following figure illustrates java pseudocode for feature normalization.

```
Feature_Normalization(double[][] features)
{
    double[][] normalized_features = new double[features.length][ features[0].length];
    for(int i = 0; i < features[0].length;i++)
    {
        double mean = 0;
        for( int j = 0; j < features.length;j++)
            mean += features[j][i];
        mean = mean/features.length;
        double standard_deviation = 0;
        for( int j = 0; j < features.length;j++)
            standard_deviation += Math.pow(features[j][i] - mean);
        standard_deviation = Math.sqrt(standard_deviation/features.length);
        for( int j = 0; j < features.length;j++)
            normalized_features[j][i] = (features[j][i] - mean) / standard_deviation;
    }
    return normalized_features
}
```

*Figure 4.6: Feature Normalization Pseudocode*

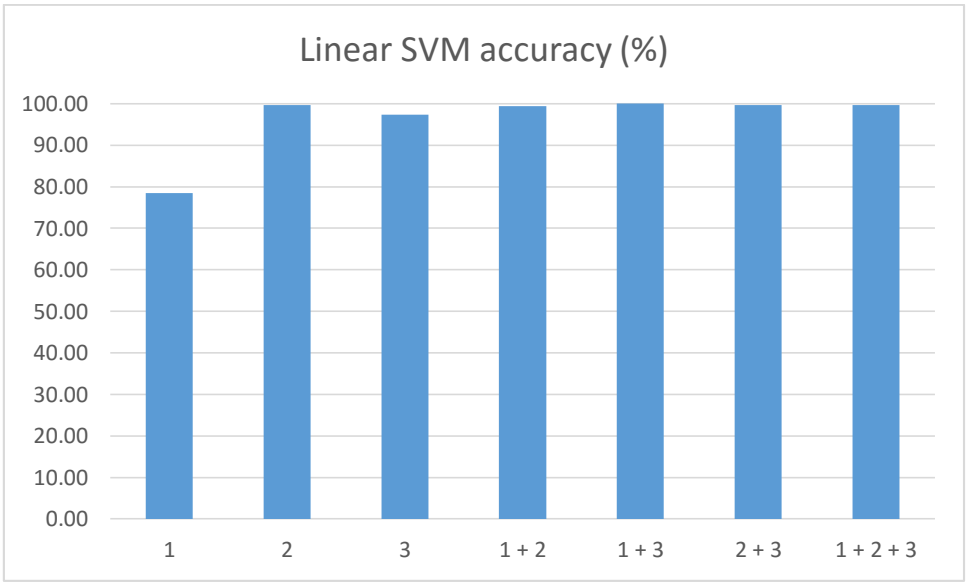



*Figure 4.7: Normalized Feature Set: Classification Accuracy Evaluation on Linear SVM*

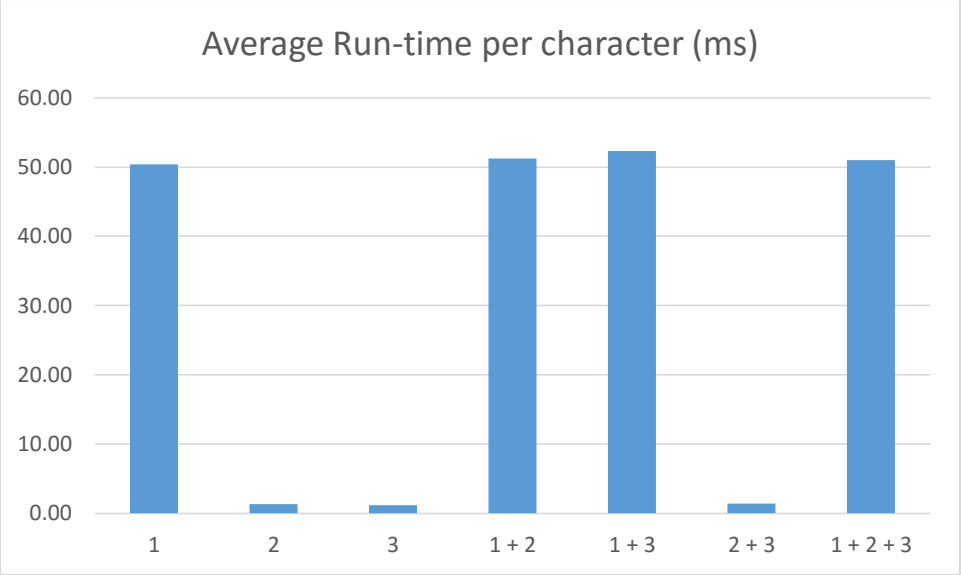



*Figure 4.8: Normalized Feature Set Average Run-time (per character) Evaluation*

The results presented above describe the influence of feature normalization on the performance of each of the evaluated feature sets. The results clearly show that feature normalization positively influences the classification accuracy and has no significant effect on feature extraction run-time. The complete implementation of these experiments and their respective results are available in the attached files, refer to Appendix A.

Implemented pre feature selection procedures include: gray-scaling, binarization, skeletonization and domain of discourse. Gray-scaling and binarization are simple image transformations that essentially change how pixels are represented. As implied by the names, gray-scaling reduces the pixel representation to the gray-scale and binarization reduces the pixel representation to the binary scale. Consequently, grayscale image pixels are represented by the range of shades of gray usually 0 - 255, whereas binary image pixels are represented by either a zero or a one (black and white respectively). Both image transformation techniques are commonly used to facilitate faster feature selection procedures; as reducing the range of pixel representation directly affects the computational requirements of image processing. Skeletonization is a prevalent algorithm in shape analysis that produces a skeleton or a thin description of the original shape. Skeletonization is used to highlight topological features in characters, such as intersections and connectivity. Finally, the domain of discourse refers to the smallest image matrix that can fit the entire image skeleton. Domain of discourse is easily calculated by traversing from each side of

an image towards the center eliminating an empty lines. The domain of discourse is used to exclude empty content enabling more accurate positioning. The following figure illustrates the implemented pre feature selection procedures on a sample image.

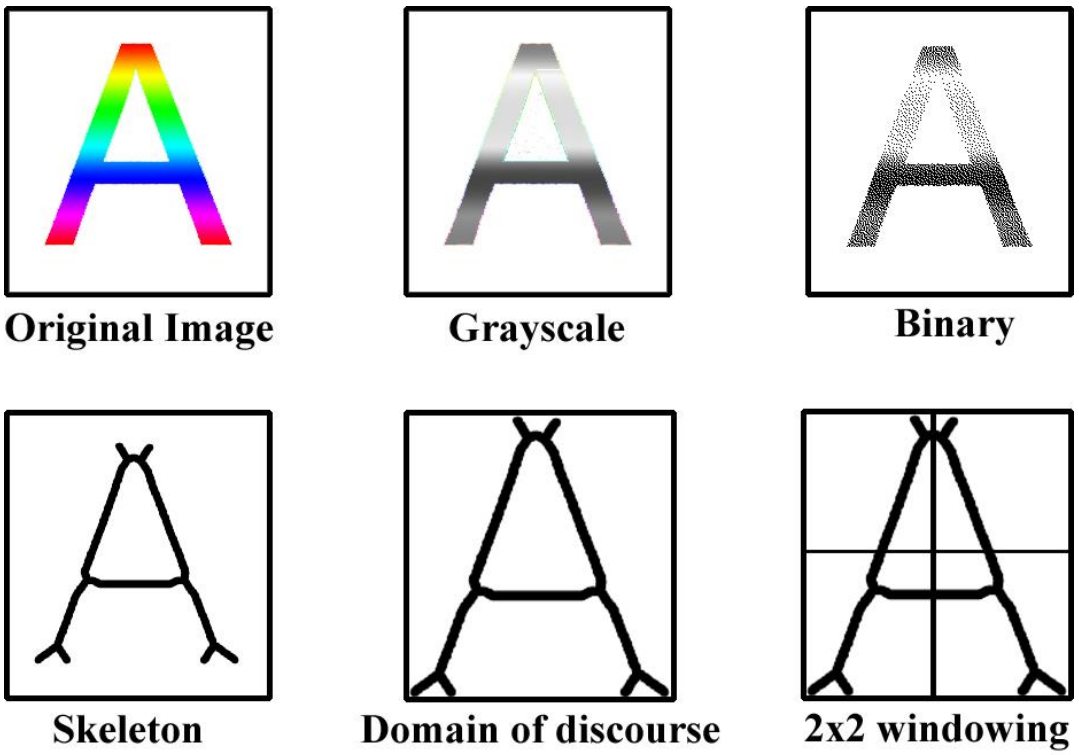



*Figure 4.9: Pre feature Selection procedures*

### 4.1.4 Classification Ensemble

This subsection revolves around the final implementation of the classification component. The classification component is developed as an ensemble of classifiers, where multiple unique classifiers are aggregated in a majority vote for classification. The implemented ensemble encompasses a diverse selection of unique classifiers including a Support Vector Machine, a K-Nearest Neighbor classifier and a Seven-Segment classifier.

First we must define the set of classes used for training the classifiers, the training dataset. Unfortunately, it is near-impossible to define a set that covers all possible classes that characters in the targeted environment might belong to. This is a direct consequence of the level of abstraction of the targeted environment. Essentially, the application targets digital displays commonly found in household appliances and

this target environment is too general for anyone to define. For example, if the application targets digital displays commonly found in ovens then it would be easier to define a set of classes that covers all possible characters found on oven displays; as ovens usually display temperature related symbols. Therefore, we build a training dataset based on various assumptions to fit our needs. First of all, we assume that all digital displays employ the English alphabet and the Arabic numerals to display letters and digits respectively. Throughout this dissertation, we refer to the combination of the English alphabet and the Arabic numerals as the English alphanumeric set. Then we assume that objects, which the application is interested in recognizing, displayed on household appliance displays can only belong to one of the three following categories: letters, digits and symbols. As implied by the combination of these two assumptions, the set of letters and digits that might be encountered in the targeted environment is well defined while the set of symbols is not. Consequently, we are able to define two distinct but complementary sets of classes that comprehensively cover all possible characters that might be encountered. The first set is a finite static set that covers the English alphanumerics; including a-z, A-Z and 0-9 across five common and distinct fonts. These fonts include Gill Sans, Calibri, Arial, Cambria, and Twentieth Century. The fonts were chosen by manually matching fonts commonly found in household appliance displays with Google's online font directory [45]. On the other hand, the second set is a continuous set of symbols, which is initially defined to include the following symbols: ' + ', ' – ', ' . ' and ' : '. The set of symbols is considered continuous as it is dynamically updated using an online server, essentially as the application encounters unknown symbols it reports back to the server which attempts to classify these symbols and appropriately updates the classification models. The initial server implementation seeks manual input to classify unknown symbols, various techniques to automate this process are considered in the final evaluation.

The implementation of a Support Vector Machine is beyond the scope of this project. This is a consequence of the diversity of options that a Support Vector Machine can operate under. Consequently, this project relies on LIBSVM's implementation of Support Vector Machines. The following pseudocode segment in Figure 4.10 illustrates the use of LIBSVM in the context of this project.

```
public svm_model train(double[][] training_dataset, svm_parameter params)
{
    svm_problem prob = new svm_problem();
    int dataCount = training_dataset.length;
    prob.y = new double[dataCount];
    prob.l = dataCount;
    prob.x = new svm_node[dataCount][];
    for (int i = 0; i < dataCount; i++){
        double[] features = training_dataset[i];
        prob.x[i] = new svm_node[features.length-1];
        for (int j = 1; j < features.length; j++)
        {
            svm_node node = new svm_node();
            node.index = j;
            node.value = features[j];
            prob.x[i][j-1] = node;
        }
        prob.y[i] = features[0];
    }
    svm_model model = svm.svm_train(prob, param);
    return model;
}
public double evaluate(svm_model model,double[] features)
{
    svm_node[] nodes = new svm_node[features.length-1];
    for (int i = 1; i < features.length; i++)
    {
        svm_node node = new svm_node();
        node.index = i;
        node.value = features[i];
        nodes[i-1] = node;
    }
    Integer[] index_array = new Integer[Util.classes.length];
    for(int i = 0; i < index_array.length;i++)
        index_array[i] = i;
    int[] labels = new int[Util.classes.length];
    svm.svm_get_labels(model,labels);
    double[] prob_estimates = new double[Util.classes.length];
    return svm.svm_predict_probability(model, nodes,prob_estimates);
}
```

*Figure 4.10: SVM Implementation*

On the other hand, the implementation of a K-Nearest Neighbor classifier is relatively straightforward. Essentially, the K-NN classifier operates by calculating the difference between the features of the object it is aiming to classify and the set of features that describe every character in the training model. The K-NN classifier then returns the most common class of the K-nearest training samples. The following code segment in Figure 4.11 describes the final implementation of KNN.

```
public double[][] classify(double[][] model, double[] features , int k)
{
    final double[] distances = new double[model.length];
    ArrayList<Integer> index_list = new ArrayList<Integer>();
    for(int i = 0; i < model.length;i++)
    {
        index_list.add(i);
        distances[i] = distance(features, model[i],true);
    }
    Collections.sort(index_list, new Comparator<Integer>()
    {
        @Override
        public int compare(Integer a, Integer b)
        {
            if(distances[a] > distances[b])
                return 1;
            if(distances[b] > distances[a])
                return -1;
            return 0;
        }
    });
    double[][] result = new double[k][];
    for(int i = 0; i < k;i++)
        result[i] = new double[]{model[index_list.get(i)][0],
                                 distances[index_list.get(i)]};
    return result;
}
private static double distance(double[] features1, double[] features2)
{
    double city_block_distance = 0;
    for(int i = 1; i< features1.length;i++)
        city_block_distance += Math.abs(features1[i] - features2[i]);
    return city_block_distance;
}
```

*Figure 4.11: KNN Implementation*

Next, the implementation of the Seven-Segment classifier is illustrated. As previously mentioned, the Seven-Segment classifier essentially works by matching the input with hardcoded data that describe all 128 possible states of the Seven-Segment format. The classifier initially analyzes pixel densities in the input to figure out which of the Seven-Segments are potentially lit and then compares the result

with hardcoded state information. The following code segment in Figure 4.12 describes the final implementation of the Seven-Segment classifier.

```
private static final String hardcoded_states = "0:1,1,0,1,1,1,1;1:0,1,0,1,0,0,0;2:1,1,1,0,1,
private static final String[] encoding_key = {";",":",","};
public String[] classify(SSModel SS_MODEL, Mat segment_mat)
{
    int[] segment_sizes = SS_MODEL.getSegmentSizes();
    int[] ss_counter = count(segment_mat);
    Boolean[] SevenSegments = findSegments(ss_counter);
    ArrayList<String> result = decode(SevenSegments);
    return result.toArray(new String[result.size()]);
}
private int[] count(SSModel SS_MODEL,Mat segment_mat)
{
    int[][] base_model = SS_MODEL.getBaseModel();
    int[] ss_counter = new int[]{0,0,0,0,0,0,0};
    for(int rows = 0; rows < segment_mat.height();rows++)
        for(int cols = 0; cols < segment_mat.width();cols++)
        {
            if(segment_mat.get(rows, cols)[0] != 0)
            {
                int index = base_model[rows][cols];
                ss_counter[index]++;
            }
        }
    return ss_counter;
}
private Boolean[] findSegments(SSModel SS_MODEL,int[] ss_counter)
{
    Boolean[] seven_segments = new Boolean[]{false,false,false,false,false,false,false};
    int[] standard_segment_size = SS_MODEL.getSegmentSizes();
    int max_count = 0;
    for(int i = 0; i < ss_counter.length-1;i++)
        if(ss_counter[i] > max_count)
            max_count = ss_counter[i];
    for(int i = 0; i < ss_counter.length-1;i++)
    {
        if(ss_counter[i]/(double)standard_segment_size[i] >= density_tolerance
        || ss_counter[i] > max_count/2)
            seven_segments[i] = true;
    }
    return seven_segments;
}
private ArrayList<String> decode(Boolean[] input)
{
    ArrayList<String> result = new ArrayList<String>();
    String[] encoded_numbers = hardcoded_states.split(encoding_key[0]);
    for(int i = 0; i < encoded_numbers.length;i++)
    {
        String[] encoded_number = encoded_numbers[i].split(encoding_key[1]);
        String[] decoded = encoded_number[1].split(encoding_key[2]);
        boolean match = true;
        for(int j = 0; j < decoded.length;j++)
        {
            boolean toBool = decoded[j].equals("1");
            if(toBool != input[j])
            {
                match = false;
                break;
            }
        }
        if(match)
            result.add(encoded_number[0]);
    }
    return result;
}
```

*Figure 4.12: Seven-Segment Classifier Implementation*

Finally, the implementation of the entire ensemble is discussed. The ensemble is implemented to mimic a static class. Unfortunately, Java does not support top-level static classes; however, a top-level class can be built to simulate static classes by following three rules. First, the class must be declared as final preventing any extension. Second, the class constructor must be a private function which prevents instantiation. Finally, ensure all members of the class (variables and functions) are declared static. The ensemble is implemented to mimic a static class in order to facilitate cross-activity utilization and loading. Essentially, the application requires the ability to construct the ensemble in the loading activity for utilization in the camera activity. Two common approaches are used to enable sharing variables between activities, the first approach involves using static public classes while the second approach relies on Android ability to pass Parcelable objects through activities. Given that the application only requires one ensemble for use throughout the application, we exploit the "Singleton" approach, detailed above, to mimic a static class [46].The following code segment in Figure 4.13 describes the final 'Singleton' implementation of the ensemble.

```
1  public class StaticEnsemble
2  {
3      private static SVM svm;
4      private static SevenSegment ss;
5      private static double[][] normalized_features;
6      private StaticEnsemble(){}
7      public static void set(SVM new_svm,
8                  double[][] new_normalized_features,SevenSegment new_ss)
9      {
10         svm = new_svm;
11         normalized_features = new_normalized_features;
12         ss = new_ss;
13     }
14     public static SVM getSVM(){return svm;}
15     public static double[][] getFeatures(){return normalized_features;}
16     public static SevenSegment getSS(){return ss;}
17 }
```

*Figure 4.13: Static Ensemble Implementation*

### 4.1.5 Auditory Interface

The final implementation of the user interface mainly relies on speech recognition and synthesis to facilitate human-machine interaction. The auditory user interface relies on three basic commands to operate the application, including 'Eyeris Detect', 'Eyeris Help', and 'Eyeris Exit'. First and foremost, the 'Eyeris Detect' command prompts the application to read the contents of the targeted display. As implied, the

application will assume that the targeted display, or at least the relevant portion of it, is completely contained within the camera's domain. Moreover, as previously stated, the application relies on the average visually-impaired user's ability to locate his/her target display. Nevertheless, further improvements regarding assistance in the localization of the display are explored in chapter 6. Whereas, the 'Eyeris Help' command prompts the application to explain its own interface. The interface explanation is a hardcoded tutorial, in the form of a set of string, that explains the basic functionality of each command. The tutorial is fundamentally split into multiple strings to enable appropriate delays when needed, e.g. between consecutive sentences. The following 9 sentences define the tutorial.

"iris offers a simple speech based interface, featuring three commands"
"iris exit"
"to exit the application"
"iris help"
"to receive an explanation about the interface"
"iris detect"
"to read the content of the targeted display"
"try your best to aim the camera directly at the target"
"double tap to interrupt the text-to-speech service"

Finally, the 'Eyeris Exit' command prompts the application to terminate. As described in the last sentence in the tutorial, the interface relies on touch-screen events to enable the interruption of synthesized speech. Given that "Iris" and "Eyeris" are Homophones, or words that share the same pronunciation regardless of their spelling, the speech recognition is implemented to identify "Iris Detect" rather than "Eyeris Detect". This is required as the word "Eyeris" is not included in the speech recognition's dictionary, an alternative approach would simple ensure that "Eyeris" is present in the utilized dictionary. Moreover, the speech synthesis component is also instructed to synthesize "Iris" rather than "Eyeris" simply for the sake of consistency.

The concurrent operation of native Android speech recognition and speech synthesis poses a challenge to this implementation. As previously stated, this is a consequence of operating a continuous speech recognition thread that is bound to encounter the synthesized speech. The final implementation relies on an alternating framework

that controls both speech recognition and synthesis operations in one asynchronous thread. The thread is built to ensure that only one operation, either recognition or synthesis, is active at a given time. This approach is preferred over incorporating extra functionality in speech recognition that enables the determination of whether the recognized speech is synthesized or not, as it would incur extra computational requirements for the speech recognition component. The following figure illustrates the final implementation of this alternating framework using pseudocode.

```
public class activity extends Activity implements RecognitionListener
    ,TextToSpeech.OnInitListener, TextToSpeech.OnUtteranceCompletedListener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        TextToSpeech tts = new TextToSpeech(this, this);
        SpeechRecognizer sr = SpeechRecognizer.createSpeechRecognizer(this);
        final android.os.Handler handler = new android.os.Handler ();
        Timer timer = new Timer();
        TimerTask doAsynchronousTask = new TimerTask()
        {
            @Override
            public void run()
            {
                handler.post(new Runnable()
                {
                    @SuppressWarnings("unchecked")
                    public void run()
                    {
                        try
                        {
                            while(tts.isSpeaking()){}
                            Intent recognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
                            if (!recognizerIntent.hasExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE))
                                recognizerIntent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,
                                                          this.getPackageName());
                            sr.setRecognitionListener(this);
                            sr.startListening(recognizerIntent);
                        }
                        catch (Exception e){Log.d("Ex", e.getMessage());}
                    }
                });
            }
        };
        timer.schedule(doAsynchronousTask, 0,0);
    }
}
```

Figure 4.14: Speech Control Framework Implementation

Last but not least, the final implementation of the contextual information analysis component is illustrated. This component is responsible for deriving meaning out of the classification results. More to the point, the classification component essentially returns a set of lines, where each line is a spatially organized set of words and each word is described by a set of characters. Each character is described by a set of possible options (letters, digits or symbols) and their respective confidences. This component attempts to derive meaning out of these options based on the set of characters found, their respective confidences and the recognition history. The

component adopts a confidence based approach to word categorization. Please refer to the attached files for the complete implementation details. Figure F.1 available in Appendix F visually illustrates the operation of this component on a given example.

### 4.1.6 Online Learning – Dynamic Classification Update Scheme

Traditionally, online machine learning refers to the case where the data used for learning becomes available in a consecutive manner such that the mapping or learning model is updated after the arrival of every new data sample. Moreover, traditionally online learning involves the use of optimization procedures such as stochastic gradient descent that enables consecutive updating of the gradient estimation. Nevertheless, the final implementation of this application employs batch learning or 'offline' learning and exploits an online server that re-trains all learning models incorporating new data samples. In this context, the term online learning simply refers to the learning models' ability to update, regardless of how it is done. This approach is chosen as batch learning techniques enable much faster classification once the model has been trained. Consequently, all models incorporated in the application are pre-trained and re-training is only required when the server updates the client's model. The ability to update the implemented classification models is required as a consequence of the level of abstraction that defines the targeted language. As previously discussed, the targeted language comprises of two sets, the English alphanumeric set (static) and the (expanding) set of symbols. Consequently, the classification models' ability to update enables a dynamic approach that expands the set of symbols supported for classification. The implemented scheme is simple, the online server regularly receives unclassified image data from various clients which is used to re-train the classification models. Initially, the server relies on manual input for labeling the unclassified image data. The use of image search engines to facilitate automated labeling is considered in chapter 4.2.

From the client's perspective, the implementation of the classification update scheme is a simple HTTP POST request. The Post request attaches all recent unclassified image data and the current classification model version available in the application. The request then awaits a response that is either empty or contains new classification model data. If a new classification model is available and the

application is running in the background or inactive then the application initiates the re-training of the models. Training an SVM model with the initial training dataset requires around 15 seconds of execution time on a Galaxy S4 and any growth in the size of the training dataset has a direct impact on training time. Given that the training dataset is expected to grow, the re-training process is expected to require more time. Fortunately, the KNN classifier does not require any re-training; in the context of KNN, the classification model is the raw training feature set. Moreover, the Seven-Segment classifier only operates on alphanumeric characters and therefore is not effected by the updates (updates are only used to expand the set of supported symbols). Nevertheless, the application does not allow re-training unless the application has not been active for two minutes or if it is running in the background; this approach aims to minimize the update's influence on the user. The following figure illustrates the implementation of the HTTP request in Android.

```
1  private HttpResponse doPostRequest(String server_ip, JSONObject jObject)
2  {
3      DefaultHttpClient http_client = new DefaultHttpClient();
4      HttpPost http_post = new HttpPost(server_ip);
5      StringEntity string_entity = new StringEntity(jObject.toString());
6      http_post.setEntity(string_entity);
7      http_post.setHeader("Content-type", "application/json");
8      ResponseHandler responseHandler = new BasicResponseHandler();
9      return http_client.execute(http_post, responseHandler);
10 }
```

*Figure 4.15: HTTP Request Android Implementation*

### 4.1.7 Miscellaneous

This section revolves around miscellaneous Android implementations used to facilitate the operation of the application. These implementations include, the use of 'Singleton' classes to implement the camera interface, the use of asynchronous threads to facilitate the application's loading process, lowering the camera's capture resolution to enable faster capture times and finally the declaration of Android permissions.

As previously stated, the 'Singleton' approach facilitates control over object creation limiting the number of instantiated objects to one. This approach is used to ensure that only one android.hardware.camera object is instantiated throughout the operation of the application and enable the use of the camera over multiple Android activities; which in term enables the instantiation of the camera in the loading activity. The hardware camera instantiation time for the Galaxy S4 I9505 is around

one and a half second. More to the point, the main activity of the application requires a fully instantiated camera object to be ready before the activity scene is created, such that once the activity scene is operational the application will be ready for detection. This approach not only ensures that the camera object is only instantiated once, but also enables dumping the camera's instantiation time on the loading activity. Please refer to StaticCamera.java in the attached code for implementation details.

On the other hand, the utilization of asynchronous threads enables a faster multi-threaded approach to loading the application. Fundamentally, as the application launches it initially enters the loading activity which is in charge of loading all the required data. The loading activity instantiates a unique asynchronous thread to load each classification model from file and one other thread to load the camera. On the other hand, the activity's main thread is only in charge of operating the loading circle and the text-to-speech tutorial (which is automatically synthesized only on the first use of the application).

Moreover, when a user instructs the application to detect the contents of the targeted display, initially the application instructs the camera to take a picture. As expected, the higher the resolution of the picture the camera is capturing the more time it requires to process the image and return it for segmentation and classification. Consequently, the capture resolution in the final implementation was lowered to 1280x720 (720p). Brief testing (also on the Galaxy S4) showed that the difference in image capture time becomes negligible as the resolution is reduced below 720p. Nevertheless, image capture time is still significant at around 1.2 second on the galaxy S4. More modern smartphones, like the Galaxy S6, have much reduced capture times (well under one second).

Finally, the application is required to declare all the permissions it needs in the manifest file. These permissions include

android.permission.camera

android.permission.write_external_storage

android.permission.record_audio

android.permission.modify_audio_settings

The permissions are self-explanatory; nevertheless, comprehensive explanations for all permissions are available on Android's developer web pages [47]. All previous permissions are marked 'required'. All these permissions are displayed to the user at installation time and if any 'required' permission is not accepted then the application will not install. This process ensures that the application has access to the resources it can not operate without. These permissions are also used to display appropriate Google app-store search results; for example, if a user's phone does not support a camera then all applications with a 'required' camera permission will not be displayed in the search results.

## 4.2 Server

This system implementation adopts a client-server model, where an online server is needed to tackle learning from new unclassified data. The server is implemented as a java servlet that runs on top of Apache Tomcat web server. A servlet is a java program that runs within a web server. Servlets receive and respond to requests from clients usually through the use of HTTP, the Hyper Text Transfer Protocol. The servlet aims to receive unclassified image data from clients and respond with new classification model data appropriately. Moreover, the servlet relies on manual input for the classification of new image data. The server implementation is minimal as it merely aims to demonstrate the utilization of an expanding classification model. Furthermore, the server incorporates JSON and Gson to transmit data objects consisting of attribute-value pairs between the server and client. This section is in term split into four sub-sections. The first of them describes the implementation environment of the server system. The second illustrates the implementation of the client-server communication protocol. Then the third sub-section describes the implementation of the re-training procedure. Finally, the fourth sub-section describes the utilization of manual input for labeling and proposes an alternative approach.

### 4.2.1 Implementation Environment

This subsection describes the server's implementation environment in an effort to facilitate the recreation of the same environment. In essence, the entire server implementation was developed on a virtual server running 64-bit Ubuntu 12.04. The

virtual server boasts one i7 core clocked at 3 GHz, 512 MB ram and a 20 GB SSD. The server is implemented as a Java servlet that is continuously running on top of Apache Tomcat web server (version 7). The server is implemented in native java with dependencies on LIBSVM, JSON, OpenCV, SQLite, and Gson. The use of native java facilitates the re-use of code written for the client implementation.

### 4.2.2 Communication Protocol

The communication between the client and server occurs over the web and is built on top of the HTTP post request. The post request is always initiated from a client, such that it uploads the client's recent unclassified object data to the server. Moreover, the server must respond to the post request either with an empty message that simply indicates the upload success or with new classification model data. The server's response is dependent on the client's current classification model version and the availability of newer classification models at the server. More to the point, Gson or Google Gson is used to serialize and de-serialize Java objects to and from JSON. Consequently, the data transferred between the client and server is in the form of Java objects. Although this approach can be entail extra bandwidth requirements, it facilitates the transfer of classification model data without any extra processing. As previously mentioned, this implementation is minimal and it merely meant to demonstrate the capabilities. A full-blown server implementation that tackles scalability would probably require the use of more efficient lower level protocols such as TCP or UDP and will also need to investigate more bandwidth friendly serialization techniques. More to the point, the final protocol utilizes a simple public-key to facilitate HTTP authentication; however, a more reliable implementation would consider further security measures that tackle common vulnerabilities such as DDoS attacks and SQL injections. The following figure illustrates the structure of the communication protocol.

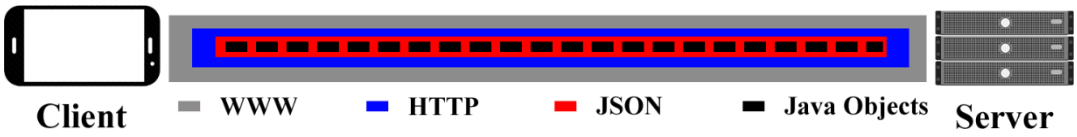



*Figure 4.16: Protocol Description Diagram*

### 4.2.3 Re-training Procedure

This sub-section describes the details of the re-training procedure implementation. Although both the client and server are written in java, some subtle differences in the implementation exist. To begin with, the server receives data from the clients in the form of OpenCV images, org.opencv.core.Mat objects, which describe unclassified cropped characters that some clients encountered. Therefore, the server implements that the exact same feature selection/extraction algorithm and SVM model. On the other hand, the server is not required to implement any of the segmentation component, the KNN model or the Seven-Segment model. First, the server receives segmented character information and is not required to perform segmentation at all. Also, the KNN model is essentially described by the raw training feature set, consequently no re-training required; whereas, the Seven-Segment model only operates on alphanumeric characters and therefore is not affected by the entire classification model update scheme (since it is designed solely for expanding the set of support symbols). Other differences in the implementation, include the utilization of a SQL database to store classification model information. The SQL database is initially created with one line that contains the first version of the training dataset embedded in the client application, and with every update (re-training) the server adds another line indicating the new version and the list of new training samples. Consequently, the use of SQL databases enables the server to easily select the set of training samples that are needed to update a client's model based on their current model's version. The SQL database does not store any specific classification model; the chosen approach relies on the client's system for re-training. Therefore, the server implements the SVM model only to test the functionality of new training samples before storing them. All in all, the server is built to receive images of unclassified objects and respond with new labeled training samples that are used to update or re-train the client's classification models. The following figure illustrates the server's response procedure in a sequence diagram.

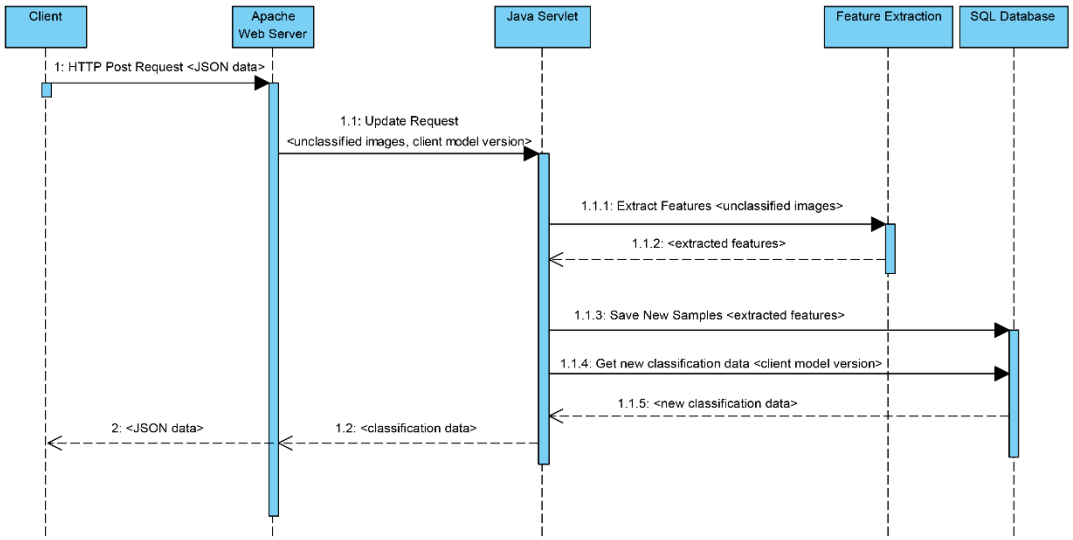



*Figure 4.17: Server Sequence Diagram*

### 4.2.4 Manual labeling Vs Image Search Engines

Finally, this sub-section describes how manual labeling is incorporated into the implementation and explores methods that would automate the process. The entire client-server model was adopted to enable the dynamic expansion of the set of symbols supported by the classification component. Consequently, as the server receives images of unclassified objects, it must be able to recognize whether any of these objects is a relevant symbol (in the context of household appliances) and must also be able to classify or label each of these symbols appropriately. Therefore, the final implementation of the server incorporates manual input from system administrators for labeling. Humans are inherently skilled at recognizing symbols and can easily rely on common search engines to understand the meaning behind them. Consequently, the server exploits human input to derive appropriate labels that describe previously unsupported symbols. Human input is performed by directly accessing the database and updating the relevant lines. Although the use of human input hinders the automation of this process, other alternatives are simply not feasible for an implementation in the scope of this project. The first alternative that comes to mind is the utilization of image search engines. Image search engines, commonly referred to as image meta search, are a genre of search engines specialized in finding similar pictures [48] [49]. These search engines exploit computer vision algorithms, very similar to the ones implemented in this project, which identify patterns across images; however, these engines take a much more generalistic approach. Even though, any search engine can be easily incorporated to

address automatic labeling, the majority of common image search engines are simply not well suited for symbol recognition. This a consequence of the generalistic approach that these engines assume, which can excel when the similarity between images are very little. Fortunately, image search engines that are tailor built for symbol and icon recognition exist [50] [51]; nevertheless, it is really hard to argue that a given image search engine supports all the symbols that might be encountered by the application and it can be even harder to implement a system that would automatically ensure that the search engine's labeling is appropriate for speech synthesis. For example, '.' this symbol can be labeled "dot" or "point", but it is clearly more appropriate for the speech synthesis component to say "fourteen point five three" rather than "fourteen dot five three" when reading "14.53".

# 5 Evaluation

This chapter contains details about the evaluation procedure for all components of the system and analysis of the results produced. In general, the evaluation is split into three parts: the continuous evaluation throughout the design and implementation phases, the independent evaluation of each final core recognition component, and a focus-group survey evaluation of the entire application. Consequently, this chapter is split into three sections, respectively.

## 5.1 Continuous Evaluation - Experiments

Throughout the development of the application, various experiments were arranged to evaluate certain design choices and facilitate the construction of the application. These experiments include the evaluation of City Block distance versus Euclidean distance for KNN classification implementations, the run-time analysis of KNN algorithm, the run-time evaluation of available OpenCV descriptors, the run-time evaluation of discrete transformations, the performance evaluations of proposed segmentation algorithms, the evaluation of native android speech recognition accuracy and finally the evaluation of feature selection and feature normalization. These experiments are scattered across this report; each experiment is presented in the relevant context either in the design or implementation chapter. Nevertheless, all experiments are included in the attached code and their organization is described in details in the Appendix A.

## 5.2 Independent Evaluation of Core Recognition Components

At its core, this report revolves around implementing an environment specific character recognition system. This system is partitioned into three core components that are executed in succession (the output of one component is the input of its successor). These components are segmentation, classification and the user interface respectively; such that the input for classification is the output of segmentation and so on. Each component was independently designed based on continuous experimentation and evaluation. Through the continuous and rigorous evaluation of the individual components of the application, the system promotes a flexible software development approach that relies on continuous evaluation to achieve continuous improvements. Given the serial approach in partitioning this system, it

is easier to perform evaluations on the system as a whole; where the segmentation component splits an image into sets of pixels that the classification component labels and the user interface attempts to derive appropriate meaning out of these labels. Nevertheless, this approach creates a clear dependence between the successive components; such that the classification accuracy is effected by the segmentation precision and so on. Consequently, the aim is to evaluate each component's performance independently. Independent evaluation can be achieved by defining a precise independent evaluation dataset for each component; for example the classification component is evaluated using 100% precise character segments produced via manual segmentation. More to the point, each evaluation dataset aims to describe all variations of the targeted environment in the context of each component. The segmentation evaluation dataset contains 36 1280x720 images ,sampled from hundreds of manually captured pictures, that describe the targeted environment across all varieties of glare, light reflections, cluttered backgrounds, and household appliances. Whereas, the classification evaluation dataset relies on manually segmented instances of the same 36 images.

### 5.2.1 Segmentation

This sub-section revolves around the performance evaluation of the final segmentation algorithm. The algorithm relies on a Canny edge detector, a connected component analysis and a RGB histogram in conjunction to perform segmentation. The algorithm's design and implementation are detailed in sub-section 3.1.4 and sub-section 4.1.2 respectively. In this sub-section, the performance of the algorithm is evaluated based on run-time requirements and accuracy results. Nevertheless, throughout the development of all proposed segmentation algorithms, each algorithm was initially assessed based on run-time requirements and accuracy results. These evaluations are presented in sub-section 3.1.6, where eleven unique segmentation algorithms were designed and evaluated (including the algorithm used in the final implementation). More to the point, all proposed segmentation algorithms must rely on some sort of color differentiation approach. The initial approach exploited the Euclidean distance metric to calculate the difference between colors, as illustrated in sub-section 3.1.1. Nevertheless, the use of the City Block distance metric should theoretically achieve much less demanding run-time requirements while achieving slightly less accurate results. Consequently, the

evaluation of this trade-off over all proposed segmentation algorithms is also presented in sub-section 3.1.6 in terms of the change in run-time and accuracy between utilizing Euclidean distance metric versus City Block distance metric. Finally, based on all aforementioned evaluation results, the algorithm with the most feasible combination of run-time requirements and accuracy results was chosen. As described in subsection 4.1.2, the final segmentation algorithm incorporates the City Block distance metric and resolution re-scaling in the edge detector in an effort to reduce the run-time requirements. Figures 5.1, 5.2, and 5.3 below describe the performance results of the final segmentation algorithm.

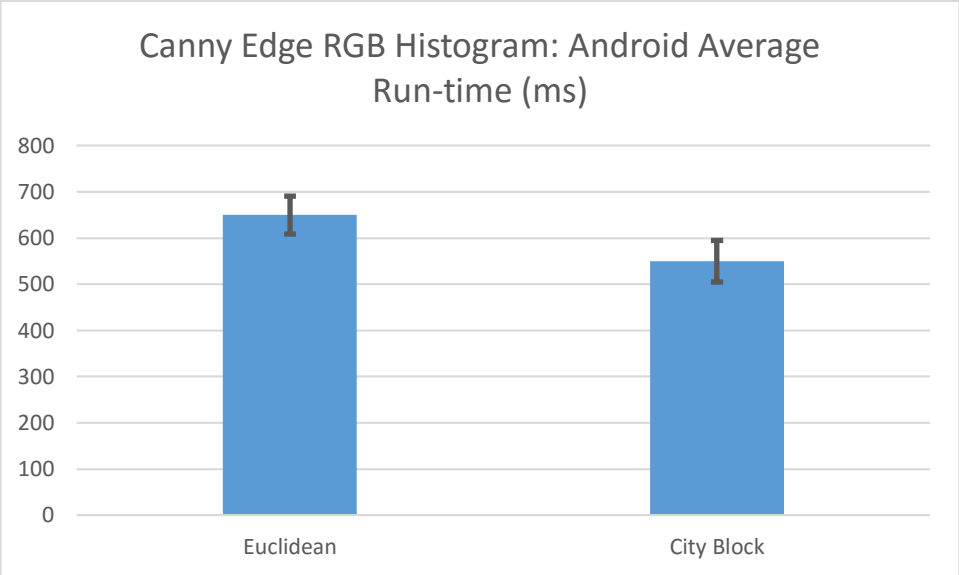



*Figure 5.1: Final Segmentation Component: Android Average Run-time in milliseconds*

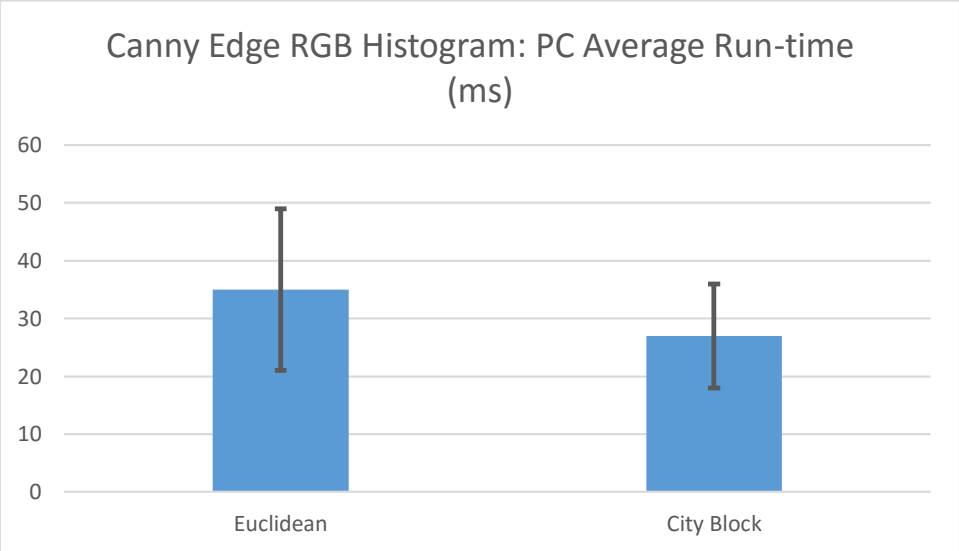



*Figure 5.2: Final Segmentation Component: PC Average Run-time in milliseconds*

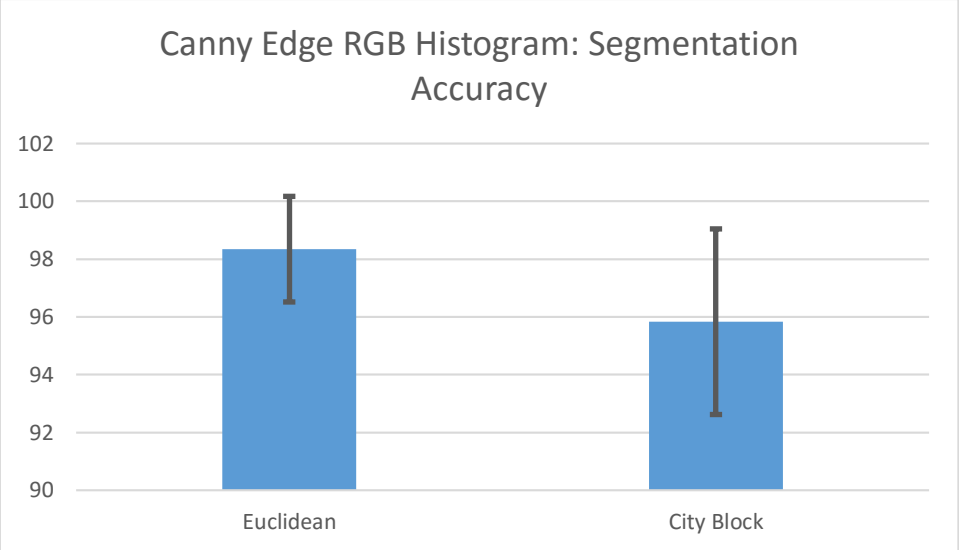



*Figure 5.3: Final Segmentation Component: Accuracy Evaluation*

These results reflect the final performance of the segmentation algorithm on the evaluation dataset, which is illustrated in appendix B. The final implementation includes the Canny edge detector's resolution optimization; for more details refer to sub-section 4.1.2. The complete experiment implementation and its results are available in the attached files, refer to Appendix A for more details.

This segmentation algorithm was intentionally built from scratch to accommodate the requirements of the targeted environment; tackling glare, light reflections, oversaturation and cluttered backgrounds. On the other hand, the majority of state-of-the-art character recognition systems employ segmentation algorithms built to tackle much simpler environments. These segmentation algorithms usually assume one constant color for the entire background and assume that no visual obstructions are present. Consequently, a direct comparison between the performance of the implemented algorithm and algorithms commonly used in character recognition is not very relevant. As a result, we evaluate the segmentation algorithm from the perspective of quasi-real-time smartphone implementation feasibility. This segmentation algorithm is capable of achieving real-time performance on a desktop computer; while, a smartphone real-time implementation is still beyond its capabilities. Nevertheless, at an average of 550 milliseconds and ~95% accuracy, the algorithm performs relatively well in a quasi-real-time smartphone implementation.

### 5.2.2 Classification

This sub-section describes the performance evaluation of the final classification component. The final implementation of the classification component consists of an ensemble of classifiers, including a Support Vector Machine classifier, a K-Nearest Neighbor classifier and a Seven-Segment classifier.

As previously mentioned, the classification ensemble is evaluated using a manually segmented set of images; the same 36 images used to evaluate the segmentation component, but manually segmented to remove the dependency on segmentation accuracy. Initially, each classifier is evaluated individually and independently on the previously mentioned evaluation dataset, then the entire ensemble is evaluated.

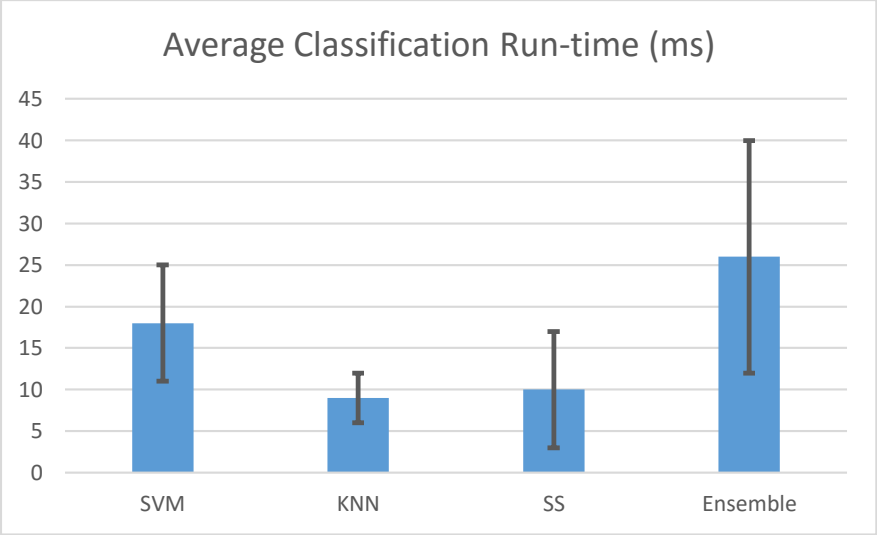



*Figure 5.4: Average Classification Run-time in milliseconds*

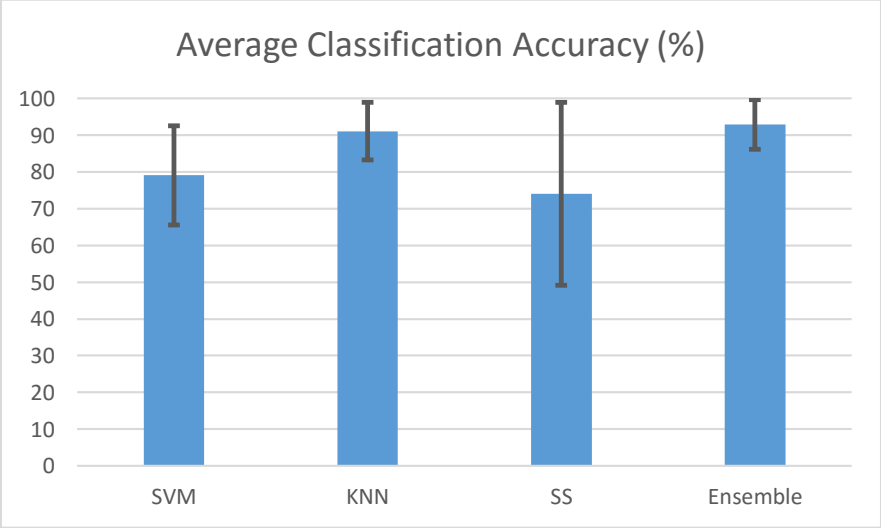



*Figure 5.5: Average Classification Accuracy*

Figures 5.4 and 5.5 describe the performance of the final classification ensemble. The results exhibit relatively high standard deviations, this is a consequence of the variation of the number of characters between different images in the evaluation dataset. Inherently, the classification component's run-time performance is directly proportional to the number of characters present on the targeted display. The dataset contains 36 images; the number of characters present in each image in the dataset ranges between three and thirty-six. The results demonstrate the classification ensemble's performance; achieving an average accuracy of ~92% in under 27 milliseconds. Current state-of-the-art classification systems achieve slightly more accurate results usually in the upper nineties, but entail longer execution time. Nevertheless, the classification component can be improved by investigating other feature sets that describe the targeted language more efficiently.

### 5.2.3 User Interface

This sub-section describes the performance evaluation of the speech recognition and contextual information analysis components incorporated in the final implementation of the user interface. In essence, the interface integrates speech synthesis, speech recognition and contextual information analysis to provide an interactive auditory interface. Both speech synthesis and recognition components rely on Android's native libraries; which integrates support for a wide-range of state-of-the-art speech processing algorithms. The performance evaluation of speech synthesis can be very challenging, as it is hard to identify a set of parameters that define the performance of speech synthesis. Generally, one desires speech synthesis to generate smooth, understandable, and consistent results. Therefore, we rely on the upcoming survey to evaluate the performance of the synthesized speech. On the other hand, it is relatively trivial to evaluate the performance of the speech recognition component; as we can easily identify recognition accuracy as the parameter that defines the performance of this component. Even though this implementation relies on state-of-the-art Android libraries, the speech recognition component extends the implementation by exploiting the static word "Eyeris" preceding every command. This extension is described and evaluated in sub-section 3.3.1. The following graph illustrates the general performance of the speech

recognition component and the performance gained by implementing the preceding extension.

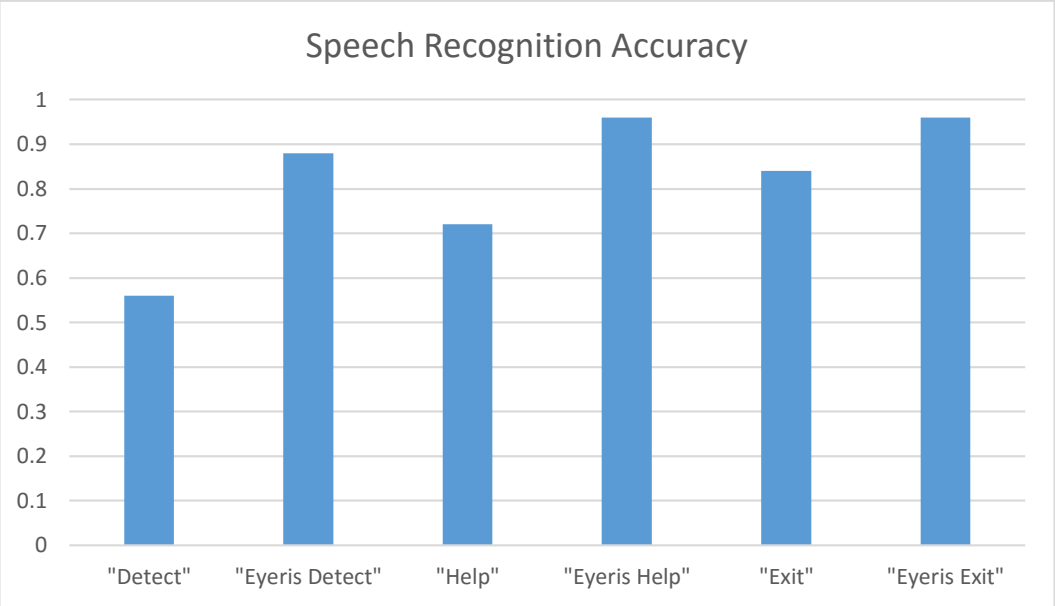



*Figure 5.6: Initial Speech Recognition Experiment*

These results show that the final speech recognition component is able to achieve ~93% average accuracy with the use of a static word and ~71% average accuracy without. These experiments were generated from 25 tests for each command; however, these experiments only consider one person's voice and do not consider other significant factors i.e. background noise, accent and distance from microphone. Consequently, another experiment is performed where three different people repeated the same experiment but with consistent background noise and distances from the microphone (at 30 cm). The results are illustrated in the table below.

| Command | "Detect" | "Help" | "Exit" |
|---|---|---|---|
| Accuracy | 56/75 | 56/75 | 65/75 |
| Command | "Eyeris Detect" | "Eyeris Help" | "Eyeris Exit" |
| Accuracy | 71/75 | 70/75 | 74/75 |

*Table 5.1: Final Speech Recognition Experiment Results*

The results are consistent with the previous experiment, as the average accuracy with the use of a static word is ~90% and ~70% without. The experiments demonstrated the capabilities of Android's native support for speech recognition and the performance optimizations gained from a simplistic extension.

Finally, an attempt at the evaluation of the performance of the contextual information analysis component. The evaluation of this component can also be quite challenging, this is a consequence of the component's dependence on accurate segmentation and classification. Essentially, this component is in charge of deriving meaning out of the classification results, by attempting to categorize words appropriately. The categorization of words relies on classification confidences, contextual information (highest-confidence label of neighboring characters) and the recognition history (as people tend to use the application on the same environment repeatedly). Initially, the idea behind the evaluation revolved around manually providing the input (as classification confidences) for the component and counting how many it correctly categorized the input. Unfortunately, this approach would eliminate the use of recognition history information and relies on manually chosen classification confidences (which have a significant influence on the results). Given the approach taken to execute this experiment, it is clear that the results will not accurately represent this component's performance on our targeted environment. Therefore, the final evaluation approach relied on the classification evaluation dataset; where the images are manually segmented to remove the dependence on the segmentation component. Consequently, this final evaluation exploits the classification component to produce results (classification confidences) that accurately represent the targeted environment which in term are used as input for the evaluation of this component. More to the point, to remove the dependence on classification accuracy, only 100% accurate classification results (ones where the highest classification confidences describe the actual character) were used to evaluate this component. Fortunately, the contextual information analysis component was able to correctly categorize all input. The contextual information analysis component is deterministic; such that it always produces the same results for the same input. Moreover, the input for this evaluation is in the form of classification confidences that were manually picked (from the results of the classification evaluation) to ensure that the classification accuracy has no influence on the evaluation. Consequently, this evaluation is hard to quantify into a table or a graph; nevertheless, this experiment is included in the attached code as an extension of the classification evaluation code; such that it can be easily repeated but still requires manual intervention.

## 5.3 Survey

This section describes the survey used to evaluate the performance of the application as a whole. Seventeen unique members participated in this survey across four different countries; the majority of the participants of this survey are computer science graduates. The survey was deployed on SurveyMonkey.com to facilitate data collection. The survey consists of six questions and is still viewable via the following link [52]. The following graphs illustrate the results for each question.

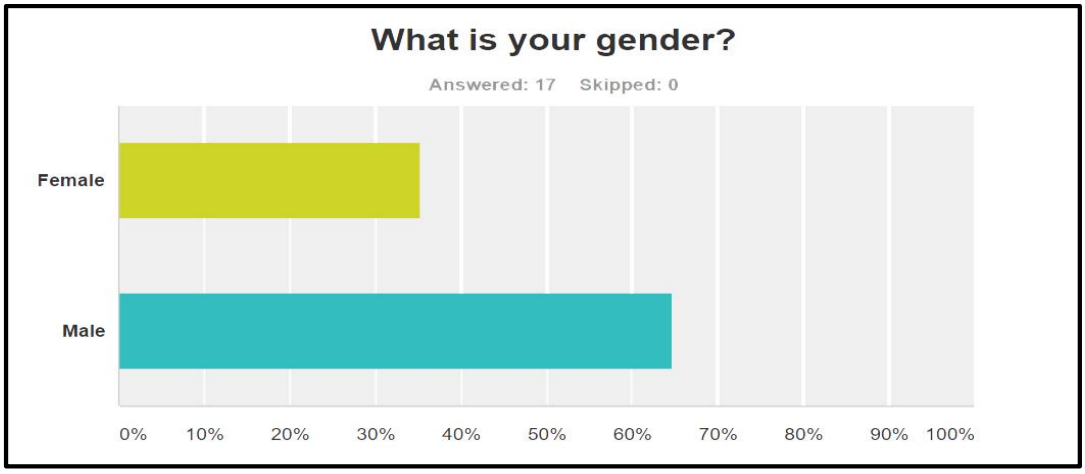



*Figure 5.7: Survey's First Question Results*

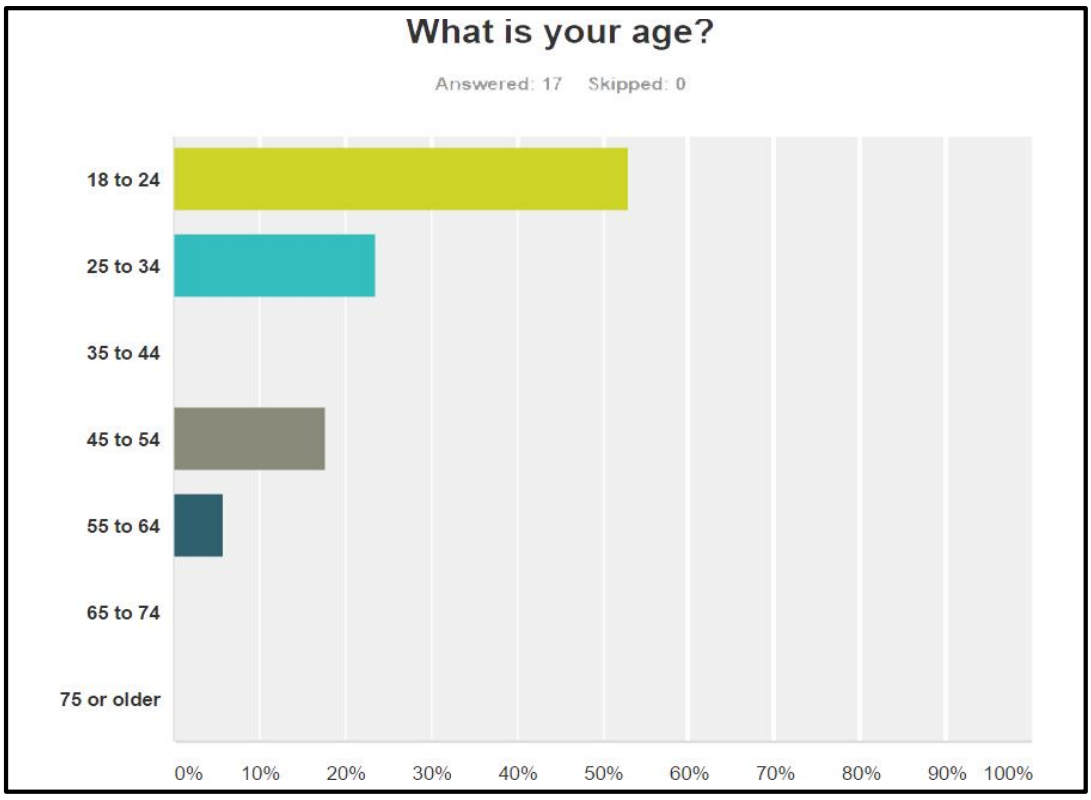



*Figure 5.8: Survey's Second Question Results*

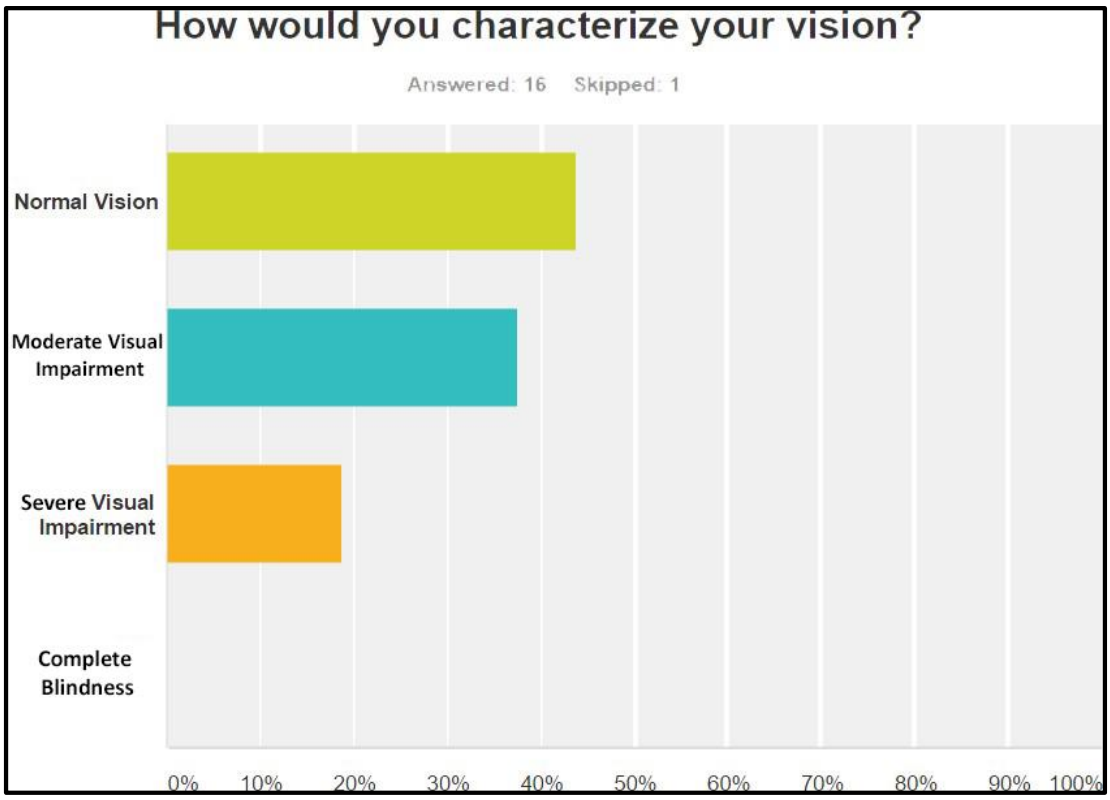



*Figure 5.9: Survey's Third Question Results*

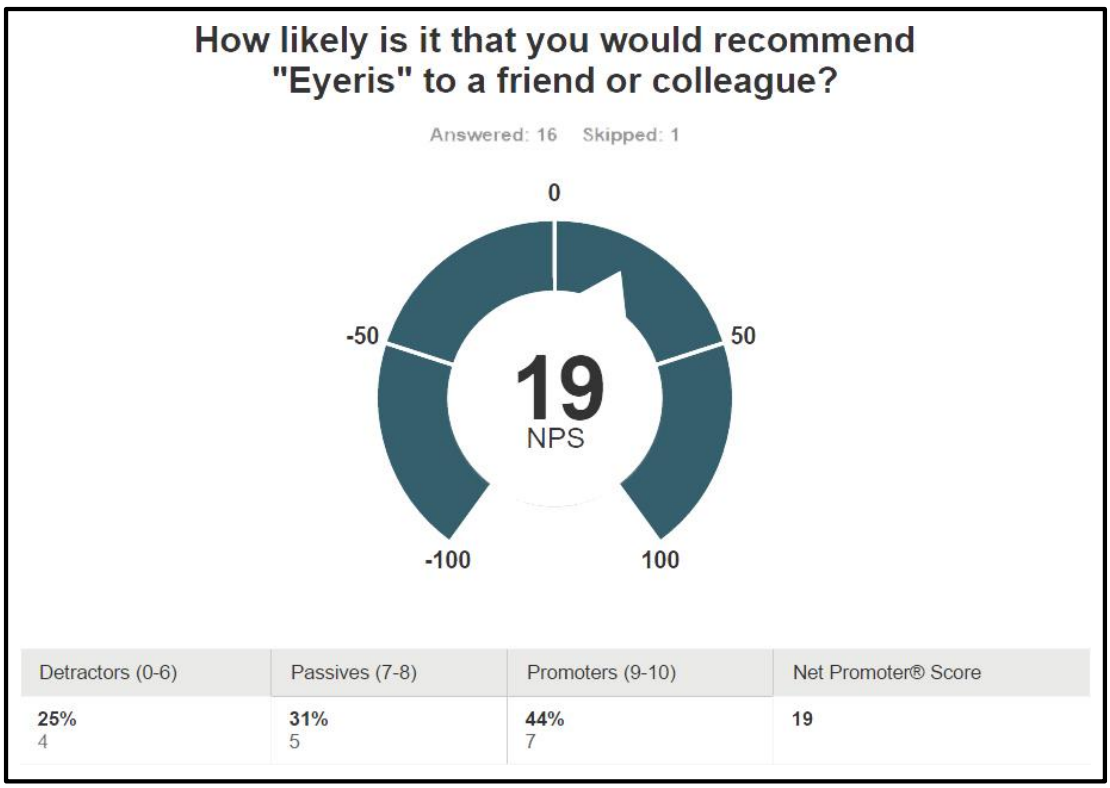



*Figure 5.10: Survey's Fourth Question Results*

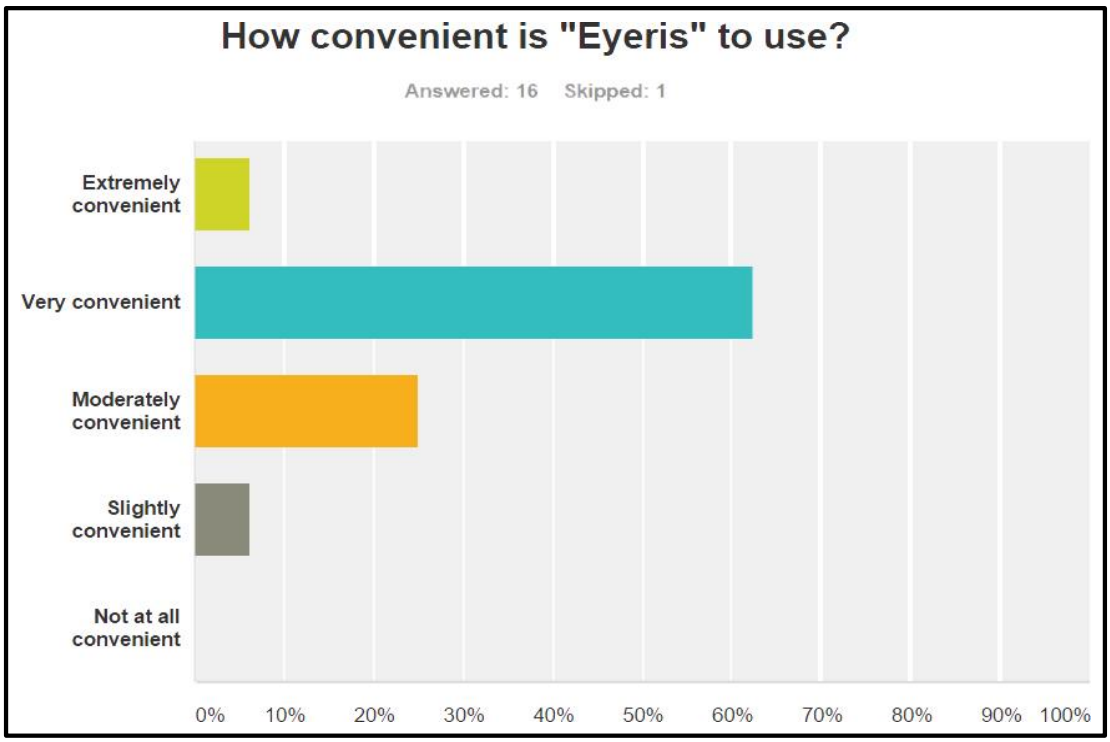
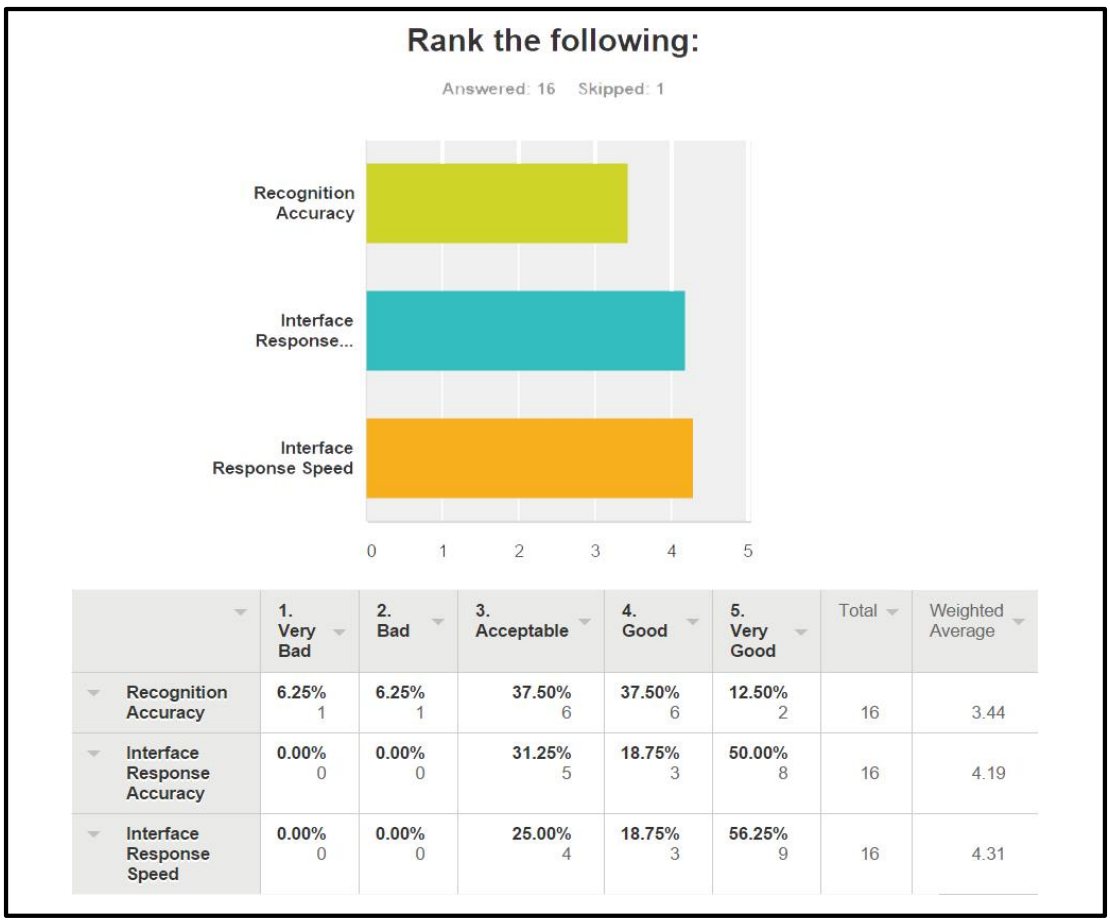



*Figure 5.11: Survey's Fifth Question Results*




| | 1. Very Bad | 2. Bad | 3. Acceptable | 4. Good | 5. Very Good | Total | Weighted Average |
|---|---|---|---|---|---|---|---|
| Recognition Accuracy | 6.25% 1 | 6.25% 1 | 37.50% 6 | 37.50% 6 | 12.50% 2 | 16 | 3.44 |
| Interface Response Accuracy | 0.00% 0 | 0.00% 0 | 31.25% 5 | 18.75% 3 | 50.00% 8 | 16 | 4.19 |
| Interface Response Speed | 0.00% 0 | 0.00% 0 | 25.00% 4 | 18.75% 3 | 56.25% 9 | 16 | 4.31 |

*Figure 5.12: Survey's Sixth and final Question Results*

To begin with, one must recognize the validity of this survey. The survey was taken by seventeen unique individuals, none of which are blind. Nevertheless, three unique individuals who suffer from severe visual impairment and six other who suffer from moderate visual impairment participated in this survey. This information speaks towards the validity and reliability of this survey as an evaluation metric. The survey concluded that the application provides a convenient interface that 44% of the population are willing to promote. On the other hand, 31% of the population took a passive stance on promoting or recommending the application to others and the remaining 25% took a negative stance. More to the point, the entire population felt that the interface response accuracy and response speed are acceptable or better; where 50% of them rated response accuracy 5 out of 5 or very good. On the other hand, the recognition accuracy demonstrates inconsistent results. The inconsistency stems from the system's inconsistency in tackling new environments; this could be further investigated by expanding the evaluation dataset. Essentially, some of the assumptions that this implementation is making (based on the targeted environment) do not hold over new environments.

# 6 Conclusion

This chapter concludes this report, presenting a brief overview of the project and suggestions about possible improvements. Therefore, this chapter is split into two sections: conclusions and future work.

## 6.1 Conclusions

Initially, this project started with the intention of developing a computer vision system that tackles the challenge of interpreting the contents of digital displays. This system aims to provide assistance for the visually-impaired to overcome the interaction barriers posed by the prevalence of digital displays in electronics and household appliances. Accordingly, this project was split into multiple objectives that contribute towards the main aim; these contributions are detailed in section 1.5. The identification of these contributions facilitated the shaping of the initial project structure; which in term revealed the relevant fields of study.

The background study mainly revolved around the field of computer vision and its role in assisting the disabled community. The study included areas such as segmentation, classification, feature selection and connected component analysis in the context of printed or digital character recognition; rather than the more common context of hand-written character recognition. Some of these areas were further investigated into more specific subjects such as seven-segment displays, color schemes, histograms, blob detection, edge detection, support vector machines, k-nearest neighbor classifiers, artificial neural networks, decision trees, feature descriptors, dimensionality reduction and discrete transformations. The background study was needed to facilitate the fulfillment of the project's objectives as it presents the relevant tools needed for implementation. Through the realization of this background study, an insight of the available techniques, tools and algorithms required for this project has been understood.

The largest amount of effort was put into the design phase, specifically in designing a segmentation component that tackles the visual obstructions commonly imposed by digital displays. Throughout the design phase, various technical observations were made about the specific nature of this project that led to the development of eleven unique segmentation algorithms. More to the point, the design phase included

a variety of experiments that continuously evaluated the development stages of various components, including segmentation, classification and the user interface. These experiments aimed to facilitate the agile development of these components where continuous evaluations led to continuous improvements. Fundamentally, the design phase incorporated available elements researched in the background study in various unique structures to design a comprehensive set of approaches for the implementation of each component. In term, all developed approaches for a given component were constantly optimized and evaluated to facilitate the final implementation.

A lot of effort was also put in the implementation phase, which revolves around the final implementation of the core system. The core system adopts a client-server model, where the client system is implemented as an Android application and the server system is implemented as Java servlet. The adopted model defines the structure of the final implementation and the distribution of the system. More to the point, this phase comprehensively describes the implementation of the client as an Android application, incorporating details about the implementation environment, third-party library dependence, technical observations, and language specific pseudocode. On the other hand, this phase describes the final implementation of the web server as a Java servlet running on top of Apache Tomcat, incorporating details about the implementation environment, third-party library dependence, communication protocol development, differences and similarities with the client implementation and the utilization of SQL databases. The implementation chapter aims to provide all the details required to recreate this project and facilitate the understanding of the core algorithms implemented in this project.

The final chapter of this project revolves around the evaluation of core system components. A significant amount of effort was spent on the evaluation of this system, providing a clear insight into the efficiency and effectiveness of the implemented system. The evaluation process was continuous throughout the design and implementation stages of development, where various experiments were undertook to evaluate certain trade-offs and facilitate the development decision making process. Consequently, this chapter mainly describes the evaluation procedure of the final implementations of core recognition components including segmentation, classification and speech recognition. Each core recognition

component was independently evaluated on pre-defined evaluation datasets that describe the targeted environment. The evaluation results were presented and analyzed in the context of each component. Moreover, this chapter presented the results of a focus-group survey that evaluated the overall performance of the application. The survey focused on assessing the convenience of using the implemented auditory user interface and overall functionality of the character recognition system. Based on the results, the application proved to be effectively convenient to use; however, inconsistent results regarding the character recognition accuracy indicate inconsistency in the system's ability to cope with different backgrounds and different segment encoding schemes (fourteen-segment and sixteen segment displays).

This project proved that current smartphones are capable of handling heavy computer vision tasks in a quasi-real-time fashion; even though most computer-vision libraries that support smartphone implementations are still in their infancy. More to the point, the project successfully demonstrated that image segmentation can cope with the various visual obstructions posed by the inherent nature of digital displays; even in a smartphone implementation where computational resources are relatively scarce. The project also showed that an intelligent contextual information analysis framework can offer several improvements over straight-forward classification. And finally the project demonstrated the use of an expanding classification model through the use of an online server. Nevertheless, given the level of abstraction that the targeted environment assumes, the system was bound to encounter environments that are inconsistent with the assumptions made. All in all, this project proves that accurate computer vision implementations are feasible given the resources that current smartphones offer; however, the project evaluation indicates that a more comprehensive training dataset that describes the targeted environment more accurately is key.

## 6.2 Future Work

This section examines various areas in this project that can be improved, and describes additional features that can complement the current implementation. Two unique improvements are proposed.

To begin with, the concept of completely automating the classification model update procedure is further explored. As previously discussed, the current implementation seeks manual input to label recently encountered unknown symbols to enable the continuous expansion of the classification model. The use of image search engines has already been proposed to replace the dependence on manual labeling, enabling a fully autonomous approach. Unfortunately, various limitations to this approach exist. Essentially, the majority of available image search engines employ a generalistic approach to comparing images; this generalistic approach usually yields accurate results when images exhibit similar patterns and have little to no difference and is simply not well-suited for symbol matching (where variations in orientation, scale and font structure are common). Moreover, another significant limitation to the complete automation of classification update approach is the ability to determine whether an image search engine, or whatever component is responsible for labeling unknown symbols, is producing suitable labels. Therefore, a compromise is proposed; where a symbol-specific image search engine would be in charge of labeling the entire set of freshly encountered symbols; however, the resulting labels will seek manual input to approve the suitability of the labels. In this approach, the system administrator responsible for manual intervention will be required to approve or modify the resultant labels rather than labeling them himself. This approach compromises complete automaticity but also reduces the amount of manual effort needed to maintain the current implementation.

Throughout the evaluation stage, the survey revealed that the implemented system copes horribly with fourteen and sixteen-segment displays. The implementation of custom classifiers that would tackle fourteen and sixteen-segment formats is proposed. Similar to the seven-segment classifier, these classifiers can exploit pixel density information in pre-defined regions to tackle classification. These classifiers can be constructed in the exact same approach taken for seven-segment classification, which relies on pre-defined regions and hardcoded state information to accurately and efficiently address classification. This proposal will ensure that the system is more capable of handling different and less common environments. Nevertheless, these extra classifiers will inherently impose further computational requirements.

# References


[1] Khan, Sabena, and Emily Wilkes. *Energy Consumption In The UK (2014)*. 1st ed. United Kingdom: The Department of Energy & Climate Change (DECC), 2014. Web. 27 Apr. 2015.

[2] http://www.knfbreader.com/pdf/KNFBReader_manual-EN.pdf, User Guide For KNFB Reader. 1st ed. K-NFB Reading Technology, INC, 2015. Web. 5 Mar. 2015.

[3] http://www.who.int, 'WHO | Visual Impairment and Blindness'. N.p., 2015. Web. 25 Apr. 2015: International Classification of Diseases (Update and Revision 2006)

[4] *Future Sight Loss UK (1): The Economic Impact of Partial Sight and Blindness in the UK Adult Population*. 1st ed. United Kingdom: Access Economics Pty Limited and RNIB, 2015. Web. 1 May 2015.

[5] Bushara, Khalafalla O., et al. "Modality-specific frontal and parietal areas for auditory and visual spatial localization in humans." *Nature neuroscience* 2.8 (1999): 759-766.

[6] Doricchi, Fabrizio, et al. "White matter (dis) connections and gray matter (dys) functions in visual neglect: gaining insights into the brain networks of spatial awareness." *Cortex* 44.8 (2008): 983-995.

[7] Karnath, Hans-Otto, Susanne Ferber, and Marc Himmelbach. "Spatial awareness is a function of the temporal not the posterior parietal lobe." *Nature*411.6840 (2001): 950-953.

[8] Marynel Vázquez, Aaron Steinfeld, Helping visually impaired users properly aim a camera, Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility, October 22-24, 2012, Boulder, Colorado, USA

[9] Torgesen, Joseph K. "Using computers to help learning disabled children practice reading: A research-based perspective." *Learning Disabilities Focus*(1986).

[10] http://www.Adaptivetr.com, 'Reading Device, Reading Machine, Visually Impaired, Blindness'. N.p., 2015. Web. 30 Apr. 2015.

[11] Disabled World, 'Assistive Technology: Devices Products & Information'. N.p., 2015. Web. 1 May 2015.

[12] Mori, Shunji, Ching Y. Suen, and Kazuhiko Yamamoto. "Historical review of OCR research and development." *Proceedings of the IEEE* 80.7 (1992): 1029-1058.

[13] Breuel, Thomas M. "The OCRopus open source OCR system." *Electronic Imaging 2008*. International Society for Optics and Photonics, 2008.

[14] Rajkumar, N., M. G. Anand, and N. Barathiraja. "Portable Camera-Based Product Label Reading For Blind People."

[15] Ghugardare, Rakhi P., et al. "Optical character recognition system for seven segment display images of measuring instruments." *TENCON 2009-2009 IEEE Region 10 Conference*. IEEE, 2009.

[16] http://www.knfbreader.com, 'KNFB Reader iOS'. N.p., 2015. Web. 1 May 2015.

[17] Blind Sight, 'Text Detective'. N.p., 2015. Web. 1 May 2015.

[18] http://www.creaceed.com, 'Prizmo: Scanning & OCR — About'. N.p., 2015. Web. 1 May 2015.

[19] Meskaldji, Khouloud, Samia Boucherkha, and Salim Chikhi. "Color quantization and its impact on color histogram based image retrieval." (2009).

[20] Ming, Anlong, and Huadong Ma. "A blob detector in color images." *Proceedings of the 6th ACM international conference on Image and video retrieval*. ACM, 2007.

[21] Hinz, Stefan. "Fast and subpixel precise blob detection and attribution." *Image Processing, 2005. ICIP 2005. IEEE International Conference on*. Vol. 3. IEEE, 2005.

[22] Kamavisdar, Pooja, Sonam Saluja, and Sonu Agrawal. "A survey on image classification approaches and techniques." *International Journal of Advanced Research in Computer and Communication Engineering* 2.1 (2007).

[23] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." *Machine learning* 20.3 (1995): 273-297.

[24] Bay, Herbert et al. 'Speeded-Up Robust Features (SURF)'. *Computer Vision and Image Understanding* 110.3 (2008): 346-359. Web.

[25] Goodall, Colin, and I. T. Jolliffe. 'Principal Component Analysis'. *Technometrics* 30.3 (1988): 351. Web.

[26] Khayam, Syed Ali. "The discrete cosine transform (DCT): theory and application." *Michigan State University* (2003).

[27] Aradhya, VN Manjunath, G. Hemantha Kumar, and S. Noushath. "Multilingual OCR system for South Indian scripts and English documents: An approach based on Fourier transform and principal component analysis." *Engineering Applications of Artificial Intelligence* 21.4 (2008): 658-668.

[28] Jawahar, C. V., MNSSK Pavan Kumar, and SS Ravi Kiran. "A bilingual OCR for Hindi-Telugu documents and its applications." *null*. IEEE, 2003.

[29] Ismail¹, SAAD M., and S. N. H. S. Abdullah. "Geometrical-Matrix Feature Extraction for on-Line Handwritten Characters Recognition." *Journal of Theoretical and Applied Information Technology* 49.1 (2013).

[30] He, Lifeng, et al. "Fast connected-component labeling." *Pattern Recognition*42.9 (2009): 1977-1987.

[31] Wu, Kesheng, Ekow Otoo, and Kenji Suzuki. "Optimizing two-pass connected-component labeling algorithms." *Pattern Analysis and Applications* 12.2 (2009): 117-135.

[32] Bhoyar, Kishore, and Omprakash Kakde. "Color image segmentation based on JND color histogram." *International Journal of Image Processing (IJIP)* 3.6 (2010): 283.

[33] Torres, Fernando, Jesús Angulo, and Francisco Ortiz. "Automatic detection of specular reflectance in colour images using the MS diagram." *Computer Analysis of Images and Patterns*. Springer Berlin Heidelberg, 2003.

[34] Deng, G., and L. W. Cahill. "An adaptive Gaussian filter for noise reduction and edge detection." *Nuclear Science Symposium and Medical Imaging Conference, 1993. IEEE Conference Record*. IEEE, 1993.

[35] Wang, Liang, and Hehua Ju. "A robust blob detection and delineation method. *"Education Technology and Training, 2008 International Workshop on Geoscience and Remote Sensing. ETT and GRS 2008. International Workshop on*. Vol. 1. IEEE, 2008.

[36] Rokach, Lior. "Ensemble-based classifiers." *Artificial Intelligence Review* 33.1-2 (2010): 1-39.

[37] Platt, John. "Fast training of support vector machines using sequential minimal optimization." Advances in kernel methods—support vector learning 3 (1999).

[38] http://www.android-developers.blogspot.co.uk, 'An Introduction To Text-To-Speech In Android | Android Developers Blog'. N.p., 2009. Web. 2 May 2015.

[39] Huang, Xuedong, and Kai-Fu Lee. "On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition." Speech and audio processing, IEEE transactions on 1.2 (1993): 150-157.

[40] http://developer.android.com/guide/topics/manifest/uses-sdk-element.html, 'Android Developers'. N.p., 2015. Web. 1 Sept. 2015.

[41] http://www.techcrunch.com/2014/06/25/google-now-has-1b-active-android-users/, Lardinois, Frederic. 'Google Now Has 1B Active Monthly Android Users'. *TechCrunch*. N.p., 2015. Web. 1 Sept. 2015.

[42] http://www.idc.com, 'IDC: Smartphone OS Market Share'. N.p., 2015. Web. 25 Aug. 2015

[43] https://developer.android.com/sdk/installing/create-project.html, 'Managing Projects From Android Studio | Android Developers'. N.p., 2015. Web. 13 Aug. 2015.

[44] Aksoy, Selim, and Robert M. Haralick. "Feature normalization and likelihood-based similarity measures for image retrieval." Pattern recognition letters 22.5 (2001): 563-582.

[45] http://www.google.com/fonts, 'Google Fonts'. N.p., 2015. Web. 2 Sept. 2015.

[46] Cooper, James W. "The design patterns Java companion." (1998).

[47] https://developer.android.com/preview/features/runtime-permissions.html, ‘Permissions | Android Developers'. N.p., 2015. Web. 1 Sept. 2015.

[48] Bach, Jeffrey R., et al. "Virage image search engine: an open framework for image management." Electronic Imaging: Science & Technology. International Society for Optics and Photonics, 1996.

[49] Luo, Bo, Xiaogang Wang, and Xiaoou Tang. "World Wide Web based image search engine using text and image content features." Electronic Imaging 2003. International Society for Optics and Photonics, 2003.

[50] http://detexify.kirelabs.org, 'Detexify Latex Handwritten Symbol Recognition'. N.p., 2015. Web. 2 Sept. 2015.

[51] Milde, B. 'Shapecatcher: Draw The Unicode Character You Want!'. Shapecatcher.com. N.p., 2015. Web. 3 Sept. 2015.

[52] https://www.surveymonkey.com/r/23RRDKL, 'Eyeris Evaluation Survey'. N.p., 2015. Web. 3 Sept. 2015.

# Appendix A: Experiments and Attached Files

All experiments, tests and evaluations are included in the attached files, alongside the final application implementation as an Android Studio project and the final server implementation as a Java program. Some experiments were implemented on Native Java, as most of them evaluated accuracy or relative run-time (elements unaffected by computational resources available). On the other hand, other experiments were implemented on Android, these experiments focused on evaluating elements that are directly affected by the resources available on a smartphone. Consequently, the experiments are split into two sections, java experiments and android experiments. Both sections include a GUI that facilitates the execution of the experiments. This appendix describes how the attached files are organized and describes how to execute all implementations.

To Execute the Native Java Experiments GUI run Experiments/Native-Java/run.bat to compile and run the GUI (this requires a windows environment to run batch files, also the bundled OpenCV library is compiled for windows and will not work on Linux or Unix); alternatively, compile and run the GUI with the following commands:

javac -cp opencv-2410.jar;libsvm.jar; *.java

java -cp opencv-2410.jar;libsvm.jar;. GUI

The native Java Experiments require Java JDK v8 and windows to run.

To execute the Android Experiments you must import “Eyeris-Android-Evaluation” folder as a project into Android Studio and either run in it on simulator or build it for testing on a smartphone. The experiment results can be viewed directly on the screen or via the debugger printout. Moreover, you can run the attached final application implementation similarly by importing the “Eyeris” folder instead. On the other hand,

the final server implementation will require a Tomcat Web Server environment to run under. Nevertheless, by simply importing the "Final Server Implementation" folder into Tomcat web server and adjusting the relevant parameters (including the server's settings and the SQL database access information) the server's functionality can be tested. Moreover, for the evaluation of the final segmentation and classification components on android, execute the attached "Eyeris" project in an Android Studio simulator or on a smartphone and use the debugger to collect information.

The attached files are available at:

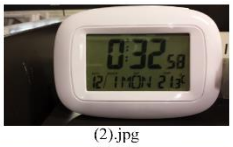
https://www.dropbox.com/s/z18mid16e3twu3l/Final_Eyeris_Attachment.zip?dl=0

The attached files are organized as follows:

1. **Experiments**
    - **1.1 Native-Java** (Java Program)
        - **1.1.1 All Implemented Segmentation Algorithms Evaluation**
        - **1.1.2 Final Segmentation Algorithm Evaluation**
        - **1.1.3 Feature Sets Evaluation**
        - **1.1.4 Feature Sets Evaluation with normalization**
        - **1.1.5 SVM Classifier Evaluation**
        - **1.1.6 KNN Classifier Evaluation**
        - **1.1.7 SS Classifier Evaluation**
        - **1.1.8 Entire Classification Ensemble Evaluation**
        - **1.1.9 KNN City Block Versus Euclidean**
        - **1.1.10 KNN Run-time Analysis**
    - **1.2 Eyeris-Android-Evaluation** (Android Studio Project)
        - **1.2.1 OpenCV Descriptors Evaluation**
        - **1.2.2 PCA, DFT, DCT Evaluation**
2. **Eyeris** (Android Studio Project)
3. **Final Server Implementation** (Java Program)
4. **All-Results** (Excel Sheets)

# Appendix B: Evaluation Image Dataset

(1).jpg



(2).jpg

(3).jpg

(4).jpg

(5).jpg

(6).jpg

(7).jpg

(8).jpg

(9).jpg

(10).jpg

(11).jpg

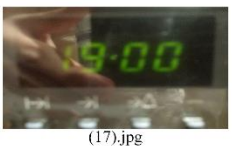
(12).jpg

(13).jpg

(14).jpg

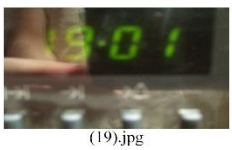
(15).jpg

(16).jpg



(17).jpg

(18).jpg



(19).jpg

(20).jpg

(21).jpg

(22).jpg

(23).jpg

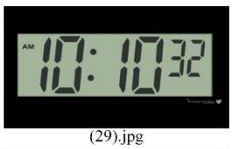
(24).jpg

(25).jpg

(26).jpg

(27).jpg

(28).jpg



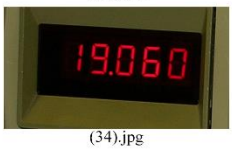
(29).jpg

(30).jpg

(31).jpg

(32).jpg

(33).jpg



(34).jpg

(35).jpg

(36).jpg

# Appendix C: OpenCV – Android Linking/Binding

Essentially two ways exist to linking the OpenCV library to an android project: static binding or dynamic linking. Dynamic linking requires the installation of a separate application "OpenCV Manager" that includes all library files, this approach enables any application to use OpenCV without binding the actual library files in the application APK but by simply linking it to the OpenCV Manager. On the other hand, static binding involves incorporating the required library files into the application itself; this approach does not require the separate installation of "OpenCV Manager" but results in a significant APK size change. The steps required to implement both approaches are described below, in the context of Android Studio.

**Static Binding:**

1. Decide which parts of the library are required in the application and place them in one folder.
2. Import that folder as a module into Android Studio and add as dependency.
   2.1 Select file tab → select new module → select import existing project → select the source directory for the folder created in step 1.
   2.2 Under project section right click the application → select open module settings → select dependencies tab → select + button → select module dependency → select OpenCV from list → select apply
3. Update gradle build files under OpenCV module to match your project
   3.1 Update compileSdkVersion to match project.
   3.2 Update buildToolsVersion to match project.
   3.3 Update minSdkVersion to match project.
   3.4 Update targetSdkVersion to match project.
4. Copy libs folder under sdk/native to Android Studio under app/src/main
5. Rename libs directory to jniLibs.
6. Be sure to use "System.loadLibrary("opencv_java");" to load OpenCV.

**Dynamic Linking:**

1. Be sure to use "System.loadLibrary("opencv_java");" to load OpenCV.

2. If "OpenCV Manager" is not installed on the smartphone, the application will automatically prompt a toast that asks the user to install the application separately.

# Appendix D: Histogram Implementation

```
Histogram()
{
    1. Initialize two data structures table A
       and table B, as illustrated below.

    2. Read the next pixel color vector in
       scan line order.

    3. Compare the new pixel color vector with
       all previous entries in table A.

        3.1 if the new color vector matches any
            entry, then accommodate it in the
            respective bin by updating table B;
            entering the current index,current
            row and column values ad go to step 4.

        3.2 if the new color vector does not match
            any entry, then increment the row index
            of table A and enter the new color vector
            in it,set population (column H in table A)
            to 1 and set the current index,current row
            and column as a new entry in table B and
            go to step 2.

    4. Repeat steps 2 and 3 for all pixels in the image.
}
```

| Color Index | R | G | B | H |
|---|---|---|---|---|
| 1 | 5 | 15 | 20 | 335 |
| 2 | 10 | 100 | 20 | 450 |
| 3 | 20 | 50 | 10 | 470 |
| . | . | . | . | . |
| K | 25 | 72 | 90 | 200 |

Table A : Color population

| X | Y | JND Color Index for $(x_i,y_i)$ from Table 1 |
|---|---|---|
| $x_1$ | $y_1$ | 1 |
| $x_1$ | $y_2$ | 1 |
| . | . | . |
| $x_i$ | $y_i$ | K |
| . | . | . |

Table B : Color index-pixel location relation

# Appendix E: Categorization Format Assumptions

1. Time Format Assumptions:
    a. Must contain at least one digit.
    b. Can only contain “AM” or “PM” alphabetic characters.
    c. Hours value must be between 0 – 23
    d. Minutes and Seconds values must be between 0 – 59.
2. Date Format Assumptions:
    a. Must contain at least four digits.
    b. Cannot contain any alphabetic characters.
    c. Years can be two digits or four digits.
    d. Months value must be between 1 – 12
    e. Days Value must be between 1 – 31
3. Temperature Format Assumptions:
    a. Must contain at least one digit and one alphabetic character.
    b. Alphabetic Characters must be either “f”, “c” or “k” for Fahrenheit, Celsius and Kelvin respectively.
4. Number Format Assumptions:
    a. Must contain at least one digit.
    b. Cannot contain any alphabetic characters.
    c. Can only contain the “.” symbol.
5. Text Format Assumptions:
    a. Can contain anything.

# Appendix F: Contextual Information Analysis Illustration

## Input


## Classification Results

| Actual Character | 1st Option | 1st Confidence | 2nd Option | 2nd Confidence | 3rd Option | 3rd Confidence |
|---|---|---|---|---|---|---|
| "1" | "1" | 0.54 | "i" | 0.33 | "L" | 0.13 |
| "7" | "7" | 0.76 | "9" | 0.16 | "Z" | 0.08 |
| "." | "." | 1.00 | N/A | N/A | N/A | N/A |
| "0" | "0" | 0.57 | "O" | 0.41 | "C" | 0.02 |
| "9" | "9" | 0.61 | "8" | 0.25 | "q" | 0.14 |
| "6" | "6" | 0.65 | "G" | 0.3 | "5" | 0.05 |
| | | | | | | |
| "1" | "1" | 0.49 | "i" | 0.45 | "L" | 0.06 |
| "9" | "9" | 0.55 | "8" | 0.34 | "q" | 0.11 |
| "." | "." | 1.00 | N/A | N/A | N/A | N/A |
| "7" | "7" | 0.72 | "Z" | 0.21 | "9" | 0.07 |
| "1" | "i" | 0.47 | "1" | 0.41 | "L" | 0.12 |
| "2" | "Z" | 0.46 | "2" | 0.45 | "7" | 0.09 |

## Initial Categorization

| Word | Category | Chosen Characters' Confidences | Word Confidence |
|---|---|---|---|
| "17.096" | "Number" | {0.54, 0.76, 1.00, 0.57, 0.61, 0.65} | 0.688 |
| "19.7iZ" | "Text" | {0.49, 0.55, 1.00, 0.72, 0.47, 0.46} | 0.615 |

## Possible Change

| Word | Category | Chosen Characters' Confidences | Word Confidence | Δ Confidence |
|---|---|---|---|---|
| "19.712" | "Number" | {0.49, 0.55, 1.00, 0.72, 0.41, 0.45} | 0.603 | 0.012 |

## Final Categorization

| Word | Category | Chosen Characters' Confidences | Word Confidence |
|---|---|---|---|
| "17.096" | "Number" | {0.54, 0.76, 1.00, 0.57, 0.61, 0.65} | 0.688 |
| "19.712" | "Number" | {0.49, 0.55, 1.00, 0.72, 0.41, 0.45} | 0.603 |