ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

FEATURE EXTRACTION AND CLASSIFICATION SCHEMES FOR ENHANCING AMHARIC BRAILLE RECOGNITION SYSTEM

SHUMET TADESSE

JUNE 2011

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

FEATURE EXTRACTION AND CLASSIFICATION SCHEMES FOR ENHANCING AMHARIC BRAILLE RECOGNITION SYSTEM

A Thesis Submitted to the School of Graduate Studies of Addis Ababa University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Science

By

SHUMET TADESSE

JUNE 2011

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION SCIENCE

FEATURE EXTRACTION AND CLASSIFICATION SCHEMES FOR ENHANCING AMHARIC BRAILLE RECOGNITION SYSTEM

By

SHUMET TADESSE

Name and signature of Members of the Examining Board

Name	**Title**	**Signature**	**Date**
Ato. Henock	Chairperson	_____	_____
Dr. Million Meshesha	Advisor,	_____	_____
Dr. Dereje Teferi	Examiner,	_____	_____

Dedicated to my mother

ACKNOWLEDGEMENT

First and foremost, I am ever grateful to the omnipresent God who made things possible to accomplish the two-year program at AAU. I am also heartily thankful to my advisor, Dr. Million Meshesha, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. His sage advice, insightful criticisms, and patient encouragement aided the writing of this thesis in innumerable ways. Besides his expertise, I really appreciate his patience and thank him for his concern and perspective advice through my thesis work as well as course of Master's program.

My greatest gratitude also goes to Mekelle University (MU) for granting me study leave with the necessary benefits, without which I could not have been able to join my M.Sc. study here in AAU. I also extend my sincere thanks to academic and administrative staffs of Department of Computer Science (MU).

My utmost gratitude is extended to my friends especially to Adane L., Belete B., Dereje Y., Esubalew K., Fentaw F., Kindie A., Minillik H., Tariku A., Tadele A. and Zewedie M. whose friendship, hospitality, knowledge, and valuable assistance have supported me in one way or another in the preparation and completion of this study.

I would like to thank AAU and all School of Information Science community (instructors, students and administrative staffs) who have a significant contribution in one way or another during my research work.

Last but not least, my sincere thanks also forwarded to my family. I am thankful to my mother, whom I dedicate my work, her love and best wishes, despite of her physical absence with me, have helped me in the successful completion of my study at AAU. It is a pleasure to express my gratitude wholeheartedly to my sisters and brothers, Aynalem, Birtukan, Fasil and Mekonnen for their inseparable support and care.

Shumet Tadesse

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

AAU	Addis Ababa University
ANN	Artificial Neural Network
AOBR	Amharic Optical Braille Recognizer
ARFF	Attribute Relation File Format
ASCII	American Standard Code for Information Interchange
DTC	Decision Tree Classifier
OBR	Optical Braille recognition
OCR	Optical Character Recognition
SMO	Sequential Minimal Optimization
SVM	Support Vector Machine
UNESCO	United Nations Educational, Scientific and Cultural Organization
WEKA	Waikato Environment for Knowledge Analysis
WHO	World Health Organization

ABSTRACT

Information in written form plays an undeniably important role in our daily lives. Recording and using information encoded in symbolic form is essential. Visually impaired people face a distinct disadvantage in this respect. To address their information need, the most widely adopted writing convention among visually impaired people is Braille. Since its inception in 1829, significant developments have taken place in the production of Braille and Braille media as well as in the transcription of printed material into Braille. Braille is understandable by visually impaired people; however vision people need not be able to understand these codes. The need to understand Braille documents by vision society and the production of huge amounts of Braille documents motivated the development of OBR for different languages (such as English, Arabic, etc.) across the world. The development of OBR for Amharic Braille has been started in recent years. However, OBR for Amharic Braille is still an area that requires the contribution of many research works. In this study an attempt has been made in exploring feature extraction and classification techniques for Amharic Braille recognizer.

To extract valid Braille dots from a Braille image and to group them into Braille cells, three feature extraction algorithms based on: fixed cell measures, horizontal and vertical projections, and grid construction are tested. The experimental result shows that feature extraction based on fixed cell measures performs well. To build classification models for prediction of Amharic characters from Braille cell representation J48 decision tree and the support vector machine (SVM) classifiers are investigated. Based on experimental results SVM outperforms decision tree classifier in predicting unseen extracted Braille features.

The explored feature extraction and classification techniques are integrated to the Amharic OBR system and are tested on real life Braille documents, in which 90.67% accuracy, on the average, is registered. This shows a promising result to design an applicable system. Handling noisy real-life Braille documents is the future research direction that needs an integration of generic segmentation and noise removal techniques.

CHAPTER ONE

INTRODUCTION

1.1. Background

Nowadays, the problem of vision has been an obstacle to access large printed contents in the information society. Globally, an estimated 40 to 45 million people are blind and 135 million have low vision according to the World Health Organization (WHO) [53]. Statistical census of 2007 indicates that there are over 500,000 blind people in Ethiopia. But those visually impaired people are part of the society and play a significant role in the society. So far, there are different means and systems created for those who are visually impaired people to reach what the world has in printed document [32]. One of the most valuable and indispensable system is through the use of Braille. Braille is a writing system with a series of dots that enables visually impaired people to read and write through touch using their fingers [1][2][65].

Braille is a tactile format of written communication for people with low vision and blindness worldwide since its inception by Louis Braille in 1829. It is a system of writing that uses patterns of raised dots to inscribe characters on paper [1]. Therefore, it allows visually-impaired people to read and write using touch instead of vision. Also it is a way for blind people to participate in a literate culture.

According to UNESCO's report reviewed in [10] Braille was not the first, or by any means the only method of touch reading. The earnest desire of the blind to find access to literature and of their sighted friends to open the door for them, led to many experiments in a variety of media. Even after Braille's invention, other forms of embossed symbols were planned and used; some employing lines and dots, others having the form of simplified Roman capitals.

Braille was more compact than any system which preceded or followed it. It was outstandingly versatile, equally able to express the languages and scripts of Europe, Asia

and Africa, and, as we have seen, readily adaptable to mathematics, musical notation and other purposes. Its main advantages, however, lay in the fact that, unlike the other embossed types, it is simple and easily used by the blind for transcribing their idea [51].

1.1.1. Braille Writing System

The Braille system includes symbols using which, the blind are able to review and study the written words. It provides a vehicle for literacy and gives a blind the ability to become familiar with spelling, punctuation, paragraphing, footnotes, bibliographies and other formatting considerations. Braille cell consists of 6 dots, 2 across and 3 down, is considered the basic unit for all Braille symbols. For easier identification, these dots are numbered downward 1, 2, 3 on the left, and 4, 5, and 6 on the right, as shown in Figure1.1.

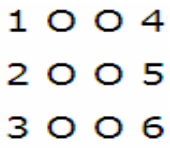


Figure1. 1The Dots that represent Braille symbols

A Braille *character* (or Braille *cell*), is a rectangular array of six points which makes up to sixty four (2^6) possible combinations of different character set or sequence of characters using 1 to 6 dots [31][37]. Each of the points in a cell can be either raised or flat. A raised point will be referred to as a *dot.* On the document, the Braille characters are embossed from left to right and from top to bottom, much like characters in ordinary documents. Though most countries adopt and/or define their Braille code so as to fit their local language characters and have their own set of defined dot pattern, in general Braille systems that are used worldwide currently are categorized into two levels [2]: *Grade 1 Braille* and *Grade 2 Braille.*

Grade 1 Braille represents each print character as one Braille cell which is a form of shorthand where group of letters may be combined in to a single Braille cell. On the other hand, Grade 2 Braille is embossed (processes of writing Braille code) by hand and/or

with a machine (Braille embosser) on a thick paper, and is read with fingers moving across on top of the dots [2].

Braille contractions represent groups of letters or whole words that appear frequently in a language. This is usually referred to as Grade 2 Braille. The use of contractions permits faster Braille reading and helps reduce the size of Braille books, making them somewhat less cumbersome [2].

As discussed in Ritchings et al. [45] a Braille document can either have the dots embossed on one side or on both. The latter is also called *inter-point* Braille when the positions of the dots from one side lie between the positions of the dots on the other side. This is designed to reduce the bulk of the document and to save on materials, given that a page of Braille is quite thick and heavy compared with ordinary paper. There are standards for the production of Braille that determine the height (protrusion) and diameter of a dot, the spacing between dots and between characters.

Modern touching education for visually impaired people in Ethiopia has started by the end of August 1924 [49]. Later, in 1934 Braille in Ethiopia was introduced by missionary and was designed by taking the 26 English characters pattern and 8 newly added patterns (Nebyeluel (1954), cited by [49]).

In Ethiopia, mainly in Addis Ababa, there are different education centers, such as AAU, Misrach Center, Entoto Blind people school, for visually impaired people at primary, secondary and tertiary levels. As a result, there has been a massive Braille documents (i.e. educational) produced and used by visually impaired people. But there is a gap in communication through documents between vision and visually impaired people.

1.1.2. Braille Recognition

Information in written form plays an undeniably important role in our daily lives. From education and leisure to casual note taking and information exchange, recording and using information encoded in symbolic form is essential [4]. In order to address this need the most widely adopted writing convention among visually impaired people is Braille.

However, although the production of Braille documents is relatively easy now, the problem of converting Braille documents into a computer-readable form still exists. This is a significant problem for two main reasons [4]. First, there is a wealth of books and documents that only exist in Braille that are deteriorating and must be preserved (digitized). Second, there is an everyday need for duplicating (the equivalent of photocopying) Braille documents and for translating Braille documents for use by non-Braille users.

OCR system involves reading scanned bitmap images of machine printed or handwritten text and translates the images into a form that the computer can manipulate. OCR examines and translates the characters into ASCII or Unicode text files that can easily be edited and manipulated [2]. As an extension to OCR, optical Braille recognition (OBR) offers many benefits to Braille users and people who work with them. A Braille optical character recognizer is interesting due to the following reasons [36]:

- It is an excellent communication tool for sighted people (who do not know Braille) with the blind writing.
- It is a cheap alternative to Braille copy machine instead of the current complex devices which use a combination of heat and vacuum to form Braille impressions.
- Braille writing is read using the finger which necessitate touching the document, for this reason the book after many readings is possibly has been deteriorated. OBR enables to preserve such valuable documents.
- It is interesting to store a lot of documents of blind authors which were written in Braille and were never converted to digital information so that it can be used by the public at large.

There are two types of OCR systems with respect to the way the input is provided to the system [54]: on-line and off-line OCR. In on-line OCR the recognition task is performed parallel with the writing process, where as in off-line OCR the recognition process is performed after the whole text entered to the OCR engine. OBR is more of an off-line recognition process.

According to Ng et al. [11], the Braille recognition system consists of six operations: scene constraints, image acquisition, image pre-processing, segmentation, feature extraction and interpretation.

Scene constraints is the first operation of the Braille OCR system in which it exploits and imposes the environmental constraints to reduce the complexity of all of the subsequent operations to a manageable level. There are two principal aims of this operation [11]: maximize the use of a prior knowledge of the scene by exploiting existing knowledge; and trivialize the problem of image analysis as far as possible by effective imposition of constraints.

Image acquisition is the most important step in any pattern recognition system. In Braille recognition systems data is provided to the system in the form of images of Braille embossed pages. The process of acquiring these images digitally can be achieved by using a number of different equipments such as scanners or digital cameras, both of which have been used by developers and researchers [31].

Image pre-processing is an essential step to detect and eliminate noise, deformation, bad illumination or blurring. Image pre-processing can be used for image enhancement by reducing noise, sharpening images, or rotating a skewed page. The preprocessing step has to deal with also binarization. Since Braille pages contains dots (foreground) and page (background) only, analysis of a binarised image is much simpler than that of gray-scale images [11]. For a simple global threshold, where the image histogram has easily identifiable peaks and valleys, the selection of the threshold value is straightforward. However, when the digitized Braille page images are noisy, and there are considerable spreads in gray level values, the selection of threshold value is problematic [18].

In the image segmentation stage the regions in the image corresponding to Braille dots are identified. Each dot in a scanned Braille image is composed of light and dark areas separated by background. In a gray-level image a dot is represented by a combination of a dark and a light region. Braille dots manifest themselves in the image as white/black region pairs.

Feature extraction is a representational mechanism of the Braille image. The function of feature extraction is to extract the Braille dots from the image and group them into cells. Extracting features from sub images that come from segmentation stage helps to simplify the recognition process.

Interpretation converts the Braille cells into their corresponding language text. At this stage, an orthogonal, binary image without noise is obtained [11]. The work of this phase is to group the Braille dots into cells and converts them into their equivalent characters.

Recently, the application of OCR technology has been the focus of many researchers [2][4][12][23][52][64] in an attempt to recognize Braille in the processes of making it accessible and computable to support visually impaired people. These days OBR systems can translate a variety of Braille documents in languages such as Arabic [1][2], English[12][45], Chinese [28], Spanish [23].

1.2. Statement of the Problem and Justification

There are more than half a million visually impaired individuals throughout the regions of Ethiopia that are ranging from student to teacher, artist to lawyer and their professional area may be politics, economics, language, history and other social fields [79]. In general, there are many visually impaired employees' who have been working in governmental and non-governmental organizations. These visually impaired individuals in their position and professional areas use Braille as the only means to codify their knowledge. Since the introduction of Amharic Braille, there have been massive Braille documents that have been produced and found at different parts of the country. Their knowledge is accessed only by those who can read and write Braille systems.

According to Nebyeluel [56], unless there is a smooth information flow from visually impaired people to sighted people and vice versa people do not have the necessary information about visually impaired people. The work of many visually impaired people would have remained buried. This would create a wide generation gap between the visually impaired and vision society.

Amharic language is the official language of Ethiopia and working language for most regional states (such as Amhara, Southern Nations Nationalities and People). According to Million [33] the present writing system of Amharic is taken from Geez which was brought to highlands by immigrants from south Arabia in the first century A.D. Currently, Amharic language consists of 33 core characters of which each occurs in seven orders, representing syllable combinations consisting of a constant and following vowel.

There are a number of attempts made on Amharic language to apply OCR technology [14][17][33][54][80][81]. In extending the OCR technology, there are also efforts done on Amharic Braille OCR system [16][49]. As a pioneer research, Teshome [49] tried to design an OBR system that applies global thresholding for binarization, mesh-grid technique for segmentation, counting black pixels at the cross-point of horizontal and vertical mesh grid lines for feature extraction and artificial neural network for recognition and prediction. He trained the system with 267 Amharic Braille characters and tested on features extracted from clean Braille documents and attained 92.5% accuracy. Based on his findings, Teshome recommends that to enhance the recognition accuracy for Amharic OBR system it is important to adopt different noise removal techniques and feature extraction algorithms that are insensitive to noise.

As a continuation of Teshome's effort, Ebrahim [16] further explored the possibility of designing Amharic Braille recognizer that works with degraded documents. This is done with the aim of enhancing the efficiency of the system by controlling the effect of noise through noise removal techniques. He incorporated the noise removal techniques with the previously adopted system to test the performance on real life Amharic Braille documents and registered an average accuracy of 86.63% on real life Braille documents. Based on his finding, since artificial neural network classifier commits substitution error in the characters and since Amharic Braille has fixed feature representation, Ebrahim [16] recommends incorporating a classifier with more generalization capability or rule based classifiers instead of artificial neural network to improve the performance of the system.

When the researchers investigated Braille writing system it gives consistently distinct combination of dots for each Amharic character representation. But noises and image

impurities are created mostly as a result of being repeatedly used, bent and scratch on Braille documents which generate unusual combination of dots which requires better feature extraction method. In addition, Amharic Braille characters have a hierarchical fixed feature representation; hence, there is a need to apply classifiers that can clearly provide information for each character class.

This study, therefore, aims to experiment feature extraction and classification techniques with the intent of designing a generic feature extraction algorithm that is insensitive to noise in real-life documents and also a suitable classification technique for Amharic OBR.

Towards solving this problem, this research attempts to answer the following basic research questions.

- Is there any consistent characters representational pattern in Amharic Braille writing system?
- Which feature extraction technique is more effective to extract all valid dots within Braille images for character representation?
- Which classifier is suitable for designing a generic model that enables Amharic optical Braille recognition system to register better performance in real life Braille documents?

1.3. Objectives of the Study

In line with the above statement of the problem the following general and specific objectives are formulated to clarify the intent of the present research.

1.3.1. General Objective

The general objective of this study is to design better feature extraction and classification schemes, in an attempt to improve the performance of the Amharic Braille OCR in real life Braille documents.

1.3.2. Specific Objectives

In order to achieve the above general objective the following specific objectives are drawn.

- To review related works in OCR specially in Braille recognition so as to understand the domain area and asses different feature extraction and classification algorithms for Braille recognition
- To analyze distinct features and patterns of Amharic Braille code style for each characters in Amharic writing system
- To design algorithms required for feature extraction and select based on their performance for Amharic Braille images representation.
- To create a classification model using suitable classifier in Amharic Braille recognition
- To evaluate the performance of the model integrated to Amharic OBR using real life Amharic Braille documents
- To report findings based on which further research issues are recommended so as to enhance the performance of Amharic OBR.

1.4. Methodology

As a matter of fact, methodology provides information that is helpful in understanding concepts, theories and principles for conducting, organizing and developing a research design. To undertake this research work, the following methods have been used.

1.4.1. Literature Review

Reviewing previous researches are important to share their idea in identifying what has been done and needs to be done in the area [16]. Related literature in the area of optical Braille recognition has been reviewed from various sources including: books, journal articles, conference papers, magazines, and the Internet. This is done to understand different OCR technologies, to map techniques and algorithms so far identified in addressing related problem domain for other languages.

1.4.2. Data Collection

Amharic Braille documents for training and testing the recognizer are collected from AAU Kennedy Library, Misrach Center and Entoto Blind School. Among these, AAU Kennedy Library offered most of the data. Table1.1 depicts the number of Braille documents collected for the study.

Source	No of Braille Documents
AAU-Kennedy library	16
Misrach Center	5
Entoto Blind School	3
Total	24

Table1. 1 Total number of Braille documents collected

1.4.3. Design and Development of Amharic Braille Character Representation

The collected Braille documents are digitized to create Braille images. Braille images are then preprocessed to reduce noise in the image and binarization is applied to convert gray-scale image into binary image. Once the image is preprocessed, it is segmented to identify Braille dots and features are extracted to group Braille dots into cells and create models for recognition.

Different models has been built using WEKA for both decision tree and support vector machine classifiers and the best models are selected based on training and predictive capacity on new features. As classification algorithm, J48 decision tree learning algorithm and SMO implementation of SVM are selected. To come up with a better representational scheme for Braille cells, various feature extraction algorithms are explored, including fixed cell measure, horizontal and vertical projection and grid construction.

1.4.4. Implementation Tools

Since this research is a continuation of the previous studies done by Teshome [49] and Ebrahim [16], visual C++ programming language is used to extract features from Braille

images and WEKA tool has been used to create models using the classification algorithms. WEKA is chosen because, WEKA:

- is freely available under the General Public License (GNU)
- is very portable because it is fully implemented in the Java programming language and thus runs on almost any computing platform
- contains a comprehensive collection of data preprocessing and modeling techniques, and
- is easy to use by a novice user due to the graphical user interfaces it contains

To map predicted classes and Amharic characters python programming language, which is suitable for text processing, has been used.

1.4.5. Testing Procedure

Testing is an important step to evaluate the performance of a given system. In the present study testing has been done on collected Amharic Braille documents and the performance of the feature extraction techniques and classification methods are measured using accuracy. Accuracy is the ratio of correctly predicted instances to the total number of instances.

The collected documents are first scanned using a flat-bed scanner and then fed to the preprocessing module designed by Ebrahim[16]. Then, the preprocessed Braille documents are submitted to the feature extraction algorithms and extracted features are generated. Based on these extracted features, the performance of the feature extraction techniques have been tested with decision tree classifier; this is done because decision tree is found to be simple to generate prediction results. After feature extraction technique has been selected based on the accuracy they register, features are also extracted using the selected feature extraction technique and the performance of the classification models have been tested. At this stage a classifier with better accuracy has been selected. Finally, based on the selected feature extraction and classification techniques the performance of the system has been tested with real life Braille documents and then the accuracy of the

system is measured by comparing the expected results with the system output which generates correctly classified and misclassified characters.

1.5. Scope and Limitation of the Study

This study is the continuation of previous works in the area. The main focus of this research is investigating different feature extraction and classification algorithms to design a generic Amharic Braille recognizer that can handle the various artifacts in Braille documents. To this end, different feature extraction and classification algorithms are investigated and suitable techniques are integrated to the Amharic OBR system. This is followed by creating classification models on training data sets and test with real life optically scanned Braille documents to report the performance of the system.

Braille can be plastic and/or paper sheet and the documents produced can be either manually produced or typewritten. In the present study typewritten documents have been used. Since the intensity of pixels in Braille image varies from one color to another, which affects the noise level, documents with gray level have been selected.

In Amharic Braille writing one to three cells are used. In the present study, however, we consider only one and two Braille cells by including 238 basic Amharic characters, 20 numerals and 23 punctuation marks. Though, writing in Amharic Braille is being embossed on both sides, this study is delimited in converting single side Braille Amharic characters, rather than double-side Braille.

Since there is no organized Braille document image corpus for training and testing the Amharic OBR system, in this research we used limited number of datasets organized from clean Braille documents to high level noisy documents.

1.6. Significance of the Study

The outcome of this study has significant contribution to bridge the communication gap between visually impaired people and vision people in different professions and work areas.

In addition, Amharic OBR offers many benefits to Braille users and those who work with them, facilitating communication, reducing storage space, and preserving out-of-print Braille texts. Everyone who works with visually impaired people and does not read Braille will benefit from using the Amharic OBR. For example: parents, teachers and public organizations communicating with blind individuals. All people in workplaces where Braille is used can read Braille easily by using Amharic OBR.

It also provides significance contribution in converting Braille documents to digital format in order to preserve and share, and also facilitate text-to-speech conversion.

1.7. Organization of the Study

This study is organized into five chapters. The first chapter provides an overview of the general background, the present research problem, objectives of the study and methodologies adopted. In chapter two an attempt has been made to review literatures in Amharic writing system, features of Amharic Braille character and characteristics of Braille recognition. Review of related research works in Braille problem domain along with the techniques used is also discussed.

Chapter three provides the techniques and algorithms used to develop the current system. It also provides the overall architecture of OCR for the recognition of Amharic Braille documents. The various decision tree and SVM learning approaches incorporated in the classification processes are described.

Chapter four deals with the experimentation activity undertaken to implement the methods and techniques described in chapter three. Experimental results are presented to show the applicability of the classifier and the feature extraction algorithm for Amharic Braille document image collections. It also shows the successes and difficulties faced while trying to implement the technique to the problem domain of interest is presented. It also presents the recognition rate achieved for different test cases after the design and training of the decision tree and SVM. Finally, in chapter five concluding remarks and recommendations for future research are forwarded.

CHAPTER TWO

LITERATURE REVIEW

2.1. Overview

These days massive amounts of documents are available in different forms, such as printed, digital and Braille, throughout the entire world. However, printed and digitized documents are not convenient to access by visually impaired people and on the other side Braille documents may not be understandable by vision society [8]. Therefore, there should be a means to build two way communications among vision and visually impaired people. Designing Braille document recognizer, optical Braille recognition system, is one means for solving the specified problem [2][11][16][49].

Optical Braille recognition (OBR) system allows visual people to read Braille documents with the help of flatbed scanner and optical character recognition (OCR) software [2]. OBR system can design for either single side Braille or both single side and double side Braille. In case of double side Braille a slight diagonal offset prevent the dots from interfering each other and this makes recognition very difficult [11].

Many attempts have been made to bridge the communication gap between visually impaired society and the vision society by developing Optical Braille Recognition systems (OBR) which are capable of converting Braille documents to printed characters. OBR system had been developed for Braille documents written in different languages, such as English, Arabic, Chinese and European languages.

2.2. Amharic Writing System

Amharic is a Semitic language that has been used for centuries. According to Million [33], Amharic is the second most-spoken Semitic language in the world, after Arabic, and the official working language of the Federal Democratic Republic of Ethiopia. Thus, it has official status and is used nationwide. Amharic is also the official or working language of several of the states within the federal system, including the Amhara regional state and the multi-ethnic Southern Nations, Nationalities, and Peoples regional state,

among others. Amharic has been the working language of government, the military, and of the Ethiopian Orthodox Tewahedo Church throughout medieval and modern times and is also spoken in a number of other countries outside Ethiopia, particularly in Eritrea, Egypt, Israel and Sweden [6].

According to [7] [33], the present writing system of Amharic is taken from Ge'ez which was indigenous and born in Axum. By the time Ge'ez was replaced by Amharic all the 26 symbols of Ge'ez were adopted in the Amharic writing system. In addition Amharic has created eight additional symbols to represent sounds not found in the Ge'ez script [7] [33]. This integration has increased the total basic symbols used in Amharic writing system to 34. Accordingly, since Amharic language has its own scripts, understanding the characteristics and distinct features of the scripts is important in the feature extraction phase of the Braille recognition system.

2.2.1. Amharic Characters

Before the Amharic language characters (**ፊደል**) have got their existing shape and number of symbols, they pass through a series of changes by adopting and using Ge'ez script over a period of time [7] [33]. Currently, Amharic consists of 33 core characters shown in Table2.1 and 1 special character (ሿ) which occurs in seven orders representing syllable combinations of a consonant followed by vowel.

ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ
ለ	ሉ	ሊ	ላ	ሌ	ል	ሎ
ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ
መ	ሙ	ሚ	ማ	ሜ	ም	ሞ
ሠ	ሡ	ሢ	ሣ	ሤ	ሥ	ሦ
ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ
ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ
ሸ	ሹ	ሺ	ሻ	ሼ	ሽ	ሾ
ቀ	ቁ	ቂ	ቃ	ቄ	ቅ	ቆ
በ	ቡ	ቢ	ባ	ቤ	ብ	ቦ
ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ
ቸ	ቹ	ቺ	ቻ	ቼ	ች	ቾ
ኀ	ኁ	ኂ	ኃ	ኄ	ኅ	ኆ
ነ	ኑ	ኒ	ና	ኔ	ን	ኖ
ኘ	ኙ	ኚ	ኛ	ኜ	ኝ	ኞ
አ	ኡ	ኢ	ኣ	ኤ	እ	ኦ
ከ	ኩ	ኪ	ካ	ኬ	ክ	ኮ

ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ
ወ	ዉ	ዊ	ዋ	ዌ	ው	ዎ
ዐ	ዑ	ዒ	ዓ	ዔ	ዕ	ዖ
ዘ	ዙ	ዚ	ዛ	ዜ	ዝ	ዞ
ዠ	ዡ	ዢ	ዣ	ዤ	ዥ	ዦ
የ	ዩ	ዪ	ያ	ዬ	ይ	ዮ
ደ	ዱ	ዲ	ዳ	ዴ	ድ	ዶ
ጀ	ጁ	ጂ	ጃ	ጄ	ጅ	ጆ
ገ	ጉ	ጊ	ጋ	ጌ	ግ	ጎ
ጠ	ጡ	ጢ	ጣ	ጤ	ጥ	ጦ
ጨ	ጩ	ጪ	ጫ	ጬ	ጭ	ጮ
ጰ	ጱ	ጲ	ጳ	ጴ	ጵ	ጶ
ጸ	ጹ	ጺ	ጻ	ጼ	ጽ	ጾ
ፀ	ፁ	ፂ	ፃ	ፄ	ፅ	ፆ
ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ
ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ
ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ

Table2. 1 Amharic core characters with seven orders

In addition to the above 238 characters, Amharic consists of 44 other characters called labialization which are depicted in Table2.2.

ኰ	ኩ	ኩ	ካ	ኬ
ጐ	ጉ	ጉ	ጓ	ጔ
ቈ	ቁ	ቁ	ቋ	ቌ
ጐ	ጉ	ጉ	ጋ	ጔ

ሳ	ማ	ሯ	ሳ	ሻ
ባ	ታ	ቿ	ና	ኗ
ኧ	ሃ	ኻ	ዳ	ጃ
ጣ	ሓ	ጻ	ፋ	ቫ

Table2. 2 Amharic labialization characters

2.2.2. Amharic Numerals and Punctuation Marks

Amharic writing system uses both Ethiopic and Hindu-Arabic numerals. The Ethiopic numbers shown in Table 2.3, when put in numeral form, are written on top and bottom frame. This helps to uniquely identify the numerals from textual character set. The Ethiopic number system does not contain symbols for zero, negative numbers, decimal points and mathematical operators for performing arithmetic operations.

፩	፪	፫	፬	፭	፮	፯	፰	፱	፲	፳	፴	፵	፶	፷	፸	፹	፺	፻
1	2	3	4	5	6	7	8	9	10	20	30	40	50	60	70	80	90	100

Table2. 3 Ethiopic numerals

Amharic writing system has also punctuation marks listed in **Figure2.1** and uses other Latin-based symbols like question mark(?), exclamation mark(!), quotes(“”) and parenthesis().

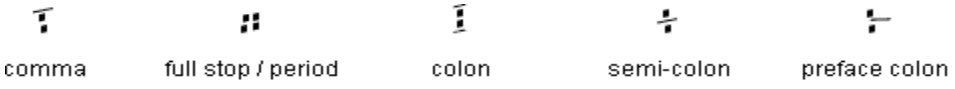


Figure2. 1 Amharic Punctuation Marks

2.3. Amharic Braille

2.3.1. Amharic Braille Evolution

Braille, a touch-reading system for the blind, was first introduced in 1829 by a French scientist Louis Braille [1] [4]. According to [10] Braille was not the first, or by any means

the only method of touch reading. The earnest desire of the blind to find access to literature and of their sighted friends to open the door for them, led to many experiments in a variety of media. Even after Braille's invention, other forms of embossed symbols were planned and used-some employing lines and dots [10]. The most famous system that is based on touching is Braille system.

The Braille format used by visually impaired people worldwide today is the Louis's Braille format [28]. It uses a character set made up of different combinations of raised dots in a 3-by-2 (3 rows and 2 columns) arrangement to represent different characters or sequences of character [1]. Nowadays the system is being extended to four rows so that each cell can represent 128 different characters [32].

According to UNESCO's 1951 report reviewed in [10], in the past years Braille alphabets were arranged for at least a dozen African tribal tongues. Missionaries were their chief authors, although more recently other voluntary organizations in co-operation with Departments of Education and Social Welfare of the country have been increasingly active.

UNESCO's 1951 report reviewed in [10] states that three languages, which are neither tribal nor linguistically African, used in Africa have their Braille forms. They are Arabic, to which Braille was adapted in 1878; Amharic, with a Braille of comparatively recent date, used in the American Mission, Western Ethiopia; and Afrikaans, to which the first adaptation was made in 1923. With the exception of Arabic and Amharic, all the Brailles of Africa were built from the traditional European symbols; but, as the adaptations were made variously from English, French, Norwegian and Dutch backgrounds, and, still further, as the Latin alphabets created for tribal writing followed different phonetic patterns, complete uniformity between them was lacking.

In Ethiopia Braille as a means of education for visually impaired people was introduced in 1923 by an American missionary. Before gaining its current representations, different improvements have been done on Amharic Braille [56]. There had been different reasons proposed for each improvement.

The first Amharic Braille was comprehensive and complete though it did not work for Ge'ez characters [56]. As a result an amendment was performed in 1952 on two aspects: the first was to eliminate the redundant Amharic characters like (ሀ፣ሐ፣ኀ), (አ፣ዐ), (ሰ፣ሠ) and the second was to replace Amharic numerals with Arabic numerals. The second improvement was made in 1956, when, Sir. Cluth Mecanize (chairperson of the world blind society) came to Ethiopia for visit, to further reduce the variants of Amharic characters.

In 2001 the Amharic Braille was subjected to change for the third time, on the third version. The reason for the amendment at that time was previous version of the Braille designed by the American missionary did not consider many issues set by the world blind association. Some of these rules and procedures which were considered in adopting Braille to local language are the following [56]:

- Different languages should have similar Braille code. This is aims in simplifying the international publication, to avoid difficulties for those who are blind and interested to study different languages, and to create common understanding among blind society in the world.
- Languages that have the same sound or related languages that are similar in sound, in character number and type should have similar Braille code structure.

The early Amharic Braille characters adopt the English character which does not represent the right sound and hence different improvements have been made. An amendment was made on the third version to make the Braille suitable for the blind child and to minimize the difficulties present as a legal document. After the last amendment the fourth version of Amharic Braille has been used throughout the country since 2001. For this study the focus is on the fourth version of Amharic Braille character.

2.3.2. Amharic Braille Code Characters

The Amharic Braille characters composed of three parts: core and special letters, numerals and punctuation marks.

2.3.2.1. Amharic Braille Character for Core Letters

In Braille, printed text is represented by Braille characters. Each character is constructed as a set of six points arranged in two columns of three called a Braille cell. The cell has been set according to the tactile resolution of fingerprints of persons. The position of each point in a cell is identified by a number and can be either raised or flat. A raised point is also called Braille dot. As a result Amharic printed symbols also represented by a set of Braille characters forming a Braille dot.

In Braille there are about 64 possible combinations of dots which can be formed in a single Braille cell. Since Amharic characters are huge, it is difficult to represent them by a combination of dots in a single Braille cell. Consequently, most of the Amharic characters are represented by two Braille cells.

As mentioned earlier Amharic has core and special characters. Each Amharic character has seven variants. Except the 6th variant, the remaining six sounds add the particular vowel code to get the required sound in character. Table 2.4 shows Amharic Braille characters for the fourth version of Amharic Braille.

In the fourth version of Amharic Braille for Amharic symbols the 6th form of a character is used as root for its variants. The other variants are formed using the form of their 6th variant and additional Braille cell consists of vowels for the given form. The 1st form includes a vowel at 2:6, the 2nd at 1:3:6, the 3rd at 2:4, the 4that 1, the 5th at 1:5 and the 7th at 1:3:5. For example: given characters ሀ ሁ ሂ ሃ ሄ ህ ሆ, the representation of character ህ is used as a root for its variants. If the required character is "ሀ" it can be written as (1:2:3:5, 2:6); where 1:2:3:5 is "ህ" where as 2:6 is a vowel for the 1st form. Similarly ሁ can be written as (1:2:3:5, 1:3:6) where 1:3:6 is a vowel for the 2nd form, ሂ as (1:2:3:5, 2:4) where 2:4 is a vowel for the 3rd form, ሃ as (1:2:3:5, 1), ሄ as (1:2:3:5, 1:5) where 1:5 is a vowel for the 5th form and ሆ as (1:2:3:5, 1:3:5) where 1:3:5 is a vowel for the 7th form.

1st form	2ndform	3rdform	4thform	5th form	6th form	7th form
ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ
ለ	ሉ	ሊ	ላ	ሌ	ል	ሎ
ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ
መ	ሙ	ሚ	ማ	ሜ	ም	ሞ
ሠ	ሡ	ሢ	ሣ	ሤ	ሥ	ሦ
ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ
ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ
ሸ	ሹ	ሺ	ሻ	ሼ	ሽ	ሾ
ቀ	ቁ	ቂ	ቃ	ቄ	ቅ	ቆ
በ	ቡ	ቢ	ባ	ቤ	ብ	ቦ
ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ
ቸ	ቹ	ቺ	ቻ	ቼ	ች	ቾ
ኀ	ኁ	ኂ	ኃ	ኄ	ኅ	ኆ
ነ	ኑ	ኒ	ና	ኔ	ን	ኖ
ኘ	ኙ	ኚ	ኛ	ኜ	ኝ	ኞ
አ	ኡ	ኢ	ኣ	ኤ	እ	ኦ
ከ	ኩ	ኪ	ካ	ኬ	ክ	ኮ
ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ
ወ	ዉ	ዊ	ዋ	ዌ	ው	ዎ
ዐ	ዑ	ዒ	ዓ	ዔ	ዕ	ዖ
ዘ	ዙ	ዚ	ዛ	ዜ	ዝ	ዞ
ዠ	ዡ	ዢ	ዣ	ዤ	ዥ	ዦ
የ	ዩ	ዪ	ያ	ዬ	ይ	ዮ
ደ	ዱ	ዲ	ዳ	ዴ	ድ	ዶ
ጀ	ጁ	ጂ	ጃ	ጄ	ጅ	ጆ
ገ	ጉ	ጊ	ጋ	ጌ	ግ	ጎ
ጠ	ጡ	ጢ	ጣ	ጤ	ጥ	ጦ
ጨ	ጩ	ጪ	ጫ	ጬ	ጭ	ጮ
ጸ	ጹ	ጺ	ጻ	ጼ	ጽ	ጾ
ፀ	ፁ	ፂ	ፃ	ፄ	ፅ	ፆ
ጸ	ጹ	ጺ	ጻ	ጼ	ጽ	ጾ
ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ
ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ

Table2. 4 Amharic Braille code for core characters

2.3.2.2. Amharic Braille Character for Numerals

Amharic uses two types of numerals: Arabic and Ethiopic. The amendment of the fourth version of Amharic Braille also touched numerals. Accordingly, the combination of dots ⠿ (1:2:3:4:5:6) are used to indicate Ethiopic numbers, while the combinations of dots ⠼ (3:4:5:6) are used to indicate Arabic numbers. Table 2.5 shows the representations of each Arabic and Ethiopic numbers used in Amharic Braille.

Numeral	Corresponding Amharic Braille Representation									
Ethiopic	⠚	⠁	⠃	⠉	⠙	⠑	⠋	⠛	⠓	⠊
Arabic	1	2	3	4	5	6	7	8	9	0

Table2. 5 Amharic Braille code for numerals

2.3.2.3. Amharic Braille Character for Punctuation Marks

The Amharic Braille also consists of different punctuation marks. Most of the punctuation marks in the Amharic Braille code are taken from English. For sample Amharic punctuation marks, their representation is shown in Table 2.6.

Punctuation	→	←	↑	↓	*	.	( )	[	]	…
Braille code	⠳	⠪	⠸	⠩	⠔	⠲	⠐⠣⠂	⠈⠣	⠐⠜⠂	⠠⠠⠠
Punctuation	;	:	!	÷	←	:	“	-	–	/
Amharic Braille code	⠆	⠒	⠖	⠌	⠤	⠐	⠦	⠤⠤	⠠⠤⠠	⠌

Table2. 6 Amharic Braille code for punctuation marks

2.3.3. Braille Writing and Reading System

Braille is a writing system that enables visual people to read and write through touch using a series of raised dots to be read with their fingers [1]. Braille is a system of touch reading and writing in which raised dots represent the letters of the alphabet. Braille also contains equivalents for punctuation marks and provides symbols to show letter groupings. Therefore, it allows visually-impaired people to read and write using touch instead of vision.

2.3.3.1. Braille Writing

Writing on Braille materials is made by, moving fingertips from right to left, physically pressing the dots into the paper so that they show up on the other side of the Braille sheet [45]. Braille writing devices have evolved into some very sophisticated pieces of equipment [51]. Braille sheet is a thick paper or plastic material designed to withstand the pressure while one write and read the Braille code. The size of the paper varies between documents produced by different means. In addition, the color of the paper varies, as it does not play any role in conveying the written information. The majority of Braille sheets are either buff colored or white [1].

There are a number of different methods for personal Braille writing that can result in tactile output. The writing devices most significant for early Braille literacy are those like pencil and paper--couple writing and reading by tying the writing process directly to the production of hard copy output. These devices include the slate and stylus as well as mechanical and electronic Braille [46] [48]. One can write:-

- Manually using a handheld stylus (to make the impressions) and slate (to hold the paper) by physically pressing each dot into Braille sheet. Writing Braille with a slate and stylus compares to write with a pen or a pencil. The stylus used to push the dots down through the paper, while the slate serves as a guide. The speed of writing Braille with the slate and stylus is about the same as the speed of writing Braille with a pen or pencil. However, the speed and accuracy would depend both on the skill and experience of the user, and the frequency of the use.

- Mechanically with the Braille typewriter. This has one key for each of the six dots in a Braille cell.
- Electrically using Braille printer that attached to computer system. Increasingly, electronic Braille embossers are being used in governmental departments, organizations, schools and other education institutions to produce material in Braille. Although there are several models of electronic Braille embossers, the focus is based on the following points[16] :
    - The speed of the embossers in characters per second
    - Whether the embossers allow for independent operation by users who are blind
    - Whether the embossers have voice output menus
    - Whether the embossers are able to produce large volumes of Braille
    - Whether the embossers have the facility to produce graphic.

2.3.3.2. Braille Reading

Braille is designed to be read by moving fingertips from left to right across the lines of dots. Both hands are usually involved in the reading process, and reading is generally done with the index fingers.

According to Robert [46], people that have already learned to read print have mastered the “reading process” skills; however, they must develop the following skills associated with reading using their fingers.

- Tactual Discrimination--The ability to discriminate discrete tactual differences is essential to efficient Braille reading. The noticeable shape or arrangement of dots is the most critical variable in Braille reading.
- Finger Dexterity--The effective Braille reader will have “curious” fingers that move quickly, with ease. Many readers use all four fingers of each hand. This speeds up the reading process by allowing the reader a view of a series of symbols rather than a single cell.
- Hand and Finger Movement--Most good Braille readers use two hands. A skilled two handed reader begins reading a line of Braille by placing both hands at the

beginning of a line. At approximately the middle of the line, the right hand continues to read to the end of the line while the left hand moves in the opposite direction to locate the beginning of the next line. The right hand finishes reading the first line, the left hand then reads the first words on the next line, and the right hand quickly joins the left hand on the second line.

- Light Finger Touch--Beginning readers may have a heavy touch; however, to be good two hand readers one must acquire a light touch.
- Page Turning--Braille readers should be instructed to turn the page quickly with the right hand when the left hand cannot find another line.

2.3.3.3. Nature of Amharic Braille Embossed

The Amharic Braille like any other Braille documents is formed by embossing the dots on the back side of the medium sheet so that they can be read from the facing side. There are also different sized Braille sheet; the size of the page varies between documents produced by different means. The Braille characters are embossed in lines from the top of the document to the bottom like printed documents. Braille documents can have the dots embossed on one side or both side. As the volume of the Braille documents become large, it is advantageous to use both sides of the sheet; however, this is not widely practiced with Amharic Braille [16][49].

2.4. Features and Challenges of Braille Documents

Braille is a particular system of representing information in tactile form. As such, Braille documents are formed by groups of protruding “dots” representing characters and various symbols (including music). The meaning of each Braille character depends on the type of Braille encoding used [4]. In Grade 1 Braille there is a one-to-one correspondence between printed characters and their Braille representation. Braille contractions representing groups of letters or whole words that appear frequently in a language. This is usually referred to as Grade 2 Braille. The use of contractions permits faster Braille reading and helps reduce the size of Braille books, making them somewhat less cumbersome [2].

A sheet of printed Braille will have a series of cells embossed on thick paper and passing one's forefinger over each line of embossed cells can sense the embossing. A standard sheet of Braille has about forty cells per line and may contain 20 or more lines [62]. The nature of Braille has direct implications on the physical characteristics of documents. The thickness of the page material (most commonly card) and the added thickness introduced by the protrusions result in very bulky documents, in comparison to printed documents containing the same information (a dictionary can occupy a whole bookcase).

The dimensions of a Braille dot have been set according to the tactile resolution of the fingertips of person. The horizontal and vertical distance between dots in a character, the distance between cells representing a word and the inter-line distance are also specified by the Library of Congress. Dot height is approximately 0.02 inches (0.5 mm); the horizontal and vertical spacing between dot centers within a Braille cell is approximately 0.1 inches (2.5 mm); the blank space between dots on adjacent cells is approximately 0.15 inches (3.75 mm) horizontally and 0.2 inches (5.0 mm) vertically [2][22]. A standard Braille page is 11 inches by 11.5 inches and typically has a maximum of 40 to 43 Braille cells per line and 25 lines. It is not possible to vary the size of the letters, to write between the lines, or to scribble in the margins, as those who write prints so often do [51].

Due to the size of a Braille cell, a page of normal size, about 11 inches wide by 11 or 12 inches high, can hold not more than 1000 characters [12]. Braille paper must also be thick enough to withstand a certain amount of pressure for tactile sensing. Therefore, printed Braille documents are very bulky. To mitigate this problem somewhat, most Braille documents are printed in "inter-point", that is with the embossing done on both sides of each page, with a slight diagonal offset to prevent the dots on the two sides from interfering with each other [12][4]. This makes the translation process more difficult as the recognition technique employed by the translation system is based on the visual perception of the Braille document, but not tactile sensing as used by the visually impaired users.

An obvious, perhaps, but significant characteristic of Braille documents is the absence of any information visible in a color contrasting the background [4]. The only information recorded on a Braille page is in terms of the protrusions created by embossing the card (under uniform illumination a Braille document page appears blank). The fact that Braille documents are not intended to convey any visual information also has repercussions on the quality of card used to produce them. It is not uncommon for the card to be of low (visual) quality, with visible grain and imperfections (dark and light regions). This fact can affect the recognition of Braille documents by visual means (the objective of any realistic automated conversion system).

Since its inception in 1829, Braille has been a very effective means of written communication for the blind and the partially sighted people and significant developments have taken place in the production of Braille and Braille media as well as in the transcription of printed material into Braille. However, although the production of Braille documents is relatively easy now, the problem of converting Braille documents into a computer-readable form still exists. This is a significant problem for two main reasons [4]. First, there is a wealth of books and documents that only exist in Braille that, as with other rare/old documents, are deteriorating and must be preserved (digitized). Second, there is an everyday need for duplicating (the equivalent of photocopying) Braille documents and for translating Braille documents for use by non-Braille users.

Since manual transcription is tedious and costly, there is a significant need for a system to recognize Braille documents. The automated recognition of Braille documents is not straightforward due to the special characteristics of the documents themselves and the constraints of the application domain. More specifically, in addition to the natural expectations for high efficiency demanded from a document conversion application, a Braille recognition system must also be easy to use by visually impaired people (it should not require complicated setup etc.) and cost-effective (should only use commercially available standard equipment).

2.5. Optical Braille Recognition System Overview

Optical Character Recognition (OCR) is the process of converting scanned images of machine printed or handwritten text into a computer processable format [58]. The basic steps in Braille recognition process can be broadly broken down into five stages:

1. Image Acquisition/Digitization
2. Pre-processing
3. Segmentation
4. Feature extraction
5. Classification

The image capturing stage captures a Braille page and converts it into a graphical image for further processing. The preprocessing stage is a collection of operations that apply successive transformations on an image. It takes in a raw image, reduces noise and distortion, removes skewness and there by simplifying the processing of the rest of the stages. The segmentation stage takes in a page image and separates the different logical parts, like text from graphics, lines of a paragraph, and characters of a word. The feature extraction stage analyzes a text segment and selects a set of features that can be used to uniquely identify the text segment. The selection of a stable and representative set of features is the heart of pattern recognition system design. The classification stage is the main decision making stage of an OCR system and uses the features extracted in the previous stage to identify the text segment according to preset rules.

2.5.1. Image Acquisition/Digitization

Image Acquisition captures a Braille page and converts it into a digitized image array. This operation is concerned with the process of translation from light stimuli falling onto the sensing elements of a capturing device to digital values [11]. The capturing device can be either scanner or digital camera.

As shown and has been verified by the authors in [32] a scanning resolution of between 80dpi and 200dpi is the most appropriate for Braille documents.

2.5.2. Image Pre-processing

Preprocessing is a technique used to prepare isolated character for recognition. Most of time the image captured in image acquisition phase is not widely in digital form rather it is in scanned form. As a result scanning introduces noise into the images even at high resolution. At this stage the image is converted into Gray scale to reduce noise. In addition preprocessing modifies and prepares the pixel values of the digitized image for subsequent operations. The preprocessing stage is a collection of operations that apply successive transformations on an image. This includes gray-scale conversion, geometric adjustment, brightness adjustment, noise removal, biniarization and edges sharpening of the Braille dots.

Full-color to Grayscale Conversion: Gray scale images are made up of only one bit value. They lack the chromatic information. They are also called as monochromatic images. Image pixel values range from 0 to 255. It shows only the luminance information. The conversion algorithm from full-color image to grayscale image is not unique, and each algorithm has its own applications [60].

Image Geometry Correction: The image acquired by scanner may be rotated or translated. Geometric correction aims to eliminate such effects and bring all scanned Braille document image to a common reference point.

Image Brightness Adjustment: Local dark or bright areas can occur, due to non-uniform illumination of scanner, or part of the surface of the document being uneven. Left untreated, these areas can cause classification errors in later stage.

Noise Removal/filtering: Noise is the error occurred during the image acquisition process that results in pixel values that do not reflect the true intensities of the real scene. This noise generally manifests itself as random fluctuations in gray-level values superimposed upon the “ideal” gray-level value, and it usually has a high spatial frequency.

Digital images are prone to be corrupted by a variety of types of noises. Common types of noises include salt and pepper noise and Gaussian noise [39]. Salt and pepper noise contains randomly distributed occurrences of both black and white impulses. Gaussian noise contains intensity variations that are drawn from a Gaussian distribution and is a very good model for many kinds of sensor noises caused by camera electronics errors [39]. Noise filtering aims to reduce the noise generated during the image capture process.

Researchers attempt to design different types of noise detection and removal techniques in order to reduce the effect of degradation during the recognition process. Among these a Gaussian filter is applied by many researchers. Gaussian filter is a low-pass spatial filter which seeks to attenuate the high spatial frequency noise from the image while at the same time preserving the detailed edge information of the Braille dots [57].

Edge Sharpening/enhancement: Edge enhancement is performed to enhance or strengthen the edge of the Braille dots. According to [12] edge enhancement mainly focuses to sharpen the fine details of the image that has been blurred.

Image binarisation: Image binarization or thresholding is performed to distinguish the Braille dots from the background. Image thresholding/binarization transfers a grayscale image into its binary version representing objects and background [2]. The quality of the binarization step is critical for subsequent analysis. If poorly binarized images are used, document understanding would be difficult or even impossible.

Thresholding or binarization of documents can be categorized in to two main classes [16] [61]: global and local thresholding. Global thresholding techniques use a single threshold; on the other hand local thresholding techniques compute a separate threshold based on the neighborhood of the pixels.

2.5.3. Segmentation

Segmentation is a process of separation of dots from Braille image that can further be grouped into a cell [16]. These cells are further grouped into character, words of any

strokes in Braille. The low level of abstraction to be extracted in the Braille recognition is the single dot that could have six alternative positions in a cell.

Since Braille pages contains dots (foreground) and page (background) only, analysis a binarised image is much simpler than that of gray-scale images [11][36]. However, the digitized Braille page images are noisy, and there are considerable spreads in gray level values, the selection of threshold value is problematic. At this stage, the regions in the image corresponding to Braille dots are identified. In a binary image, dots can be identified as small connect components. This situation is valid when single-sided Braille documents are encountered [18].

There are different segmentation techniques particularly applied in Braille images such as mesh-grid and local measure. Local measure works using the difference dot pixel intensity level as a threshold value. Because dots produce different color level at the top and bottom. On the other hand mesh grid work based on constructing a vertical and horizontal grid on strictly arranged Braille cells. It is applicable and widely used in Braille because Braille cells arranged strictly following vertical and horizontal layout of the document.

2.5.4. Feature Extraction

Feature extraction is a representational mechanism of the Braille image. According to [12] feature extraction is the representation of Braille dots to extract Braille dots from the binarised image and group them to cells.

The above three stages (image capturing, preprocessing and segmentation) aim to make the image suitable for different feature extraction algorithms. Some feature extraction algorithms only deal with the contours of the image while some algorithms calculate every pixel of the image. On the other hand, the initial image may be noise affected, or blurred by other reasons.

The dimension of Braille dots have been according to the tactile recognition of fingertips of a person. The horizontal and vertical distance between cells representing a character

and the inter-line distance between dots within a cell are also specified by the Library of Congress [52]. It was also indicated in [59] that the average dot height is 8 pixels. Each dot is composed of a bright and a dark region with a small space between them. Knowing both vertical and horizontal indentations and spacing, we are able to locate and detect the dots in the image. As indicated in [64] it is easier to detect the vertical parameters of the Braille image since the spacing is usually much larger compared with the horizontal ones. Horizontally it is more likely to have the undesired situation that adjacent dots would overlap.

As the dots are embossed on both sides of a Braille page, dots on the front page are raised above the page and those on the back side made holes on the front side. According to [12] these convex and concave characteristics of the dots reflect the illumination light in two different angles, creating an illuminated hole at the left side of the captured dot image for those front face Braille dots and at the right side for those back face dots. Using this illumination characteristics, the position of the illuminated hole can be used as the feature to distinguish the front face and back face dots. The dots on the back side must be removed before further processing.

Sometimes the image features are not directly or obviously associated to any part of the image. Detectable means the extraction algorithm corresponding to the feature must exist, otherwise the feature is useless. Different features are associated with different extraction algorithms that output collections of the feature descriptors. Good image features should satisfy the following conditions [63]:

a. robust to transformations – the image features should be as invariant as possible to image transformations including translation, rotation, and scaling, etc.
b. robust to noise – the image features should be robust to noises and various degraded situations.
c. feature extraction efficiency – image features can be computed efficiently.
d. feature matching efficiency – the matching algorithms should only require a reasonable computational cost.

Extraction of good features is the main key to correctly recognize an unknown character. A good feature set contains discriminating information, which can distinguish one object from other objects. It must also be as robust as possible in order to prevent generating different feature codes for the objects in the same class. The selected set of features should be a small set whose values efficiently discriminate among patterns of different classes, but are similar for patterns within the same class. Features can be classified into two categories [58]:

a. Local features, which are usually *geometric* (e.g. concave/convex parts, number of endpoints, branches, joints).

b. Global features, which are usually *topological* (connectivity, projection profiles, number of holes, etc) or *statistical* (invariant moments).

The selection of image features and corresponding extraction methods is probably the most important step in achieving high performance for an OCR system [39]. At the same time, the image feature and the extraction method also decide the nature and the output of the image-preprocessing and segmentation steps. Some image features and the extraction algorithms work on color images, while others work on gray level or binary images. Moreover, the format of the extracted features must match the requirements of the classifier [39]. Some features like the graph descriptions and grammar-based descriptions are well suited for structural and syntactic classifiers. Numerical features are ideal for statistical classifiers. Discrete features are ideal for decision trees. As mentioned in [34] there are different methods used to extract features. Some are highly language specific like profiles, structural descriptors and transform domain representations and others consider the entire image as the feature.

2.5.5. Classification

As mentioned in [34] feature extraction and classification are the heart of OCR. Classification is a mechanism which takes feature of objects as its input and then label to which class the object belongs to. The classification stage is the main decision making

stage of an OCR system and uses the features extracted in the previous stage to identify the text segment according to preset rules.

Training and testing are the two basic phases of any pattern classification problem. During training phase, the classifier learns the association between samples and their labels from labeled samples. The testing phase involves analysis of errors in the classification of unlabelled samples in order to evaluate classifier's performance. It is desirable to have a classifier with minimal test error [34].

2.6. Review of Related Works

There has already been a lot of research done in the area of Braille recognition for Amharic, English, and other languages. The researcher tried to summarize some of them as follows:

2.6.1. Optical Braille Recognition System for English Characters

Several researchers have made efforts to recognize English Braille documents. The work of Ritchings et al. [45] is one of the first approaches to use commercially available flat-bed scanner to English Braille document recognition. It applied for both single and double-sided Braille documents, scanned at 100dpi at 16 grey levels.

The system proposed by Ritchings et al. [45] follows a series of four steps: image acquisition, identification of Braille dots, segmentation and recognition of characters. After the Braille document is scanned, the system proceeds to identify the locations of the protrusions and the depressions. This can be achieved by exploiting the differences in grey levels in the image. These arise during scanning from the reflected light and the shadows created by the protrusions and the depressions on the document surface. It uses simple rules to distinguish between Braille dots on the two different sides. Then, the dots of each side are grouped together to form characters which are then recognized and encoded [45].

According to Ritchings et al. [45] it is possible to see distinct zones in the represented points, as this is a partial scanned test image. One zone is brilliant, above the point, and

the other zone is darker, under the point [45]. This is due to the emboss of the points, illuminated by the oblique light source from the scanner. This discrepancy has to be enhanced to locate the points more precisely, reducing the probability to detect false points, introduced by eventual noise in the image.

In the segmentation module of Ritchings et al. [45] characters are identified based on the location of the lines of the Braille character and average height of the text line. Having located the lines of the Braille, segmentation is performed based on the analysis of the horizontal spacing between columns within dots. The segmented characters are then recognized base on the features extracted [45].

The proposed system in Ritchings et al. [45] performs few image-based operations and it is relatively flexible to skew as it identifies Braille characters based on character-region search. Results reported for double-sided Braille documents were of just over 98.5% of the protrusions and 97.6% of the depressions on average. As indicated in Ritchings et al. [45] the errors that occurred in the experimentation are mainly due to the following three factors: the existence of naturally occurring reflections and shadows other than those created; the identification of dots can be influenced by the size of merged regions in the image of inter-point Braille; the position of the dots of a character in the image.

The authors recommend that Braille characters do not have between them distinct differences in shape does not enhance recognition (as in the case of OCR). If there are one or more dots at valid positions then the corresponding character will be recognized. It is difficult to assess whether there is an extra dot or whether a dot is missing. Hence, on a single character basis, it is not possible to assess the correctness of the recognition result. However, it is possible to recover from some errors by performing contextual analysis.

In addition, the majority of the errors can be attributed to the quality of the image of the Braille document. Also, it should be pointed out that the quality of the Braille document itself is important. Very old documents with some of the protrusions flattened due to heavy use will give rise to more incorrectly recognized characters. Experiments include scanning at different resolutions and greater grey level ranges are important to improve

the performance of the system. Also considered is scanning of both sides of an inter-point Braille document and correlation of results.

In 1999 Ng. et al. [12] proposed a system that translates Braille characters to their equivalent English characters. The proposed translation system consists of four major steps: image capturing, image preprocessing, feature extraction and translation. In Ng et al. [12] shape from shading illuminating method for image capturing was considered. This method helps to handle the shadows of embossed dots from one side which fall on dots of the other side for double sided Braille documents. Digital camera was used as a device for capturing the Braille pages.

In Ng. et al. [12] preprocessing consists of two sub operations: noise filtering and edge enhancement. Noise filtering is achieved via a low pass Gaussian filter. According to [12] Gaussian is a low-pass spatial feature which seeks to attenuate the high spatial frequency noise from the while at the same time preserving the detailed edge information of the Braille dot. Edge detection is achieved using convolution Sobel kernels. Edge enhancement is used to sharpen fine details of the image that has been blurred [12].

For feature extraction purpose, authors [12] used Chain Code algorithm to detect boundary coordinates. Feature extraction is used to extract Braille dots from the binarised image. The system proposed by the authors has four distinct activities of feature extraction: boundary detection to highlight the dots; back-face removal to remove the dots embossed on the back side; centeroid determination to locate the central point of the dots and dot alignment to align the extracted dots with the coordinate system.

For translation purposes, authors [12] determined centroid distances between each dot and its four possible neighbors. Dots are then grouped into cells; within each cell the dot pattern is determined and represented by a bit string. The bit strings of the cells are searched against the Braille dictionary and the retrieved characters are grouped into words. Based on boundary coordinates, information and illumination characteristics, two standard templates were then constructed to represent the front-face dots and back-face dots.

The system proposed by Ng. et al. [12] takes the advantage of regular spacing between Braille dots within a cell, and the regular spacing between cells. Results were encouraging: 100% accuracy for single-sided and 97% accuracy for double-sided Braille pages was registered. The system is also capable in handling documents which can be embossed on a range of media with different specifications.

Apostolos et al. [4] developed a system that works on both single-sided and inter-point (double-sided) Braille documents. In the case of double-sided documents, the Braille characters on both sides are recognized from the image of only one side of the page. The system comprises: image acquisition, pre-processing, grid formation, Braille dot recovery, Braille character recognition, Braille character interpretation and character correction.

The approach in Apostolos et al. [4] used an inexpensive flat-bed scanner to capture an image of a Braille document page. The system uses context at different levels (from the pre-processing of the image through to the post-processing of the recognition results). A preprocessing step has been conducted to reduce the grey levels in the image and then the image is thresholded so that only three classes of regions exist: dark, light and background. Having labeled each of the different types of regions, an initial identification of Braille dots is performed. A flexible grid of possible dot locations is then constructed and any dots that were not previously detected are recovered. Braille characters are subsequently recognized and they are translated into the equivalent printed text. Finally, using the interpretation, post-processing is performed to correct wrongly recognized Braille characters.

Based on experimental results the authors reported that a scanning resolution between 80dpi and 200dpi is the most appropriate for Braille documents and the authors tested their system with images in the full range. Over 99% accuracy reported on both single and double-sided documents of average quality.

The paper presented by Wong et al. [28], in 2004 at University of Auckland, New Zealand, proposed an OBR system that is capable of recognizing a single sided Braille

page. In addition the system preserves the format of the original document in the produced text file as well as the efficiency and accuracy of the recognition algorithm. The algorithm processes the image one row at a time reducing the computation time significantly. The proposed system consists of three major steps: half-character detection to determine the where about of the character by detecting the possible dot positions; half-character recognition to determine the half character that the dots represented by using a probabilistic neural network and grid determination using transcription algorithm to produce a Braille text file where the format is preserved.

The half-character detection module in [28] is capable of handling one row at a time rather than as a whole image. During this process the row of pixels is changed into a two-tone image to distinguish the pixels that are likely to be part of the shadows of the dots from the others. The authors pointed out that when a row of pixels is found that contains one or more characters, it was buffered until the entire pixels from that row of characters are stored in the buffer. At this stage, the columns of the buffered image will be processed in a similar manner to determine the position of the half-characters. The result from this process will be two-tone images and the position of the character [28].

The recognition module in [28] is designed to work with thresholded images resulting from the half-character detection module. The half-characters are processed and classified as one of seven possible arrangements. The seven arrangements come from having three possible dot positions in each half-character, without considering “empty half character”. The classification process is carried out using a probabilistic neural network.

After the half-characters detected the position of the original parent whole characters is determined in the text file transcript module [28]. This stage uses the fact that Braille format has very strict rules on the spacing between dots within a character and between neighboring characters; these rules can be used to help determine the positions of the characters in the document. By looking at the distances between the dots, a grid of all possible positions was determined by the authors to calculate the position of the characters in the document.

The results of the experiment made by Wong et al. [28] were 99%, which were promising. The fact that the recognition algorithm does not involve convolution of a mask of any kind has shortened the processing time required to pre-process the image before the recognition phase. The authors also suggest that the recognition rate for the half-character recognition has room for improvement. One way to improve it is to implement a syntactic spell check algorithm for post processing.

2.6.2. Optical Braille recognition System for Spanish Characters

The system, OCR for the Braille code, developed by Hermida's et al. [23] passed through five major tasks: scanner control to get the input images; processing of the scanned images to locate the positions of the points that make up the Braille text; processing of the found points to get the lines and columns in the text; application of Braille alphabet to convert the located points into an ASSIC text and finally design of user interface.

Hermida's et al. [23] system employed thresholding before the image is passed on to the Braille dot extraction module. The scanned images contain mountains that had a brilliant zone above the point and a dark zone below it. Their algorithm converts a digital image of a scanned Braille page into one consisting mainly of black and white spots denoting the dots. The thresholds used were adaptively calculated from the histogram of the input image. These threshold values are found as points that leave above/below a given percent of the total area of the histogram.

The localization and extraction algorithm developed by Hermida et al. [23] takes a threshold image consisting of couples of white spots above and black spots below, where each couple denotes a single Braille dot. Though this method is easy and quick, it suffers from two problems: some legitimate points are lost and false ones are produced. These errors may arise due to spots that are very near to each other and that get joined after the threshold process. To solve these problems two strategies were suggested by the authors: mesh detection and intuitive method of working with the spots in proving other pattern recognition methods.

The mesh detection algorithm proposed by Hermida et al. [23] used to detect the positions and directions of the lines and columns of the text. These positions make up the Braille mesh. If we have N characters, the distance between them is normalized. This algorithm detects the entire mesh by going over it and inspects the presence of spots near each potential point. With this algorithm it is also possible recovering the lost points. The first stage of the algorithm is the detection of the skew angle afterwards the algorithm chooses a starting point and goes moving the standard distances to construct the entire mesh. Hermida et al. [23] considered that the key fact at this stage is to avoid choosing a false point as the starting one. The false points are points that don't observe the distance rules. Another important issue is to detect the relative position of the starting point in its character. Their algorithm assumes all the possible positions and then use the method of maximal counts (explained above): first determine to which column the point belongs to and then to which row. The final stage is the construction of the entire mesh beginning from the starting point. This movement is accomplished in an adaptive fashion.

The translator presented by Hermida et al. [23] takes as an input the image produced from the dot extraction module, where characters are represented as a group of dots, each dot is in turn represented by a single bit, with 1 or 0 values. Using this representation, the Braille-to-ASCII conversion is accomplished.

2.6.3. Optical Recognition of Braille Writing Using Standard Equipment

An attempt was made by Mennens et al. [32] to develop a system that run on commercially available scanners that are of moderate cost and easy to maintain (standard equipment) for both single- and double sided-character recognition. The authors addressed the problem of false shadows in the image caused by the fact that Braille pages are never perfectly flat due to the tension in the paper's surface, by subtracting a locally averaged image from the original. To align the calculated grid with the main axes of the text the authors rotate skewed images by using the deviation over a vertical projection of the image. They employed this strategy because it has certain advantages in the organization of the data [32].

The authors also used deviation over the sum of rows rather than Discrete Fourier Transform (DFT) to calculate the rotation angle. Their reason is the fact that image's structure causes important and recognizable data peaks in the DFT image to lie too close to the origin and this gives an error on the calculated rotation angle. In the case of deviation over the sum of rows, the image was slanted over an angle. Each time the image is slanted one pixel in the vertical direction, deviation over the sum of rows is calculated and a maximum was obtained when dots are aligned horizontally.

The approach adopted by Mennens et al. [32], represents dots in the digitized Braille page, with light and dark areas by making a three value image. For extracting Braille dots their approach is based on several assumptions, one of which is that a single dot is represented by two gray level intensities, a light area right above a dark one. Dots are located using image-based operations (performing correlation with a particular mask) and this mask is an absolute simplification of a Braille dot. On their approach the vertical size of the mask depends on the size of Braille dots and equals the distance between the center of the dark and light area of an average. Also their system is designed to recognize recto and verso dots (double-sided Braille page). As a result two vertically neighboring dots of the same side may produce false core regions (ghost points) for the other side.

To identify Braille characters, a grid is placed on the image where Braille dots are expected to be. To calculate grids Mennens et al. [32] approach searches grid lines by making histograms of rows and columns in five value ranges (-2,-1,0,1,2). To reduce computer time and provide extra protection against other disturbances first histograms for rows calculated and followed by calculating histograms for columns.

Mennens et al. [32] adopted Binary Braille cell sets as basis for their classifier, which is grouping the dots and representing each dot by a bit position. They tested algorithms on both single-sided and double-sided Braille, printed on carriers with different textures and colors. Results reported are 99.75% correct Braille character recognition on documents without major defects but the approach fails when distortions are present (due to the fixed grid).

Mennens et al. [32] suggested that to convert a text containing Braille characters containing Braille characters at maximum speed it is better to use commercially available scanner and a set of fairly calculation methods. Moreover, they pointed out that the most important characteristics of all their work is to work on local basis without losing contact with the global context and try to postpone complex calculations until the data set has been significantly reduced.

2.6.4. Image Processing Techniques for Braille Writing Recognition

Nestor [36] creates a database which contains both single and double sided documents. They acquired the Braille image using a flat-bed scanner with images of different resolution than 100 dpi since it uses interpolation methods to resize the input image. During image pre-processing the following tasks were performed in Néstor [36]: dynamic thresholding, pattern detection, Braille grid creation and dot recovery using Braille grid. They also apply iterative thresholding algorithms for classifying the gray scale image into black and white and an adaptive algorithm in order to make mesh from the detected dots. The adaptive algorithm builds columns in a first stage: since distances between points are normalized [36], the process begins searching for groups of dots in the same vertical plane that respect these distances. Based on this the protrusion and depression areas are identified.

To determine skew angle of the scanned documents the authors in [36] apply horizontal histograms and mass center calculations. Additionally to identify dots that are missed in the identification of protrusions and depressions, they adopted mesh grid algorithm. After mesh building, all valid Braille positions are known. Those intersections between rows and columns define a valid position for a Braille dot [36].

In addition the system [36] checks the potential positions of dots and recovers original dots that belong to correct Braille positions and discriminate false dots that are out of the valid places for Braille dots. Based on the mesh the finalized image is analyzed and text is segmented in rows and characters. Every character is then converted into binary number according to the active dots in the image. This way of Braille text binarization

makes the global system independent of the language of the document and easily configurable for different alphabet [36].

The developed system in [36] used global image processing algorithm which is very fast and robust. Results of the system are encouraging and perform an accuracy of 99.9% for double sided Braille documents.

2.6.5. Optical Braille Recognition System for Arabic Characters

AbdulMalik et al. [2] attempt to develop algorithms to recognize image of Arabic Braille embossed on both single-sided and double-sided material obtained by regular flat-bed scanner. Two main tasks are performed: the first is recognition of the printed Braille cell and the second is converting the Braille in to a text with simple one-to-one mapping. The proposed Arabic OBR system comprises the following stages: Image acquisition, converting the image to gray level, cropping the image frame, image thresholding, image de-skewing, dot part detection, Braille cells recognition and correcting the feature layout.

The study in [2] tried to minimize the effect of black or white frame in the image on thresholding by cropping the image frame. For cropping purpose average gray level of the whole image is calculated and 15% of the whole image above or below of the average gray level was considered an indication for deletion. Based on the threshold values of the cropped image the Braille pages are classified into foreground (dark) and background (white), to do this, the average gray level is calculated for the whole image and then for each of the rows and columns separately,. A binary search algorithm was developed and used to correct skew in tilted scanned document images. De-skew image tilted 4 degree from left or right is recognized.

AbdulMalik et al. [2] preferred the detection of Braille dot parts than the whole dot. In addition, they discussed recto dot-bright comes at the top of the dark and the contrary results as verso dot. They suggest two strategies for correcting errors that arise at this step: local measure and global measure. For dot detection global measure showed better results.

Having all possible dots, AbdulMalik et al. [2] defines the region containing all the dots by adding row and column in the array separately. In doing so, two algorithms are evaluated. The one which does not depend on fixed length for cells heights and distance between them is selected. In addition for correcting reversed paper layouts the authors [2] have used alphabet determination percentage on each page, 85 % and above showed the page is on the right layout but below 45% showed the page is on the reverse layout.

The proposed system in [2] is tested using several Braille documents -- skewed, reversed or worn-out-, written in Arabic language and scanned with different scanners. For both single-sided and double sided documents 99% of the dots on the Braille documents are correctly recognized.

2.6.6. Optical Braille Recognition System for Amharic Characters

Even though there are a number of attempts made to facilitate two way communication and knowledge transfer between visually impaired and vision society on different languages, only two attempts are made to developed Amharic OBR systems for Amharic Braille documents. Both researchers developed Amharic Braille recognizer only for single-sided Braille documents using neural network classifier.

The system proposed by Teshome [49] is the first attempt to develop Amharic Braille recognizer. The researcher collected both manual and typewritten single-sided Braille documents. The Braille documents are digitized using a flat-bed regular scanner with 200 dots per inch for both colored and gray scaled Braille images. His research adopted different Braille image recognition techniques: image acquisition, binariazation, segmentation, feature extraction and recognition for recognition of Braille characters and neural network classifier had been trained with Amharic character to translate the Braille character to printable character.

The binariazation technique is used in identifying the foreground (black color) or content from the background (white color) [49]. In doing so for global threshold a single threshold calculated from the total intensity and local threshold is calculated by taking into account the pixel of sub-images area. Even if both local and global threshold are

used for identification purpose, only global threshold were used due to computational expensiveness of the local threshold. The threshold value had been set on the color frequency through experimentation. The result of binarization has the feature that the foreground is represented by black color and background with white color.

The image segmentation operation separates dots from the Braille image that can further be grouped into a cell. These cells also grouped to form characters, words or any storks in the Braille document. Segmenting the Braille image can be performed in two ways: grid mesh construction and display the dot with single white pixel on black background. The researcher [49] prefers mesh-grid for segmentation purpose; this is because of the fact that Braille cells arranged strictly following vertical and horizontal layouts of the documents. The segmentation is performed in two steps: first gridlines mesh was constructed and then dots are searched following the grids mesh with thresholding. Both global and adaptive mesh techniques had been tested for segmentation purpose.

Feature extraction and recognition process identifies the representation mechanism of Braille image. For feature extraction, Teshome [49] adopted modified region based approach. Each character described in terms of dot positions and the six possible positions of the dots are known for each individual Braille character in the image. Then content analysis is performed to recognize the cells as part of Braille character based on rules defined. MATLAB implementation of the feed forward artificial neural network had been trained on the extracted features to create a model which can be used for Braille characters recognize.

Experimental results show that the system developed by Teshome [49] registers 92.5% recognition performance as a model for the system among the three models. The performance of the system is decreased with increase in the test set.

The author recommended that since the current system sets threshold for binarzation with trial and error, it is important to devise automatic thresholding system that may work with different image color. Further effort is also important towards creating Braille digital library, retrieval system and Braille dictionary. The other way to improve the

performance of the system is applying the neural network in proceeding steps such as future extraction.

The other effort made on Amharic Braille recognition system is the work of Ebrahim [16] which is an extension of [49]. The work of Ebrahim was aimed at exploring different noise detection and removal algorithm in order to enhance the performance of the Amharic Braille recognition system (designed by [49]) in real life Braille document images.

Pre-processing of document images is used specifically for noise reduction and smoothing of background area of the Braille document. According Ebrahim the pre-processing operation consists of four steps: histogram equalization, noise filtering, image binarization/threshold and morphological operations.

Ebrahim [16] used Gaussian filtering for noise reduction introduced at time of acquisition. The Image binariazation pre-processing operation converts gray scale images into a binary image aiming to distinguish text areas from background areas. For image binariazation sub operation global threshold had been used by the author. To remove some noise dots that are crated at the time of binariazation and to have better recognition performance morphological operations was performed.

Ebrahim [16] integrated the preprocessing scheme (Gaussian noise filtering followed by morphological operations) to the previously designed OBR system by Teshome [49]. Then he tested the performance of the proposed OBR system on real life Amharic Braille documents. Accordingly, the performance rate for clean Braille, small noisy Braille, medium level noisy Braille and high level noisy Braille using Gaussian filtering technique are 95.5%, 95.5%, 90.5%, 65.5%, respectively. The performance of the system shows that the accuracy decreases as level of noise in Braille documents increase. This initiates to design a feature extraction and classification scheme that tolerate the challenge posed by real life Braille images. It is therefore the objective of the present study to explore feature extraction and classification algorithms suitable for Amharic Braille document images.

CHAPTER THREE

BRAILLE RECOGNITION TECHNIQUES

The process of identifying and recognizing Braille characters is to some extent of different nature than that of reading printed characters. Yet, there are many interesting parallels that can be drawn. Braille documents occupy considerably greater volume than printed ones (40 to 60 % more) for recording the same information. This is because the thickness of the material, the length of the protruding dots and the low information recording density resulting from the relatively large spatial distribution of Braille characters [52]. The process of Braille document recognition passes through many stages starting from Braille acquisition to recognition of characters.

3.1. Design of Amharic OBR system

This research is concerned with designing different Braille image recognition techniques for Amharic Braille documents. The overall architecture of Amharic Braille recognition system is shown in Figure 3.1. In translating Amharic Braille code characters to their equivalent Amharic representation a series of steps are required. As depicted in Figure3.1, Braille recognition techniques labeled with double rectangle are the main focuses of this study.

3.2. Image Acquisition/Digitization

In optical Braille recognition (OBR), digitization enables to produce a digital image of the scanned document in the form of bitmap. In optical Braille recognition (OBR), digitization enables to produce a digital image of the scanned document in the form of bitmap. Acquiring images from Braille documents can be done using scanner or digital camera [2][4][11]. The mechanism used for image acquisition in this study is a flat-bed scanner instead of digital camera. This is because flat-bed scanner is a cheap alternative which can be used for so many other applications and easy and quick to use. The system is also able to work with images of different resolution ranging from 100 to 600 dpi.

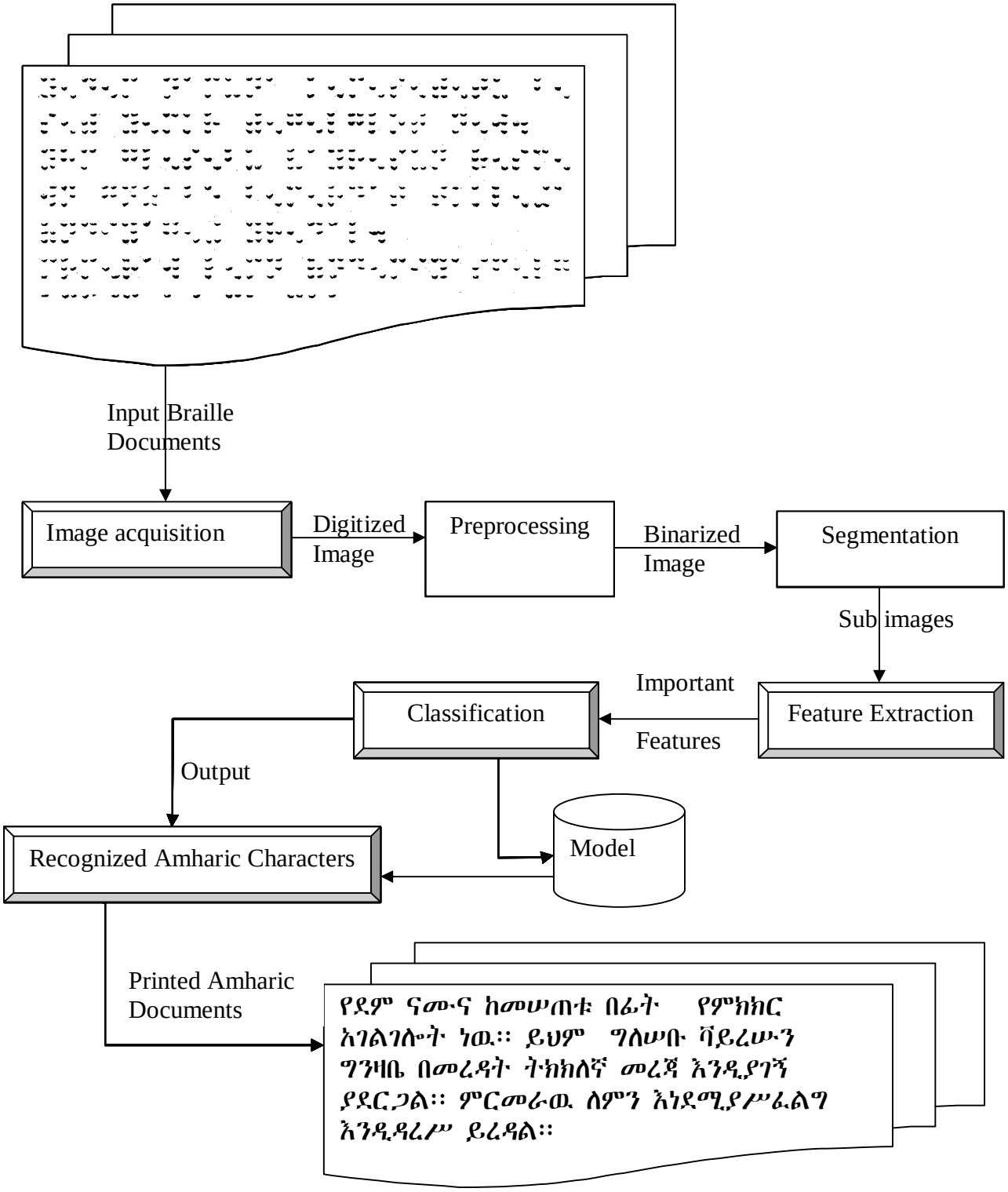


Figure3. 1 Block Diagram of Amharic Braille Recognition System

3.3. Feature Extraction

Feature extraction is a representational mechanism of the Braille image [49]. The main function of this process is to extract the Braille dots from the binarised image. The image capturing, preprocessing and segmentation stages aim to make the image be suitable for different feature extraction algorithms. Some feature extraction algorithms only deal with the contours of the image while some algorithms calculate every pixel of the image. On the other hand, the initial image may be noise affected, or blurred by other reasons.

Extraction of good features is the key to correctly recognize an unknown character. A good feature set contains discriminating information, which can distinguish one object from other objects. It must also be as robust as possible in order to prevent generating different feature codes for the objects in the same class. The selected set of features should be a small set whose values efficiently discriminate among patterns of different classes, but are similar for patterns within the same class.

The selection of image features and corresponding extraction methods is probably the most important step in achieving high performance for an OCR system [39]. At the same time, the image feature and the extraction method also decide the nature and the output of the image-preprocessing and segmentation steps. Some image features and the extraction algorithms work on color images, while others work on gray level or binary images. Moreover, the format of the extracted features must match the requirements of the classifier [26]. Some features like the graph descriptions and grammar-based descriptions are well suited for structural and syntactic classifiers. Numerical features are ideal for statistical classifiers. Discrete features are ideal for decision trees. As mentioned in [34] there are different methods used to extract features. Some are highly language specific like profiles, structural descriptors and transform domain representations and others consider the entire image as the feature.

There are different feature extraction algorithms for Optical Braille recognition system. The feature extraction algorithms implemented in this study are described in the following sub sections.

3.3.1. Fixed Cell Measures

This algorithm assumes that the average line and column width shown in Figure 3.2 below. Having decided the average line and column width, it is important to determine the number of rows and columns in the defined region. To calculate the number of rows and columns the following two equations are used:

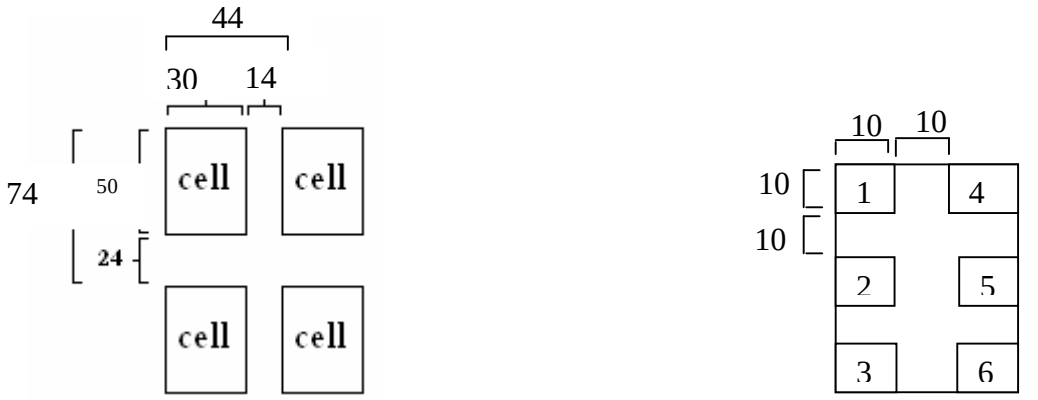
$$linNum = (yMax - yMin) / 50 \quad (3.1)$$

$$colNum = (xMax – xMin) / 30 \quad (3.2)$$

Then we can reach the beginning of any cell by using the line and column number, as in the following equations:

$$i = yMin + (lin-i) * 74 \quad (3.3)$$

$$j = xMin + (col-j) * 44 \quad (3.4)$$



(a) average cell measures **(b) one cell measures**

Figure3. 2 Cell measures

In order to convert the cell to binary code, the region within one row and column (a cell) is divided into six regions as in Figure3.2 (b). Then taking each region separately, if the sum of its elements is more than the average Braille dot pixels then it assigns the number 1, otherwise it assigns 0. The detail of this technique is shown in Algorithm3.1 below:

Algorithm3. 1 Feature extraction algorithm based on fixed cell measures

```
1.  while j< the beginning of the cell plus the cell width measure do
2.      while i< the beginning of the cell plus the cell length measure do
3.          for p=i to i+α do
4.              for s=j to j+ α do
5.                  if there is black pixel at (s,p) then
6.                      dotcount++
7.                  End if
8.              End for
9.          End for
10.         if dotcount>=β then
11.             dd 1 to array
12.         else
13.             add 0 to array
14.         End if
15.     End while
16. End while
```

3.3.2. Horizontal and Vertical Projection

This algorithm does not depend on fixed lengths for cells heights and distances between them. To work on this algorithm, first it is important to identify all valid Braille dots by adding each row and column in an array separately. Having identified all possible valid Braille dots, the system defines the region containing all the dots such that no dots exist outside this region. For the valid Braille dots it is important to determine the horizontal projection and the vertical projection. After determining the horizontal and vertical projections, we can reach any cell by considering any consecutive 3 rows and 2 columns as a cell. The steps involved in this technique are described as follows in Algorithm3.2:

Algorithm3.2 Feature extraction algorithm based on horizontal and vertical projection

1. Identify all valid Braille dots in the image and add each row and column in the array separately
2. Determine the horizontal projection array (DotHorProj[])
3. Determine the vertical projection array (DotVerProj[])
4. While DotHorProj[j]) do
5. While(DotVerProj[i]) do
6. for w=i to i+2 do
7. for h=j to j+3 do
8. val=FALSE
9. While valid Braille dot array
   If valid Braille dot row value equal to the horizontal projection and valid Braille dot column value equal to the vertical projection then
10. Add 1 to the array
11. val=TRUE;
12. End if
13. End while
14. If val==FALSE then
15. Add 0 to the array
16. End if
17. End for
18. End for
19. End while
20. End while

3.3.3. Grid Construction

Braille writing cannot be processed with standard optical character recognition (OCR) software. This is due to the fragment nature of the character and hence a different approach has to be considered. Using the property that Braille characters are always positioned on a fixed matrix, the authors in [11][49] first tried to build a grid consisting of horizontal and vertical lines that run through all the dots and then they checked if

there is a dot present on each of the intersection point. The grid construction must be flexible because there are cases where it can be deformed or irregular.

Although Braille dots are placed on a fixed matrix, certain Braille production techniques cause this matrix irregular. Two major types of deformation problems can be distinguished [11].

a. Deformation of the Braille cell

    - Spacing between the dots and the cells may vary in both directions
    - Degradation of the dots

b. Deformation of the grid where Braille characters are positioned

    - The page went askew while the text was being typed
    - The text's layout sometimes causes large gaps in the text, and as a consequence, in the calculated grid

To handle the above two problems the following strategy is proposed. First it is conceivable to create a mask for a standard Braille dots. This mask could then be convolved with image, resulting in peaks on the position of the Braille dots. These peaks then have to be grouped in cells to convert them to characters. This means that we also have to find the grid on which dots are positioned.

Using a very simple mask, we first try to find the position of the grid lines. In order to cope with problem 2, the image is divided in sub-images and grids will be calculated for each sub image separately (local linearization of the deformations). Grids of horizontally and vertically adjacent sub-images are then taken together in horizontal and vertical strips: horizontal strips contain vertical grid lines; vertical strips contain horizontal grid lines. The lines in a strip will then be grouped in atoms. An atom is a group of lines that belongs to a single Braille cell. The grouping in atoms has to be flexible because lines are not at fixed distance (problem 1). Finally, the relationship between the strips is restored and at the cross sections of horizontal and vertical atoms Braille characters can be found. The details of this algorithm are summarized in Algorithm3.3 below.

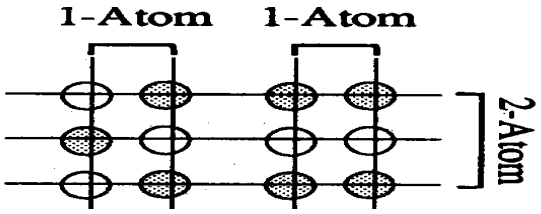


Figure3. 3 Example of atoms in a six-dot Braille cell

One pair of lines is called 1-atom; two pairs of lines are called a 2-atom.

Algorithm 3. 3 Feature extraction algorithm based on grid construction

```
1.  Get each sub-images in the form of arrays
2.  while(col[h][2]) do// for each  2-atom
3.      while(line[w][1]) do //for each 1-atom
4.          for d=line[w][i] to line[w][i]+α do
5.              for g=col[h][j] to col[h][j]+α do
6.                  if GetPixel(d,g) is black pixel then
7.                      dotcount++
8.                  End if
9.              End for
10.         End for
11.         if dotcount>=β then
12.             add 1 to an array
13.         Else
14.             add 0 to an array
15.         End if
16.     End while
17. End while
```

3.4. Classification

Classification can be described as a supervised learning algorithm in the machine learning process [66][69]. It assigns class labels to data objects based on prior knowledge of class which the data records belong. Classification, which is the task of assigning objects to one of several predefined categories, is pervasive problem that encompasses many diverse applications.

As shown in Figure3.4 below, classification is the task of learning a target function *f* that maps each attribute set to one of the predefined class labels. The target function is also known as a classification model.

Figure3. 4 Classification as the task of mapping an input set into class label

In classification, a given set of data records is divided into training and test data sets [[19] [20]. The training data set is used in building the classification model, while the test data record is used in validating the model. The model is then used to classify and predict new set of data records.

Supervised learning algorithm (like classification) is preferred to unsupervised learning algorithm (like clustering) because its prior knowledge of the class labels of data records makes feature/attribute selection easy and this leads to good prediction/classification accuracy [29]. Some of the common classification algorithms used in data mining and decision support systems are: decision tree classifier, rule-based classifier, neural networks, support vector machines and naïve Bayes classifier. Among these classification algorithms decision tree algorithms is the most commonly used because it is easy to understand and cheap to implement. It provides a modeling technique that is easy for human to comprehend and simplifies the classification process [50].

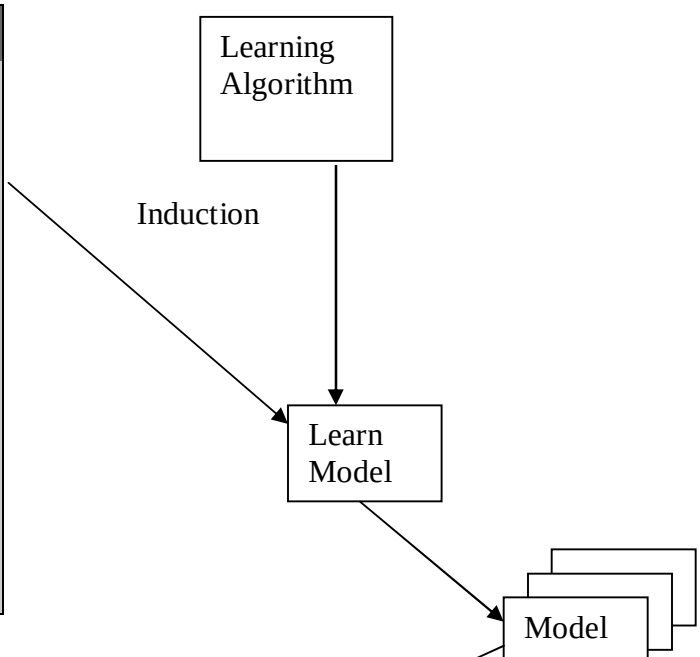
3.4.1. General Approaches for Solving a Classification Problem

A classification technique (or classifier) is a systematic approach to build classification models from an input data set. The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records it has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability; i.e., models that accurately predict the class labels of previous unknown records.

Figure3.5 shows a general approach for solving classification problems. First, a training set consisting of records whose class labels are known must be provided. The training set is used to build a classification model, which is subsequently to the test set, which consists of records with unknown class labels.

Training Set

TId	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125k	No
2	No	Medium	100k	No
3	No	Small	70k	No
4	Yes	Medium	120k	No
5	No	Large	95k	Yes
6	No	Medium	60k	No
7	Yes	Large	220k	No
8	No	Small	85k	Yes
9	No	Medium	75k	No
10	No	Small	90k	Yes



Testing Set

TId	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55k	No
12	Yes	Medium	80k	No

Figure3. 5 General approaches for building a classification model

Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a confusion matrix. Table 3.1 depicts the confusion matrix for a binary classification problem. Each entry f_{ij} in this table denotes the number of records from class i predicted to be of class j. For instance, f_{01} is the number of records from class0 incorrectly predicted as class1. Based on the entries in the confusion matrix, the total

number of correct predictions made by the model is (f_{11} + f_{00}) and the total number of incorrect predictions is (f_{10} + f_{01}).

		Predicted Class	
		Class=1	Class=0
Actual Class	Class=1	f_{11}	f_{10}
	Class=0	f_{01}	f_{00}

Table3. 1 Confusion matrix for a 2-class problem

Although a confusion matrix provides the information needed to determine how well a classification model performs, summarizing this information with a single number would make it more convenient to compare the performance of different models. This can be done using a performance metric such as accuracy, which is defined as follows [55][67]:

$$Accuracy = \frac{Number\ of\ \ correct\ \ predictions}{Total\ \ number\ \ of\ \ preditions} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} \tag{3.5}$$

Equivalently, the performance of a model can be expressed in terms of its error rate, which is given by the following equation:

$$Error\ rate = \frac{Number\ of\ \ wrong\ \ predictions}{Total\ number\ of\ \ predctions} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}} \tag{3.6}$$

Most classification algorithms seek models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set.

3.4.2. Decision Tree Classifier

Decision tree is a hierarchical structure consisting of nodes and directed edges [43][55]. The tree has three types of nodes: A root node that has no incoming edges and zero or more outgoing edges; internal nodes, each of which has exactly one incoming edge and two or more outgoing edges; leaf or terminal nodes, each of which has exactly one incoming edge and no outgoing edges. In a decision tree, each leaf node is assigned a

class label. The non-terminal nodes, which include the root and other internal nodes, contain attribute set conditions to separate records that have different characteristics.

Decision tree classification technique is performed in two phases: tree building and tree pruning. During tree building the tree is recursively partitioned till all the data items belong to the same class label [24]. Tree pruning is used to improve the prediction and classification accuracy of the algorithm by minimizing over-fitting (noise or much detail in the training data set) [30]. Over-fitting in decision tree algorithms result in misclassification error. Tree pruning is less tasking compared to the tree growth phase as the training data set is scanned only once.

Decision Tree Classifier (DTC) is used successfully in many diverse areas such as radar signal classification, character recognition, remote sensing, medical diagnosis, expert systems, and speech recognition, to name only a few. Perhaps, the most important feature of DTC is its capability to break down a complex decision-making process into a collection of simpler decisions, thus providing a solution which is often easier to interpret [43].

3.4.2.1. Design of a Decision Tree Classifier

The main objectives of decision tree classifiers are: to classify correctly as much of the training sample as possible; generalize beyond the training sample so that unseen samples could be classified with as high of an accuracy as possible; be easy to update as more training sample becomes available (i.e., be incremental) ; and have as simple a structure as possible. The design of a DTC can be decomposed into the following tasks [5][35]: The appropriate choice of the tree structure; the choice of feature subsets to be used at each internal node and the choice of the decision rule or strategy to be used at each internal node.

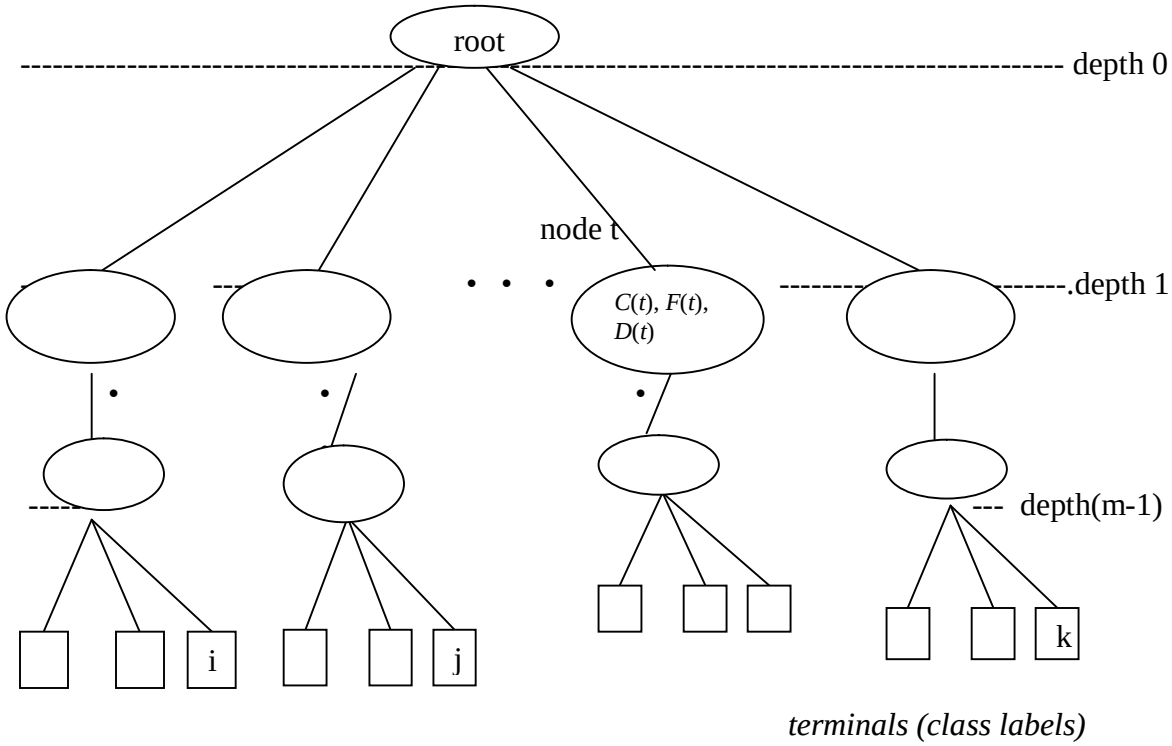


Figure3. 6 Example of a general decision tree

C(t) - subset of classes accessible from node t

F(t) - feature subset used at node t

D(t) - decision rule used at node t

3.4.2.2. Design issues of Decision Tree Induction

A learning algorithm for inducing decision tree must address the following two issues [55].

a. How should the training records be split? Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.

b. How should the splitting procedure stop? A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values. Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier.

Some of the common optimality criteria for tree design are [43]: minimum error rate, min-max path length, minimum number of nodes in the tree, minimum expected path length, and maximum average mutual information gain. The various heuristic methods for construction of DTC can roughly be divided into four categories: *Bottom-Up* approaches, *Top-Down* approaches, the *Hybrid* approach and tree *Growing-Pruning* approaches.

In bottom-up approach [21] one starts with the information classes and continues combining classes until one is left with a node containing all the classes (i.e., the root). In a bottom-up approach, a binary tree is constructed using the training set. Using some distance measure, pair-wise distances between a priori defined classes are computed and in each step the two classes with the smaller distance are merged to form a new group. The mean vector and the covariance matrix for each group are computed from the training samples of classes in that group, and the process is repeated until one is left with one group at the root. In a tree constructed this way, the more obvious discriminations are done first, near the root, and more subtle ones at later stages of the tree. It is also recommended [21] that from a processing speed point of view, the tree should be constructed such that the most frequently occurring classes are recognized first.

In top-down approach, where starting from the root node using a splitting rule, classes are divided until a stopping criterion is met. In this approach, the design of a DTC reduces to the following three tasks: the selection of a node splitting rule; the decision as to which nodes are terminal; the assignment of each terminal node to a class label. Of these three tasks, the class assignment problem is by far the easiest [43]. Basically, to minimize the misclassification rate, terminal nodes are assigned to the classes which have the highest

probabilities. These probabilities are usually estimated by the ratio of samples from each class at that specific terminal node to the total number of samples at that specific terminal node. Then this is just the basic majority rule; i.e., assign to the terminal node the label of the class that has most samples at that terminal node.

In Hybrid approach one uses both bottom-up and top-down approaches sequentially. The rational for the proposed method is that in a top-down approach such as hierarchical clustering of classes, the initial cluster centers and cluster shape information are unknown. It is also well known that the proper choice of initial conditions could considerably influence the performance of a clustering algorithm and this information can be provided by a bottom-up approach.

Tree growing-pruning approach used in order to avoid some difficulties in choosing a stopping rule; one grows the tree to its maximum size where the terminal nodes are pure or almost pure, and then selectively prunes the tree.

3.4.2.3. Methods for Expressing Attribute Test Conditions

Decision tree induction tree algorithm must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types [55]: binary attributes, nominal attribute, ordinal attributes and continuous attributes.

The test condition for binary attribute generates two potential outcomes. Nominal attributes can have many values and its test condition can be expressed in two ways: for a multi-way split, the number of outcomes depends on the number of distinct values for the corresponding attribute. On the other hand, some decision tree algorithms produce only binary by considering 2^{k-1}-1 ways of creating binary partitions of k attribute values. Ordinal attributes can also produce binary or multi-way splits. Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values. For continuous attributes, the test condition can be expressed as a comparison test ($A < v$) or ($A \geq v$) with binary outcomes, or a range query with outcomes of the form $v_i \leq A < v_{i+1}$, for i=1,…, k. for the binary case, the decision tree algorithm must consider all

possible split positions v, and it selects the one that produces the best partition. For the multi-way split, the algorithm must consider all possible ranges of continuous values.

3.4.2.4. Measures for Selecting the Best Split

There are many measures that can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after splitting. Let p(i|t) denote the fraction of records belonging to class i at a give node t. In a two-class problem, the class distribution at any node can be written as (p_0, p_1), where p1=1- p_0.

The measures developed for selecting the best split are often based on the degree of impurity, of the child nodes. The smaller the degree of impurity, the more skewed the class distribution [55]. For example, a node with class distribution (0, 1) has zero impurity, whereas a node with uniform class distribution (0.5, 0.5) has the highest impurity. Examples of impurity measure include [43] [55]:

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i \mid t) \log_2 p(i \mid t) \tag{3.7}$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} | p(i \mid t) |^2 \tag{3.8}$$

$$Classificatin\ Error(t) = 1 - \max_i \lceil p(i \mid t) \rceil \tag{3.9}$$

Where *c* is the number of class and p (i|t) is the relative frequency of class i at node t.

To determine how well a test condition performs, we need to compare the degree impurity of the parent node (before splitting) with the degree of impurity of the child (after splitting). The larger their difference means the better the test condition. The gain is a criterion that can be used to determine the goodness of a split:

$$\Delta = I(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j) \tag{3.10}$$

Where I (parent) is the impurity measure of a given node, N is the total number of records at the parent node, k is the number of attribute values, and N(v_j) is the number of records

associated with the child node, v_j. Decision tree induction algorithm often chooses a test condition that maximizes the gain, Δ. Since I(parent) is the same for all test conditions, maximizing the gain is equivalent to minimizing the weighted average impurity measures of child nodes. Finally, when entropy is used as impurity measure in equation 3.10, the difference in entropy is known as the information gain, Δ_{info}.

Impurity measures such as entropy and Gini index tend to favor attributes that have a large number of distinct values [55]. Even in a less extreme situation, a test condition that results in a large number of outcomes may not be desirable because the number of records associated with each partition is too small to enable us to make any reliable predictions. There are two strategies for overcoming this problem. The first strategy is to restrict the test conditions to binary splits only. This strategy is employed by decision tree algorithm such as CART. Another strategy is to modify the splitting criterion to take into account the number of outcomes produced y the attribute test condition. For example, in the C4.5 decision tree algorithm, a splitting criterion known as gain ratio is used to determine the goodness of a split. This criterion is defined as:

$$Gain\ Ratio = \frac{\Delta_{\inf o}}{-\sum_{i=1}^{k} p(v_i)\log_2 p(v_i)} \tag{3.11}$$

Here, k is the total number of splits. For example, if each attribute value has the same number of records, then $\forall i : p(v_i) = 1/k$ and the split information would be equal to $\log_2 k$. This example suggests that if an attribute produces a large number of splits, its split information will also be large, which in turn reduces its gain ratio.

3.4.2.5. Algorithm for Decision Tree Induction

Decision tree algorithm is a data mining induction techniques that recursively partitions a data set of records using depth-first greedy approach [24] or breadth-first approach [47] until all the data items belong to a particular class. In principle, there are exponentially many decision trees that can be constructed from a given set of attributes. While some of the trees are more accurate than others, finding the optimal tree is computationally infeasible because of the exponential size of the search space. Nevertheless, efficient

algorithms have been developed to induce a reasonably accurate decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy that grows a decision tree by making a series of locally optimum decisions about which attribute to use for partitioning the data. One such algorithm is Hunt's algorithm, which is the basis of many existing decision tree induction algorithms, including ID3, C4.5 and CART.

In Hunt's algorithm, a decision tree is grown in recursive fashion by partitioning the training records into successively purer subsets. Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2 \ldots, y_c\}$ be the class labels. The following is a recursive definition of Hunt's algorithm.

Step1: if all the records in D_t belong to the same class y_t, then t is a leaf node labeled as y_t.

Step2: if D_t contains records that belong to more than one class, an attribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D_t are distributed to children based on the outcomes. The algorithm is then recursively applied to each child node.

A skeleton of decision tree algorithm called TreeGrowth is shown in Algorithm 3.4. The input to this algorithm consists of the training record E and the attribute set F. the algorithm works by recursively selecting the best attribute to split the data (step 7) and expanding the leaf node of the tree (step 11 and 12) until the stopping criterion is met(step1). The details of Algorithm3.4 are explained below:

1. The createNode() function extends the decision tree by creating a new node. A node in the decision tree has either the test condition, denoted as node.test_cond, or a class label, denoted as node.label.
2. The find_best_split() function determines which attribute should be selected as test condition for spitting the training records. As previously noted, the choice of test condition depends on which impurity measure is used to determine the goodness of a split. Some widely used measures include entropy, the Gini index, and χ^2 statistic.

3. The Classify() function determines the class label to be assigned to a leaf node. For each leaf node t, let p(i|t) denote the fraction of training records from class i associated with the node t. in most cases, the leaf node is assigned to the class that has the majority number of records:

$$leaf\ \ f.label = \arg\max_i p(i \mid t)$$

Where the argmax operator returns the argument i that maximizes the expression p(i|t). Besides providing the information needed to determine the class label of a leaf node, the fraction p(i|t) can also be used to estimate the probability that a record assigned to the leaf node t belongs to class i.

4. The stopping_cond() function is used determine the tree-growing process by testing weather all the records have the same class label or the same attribute values. Another way to terminate the recursive function is to test whether the numbers of records have fallen below some minimum threshold.

Algorithm3.4 A skeleton of decision tree induction algorithm

```
TreeGrowth(E,F)
1: if stopping_cond(E,F)= true then
2:    leaf=createNode().
3:    leaf.label=Classify(E).
4:    return leaf.
5: else
6:    root=createNode().
7:    root.test_cond=find_best_split(E,F).
8:    let V={v|v is a possible outcome of root.test.cond}.
9:    for each v∈V do
10:   Ev={e|root.test_cond(e)=v and e∈E}.
11:   child=TreeGrowth(Ev,F).
12:   add child as descendent of root and label the edge (root→ child) as v.
13:  end for
14: end if
15: return root.
```

Decision tree algorithms can be implemented parallel or serial ways [29]. Parallel implementation tends to be scalable, fast and disk resident and can be implemented in computer architecture with many processors [47]. Serial implementation on the other

hand is fast, memory resident and easy to understand. There are various commonly used decision tree algorithms as discussed below:

IDE3 (Iterative Dichotomiser 3) decision tree algorithm: IDE3 algorithm was introduced in 1986 by Quinlan Ross [42] [43]. It is based on Hunt's algorithm and it is serially implemented. Like other decision tree algorithms, the tree is constructed in two phases; tree growth and tree pruning. Data is sorted at every node during the tree building phase in-order to select the best splitting single attribute [47]. IDE3 uses information gain measure in choosing the splitting attribute. It only accepts categorical attributes in building a tree model [42] [43].

IDE3 does not give accurate result when there is too-much noise or details in the training data set, thus an intensive pre-processing of data is carried out before building a decision tree model with IDE3.

C4.5 algorithm: This algorithm is an improvement of IDE3 algorithm [44]. It is based on Hunt's algorithm and also like IDE3, it is serially implemented. Pruning takes place in C4.5 by replacing the internal node with a leaf node thereby reducing the error rate [38]. Unlike IDE3, C4.5 accepts both continuous and categorical attributes in building the decision tree. It has an enhanced method of tree pruning that reduces misclassification errors due to noise or too-much details in the training data set. Like IDE3 the data is sorted at every node of the tree in order to determine the best splitting attribute. It uses gain ratio impurity method to evaluate the splitting attribute [44].

CART (Classification and regression trees): This algorithm was introduced by Breiman [9]. It builds both classifications and regressions trees. The classification tree construction by CART is based on binary splitting of the attributes. It is also based on Hunt's model of decision tree construction and can be implemented serially [9]. It uses gini index splitting measure in selecting the splitting attribute. Pruning is done in CART by using a portion of the training data set [38]. CART uses both numeric and categorical attributes for building the decision tree and has in-built features that deal with missing attributes [27]. CART is unique from other Hunt's based algorithm as it is also use for

regression analysis with the help of the regression trees. The regression analysis feature is used in forecasting a dependent variable (result) given a set of predictor variables over a given period of time [9].

SLIQ (Supervised Learning In Ques): SLIQ was introduced by Mehta et al [30]. It is a fast, scalable decision tree algorithm that can be implemented in serial and parallel pattern. It is not based on Hunt's algorithm for decision tree classification. It partitions a training data set recursively using breadth-first greedy strategy that is integrated with pre-sorting technique during the tree building phase [30]. With the pre-sorting technique sorting at decision tree nodes is eliminated and replaced with one-time sort, with the use of list data structure for each attribute to determine the best split point [30][47]. In building a decision tree model SLIQ handles both numeric and categorical attributes. It uses Minimum Description length Principle (MDL) in pruning the tree after constructing it. MDL is an inexpensive technique in tree pruning that the least amount of coding in producing tree that are small in size using bottom-up technique [3][30].

SPRINT (Scalable Parallelizable Induction Tree): This decision tree algorithm was introduced by Shafer et al [47]. It is a fast, scalable decision tree classifier. It is not based on Hunt's algorithm in constructing the decision tree; rather it partitions the training data set recursively using breadth first greedy technique until each partition belong to the same leaf node or class [3][47]. It is an enhancement of SLIQ as it can be implemented in both serial and parallel pattern for good data placement and load balancing [47]. Like SLIQ it uses one time sort of the data items and it has no restriction on the input data size. Unlike SLIQ it uses two data structures: attribute list and histogram which is not memory resident making SPRINT suitable for large data set, thus it removes all the data memory restrictions on data [47]. It handles both continuous and categorical attributes.

CART, SLIQ, and SPRINT use the *gini* index to derive the splitting criterion at every internal node of the tree.

J48: J48 is a decision tree induction algorithm which is an improved version of the C4.5 algorithm [71]. This algorithm generates decision trees using an information theoretic

methodology. The basic algorithm for decision tree induction is a greedy algorithm that constructs decision trees in a top-down recursive divide-and-conquer manner. The goal is to construct a decision tree with minimum number of nodes that gives least number of misclassifications on training data. According to [70] J48 decision tree algorithm is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. It can be applied on discrete data, continuous or categorical data.

The decision tree induction algorithm used in this study for classifying extracted Braille features is J48 algorithm. J48 decision tree can serve as a model for classification as it generates simpler rules and remove irrelevant attributes at a stage prior to tree induction.

It is important to understand the variety of options available when using an algorithm, as they can make a significant difference in the quality of results. In several cases, it was seen that j48 decision trees had a higher accuracy than other algorithms [69]. J48 offer also a fast and powerful way to express structures in data.

J48 employs two pruning methods [69]: sub tree replacement and sub tree raising. In sub tree replacement nodes in a decision tree may be replaced with a leaf -- basically reducing the number of tests along a certain path. This process starts from the leaves of the fully formed tree, and works backwards toward the root. In the case of sub tree raising a node may be moved upwards towards the root of the tree, replacing other nodes along the way. Sub tree raising often has a negligible effect on decision tree models. Many algorithms attempt to "prune", or simplify, their results. Pruning produces fewer, more easily interpreted results. More importantly, pruning can be used as a tool to correct for potential over fitting.

Algorithm3.5 J48 decision tree classifier algorithm

a. For training instances
1. Tree is constructed in top down recursive divide and conquer manner
2. Feature that is having the highest information gain is selected as root node of the decision tree.
3. Select the attribute that gives us the next highest information gain.
4. Repeat step 2 and 3 until reach from the root node to the leaf node

b. For test instances
1. Classify the new instance based upon this decision tree
Stopping criteria
1. All sample for a given node belong to the same class
2. If there is no remaining attribute for further partitioning

3.4.2.6. Model Over-fitting

After building the decision tree, a tree-pruning step can be performed to reduce the size of the decision tree. Decision trees that are too large are susceptible to a phenomenon known as over-fitting. Pruning helps by trimming the branches of the initial tree in a way that improves the generalization capability of the decision tree.

The errors committed by a classification model are generally divided into two types [55]: training errors and generalization errors. Training error, also known as resubstitution error or apparent error, is the number of misclassification errors committed on training records, whereas generalization error is the expected error of the model on previously unseen records.

A good classification model must not only fit the training data well, it must also accurately classify records it has never seen before. In other words, a good model must have low training error as well as low generalization error [55]. This is important because a model that fits the training data too well can have a poorer generalization error than a model with a higher training sample. Such a situation is known as model over-fitting [55][72][73].

The training and test error rates of the model are large when the size of the tree is very small. This situation is known as model under-fitting [55][72][73]. Under-fitting occurs because the model has yet to learn the true structure of the data. As a result, it performs poorly on both the training and the test sets. As the number of nodes in the decision tree increases, the tree will have fewer training and test errors. However, once the tree becomes too large, its test error rate begins to increase even though the training error rate continuous to decrease. This phenomenon is known as model over-fitting. Over-fitting and under-fitting are two pathologies that are related to the model complexity.

Models that make their classification decisions based on small number of training records are also susceptible to over-fitting. Such models can be generated because of lack of representative samples in the training data and learning algorithms that continue to refine their models even when few training records as available.

Tree pruning is done to improve the prediction and classification accuracy of the algorithm by minimizing over-fitting (noise or much detail in the training data set) [30].

3.4.3. Support Vector Machines (SVMs)

SVMs are a set of related supervised learning methods used for classification and regression. It is a classification technique that seeks to find a hyper plane that partitions the data by their class labels and at the same time avoid over fitting the data by maximizing the margin of the separating hyper plane. It makes a binary classification based on a separating hyper plane on a remapped instance space. As mentioned in [33] SVM has effective training and testing algorithms that are suitable for OCR problems with high dimensional input data.

SVMs learn from labeled examples from a training set including both positive and negative samples. A hyper plane which classifies the positive and negative data in the training set is found based on the attributes of data in the training set. Many such hyper planes which separate the data exist. The one that achieves maximum separation needs to be chosen and only one such hyper plane exists [47]. The data points nearest to the margin on both sides are called support vectors. From the data and its labels the SVMs

learn a mapping function. A kernel function, which is a dot product, is used in finding the hyper plane by remapping input feature vectors. Once the hyper plane is found, unlabeled examples from the test are classified based on the support vectors.

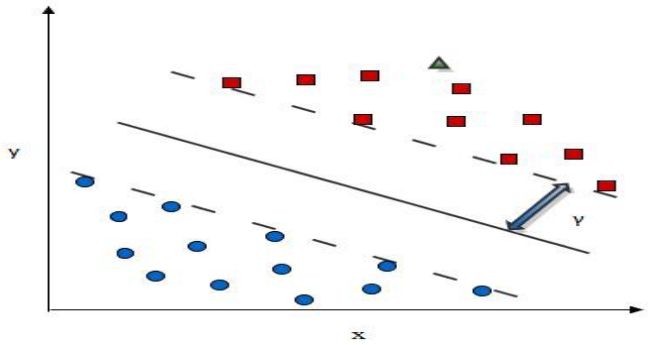


Figure3. 7 SVM for two class problem

In Figure3.7 above, a hyper plane classifies two classes of training data which is two dimensional. The two classes of data are represented by squares and circles. The bold line, which is separated from the closest training vectors by distance γ is the hyper plane. The classification of triangle, which is an unknown sample, is done by determining which side of the hyper plane it falls. In this example, the prediction for the unknown sample would be square.

The goal of SVM is to find the best boundary that separates the positive and negative samples, where 'best' means the boundary separates the negative and positive samples with the largest margin or 'distance'. The decision boundary must classify all points correctly and should prevent data points from falling into the margin. Let *n* be the dimension of the feature vector and $\{x_1,\ldots, x_n\}$ be the representation of a sequence x and let $y_x \varepsilon \{1,-1\}$ be the class label of x. The feature vectors of the training set are used to build the classifier. Weight vector w is of the same dimension as the feature vector and it is represented as $w = \{w_1,\ldots,w_n\}$. The label of the sequence is predicted as 1 if $w_x + b \geq 1$ and it is predicted as -1 if $w_x + b < 1$ where, b is a threshold.

Therefore,

i) if $y_x = 1; w_x + b \geq 1$

ii) if $y_x = -1; w_x + b < 1$

The equation of the margin γ_x is given by

$$\gamma_x = y_x(w_x + b)$$

The value of γ determines the accuracy of SVMs in predicting the label of the sequence. If γ is positive, it implies that the prediction is correct and a negative value of γ implies that the prediction is incorrect. The values of the weight vector w and threshold b are updated every time there has been an incorrect prediction. After the optimal separating hyper-plane is obtained, we can then use it to determine whether new data belongs to positive or negative group, a process called SVM testing.

3.4.3.1. Kernel Trick and Functions

Kernel Trick: A separating hyper plane is used in the classification of linear data. However, in real world problems the data sets are mostly non-linear. In such cases, kernels are used to non-linearly map the input data to a high-dimensional space. The new mapping is then linearly separable. The space x_i are in is called the input space and the space of $\phi(x_i)$ after transformation is called feature space. As discussed in [74] linear operation in the feature space is equivalent to non-linear operation in input space [74]. The training set is linearly separable in the feature space. This is called the "Kernel trick" [76].

Kernel Functions: The idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. We want the function to perform mapping of the attributes of the input space to the feature space. The kernel function plays a critical role in SVM and its performance [77].

The four types of Kernel Functions usually used with SVM are:

1. Linear Kernel $K(x, x') = (x,x' + 1)$

2. Polynomial Kernel $K(x, x') = (kx .y + c)^p$

3. Sigmoid Kernel $K(x, x') = \tan h(kx.y + c)$

4. Radial Basis Kernel $K(x, x') = e^{-\gamma||x-y||2}$

Simple linear SVM is used in this study because it provides good generalization accuracy and is fast to learn [78].

3.4.3.2. SVM Algorithms

Despite its advantages for global solution, good generalization and having common ground for linear and non linear solutions, SVM has a remarkable drawback in the fact that it implies solving a large constrained quadratic programming (QP) problem, its size making often standard solvers impractical as memory requirements are quadratic in the number of patterns [75]. This has given rise to a great deal of SVM specific training algorithms, which nearly always not solve the original SVM problem but an equivalent formulation in the dual space. As a result, the solution hyper plane can be expressed as a linear combination of the patterns associated with non-zero multipliers (support vectors).

There are two main classes of training algorithms: Decomposition and Geometry-based algorithms. The decomposition algorithms for training SVM rely on the idea of not solving the whole dual quadratic programming problem, but on solving iteratively (with a specific quadratic programming solver or another SVM algorithm) for a subset of multipliers. The common decomposition algorithms are: chunking, SMO and SVM-Light.

The chunking algorithm started with arbitrary subset or ‘chunk’ data. Then, the support vectors remain in the chunk while other points are discarded and replaced by a new working set. In chunking, the size of the working set tends to grow with time, though it

can occasionally shrink. The SMO is an original implementation for the SVM classifier designed to avoid the likely large quadratic programming optimization problem that appears as part of the SVM model. The idea behind SMO is that the quadratic programming problems can be broken up into a series of the smallest possible quadratic programming problems and solved analytically by optimization. SVM-Light is more general than SMO and the working set size is set in advance to a fixed number with the constraint that this number is even. The main contribution of SVM-Light, apart from its numerous computational tricks that make it very fast, is the way to select the multipliers for the working set

On the other hand, geometry-based algorithms exploit the geometric reinterpretation of SVM classifiers. These algorithms are often referred in the literature with the generic name of nearest point algorithms (NPA). These are iterative algorithms that asymptotically approximate either the closest point of a convex hull to the origin, either the closest points belonging to two different convex hulls, or the closest points belonging to two different reduced convex hulls, depending on the SVM reformulation exploited.

In this study sequential minimal optimization (SMO) method is employed to learn the vector of feature weights, $\vec{w}$. Once the weights are learned, new items are classified by computing $\vec{w} \cdot \vec{x}$ where, $\vec{w}$ is the vector of learned weights, and $\vec{x}$ is the binary vector representing a new document.

3.5. Tools used

3.5.1. WEKA: Machine Learning Software

WEKA stands for Waikato Environment for Knowledge Analysis and was developed at the University of Waikato in New Zealand [69]. It is an open source machine learning suite written in Java which provides an interface to various machine learning algorithms. It supports pre-processing of data sets and helps in evaluating performances of learning algorithms. WEKA has a graphical user interface, called the WEKA Explorer, and a command line interface is also available. Both the WEKA's Explorer as well as the command line interface were used in this study.

Training schemes that perform classification are used to classify the preprocessed data. Many classification schemes like decision trees, rule learners, naive Bayes, decision tables, locally weighted regression, SVMs, instance based learners, logistic regression, and multi-layer perceptrons are supported by WEKA. To build classification model and to predict unknown class labels WEKA's J48 decision tree and SMO SVM implementation have been used in this study.

3.5.2. Python

Python is defined as an object-oriented scripting language. It is an object oriented programming language which is often used for scripting purposes. Here, python is used to map number class labels and equivalent Amharic characters. The data structure Dictionaries have been extensively used. Dictionaries are nothing but associative arrays.

CHAPTER FOUR

EXPERIMENTATION

Information in written form plays an undeniably important role in our daily lives. Recording and using information encoded in symbolic form is essential. Visually impaired people face a distinct disadvantage in this respect. To address their need, the most widely adopted writing convention among visually impaired people is Braille. Different attempts have been made to recognize Braille writing system for different languages. To recognize Amharic Braille documents an attempt has also been made by Teshome [49] and Ebrahim [16]. As a continuation, this study explores feature extraction and the classification modules of Amharic Braille recognizer. This is because of the fact that the performance of the recognizer depends mainly on the feature extraction and classification schemes. In this study we experiment various feature extraction and classification algorithms, in an attempt to select suitable once to enhance the performance of Amharic OBR.

As depicted in Figure 3.1, the Amharic OBR system works as follows. First single sided Amharic Braille documents are scanned using flatbed scanner at a resolution of 200 dpi; this creates digitized images. Then Braille images are preprocessed and segmented into Braille dots. The segmented bunch of Braille dots are passed to the feature extraction module where features of Braille dots are extracted and grouped into Braille cells. The extracted features are used for training the classifier and create classification model. Finally the recognition module converts Braille images into their equivalent textual Amharic characters.

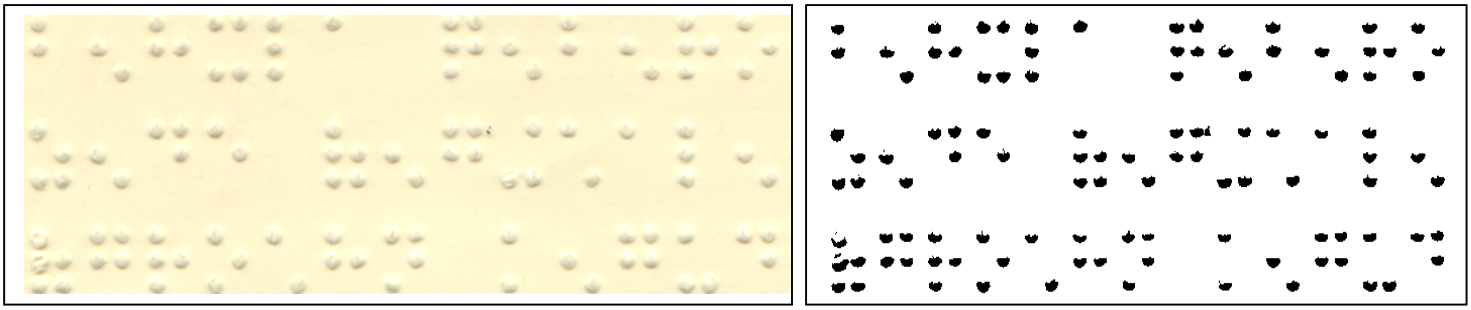
4.1. Dataset Preparation

Braille documents have been collected mainly from AAU-Kennedy library. The collected documents are single-sided typewritten Braille documents with the standard Braille size of 11×11.5 inch, which is the standard in Braille writing. The details of sample Braille documents used in this study are summarized in Table 4.1.

Braille document property	Description
Braille sheets	24
Total number of characters	9306
Average number of characters per sheet	387
Resolution	200 dpi
Digital format	Gray scale
Image size	11 X 11.5 inches
Braille type	Single sided
Image format	Bitmap(bmp)
Main document source	AAU-Kennedy library

Table4. 1 Summary on Amharic Braille documents collected for this study

After Braille documents have been collected, scanning was performed. In OBR scanning Braille documents enables us to perform digitization which produces a digital image of the scanned Braille document. A flat-bed scanner has been used for digitization process. This is because it has been found as a cheap alternative to scan Braille images. Scanning has been performed with horizontal and vertical resolution of 200 dpi, which is recommended to get quality images [16][49] and the images are stored in bitmap format. Using this configuration Braille documents that are collected from different sources are scanned and prepared for subsequent processes. The image shown in Figure 4.1 is sample scanned Braille document image using flat-bed scanner with 200 dpi resolution.



(a) **gray scale** (b) **binarized**

Figure4. 1 Sample scanned Braille image

This Braille document images are used for training and testing the system. To train the classifiers three training data sets are prepared with the following proportion:

Training dataset 1: contains 238 instances of basic Amharic alphabets with 12 bits pattern sequence and corresponding class values.

Training dataset 2: contains 258 instances consisting of 238 basic Amharic alphabets and 20 numerals with 12 bits pattern sequence and corresponding class values.

Training dataset 3: contains 281 instances consisting of 238 basic Amharic alphabets, 20 numerals and 23 punctuation marks with 12 bits pattern sequence and corresponding class values.

For measuring the performance of the classification model, test datasets from real life Braille documents have been prepared with the following proportion.

Test set1: contains a total of 2406 characters taken from clean Braille documents.

Test set2: contains a total of 1705 characters taken from Braille documents with small level noise.

Test set3: contains a total of 1257 characters taken from Braille documents with medium-level noise.

Test set4: contains a total of 1301 characters taken from Braille documents with high-level noise.

The training file consists of the sequence patterns (bit representation) for Amharic Braille alphabets, numerals and punctuation marks as well as the class for each sequence. Before making decisions on output nodes it is worth to mention that all Amharic Braille characters class values are represented in numbers. To come up with uniform mapping the characters are assigned index value from 1 to 281; 1 is for "***U***", 2 is for "***U·***" (the detail appears in Appendix VII).

The input files for training set are prepared for 281 Amharic characters and trained with J48 decision tree classifier and SMO implementation of SVM. Now, again the testing file is prepared from the extracted feature in the form of ARFF file format with missing class label represented with "?". The test is performed using testing set and predicted class labels are produced by the classifiers.

The input data for training is fed to WEKA machine learning tool in the form ARFF file format. The ARFF file structured to train the classifier is described in Appendix VI. The name of the relation appears at the beginning of the file; @relation declaration associates a name with the dataset. Here, the name of the relation is "Braille" and it is represented as @relation Braille. Following the name of the relation is a block defining the attributes; @attribute declaration specifies the name and type of an attribute. There are 13 attributes consisting of 12 cells and one class; all of them take nominal value. Finally, @data section specifies feature values extracted for each Braille character. Except class value which takes numbers 1 to 281, other attributes assigned either 0 or 1 value depending on their positional information in the Braille.

4.2. Feature Extraction Techniques

A Braille character is made up of six dots arranged in two columns and three rows. Dots on the six possible places can be raised or flat to create a Braille character. The feature extraction techniques help us to group Braille dots of OBR in to cells. In Amharic Braille writing system one cell (6 dots), two cells (12 dots) and three cells (18 dots) may be used to represent the corresponding Amharic characters, numerals and punctuation marks.

However, in this study we consider the most commonly used 238 Amharic characters, Arabic numerals, basic Ethiopic numerals and frequently used punctuation marks. To make the representation uniform all Amharic characters, numerals and the selected punctuation marks are represented in two cells (12 dots).

Braille character is described as a region in the image and this region is divided into six equal compartments (two across, three down) in which searching for dot is performed. To group the possible valid Braille dots in cells three feature extraction techniques, such as

fixed cell measures, horizontal and vertical projections and grid construction, are implemented in this study. In each technique the extracted dots are grouped as one and two cells to prepare different patterns for the classifier. To do this, each algorithm scan one cell at a time and the first cell may define the basic Amharic Braille character (prefix), number mode or punctuation; and the next cell (next six Braille dots) defines the corresponding vowel, number or punctuation mark for a particular Braille character.

4.2.1. Fixed Cell Measures

To extract Braille dots in a cell this technique assumes that the average cell height and width are to be fixed. Based on these measures the algorithm divides the Braille image in certain number of rows and columns and then it moves through each column and row.

To extract dots and group them in a cell, the algorithm begins from the first row and the first column then it divides the image into sub images using cell height and cell width. The average cell height is found to be 50 and average cell width is 30 through experimentation. Then each sub image is divided into sub images of 10 X 10. In each sub image the code searches a Braille dot: If the number of pixels is greater than or equal to 10 then it considers as a Braille dot and it stores 1 with an array (*BTemp[m++]=1*) otherwise 0 is stored in an array (*BTemp[m++]=0)*. To get the next possible dot position it moves horizontally until it reaches to the end of the Braille width. After it reaches to the Braille width it increments the height of the Braille image and it searches all possible Braille dots until it reaches to the Braille height. The visual C++ code for this technique is depicted in Figure4. 3. To illustrate the situation we can consider the Braille image given in Figure4. 2, which indicates the possible Braille dot positions in fixed cell measures.

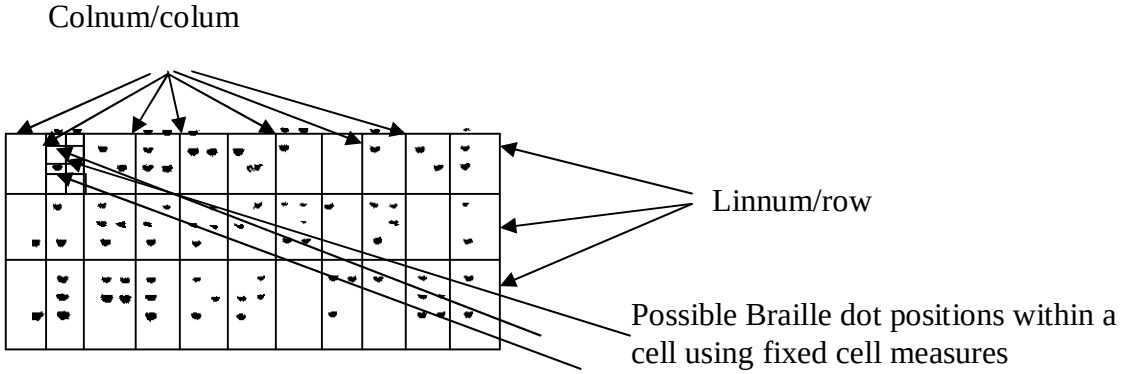


Figure4. 2 Braille dot positions using fixed cell measures

There is a challenge encountered during implementing this techinque. The challenge is that when there is skewness the technique is incapable of extracting all valid Braille dots. The Visual C++ implementation of this technique is described in Figure4.3. Here, the variables x,y,m, i and j are initialized with 0. The variables i and j are used to move to the beginning of a cell horizontally and vertically respectively.

```
k=i;
while(k<=i+30){
  n=j;
        while(n<=j+50){
              for(int p=n;p<n+10;p++){
                    for(int s=k;s<k+10;s++)
                          if(pDC->GetPixel(s,p)==0)
                                {dotcount++;}}
               if(dotcount>=10){BTemp[m++]=1;}
                        else{BTemp[m++]=0;}
                        dotcount=0;
                       n+=20;}}
             k+=20;}x++;}y++;}
```

Figure4. 3 Visual C++ code for feature extraction based on fixed cell measures

4.2.2. Horizontal and Vertical Projection

Grouping of dots in a cell in this technique is based on horizontal and vertical projection of Braille dots. First, the position for all valid dots is detected using horizontal and vertical grid lines. Following the horizontal and vertical grid lines the position of active dot is stored in an array. Similarly the positions of all valid dot positions are also stored in array using vertical and horizontal projections. As a result all possible dot positions on the same horizontal projection have the same "y" coordinate value and dot positions on the same vertical projection have the same "x" coordinate value. To identify valid dots in a cell this technique checks the position of the projection values stored in an array against the coordinate values of the active dots position. If the horizontal projection as well as the vertical projection matches the positions of active dots, then it is a valid dot; otherwise it is a background. To illustrate the situation horizontal and vertical projection of Braille dots are shown in Figure 4.4.

Since the segmentation is not flexible enough and the dots are not always aligned horizontally and vertically, this technique encounters a problem when there are differences in dots size and skewness. The Visual C++ code for this technique is described in Figure 4.5. Here, the variables i, j,p and u are initialize with zero.

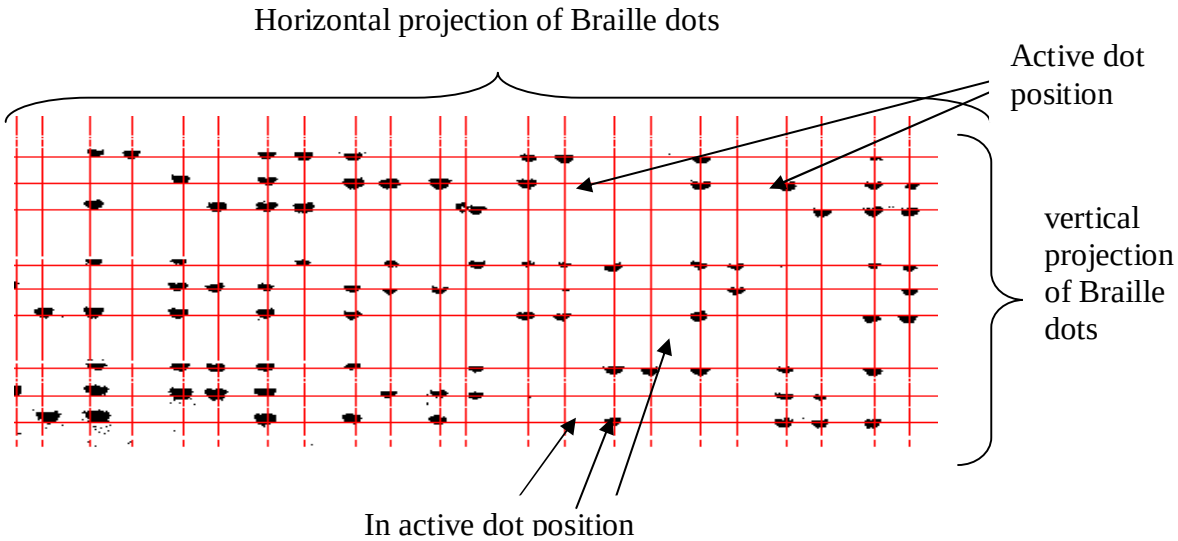


Figure4. 4 Braille dot positions based on projections

```
while(DotHorProj[j])
    {
      while(DotVerProj[i])
      {for(int w=i;w<i+2;w++)
                  {for(int h=j;h<j+3;h++)
                        {u=0;val=FALSE;
                        while(dot[u][0]){

      if((dot[u][1]==DotHorProj[h])&&(dot[u][0]==DotVerProj[w]))
                                    {BTemp[p++]=1;
                                          val=TRUE;}
                                    u++;}
                              if(val==FALSE)
                                    {BTemp[p++]=0;

                                    }}}j+=3;}i+=2;}
```

Figure4. 5 Visual C++ code for feature extraction using projection

4.2.3. Grid Construction

As described in [4] the placement of dots within Braille character is regular. The space between adjacent characters in the same character line and between adjacent character lines is also regular. Therefore, one could construct a grid whose intersections determine the possible position of dots on the image. Tehome [49] also adopted it using global mesh construction but in a fixed way that does not account for possible variations in character positioning in different lines. The grid construction technique described here constructs a relatively flexible grid by allowing variations in the positions of characters between different lines (i.e., the Braille characters need not be aligned in the vertical direction).

In order to group Braille dots in a cell, this technique uses the grid lines constructed for Braille dots horizontally and vertically. Using a very simple array, we first try to find the position of the grid lines. Then the image is divided in to sub-images in order to calculate grids for each sub image separately (local linearization of the deformations). Grids of horizontally and vertically adjacent sub-images are then taken together in horizontal and

vertical strips: horizontal strips contain vertical grid lines; vertical strips contain horizontal grid lines. The lines in a strip are then grouped in atoms. An atom is a group of lines that belongs to a single Braille cell. The representation for 1-atom and 2-atom is described in Figure 4.6. Finally, the relationship between the strips is restored and at the cross sections of horizontal and vertical atoms Braille characters can be found and searching pixels near the cross section is performed. If a Braille character is found then 1 is restored in the array, otherwise 0 is restored in the array.

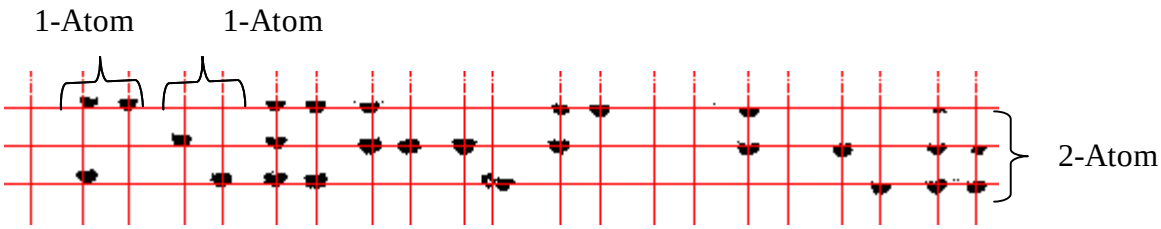


Figure4. 6 Examples of atoms in a six-dot Braille cell

The challenge here is the presence of noise at the edge of scanned Braille image causes the construction of the grid to be incorrect. The Visual C++ code for this algorithm is presented in Figure 4.7. Here, the variables w, h, and r are initialized with zero.

```
while(col[h][2])
    {   while(line[w][1])
            { for(int i=0;i<2;i++)
                {for(int j=0;j<3;j++)
                    {for(int d=line[w][i];d<line[w][i]+10;d++)
                        {for(int g=col[h][j];g<col[h][j]+10;g++)
                            {if(pDC->GetPixel(d,g)==0)
                                {dotcount++;}
                                } }
                        if(dotcount>=10){BTemp[k++]=1;}
                        else{BTemp[k++]=0;}

                        }}w++;}   h++;}
```

Figure4. 7 Visual C++ code for feature extraction using grid construction

Having grouped Braille dots in a cell, context analysis is performed to determine the status of dots in a cell. Based on the contextual analysis result the cell can be recognized as Braille character alone or part of a Braille character. This is because depending on the context a Braille character may have one or two cells. Based on the analysis if the cell content gives the required Amharic Braille character then the content of the cell is registered with six additional 0s because this cell can make Amharic Braille character alone. On the other hand, if the cell content requires the second cell to make the Braille character or is part of a Braille character then the contents of the two Braille cells are restored as a twelve bit representation for a particular Braille character.

During contextual analysis the content of a cell has been checked for three conditions: Braille vowel, numerals and punctuation. In order to make contextual analysis for Amharic Braille core characters, numerals and punctuation marks we define first the criteria for each condition. Amharic Braille core characters' contextual analysis is done in two ways: all the basic forms (the 6th form of a given character), which are 34 in number, are represented with one Braille cell and the rest (34*6) requires two Braille cells (the first cell is their basic form and the second cell is the vowel for one of the six forms). To illustrate sample basic characters are given in Table 4.2 with Braille code representation, dot position representation in bits and equivalent Amharic characters.

Braille code	**Dot positions**	**Representation**						**Amharic character**
		d1	**d2**	**d3**	**d4**	**d5**	**d6**	
	125	1	1	0	0	1	0	ሀ
	134	1	0	1	1	0	0	ፃ
	146	1	0	0	1	0	1	ሸ
	16	1	0	0	0	0	1	ኍ
	12356	1	1	1	0	1	1	ኧ
	1356	1	0	1	0	1	1	ዘ
	245	0	1	0	1	1	0	ጅ
	12346	1	1	1	1	0	1	ዕ

Table4. 2 Basic Amharic characters one cell representation

Further Amharic core characters other than basic characters are represented using two cells; the first cell defines the basic character for that variant and the second cell defines the vowel. To make the representation clear consider the characters "ሀ" and "ሁ". The basic character for the two characters is the same which is "ህ" represented using "110010". Their difference lies on the vowel representation: "ሀ" uses "010010" as vowel where as "ሁ" uses "101001" to its vowel. The vowel representation for other variants is done in similar way: the vowels are "010100", "100000", "100010", "101010", for the third, fourth, fifth and seventh variants. Table 4.3 shows one to one mapping between Braille to print Amharic characters for "ሀ" variants. The dot position within a cell are represented with d1, d2, d3, d4, d5 and d6 and the presence of a dot at a given cell in one of the six possible dot positions is represented by 1 where as the absence of a dot is represented by 0.

Braille code	Dot position	Representation												Amharic character
		cell1						cell2						
		d1	d2	d3	d4	d5	d6	d1	d2	d3	d4	d5	d6	
	125 and 26	1	1	0	0	1	0	0	1	0	0	0	1	ሀ
	125 and 136	1	1	0	0	1	0	1	0	1	0	0	1	ሁ
	125 and 24	1	1	0	0	1	0	0	1	0	1	0	0	ሂ
	125 and 1	1	1	0	0	1	0	1	0	0	0	0	0	ሃ
	125 and 15	1	1	0	0	1	0	1	0	0	0	1	0	ሄ
	125	1	1	0	0	1	0	0	0	0	0	0	0	ህ
	125 and 135	1	1	0	0	1	0	1	0	1	0	1	0	ሆ

Table4. 3 Braille to print mapping for the "ሀ" Amharic character variants

The contextual analysis for Amharic Braille core characters is done using the visual C++ code depicted in Figure 4.8. In analyzing Amharic Braille core characters the vowels are stored in an array and whenever the content of a Braille cell matches these vowels the vowels are restored as the suffix of the previous cell; otherwise the consonants are restored in the current array and their corresponding next six bits are set to 0.

```
if(consonant(BTemp)==0)
        {
        if(BCount-1>=0)
          {int temp=BCount-1;
                for(int a=0;a<6;a++)
                  {ABChar[temp].vd[a]=BTemp[a];  }}
        else
        {for(int a=0;a<6;a++){
        ABChar[BCount].d[a]=0;
        ABChar[BCount].vd[a]=BTemp[a];
        }BCount++;}}
else if(consonant(BTemp)==1)
        {for(int a=0;a<6;a++){
        ABChar[BCount].d[a]=BTemp[a];
        ABChar[BCount].vd[a]=0;}
        BCount++;}
```

Figure4. 8 Visual C++code to detect Amharic characters

To make the concept clear (the code in Figure4.8) consider the Braille image given in Figure 4.9. The first cell contains the consonant 101100, then it restores in the array and it searches for a vowel. Since the second cell matches one of the vowels then it restores as suffix of the previous cell. In the figure below the consonant represents "ሀ". Since the vowel is representation of the first form of the alphabet, instead of restoring "ሀ" it restores "ለ" and registers 101100010001 to an array.

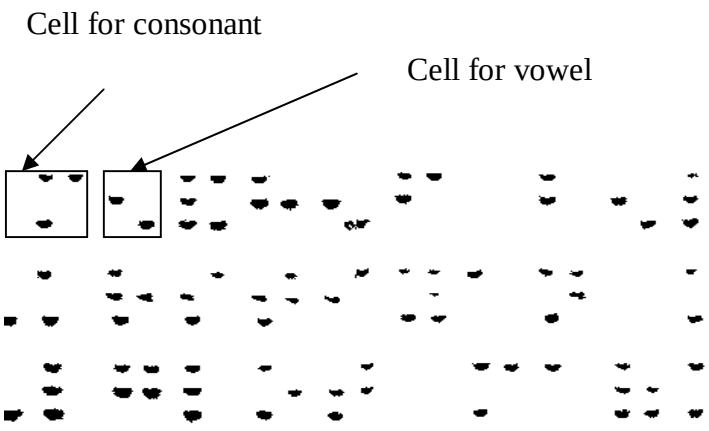


Figure4. 9 Sample Braille image to check consonants and vowels

In addition to Amharic core characters, Amharic Braille has a representation mechanism for numerals. Amharic Braille uses both Ethiopic and Arabic numerals. Both numerals are represented in two Braille cells. Their difference lies on the numeral mode that precedes the numbers. The numeral mode for Ethiopic number is “111111”, while “001111” used for Arabic numbers. Accordingly the first cell in numerals defines the numbers mode (Ethiopic or Arabic) and the second cell defines the numbers 0-9 for Arabic and ⠚-⠊ for Ethiopic numbers. To illustrate the situation the representational mechanism and the mapping for Arabic and Ethiopic Braille numerals are given in Appendix IV.

Amharic Braille also has a representation for different punctuation marks which are used in Amharic writing system. In Amharic Braille punctuation marks can be represented in one, two or three Braille cells. The mapping between Amharic Braille punctuation marks and their corresponding punctuation marks in print for sample punctuation marks with one Braille cell and two Braille cells are given in Appendix IV.

Similarly contextual analysis is made on Amharic Braille punctuation marks since their representation is based on either one or two Braille cells. To handle the situation punctuation marks with one cell and punctuation marks with two cells are defined separately. If the content of a Braille cell matches one of the punctuation marks with one cell then the first array restores the defined punctuation mark representation and the second cell is filled with 0s. Those punctuation marks which require two Braille cells are represented in punctuation prefix and suffix. If the punctuation prefix is found in the first cell then the second cell is considered as the suffix of that punctuation mark.

Moreover to give room for some Braille characters that does not considered in this study; such as the remaining punctuation marks and extra Amharic characters, all other cells that fail to meet the above criteria are labeled as others cell content. The full Visual C++ code for feature extraction is given in Appendix V.

To prepare patterns input for the classifiers, the extracted features from Braille images are represented in twelve bits and class labeled with “?”. At this stage dots in a cell are

converted to a binary digit, 1 indicates the presence of dots whereas 0 represents the absence of dots in the corresponding dot position as shown in Table4.4. The extracted features are then fed to the J48 decision tree classifier and SMO implementation of SVM to predict their class label.

c1d1	c1d2	c1d3	c1d4	c1d5	c1d6	c2d1	c2d2	c2d3	c2d4	c2d5	c2d6	class
1	1	0	0	0	0	0	0	0	0	0	0	?
0	1	1	1	1	0	0	0	0	0	0	0	?
0	0	1	1	0	1	0	1	0	0	0	1	?
1	0	1	1	1	1	0	0	0	0	0	0	?
1	1	1	1	1	0	0	0	0	0	0	0	?
0	0	0	0	0	0	0	0	0	0	0	0	?
0	1	1	1	1	0	1	0	0	0	0	0	?
1	1	0	1	1	1	0	1	0	0	0	1	?
0	0	1	1	0	1	0	1	0	0	0	1	?
0	1	0	1	1	1	1	0	0	0	0	0	?
1	1	1	0	0	0	0	1	0	0	0	1	?
0	1	0	0	1	0	0	0	0	0	0	0	?
0	1	0	0	1	1	0	0	0	0	0	0	?

Table4. 4 Sample extracted features represented in 12 bits

4.2.4. Performance Evaluation for Feature Extraction Techniques

The feature extraction techniques discussed so far and the feature extraction technique (i.e. intersection of mesh grid lines) employed in the previous study are used to extract and group Braille dots into cells. Here, Braille characters are given for each technique to analyze its performance. The total numbers of Braille characters, which are extracted from clean Braille images, used for testing are 437. The extracted features are then fed to the J48 decision tree classifier to measure the accuracy for each technique. J48 decision tree classifier is used to compare the performance of each technique because J48 is found to be simple, as compared to SMO, to build model and generate predictions easily. The accuracy of each algorithm is then measured by dividing correctly extracted Braille characters to the total number of characters given. Table4.5 depicts the performance (accuracy) of the feature extraction techniques employed in this study.

Feature Extraction Technique	Total number of characters	Correctly Extracted Braille Characters	Accuracy (%)
Fixed Cell Measures	437	432	98.85
Horizontal and Vertical Projection	437	399	91.30
Grid Construction	437	419	95.88
Intersections of grid lines (previous study)	437	397	90.85

Table4. 5 Performance comparisons for feature extraction techniques

As can be seen in Table4.5, the feature extraction algorithm with fixed cell measures shows better results in extracting Braille dots and grouping them into cells but the algorithm based on intersection of grid lines performs least. Since Braille characters are represented by six points, 3 per column and 2 per row, and have standard distance between them, the performance of fixed cell measures feature extraction algorithm is reasonable. But the algorithm based on intersection of grid lines performs least. The reason is that all Braille dots could not found at the intersection point of the grid lines. Hence, Braille dots in cells near at the intersection of mesh grid lines (above or below) are not extracted well in the previously adopted feature extraction technique.

4.3. Amharic Braille Character Classification and Recognition

The classification and recognition module relies on the results of the feature extraction module and predicts the target class values for extracted features based on the training model. The overall task of the classification and recognition module is to be able to classify extracted input Braille characters into one of the possible Amharic characters. The feature patterns that are extracted from the Braille image, as shown in Table4.4, are stored in a file with 12 bits sequence that can be either 1 or 0 and their class value with "?".

For Amharic Braille characters a training dataset is created. To train and classify test data sets, to their respective class labels, WEKA tool is used. Supervised machine learning algorithms J48 decision tree and Sequential Minimal Optimization (SMO)

implementation of SVM are used in this study for training the classification model and to generate rules. The trained decision tree and SVM models contain a set of rules for predicting class label. When predicting the class labels for a given sequence of features, the most probable class label for each Braille feature is predicted using the respective trained model. Both decision tree and SVM are suitable for hierarchical problem; decision tree works better in linearly separable problems but SVM works in complex and non-linear separable problems.

4.3.1. Training the Decision Tree and SVM

To train the classifiers, different parameters are specified. In this study default values for each parameter are accepted. The reason is that the default values perform some pruning. The selected parameters for the decision tree classifier and SVM classifier are depicted in Table4.6.

Using training instances it is possible to generate a classification model which can be used in classifying new instances. The advantage of building a model and storing it is that it can be applied at any time to different sets of unclassified feature patterns. To generate the classification model both the GUI and command line versions of WEKA are used in this study.

Parameters	**Classifier**	
	J48 default value	**SMO default value**
Confidence factor	0.25	-
minNumObjects	2	-
Cross validation	10-folds	10-folds
Complexity constant	-	1.0
Epsion	-	1.0E-12
Kernel	-	PolyKernel -C 250007 -E 1.0
Tolerance parameter	-	0.0010

Table4. 6 Default parameters for J48 and SMO classifiers

To generate models in WEKA's command line interface the following lines of instruction are used. The first line of instruction is for J48 decision tree classifier whereas the second is for SMO SVM implementation.

java weka.classifiers.trees.J48 -C 0.25 -M 2 -t *directory-path****Braille.arff -d*** *directory-path ****Braille.model***

java weka.classifiers.functions.SMO –C 1.01 -L 0.0010 -N 0 -t *directory-path****Braille.arff -d*** *directory-path ****SMO.model***

Where the options,

- -C is the confidence factor used for pruning whose default value is 0.25 for J48 and the complexity constant whose default value is 1.01 for SMO
- -M is the minimum number of instances per leaf whose default value is 2
- -L is the tolerance parameter whose default value is 0.0010
- -N is option to specify whether to 0=normalize/1=standardize/2=neither, default value is 0=normalize
- *-t* option specifies that the next string is the full directory path to the training file (in this case "**Braille.arff**").
- *directory-path* is the full directory path where the training file resides.
- *-d directory-path* option specifies the name (and location) where the model will be stored.

4.3.1.1. Classifiers' Performance Evaluation on Model Building

At this point whether the classifiers can recognize the character set used for training is analyzed. To see whether the classifiers can learn all the character sets used for training, the three datasets prepared for training have been fed to the decision tree and the SVM classifier independently. Moreover, each training set contains three iterations. In the first iteration each instance has one value, in the second iteration two repeated values and similarly in the third iteration each instance has three duplicate values.

The classification accuracies of the models created using the three training sets, with three iterations, for J48 decision classifier and SMO implementation of SVM are depicted in Table 4.7. To build models on training datasets for both J48 and SMO are run using 10-fold cross validation as evaluation approach.

As can be seen in Table4.7, analysis of experiments show that the models created using training dataset 1, 2 and 3 results 100% accuracy in iteration2 and iteration3. This indicates that both the J48 decision tree and SMO SVM implementation learn (fully recognizes) all Amharic alphabets and numerals. However, the classification accuracy of the model created using J48 decision tree in for all trainings in iteration1 is around 43%: this indicates that J48 decision tree learns all characters when we increase the training record.

Dataset (Unique instances)	**Iteration (total records)**	**Classifier**			
		Decision Tree (J48)		**SVM (SMO)**	
		Accuracy (%)	**Time[1] (sec)**	**Accuracy (%)**	**Time (sec)**
Training dataset 1 (238)	1 (238)	42.86	0.22	100	215.96
	2 (476)	100	0.04	100	102.96
	3(714)	100	0.05	100	242.43
Training dataset 2 (258)	1 (258)	42.63	0.03	100	252.5
	2 (516)	100	0.06	100	549.52
	3(774)	100	0.04	100	577.26
Training dataset 3 (281)	1 (281)	43.06	0.11	100	142.1
	2 (562)	100	0.11	100	148.41
	3(843)	100	0.11	100	436

Table4. 7 Training model performance for J48 and SMO

[1] Time taken to build a mode in seconds

From the above models, the models created using training set 3, iteration 3 for both the decision tree and the SVM, have been selected as a model for testing new extracted features. The rationale behind is that the models at this training set are built using all the characters that are in other training sets plus additional punctuation marks. Generally, Evaluation on the performance of the models with the training set indicates that for all training sets almost the same level of performance is achieved by both the decision tree and SVM. However, in terms of time to build a model SVM is very expensive.

4.3.2. Testing the Decision Tree and SVM Performance

Once the decision tree (J48) and SVM (SMO) classifiers are trained, it is required to test the performance of the models with different test cases. For the test cases input patterns are extracted from Braille images using fixed cell measures technique, which is found to be the better feature extraction technique. These patterns or features are prepared and fed to the J48 decision tree and SMO SVM classifiers. Since the extracted features have missing class label, it is necessary to predict the class label for each pattern sequences using the classification models built earlier. The prediction on class label for extracted features is done in two ways. The first one is using the GUI version of WEKA and the second is using the command line version of WEKA.

To test new extracted features, commands for J48 and SMO are given below respectively.

java weka.classifiers.trees.J48 -l directory-path\Braille.model -T directory-path \Braille-newfeature.arff -p 0 > directory-path\filename

java weka.classifiers.functions.SMO -l directory-path\Braille.model -T directory-path \Braille-newfeature.arff -p 0 > directory-path\filename

In the above commands, the -l options specifies the directory path and name of the model file, the -T option specifies the name (and path) of the test data. In this case, the test data is extracted features stored in a file “Braille-newfeature.arff”. The option –p 0 indicates prediction and > option is used to redirect (write) to a file called *file name*.

The performance of the classifiers is tested on datasets taken from clean Braille image (test set1) and noisy Braille images (test set2 and test set3). Table 4.8 presents test results obtained using models created by J48 decision tree and SOM SVM.

Test set	Total Characters given to the classifier	Classifier			
		J48		SMO	
		Correctly classified	Accuracy (%)	Correctly classified	Accuracy (%)
Test set 1	402	399	99.25	399	99.25
Test set 2	400	381	95.25	389	97.25
Average		390	97.25	394	98.25

Table4. 8 Performance of J48 and SMO on test data set

As can be seen in Table4.8, SMO implementation of SVM and J48 decision tree classifiers register, on the average, 97.25% and 98.25% accuracy respectively. The performance of both the J48 decision tree and SMO SVM implementation is similar for test set1. This indicates that both classifiers are suitable in classifying Braille characters extracted from clean Braille documents. For test set2, which is strategically selected from noisy documents, SMO implementation of SVM outperforms J48 decision tree classifier in predicting unseen features. This indicates that as the noise level increases (for testing on real life documents) SMO implementation of SVM registers better results. In general, as the level of noise increases the performances of both classifiers decrease. Hence, noise removal technique is required to improve the performance of the classifiers.

To illustrate Table4.9 shows sample characters that are misclassified in test set1 and test set 2 for both classifiers. As can be seen in Table4.9, some characters are incorrectly classified by both classifiers whereas some are correctly classified by SMO but incorrectly classified by J48. For example, the character "***U***" which is represented in 110010010001 (dot position: 125 and 26) is replaced by character "***¿***", which is represented in 111010010001 (dot position: 1235 and 26). Here, addition of a dot at position 3 is committed by J48. On the other hand, both J48 and SMO replaced the target

character "ዖ", which is represented in 1110101101010 (dot position: 12346 and 135), with character "ዘ", which is represented in 101011101010 (dot position: 1356 and 135). Here, deletion on dot position 2 and substitution of dot position 4 by dot position 5 is performed.

Test set	Input Dot representation												Expected	Predicted	
														J48	SMO
Test set1	1	1	1	1	1	1	0	1	0	1	0	0	ኡ	፱	ቂ
	1	0	1	1	1	1	1	0	0	0	0	0	ዖ	፦	ኛ
	1	0	0	0	1	1	0	0	0	0	0	0	ጎ	ች	፡
Test set2	1	1	1	0	1	1	0	1	0	0	0	1	አ	ጸ	አ
	1	0	1	1	1	1	1	0	0	0	0	0	ዖ	ማ	ማ
	0	1	1	1	1	1	0	1	0	1	0	0	ጢ	።	ዊ
	1	1	0	0	1	1	1	0	1	0	1	0	ዖ	ዘ	ዘ
	1	1	0	0	1	0	0	1	0	0	0	1	ሀ	ረ	ሀ

Table4. 9 Error types in validating DTC and SVM classifiers

Even though SVM takes much time to build a model, once the model has built it generates prediction results in minimal time. In addition, SVM has been more suitable for noisy documents as compared to decision tree. Hence, SVM has been used to measure the performance of the system: in the next section in order to measure the performance of the system fixed cell feature extraction technique and SMO SVM implementation has been used as a feature extraction and classification techniques respectively.

4.4. Performance Evaluation of the Amharic OBR System

The performance of the system, prediction accuracy of SMO, is tested with test sets extracted from real life Braille documents using fixed cell measure feature extraction technique. This test helps to measure the performance of the SVM classifier on real life Braille documents. The real life documents are grouped into four noise levels as Ebrahim [16] did: clean, small noise level, medium noise level and high noise level Braille documents. In measuring the performance of the classifier four test sets have been

prepared with a total of 5867; 2004 for clean characters, 1462 small noise level characters, 1100 for medium noise level characters and 1301 for high noisy level characters.

Accordingly, test set1, test set2, test set3 and test set4 have been submitted for recognition. The system is tested with Braille images filtered with the proposed noise removal technique, Gausian filtering, by Ebrahim [16]. The performance of the system is depicted in Table4.10**.**

Test set	**Test size**	**Prediction accuracy (%)**
Test set1	2004	98.5
Test set2	1462	97.26
Test set3	1100	94.36
Test set4	1301	72.56
Average		90.67

Table4. 10 Performance rates for the system on real life documents

As can be seen in Table 4.10, the performance of the system for clean Braille documents (test1), small-level noise Braille documents (test2) and medium-level noise Braille documents (test3) is more comparable and better than high level noise Braille documents (test4). This indicates that the proposed feature extraction and classification techniques, handles small and medium noises encountered in Braille images. On the other hand, the performance of the system decreases significantly for documents with high noise level. The rationale behind is that since high noise levels created due to repetitive use, bend and highly connected Braille dots, in attempt to reduce noises in documents the noise removal technique also adds and deletes Braille dots. To illustrate consider the examples given Table4.11.

Braille Document		Translation
scanned		Experts': ብትጠይቅ ታገኘዋለህ::
preproc essed		System's: ብትኘይቅ ታሰኘዋለ፥::

Table4. 11 Sample test result on real life document

As can be seen in Table 4.11, there are differences between the experts and system translation: character "ጠ" is replaced with "ኘ", "ገ" with "ሰ" and "ህ" with "፥". The rationale is that during preprocessing Braille dots are deleted or added. The detail for each misclassification is shown below.

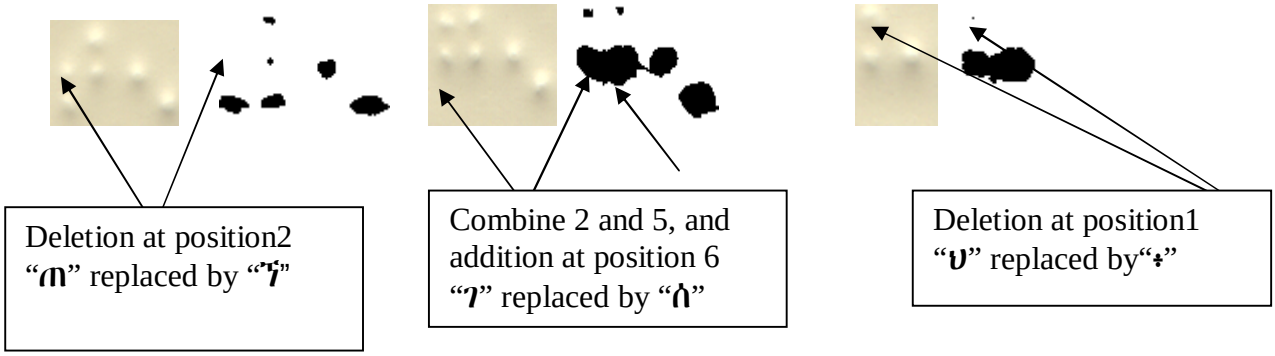


Figure4. 10 Misclassified characters on real life document

In addition, the mesh grid segmentation is not capable of handling large dots found at the edge of the Braille image. As a result, to improve performance of the system with high level noise documents a noise removal technique that can clearly separates the noise with the content is required besides an effective segmentation algorithm that is insensitive to noise.

4.5. Discussion and Challenges

As a continuation of the previous researches in the area, the present study explores feature extraction and classification schemes for Amharic OBR system. In this study,

during experimentation the feature extraction technique with fixed cell measures and SVM classifier have been found to work very well in Amharic OBR system. The results of the present system with the proposed feature extraction and classification techniques, and the previous researches [49][16] are depicted in Table 4.12.

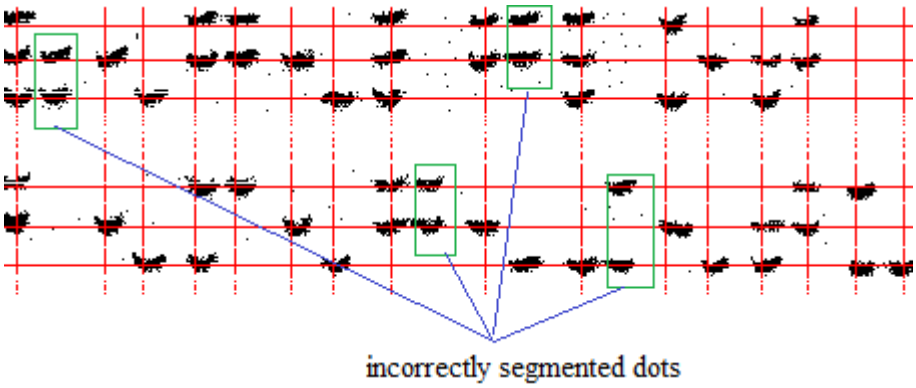
Researches	Training size	Feature extraction technique	Classifier	Noise removal	Performance on real life documents (%)			
					Clean	small noise	medium noise	high noise
[49] (2009)	267	Count black pixels at intersection of mesh grid lines	ANN	-	92.5	-	-	-
[16] (2010)	267	Count black pixels at intersection of mesh grid lines	ANN	Gaussian	95.5	95.5	90.5	65.5
Present study	281	Fixed cell measures	SVM	Gaussian	98.5	97.26	94.36	72.56

Table4. 12 Performance comparison between related works

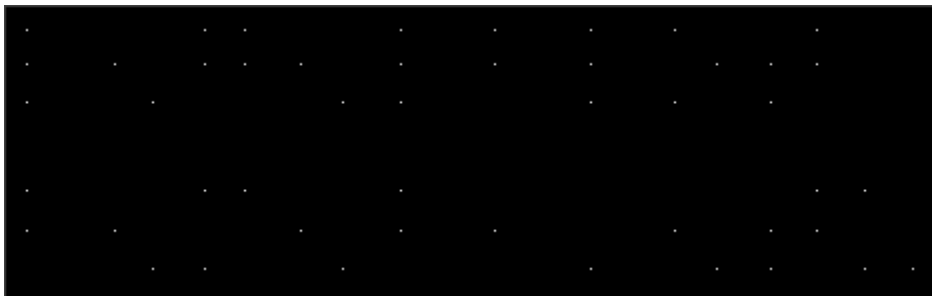
As can be seen in Table 4.12, the performance of the Amharic OBR system is improved in the present study by integrating better feature extraction and classification schemes. There is also an improvement on the performance of the system for real life documents (for small noise level, medium noise level and high noise level documents). However, the performance of the system decreases with an increase in the noise level of Braille documents. The rationale is that the noise removal technique (Gaussian) removes highly connected dots without separating the content and dots. Hence, it is better to integrate advanced noise removal technique for high noise level documents.

Besides to correctly recognize Braille documents with many artifacts, an algorithm that correctly segments the Braille images into cells is crucial in Braille recognition.

However, the segmentation algorithm encounters challenges that are created by real life documents. The variation of the Braille dots size greatly affects the construction of grid lines and in some Braille documents Braille dots segmented incorrectly. To illustrate consider the example given in Figure4.11.



(a) mesh grid construction for segmentation



(b) Segmented Braille image

Figure4. 11 Sample segmented Braille image

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1. Conclusion

The Braille system is a tactile method widely used by visually impaired people to read and write. Braille documents contain lines consisting of a series of characters. Each character has six dots arranged in three rows and two columns and each dot can either be raised or be flat according to the corresponding symbol the character represents. Braille is understandable by visually impaired people; however, vision people need not be able to understand these codes. The development of Braille recognition systems can bridge the communication gap between visually impaired and vision people. A lot of effort has been made world-wide by different researches to bridge this gap. In this study an attempt has been made on the feature extraction and classification modules of Amharic Braille recognizer which are the two main components to improve the performance of the system.

After collecting Braille documents from different sources, they are scanned, preprocessed and segmented to identify a collection of Braille dots which represents Amharic characters. From the Braille dots features are extracted using fixed cell measure, horizontal and vertical projection, and grid construction feature extraction techniques. Test results show that, among the feature extraction techniques explored in this study, fixed cell measure performs well in terms of grouping valid Braille dots to Braille cells. This technique takes advantage of the regular spacing between Braille dots within a cell, and the regular spacing between cells. The algorithm scans a Braille cell once and checks the context to determine on cell alone or two cells as a Braille character. From this information, the dots are grouped into cells and the dot patterns are determined and represented by a bit string.

Having grouped dots in a cell, contextual analysis is made to determine the status of dots in a cell. Based on the result the cell can be recognized as Braille character alone or part of a Braille character. This is because depending on the context a Braille character may have one or two cells. Amharic Braille characters use one, two and three cells to represent the corresponding Amharic character. For this study one cell and two cells Braille characters are considered and to get uniform representation for one and two Braille, one cell Braille characters are converted to two cells by filling their second cell with 0s. The bit strings of the cells are then stored in a file and then fed to the WEKA J48 decision tree and SMO implementation of SVM classifier models to predict the class labels for each twelve bit sequences.

The classification models have been trained with three different datasets in order to get better prediction results. After training and building classification model have been completed, testing has been performed to evaluate the performance of the classifiers. Test results show that SVM outperforms decision tree in terms of predicting class labels for features extracted from real life documents.

Finally, the performance of the system has been tested with four test sets based on the level of noises in Braille documents. In addition the system has been tested using Gaussian noise removal technique. The results are 98.5%, 97.26%, 94.36% and 72.56% for clean, small noise, medium noise and high noise Braille documents respectively**.**

The majority of the misclassification errors are attributed to the challenges faced by the segmentation and noise removal techniques from poor quality Braille document images. Very old documents with some of the protrusions flattened due to heavy use give rise to more incorrectly recognized characters.

5.2. Recommendation

Based on the investigation and findings of the study, the following recommendations are forwarded for further research.

1. The segmentation algorithm adapted to Amharic Braille recognizer is not flexible to handle Braille dots with different sizes. Hence, future works should adopt different

flexible segmentation algorithms to extract different dot sizes effectively.

2. In removing noises from real life Braille documents Gaussian filtering with morphological operations works well for clean, small noise and medium noise Braille documents. However, this noise removal technique cuts highly connected Braille dots in high noise documents. Therefore, to improve the performance of the system on high level noise documents future works need to integrate advanced noise removal techniques that clearly separate noises from content in high noise level documents.
3. In the Amharic OBR after all valid dots are extracted they passed to the classifier and then recognition has been made. To enable the Amharic OBR system more effective and robust it is necessary to incorporate post processing techniques (with the help of dictionary, thesaurus and grammar) that can automatically correct grammatical and spelling errors in the recognized words.
4. In this study the classification model has been built based on 281 Amharic Braille characters. Diaphones, punctuation marks with three cells and some Ethiopic numbers, multiples of 10 and multiples of 100 such as ⠸⠁,⠸⠊,⠸⠍,⠸⠎ are not included. As a result, to realize full-fledged Amharic OBR future works should consider the whole Amharic Braille character set.
5. In this study properly scanned documents have been used. The slanting of Braille affects the mesh grid construction. As a result future work should investigate skewness detection and correction techniques.
6. In considering color variation gray level documents have been tested. However, Braille documents with different color images can be produced. Since the intensity level of pixel in Braille image varies from one color to another color, future work should include Braille documents with different colors.
7. A Braille document can either have the dots embossed on one side or on both. The latter is also called *inter-poin*t Braille when the positions of the dots from one side lie between the positions of the dots on the other side. This is designed to reduce the bulk of the document and to save on materials, given that a page of Braille is quite thick and heavy compared with ordinary paper. In this study only single sided Braille documents have been considered. As a result future work needs to consider double sided Braille documents.

REFERENCE

[1]. M. Abdallah, Abualkishik, and Khairuddin Omar, "Quranic Braille system", International Journal of Human and Social Sciences, vol. 4, no.8, pp.600-606, 2009.

[2]. AbdulMalik Al-Salman, Yosef AlOhali, Mohammed AlKanhal and Abdullah AlRajih, "An Arabic Optical Braille Recognition System", *Proceedings of the First International Conference in Information and Communication Technology and Accessibility(ICITA 2007)*, Hammamet, Tunisia, pp.81-87,2007

[3]. M. Anyanwu and S. Shiva, "Application of enhanced decision tree algorithm to churn analysis", *2009 International Conference on Artificial Intelligence and Pattern Recognition (AIPR-09)*, Orlando Florida, 2009.

[4]. Apostolos Antonacopoulos and David Bridson, "A robust Braille recognition system", *Pattern Recognition and Image Analysis group*, S. Marinai and A. Dengel (Eds.): DAS 2004, LNCS 3163, 2004, pp. 533–545.

[5]. A. V. Kulkarni and L. N. Kanal "An optimization approach to hierarchical classifier design", *Proc. 3rd Int. Joint Conf. on Pattern Recognition*, San Diego, CA, 1976, pp. 459-466.

[6]. Baye Yimam, "Ethiopian writing system (Feedel)", (Translated by Samuel Kinde and Minga Negash)

Available:http://hageregziabher.wordpress.com/2010/08/18/ethiopian-writing-system-feedel [Accessed: January 11, 2011].

[7]. M. L. Bender, J. D Bowen, R. L. Cooper, and C.A. Charles, "Language in Ethiopia", London, Oxford University, pp.371-380, 1976.

[8]. P. Blenkhorn, "A system for converting Braille into print", *IEEE Transactions on Rehabilitation Engineering*, vol.3, no.2, pp. 215-221, 1995.

[9]. L. Breiman, J. Friedman, L. Olshen and J. Stone, *Classification and Regression trees*. Wadsworth Statistics/Probability series, *CRC press Boca Raton*, Florida, USA, 1984.

[10]. Cluth Mackenzie, "World Braille usage: A survey of efforts towards Uniformity of Braille notation", *UNESCO PUBLICATION MC*, 1953.

[11]. C.M. Ng, Vincent Ng and Y. Lau, "Regular feature extraction for recognition of Braille", *In proceeding of Third International Conference on Computational Intelligence and Multimedia applications*, *ICCIMA*, pp. 302-306, 1999.

[12]. C.M. Ng, Vincent Ng and Y. Lau, "Statistical Template Matching for Translation of Braille", *In Proceedings of the Spring Conference on Computer Graphics (SCCC 1999)* , 1999, pp.197-200.

[13]. D. E. Gustafson, S. B. Gelfand, and S. K. Mitter, "A nonparametric multiclass partitioning methods for classification," *in proc. 5th int. conf. pattern Recognition*, 1980, pp.654-659.

[14]. Dereje Teferi, "Optical character recognition of typewritten text", M.Sc. Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa, 1999.

[15]. Dirk Wilking and Thomas Rofer, "Real time object recognition using decision tree learning", in RoboCup2004, Lecture notes in Artificial Intelligence, Springer, pp.556-563, 2005.

[16]. Ebrahim Chekol, "Recognition of Amharic Braille documents", M.Sc. Thesis, Faculty of Informatics, Addis Ababa University, Addis Ababa, June 2010.

[17]. Ermias Abebe, "Recognition of formatted Amharic text using optical character recognition", M.Sc. Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa, 1998.

[18]. F. Zahedi and R. Thomas, "Hybrid image segmentation within a computer vision hierarchy", *IJEEE*, vol. 30, pp.57-64, 1993.

[19]. M. Garofalakis, D. Hyun, R. Rastogi and K. Shim, "Efficient algorithms for constructing decision trees with constraints", *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 335-339.

[20]. J. Gehrke, R. Ramakrishnan, V. Ganti, "RainForest - a framework for fast decision tree construction of large datasets", *Proceedings of the 24th VLDB conference*, New York, USA, 1998, pp.416-427.

[21]. G. Landeweerd, T. Timmers, E. Gelsema, M. Bins and M. Halic, "Binary tree versus single level tree classification of white blood cells", Pattern Recognition, vol. 16, pp.571-577, 1983.

[22]. Hamid Reza, Shahbazkia Telmo, Tavares Silva, and Rui Guerreiro, "Automatic Braille code translation system**",** *CIARP 2005, LNCS 3773*, pp. 233–241, 2005.

[23]. X. F. Hermida, Andrés Corbacho, Fernando Martín, "A Braille O.C.R. for Blind People", *Proceedings of ICSPAT-96,* Boston (U.S.A.), October 1996.

Available: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.7727&rep=rep1&type=pdf [Accessed: December 23, 2010]

[24]. E.B. Hunt, Marin and P.J. Stone, *Experiments in induction*. Academic Press, NewYork, 1996.

[25]. S. I. Ibrahim and Abuhaiba, "Efficient OCR using simple features and decision trees with backtracking", t*he Arabian Journal for Science and Engineering*, vol.31, no,2B, pp.223-243, October 2006.

[26]. K. S. Fu, *Syntactic pattern recognition and application.* Prentice Hall, 1982.

[27]. R. J. Lewis, "An introduction to classification and regression Tree (CART) analysis", *2000 Annual Meeting of the Society for Academic Emergency Medicine,* Francisco, California, 2000.

[28]. Lisa Wong, Waleed Abdulla and Stephan Hussmann, "A software algorithm prototype for optical recognition of embossed Braille", *Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04),* pp.23-26, August 2004.

[29]. N. Matthew, Anyanwu and G. Sajjan, "Comparative analysis of serial decision tree classification algorithms" *International Journal of Computer Science and Security (IJCSS)*, vol. 3 , no.3, pp. 230-240, 2009.

[30]. M. Mehta, R. Agrawal and J. Rissanen, "SLIQ: a fast scalable classifier for data mining", *In EDBT96,* Avignon, France, 1996.

[31]. J. Mennens, "Optical recognition of Braille writing", *IEEE*, 1993, pp.428-431.

[32]. J. Mennens, L. Tichelen, G. François and J. J. Engelen, "Optical recognition of Braille writing using standard equipment", *IEEE transactions of rehabilitation engineering*, vol.2, no.4, pp.207-211, December 1994.

[33]. Million Meshesha, "A generalized approach to character recognition of Amharic texts", M.Sc. Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa, 2000.

[34]. Million Meshesha, “Recognition and retrieval from document image collections”, PhD Dissertation, International Institute of Information Technology Hyderabad 500 032, India, August 2008.

[35]. M. W. Kurzynski, “Decision rules for a hierarchical classifier,” *Pattern Recognition Lett*, vol.1, pp.305-310, 1983.

[36]. Néstor Falcón, M. T. Carlos, B. A. Jesús, and A. F. Miguel, “Image Processing Techniques for Braille Writing Recognition”, *EUROCAST, LNCS3643*, pp. 379 – 385, 2005.

[37]. Y. Oyama, T. Tajima, and H. Koga, “Character recognition of mixed convex-concave Braille points and legibility of deteriorated Braille points”, System and Computer in Japan, vol.28, no. 2, 1997.

[38]. V. Podgorelec, P. Kokol, B. Stiglic, I. Rozman, “Decision trees: an overview and their use in medicine”, *Journal of Medical Systems Kluwer Academic/Plenum Press*, vol. 26, no. 5, pp. 445-463, 2002.

[39]. Qing Chen, “Evaluation of OCR algorithms for images with different spatial resolutions and noises”, M.Sc. Thesis, School of Information Technology and Engineering Faculty of Engineering, University of Ottawa, 2003.

[40]. J. R. Quinlan, “Induction of decision trees”, Machine Leaning, vol.1, pp.81-106, 1986.

[41]. J. R. Quinlan, “Simplifying decision trees”, International Journal of Machine Studies, no. 27, pp. 221-234, 1987.

[42]. J. R. Quinlan, *C45: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[43]. Rasoul Safavian and David Landgrebe, “A Survey of Decision Tree Classifier Methodology”, Reprinted from IEEE Transactions on Systems, Man, and Cybernetics, IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 3, pp 660-674, May 1991.

[44]. R. Rastogi and K. Shim, “PUBLIC: a decision tree classifier that integrates building and pruning”, *Proceedings of the 24th VLDB Conference*, New York, 1998, pp. 404-415.

[45]. R. T. Ritchings, A. Antonacopoulos and D. Drakopoulos, “Analysis of scanned Braille documents”, Document Analysis Systems: World Scientific Publishing Company, 1995, pp. 413–421.

[46]. R. Robert, J. Gotwals," BRL: Braille through Remote Learning"

Available: http://www.brl.org/intro/session11/teaching.html [Accessed: December 25, 2010]

[47]. J. Shafer, R. Agrawal and M. Mehta, "Sprint: a scalable parallel classifier for data mining", Proceedings of the 22nd international conference on very large data base. Mumbai (Bombay), India, 1996, pp. 1-12.

[48]. A. Taylor, "Choosing your Braille embosser", The Braille Monitor44, vol.44, no.9, October 2001.

[49]. Teshome Alemu, "Recognition of Amharic Braille", M.Sc. Thesis, Department of Information Science, Addis Ababa University, Addis Ababa, March 2009.

[50]. P. Utgoff, and C. Brodley, "An incremental method for finding multivariate splits for decision trees", Machine Learning: *Proceedings of the Seventh International Conference*, 1990, pp.58-65.

[51]. UNESCO, *World Braille usage: national library service for the blind and physically handicapped library of congress.* Washington D.C., USA, 1990.

[52]. Vidyashankar, Hemantha Kumar, P. Shivakumara,"Rotational invariant histogram feature for recogntion of Braille symbols",University of Mystroe, Karnataka, India, 2004.

[53]. WHO Available at: http://www.who.int Active on April 1st, 2005.

[54]. Wondwossen Mulugeta , "OCR for special type of handwritten Amharic text ("YEKUM TSIFET"), M.Sc. Thesis, Department of Information Science, Addis Ababa University, Addis Ababa, 2004.

[55]. Tan, Steinbach and Kumar, *Introduction to data mining*. Addison Wesley, 2006.

[56]. ኑብየ ልዑል ዮሃንስ ዘመነ ብርሃን አዲስ አበባ ብርሃንና ሰላም ማተሚያ ቤት 1954።።

[57]. T. Agui and T. Nagao, *Computer image processing and recognition.* Tokyo: Shoho-do, 1994.

[58]. Lehal G,S. and Chandan Singh, "Feature extraction and classification for OCR of Gurmukhi script", vol.12, no.2: pp. 2-12 ,1999.

[59]. T. W. Hentzschel and P. Blenkhorn, "An optical reading systems for embossed Braille characters using a twin shadows approach", *Journal of Microcomputer Applications*, 1995, pp. 341-345.

[60]. Jie Li and Xiaoguang Yan, "Optical Braille character recognition with Support-vector machine classifier", International Conference on Computer Application and System Modeling (ICCASM 2010), vol.12, pp.219-222, 2010.

[61]. K. Jithesh, K. G. Sulochana and R. Kumar, "Optical character recognition for Malayalam",

Available:http://www.cdactvm.in/OPTICAL%20CHARACTER%20RECOGNITION%20FOR%20MALAYALAM.pdf [Accessed: February 8, 2011]

[62]. Omar Khan Durrani and K. C. Shet, "A new architecture for Brailee transcription from optically recognized Indian languages", 3rd *International CALIBER* - 2005, Cochin 2-4 February, pp.22-31, 2005.

[63]. Y. Rui, A. C. She, and T. S. Huang, "Modified fourier descriptors for shape representation–a practical approach," Available: http://citeseer.nj.nec.com/296923.html [Accessed: January 7, 2011]

[64]. Zhenfei Tai, Samuel Cheng, and Pramode Verma, "Braille document parameters estimation for optical character recognition", *ISVC 2008*, Part II, LNCS 5359, pp. 905–914, 2008.

[65]. Amany Al-Saleh, Ali El-Zaart and AbdulMalik AlSalman, "Dot detection of optical Braille images for Braille cells recognition", *ICCHP 2008*, LNCS 5105, pp. 821–826, 2008.

[66]. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 978-1-55860-901-3, San Francisco, 2006.

[67]. Farhan Abdel-Fattah and Zulkhairi Md, "Distributed and cooperative hierarchical Intrusion detection on MANETs", International Journal of Computer Applications (0975 – 8887), vol.12, no.5, pp. 32-40, 2010.

[68]. M. Gandhi and K. Srivats, "Classification algorithms in comparing classifier categories to predict the accuracy of the network intrusion detection- a machine learning approach", Research India Publications ,ISSN 0973-6107, vol.3, no.3, pp. 321–334, 2010.

[69]. I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Second edition, 2005.

[70]. Kusm Bhart, Shweta Jain and Sanyam Shukla, "Fuzzy k-mean clustering via J48 for intrusiion detection system", Kusum Bharti International Journal of Computer Science and Information Technologies (IJCSIT) , vol. 1, no.4, pp 315-318 , 2010.

[71]. Georgios Paliouras, Vangelis Karkaletsis and Constantine Spyropoulos, "Learning rules for large-vocabulary word sense disambiguation: a comparison of various classifiers", Proceedings of the 2nd International Conference on Natural Language Processing (NLP) 1835, pp. 383 – 394, Springer, 2000.

[72]. D. Han, L. Chan , and N. Zhu, "Flood forecasting using support vector machines", IWA Publishing 2007 journal of Hydroinformatics, vol.9, no.4, pp.267-276, 2007.

[73]. Alex Freitas, "Understanding the crucial differences between classification and discovery of association rules – a position paper", SIGKDD Explorations, vol. 2, no.1, July 2000.

[74]. K. Cheung, J. Kwok and C. Tsui, "Mining customer product ratings for personalized marketing" Decision Support System, vol.35, no.2, pp. 231-243, 2003.

[75]. E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines", *Proc. 1997 IEEE Workshop*, 1997, pp.276-285.

[76]. K. Cnrs, Y. Kodratoi and S. Moscatelli, "Machine learning for object recognition and scene analysis" Internationa Journal of Pattern recognition and AI, vol.8, pp.8-11, 1994.

[77]. R. Rosipal and L. Trejo, "Kernel partial least squares regression in reproducing kernel hilbert space" J. Machine learning Learn, vol. 2: pp.97-123, 2002.

[78]. M. A. Hearst, "Support vector machines," IEEE Intelligent Systems, vol.13, pp.18-28, 1998.

[79]. Berihun Girma, The educational situation of the blind: center for educational staff development, 1994.

[80]. Yaregal Assabie, "Optical character recognition of Amharic text: an integrated approach", M.Sc. Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa, 2002.

[81]. Worku Alemu, "The application of OCR techniques to the Amharic script," M.Sc. Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa, 1997.

APPENDIX

I. The First Version Amharic Braille (1917 E.C)

Vowel							
Character variant	1st	2nd	3rd	4th	5th	6th	7th

Table I. 1 List of vowels for Amharic Braille characters in the first version

Braille code	Amharic character	Braille code	Amharic character
	ሀ		ኸ
	ለ		ወ
	ሐ		ዐ
	መ		ዘ
	ሠ		ዠ
	ረ		የ
	ሰ		ደ
	ሸ		ጀ
	ቀ		ገ
	በ		ጠ
	ተ		ጨ
	ቸ		ጰ
	ኀ		ፀ
	ነ		ጸ
	ኘ		ፈ
	አ		ፐ
	ከ		

Table I. 2 List of first variant Amharic Braille characters in the first version

II. The Second Version Amharic Braille (1945 E.C)

Vowel	none					none	
Character variant	1st	2nd	3rd	4th	5th	6th	7th

Table II. 1 List of vowels for Amharic Braille characters in the second version

Braille code	Amharic character	Braille code	Amharic character
	ሀ		ኸ
	ለ		ወ
Not apply	ሐ	Not apply	ዐ
	መ		ዘ
	ሠ		ዠ
	ረ		የ
Not apply	ሰ		ደ
	ሸ		ጀ
	ቀ		ገ
	በ		ጠ
	ተ		ጨ
	ቸ		ጸ
Not apply	ኀ		ፀ
	ነ	Not apply	ጸ
	ኘ		ፈ
	አ		ፐ
	ከ		

Table II. 2 List of first variant Amharic Braille characters in the second version

Braille code	Amharic character	Braille code	Amharic character
	ሀ		ኸ
	ለ		ወ
Not apply	ሐ	Not apply	ዐ
	መ		ዘ
	ሠ		ዠ
	ረ		ይ
Not apply	ሰ		ደ
	ሸ		ጀ
	ቀ		ግ
	በ		ጥ
	ተ		ጭ
	ቸ		ጸ
Not apply	ኀ		ፀ
	ነ	Not apply	ጸ
	ኘ		ፍ
	አ		ፕ
	ከ		

Table II. 3 List of the sixth variant Amharic Braille characters in the second version

III. The Third Version Amharic Braille (1949 E.C)

Vowel						None	
Character variant	1st	2nd	3rd	4th	5th	6th	7th

Table III. 1 List of vowels for Amharic Braille characters in the third version

Braille code	Amharic character	Braille code	Amharic character
	ሀ		ኸ
	ለ		ወ
Not apply	ሐ	Not apply	ዐ
	መ		ዘ
	ሠ		ዠ
	ረ		ይ
Not apply	ሰ		ደ
	ሸ		ጀ
	ቀ		ገ
	በ		ጠ
	ተ		ጨ
	ቸ		ጸ
Not apply	ኀ		ፀ
	ነ	Not apply	ጰ
	ኘ		ፍ
	አ		ፕ
	ከ		

Table III. 2 List of basic Amharic Braille characters in the third version

IV. The Fourth Version Amharic Braille (1993 E.C)

Braille code	Dot position	Representation	Punctuation mark
	-	000000000000	ባዶ ቦታ
	3	001000000000	፡
	6 and 3	000001001000	፡
	5 and 2	000010010000	/
	36	001001000000	፦
	36 and 36	001001001001	_
	2	010000000000	፤
	45	000011000000	$
	256	010011000000	::
	2356	011011000000	( )
	356 and 6	001011000001	?
	4	000100000000	’
	235 and 5	011010000010	!
	35 and 35	001010001010	*
	456 and 35	000111001010	↑
	456 and 25	000111010010	↓
	356 and 3	001011001000	”
	25	010010000000	፥
	23	011000000000	፤
	46 and 46	000101000101	–
	34	001100000000	እና/ወይም
	6 and 2356	000001011011	[
	3 and 2356	001000011011	]

Table IV. 1 List of punctuation marks

Braille code	Dot position	Representation	Numeral
	3456 and 245	001111010110	0
	3456 and 1	001111100000	1
	3456 and12	001111110000	2
	3456 and14	001111100100	3
	3456 and 145	001111100110	4
	3456 and 15	001111100010	5
	3456 and124	001111110100	6
	3456 and 1245	001111110110	7
	3456 and 125	001111110010	8
	3456 and 24	001111010100	9
	3456 and245	001111010110	10
	123456 and1	111111100000	**a**
	123456 and12	111111110000	**b**
	123456 and14	111111100100	**c**
	123456 and145	111111100110	**d**
	123456 and15	111111100010	**e**
	123456 and124	111111110100	**f**
	123456 and1245	111111110110	**g**
	123456 and 125	111111110010	**h**
	123456 and 24	111111010100	**i**
	123456 and 245	111111010110	**j**

Table IV. 2 List of numerals

V. Visual C++ Code for Feature Extraction

```
void CABOCRView::FeatureExtraction(CDC* pDC)//module to extract feature and write to .xls file
{ CString msg;msg="Feature Extraction has completed!!!!!!!";
FILE *BFeature=fopen("Bfeature.xls","w");
CClientDC dc(this);FILE* feature=fopen("future.txt","w");
FILE* ch=fopen("normal.txt","w");
int h=0,w=0,BCount=0,k=0,n=0;bool val=FALSE,num_m=0;
int i=0,j=0,x=0, y=0,dotcount=0,u=0,m=0,p=0,r=0;
struct BCode
{int d[6];//for consonont int vd[6];//for vowel};
BCode ABChar[2000];
int BTemp[6];//to hold valid braille dots
//Algorithm 1
int xmin=DotWAry[0]; int ymin=DotHAry[0];        int xmax=bm.bmWidth;
int ymax=bm.bmHeight; int linnum=(ymax-ymin)/50; int colnum=(xmax-xmin)/30;

while(y<linnum)//for(int y=0;y<linnum;y++){
          x=0;  j=ymin+y*74;
         while(x<colnum) {
           m=0;  i=xmin+x*44;
                if(i+30<bm.bmWidth)    { k=i;
                        while(k<=i+30) {
                                if(j+50<bm.bmHeight) {  n=j;
                                        while(n<=j+50) {
                                                for(int p=n;p<n+10;p++){
                                                   for(int s=k;s<k+10;s++){
                                                           if(pDC->GetPixel(s,p)==0)
                                                           {dotcount++;} }
                        if(dotcount>=10) {BTemp[m++]=1;}
```

Continued in the next page…

Figure V. 1 Visual C++ Code for Feature Extraction

```
                    else  {BTemp[m++]=0;}
                    dotcount=0;       n+=20; }} k+=20; }}
//Algorithm 2
while(DotHorProj[j]) { i=0;

        while(DotVerProj[i]) { p=0;
                    for(int w=i;w<i+2;w++){
                        for(int h=j;h<j+3;h++){
                            u=0; val=FALSE;
                            while(dot[u][0]) {

        if((dot[u][1]==DotHorProj[h])&&(dot[u][0]==DotVerProj[w])) {
                            BTemp[p++]=1; val=TRUE; }
                            u++;    }
                        if(val==FALSE) { BTemp[p++]=0; }}}
/Algorithm 3
while(col[h][2]) { w=0;
        while(line[w][1]) { k=0;  r=0;
                for(int i=0;i<2;i++) {
                    for(int j=0;j<3;j++){
                        for(int d=line[w][i];d<line[w][i]+10;d++){
                            for(int g=col[h][j];g<col[h][j]+10;g++){
                                if(pDC->GetPixel(d,g)==0) {
                            dotcount++;}}};
                        if(dotcount>=10) {BTemp[k++]=1;}
                        else     {BTemp[k++]=0;}
                        dotcount=0; }}
        if(consonant(BTemp)==0) {
            if(BCount-1>=0) {
        int temp=BCount-1;
```

Continued in the next page…

```
        for(int a=0;a<6;a++){
            ABChar[temp].vd[a]=BTemp[a];}}
         else{  for(int a=0;a<6;a++){
                        ABChar[BCount].d[a]=0;
                        ABChar[BCount].vd[a]=BTemp[a];
                        }BCount++;}}
 else if
((numberMode(BTemp)==1)||(PunctuationMarkPfix(BTemp)==1)||(puncMark(BTem
        { for(int a=0;a<6;a++){
                ABChar[BCount].d[a]=BTemp[a];
                ABChar[BCount].vd[a]=0;
                } BCount++;}
else if((BCount-1)>=0 &&
(numberMode(ABChar[BCount1].d)==1)||(puncMark(ABChar[BCount-1].d)==1))
        { int temp=BCount-1; for(int a=0;a<6;a++){
                ABChar[temp].vd[a]=BTemp[a]; }}
else if(consonant(BTemp)==1) {for(int a=0;a<6;a++){
                                ABChar[BCount].d[a]=BTemp[a];
                                ABChar[BCount].vd[a]=0; }
                                BCount++;}
else if(consonant(BTemp)==0 && PunctuationMarkPfix(BTemp)==0)
                {int temp=BCount-1; for(int a=0;a<6;a++){
                                ABChar[temp].vd[a]=BTemp[a]; }}
else{   for(int a=0;a<6;a++){
                ABChar[BCount].d[a]=2;
                ABChar[BCount].vd[a]=2; } BCount++;}
        w+=2;//w++;//i+=2;//x++;} h+=3;//h++;//j+=3;//y++;
        }
        int Count=0,c=0;
```

Continued in the next page…

```
fprintf(BFeature,"%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t","c1d1","c1d2","c1d3","c1d4","c1d5","c1d6","c2d1","c2d2","c2d3","c2d4","c2d5","c2d6","class"); fprintf(BFeature,"\n","");

while(ABChar[Count].d[0]==0 ||ABChar[Count].d[0]==1 || ABChar[Count].d[0]==2)
{
fprintf(BFeature,"%d\t%d\t%d\t%d\t%d\t%d\t",ABChar[Count].d[0],ABChar[Count].d[1],ABChar[Count].d[2],ABChar[Count].d[3],ABChar[Count].d[4],ABChar[Count].d[5]);
fprintf(BFeature,"%d\t%d\t%d\t%d\t%d\t%d\t%s\t",ABChar[Count].vd[0],ABChar[Count].vd[1],ABChar[Count].vd[2],ABChar[Count].vd[3],ABChar[Count].vd[4],ABChar[Count].vd[5],"?"); fprintf(BFeature,"\n",""); Count++;

}fprintf(BFeature,"Total Braille Character=%d",BCount);
fclose(BFeature); AfxMessageBox(msg); }
bool CABOCRView::numberMode(int BTemp[])
{int const numMode[2][6]={0,0,1,1,1,1, 1,1,1,1,1,1}; bool val=FALSE;
        for(int a=0;a<2;a++){
                for(int b=0;b<6;b++){   val=FALSE;
                        if(numMode[a][b]==BTemp[b]) { val=TRUE; }
                        else    return 0; }}
        return 1; }
bool CABOCRView::puncMark(int BTemp[]){
int const Punc[10][6]={0,0,0,0,1,0, 0,0,1,0,0,1, 0,0,1,0,1,1, 0,0,1,0,1,0, 0,0,0,0,0,1, 0,0,1,0,0,0, 0,0,0,1,1,1, 0,1,1,0,0,1, 0,0,0,1,0,1, 0,1,1,0,1,0}; bool val=FALSE;
  for(int a=0;a<10;a++) {
        for(int b=0;b<6;b++){ val=FALSE;
                if(Punc[a][b]==BTemp[b]) { val=TRUE; }
                else      return 0; }
        return 1; }}
bool CABOCRView::consonant(int BTemp[]){
        bool val=TRUE; int const vowel[6][6]={0,1,0,0,0,1,1,0,1,0,0,1,0,1,0,1,0,0
                                        ,1,0,0,0,0,0,1,0,0,0,1,0,1,0,1,0,1,0};
```

Continued in the next page…

```
        for(int a=0;a<6;a++){ val=FALSE;
                for(int b=0;b<6;b++){
                        if(vowel[a][b]==BTemp[b]){ val=TRUE;}
                        else { val=FALSE;break; }}
                if(val) return 0; }
        return 1; }
bool CABOCRView::PunctuationMarkPfix(int BTemp[]){bool val=FALSE;
int const PuncMarkS[10][6]={0,0,1,0,0,0, 0,1,0,0,0,0, 0,0,1,1,0,0, 0,1,1,0,1,1,
0,0,0,1,1,0, 0,0,1,1,0,0, 0,1,0,0,1,1, 0,1,1,0,0,0, 0,1,0,0,1,0, 0,0,0,1,0,0 };
for(int a=0;a<10;a++){ val=FALSE;
        for(int b=0;b<6;b++){
                if(PuncMarkS[a][b]==BTemp[b])
                        val=TRUE;
                else { val=FALSE;break; }}
                if(val) return 0; } return 1; }
```

VI. Sample ARFF File for Training J48 and SMO in WEKA

```
@relation Braille

@attribute cell1d1 {0,1}
@attribute cell1d2 {0,1}
@attribute cell1d3 {0,1}
@attribute cell1d4 {0,1}
@attribute cell1d5 {0,1}
@attribute cell1d6 {0,1}
@attribute cell2d1 {0,1}
@attribute cell2d2 {0,1}
@attribute cell2d3 {0,1}
@attribute cell2d4 {0,1}
@attribute cell2d5 {0,1}
@attribute cell2d6 {0,1}
@attribute character
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,2
5,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,
47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68
,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,9
0,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,
109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,1
25,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,14
1,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157
,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,
174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,1
90,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,20
6,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222
,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,
239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,2
55,256,257,258,259,260,261,262,263,264,265,266,267,268,269,270,27
1,272,273,274,275,276,277,278,279,280,281}

@data

1,1,0,0,1,0,0,1,0,0,0,1,1
1,1,0,0,1,0,1,0,1,0,0,1,2
1,1,0,0,1,0,0,1,0,1,0,0,3
1,1,0,0,1,0,1,0,0,0,0,0,4
1,1,0,0,1,0,1,0,0,0,1,0,5
1,1,0,0,1,0,0,0,0,0,0,0,6
1,1,0,0,1,0,1,0,1,0,1,0,7
1,1,1,0,0,0,0,1,0,0,0,1,8
1,1,1,0,0,0,1,0,1,0,0,1,9
1,1,1,0,0,0,0,1,0,1,0,0,10
1,1,1,0,0,0,1,0,0,0,0,0,11
1,1,1,0,0,0,1,0,0,0,1,0,12
1,1,1,0,0,0,0,0,0,0,0,0,13
1,1,1,0,0,0,1,0,1,0,1,0,14
1,1,0,0,0,1,0,1,0,0,0,1,15
1,1,0,0,0,1,1,0,1,0,0,1,16
1,1,0,0,0,1,0,1,0,1,0,0,17
1,1,0,0,0,1,1,0,0,0,0,0,18
1,1,0,0,0,1,1,0,0,0,1,0,19
1,1,0,0,0,1,0,0,0,0,0,0,20
1,1,0,0,0,1,1,0,1,0,1,0,21
1,0,1,1,0,0,0,1,0,0,0,1,22
1,0,1,1,0,0,1,0,1,0,0,1,23
1,0,1,1,0,0,0,1,0,1,0,0,24
1,0,1,1,0,0,1,0,0,0,0,0,25
1,0,1,1,0,0,1,0,0,0,1,0,26
1,0,1,1,0,0,0,0,0,0,0,0,27
1,0,1,1,0,0,1,0,1,0,1,0,28
0,1,1,1,0,0,0,1,0,0,0,1,29
0,1,1,1,0,0,1,0,1,0,0,1,30
0,1,1,1,0,0,0,1,0,1,0,0,31
0,1,1,1,0,0,1,0,0,0,0,0,32
0,1,1,1,0,0,1,0,0,0,1,0,33
0,1,1,1,0,0,0,0,0,0,0,0,34
```

VII. Mapping between Number Class Labels and Amharic Characters

ሀ	1	ሁ	2	ሂ	3	ሃ	4	ሄ	5	ህ	6	ሆ	7
ለ	8	ሉ	9	ሊ	10	ላ	11	ሌ	12	ል	13	ሎ	14
ሐ	15	ሑ	16	ሒ	17	ሓ	18	ሔ	19	ሕ	20	ሖ	21
መ	22	ሙ	23	ሚ	24	ማ	25	ሜ	26	ም	27	ሞ	28
ሠ	29	ሡ	30	ሢ	31	ሣ	32	ሤ	33	ሥ	34	ሦ	35
ረ	36	ሩ	37	ሪ	38	ራ	39	ሬ	40	ር	41	ሮ	42
ሰ	43	ሱ	44	ሲ	45	ሳ	46	ሴ	47	ስ	48	ሶ	49
ሸ	50	ሹ	51	ሺ	52	ሻ	53	ሼ	54	ሽ	55	ሾ	56
ቀ	57	ቁ	58	ቂ	59	ቃ	60	ቄ	61	ቅ	62	ቆ	63
በ	64	ቡ	65	ቢ	66	ባ	67	ቤ	68	ብ	69	ቦ	70
ተ	71	ቱ	72	ቲ	73	ታ	74	ቴ	75	ት	76	ቶ	77
ቸ	78	ቹ	79	ቺ	80	ቻ	81	ቼ	82	ች	83	ቾ	84
ኀ	85	ኁ	86	ኂ	87	ኃ	88	ኄ	89	ኅ	90	ኆ	91
ነ	92	ኑ	93	ኒ	94	ና	95	ኔ	96	ን	97	ኖ	98
ኘ	99	ኙ	100	ኚ	101	ኛ	102	ኜ	103	ኝ	104	ኞ	105
አ	106	ኡ	107	ኢ	108	ኣ	109	ኤ	110	እ	111	ኦ	112
ከ	113	ኩ	114	ኪ	115	ካ	116	ኬ	117	ክ	118	ኮ	119
ኸ	120	ኹ	121	ኺ	122	ኻ	123	ኼ	124	ኽ	125	ኾ	126
ወ	127	ዉ	128	ዊ	129	ዋ	130	ዌ	131	ው	132	ዎ	133
ዐ	134	ዑ	135	ዒ	136	ዓ	137	ዔ	138	ዕ	139	ዖ	140
ዘ	141	ዙ	142	ዚ	143	ዛ	144	ዜ	145	ዝ	146	ዞ	147
ዠ	148	ዡ	149	ዢ	150	ዣ	151	ዤ	152	ዥ	153	ዦ	154
የ	155	ዩ	156	ዪ	157	ያ	158	ዬ	159	ይ	160	ዮ	161
ደ	162	ዱ	163	ዲ	164	ዳ	165	ዴ	166	ድ	167	ዶ	168
ጀ	169	ጁ	170	ጂ	171	ጃ	172	ጄ	173	ጅ	174	ጆ	175
ገ	176	ጉ	177	ጊ	178	ጋ	179	ጌ	180	ግ	181	ጎ	182
ጠ	183	ጡ	184	ጢ	185	ጣ	186	ጤ	187	ጥ	188	ጦ	189
ጨ	190	ጩ	191	ጪ	192	ጫ	193	ጬ	194	ጭ	195	ጮ	196
ጰ	197	ጱ	198	ጲ	199	ጳ	200	ጴ	201	ጵ	202	ጶ	203
ፀ	204	ፁ	205	ፂ	206	ፃ	207	ፄ	208	ፅ	209	ፆ	210
ጸ	211	ጹ	212	ጺ	213	ጻ	214	ጼ	215	ጽ	216	ጾ	217
ፈ	218	ፉ	219	ፊ	220	ፋ	221	ፌ	222	ፍ	223	ፎ	224
ፐ	225	ፑ	226	ፒ	227	ፓ	228	ፔ	229	ፕ	230	ፖ	231
ቨ	232	ቩ	233	ቪ	234	ቫ	235	ቬ	236	ቭ	237	ቮ	238
.	239	፡	240	/	241	፦	242	--	243	፧	244	$	245
::	246	()	247	[	248	]	249	_	250	እና	251	፤	252
'	253	፥	254	*	255	↑	256	↓	257	ባዶ ቦታ	258	1	259
2	260	3	261	4	262	5	263	6	264	7	265	8	266
9	267	0	268	፩	269	፪	270	፫	271	፬	272	፭	273
፮	274	፯	275	፰	276	፱	277	፲	278	?	279	!	280
”	281												

Table VII. 1 Mapping between Amharic characters and class labels

VIII. Python Code for Translation

```
option=input(" Well Come to AOBR:\n Choose 1 or 2: 1 for CLI, 2 for GUI:")
if(option=='1'):
   file=input("Enter the file name:")

dict1={'1:1':'ሀ','2:2':'ሁ','3:3':'ሂ','4:4':'ሃ','5:5':'ሄ','6:6':'ህ','7:7':'ሆ','8:8':'ለ','9:9':'
ሉ','10:10':'ሊ','11:11':'ላ','12:12':'ሌ','13:13':'ል','14:14':'ሎ','15:15':'ሐ','16:16':'ሑ','
17:17':'ሒ','18:18':'ሓ','19:19':'ሔ','20:20':'ሕ','21:21':'ሖ','22:22':'መ','23:23':'ሙ','2
4:24':'ሚ','25:25':'ማ','26:26':'ሜ','27:27':'ም','28:28':'ሞ','29:29':'ሠ','30:30':'ሡ','31:
31':'ሢ','32:32':'ሣ','33:33':'ሤ','34:34':'ሥ','35:35':'ሦ','36:36':'ረ','37:37':'ሩ','38:38':'
ሪ','39:39':'ራ','40:40':'ሬ','41:41':'ር','42:42':'ሮ','43:43':'ሰ','44:44':'ሱ','45:45':'ሲ','46
:46':'ሳ','47:47':'ሴ','48:48':'ስ','49:49':'ሶ','50:50':'ሸ','51:51':'ሹ','52:52':'ሺ','53:53':'ሻ
','54:54':'ሼ','55:55':'ሽ','56:56':'ሾ','57:57':'ቀ','58:58':'ቁ','59:59':'ቂ','60:60':'ቃ','61:6
   list1=[]
   fname=open(file,'r')
   ftxt=open('D:/Thesis/Experiment/Amharic1.txt','w',encoding='utf-8')
   for line in fname.readlines():
      if line.isspace():
         Continue
      if ((not line.startswith('P'))| (not line.startswith(','))| (not line.startswith('i'))):
         columns = line.split(",")
         list1.append(columns[2].rstrip('\n'))
   fname.close()
   for i in range(0,len(list1)):
      for x in dict1.keys():
         if list1[i]==x:
           // print (dict1[x])
            ftxt.write(dict1[x])
   ftxt.close()
elif(option=='2'):
   f=input("GUI \n Enter the file name:")

dict2={'1':'ሀ','2':'ሁ','3':'ሂ','4':'ሃ','5':'ሄ','6':'ህ','7':'ሆ','8':'ለ','9':'ሉ','10':'ሊ','11':'ላ
','12':'ሌ','13':'ል','14':'ሎ','15':'ሐ','16':'ሑ','17':'ሒ','18':'ሓ','19':'ሔ','20':'ሕ','21':'
ሖ','22':'መ','23':'ሙ','24':'ሚ','25':'ማ','26':'ሜ','27':'ም','28':'ሞ','29':'ሠ','30':'ሡ','
31':'ሢ','32':'ሣ','33':'ሤ','34':'ሥ','35':'ሦ','36':'ረ','37':'ሩ','38':'ሪ','39':'ራ','40':'ሬ','4
1':'ር','42':'ሮ','43':'ሰ','44':'ሱ','45':'ሲ','46':'ሳ','47':'ሴ','48':'ስ','49':'ሶ','50':'ሸ','51':'
ሹ','52':'ሺ','53':'ሻ','54':'ሼ','55':'ሽ','56':'ሾ','57':'ቀ','58':'ቁ','59':'ቂ','60':'ቃ','61':'ቄ',
'62':'ቅ','63':'ቆ','64':'በ','65':'ቡ','66':'ቢ','67':'ባ','68':'ቤ','69':'ብ','70':'ቦ','71':'ተ','72':
'ቱ','73':'ቲ','74':'ታ','75':'ቴ','76':'ት','77':'ቶ','78':'ቸ','79':'ቹ','80':'ቺ','81':'ቻ','82':'
ቼ','83':'ች','84':'ቾ','85':'ኀ','86':'ኁ','87':'ኂ','88':'ኃ','89':'ኄ','90':'ኅ','91':'ኆ','92':
   lst=[]
   a=open(f,'r')
```

Continued in the next page…

Figure VIII. 1 Sample Python code for translation

```
    file1=open('D:/Thesis/Experiment/Amharic.txt','w',encoding='utf-8')

    for lne in a.readlines():
        if lne.isspace():
            Continue
        if not lne.startswith('@'):
            column = lne.split(",")
            lst.append(column[12].rstrip('\n'))
    a.close()

    for j in range(0,len(lst)):
        for y in dict2.keys():
            if lst[j]==y:
                print (dict2[y])
                file1.write(dict2[y])
    file1.close()
else:
    print("enter the correct option")
```

IX. Sample Recognized Amharic Braille Characters

Braille code	Amharic character
	ጥንቃቄ
	ክርሥቲያን
	ማርያም
	የምክርና
	ተዘጋ
	ቀድሞ
	መፅሀፍ
	ኢትዮጵያ
	ለህብረተሠብ

Table IX. 1 Sample recognized Braille images

Declaration

I declare that the thesis is my original work and has not been presented for a degree in any other university.

_____

July 13, 2011

This thesis has been submitted for examination with my approval as university advisor.

_____

Dr. Million Meshesha