IMPROVING THE COMPRESSION ALGORITHMS PERFORMANCE FOR SCANNED AMHARIC PDF FILES

A Thesis Presented

by

Haimanot Andargachew

to

The Faculty of Informatics

of

St. Mary's University

In Partial Fulfillment of the Requirements
for the Degree of Master of Science

in

Computer Science

January 2020

ACCEPTANCE

Improving the Compression Algorithms Performance for

Scanned Amharic PDF Files

By

Haimanot Andargachew

Accepted by the Faculty of Informatics, St. Mary's University, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Thesis Examination Committee:

____

Internal Examiner

____

External Examiner

____

Dean, Faculty of Informatics

January 2020

DECLARATION

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been duly acknowledged.

Haimanot Andargachew

Signature

Addis Ababa

Ethiopia

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

Million Meshesha (Ph.D)

Signature

Addis Ababa

Ethiopia

January 2020

ACKNOWLEDGEMENTS

First and for most I would like to thank Almighty GOD for giving me strength, courage and patience in order to accomplish this research.

I would like to express my special appreciation and thanks to my advisor Dr. MILLION MESHESHA, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a researcher. His sage advice, encouragement, guidance and support enabled me to develop an understanding of the subject. Besides his expertise, I really appreciate for his concern and perspective advice throughout my thesis work. This accomplishment would not have been possible without him. He is honestly the GREATEST teacher and advisor I have ever known.

I would like to extend my appreciation to my best friends Berekeab, Addissu, Yohannes and Gizachew for their constructive comment and constant assistance and encouragement they rendered to me.

Abstract

The advancement and accessibility of digital computers and the introduction of the Internet and World Wide Web lead to in massive information explosion all over the world. There for large amount of newspapers, magazines and printed documents available with numerous information and knowledge of different areas. PDF file format documents, facilitates office automation and the move towards paperless office. PDFs can become inconveniently large when they contain a large amount of high-resolution content such as images and Graphics, or even just a very large number of pages. To make the information and knowledge embedded in these PDF documents accessible and share to the public there is a need to minimize the data size using different mechanisms.

This study has been conducted to develop Amharic PDF file document compression system by applying an effective page segmentation technique that can identify text and non-text blocks with the aim of reconstructing PDF document layouts to optimize memory space requirement and bandwidth for transmission.

The first step of the proposed approach is separating textual and non-textual objects. After a applying combination of page segmentation techniques, namely: connected component with Dilation and connected components Area, Height and width analysis techniques is applied to detect a graphics part of a document. Based on the experiment on the average 78% accuracy rate is achieved from the proposed approach. The next step after textual and non-textual separation is column block detection for textual objects. Similar page segmentation techniques are applied to segment column layout. The proposed technique accurately identified column layout with an accuracy of 89%, thereby all coordinate information's about column block is stored for reconstructing stage. Finally, the extracted objects are compressed using Huffman compression algorithms. The proposed approach experimented on different PDF documents and compresses the extracted objects with compression ratio of less than 50%, which is better compression result than existing commercial compression tools.

The proposed approach also capable of reconstructing the compressed data after decompression. Based on the stored layout coordinate information the original PDF documents non-textual blocks and textual columns reconstructed on the average 74% accuracy. From correctly segmented column and paragraph block the proposed techniques 92% accuracy rates. However, the performance of the proposed method greatly affected black shades in PDF document images while scanning, irregular shaped images with non-rectangular shaped text blocks results in loss of some text and difficult to segmentation.

Key Words: Data Compression, OCR, Page layout segmentation, Portable Document file

List of Acronyms

ASCII	American Standard Code for Information Interchange
CDIS	Compression for Document Image System
CMYK	Cyan, Magenta, Yellow, and Key color mode
HTTP	Hyper Text Transfer Protocol
LZ77	Lempel–Ziv 1977
LZW	Lempel–Ziv–Welch
MATLAB	Matrix Laboratory
OCR	Optical Character Recognition.
PDF	Portable Document Format
PPM	Prediction by Partial Matching
SVG	Scalable Vector Graphics
UNICODE	Universal Character Encoding
XML	Extensible Markup Language

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER ONE

INTRODUCTION

1.1 Background

Data Compression is the process of converting an input data stream to another data stream that has a smaller size[1]. Data compression will encode or replace the original information or representation by a fewer bit characters which is reduced in size. Compression is very useful because it reduces the usage of resources required to store and transmit. This compressed file can be reversed to obtain the original file with the help of decompression[1]. The two types of compression techniques available these are lossy compression and lossless compression[1]. The lossy compression results in some loss of data from the original while performing the decompression process. The lossless compression, on the other hand, retains its original file exactly without any loss of data.

The Portable Document Format (PDF) is a file format developed by Adobe in the 1990s to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems[2]. Based on the PostScript language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, vector graphics, raster images and other information needed to display it. PDF file format is highly interesting files to compress, because they contain already pre-compressed data, but still have enough 'room' for further compression[3].

PDF compression is the reduction in size of a PDF image in order to make it compatible with processes such as web uploads, printing, and attaching to emails[4]. PDF compression is important because PDF files can be very bulky, given the amount of information that they contain. But compressing PDF file format is not easy like other textual file formats because the primarily use of PDF file format is for presentation purpose and we need another third-party tool to access the internal object of the PDF documents. One possibility for textual image compression is to perform optical character recognition (OCR) on the text and transmit (or store) the ASCII codes for the characters, along with some information about their position on the page[4]. The problem with this is that recognition is not completely reliable, particularly if unusual fonts, foreign languages or mathematical expressions are being scanned.

Language dependent compression systems are developed and well demonstrated for many languages in the worlds such as Arabic, Chinese, Hindus[5][6][7]. Document images segmentation and compression systems with high accuracy rate are commercially available for use for Latin scripts and some of the oriental languages[8]. However, for those languages in Africa and Ethiopia with their own scripts, much attention is not given for developing robust compression system to minimize memory space and speed up data communication. Since developing effective data extraction and compression system (that successfully extract PDF contents and compress of varying quality, font, size and style) is a continuing process, this study aims to propose a data compression approach for Amharic PDF files.

1.2 Statement of the Problem

Different types of information are uploaded in the Internet, such as text document, document images and multimedia data. PDF formats are usually designed to compress information as much as possible (since these can tend to become very large files)[9]. PDF file format documents, facilitates office automation and the move towards paperless office. Nowadays the availability of the internet anyplace and on anything facilitates the distribution and publication of documents in different languages including Amharic. PDF files are especially useful for documents such as magazine articles, product brochures, or flyers in which you want to preserve the original graphic appearance online.

Text compression algorithms are normally defined in terms of a source alphabet Σ of 8-bit ASCII codes and 16-bit UNICODE. We consider choosing Σ to be an alphabet whose symbols are the words of English or, in general, alternate maximal strings of alphanumeric characters and non-alphanumeric characters.

These days a number of Amharic documents are available in PDF format. The difference in compression ratios between different compression tools is quite small. This is probably caused by the pre-compressed nature of the PDF file[7]. To show compression performance for Amharic PDF file a comparison is made among 7Z, Gzip and Winzip in terms of compression ratio.

- ZIP file format: one popular instance of compression, that many computer users are familiar with is the ZIP file format, which, as well as providing compression, acts as an archiever, storing many files in a single output file.
- GZip is also the other data compression software: it takes a chunk of data and makes it smaller in size. The original data can be restored by un-zipping the compressed file. It is relevant to web apps and web sites because the HTTP protocol includes the ability to gzip data that is being sent.
- 7Z file format, a popular Open Source archive format introduced by 7-Zip, providing higher compression ratio than RAR, and now supported by many archive managers.

Experimental results of ZIP, Zip and 7Z software's in compressing Amharic PDF files is presented blow in Table 1.1 Comparison of Amharic PDF Compression performance of known software

Table 1-1 Comparison of Amharic PDF Compression performance of known software

File Name	**Original Size**	**Compression Using 7Z**		**Compression Using Zip**		**Compression Using GZip**	
		Size After Compression	**Performance**	**Size After Compression**	**Performance**	**Size After Compression**	**Performance**
addis-times.PDF	3.69 MB	3.46 MB	7%	3.46 MB	7%	3.55 MB	4%
feteh-gazeta-no-185.PDF	3.42 MB	3.29 MB	4%	3.31 MB	4%	3.31 MB	3%
News.PDF	172KB	152 KB	12%	155 KB	10%	155 KB	6%

In the above experiment sample PDF Amharic Online News and Magazines with MB and KB were used to make a compression performance analysis. Experiment conducted on the three-compression application software's (7Z, Zip and GZIP), shows that they are not effective for Amharic text PDF files at all. This is because the compression software considers the Amharic PDF as an image whether it contains text or not. As a result, there is a need to explore the way to design a compression that recognize Amharic textual image files.

Different researchers attempted to develop language dependent compression algorithm for Arabic[5], Chinese[6] and Hindus[10] languages to address the problem of efficiency and effectiveness of the compression algorithm developed for Latin scripts. To the researcher's knowledge goes, there is no work on the development of a compression approach for Amharic language script PDF files.

PDF files are often ideal candidates for compression since the purpose of PDF as a format in the first place is ease of sharing and reading[11]. PDF is often the preferred format for document sharing and storage due to its universality. PDF files look the same regardless of the machine or operating system on which they are opened; all that is required is a free PDF reader such as Adobe Reader[11]. However, PDFs can become inconveniently large when they contain a large amount of high-resolution content such as images and Graphics, or even just a very large number of textual pages. Large PDF files sometimes exceed email attachment limits; load slowly, especially from web pages; take up a lot of storage space; and are generally difficult to work with. This, of course, is where PDF compression comes in. Hence, this study explores on various PDF pages layout segmentation techniques for identifying text, non-text and column blocks, for recognizing and compressing texts in them and reconstructs back to original PDF file. To this end, the following research questions are formulated, investigated and answered.

- Which page segmentation technique is effective for identifying textual, non-textual and column block in real life PDF documents?
- Which compression algorithm is more suitable to compress Amharic PDF document files?
- What kind of page column layout detection and reconstruction techniques is fitting in order to maintain the original PDF document layouts?
- To what extent the proposed model improves the degree of compression of Amharic PDF files.

1.3 Objective of the Study

1.3.1 General objective

The general objective of the study is to design Amharic PDF files compression approach that identify layout of the PDF documents with the aim of recognizing the text block and reconstructing PDF document layouts to optimize memory space requirement and bandwidth for transmission.

1.3.2 Specific objectives

On the way of attaining the general objective, the study specifically achieves the following objectives.

- To review previously proposed related works on non-English languages compression and identifies suitable techniques and approach for Amharic PDF files compression.
- To identify different layouts of real-life PDF file documents.
- To prepare dataset for testing and for evaluation of the proposed approach
- To explore and select potential page segmentation techniques for identifying textual, non-textual and column layouts of real-life PDF documents
- To design a technique for Amharic PDF page layout segmentation, compression and reconstruction after recognition
- To evaluate the performance of the proposed approach in compressing Amharic PDF files.

1.4 Methodology

It is evident that certain set of steps are usually required to accomplish a certain task. These set of steps could guide which activity to do first and keep on doing in a chronological order. The word methodology refers to a documented approach which is used to perform activities in a manner which is coherent, consistent, accountable and repeatable. Methodology is a process that mainly consists of intellectual activities. Usually only the end goal of the methodological process is manifested as the product or result of the physical work[12].

Methodology provides a way to achieve the objectives of a research problem. Literatures, such as books, journal articles, conference proceedings and the Internet about PDF data extraction in general and PDF layout segmentation and data compression in particular have been intensively reviewed in order to acquire detailed understanding of the subject matter and the research areas. Also, the past and present research works on semantic and other languages reviewed to have a better background on the best performing algorithms and techniques regarding to data compression. Therefore, in order to achieve the specific and general objectives of the study and answer the research questions, the following step by step procedure and methods are used.

1.4.1. Study design

This research follows an experimental research, which uses manipulation and controlled testing to understand causal processes. This type of research come up with conclusions which are capable of being verified by observation or experiment[13]. Following experimental research dataset preparation, techniques identification, system design, system development and evaluation are the procedures that this study follows[13].

1.4.2 Dataset collection and preparation

Amharic PDF Documents that contain graphics, columns and paragraph have been collected from various sources to measure the performance of the proposed approach. PDF Amharic documents with various font styles, sizes and types collected from Amharic news PDF files from different magazines and from online sites, such as the Walta Information Center, BBC and FANA Amharic.

1.4.3 Implementation tool

Different researchers had used Java, MATLAB and Python for image pre-processing, segmentation, feature extraction and to compressing documents. The reason is that this programming languages have a lot of built in methods for image processing. Due to this, this research uses MATLAB for Image segmentation, extraction and developing a prototype and Python to implement compression and reconstruction.

1.4.4 Evaluation procedure

The performance of the proposed approach is tested at several stages. Since, this study focuses on PDF objects layouts segmentation, compression of extracted objects and reconstruction, the performance of the proposed approach layout segmentation and reconstruction is measured by direct mapping, which determines the performance by finding the correspondences between detected entities and ground truth.

Depending on the nature of the application there are various criteria to measure the performance of a compression approach. When measuring the performance, the main concern would be the space efficiency. Since the compression behavior depends on the redundancy of symbols in the source file, it is difficult to measure performance of a compression algorithm in general[14]. The performance depends on the type and the structure of the input source. Additionally, the compression behavior depends on the category of the compression algorithm: lossy or lossless. If a lossy compression algorithm is used to compress a particular source file, the space efficiency and time efficiency would be higher than that of the lossless compression algorithm. There are different measurements to evaluate the performances of those compression systems. Following are measurements used to evaluate the performance of the proposed approach.

Degree of Compression: Compression Ratio calculated by finding the ratio between the compressed and original file.

$$\frac{Uncompressed\ Size - compressed\ size}{Uncompressed\ Size} \qquad \text{Eq. (1-1)}$$

The above Equation 1.1 shows the formula to calculate Degree of Compression. Where, Uncompressed Size is the size of the original file before applying compression algorithms and compressed size is the size of the file after applying compression algorithms.

1.5 Scope and limitation of the study

This research is a pioneer research on PDF files compression in Amharic language. The main intent of this research is investigating the way to compress the contents of the PDF documents using different approaches that are insensitive to font difference and word variants thereby enhance effectiveness and efficiency of the existing compression systems. This research follows different approaches for segmentation, extraction and compression of the PDF files. In addition, for data compression, the research applies lossy compression for extracted text data. This study experiments some available page column layouts segmentation and reconstruction techniques for multiple columns in Amharic PDF files. Among many available algorithms, preferred techniques study and test in real-life PDF documents. The performance of the proposed page column and paragraph segmentation and reconstruction techniques are measured by sample scanned PDF documents from website documents, newspapers and magazines.

Real-life PDF documents have different physical and logical page layouts. However, this study only focuses on text and non-text separation, column block detection and reconstruction. However, segmentation and reconstruction of other page layouts such as header, table, footer and other formats are out of the coverage of the study. Also handwritten documents recognition are out of the scope of this study. Preprocessing, text segmentation and recognition stage of an OCR are adopted from Google developed Tesseract OCR engine.

1.6 Significance of the study

Different PDF Documents articulated in Amharic scripts are loaded highly in information centers, libraries, public media and government institutes. Storing and sharing this document need high storage capacity and high bandwidth for transferring. Compression reduces data size and transmission time. This study puts its own contribution in developing a new approach for Amharic PDF files compression system. It can also be implemented in different governmental and non-governmental institutions to move towards efficient storage management. In addition, it has an inevitable contribution for magazines and newspaper as a data archiving tool. Furthermore, it can be used as an input for future works which might be aimed to develop a full-fledged Amharic PDF file Compression approach.

1.7 Organization of the Study

The first chapter presents the background of the study, statement of the problem, general and specific objectives of the study, methodology of the study, scope and limitation of the study and significance of the research results.

The second chapter is a literature review on data compression, document image page segmentation techniques and overview of PDF file format. It also includes a review on the history and characteristics of the Amharic writing system and different type of documents written in Amharic language. Finally, related research works on document image compression and page segmentation are presented.

In the third chapter, the algorithms and techniques used in the study are presented. The chapter also includes the performance measurement techniques and formulas for the proposed PDF documents segmentation and compression approach.

The fourth chapter presents implementations of the proposed system, techniques and different experimentations. This chapter also addresses integration and evaluation of the performance of the system. The chapter finally addresses findings and challenges during the experiment.

The last chapter, chapter five presents conclusions based on the findings of the study and forward recommendations for further research works.

CHAPTER TWO
LITERATURE REVIEW

2.1 Overview

In the last decade, we have been witnessing a transformation or evolution in the way we communicate, and the process is still under way. This transformation includes the ever-present, ever-growing Internet; the explosive development of mobile communications; and the ever-increasing importance of video communication[1]. Data compression is one of the enabling technologies for each of these aspects of the multimedia revolution. It would not be practical to put images, let alone audio and video, on websites if it were not for data compression algorithms. Cellular phones would not be able to provide communication with increasing clarity without proper compression. The advent of digital TV would not be also possible without compression. Data compression, which for a long time was the domain of a relatively small group of engineers and scientists, is now ubiquitous[1].

Data compression is the process of encoding information using fewer bits than the original representation will use; it is the process that is used to reduce the physical size of information[15]. Compression is just about everywhere, all images that can be gotten from the web are compressed[16]. The reason we need data compression is that more and more of the information that we generate, and use is in digital form consisting of numbers represented by bytes of data. And the number of bytes required to represent multimedia data can be huge.

2.2 Data Compression Techniques

Compression techniques are classified into two; namely, Lossless and Lossy compression algorithms[17]. In Lossless compression technique, since no data is lost, during the compression process, the exact replica of the original file can be retrieved by decompressing back to the original file. Text compression is generally of lossless type. In this type of compression generally the compressed file is used for storing or transmitting data[18]. In lossy compression technique, the compression process ignores some less important and non-observable data and the exact replica of the original file can’t be retrieved from the compressed file. To decompress the compressed data, we can get a closer approximation of the original file. Lossy Compression is generally used for image, audio, video[14]. The classification hierarchy of data compression is represented in Fig. 2.1.

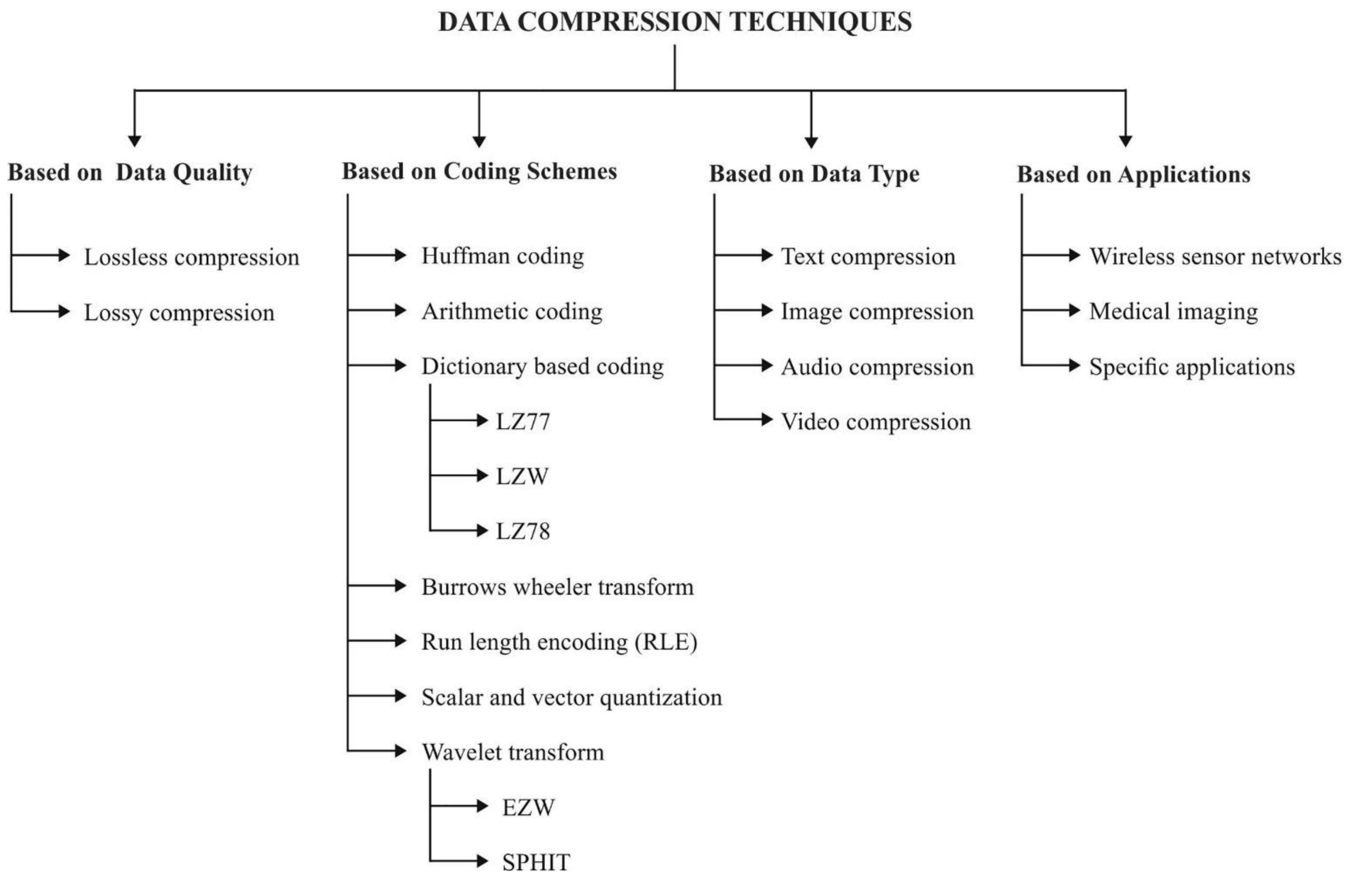


Figure 2-1 Method for Data compression [17]

2.2.1 Lossless Compression

In lossless data compression, the integrity of the data is preserved. The original data and the decompressed data after lossless compression are the same because, for the methods under this subcategory, the compression and decompression algorithms are exact inverses of each other: no part of the data is lost in the process[1].

The Lossless compression methods mean receiving the data without loss. The initial data might be retrieved exactly from the data compressed. It is used in various fields that can't ensure any variation between the original data. The main lossless compression techniques includes Run length encoding, Huffman encoding, LZW coding and Shannon Fanon algorithm[19].

2.2.1.1 The Run-Length Compression Technique

Run-length encoding is probably the simplest method of compression. It can be used to compress data made of any combination of symbols. It does not need to know the frequency of occurrence of symbols and can be very efficient if data is represented as 0s and 1s [15].

The general idea behind this method is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences. The method can be even more efficient if the data uses only two symbols in its bit pattern and one symbol is more frequent than the other[15]. The algorithm works as follows [14]:

a) Pick the first character from source string.
b) Append the picked character to the destination string.
c) Count the number of subsequent occurrences of the picked character and append the count to destination string.
d) Pick the next character and repeat steps b) to d) if end of string is NOT reached.

According to Grain and Chakraborty [10] the general run-length-algorithm is presented as follows in algorithm 2.1

```
count = 0
   REPEAT
         get next symbol
          if the symbol equals the previous one
         count = count + 1
       UNTIL (symbol different to next one)
        IF count > 1
       GOTO step one
```

Algorithm 2-1 run-length-algorithm

2.2.1.2 Shannon – Fano algorithm

Shannon – Fano algorithm was named by its developer Shannon and Fano [20]. It is used to encode messages depending upon their probabilities. It allots a smaller number of bits for highly probable messages and more number of bits for rarely occurring messages. The algorithm of Shannon – Fano is presented as follows:

For a given list of symbols and their frequency or probability table.

1. Sort the table according to the frequency of symbols, with the most frequently occurring symbol at the top.
2. Divide the table into two halves with the total frequency count of the upper half being as close to the total frequency count of the bottom half as possible.
3. Assign the upper half of the list a binary digit ‘0’ and the lower half a ‘1’.
4. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding leaf on the tree.

Generally, Shannon-Fano coding does not guarantee that an optimal code is generated. Shannon – Fano algorithm is more efficient when the probabilities are closer to inverses of powers of 2[14]. Details of Shannon – Fano algorithm is presented in blow algorithm. 2.2.

```
Procedure Shannon-Fano ()
    count source units
    sort source units to non-decreasing order
    SF-Split (S)
    output (count of symbols, encoded tree, symbols)
    write output
  end procedure
 procedure SF-Split(S)
   if (|S|>1) then
     divide S to S1 and S2 with about same count of units
     add 1 to codes in S1
     add 0 to codes in S2
     SF-Split(S1)
     SF-Split (S2)
    End if
 End procedure
```

Algorithm 2-2 Shannon – Fano encoding algorithm

2.2.1.3 The Huffman Compression Technique

The Huffman coding algorithm [21], is named after its inventor, David Huffman, who developed this algorithm as a student in a class on information theory at MIT in 1950. It is a more successful method used for text compression. The Huffman coding technique assigns shorter codes to symbols that occur more frequently and longer codes to those that occur less frequently. Before we can assign bit patterns to each symbol, we assign each symbol a weight based on its frequency of use in the document.

The Huffman algorithm is simple and can be described in terms of creating a Huffman code tree. The procedure for building this tree is presented as follows[21]:

1. Start with a list of nodes, where each node corresponds to a symbol in the alphabet.
2. Sort the list in ascending order and select two nodes with the lowest weight from the list.
3. Create a parent node for these two nodes selected and the weight is equal to the weight of the sum of two merged child nodes.
4. Remove the two child nodes from the list and the parent node is added to the list of nodes.
5. Repeat the process starting from step-2 until only a single tree remains.
6. After building the Huffman tree, the algorithm creates a prefix code for each symbol from the alphabet simply by traversing the binary tree from the root to the node, which corresponds to the symbol. It assigns 0 for a left branch and 1 for a right branch

The algorithm used in Huffman Compression is summarized as follows in algorithm 2.3

```
procedure Huffman ()
  Split the source document into distinct symbols
  count frequencies of single symbol (source units)
  sort symbols into non-decreasing sequence
  create a leaf node (symbol, frequency c, left = NULL, right = NULL) of the tree for each symbol
  while (|F|>=2) do
    pop the first two nodes (u1, u2) with the lowest frequencies from sorted queue
    create a node evaluated with sum of the chosen units, successors are chosen units
    insert new node into queue
  end while
  node evaluate with way from root to leaf node (left son 1, right son 0)
  create output from coded input symbols
end procedure
```

Algorithm 2-3 step by step procedure followed by Huffman encoding [21]

2.2.1.4 Lempel-Ziv Compression Technique

Ziv and Lempel have presented their dictionary-based scheme in 1977 for lossless data compression [22]. Today this technique is much remembered by the name of the authors and the year of implementation of the same.

The Lempel-Ziv (LZ) compression method is among the most popular algorithms for lossless compression, the code that the LZ algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight-bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds[23]. LZ encoding is an example of a category of algorithms called dictionary-based encoding. The idea is to create a dictionary (a table) of strings used during the communication session. If both the sender and the receiver have a copy of the dictionary, then previously encountered strings can be substituted by their index in the dictionary to reduce the amount of information transmitted. There are two concurrent events: building an indexed dictionary and compressing a string of symbols. The algorithm extracts the smallest substring that cannot be found in the dictionary from the remaining uncompressed string. It then stores a copy of this substring in the dictionary as a new entry and assigns it an index value. Compression occurs when the substring, except for the last character, is replaced with the index found in the dictionary. The process then inserts the index and the last character of the substring into the compressed string [22]. Here under in algorithm 2.4 the steps followed by Lempel-Ziv compression is shown

```
procedure LZ
 fill view from input
     while (view not empty) do
        find longest prefix p of view starting in coded part
        i := position of p in window
        j := length of p
        X := first char after p in view
        output(i,j,X)
       add j+1 chars
     end while
end procedure
```

Algorithm 2-4 The steps followed in LZ compression [22]

The blow Table 2.1 shows comparison of various compression coding schemes.

Table 2-1: Comparison of various coding schemes.

Coding	Feature	Compression Type	Versions	Advantages	Applications
Huffman coding[21]	Entropy based	Lossless	Minimum variance Huffman code. Length Limited Huffman code. Adaptive non-binary. Golomb-rice coding, Tunstall code	Effective in all file formats	ZIP, ARG, JPEG, MPEG, PICZ1P
Arithmetic coding[20]	Entropy based	Lossy and Lossless	Adaptive arithmetic coding Binary arithmetic coding	Flexibility	JPEG, multimedia applications
Fractal[17]	Block based coding	Lossy compression	-	Suitable for textures and natural images	Live video broadcasting
RLE[15]	Employs in high redundant data	Lossless coding	-	Faster	TIFF, BMP, PDF and fax
Scalar and Vector Quantization[14]	Represents larger set of values to a smaller set	Lossless and Lossy	-	Less complexity	
BWT compression[10]	Block sorting	Lossless	-	No need to store additional data for compression	Bzip2
LZ coding[22]	Dictionary based	Lossless coding	LZ77, LZ78, LZW	Compress all kinds of data	TIFF, G1F, PDF, Gzip, ZIP, V.42, Deflate and PNG

2.2.2 Lossy compression technique

In information technology, lossy compression or irreversible compression is the class of data encoding methods that uses inexact approximations and partial data discarding to represent the content. These techniques are used to reduce data size for storing, handling, and transmitting content. The amount of data reduction possible using lossy compression is much higher than through lossless techniques[24].

Well-designed lossy compression technology often reduces file sizes significantly before degradation is noticed by the end-user. Even when noticeable by the user, further data reduction may be desirable for real-time communication, to reduce transmission times, or to reduce storage needs[24]. Nowadays there are different lossy compression techniques; the most widely used one for image compression are JPEG compression.

2.2.2.1 JPEG Compression Techniques

JPEG is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality[24].

JPEG uses a lossy form of compression based on the discrete cosine transform (DCT). This mathematical operation converts each frame/field of the video source from the spatial (2D) domain into the frequency domain. A perceptual model based loosely on the human psych visual system discards high-frequency information, i.e. sharp transitions in intensity, and color hue. In the transform domain, the process of reducing information is called quantization[24]. In simpler terms, quantization is a method for optimally reducing a large number scale (with different occurrences of each number) into a smaller one, and the transform-domain is a convenient representation of the image because the high-frequency coefficients, which contribute less to the overall picture than other coefficients, are characteristically small-values with high compressibility. The quantized coefficients are then sequenced and losslessly packed into the output bit stream. Nearly all software implementations of JPEG permit user control over the compression ratio (as well as other optional parameters), allowing the user to trade off picture-quality for smaller file size. In embedded applications (such as miniDV, which uses a similar DCT-compression scheme), the parameters are pre-selected and fixed for the application.

Discrete Cosine Transform

The ideas behind JPEG compression come from an engineering background. Electrical and sound waves can be represented as a series of amplitudes over time. Discrete cosine transforms (DCT) is one of the basic building blocks for JPEG. The discrete cosine transform was first applied to image compression in Ahmed, Natarajan and Rao’s pioneering work [25]. Another important aspect of the DCT is the ability to quantize the DCT coefficients using usually weighted quantization values.

In image transformation, a continuous tone image can be represented by a series of amplitudes, for each color component, over 2-dimensional space. Discrete cosine transform is used to discard higher frequency information that has little visual effect on the image. For the still image representation, the frequencies here are referring to spatial frequencies rather than time frequencies.

2.3 Amharic Writing System

The Ethiopic Script

At about the beginning, South-Arabian Semitic migrated from Habashat on the Arabian coast across to Africa and founded a kingdom there with its capital at Axum. The immigrants called themselves Ge’ez, which means the ‘emigrants’ [26]. Besides, according to Feren [27] the name Ge’ez comes from the South Arabian people, the Agazyān, who crossed the Red Sea and settled in North Ethiopia and along the Red Sea coast.

The Ge'ez or Ethiopic script possibly developed from the Sabaean/Minean script. The earliest known inscriptions in the Ge'ez script date to the 5th century BC. At first the script represented only consonants. Vowel indication started to appear in 4th century AD during the reign of king Ezana, though might have developed at the earlier date[28].

As described by [29]Notable Features of Ge'ez writing system are the following

- Type of writing system: abugida
- Writing direction left to right in horizontal lines.
- Each symbol represents a syllable consisting of a consonant plus a vowel. The basic signs are modified in a number of different ways to indicate the various vowels.
- There is no standard way of transliterating the Ge'ez script into the Latin alphabet.

Ge'ez, the classical language of Ethiopia is still used as a liturgical language by Ethiopian Christians and the Beta Israel Jewish community of Ethiopia[28].

Amharic is the national working language of Ethiopia, has about 27 million speakers. It is spoken mainly in North Central Ethiopia. There are also Amharic speakers in a number of other countries, particularly in Egypt, Israel and Sweden[28].

2.3.1 The Amharic Writing

Amharic is written using a writing system called fidel - ፊደል ("alphabet", "letter", or "character") adapted from Ge'ez (the liturgical language of the Ethiopian Orthodox Church) language. In Amharic writing system, there are Amharic characters, numeric and punctuation marks. Amharic characters were represented by computer using Unicode. Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language [30]. Ethiopic characters (fidel - ፊደል) have more than 380 Unicode representations including punctuations and special characters (U+1200- U+137F)[29].

2.3.1.1 Amharic Characters

There are 33 basic characters, each of which has seven forms called orders depending on which vowel is to be pronounced in the syllable. The seven orders were representing seven vowel sounds. Therefore, these 33 basic characters with their seven forms will give 7*33 syllable patterns (syllographs), or fidels.

Amharic writing system has of thirty-three core characters. The thirty-three characters occur in one basic form and six other forms known as orders, as shown in Table 2.2. These orders are derived from the basic forms by more or less regular modification [29]. The seven orders of the Ethiopic represent the different sounds of a consonant- vowel combination known as syllabic.

Table 2-2: Amharic Characters

Geéz	Kaéb	Salis	Rabé	Hamis	Sadis	Sabé
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
ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ
ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ

2.3.1.2 Amharic Numeration System

As shown in Table 2.3 below Amharic numeration system consists of basic single symbols for one to ten, for multiple of ten (twenty to ninety), hundred and thousand. These numerals are derived from the Greek numerals with some modifications. Each symbol has a horizontal stroke below and above. There is no symbol for representing zero value in Amharic number system, and it is not a place value system, thus arithmetic computation using this system is very difficult. As a result, in most printed document Hindu- Arabic numerals are used.

Table 2-3: Amharic Numeric System

Ethiopic	Arabic	Ethiopic	Arabic	Ethiopic	Arabic
1	፩	11	፲፩	30	፴
2	፪	12	፲፪	40	፵
3	፫	13	፲፫	50	፶
4	፬	14	፲፬	60	፷
5	፭	15	፲፭	70	፸
6	፮	16	፲፮	80	፹
7	፯	17	፲፯	90	፺
8	፰	18	፲፰	100	፻
9	፱	19	፲፱	1,000	፻፻
10	፲	20	፳		

2.3.1.3 Amharic Punctuation Marks

There are about 17 punctuation marks in Amharic writing system [12]. These punctuation marks are shown in Table 2.4 blow:

Table 2-4: Amharic Punctuation Marks

Amharic Punctuation Marks	Symbol	Description
ሁለት ነጥብ	፡	word delimiters
አራት ነጥብ	።	sentence delimiters (the equivalent of the full stop)
ነጠላ ሠረዝ	፣	the equivalent for comma
ድርብ ሠረዝ	፤	the equivalent of semi-colon

Word delimiter (ሁለት ነጥብ) is most commonly used in historic documents. However, nowadays, in computer writing style it is common to use a space as a word separator instead of using "፡" (ሁለት ነጥብ) [12]. There are also some borrowed symbols; ‗? ‗ (question mark), ‗! '(Exclamation mark), arithmetic operators such as ‗+ ', '-‗, '*', '/ ', brackets (‗ (‗, ') '), quotation marks (―, ‖), etc).

2.3.2 Amharic Printed Documents

There are a number of Amharic computer fonts available these days for Amharic writing system. 'Power Geez', 'Visual Geez', and 'Nyala' are some of the commonly used fonts in computer printed Amharic documents. An Amharic word written in different fonts is shown in Table 2.5 below.

Table 2-5: Different Amharic Font Types

Amharic Word	Font Type
›=fÄåÁ	Geez-1
›=fÄåÁ	Geez -2
›=fÄåÁ	Geez-3
x!T×ùÃ	Visual Geez 2000 Main Font

From the above example, we can see that words belonging to the same class but printed using different fonts greatly vary both in shape, width, height and line thickness

2.4 Overview of PDF file format

The PDF file format was first introduced in 1990 as a way to reliably view, print, and share information with other people[30]. PDF was a simplification and evolution of the ageing Adobe PostScript page description language. Whereas Postscript is actually a programming language primarily used to drive laser printers, PDF has been designed specifically for exchanging documents to be rendered on screens. Thus, PDF is an elaborated descriptive format allowing each page to be rendered independently. On the contrary, PostScript needs a dedicated interpreter executing a program and containing a global state for the entire document, i.e., rendering a single page in PostScript imposes the rendering of all the previous pages in order to obtain the current page's state[31].

PDF is an elaborated page description language allowing complex documents containing text, graphics, and images to be unfailingly reproduced, i.e., they will look exactly the same even when rendered on different systems and physical devices (screens, printers)[30].

2.4.1 PDF File Structure

The internal structure of a PDF file is decomposed into four primary sections as illustrated in Figure 2.5, such us header, body, cross reference table and trailer,

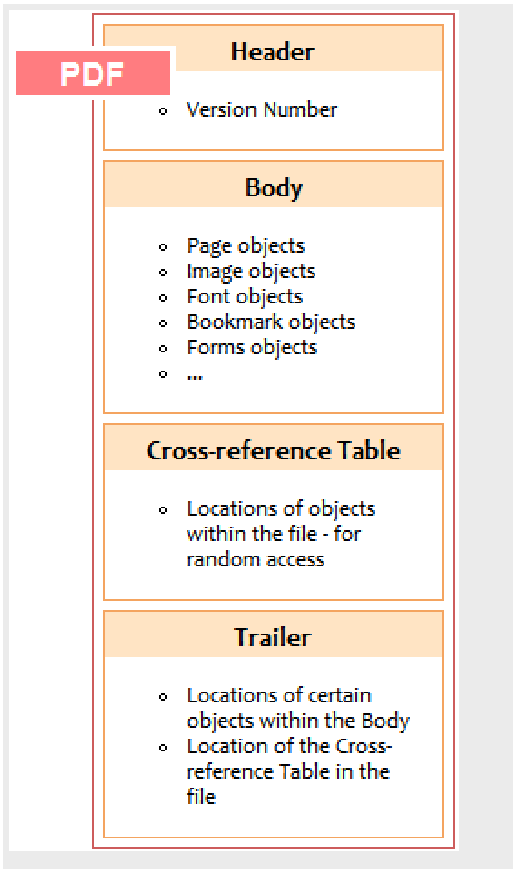

PDF
Header
Version Number
Body
Page objects
Image objects
Font objects
Bookmark objects
Forms objects
...
Cross-reference Table
Locations of objects within the file - for random access
Trailer
Locations of certain objects within the Body
Location of the Cross-reference Table in the file

Figure 2-2 file structure of PDF files[9]

The Header defines the version of PDF specification for example, "%PDF-1.4". The body contain the actual text and image content that will be displayed on the output. Cross-reference table further shows a table for PDF viewers to quickly access different objects within the body. Finally, Trailer, defines other Meta information of a PDF file that describes the location of objects within the body and location of cross reference table.

Sample PDF file with its internal content organization as per the structure is depicted in the following figure 2.6

```
%PDF-1.7
1 0 obj
  << /Type /Catalog
     /Pages 2 0 R
  >>
endobj
2 0 obj
  << /Type /Pages
     /Kids [3 0 R]
     /Count 1
  >>
endobj
3 0 obj
  << /Type /Page
     /Parent 2 0 R
     /MediaBox [0 0 600 400]
     /Resources << >>
  >>
endobj
xref
0 4
00000000000 65535 f
00000000010 00000 n
trailer
  << /Root 1 0 R
     /Size 4
  >>
startxref
249
%%EOF
```

Figure 2-3 internal content of PDF file

Header: The first line of the PDF specifies the version of a PDF file format. These headers are the topmost portion of a document. It reveals the basic information of a PDF file, for example, "%PDF-1.4", it means that this PDF format is the fourth version[30].

Body: The contents below the header and above the line xref are the body. The body of a PDF file consists of objects that compose the contents of the document. These objects include image data, fonts, annotations, text streams and so on. Users can also integrate invisible objects or elements. These objects embed the interactive features in a document like animation or graphics. A user can also implement logical structure in the document and also users make the content of a PDF document more secure by implementing security features. One can protect the content of a document from unauthorized printing, viewing, editing or modifying [30].

Cross-Reference Table: The Cross-Reference Table (or called xref table): The cross-reference table consists of links to all the objects or elements in a file. It has a future that helps to navigate to other pages or content in a document. When users update their PDF files, they will automatically get updated in the cross-reference table. One can also trace the updated changes in the cross-reference table. This part is the hardest part to understand. If incorrectly set, the PDF viewer will give out errors. The cross-reference table are used for quick accessing every object appear in the body. So, we need to give every object a cross-reference entry[30].

Trailer: The trailer contains links to cross-reference table and always ends up with "%%EOF" to identify the end of a PDF file. The "%%EOF" is necessary for a PDF file, if this line missed, the PDF-file is not complete and may not be processed correctly. This is not same as PostScript files. If the last few lines of a PostScript file missed, we will still print most of the pages. The trailer enables a user to navigate to the next page by clicking on the link provided. Trailer section gives us the overall information of the PDF documents, it must contains a dictionary, which should have at least two entries: /Root and /Size[30].

/Root refers to the Catalog of the body (Next section). /Size refers to the total number of entries in the file's cross-reference table. Start xref follows by a line of a number, indicates the start offset of the cross-reference table. i.e. the offset of the keyword xref. %%EOF indicates the end of the file[30].

2.4.2 PDF Document Structure

The document structure of a PDF file consists in a series of objects hierarchically organized within the body. These objects, most of them of dictionary type, are sorted out in page objects, which are organized in a page tree, specified in the document catalogue, as shown in Figure 2.4. In order to reach the document catalogue dictionary object, as specified before, you can check the /Root value in the trailer of the file. This catalogue contains references to other objects which provide information about the document contents, as well as information about how these contents should be displayed[30].

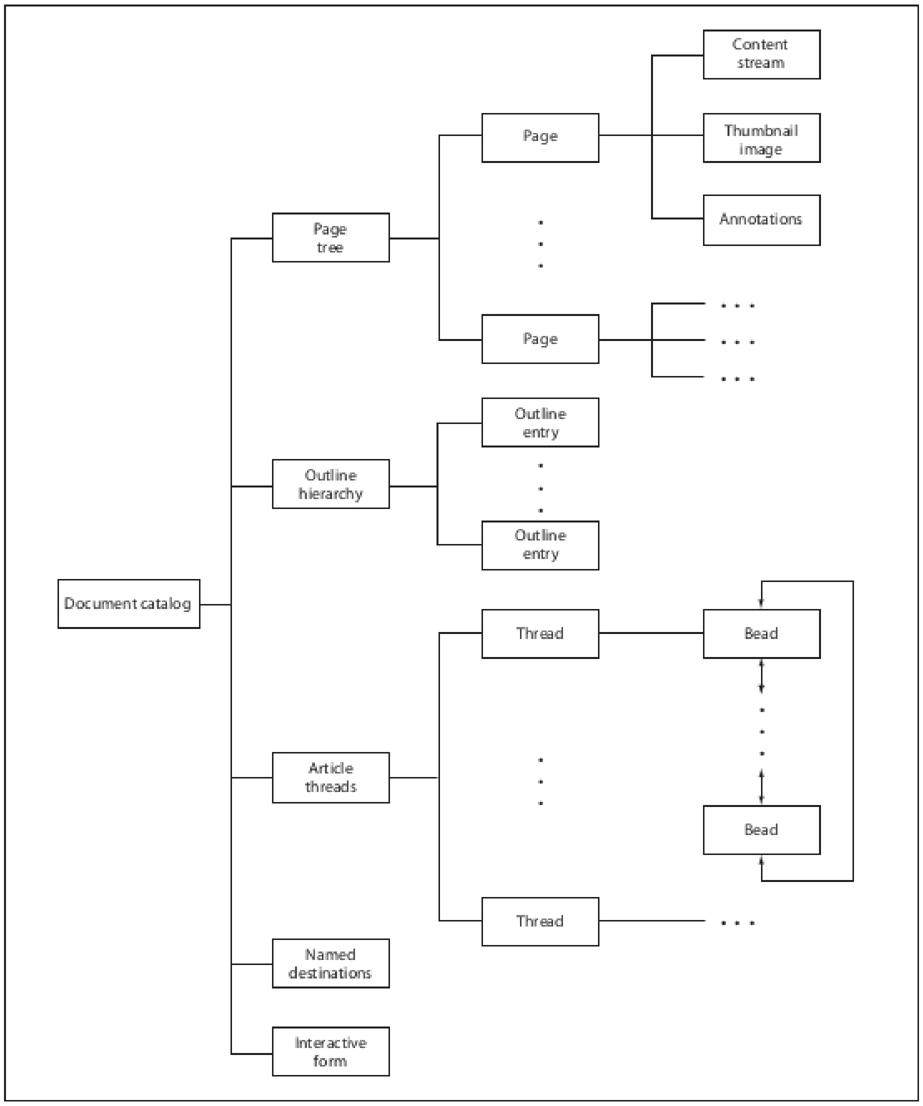


Figure 2-4 Object structure of PDF file [9]

2.4.3 Common file representation in PDF file

File presentation of PDF depends on whether the content is text, images, graphics or other objects in the document.

Content Streams: The graphical content (i.e., text, images, and graphics) of PDF document pages is expressed exclusively inside dedicated PDF streams called content streams. A content stream consists of a sequence of instructions which are used to describe the graphical elements that are to be painted onto the page[32].

Text Representation: In PDF, the textual content is handled by text instructions defining graphical text primitives embedded in page content streams. These text primitives are determinate by various attributes, including a font id (that maps to a font object held in the page resource dictionary), a font size, some character codes and their relative coordinates on the page.

Most of the time, text is represented as vector graphics and is rendered in the form of a set of character shapes mapping character codes. Since PDF considers glyphs, i.e., character shapes, as graphical objects, many of the text operators are handled using the graphics state and standard painting operators[32].

Graphics Representation: A graphic, i.e., a vectoral image, is built from a sequence of paths contained in a page content stream. Each path inherits the attributes of the current graphics state and is itself described as a set of consecutive line segments and curves. Paths can be used for painting strokes, filling areas, describing font glyphs, and even clipping zones[32].

Images Representation: Raster images, i.e., digital images, are most of the time embedded in the page resources as stream objects. They are then simply referenced from the content streams by their resource names and positioned (and resized) according to the CTM. Images can also be referenced as external resources, but PDF recommendations discourage such practice since this would link a PDF document with many external files[32].

Fonts Representation: PDF is able to deal with many different font formats including the so-called simple and composite fonts. Simple fonts are limited to 256 glyphs, while composite fonts are unlimited in their glyph space. Since file size is a key issue in PDF, fonts are frequently reduced to subsets, describing only used glyphs. Moreover, fonts are most of the time represented as compact descriptions, post-filtered by a compressing algorithm, again to reduce the overall file size. This leads to quite complicated solutions for representing font information[32].

Metadata and Content Structures: PDF provides various mechanisms to incorporate metadata describing its content, such as annotations and structural information. For instance, a PDF annotation associates a file or data such as a note, link, hyperlink, sound, or movie with a location on the page (using the default user space). PDF annotations also provide a way to interact with the user by means of page locations linked with mouse and keyboard events[32].

2.5 Page Layout Segmentation

Segmentation is a process of separation of an image into regions that contain pixel groups that are similar in value[33]. It is also explained as symbolization or extraction of characters from pixel array. It is considered as the most important part of recognition system because of the direct dependency of correct recognition on correct segmentation of characters. It is applied after the image filtering and other preprocessing tasks are done[34].

Segmentation occurs at two levels; on the first level, blocks of text, graphics and other parts are separated. Then, text lines and words in the text image are located[34]. Text/graphic segmentation and extraction of words and characters in document images are very important. The degraded quality of documents poses different problems such as characters broken into multiple components, text at the back of document images appearing on the front, etc., thus making extraction of the words and characters very difficult[34] [35].

2.5.1 Text/Graphic Segmentation

Page segmentation into text and non-text components is an essential preprocessing step before other operation and for further processing. In document images, basic shapes of text characters are limited in number, but shapes of the non-text components including drawing, logos, graphs, etc. are unlimited. Figure 2.3 shows the composition of a given real world document that contains printed text, table, figure and noise.

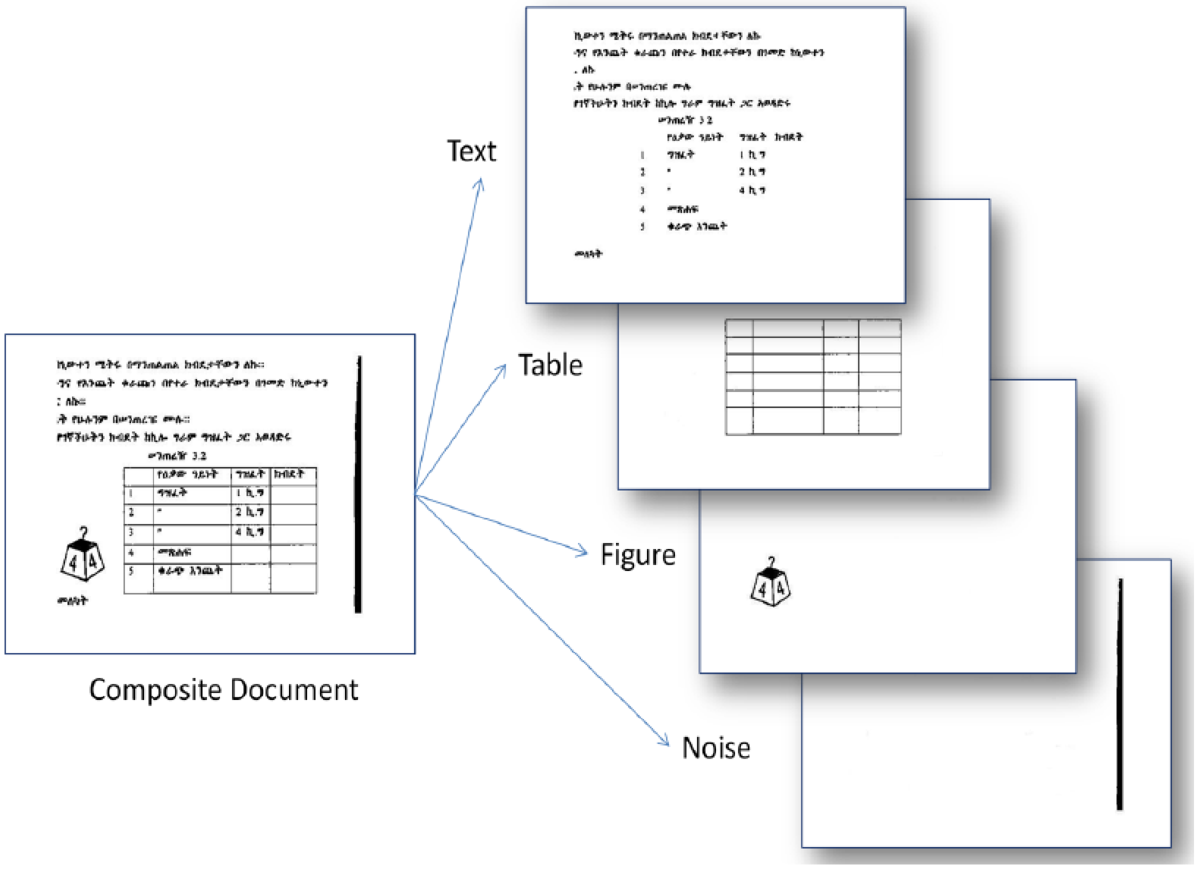


Figure 2-5 composition of a given real world document

2.5.2 Text Line and Word Segmentation

According to Million[36] after text part is separated from graphics, tables and other parts, the next step is extracting text lines and words in the image. Line segmentation is a process of scanning horizontally text part of an image document for identifying parts that hold text line and parts that are blank. After lines that contain text are identified, the next step is word identification (segmentation). Identification of word boundaries requires the task of distinguishing words from word spaces. The presence of spaces that precede or succeed a character makes it difficult to identify a word separator from a character separator. Hence, it complicates the job of word boundary identification as one of the challenges in word level segmentation[37].

2.6 Segmentation Techniques

Different authors categorize image segmentation techniques differently. Skarbek[38], categorize image segmentation techniques as: pixel based, edge based, area based and physics based techniques. Yatharth [36], grouped segmentation techniques into: threshold techniques, edge based methods, region based methods and connectivity-preserving relaxation-based methods. Cattoni[39], provide approaches for page segmentation as: Smearing based techniques, projection profile methods, texture based (local analysis) techniques and structure based techniques. Usually, page segmentation methods are divided into three main groups: top-down, bottom-up and hybrid approaches[34].

2.6.1 Top-Down Techniques

Top-down techniques divide document images recursively from entire image to smaller regions. Khurram [40] noted that for top down segmentation methods to be effective, they need to have a prior knowledge about the document class and type (number of columns, width of margins). Most of well-known top-down methods are XY cut, Projection Profile Methods, Whitespace Cover, Histogram Analysis and Space Transforms Fourier transform, Hough transform [41][42].

X-Y cut algorithm also called recursive x-y cuts (RXYC) algorithm is a top-down algorithm which partitions a document into rectangular components which represent the nodes of the tree. It follows a tree-based approach; the root of the tree represents the entire document page and all the leaf nodes together represent the final segmentation. The bits of the binary transformed image are summed by this algorithm. In this manner a density graph is obtained. The low ends of the graph stand for empty spaces that are lines in the segmented document. If the values reach a higher point than it, the segmentation process is stopped, and the layout component is identified. This algorithm is usable for both horizontal and vertical projections. The process is continued until an empty line threshold is reached. At that point the segmentation algorithm ends[43].

A positive aspect of the algorithm is that the threshold controls the size of the found component. This way, the configuration setting makes the algorithm suitable for finding paragraphs, lines or words and many other elements of the page by simply performing an adjustment to one parameter. More than that, the threshold controls the size of the segmented clusters which makes the algorithm scalable. The detection of the rows, paragraphs, section and so on is therefore possible [42]

The whitespace cover algorithm considers a collection of rectangular components as well as another component that represents the entire page and it is a container for all the other components. The main idea behind this algorithm is the maximization of the white spaces in order to obtain the optimal page. The algorithm principle is similar to quick sort[44]. The starting point of the algorithm is represented by a rectangular component that bounds the whole page and a group of white rectangles are considered obstacles. One of these obstacles is chosen as a pivot. Usually it is chosen one that is as central as possible. With this pivot the space is split into 4 components (right, left, top, down) which become candidates for being processed in the same way recursively. Every component is tested with a quality function in order to evaluate if there is a white space or not [42][34].

2.6.2 Bottom-Up Techniques

Bottom-up techniques start with the smallest elements (pixels), merge them recursively in connected components or regions, and then in larger structures [38]. It uses Area Height analysis to identify figures (big components) and extract near connected components as figure caption candidates.

Some of the methods used here are Connected Component (CC) Analysis, Region-Growing Methods, Docstrum, Voronoi-Diagram Based, Run Length Smoothing, Smearing, Neural Networks and Active Contours [45] [41][46].

Connected component (CC) analysis is a bottom-up technique that scans all the pixels of document image and recursively label them based on pixel connectivity, i.e. all pixels in the connected component share similar pixel intensity values and are in some way connected with each other. It uses CC area height analysis, which takes the width and height of the bounding box of a labeled component to calculate the area in order to get some threshold value, to identify big components and extract near connected components as figure caption candidates[47].

Voronoi-Diagram Based Algorithm: as prposed by Kise [48], Voronoi-diagram based algorithm first extracts sample points from the boundaries of the connected components using a sampling rate. Then, noise removal is done using a maximum noise zone size threshold, in addition to width, height, and aspect ratio thresholds. After that a Voronoi diagram is generated using sample points obtained from the borders of the connected components. The Voronoi edges that pass through a connected component are deleted to obtain an area Voronoi diagram. Finally, superfluous Voronoi edges are deleted to obtain boundaries of document components. An edge is declared superfluous if it satisfies any of the following criterion[48]:

- The minimum distance d between its associated connected components is less than the inter-character gap in body text regions.
- The minimum distance d between its associated connected components is less than the inter-line spacing times a margin control factor, or the area ratio of the two connected components is above an area ratio threshold.
- At least one of its terminals is neither shared by another Voronoi edge nor lies on the edge of the document image.

The output of the algorithm consists of arbitrarily shaped regions bounded by Voronoi edges; each Voronoi region is represented by its bounding box[48].

Docstrum: this is one of the bottom-up algorithms which is proposed by O‘Gorman[49]. This approach is based on nearest-neighborhood clustering of connected components extracted from the document image. O‘Gorman[49] proposed, after noise removal, the connected components are separated into two groups, one with characters of the dominant font size and another one with characters in titles and section headings, using a character size ratio factor. Then, K nearest neighbors are found for each connected component. A histogram of the distance and angle of each connected component from its K nearest neighbors is computed. The peak of the angle histogram gives the dominant skew in the document image. This skew estimate is used to compute within-line nearest neighbor pairs. Then, text-lines are found by computing the transitive closure on within-line nearest neighbor pairings using a threshold. Finally, text-lines are merged to form text blocks using a parallel distance threshold and a perpendicular distance threshold.

2.6.3 Hybrid Techniques

There are also hybrid methods that combine and make use of both bottom-up and top-down approaches [32], for example, connected component analysis for shape information and block separation for background block map [36]. Hybrid methods work very well for major text/graphic segmentation in both historical and contemporary pages (magazines, newspapers, journals, etc.), but not for a very fine level segmentation of words and their individual characters in historical books [32].

Watershed Based Image Segmentation: Watershed transformation is a powerful mathematical morphological tool for the image segmentation [36]. The technique is more popular in the fields like biomedical and medical image processing, and computer vision. In geography, watershed means the edge that divides areas drained by different river systems. If image is viewed as geological landscape, the watershed lines determine boundaries which divide image regions. The watershed transform computes regions and edge lines (also known as watershed lines). Watershed Segmentation adopts the concepts from the techniques such as threshold based, edge based and region based segmentation [36]. There are mainly two classes of watershed algorithms: the flooding based watershed algorithms and rain falling based watershed algorithms [28].

Flooding Based Watershed Algorithms: In traditional flooding based approach of watershed based image segmentation, image is considered as a topographic surface which contains three different types of points: points which indicate regional minimum, points where the water falling has highest probability to fall into a single minimum region and points where the water falling has probability to fall into more than one such a minimum region. For regional minimum, the groups of points satisfy second condition called watershed or catchment basin of that minimum and the groups of point satisfy third condition makes a crest line on topographic surface termed as a watershed line. Figure 2.5 shows an example of the watershed line and catchment basin [28].

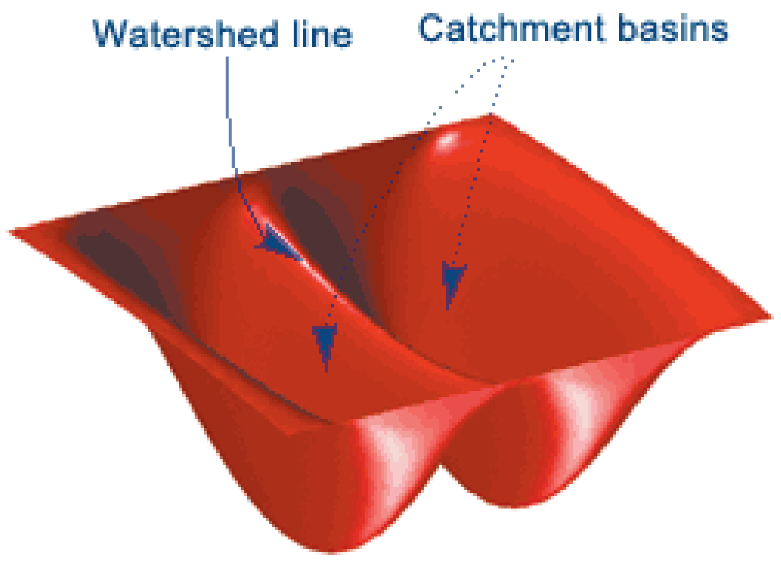


Figure 2-6 Watershed Lines and Catchment Basins [28]

The basic concept of watershed algorithm used for the image segmentation is to find the watershed lines (boundaries). Imagine, holes at each regional minimum, and water is flooded into these holes with constant rate. The level of the water will rise in the topographic surface uniformly. When the rising water in different catchment basins is going to merge each other, then a dam is built to prevent merging of the water. Finally, flooding of water will reach at the point where only top of the dams is visible. These continuous dam boundaries are called the watershed lines [36].

Rainfalling Watershed Algorithms: Unlike traditional flooding-based algorithm, the rainfalling algorithm extract mountain boundaries. The concept of the algorithm is that rainwater drops fall on the mountain (topographic surface) and move to descending direction because of the gravity until they reach to the local minimum surface. The algorithm tracks the path of water drop for each point on the surface towards the local minimum, if raindrops pass through that point or fall on that point. A group of points make a segment when water drops related to them flow downwards to the same deepest location. When a point has more than one path towards the different steepest surfaces then it can be allocated to any one of the local minimum [28].

2.7 Related Works

Several authors have tried to study language dependent data compression system especially for Arabic, Hebrew, Chinese and Indian languages that are using non-Latin characters. These research attempt to combine different compression algorithms. Some of them are listed as follows:

Mtimet and Amiri [9] introduce new Arabic Textual Images Compression. They propose a compression method for large Arabic text in document images. They used three steps to accomplish the desired purpose. In the first step, which involves segmentation module, the input image is partitioned into individual symbols. In the second step, a pattern matching technique used to determine similar patterns. Finally, a neighborhood coding is performed to encode each symbol. In their proposed work, they developed an algorithm for large Arabic textual image compression. The main idea of this approach is Neighborhood coding with reduction. Based on the extracted patterns, that can improve the pattern-matching procedure, and this lead to further compression by Neighborhood coding. Experimental results show the proposed approach can improve compression ratio on large Arabic textual images. The compression results satisfy the lossless context, namely no error happens during compression. There future work will concentrate on integrating on their approach in compound image compression framework.

On another study of Grailu and Yazdi [50] they considered Farsi/Arabic script. In the Farsi/Arabic script contrary to the printed Latin script, letters routinely affix to each other and produce diverse samples. Along these lines a couple of illustrations are totally/partially subsets of a couple of others. Recognizing such circumstances and mishandling them to reduce the amount of library models has a great effect on the weight profitability, in light of the way that the amount of occasion of such circumstances in Farsi and Arabic substance pictures is high. Such joining characters exist in some diverse scripts and printed style faces. Likewise, undesired touching characters exist in some substance pictures, for instance, Latin ones. The proposed weight methodology is not confined to simply Farsi and Arabic substance pictures.

In their proposed system three procedures were utilized the number of library models was decreased by recognizing and precluding the completely/mostly comparative models or those models which have a straightforward connection to one another.

A new viable example encoding plan was proposed for encoding a wide range of examples. The proposed encoding plan has two operation methods of chain coding and delicate example coordinating each of which is utilized for various types of examples. The previous is utilized for minimal examples and the recent is utilized for scanty examples. Exchanging between these modes is finished by a measure of info example which is equivalent to the proportion of the example territory to the powerful length of its chain code grouping. In chain coding mode, it utilizes a blend of chain coding, run length and 1D versatile connection-based math coding procedures and in the delicate example coordinating mode, it utilizes the delicate example coordinating method. They have altered and utilized the multi-image QM-coder for encoding the arrangements of numbers.

Three distinctive levels are proposed for lossy compression each of which further expands the compression proportion. All or some of these levels may be utilized as a part of request to accomplish distinctive compression proportions. The main level incorporates applying some chain code area methods, for example, exclusion of little examples and openings, oversight of inward gaps of letters, and smoothing the limits of the examples. The second level incorporates utilizing the specific pixel inversion system, and the third level incorporates utilizing the proposed technique for organizing the lingering examples for encoding. These three levels cause the proposed technique to be more adaptable than the current compression strategies or benchmarks in light of the fact that diverse compression proportions are achievable. Exploratory results demonstrated that the proposed system works better, as high as from 1.4–3.3 times in lossy case and 1.2–2.7 times in lossless case at 300 dpi, than the best existing compression routines or measures. The most extreme compression proportions were accomplished for Farsi/Arabic scripts[9].

Garain and Debnath[10] present a new compression scheme for Indian language textual documents images. As OCR may not be able to compress documents due to unavailability of data set in most of the Indian language scripts and even if the data set is available, it is not efficient enough to perform the conversion job. In this paper new compression technique presented first time for Indian languages. The proposed compression technique is lossy in nature, compresses document images up to readable level. This method is based on the symbolic compression technique. The proposed technique has been accomplished with an efficient segmentation-based clustering approach.

Wiseman and Gefner[51] suggested a suitable technique for Hebrew languages. This technique includes two phases: at the first phase, morphological analysis is used to segment the text into two files. The first file includes index values for each pattern. The second file includes the root of the words. The second phase included compressing both files, using traditional BWT. The Hebrew bible file had been used for testing the compression. The size of file was 260 KB. Results showed that using the technique which had been suggested achieved 28.97% of compression rate whereas the traditional BWT compressed this file to 40.13%.

CDIS (Compression for Document Image System) is an example based about lossless compression framework for checked record pictures. Zhang and Danskin [52], present a progressive lossy example occurrence position coding method which brings about a noteworthy change in compression with no obvious artifacts. CDIS codes content positions via consequently designing pieces of content, then transmitting the position blunders for every example. The lossy coding so is accomplished with diminished accuracy, subject to quality guarantees. CDIS exploits the learning of record pictures one stage further, CDIS exploits the auxiliary format of report pictures. This framework adequately codes example groupings and positions in the picture by coding the example positions progressively: -

- First it isolates the content picture into pieces and consequently designs the content inside of every square by assessing every design's position.
- Then it transmits the position lapse (the contrast between the design's genuine and assessed positions)

In almost lossless mode, CDIS packs the same test record set (single page) around 13% more than the best past results. Better results (18% sparing) are acquired on multi-page reports. CDIS imitates report pictures in about lossless mode. The remade picture is a rough guess to the first picture. The nature of the remade pictures is ensured by traditionalist example coordinating and position coding routines. CDIS's example coordinating strategy guarantees that just firmly coordinated examples are substituted from the first pictures. Its example position coding strategy guarantees that the position of every example is coded inside of a couple of pixels of its unique position. Besides, CDIS gives the alternative to change and control the level of exactness used to code example positions [52].

Chao and Fan[32] has developed techniques that identifies the logical components of a PDF document Page into different logical structure regions such as text blocks, images, vector graphics blocks and compound blocks for further PDF file manipulation. The PDF document page is separated into three layers of text, image and vector graphic. Each layer become an individual PDF document. Hence the logical structural component is placed in each stratum. For text component's font, size, line spacing and color are identified. For image component the shapes are named. For vector graphics component the object is identified. After the route object is converted Scalable Vector Graphics (SVG) is an XML based vector image format for 2D graphics. all separated component are combined and converted to compound component.

Most of the previous studies are done for English Arabic [2], Chinese[6], Hindus [7] and other non-English languages to address the problem of memory management and bandwidth utilization. But when we come to our country, with their own scripts there is no research effort to compress PDF documents written in Amharic language. However, with the increasing published PDF documents in Amharic to disseminate news and reports using newspapers, magazines and webpages, there is a need to decrease the size. So as to reduce memory space requirement and speed up transmission with the limited bandwidth. Hence this study is initiated to design an approach for compressing Amharic PDF formats.

CHAPTER THREE

METHODS AND ALGORITHMS

The task of page segmentation is to divide the document image into homogeneous zones, each consisting of only one physical layout structure (text, tables, pictures)[33]. Page segmentation techniques also help segment text part of the image to lines, words and characters. Therefore, the performance of compression systems highly depends on the page segmentation algorithm used.

Among a number of physical and logical page layouts this study focuses on PDF Image segmentation, extraction and compression, also positional information about those segmented blocks of PDF documents in order to reconstruct after the decompression. Therefore, this chapter in particular explores architecture of the proposed approach, techniques and evaluation.

3.1 Design of the System

Figure 3.1 shows the architecture for the proposed approach. The proposed approach uses PDFMiner python module to segment the logical and physical structure of the file and extract the result then appended to XML or HTML file. Finally encoding the extracted output is done with different compression algorithms such as Huffman and LZ compression.

On the other hand, the given PDF files is password protected or scanned PDFMiner can’t access the PDF object directly. In this case the system first obtains the PDF file and converts each page into a series of Image files. PDF elements segmentation stage detects the PDF document image files, categorizes them into homogeneous block and stores all the information of the segmented blocks. Segmented elements classified into text and non-text blocks. Extracted text regions faded to tesseract OCR to recognize and extract the text file. Finally encoding the extracted output is done with different compression algorithms.

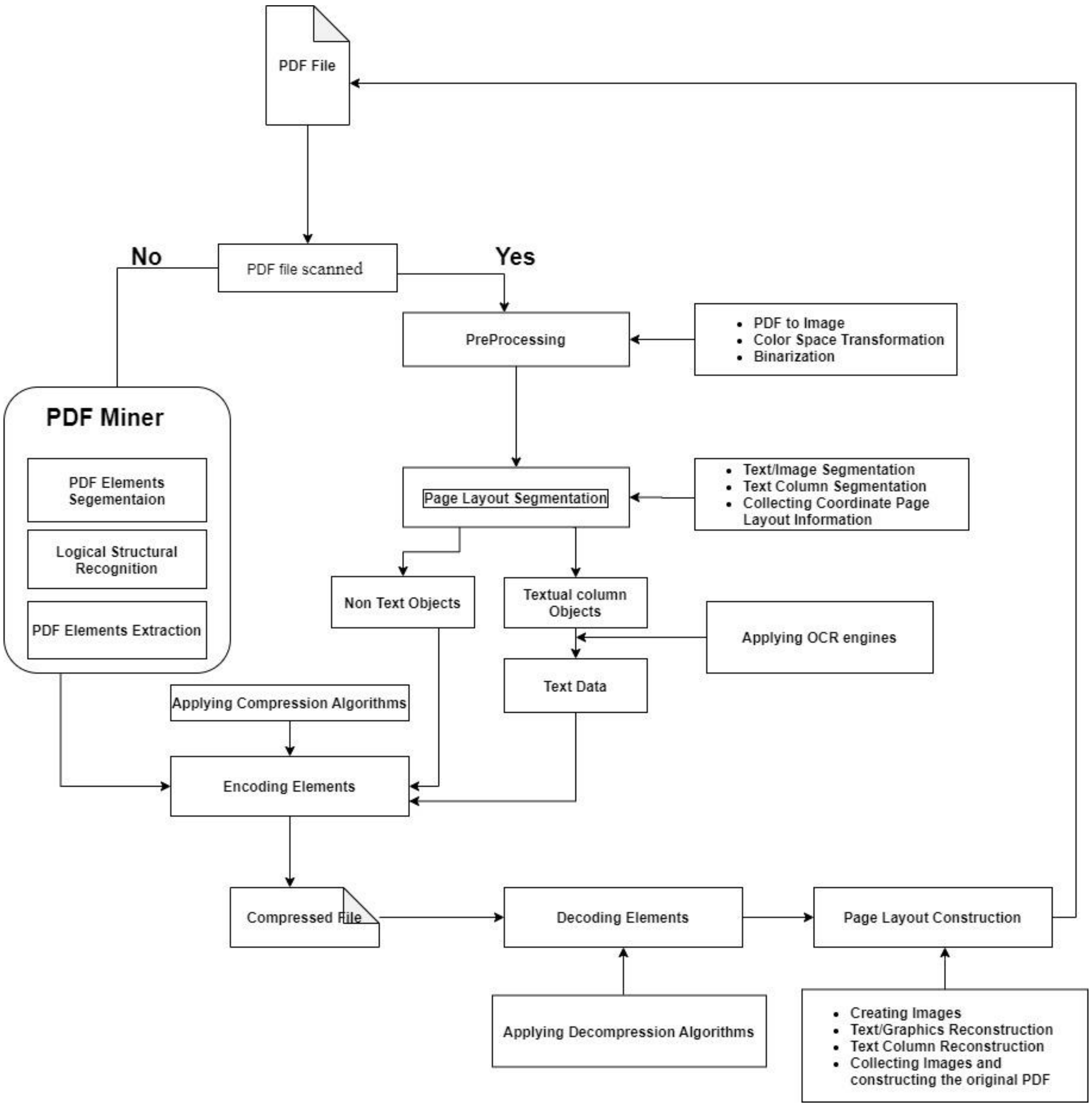


Figure 3-1 Architecture of the proposed approach

The proposed approach uses page segmentation step not only for segmenting Images and column and paragraph blocks of the original document image. It also used to extract positional information about those segmented blocks of PDF documents in order to reconstruct after the decompression

3.1.1 Preprocessing

This is the most important and crucial step that is very helpful to enhance the performance of next segmentation and extraction steps. Major preprocessing tasks while working with scanned PDF documents are PDF to image conversion, color space Transformation and binarization.

3.1.1.1 PDF to image conversion

To access and manipulate objects inside the PDF file we need to convert the PDF file to Image. This study uses PDF to Image python library to convert the given Amharic PDF file to its equivalent image format.

3.1.1.2 Color space Transformation

A color space is a method by which we can specify, create and visualize color. As humans, we may define a color by its attributes of brightness, hue and colorfulness. Different color spaces are better for different applications[38]. A computer may describe a color using the amounts of red, green and blue phosphor emission required to match a color. A printing press may produce a specific color in terms of the reflectance and absorbance of cyan, magenta, yellow and black inks on the printing paper[53].

The converting file format in PDF to image is CMYK color mode. CMYK is a subtractive based color space and is mainly used in printing and hard copy output. The fourth, black, component is included to improve both the density range and the available color range (by removing the need for the CMY inks to produce a good neutral black it is possible to used inks that have better color reproductive capabilities). This mode is not supported by MATLAB programming language. Hence, we need to transform the color space from CMYK to the primary color mode, RGB.

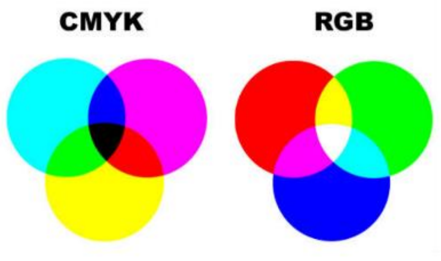


Figure 3-2 CMYK and RGB color space[53]

The Red, Green, Blue values are given in the range of 0..255, the red color(R) is calculated from the cyan(C) and black(K) colors, the green color(G) is calculated from the magenta(M) and black(K) colors, The blue color(B) is calculated from the yellow(Y) and black(K) colors. Below is the formula of CMYK to RGB conversion[53].

- Red = 255 × (1 - Cyan ÷ 100) × (1 - Black ÷ 100)
- Green = 255 × (1 - Magenta ÷ 100) × (1 - Black ÷ 100)
- Blue = 255 × (1 - Yellow ÷ 100) × (1 - Black ÷ 100)

3.1.1.3 Binarization

The Binarization Method converts the grey scale image (0 up to 256 gray levels) in to black and white image (0 or 1). The result of page layout segmentation highly depends upon the binarization. The high quality binarized image can give more accuracy.

The common method used to binarize is known as thresholding. In thresholding, the grayscale or color images are represented as binary images by picking a threshold value. There are two categories of thresholding[54].

Global thresholding: a threshold value is selected for the entire document image which is frequently based on an estimation of the background intensity level with that of the image using an intensity histogram [55].

Local or adaptive thresholding: use different values for each pixel depending on the information at different pixel points [55]. Local thresholding is commonly used in works that involve images that are of varying level of intensities, such as pictures from satellites cameras or scanned medical images.

3.1.2 Page Layout Segmentation

Page layout segmentation is the next step to follow after the prepressing of a PDF converted to images and binaries. It is performed to separate text from non-text region and to store layout information of segmented blocks. Then, the next image processing is applied over the textual area to recognize textual data. Whereas, the information stored plays an important role in the course of maintaining the original document image page layouts. Therefore, page layout segmentation is an important stage of the proposed approach because the remaining stages including text recognition and layout preservation heavily depends on this stage. Thus, in this study, page layout segmentation techniques are applied to extract text regions from non-text regions and to store column and positional information with the aim of reconstructing original document during the decompression process.

This study explored five segmentation techniques namely: watershed transforms, run length smoothing, connected component labeling, whitespace analysis and morphological dilution. These techniques are experimented in different combinations Amharic PDF document.

3.1.2.1 Morphology Dilution approach

The morphology approach quantitatively describes the shape of objects in an image and has recently attracted much attention [28]. The mathematical morphology describes such operations by combinations of basic set operations between an image and a small object called a structuring element. It is very attractive for this purpose because it efficiently deals with geometrical features such as size, shape, contrast, or connectivity that can be considered as segmentation-oriented features. One of the advantages of using morphological approach is its low computational cost. The simplification for segmentation can be efficiently achieved by filters based on opening and closing by partial reconstruction. The size of structuring element is progressively decreased to allow the introduction of more local information to improve the segmentation [28]. In this paper, considering these advantages of morphological approach, it is used for segmenting the document PDF images.

Dilation is one of the most basic morphological operations. It is used to connect characters in words, words in a text line, and text lines in a column by adding pixels to the boundaries of objects in an image. The number of pixels added to the objects in an image depends on the size and shape of the structuring element used to process the image. In the morphological dilation operation, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The dilation rule used to process the pixels is; the value of output pixel is the maximum value of all pixels in the input pixel's neighborhood. For instance, in a binary image, if any of the neighborhood pixels values are 1, the output pixel is set to 1 and if both neighborhood values are 0, the output pixel is set to 0 [37].

The dilation function applies the appropriate rule to the pixels in the neighborhood and assigns a value to the corresponding pixel in the output image by using structuring element. In figure 3.2, the morphological dilation function sets the value of the output pixel to 1 because one of the elements in the neighborhood defined by the structuring element is on. Structuring element is an essential part of the dilation operation which is used to probe the input image. It is a matrix consisting of only 0's and 1's that can have any arbitrary shape and size. It can be vertical, horizontal, cross-shaped, multi directional. Based on its shape, structuring element determine to what direction it increases the pixel value of an image [37].

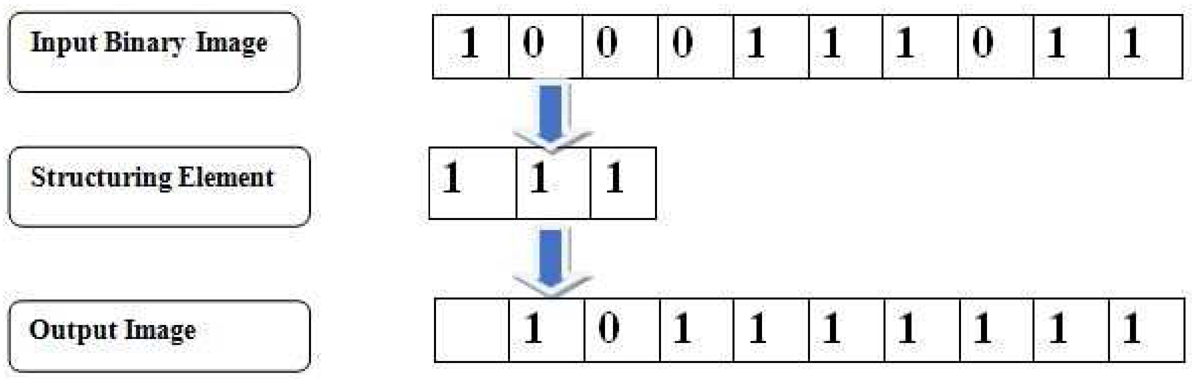


Figure 3-3 Morphology Dilution approach

3.1.2.2 Connected Component Analysis

Connected component (CC) labeling is used in image processing to detect connected regions in binary images. It is an algorithmic application of graph theory, where subsets of connected components within an image are uniquely labeled based on a given heuristic [36]. Connected components scans an image and groups its pixels into components based on pixel connectivity. All pixels in a connected component share similar pixel intensity values and are in some way connected with each other. Once all groups have been determined, each pixel is labeled with a gray level or a color (color labeling) according to the component it was assigned to.

Connectivity of pixels divided in to 4 and 8 connectivity in order to find the CC of the given image depending on its purpose. The difference between 4 and 8 CC connectivity labeling is how the algorithm defines connected pixels. For example, for the pixel P, 4 connectivity only checks the four neighbors, called direct-neighbors i.e. right, left, up and down neighbors of P whereas 8 connectivity is known as indirect-neighbors checks all the surrounding pixels around P including diagonal pixels. The labeled pixels represent pixels that are considered as connected to the central pixel in both approaches [36].

Once all groups have been determined, each pixel is labeled with a gray-level or color labeling according to the component it was assigned to. Extracting and labeling of various disjoint and connected components in an image is central to many automated image analysis applications such as OCR systems.

There are two types of connected component labeling algorithm; one pass and two pass. The one pass version goes through each pixel only once and for each pixel in an image, all the neighbor pixels are tested for connectivity to label connected components and the two pass scans the image two times. The first pass goes through each pixel and checks each pixel and using these pixel labels, it assigns a label to the current pixel and the second pass cleans up any mess it might have created. Two pass labeling takes high processing time and memory space than one pass. Algorithm 3.1 shows the steps followed in one pass connected component analysis.

```
1. Connected-component matrix is initialized to size of image matrix.
2. A marker is initialized and incremented for every detected object in the image.
3. A counter is initialized to count the number of objects.
4. A row-major scan is started for the entire image.
5. If an object pixel is detected, then following steps are repeated until (Index!=0)
        5.1. Set the corresponding pixel to 0 in Image.
        5.2. A vector (Index) is updated with all the neighboring pixels of the currently set
        pixels.
        5.3. Unique pixels are retained and already marked pixels are removed.
        5.4. Set the pixels indicated by Index to 1 in the connected-component matrix.
6. Increment the marker for another object in the image
```

Algorithm 3-1 One pass connected component labeling algorithm (36)

Two pass labeling scans the image two times as it has been mentioned earlier and algorithm 3.2 presents the two-pass connected component labeling algorithm [36].

```
        First Pass:
        1. Iterate through each element of the data by column, then by row (Raster Scanning)
        2. If the element is not the background
```

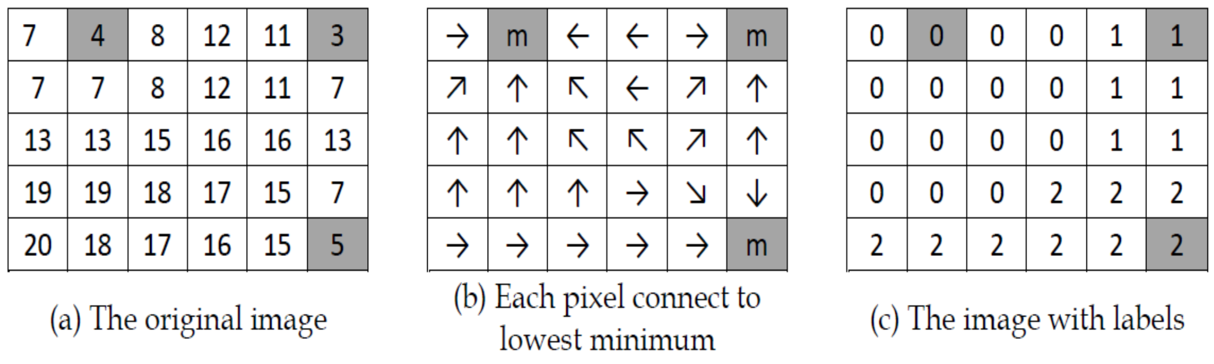
```
    2.1. Get the neighboring elements of the current element
    2.2. If there are no neighbors, uniquely label the current element and
    continue
    2.3. Otherwise, find the neighbor with the smallest label and assign it to the
    current element
    2.4. Store the equivalence between neighboring labels
Second Pass:
1. Iterate through each element of the data by column, then by row
2. If the element is not the background
2.1. Relabel the element with the lowest equivalent label
```

Algorithm 3-2 Two pass connected component labeling algorithm (36)

3.1.2.3 Watershed Algorithm Based on Connected Components

Watershed algorithm based on connected components is one of the algorithms used to segment Amharic PDF document in this study. This algorithm gives the same segmentation results as the traditional watershed algorithm. At the same time, it has an advantage of lower complexity, simple data structure and short execution time. It connects each pixel to its lowest neighbor pixel and all pixels connected to same lowest neighbor pixel, make a segment [55]. In the analysis of the objects in images it is essential that we can distinguish between the objects of interest and the rest. This latter group is also referred to as the background. The techniques that are used to find the objects of interest are usually referred to as segmentation techniques – segmenting the foreground from background[55].

The basic concept of connected components-based watershed algorithm is shown in Figure 3.2. The original 6 x 6 image has three local minimum values indicated by gray boxes (3.2a). If a pixel is not a local minimum then it is connected to its lowest neighbors as shown by arrows in (3.2b), where m indicates a local minimum. All components directed towards the same local minimum make a segment and are given the same label value (3.2c)[56].


(a) The original image
(b) Each pixel connect to lowest minimum
(c) The image with labels

7	4	8	12	11	3
7	7	8	12	11	7
13	13	15	16	16	13
19	19	18	17	15	7
20	18	17	16	15	5

→	m	←	←	→	m
↗	↑	↖	←	↗	↑
↑	↑	↖	↖	↗	↑
↑	↑	↑	→	↘	↓
→	→	→	→	→	m

0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	2	2	2
2	2	2	2	2	2

Figure 3-4 watershed algorithm[56]

3.1.2.4 Run Length Smoothing

The Run Length Smoothing Algorithm (RLSA) is a method that can be used for Block segmentation and text discrimination. The method developed for the Document Analysis System consists of two steps. First, a segmentation procedure subdivides the area of a document into regions (blocks), each of which should contain only one type of data (text, graphic, halftone image, etc.). Next, some basic features of these blocks are calculated[57].

The basic RLSA is applied to a binary sequence in which white pixels are represented by 0’s and black pixels by 1’s. The algorithm transforms a binary sequence x into an output sequence y according to the following rules[57]:

1. if the number of adjacent 0‘s is less than or equal to a predefined threshold C, then change 0‘s in x to 1‘s in y.

2. 1‘s in x are unchanged in y.

3.1.2.5 CC width, height and area analysis

Connected components width, height and area analysis is used to identify big connected areas like graphics, columns, titles, etc. in addition small connected elements like punctuation marks comas, and small dots. It is an algorithm that takes the width and height of the bounding box of a labeled component to calculate the area in order to get some threshold value. Column and graphical areas usually have larger area (height and width) than normal text while punctuation marks and dots have smaller area and height or width. Thus, finding the area of connected component is very important for the next object extraction.

3.1.2.6 Object Classification and Extraction

3.1.2.6.1 Object Classification

Different types of regions in the image have different textural characteristics. Having the information about the white tiles from page segmentation, the most appropriate textural property to exploit is the white space inside regions. Furthermore, it is natural that a practical page classification method should exploit this information instead of performing new computations on the image data. In the proposed approach, description of the regions identified during page segmentation is adapted to allow for efficient classification and object extraction.

- Text regions: contain a significant number of narrow white tiles. These are evenly distributed inside the region and the white area that they describe is large in proportion to the total area of the region.
- Graphics regions: usually contain less white space. There are more wide tiles than in text regions. The size of the tiles may vary significantly, and they are not evenly distributed.

3.1.2.6.2 Object Extraction

The features describing the properties of each block are extracted based on its edges. The coordinates of the edges specify the size of the block. Each extreme edges of block are defined by considering the origin of binary image. Features such as height, area, aspect ratio, perimeter, perimeter/height ratio, average horizontal length are extracted to differentiate the non-text block from text block. Textual and non-textual blocks are represented by (Xmin, Ymin) and (Xmax, Ymax). Where,

- Ymin - Left-most pixel value of column.
- Ymax - Right-most pixel value of column.
- Xmin- Left-most pixel value of row.
- Xmax - Right-most pixel value of row.

The following block features are computed for extracting the document region.

Height (H) - Height of the block is determined by subtracting the leftmost pixel from rightmost pixel of column.

H=Dx = (Ymax-Ymin) +1

Width (W) - Width of the block is computed by subtracting the leftmost pixel from rightmost pixel of row.

W=Dy = (Xmax-Xmin) +1

Area (A) - Area of the block is obtained by multiplying the height and width.

A=H*W

Non-Text Components Extraction

This stage performs extraction of non-text areas such as images, pictures and graphical areas based on the above segmentation result. It also intended to extract implicit information from non-textual PDF objects such as figures and images. After figure extraction from the page has been completed, the figure block is saved in Image folder based on its page number and (x,y) coordinate.

Text Area Extraction

In this stage extracting text areas such as text titles, paragraphs and columns based on the above segmentation result. After text areas extraction from the page has been completed, the block is saved in Text folder based on its page number and (x,y) coordinate and transfer to the next stage for OCR application.

3.1.4 Text Recognition using OCR

OCR is a technology that allows you to convert scanned images of text into plain text. This study explored three open source OCR engines namely Tesseract OCR, Finereader and FreeOCR. These OCR engines are experimented in different combinations on real-life Amharic document images.

3.1.4.1 Tesseract OCR

Tesseract is an open source optical character recognition (OCR) engine [7]. HP originally was originally started it as a project [44]. Later it was modified, improved and taken over by Google and later released as open source in year 2005. It is very portable as compared to others and supports various platforms. Its focus is more towards providing less rejection and improved accuracy. Currently only command base version is available but there are many projects with UI built on top of it which could be forked. As of now Tesseract version 3.02 is released and available for use. which provides support for around 139 language including Amharic. Figure 3.4 presents python code used to recognize Amharic text image using Tesseract.

```
for file in file_list:
    name=os.path.splitext(file)[0]
    # selecting image file type
    if file.endswith(".jpg"):
        txt=ocr(file) # calling the ocr function
                                    os.remove(name+".jpg")
        file = open(directory+"\\"+name+".txt",'a+', encoding="utf-8")
                                    #os.remove(file)


        file.write(str(txt))


print (succsess)
```

Figure 3-5 python code used to recognize Amharic text image using Tesseract

3.1.4.2 FineReader OCR

FineReader is produced commercially by a global company, called ABBYY, as advanced OCR engine. The performance of FineReader has been enhanced by ABBYY for many years. FineReader 12 supports 190 languages including Amharic script using dictionary support [58]. It supports multi-font types, multi-size and multi-resolution images.

3.1.4.3 FreeOCR

FreeOCR is a free Optical Character Recognition engine supports different popular image file formats. FreeOCR outputs plain text and can export directly to Microsoft Word format

3.1.5 Data Compression

After separation of graphics and text, this study applies and tested three lossless compression algorithms namely; Huffman compression algorithm, LZW Compression algorithm and RLA compression algorithm for the purpose of compressing all extracted textual and non-textual data.

3.1.5.1 The Huffman Compression Technique

Huffman's scheme uses a table of frequency of occurrence for each symbol (or character) in the input. This table may be derived from the input itself or from data which is representative of the input. For instance, the frequency of occurrence of letters in normal English might be derived from processing a large number of text documents and then used for encoding all text documents[23]. We then need to assign a variable-length bit string to each character that unambiguously represents that character. This means that the encoding for each character must have a unique prefix. If the characters to be encoded are arranged in a binary tree. The Huffman algorithm is simple and can be described in terms of creating a Huffman code tree[59].

3.1.5.2 Lempel-Ziv Compression Technique

LZW compression method is simple and is dictionary based. A file is scanned for a sequence of repetitive data occurring in the program. These sequences are stored in the dictionary within the compressed file and references are inserted wherever the repetitive data occurs[10].

LZW compression process is simple. It replaces strings of characters with single codes. No analysis is done of the incoming text. A new string of characters is added every time it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters. The LZW algorithm output code may be of arbitrary length, but it must have more bits in it than a single character[60].

3.1.5.2 Run-Length Encoding Compression Technique

Run-Length Encoding is one of the simplest compression techniques known created especially for data with strings of repeated symbols (the length of the string is called a run). It can be used to compress data made of any combination of symbols. It does not need to know the frequency of occurrence of symbols and can be very efficient if data is represented as 0s and 1s (Dipperstain, 1998). The general idea behind this method is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences. The method can be even more efficient if the data uses only two symbols (for example 0 and 1) in its bit pattern and one symbol is more frequent than the other.

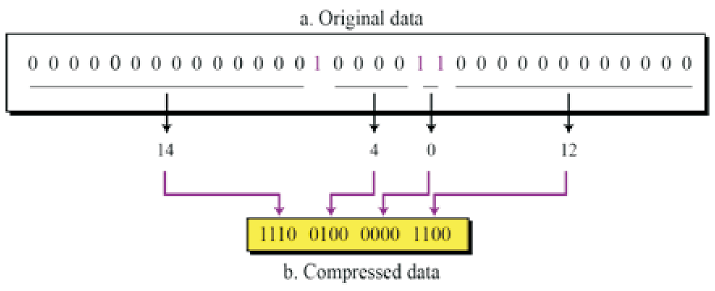


Figure 3-6 Run length Encoding

3.2 Performance Evaluation

For measuring the performance of page segmentation and reconstruction direct mapping is used. It determines the performance of layout segmentation and reconstruction by finding the correspondences between detected entities and ground truth. For page column segmentation it counts the expected correct and error segmentation using Over-segmentation. A region of the reference is represented by two or more regions in the examined segmentation. made by the proposed approach and calculate the segmentation accuracy percentage. The expected correct segmentation represents the expected number of segmented blocks. Similar procedures are followed to calculate accuracy percentage for page column and paragraph layout reconstructions.

The performance of data compression algorithms can be analyzed in several aspects. We can measure the algorithm complexity, computational memory, speed, amount of compression and quality of reconstructed data. The most common measure to calculate the efficiency of a compression algorithm is degree of compression. It is defined as the ratio of total number of bits required to store uncompressed data and total number of bits required to store compressed data.

CHAPTER FOUR
EXPERIMENTATION AND ANALYSIS

The main purpose of this study is designing Amharic PDF files compression approach by applying an effective page segmentation technique that can identify text and non-text blocks with the aim of reconstructing those segmented blocks based on the information stored during page segmentation and object extraction stage. The proposed techniques are integrated with Google developed Amharic OCR systems tesseract to recognize Amharic texts from textual objects.

For the experimentation purpose, HP Intel(R) Core (TM) I7 CPU 3110M @ 2.4GHz (2 CPUs), 8GB RAM and Windows 10 Ultimate operating system were used. MATLAB™ image processing toolbox R2019 and Python programming language using Spider development tool are used for developing prototype and integration.

4.1 Dataset preparation

Since the goal of this research is to compress Amharic PDF files by segmenting and extracting non-text and text object layouts of real-life PDF documents. Those documents are collected from the popular Amharic newspapers and from different magazines that encompasses multiple columns having graphics. Scanned PDF files from those documents are selected because they contain a number of page layouts, they have real-life features and they are easily accessible. Newspapers and magazines commonly have two columns page layout.

The proposed approach performance is measured on different stages. To measure the performance of the proposed segmentation, we use direct mapping. This method counts correct segmentation made by the proposed technique and compare it with the expected once to calculate the accuracy in percentage. The expected correct segmentation represents the expected number graphics, column blocks each page segmentation and reconstruction methods individually. For measuring the performance, the test set contains real life Amharic document images with multiple columns, such as graphics with columns and paragraphs inside.

The collected datasets enable to evaluate if the proposed techniques are adequate to preserving the original layouts after decompression process. The dataset doesn't contain Tables, handwritten and typewritten PDF documents; rather it contains multiple column real life documents that have graphics and a number of paragraphs inside. Table 4.1 summarizes PDF documents collections used in this study.

Table 4-1: Summarizing PDF documents collections

PDF Name	PDF Type	Documents Contain		Scanned	Pages	Size (MB)
		Images	Columns			
PDF1	Magazine	No	No	Yes	16	21
PDF2	Web document	No	No	Yes	4	1.3
PDF3	Web document	No	Yes	Yes	7	2.2
PDF4	News paper	No	Yes	Yes	18	4.6
PDF5	Web document	Yes	No	Yes	8	3.5
PDF6	News paper	Yes	No	Yes	8	2.1
PDF7	News paper	Yes	No	Yes	10	6.3
PDF8	Magazine	Yes	Yes	Yes	7	3.7
PDF9	Magazine	Yes	Yes	Yes	22	17.6
PDF10	Magazine	Yes	Yes	Yes	7	5.6
PDF11	News paper	Yes	Yes	Yes	13	6.9
PDF12	News paper	Yes	Yes	Yes	11	4.4

4.2 Preprocessing

Preprocessing component performs the job of preparing the input PDF documents for compression. to this end, different tasks are done, such as converting PDF to image, binarization and color space transformation.

4.2.1 PDF to Image conversion

To access and manipulate objects inside the PDF file we need to convert the PDF file to Image. This study uses PDF to Image python library to convert the given Amharic PDF file to its equivalent image format. Figure 4.1 depicts the python code used for converting PDF to jpeg image file format.

```
from wand.image import Image

Input path = "path"

Output path="output path"
```

```
with(Image(filename = "input path resolution=300, Output path=Output path)) as source:
    images = source.sequence
    pages = len(images)
    for i in range(pages):
        n = i + 1
        newfilename =  str(n) + '.jpeg'
        Image(images[i]).save(filename=newfilename)
```

Figure 4-1 PDF to Image conversion

The converting file format is CMYK color mode. This mode is not supported by MATLAB programming language. Hence, there is a need to transform the color space from CMYK to the primary color mode, RGB.

4.2.2 Color Space Transformation

To transform the color space from CMYK to RGB, we used python code presented in figure 4.2 this simplifies further process of images using MATLAB for binarization and experimenting different segmentation algorithms to detect text and non-text regions

```
from PIL import Image
import os, sys
input path = " input path
dirs = os.listdir( path )
def torgb():
    dirs2 = os.listdir(path2)
    for item in dirs:
        if item == '.jpeg':
            continue
        if os.path.isfile(path+item):
            im = Image.open(path+item)
            f, e = os.path.splitext(path+item)
            image = im.convert('RGB')
            image.save(f + '.jpeg')
torgb()
```

Figure 4-2 CMYK to RGB color space conversion

4.3.2 Binarization

Binarization is a process by which the gray scale or colored images are converted to binary images[15]. When an image is captured, it is frequently stored in the form of pixel density value, which means each pixel has a value between 0 and 255 for a grayscale image.

The technique used in this study is, the Otsu threshold. The Otsu threshold is popular threshold method, to convert gray-scale image into binary image. This threshold works to minimize the intra class variance of the black and white pixels.

In MATLAB this threshold can be found using a function called graythresh. Gray thresh compute global image threshold using Otsu's method. The syntax is given by:

level = graythresh (I);

where level is the global image threshold value, I is the global image.

Once the global image threshold value is obtained, binarization is accomplished by converting pixels which have greater intensity value than global image threshold value into “1” and pixel which have lesser intensity value than global image threshold value in to “0”. The typical MATLAB code of binarization is given by:

```
bw=im2bw (I, level)
```

Figure 4-3 Implementation of binarization

where “I” is the global image and level in the global image threshold value for I. Im2bw is a MATLAB function used to convert gray scale or colored image into binary image.

4.3 Page Layout Segmentation

In the proposed methods, once the PDF image is converted into binary document image, the next stage is applying page layout segmentation to separate text from non-text region and to store layout information of the segmented objects. Page layout segmentation is a basic stage of this study, because the remaining stages including object extraction, compression and reconstruction depends on this stage.

The proposed page layout segmentation techniques for separating graphics from text area, detecting and segmenting column blocks, collecting information about segmented column and identifying text areas from the PDF document images for further processing. Thus, page layout reconstruction techniques maintain the original document image layout based on the information collected in page layout segmentation stage.

4.3.1. Text and Graphic separation

Text/graphic segmentation in document images, which separate graphics from text area, is crucial for the next sub-sequential stage to separating text from non-text regions enables to use OCR for converting image into text that can optimize grater memory space requirement of PDF file. This study explored five segmentation techniques namely: morphological dilution, Horizontal Run Length Smoothing, watershed transforms, connected component labeling and Components Width, Height and Area Analysis. These techniques are experimented in different combinations on real-life Amharic PDF documents.

4.3.1.1 Morphological Dilution

The first step in the course of in textual and non-textual separation is dilation. It is used to connect disconnected characters, word and lines by increasing the pixel values of a PDF images. Depending on the structuring element there are different types of dilation. In this study vertical and horizontal direction dilations are applied. figure 4.5 shows MATLAB dilution algorithm that connects in a given image.

```
[dialatedIm] = Dilat(bw,tresh)

dialatedIm = bwdist(~bw) >= tresh;
```

Figure 4-4 Implementation of Dilation

The blow figure 4.5 shows results of experiment after applying the dilution using different threshold values

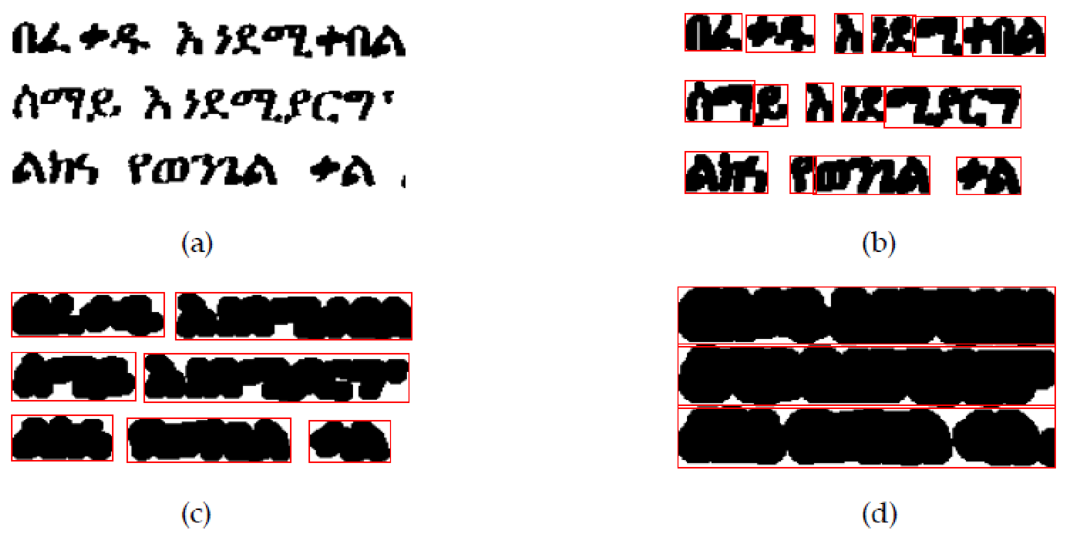


Figure 4-5 Dilation Using Different Thresholds:

(a) Original image (b) Result of smaller threshold (c) Dilated by medium threshold (d) Result of larger threshold

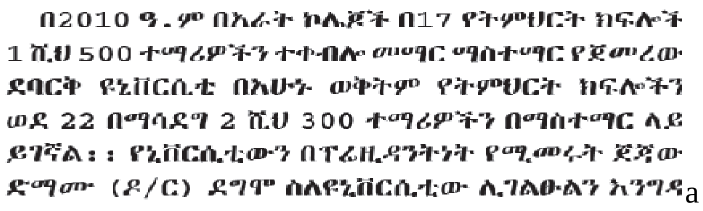
4.3.1.2 Horizontal Run Length Smoothing (HRLS)

Horizontal Run Length Smoothing is the other algorithm used to connect components in the given image. The algorithm provides the same result with morphological dilation. In dilution pixels expands in all direction (right, left, up and down) whereas, in Horizontal Run Length Smoothing expand characters in only one direction (to the right) that is the only difference of the two algorithms

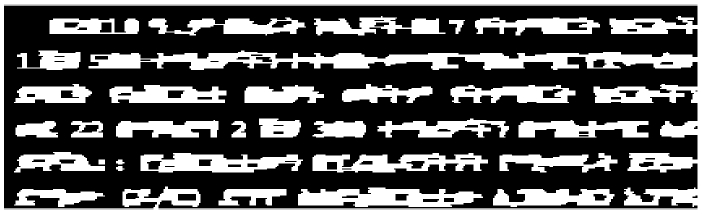
```
function [HRLSResult] = HRLS(bw1,bw2)
[rows, columns] = size(bw1);
for i = 1 : rows
    for j = 1 : columns
        if (j>4)
            if (bw2(i,j)==1)
                continue;
            else
                if(bw2(i,j-1)==1|bw2(i,j-2)==1|…
                   bw2(i,j-3)==1|bw2(i,j-4)==1)
                   bw1(i,j)=110;
                end
            end
        end
    end
end
```

Figure 4-6 Implementation of Horizontal Run Length Smoothing

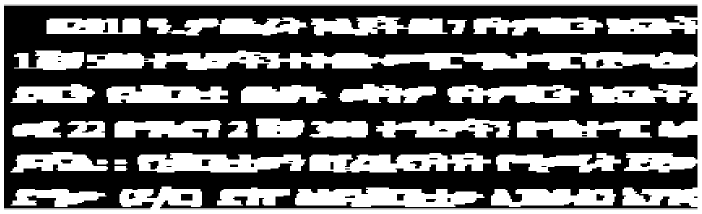
However, as it is observed from Figure 4.7 below the result of both algorithms (Dilation and HRLS) to segment words is the same in our document collection.


በ2010 ዓ.ም በአራት ኮሌጆች በ17 የትምህርት ክፍሎች
1 ሺ.ህ 500 ተማሪዎችን ተቀብሎ መማር ማስተማር የጀመረው
ደባርቅ ዩኒቨርሲቲ በአሁኑ ወቅትም የትምህርት ክፍሎችን
ወደ 22 በማሳደግ 2 ሺ.ህ 300 ተማሪዎችን በማስተማር ላይ
ይገኛል፡፡ ዩኒቨርሲቲውን በፕሬዚዳንትነት የሚመሩት ጀኛው
ድማሙ (ዶ/ር) ደግሞ ስለዩኒቨርሲቲው ሊገልፁልን እንግዳ a

Original image(a)



HRLS connecting characters(b)



Dilation connecting characters(c)

Figure 4-7 Dilation and HRLS Connected Words

In the proposed approach Dilation is used because the MATLAB built in function for Dilation is faster and easier to implement than HRLS algorithm and both performed equally in our document collection.

4.3.1.3 Connected Components Labeling

After applying morphological dilution, the next step is connected component analysis. Connected components (CC) labeling algorithm is applied to identify and label each connected component in a given binary image. MATLAB built-in method bwconncomp() and bwlabel() are used to identify connected component and to label them in a given binary image respectively. In this study 4 connectivity of pixel are used to identify connected components. figure 4.3 shows CC algorithm that identifies connected components in a given image.

Connected components (CC) labeling algorithm which identify and label each connected component in a given binary image is implemented using MATLAB built-in method bwconncomp (). The code used to implement CC is given in Listing 4.1.

```
[cc,num] = ConnectedComp(bw)
cc = bwconncomp(bw,4) % using 4 connectivity
num=cc.NumObjects;
```

Figure 4-8 Implementation of Connected Components

Figure 4-10 shows results of experiment after applying the connected component analysis

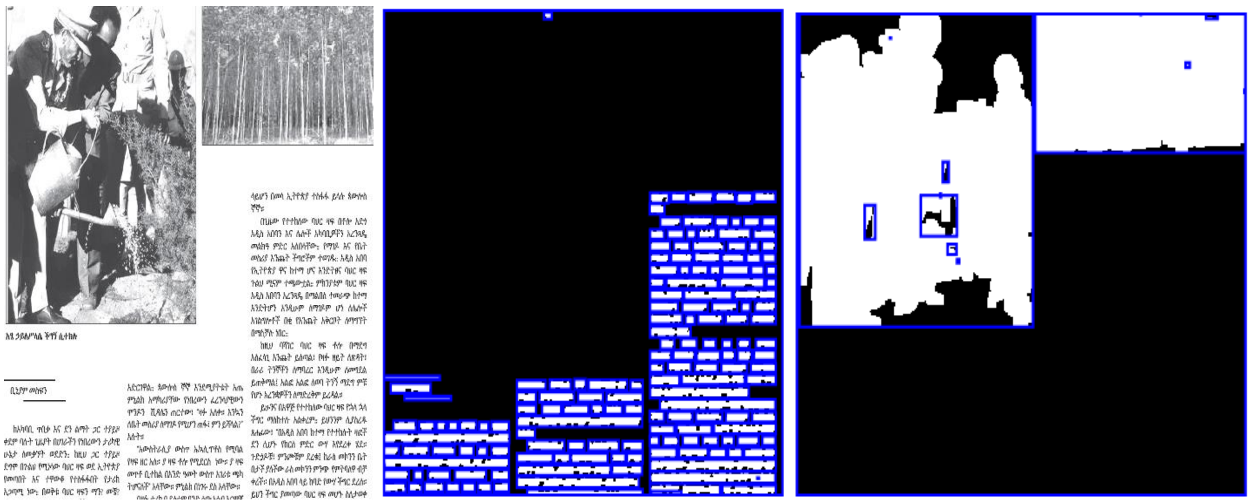


Figure 4-9 Experimental result connected component segmentation

(a) Original image (b) connected component texts regions (c) connected component non text regions

4.3.1.4 Watershed Segmentation

Watershed algorithm is implemented using MATLAB built-in function WaterSh(). The algorithm segments document images to word images when integrated with dilation or HRLS The code below Figure 4. 11 is used to apply watershed algorithm.

```
function [Segmented] = WaterSh(bw)

Segmented = watershed(bw);
```

Figure 4-10 Implementation of Watershed

The blow figure 4.12 shows results of experiment after applying the Watershed Segmentation

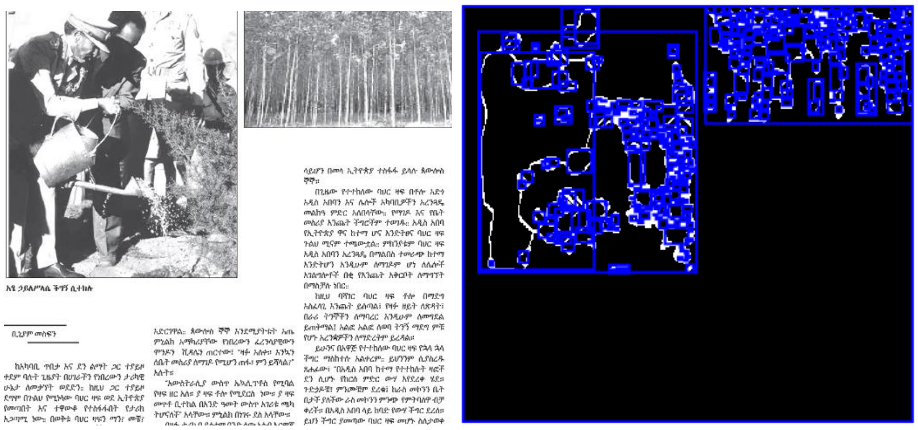


Figure 4-11 Experimental result Watershed Segmentation

(a) Original image (b) water shade segmentation of non-text regions

4.3.1.5 Connected Components Width, Height and Area Analysis

Components width, height and area analysis is used to identify big connected elements like images, graphics, logos, etc. and small connected elements like punctuation marks and small dots. Images and graphics usually have larger area and height or width than normal text while punctuation marks and dots have smaller area and height or width. Once connected components are extracted, it is used to identify big connected elements like graphics, column, etc. and small connected elements like punctuation marks and small dots. Graphics and Images usually have larger height, width and area than normal text while punctuation marks, dots and others have smaller area as well as height and width. Thus, a threshold value in order to separate text from graphics is set by taking the fact that graphics have larger area than text. The width and height of the bounding box are used to compute the area for each component and saved on array size info to compare the results. This study adopts a threshold value of 8000 from previse study[61], which is found to be a better threshold value. Detail implementation is shown in figure 4.13.

```
size_info = [];
cc = 1;
forcnt = 1:num
        x = Ibox(:,cnt);
        size_info (cc,1) = x(3,:,1);
        size_info (cc,2) = x(4,:,1);
        size_info (cc,3) = x(3,:,1) * x(4,:,1);
        cc = cc + 1;
        if (size_info(cnt,3) >8000)
        rectangle ('position',Ibox(:,cnt),'edgecolor','r');
        end
end
```

Figure 4-12 analysis for Text\Graphics separation

Figure 4.14 shows result of experiment after applying the text/graphics separation techniques, morphological dilation, connected component analysis and components Width, Height and Area Analysis

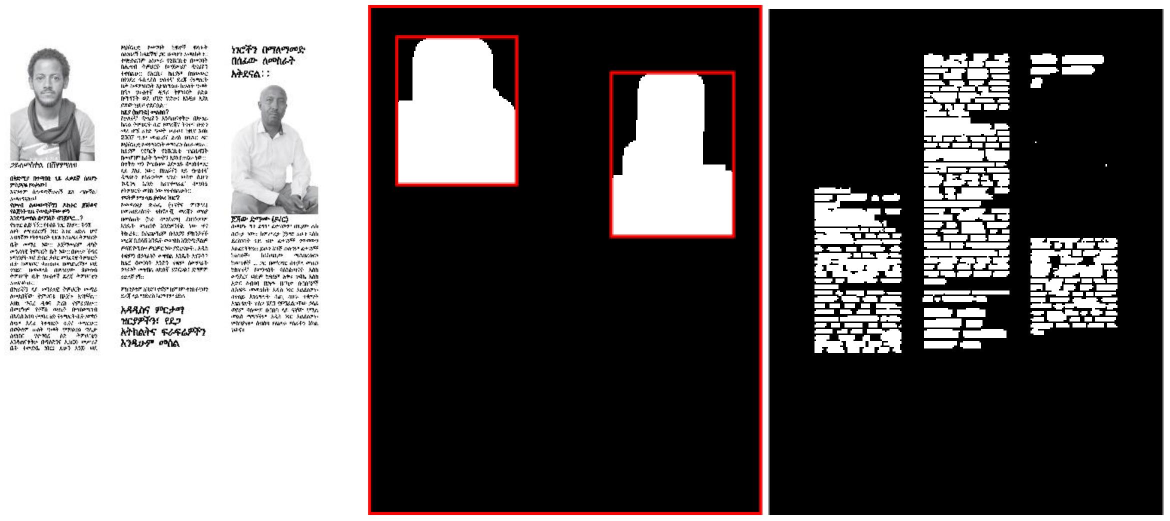


Figure 4-13 Experimental result after applying the proposed text/graphics separation approach.

4.3.1.6 Experimental Results for text/graphics separation

To choose whether connected component or watershed best perform when integrated with Dilation and connected components Area, Height and width analysis, on different type of PDF documents, the blow experiments are conducted on different PDF documents.

Real life PDF documents collected for this study has a total number of 12 PDF documents with different pages sizes. From the total datasets eight of the PDF documents have graphics content inside. The blow tables Table 4.2 and Table 4.3 shows the CC or Watershed best perform when integrated with Dilation and connected components Area, Height and width analysis performance of the text/graphics separation proposed in this study.

Table 4-2: Experimental result of the proposed text/graphics separation using connected component with Dilation and connected components Area, Height and width analysis techniques

PDF Name	Pages	Correctly Segmented	Erroneously Segmented	Accuracy (%)
PDF1	16	11	5	68.75
PDF2	4	4	0	100.00
PDF3	7	6	1	85.71
PDF4	18	14	4	77.78
PDF5	8	6	2	75.00
PDF6	8	7	1	87.50
PDF7	10	8	2	80.00
PDF8	7	5	2	71.43
PDF9	22	16	6	72.73
PDF10	7	5	2	71.43
PDF11	13	10	3	76.92
PDF12	11	8	3	72.73
Average				78.33

Table 4-3: Experimental result of the proposed text/graphics separation using watershed segmentation with Dilation and connected components Area, Height and width analysis techniques

PDF Name	Pages	Correctly Segmented	Erroneously Segmented	Accuracy%
PDF1	16	9	7	56.25
PDF2	4	4	0	100.00
PDF3	7	5	2	71.43
PDF4	18	12	6	66.67
PDF5	8	5	3	62.50
PDF6	8	5	3	62.50
PDF7	10	8	2	80.00
PDF8	7	5	2	71.43
PDF9	22	17	5	77.27
PDF10	7	5	2	71.43
PDF11	13	8	5	61.54
PDF12	11	8	3	72.73
Average				71.15

The above tables Table 4.2 and Table 4.3 shows the result of integrating connected component algorithm with components Area, Height and width analysis and Dilation result better result than Watershed implementation. Therefore, the integration of connected component, with Dilation and connected component Area, Height and Width analysis, is proposed. The result indicates that the proposed text and non-textual separation technique based on connected component labeling works better on real life PDF documents to separate graphics from text and on average 78.33 %

accuracy rate achieved. However, the algorithm failed to recognize smaller graphics which have smaller area below the threshold value set.

The algorithm works well in identifying connected components. However, parts of broken fonts and disconnected figure elements are considered as different components.

A serious challenge in some documents is a problem that arises due to underlines. Underlines that appear in the middle of a text region as shown in figure 3.15. These underlines are segmented as graphical components and are removed from the set of text components. Moreover, in some situations where text characters are attached to the underline, not only the underline disappear from the text region, it takes some characters with it and leaves large gaps in the middle of a text region. This has a negative effect on our region detection stage when it happens.

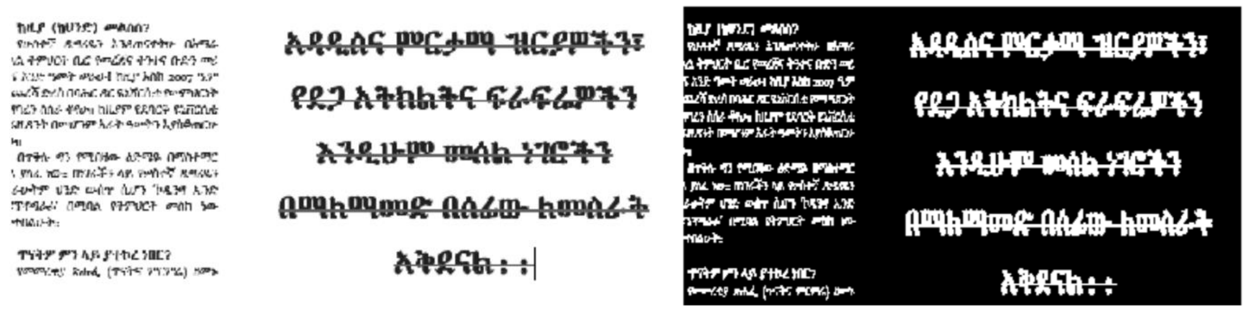


Figure 4-14 Effect of Underlines in segmentation:

Another challenge irregular shaped images with non-rectangular shaped text blocks may result in loss of some text. In this case, image might be misinterpreted as text. figure 3.15 shows the effect of irregular shaped images.

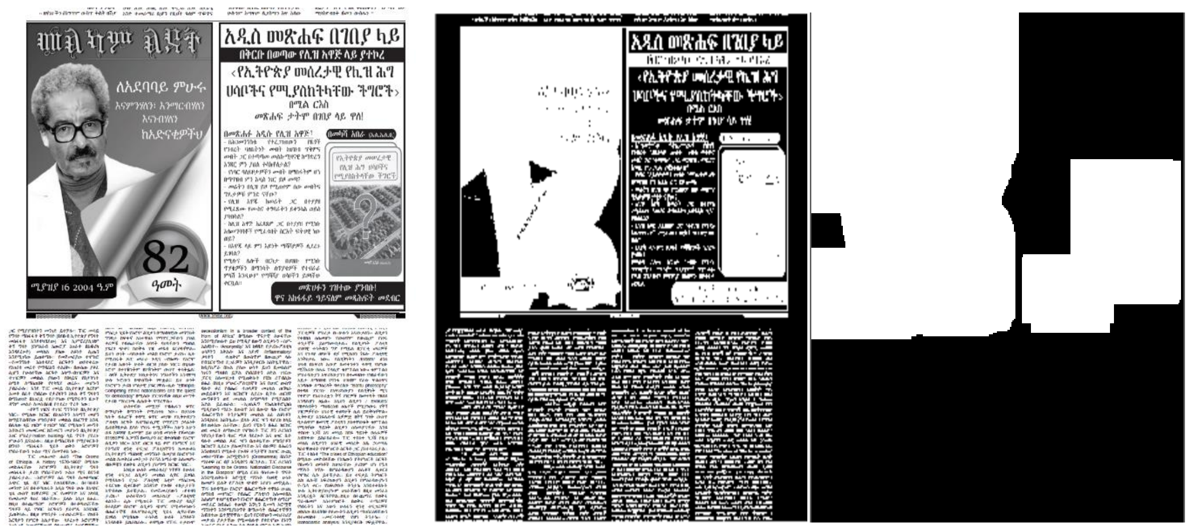


Figure 4-15 Effect of irregular shaped images

4.4.2 Column Block Segmentation

After separating text area from graphics, the next step in layout segmentation is column block detection and segmentation. Document images might contain different column blocks. So, it is important to detect and segment those regions for the next OCR process. For text/graphics separation, this study proposed morphological dilation and CC labeling for column block detection. But, in order to keep the white space between the column blocks during column block segmentation vertical dilation is applied. Implementation of vertical dilution detail is shown in figure 4.17 blow.

```
textRegions = bwconvhull(textRegions, 'objects');
textRegions = bwareafilt(textRegions, [, inf]);
se = strel('line', ,);
Verticald = imdilate(textRegions,se);
```

Figure 4-16 Implementation of vertical dilution

Then the area of connected component analyzed in order to find a threshold value to identify column block employed. Figure 4.18 shows a MATLAB algorithm to implement CC width, height and area analysis.

```
size_info = [];
sumArea = 0;
forcnt = 1:num
            component_area = component_width * component_height;
            size_info (cnt,1) = component_width;
            size_info (cnt,2) = component_height;
            size_info (cnt,3) = component_area;
            sumArea = sumArea + component_area;
end
maxArea = max(size_info);
forcnt = 1:num
x = Ibox(:,cnt);
if (size_info (cnt,2) >maxArea(1,2)/4 &&size_info (cnt,1) >maxArea(1,1)/4
            size_info (cnt,1) >maxArea(1,1)/4)
            rectangle('position',Ibox(:,cnt),'edgecolor','r');
end
end
```

Figure 4-17 Analysis for Text column segmentation

Andualem[61] conducted an iterative experiment, and proposed the height and width of labeled connected component which are greater than one fourth of the maximum area are considered as column block of an image document. Figure 4.19 shows the result of connected pixels after dilation algorithm is performed that connects characters, words and text lines in the same column.

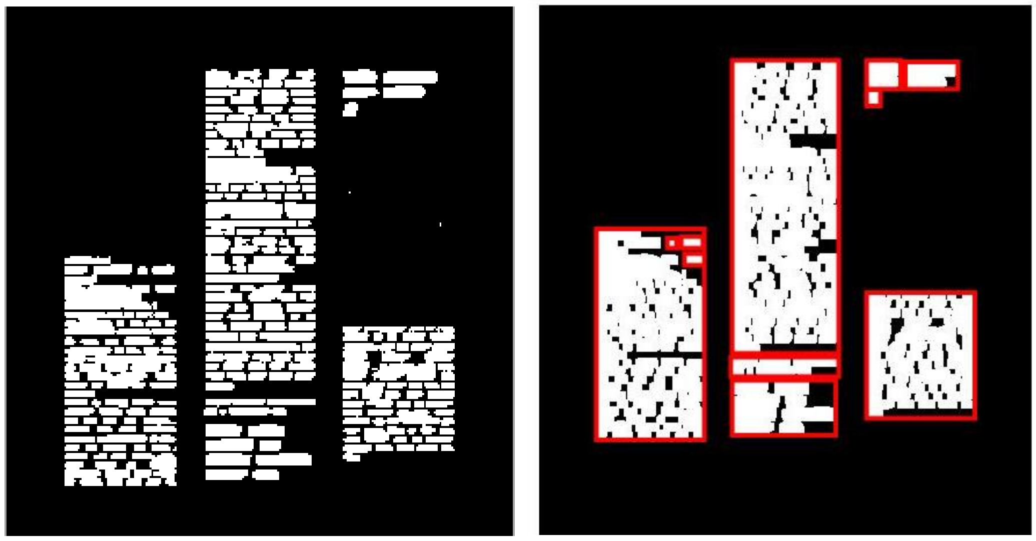


Figure 4-18 Experimental result after the proposed Text column segmentation applied.

As we can see from the above vertically dilated image (figure 4.19 (b)), the vertical dilation algorithm connects all the pixels only in vertical direction so that the space between the two columns is kept. Once the dilation process is done CC labeling analysis is applied and labels all the connected component regions. Then the area of connected component analyzed in order to extract the segmented text region to the next OCR stage.

4.4.2.1 Experimental results for column layout segmentation

Most of Amharic PDF documents such as newspapers and magazine have more than two column layouts. However, Thus, the proposed algorithm tested on PDF documents which have up to four numbers of columns. The blow table 4.4 shows the excremental result of the proposed approach column layout segmentation.

Table 4-4 The performance of the proposed column blocks segmentation techniques

PDF Name	Pages	Documents contain column	Correctly Segmented	Erroneously Segmented	Accuracy%
PDF3	7	2	7	0	100.00
PDF4	18	3	16	2	88.89
PDF8	7	3	6	1	85.71
PDF9	22	2	20	2	90.91
PDF10	7	4	7	0	100.00
PDF11	13	3	10	3	76.92
PDF12	11	3	9	2	81.82
Average					89.18

The experimental result shows that the proposed technique works fine for any number of columns with on average accuracy of 89.1%. However, the proposed approach erronsly segment paragraphs as a column when the whitespace between two consecutive paragraphs is larger. It also merged different blocks of columns as one when it failed to keep the whitespace between the columns. The experiment showed that using larger thresholds will result in merged words and using small threshold will result in over segmentation. Another challenge is broken titles. Figure 4.20 shows this problem in part of a document image. The title on this page is divided into six parts when there is enough empty space around.

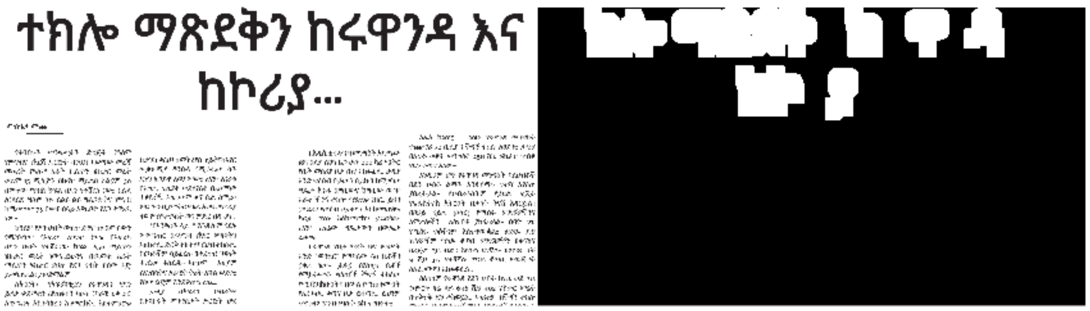


Figure 4-19 Effect of broken titles

4.4.3 PDF Objects Extraction

After all PDF images segmentation is performed PDF Image objects extraction process is started based on the output of segmentation process. In order to extract each text and non-text region the proposed approach applies the bounding box (x,y) coordinates for each objects.

L = {S1, Sn}, where L and S represent layout and segment respectively

A segment is a pixel collection encapsulated within a bounding box defined by its lower left and upper right corner pixels:

S = (P1, P2), where S and P represent segments and pixels accordingly

Each pixel is defined by a coordinate pair:

P = (x, y)

Based on this structure, in the proposed approach object extraction is done through the following steps:

- computing the bounding box for the connected components
- finding all edge pixels of the bounding box
- calculate maximum y, minimum y, maximum x and minimum y coordinate values, whereas (max Y, min X) and (min Y, Max X) will be the left upper and the right lower corner values of the segmented image

Based on the above coordinate the object is cropped from the original RGB PDF Image.

4.4.3.1 Non-Text Components Extraction

This stage performs extracting non text areas such as images, pictures and graphical areas based on the above segmentation result. Figure 4.21 shows the source code use to extract non text components.

```
max(max(Textlabel)); % This Command gives us the maximum objects detected.
mkdir([‘Output path', savefilename,'\\Text']);
pagefolderColumens=['C Output path ', savefilename,'\Text'];
measurements2 = regionprops(BW3, 'Area', 'BoundingBox');
allAreas2 = [measurements2.Area];
% Crop out each word
for blob = 1 : length(measurements2)
  % Get the bounding box.
  thisBoundingBox = measurements2(blob).BoundingBox;
  % Crop it out of the original gray scale image.
  thisWord = imcrop(rgbImage, thisBoundingBox);
  x=fix(thisBoundingBox(1));
  y=fix(thisBoundingBox(2));
  FileName = sprintf('%d, %d.jpg', x, y);
  fullFileName = fullfile(pagefolderColumens, FileName);
  imwrite(thisWord, fullFileName);
```

```
end
```

Figure 4-20 the source code used to extract non text components

Experimental results of image extraction from PDF file is presented blow in figure 4.22

Figure 4-21 Experimental results of image extraction from PDF file

After non text regions extraction from the page has been completed, the image is appended to its own folder with proper naming for easy identification during reconstruction.

4.4.3.2 Textual Components Extraction

After non textual objects extracted the next steps is extracting textual objects from the above page layout segmentation. The output of this stage is used for text recognition by applying Tesseract. Like the non text abjects, the text block is appended to its own folder and the information collected in page layout segmentation stage is stored for the application of next reconstruction of the original image. Implementation detail used for text region extraction is shown in figure 4.23

```
max(max(Textlabel));
mkdir([‘Output path', savefilename,'\\Text']);
pagefolderColumens=['C Output path ', savefilename,'\Text'];
measurements2 = regionprops(BW3, 'Area', 'BoundingBox');
allAreas2 = [measurements2.Area];
% Crop out each word
for blob = 1 : length(measurements2)
  % Get the bounding box.
  thisBoundingBox = measurements2(blob).BoundingBox;
  % Crop it out of the original gray scale image.
  thisWord = imcrop(rgbImage, thisBoundingBox);
  x=fix(thisBoundingBox(1));
```

```
 y=fix(thisBoundingBox(2));
 FileName = sprintf('%d, %d.jpg', x, y);
 fullFileName = fullfile(pagefolderColumens, FileName);
 imwrite(thisWord, fullFileName);
end
```

Figure 4-22 Implementation detail used for text region extraction

4.4 OCR Application

Once the text region is detected and separated from non-text region, the next step is recognizing the text regions. In this study to get a better text recognition three OCR engines are tested namely: tesseract OCR, FineReader OCR and FreeOCR.

4.4.1 Text recognition Using tesseract OCR

Python built-in method pytesseract () is used to perform tesseract OCR process as shown blow in figure 4.24.

```
for file in file_list:
    name=os.path.splitext(file)[0]
    # selecting image file type
    if file.endswith(".jpg"):
        txt=ocr(file) # calling the ocr function
                                        os.remove(name+".jpg")
        file = open(directory+"\\"+name+".txt",'a+', encoding="utf-8")
                                        #os.remove(file)

        file.write(str(txt))

print ("Image Conversion completed")
```

Figure 4-23 Use of tesseract OCR for Amharic text recognition

4.4.2 Text recognition Using FineReader OCR

Python implementation of FineReader OCR is presented on figure 4.24

```
def process (self, binary):
        hocr = ""
        with tempfile.NamedTemporaryFile(delete=False) as tmp:
            tmp.close()
            with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as btmp:
```

```
            btmp.close()
            self.write_binary(btmp.name, binary)
            self.set_image_dpi(btmp.name)
            args = self.get_command(tmp.name, btmp.name)
            self.logger.debug("Running: '%s'", " ".join(args))
            proc = sp.Popen(args, stderr=sp.PIPE)
            err = proc.stderr.read()
            if proc.wait() != 0:
                return "!!! %s CONVERSION ERROR %d: %s !!!" % (
                        os.path.basename(self.binary).upper(),
                        proc.returncode, err)
            hocr = utils.hocr_from_abbyy(tmp.name)
        os.unlink(tmp.name)
        os.unlink(btmp.name)
    utils.set_progress(self.logger, self.progress_func, 100, 100)
    return hocr
```

Once the extracted textual object is recognized using OCR it is removed from the folder to minimize the size of the extracted objects.

4.4.3 Text recognition Using FreeOCR

FreeOCR is a free Optical Character Recognition Software for Windows and supports scanning from most Twain scanners and can also open most scanned PDF's and multi-page Tiff images as well as popular image file formats. FreeOCR outputs plain text and can export directly to Microsoft Word format.

4.4.1.4 Experimental result for OCR Recognition

Table 4.5 presents performance of the different OCR engines to recognize texts from segmented text blocks.

Table 4-5: Performance of the three OCR Engines

PDF file	Text Blocks	Tesseract OCR (%)	FineReader OCR(%)	FreeOCR(%)
PDF1	16	97.37	92.11	92.11
PDF2	4	85.35	88.24	82.35
PDF3	7	82.78	77.78	72.22
PDF4	18	95.45	81.82	86.36
PDF5	8	93.33	93.33	86.67
PDF6	8	90.91	81.82	86.36
PDF7	10	90.32	83.87	87.10
PDF8	7	87.50	80.36	82.14
PDF9	22	96.18	91.72	95.54
PDF10	7	82.00	64.00	68.00
PDF11	13	96.15	91.03	93.59
PDF12	11	93.51	88.31	88.31
Average		90.90	84.53	85.06

From the above Table shows that the tesseract provides better recognition accuracy of 90.9% for extracted text blocks. This is due to the preprocessing in tesseract OCR engines is better than the other two OCR engines. Therefore, tesseract OCR integration for this study proposed.

4.5 Encoding

After all of the above stages are completed the final stage of the proposed approach is encoding all of the extracted non-text objects and the OCR generated textual files. The proposed approach applied LZW, Huffman Compression and RLE algorithms for encoding.

All Compression algorithms is implemented using Python. The algorithm encodes and compresses all extracted data and appended to .hc file format.

4.5.1 Compression using Huffman Compression

Huffman coding [24] is based on the frequency of occurrence of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently. Codes are stored in a Code Book which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding. The code shown blow in figures shows implementation of Huffman Compression.

```
def main():
# Assign the name of the directory to zip
  dir_name = "file path"
  # Call the function to retrieve all files and folders of the assigned directory
  filePaths = retrieve_file_paths(dir_name)

  # printing the list of all files to be zipped
 # print('The following list of files will be zipped:')
  for fileName in filePaths:
    print(fileName)

  # writing files to a zipfile
  zip_file = huffman.huffman(dir_name+'.hc', 'w')
  with zip_file:
    # writing each file one by one
    for file in filePaths:
      zip_file.write(file)

  print(dir_name+'.zip file is created successfully!')
# Call the main function
if __name__ == "__main__":
  main()
```

Figure 4-24 Implementation of Huffman compression

4.5.2 Compression using LZW Compression

Python implementation detail used for LZW compression is shown on blow figure 4.25

```
def main ():
for symbol in data:
    string_plus_symbol = string + symbol # get input symbol.
    if string_plus_symbol in dictionary:
        string = string_plus_symbol
    else:
        compressed_data.append(dictionary[string])
        if(len(dictionary) <= maximum_table_size):
            dictionary[string_plus_symbol] = dictionary_size
            dictionary_size += 1
        string = symbol
if string in dictionary:
    compressed_data.append(dictionary[string])
# storing the compressed string into a file (byte-wise).
```

```
out = input_file.split(".")[0]
output_file = open (out + ".lzw", "wb")
for data in compressed_data:
    output_file.write(pack('>H',int(data)))
output_file.close()
file.close()
```

Figure 4-25 Implementation of Huffman compression

4.5.2 Compression using RLE Compression

Python implementation detail used for LZW compression is shown on blow figure 4.26

```
def encode(string):
    if string == '':
        return ''
    i = 0
    count = 0
    letter = string[i]
    rle = []
    while i <= len(string) - 1:
        while string[i] == letter:
            i+= 1
            count +=1
            if i > len(string) - 1:
                break
        if count == 1:
            rle.append('{0}'.format(letter))
        else:
            rle.append('{0}{1}'.format(count, letter))
        if i > len(string) - 1:
            break
        letter = string[i]
        count = 0
    final = ''.join(rle)
    return final
```

Figure 4-26 Implementation of RLA compression

Several experiments had been performed to Study of efficiency LZW, Huffman compression and RLA Techniques in Amharic PDF documents. Experimental result of the three compression algorithms presented on the blow table 4.6.

Table 4-6: Experimental result of the three compression algorithms

File Name	Original Size	Proposed Approach using Huffman		Proposed Approach Using LZW		Proposed Approach Using RLE	
		Size After Compression(mb)	Performance (%)	Size After Compression (mb)	Performance (%)	Size After Compression(mb)	Performance (%)
PDF1	21	3	85.71	2.3	89.05	2.1	90.00
PDF2	1.3	0.5	61.54	0.8	38.46	0.9	30.77
PDF3	2.2	0.7	68.18	0.7	68.18	1.3	40.91
PDF4	4.6	0.9	80.43	0.7	84.78	0.7	84.78
PDF5	3.5	1.5	57.14	1.8	48.57	1.2	65.71
PDF6	2.1	1.1	47.62	1.5	28.57	1.2	42.86
PDF7	6.3	4.2	33.33	4.6	26.98	3.9	38.10
PDF8	3.7	1.2	67.57	1.2	67.57	1.2	67.57
PDF9	17.6	9.2	47.73	7.8	55.68	1.8	89.77
PDF10	5.6	3.2	42.86	4.3	23.21	3.7	33.93
PDF11	6.9	3.2	53.62	4.5	34.78	4	42.03
PDF12	4.4	1.3	70.45	0.6	86.36	1.1	75.00
Average		2.5	59.68	2.57	54.35	1.93	58.45

From the above experiment we find that every data has decreased from its original size. The performance of the data compressor depends on the content of the extracted files, the different images contained in it, and symbols frequencies. From the above experiment, it is noted that compression efficiency of LZW is effective for textual segmented, and it give better results than Huffman and RLA compression. Huffman compression is better compression result for extracted objects containing both textual and non-textual objects. After testing those techniques on different files of different sizes, we propose that Huffman compression overall performance gives better result for all categories of the Amharic PDF file.

4.6 Decompression and Reconstruction

In order to use a compressed file when it’s needed, we must first decompress it. After the decompression the original PDF file is reconstructed.

4.6.1 Decompression

Any compression algorithm will not work unless a means of decompression is also provided due to the nature of data compression. The proposed approach decompression process has been designed such that when an already compressed file (.hc file) from the above process is located and chosen to be decompressed, the compression technique that the proposed approach applies Huffman and LZW decompression algorithm for the purpose decompressing the compressed file.

4.6.2 Reconstruction

The proposed approach is also reconstructing the original PDF file after decompression.
the following steps are followed to reconstruct the original PDF file.

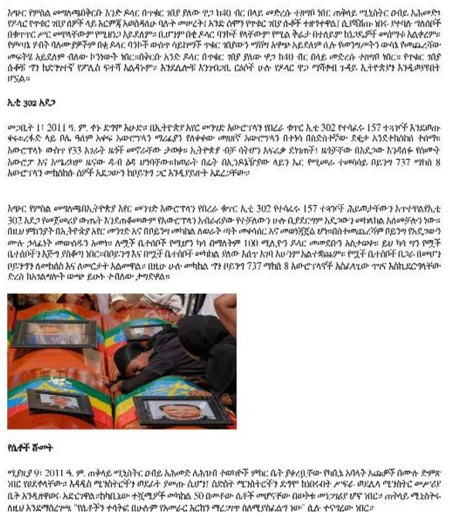
1. Decompose the compressed data and obtain the extracted file with their own coordinates.
2. According to the (x, y) coordinates, the proposed approach put in their position text and images.
3. The above step iterates over the entire pages of the PDF file.

Figure 4.27 presents reconstruction of text and non-text regions into original position for viewing to users.

```
def createImageText():
    dirs = len(os.walk(path).__next__()[1])
    for i in range(1,dirs+1):
        img = np.zeros([3300,2550,3],dtype=np.uint8)
        img.fill(255)
        cv2.imwrite(outpath + str(i) + '.jpeg',img)
        image = Image.open(outpath + str(i) + '.jpeg')
        filelistall = path + str(i) + '/Text/'
        dirs2 = os.listdir(filelistall)
        for item in dirs2:
            textfile=open(filelistall+item, "r", encoding='utf-8')
            contents =textfile.read()
            font = ImageFont.truetype('C:/Users/HP ADMIN/Desktop/matlab code/gfzemen.ttf', 25)
            f, e = os.path.splitext(os.path.basename(filelistall+item))
            x, y = [x.strip() for x in f.split(',')]
            x2 = int(x)
            y2 = int(y)
            image = Image.open(outpath + str(i) + '.jpeg')
            d = ImageDraw.Draw(image)
            location = (x2, y2)
            text_color = (100, 100, 200)
            d.text(location, contents, fill=text_color, font=font)
            #image.paste(croped, (x2, y2))
        image.save(outpath + str(i) + '.jpeg', quality=100)
createImageText()
```

Figure 4-27 presents reconstruction of text and non-text regions

Sample experimental result of the proposed reconstruction step is shown in figure 4.28



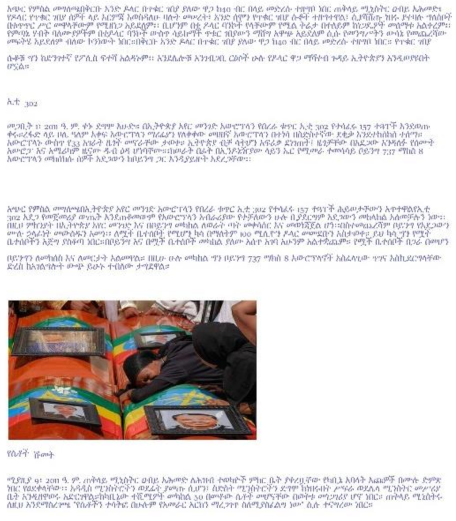


Figure 4-28 Experimental result after the original Image and constructed image after decompression

The proposed layout reconstruction technique properly reconstructs extracted blocks which are correctly segmented in terms of their physical coordinates. That means, the performance of the proposed technique is heavily depending on the extent to which the page segmentation method properly detects column and paragraph block.

4.6.3 Experimental result for PDF document reconstruction

Page column and paragraph reconstruction of the proposed technique uses the stored information from page segmentation stage to reconstruct the segmented blocks. Table 4.8 and Table 4.9 presents performance of the proposed column and paragraph block reconstruction.

Table 4-7: Experimentation result of the proposed reconstruction techniques from the whole test PDF files

	Experiment Result Decompression and PDF Reconstruction			
	From the whole test PDF files			
	No of pages	**Correctly reconstructed**	**Erroneously reconstructed**	**Accuracy (%)**
PDF1	16	10	6	62.50
PDF2	4	4	0	100.00
PDF3	7	6	1	85.71
PDF4	18	12	6	66.67
PDF5	8	4	4	50.00
PDF6	8	6	2	75.00
PDF7	10	9	1	90.00
PDF8	7	5	2	71.43
PDF9	22	18	4	81.82
PDF10	7	5	2	71.43
PDF11	13	8	5	61.54
PDF12	11	8	3	72.73
Average				74.07

The above experimental result shows layout reconstruction from the whole test PDF files which includes wrongly segmented because to reconstruct the image it uses the positional information from in page layout segmentation.

Table 4-8: Experimentation result of the proposed reconstruction techniques from correctly segmented PDF files

	Experiment Result Decompression and PDF Reconstruction			
	From the Correctly segmented PDF files			
	No of pages	**Correctly reconstructed**	**Erroneously reconstructed**	**Accuracy (%)**
PDF1	11	10	1	90.91
PDF2	4	4	0	100.00
PDF3	6	6	0	100.00
PDF4	14	12	2	85.71
PDF5	6	4	2	66.67
PDF6	7	7	0	100.00
PDF7	8	7	1	87.50
PDF8	5	5	0	100.00
PDF9	16	16	0	100.00
PDF10	5	5	0	100.00
PDF11	10	8	2	80.00
PDF12	8	8	0	100.00
Average				92.57

The experimental result shows that the proposed PDF layout reconstruction techniques depends on the page segmentation stage. It accurately reconstructs column block layouts that are correctly segmented in terms of the number. Because the reconstruction process only focuses on the number of segmented layout block, it doesn’t reconstruct column layouts based on their width size. As a result, the reconstructed column layouts have equal width size

4.7 Experimental result of the proposed compression approach

Based on the implementation and test results, this study proposed page segmentation and compression method that integrates some MATLAB and Python functions which are based on morphological dilation, CC Labeling, and CC height, width and area analysis algorithms. The procedure of the proposed page layout segmentation and compression techniques is presented in figure 4.27.

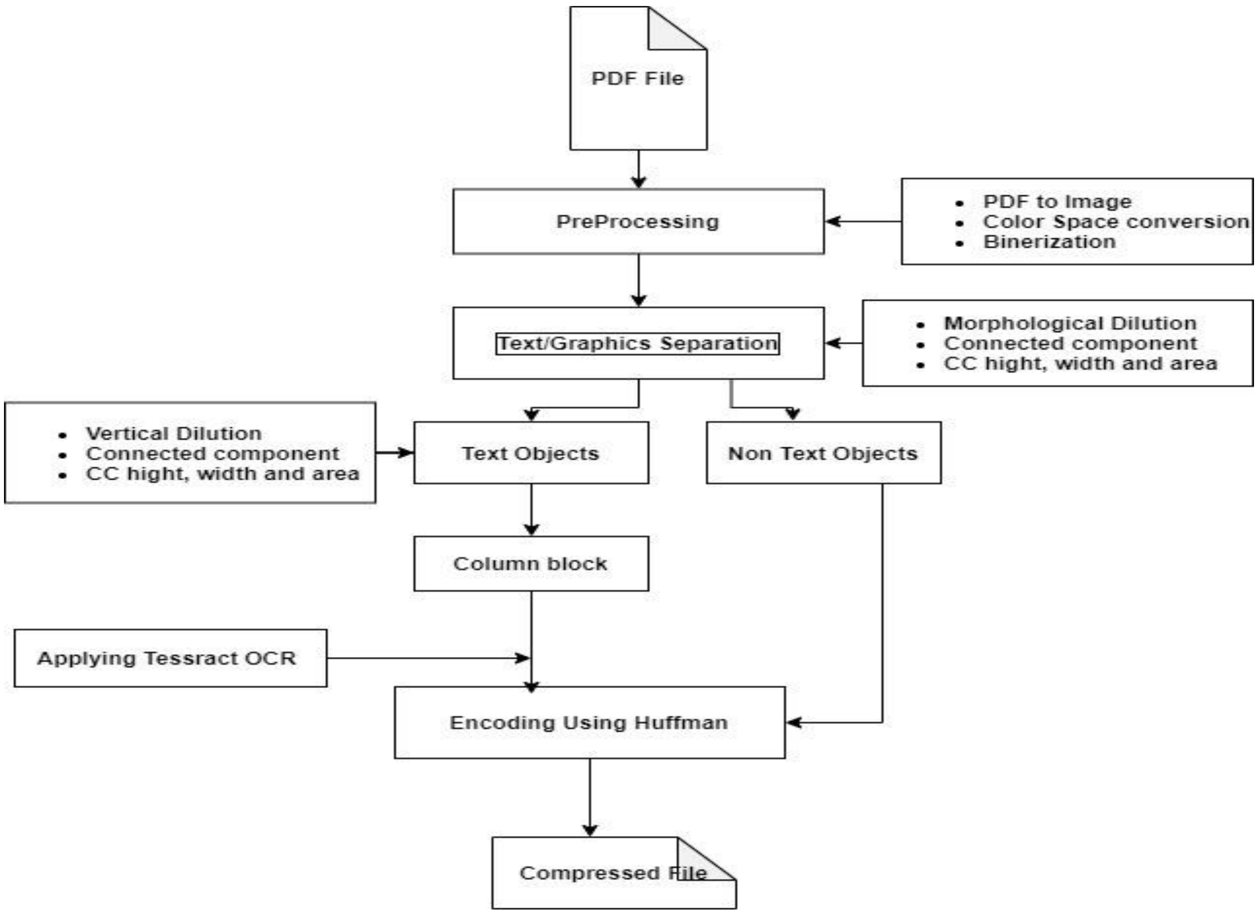


Figure 4-29 The proposed page segmentation & compression approach for Amharic PDF documents

We experimented the proposed approach on different PDF documents data sets containing in total of about 12 different PDF files they contain different objects inside. Performances of the approach were evaluated by computing the compression ratio compering with existing commercial compression tools. Table 4.7 shows the compression performance of the proposed approach with compared to existing popular compression systems.

Table 4-9: Performance of proposed approach compared with existing popular compression systems

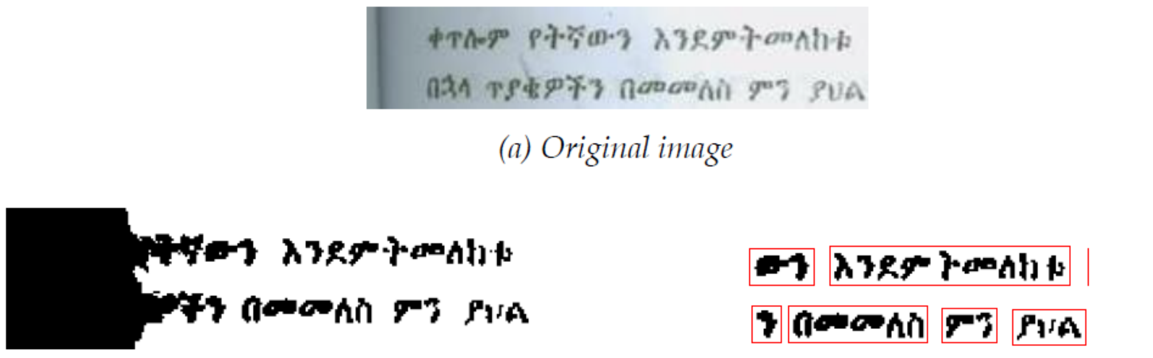
File Name	**Original Size(mb)**	**Compression Using 7Z**		**Compression Using Zip**		**Compression Using GZip**		**Proposed Approach using Huffman**	
		Size After Compression(mb)	**Performance (%)**	**Size After Compression(mb)**	**Performance (%)**	**Size After Compression(mb)**	**Performance (%)**	**Size After Compression(mb)**	**Performance (%)**
PDF1	21.00	20.10	4.29	20.60	1.90	19.50	7.14	3	85.71
PDF2	1.30	1.10	15.38	1.30	0.00	1.10	15.38	0.5	61.54
PDF3	2.20	2.10	4.55	2.10	4.55	2.10	4.55	0.7	68.18
PDF4	4.60	4.50	2.17	4.40	4.35	4.60	0.00	0.9	80.43
PDF5	3.50	3.30	5.71	3.30	5.71	3.40	2.86	1.5	57.14
PDF6	2.10	2.10	0.00	2.00	4.76	2.00	4.76	1.1	47.62
PDF7	6.30	6.00	4.76	6.10	3.17	6.10	3.17	4.2	33.33
PDF8	3.70	3.50	5.41	3.60	2.70	3.60	2.70	1.2	67.57
PDF9	17.60	17.40	1.14	17.40	1.14	17.60	0.00	9.2	47.73
PDF10	5.60	5.50	1.79	5.40	3.57	5.30	5.36	3.2	42.86
PDF11	6.90	6.70	2.90	6.30	8.70	6.60	4.35	3.2	53.62
PDF12	4.40	4.10	6.82	4.30	2.27	4.30	2.27	1.3	70.45
Average	6.60	6.37	4.58	6.40	3.57	6.35	4.38	2.5	59.68

The experimental result shows that the proposed compression approach using Huffman shows better performance than existing commercial compression systems. From the experimented PDF files, the proposed compression approach is effective for extracted objects containing more textual objects.

4.8 Findings and challenges

This study attempts to compress a PDF documents by applying effective page layout segmentation in addition reconstructs column and paragraph blocks using the information stored during page segmentation stage. This study explored five segmentation techniques namely: morphological dilution, Horizontal Run Length Smoothing, watershed transforms, connected component labeling and Components Width, Height and Area Analysis and experiments on different real-life PDF documents. It performs better to identify textual and not-textual blocks.

In general, page layout reconstruction is depending on the page segmentation stage. One of the challenges in this study is the existence of black shade in scanned PDF documents. Dark shades often introduced to PDF document images while scanning. The shade hides part of text and connected to characters nearby in document images. While trying to remove such shade as background by increasing the threshold value it highly degrades features of characters in the image. Figure 4.29 below shows the effect of the existence of black shade thresholding and segmentation.



(a) Original image

(b) Effect of thresholding *(c)Final effect on segmentation*

Figure 4-30 The Effect of Thresholding in the Presence of Shadow

Although the technique proposed in this study performs well in column block segmentation. However, it segments text with large font sizes such as page titles and heading as part of images because it mainly focuses on size to identify images, graphics and columns.

This study applies LZW and Huffman compression algorithm to compress the segmented and extracted objects, the given PDF document contains more textual files inside it performs better compression result.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

In this study an attempt is made to minimize the size of the Amharic PDF files which is a gap observed in current popular compression software's. For this purpose, page layout segmentation is developed to detect and store information of text and non-text areas as well as text columns. Various researches for non-English languages have been conducted in the course of developing textual image compressions.

Data Compression is particularly useful in communication because it enables devices to transmit or store the same amount of data in fewer bits. Data Compression is also widely used in File Storage and Distributed Systems, Backup utilities, Spreadsheet applications and Database Management Systems. This thesis is also to add an approach that improved and optimal data compression that will make the use of computers more effective in maximizing memory space and data transmission bandwidth.

5.1 Conclusions

The main objective of this study is compressing the content of the Amharic PDF files by applying effective page segmentation and extraction technique that is capable of identifying non-text objects, text column and text blocks of a page in real life Amharic PDF documents. Towards achieving this goal, recursive page layout segmentation is performed to detect, and store object information that is captured from the PDF files.

To simplify compression of Amharic PDF file, the proposed approach is separating text from non-text regions. This is done by applying page layout segmentation techniques; namely connected component with Dilation and connected components Area, Height and width analysis Based on the experiment on the average 78.3% accuracy rate is achieved by the proposed approach. also, page column segmentation is done using vertical dilution followed by connected component analysis. The proposed technique accurately identified column layout with an accuracy rate of 89.1%, thereby all coordinate information' about column block is stored for reconstructing stage.

Finally, the extracted objects are compressed using LZW, Huffman and RLE compression algorithms. The proposed approach experimented on different PDF documents and compresses the extracted objects with compression ratio of less than 50%, which is better compression result than existing commercial existing compression tools such as ZIP, 7Z and winRAR.

The proposed approach also capable of reconstructing the compressed data after decompression. Based on the stored layout coordinate information the original document image non text blocks and textual columns reconstructed on the average with 74% accuracy rates. From correctly segmented block the proposed techniques 92% accuracy rates. However, reconstructs equal font size color of textual contents which is font size equals to 12 and block color even the sentence is heading or page titles. Because the reconstruction process only focuses on the coordinate information stored on segmented layout blocks, it doesn’t reconstruct in terms of the font size and color.

The major challenges that the proposed approach faces include segmenting PDF documents with a large number of graphics and also detecting paragraph blocks when every lines of a paragraph have equal length, a large gap between two consecutive paragraphs, the presence of tiny pixels on a whitespace that separate two different column blocks. In addition, the proposed approach captured large font sizes considers them as image regions.

5.2 Recommendations

This Study tried to segment PDF layout and attempt to compress internal content of Amharic PDF files. Based on the finding of the study, the following recommendations are formulated to future research direction.

- Real life PDF documents have different physical and logical layouts such as table, graphics, header, footer, etc. Hence, future studies can explore on table and other layouts of real-life PDF document preservation.
- The page segmentation technique segments text with large font sizes as part of images or graphics because it mainly focuses on size (Area, Height and Width) to identify images, graphics, tables, etc. Therefore, further researches to improve segmentation need to be conducted.

- Developing adaptive page segmentation algorithm which can identify different blocks of a page intelligently should be one of the future research area to consider.
- This study reconstructs equal font size color of textual contents which is font size equals to 12 and block color even the sentence is heading or page tiles. Thus, reconstructing text font size and font color based on its information should be one of future research direction to consider.
- PDF Document images might have overlapping columns. Thus, developing page column segmentation algorithm that can handle columns with different width and overlapping columns is a future research area need to consider.
- The proposed study adopts google tesseract OCR for Amharic textual documents recognition; However, the recognition accuracy depends on the quality of document future researches also need to explore on a better recognition algorithm in the course of developing applicable Amharic OCR.
- Skew detection and correction are not included in the proposed study, Thus, to increase the segmentation and recognition accuracy there is a need to apply skew detection and correction.

References

[1] G. M. and D. Bryant, *Data Compression*, Fourth Edi. California State University: British Library Cataloguing in Publication Data, 2006.

[2] Adobe Reader [Online]. Available: https://acrobat.adobe.com/us/en/acrobat/about-adobe-pdf.html.

[3] A. Nazemi, I. Murray, and D. A. Mcmeekin, "Layout Analysis for Scanned PDF and Transformation to the Structured PDF Suitable for Vocalization and Navigation," *Internnational Journnal Hum. Comput. Interact.*, vol. 7, no. 1, pp. 162–171, 2014.

[4] P. Laptev, "Method for Effective PDF Files Manipulation Detection," University Of Tartu, 2017.

[5] J. Mtimet and H. Amiri, "Arabic textual images compression approach," *Procedia Comput. Sci.*, vol. 35, pp. 118–126, 2014.

[6] W. J. Teahan, R. Mcnab, and I. H. Witten, "A Compression-based Algorithm for Chinese Word Segmentation," *Int. J. Appl. Sci. Eng.*, 2000.

[7] S. V Khangar and L. G. Malik, "Compression Method for Handwritten Document Images in Devnagri Script," vol. 3, no. 3, pp. 4305–4309, 2012.

[8] F. Informatik and D. Keysers, "Document Layout Analysis," *Int. Conf. Pattern Recognit.*, vol. 6, no. September 2012.

[9] J. Mtimet and H. Amiri, "Arabic textual images compression approach," *Procedia Comput. Sci.*, vol. 35, pp. 118–126, 2014.

[10] U. Garain, M. P. Chakraborty, and B. Chanda, "Lossless compression of textual images: A study on indic script documents," *Proc. - Int. Conf. Pattern Recognit.*, vol. 3, no. September 2014, pp. 806–809, 2006.

[11] A. S. Incorporated, *PDF Reference*, 3rd ed. New York: Adobe Portable Document Format, 2000.

[12] M. Rossi, W. Hui, and J. Bragge, "The design science research process : A model for producing and presenting information systems research," *J. Manag. Inf. Syst.*, no. February, 2006.

[13] C. R. Kothari, "Research Methodology: Methods and Techniques," in *IEEE Transactions on Knowledge and Data Engineering (TKDE).*, 2004, pp. 1–4.

[14] S. R. Kodituwakku and U. S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms," *Indian J. Comput. Sci. Eng.*, vol. 1, no. 4, pp. 416–425, 2001.

[15] B. Souley, P. Das, and S. Tanko, "A Comparative Analysis of Data Compression Techniques," *Int. J. Appl. Sci. Eng.*, vol. 2, no. 10, pp. 63–82, 2014.

[16] S. K. Chachra, Z. Xue, S. Antani, D. Demner-fushman, and G. R. Thoma, “Extraction and Labeling High-resolution Images from PDF Documents,” *J. Comput. Sci. Eng.*, 2009.

[17] D. Salomon, *Data Compression The Complete Reference FourthEdition*, vol. 53, no. 9. 2007.

[18] I. Akman, H. Bayindir, S. Ozleme, Z. Akin, and S. Misra, “Lossless Text Compression Technique Using Syllable Based Morphology,” *Int. Arab J. Inf. Technol.*, no. January, 2011.

[19] D. Measures, “Lossy Compression Algorithms,” *J. Comput. Sci. Eng.*, 2003.

[20] C. E. SHANNON, “A Mathematical Theory of Communication,” *Bell Syst. Tech. J.*, vol. 27, no. April 1928, pp. 379–423, 1948.

[21] D. HUFFMAN, “A Method for the Construction of Minimum-Redundancy Codes,” *Int. Work. Doc. Anal. Syst.*, 1948.

[22] J. Ziv and A. Lempel, “A Universal Algorithm for Sequential Data Compression,” vol. I, no. 3, 1977.

[23] Z. M. Alasmer, B. M. Zahran, B. A. Ayyoub, and M. A. Kanan, “A Comparison between English and Arabic Text Compression,” vol. 6, no. 3, pp. 111–119, 2013.

[24] D. Measures, “Chapter 8 Lossy Compression Algorithms,” 2003.

[25] R. Dqg W R, “Discrete Cosine Transform,” in *Discrete Cosine Transform*, 2006.

[26] H. Jensen, “Sign, Symbol and Script: An Account of Man’s Efforts to Write,” in *Sign, Symbol and Script: An Account of Man’s Efforts to Write*, 3rd ed., G. A. and U. Ltd, Ed. London, England.: Allen and Unwin Ltd, 1969.

[27] A. Feren, “Writing and Literature in Classical Ethiopic (Ge’ez),” in *Literatures in African Languages*, 1985.

[28] S. Ager, “Ge’ez script.” [Online]. Available on: http://www.omniglot.com/writing.html

[29] H. Hi, “Typing in Amharic,” in *Typing in Amharic*, 1998.

[30] A. S. Incorporated, *PDF Reference*, 3rd ed. New York: Adobe Portable Document Format, 2000.

[31] C. Stahl, S. Young, D. Herrmannova, R. Patton, and J. Wells, “DeepPDF : A Deep Learning Approach to Analyzing PDFs,” *Int. Conf. Comput. Lin- guistics*, vol. 4, 2014.

[32] C. Paper, H. Chao, S. Alliance, and J. Fan, “Layout and Content Extraction for PDF Documents Layout and Content Extraction for PDF Documents,” *Hewlett-Packard Lab.*, no. May, 2014.

[33] T. Tran, I. Na, and S. Kim, “Separation of Text and Non-text in Document Layout Analysis using a Recursive Filter,” vol. 9, no. 10, pp. 4072–4091, 2015.

[34] T. Pavlidis and J. Zhou, “Page segmentation and classification,” *CVGIP Graph. Model. Image Process.*, vol. 54, no. 6, pp. 484–496, 1992.

[35] E. Haneda, S. Member, and C. A. Bouman, “Text Segmentation for MRC Document Compression Text Segmentation for MRC Document Compression,” *Int. Res. J. Eng. Technol.*, 2012.

[36] C. Science and M. Meshesha, “Recognition and Retrieval from Document Image Collections Thesis submitted in partial ful llment of the requirements for the degree of International Institute of Information Technology,” no. August, 2008.

[37] M. Javed, P. Nagabhushan, and B. B. Chaudhuri, “Extraction of Line Word Character Segments Directly from Run Length Compressed Printed Text Documents,” *Comput. Vis. Pattern Recognit.*, no. March, 2014.

[38] and K. Skarbek, W., Koschan, A., Bericht, T., Veroffentlichung, Z., “Color Image Segmentation: A Survey,” 1994.

[39] R Cattoni et, “Geometric Layout Analysis Techniques for Document Image Understanding: a Review.,” *ICT-IRST Trento, Italy,*.

[40] K. K., “Recognition. Analysis and Retrieval of Historical Document Images.,” Universite Paris Descarte, Paris .

[41] T. Randen and H. John, “Segmentation of text / image documents using texture approaches 1 Introduction,” in *International Conference on Acoustics, Speech & Signal Processing*, 1999.

[42] 2006., “Algorithms for Image Segmentation,” Birla Institute of Technology and Science.

[43] A. O. Shigarov and R. K. Fedorov, “Simple Algorithm Page Layout Analysis,” *Pattern Recognit. Image Anal.*, vol. 21, no. 2, pp. 324–327, 2011.

[44] F. Shafait, D. Keysers, and T. M. Breuel, “Performance comparison of six algorithms for page segmentation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3872 LNCS, pp. 368–379, 2006.

[45] J. L. Fisher, S. C. Hinds, and D. P. D’Amato, *A Rule-based System for Document Image Segmentation*, vol. 1. 1990.

[46] S. Mandal, S. P. Chowdhury, A. K. Das, and B. Chanda, “Automated Detection and Segmentation of Table of Contents Page from Document Images,” in *International Conference on Document Analysis and Recognition*, 2003, no. Icdar.

[47] K. Schwenk and F. Huber, “Connected Component Labeling algorithm for very complex and high-resolution images on an FPGA platform,” *High-Performance Comput. Remote Sens. V*, vol. 9646, p. 964603, 2015.

[48] K. Kise, “Segmentation of page images using the area Voronoi diagram.,” *Comput. Vis. Image Understanding,* vol. 70(3):, pp. 7370–382.

[49] L. O‘Gorman., “The document spectrum for page layout analysis.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, 1993.

[50] H. Grailu, M. Lotfizad, and H. Sadoghi-Yazdi, “Farsi and arabic document images lossy compression based on the mixed raster content model,” *Int. J. Doc. Anal. Recognit.*, vol. 12, no. 4, pp. 227–248, 2009.

[51] Y. Wiseman and I. Gefner, “Conjugation-based compression for Hebrew texts,” *ACM Trans. Asian Lang. Inf. Process.*, vol. 6, no. 1, pp. 1–10, 2007.

[52] N. Zhang and J. M. Danskin, “A pattern-based lossy compression scheme for document images,” *Int. J. Intell. Comput. Inf. Sci.*, vol. 8, no. June, pp. 221–233, 1996.

[53] A. Ford and A. Roberts, *Colour Space Conversions*, vol. 1998. 1998.

[54] R. Sharma and R. Sharma, “Image Segmentation Using Morphological Operation for Automatic Region Growing,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 6, pp. 5686–5692, 2014.

[55] P. Vartak, V. Mankar, A. Prof, and H. V. P. M. College, “Morphological Image Segmentation Analysis,” *Int. J. Comput. Sci. Appl.*, vol. 6, no. 2, pp. 161–165, 2013.

[56] S. Ruparelia, “Implementation of Watershed Based Image Segmentation Algorithm in FPGA,” Stuttgart University, 2012.

[57] A. P. Borlepwar, S. R. Borakhade, and M. S. B. Pradhan, “Run Length Smoothing Algorithm,” *Int. J. Adv. Res. Sci. Eng.*, vol. 06, no. 09, pp. 766–771, 2017.

[58] C. Boiangiu, C. Avatavului, M. Prodan, and I. I. Bucur, “Combining Tessract and Asprise to Improve OCR Text,” no. June, 2019.

[59] D. HUFFMAN, “A Method for the Construction of Minimum-Redundancy Codes *,” *Int. Work. Doc. Anal. Syst.*, 1948.

[60] Y. Qinghui, N. Xiujun, and Y. Qingfei, “Implementation of LZW Data Lossless Compression Algorithm Based on VB,” no. Iccse, pp. 37–44, 2016.

[61] A. Ayidagne, “Page Column and Paragraph Layouts Segmentation and Reconstruction for Recognizing Real Life Documents,” Addis Abeba University, 2016.

Appendix I: Amharic Characters

Geéz	Kaéb	Salis	Rabé	Hamis	Sadis	Sabé
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
ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ
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

Appendix II: Sample Codes

PDF to Image

```
import os
from wand.image import Image
filename = "PDF Path"
OUTPUT_FOLDER="Out put Path"
with(Image(filename = filename, resolution=300)) as source:
    images = source.sequence
    pages = len(images)
    for i in range(pages):
        n = i + 1
        newfilename = str(n) + '.jpeg'
        Image(images[i]). save(filename=OUTPUT_FOLDER+newfilename)
```

Color Space Change to RGB

```
#!/usr/bin/python
from PIL import Image
import os, sys
path = "Image Path
dirs = os.listdir( path )
def torgb():
    dirs = os.listdir(path)
    for item in dirs:
        if item == '.jpeg':
            continue
        if os.path.isfile(path+item):
            im = Image.open(path+item)
            f, e = os.path.splitext(path+item)
            image = im.convert('RGB')
            image.save(f + '.jpeg')

torgb()
```

Pre-Processing Process

```
% Get the dimensions of the image.
[rows, columns, numberOfColorChannels] = size(rgbImage);
% Enlarge figure to full screen.
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0 0.05 1 0.95]);
% Get a gray scale version
grayImage = rgb2gray(rgbImage);
% Get a mask of the colored images, which have high range.
level = graythresh (grayImage);
binaryImage =im2bw (I, level)
% Do an opening to break connections of text to photo.
binaryImage = imopen(binaryImage, ones(10));
% Fill in holes.
binaryImage = imfill(binaryImage, 'holes');
% Filter out blobs smaller than 50 pixels by 50 pixels.
```

Dilation function

```
function [dialatedIm] = Dilat(bw,tresh)
```

```
%Apply dilation using bwdist() with a given threshold
dialatedIm = bwdist(~bw) >= tresh;
```

Connected components function

```
function [cc,num] = ConnectedComp(bw)
%Extracting connected components to variable cc %using 4 connictivity
cc = bwconncomp(bw,4)
%storing number of connected components num=cc.NumObjects;
```

Extracting Images using bounding Box

```
% removing objects containing fewer than 100 pixels using bwareaopen function.
BW2 = bwareaopen(photoRegion, 100);
% find both black and white regions
stats = [regionprops(BW2)];
% show the image and draw the detected rectangles on it
imshow(BW2);
label=bwlabel(BW2); % Finds out the independent objects and labels them.
max(max(label)); %maximum objects detected.
mkdir([new path]);
pagefolderImages=[newpath2];

measurements = regionprops(photoRegion, 'Area', 'BoundingBox');

allAreas = [measurements.Area];
% Crop out each word
for blob = 1 : length(measurements)
  % Get the bounding box.
  thisBoundingBox = measurements(blob).BoundingBox;
  % Crop it out of the original gray scale image.
  thisWord = imcrop(rgbImage, thisBoundingBox);
  x=fix(thisBoundingBox(1));
  y=fix(thisBoundingBox(2));
  FileName = sprintf('%d, %d.jpg', x, y);
  fullFileName = fullfile(pagefolderImages, FileName);
  imwrite(thisWord, fullFileName);
end
```

Extracting Text Areas using bounding Box

```
% removing objects containing fewer than 100 pixels using bwareaopen function.
BW3 = bwareaopen(textRegions, 100);
% find both black and white regions
stats2 = [regionprops(BW3);regionprops(not(BW3))];
% show the image and draw the detected rectangles on it
imshow(BW3);
Textlabel=bwlabel(BW3); % Finds out the independent objects and labels them.
max(max(Textlabel)); % This Command gives us the maximum objects detected.
mkdir([textpath']);
pagefolderColumens=[textpath'];
measurements2 = regionprops(BW3, 'Area', 'BoundingBox');
allAreas2 = [measurements2.Area];
% Crop out each word
for blob = 1 : length(measurements2)
  % Get the bounding box.
  thisBoundingBox = measurements2(blob).BoundingBox;
  % Crop it out of the original gray scale image.
  thisWord = imcrop(rgbImage, thisBoundingBox);
  x=fix(thisBoundingBox(1));
  y=fix(thisBoundingBox(2));
  FileName = sprintf('%d, %d.jpg', x, y);
  fullFileName = fullfile(pagefolderColumens, FileName);
  imwrite(thisWord, fullFileName);
end
```

OCR function For Textual Areas

```
def createImageText():
    dirs = len(os.walk(path1).__next__()[1]) # finding number of images
    for i in range(1,dirs+1):
        path = ("outputpath")
        def ocr(file_to_ocr):
            im = Image.open(path+"\\"+file_to_ocr)
            txt=pytesseract.image_to_string((im), lang="Amh")
            return txt

        file_list = os.listdir(path) # file names in list (not sorted)
        directory = os.path.join(path) # path for storing the text file

        # function to sort the file names in order of numerical value
        def atoi(text):
            return int(text) if text.isdigit() else text

        def natural_keys(text):

            return [ atoi(c) for c in re.split('(\d+)', text) ]

        file_list.sort(key=natural_keys)
        for file in file_list:
            name=os.path.splitext(file)[0]
            # selecting image file type
            if file.endswith(".jpg"):
                txt=ocr(file) # calling the ocr function
                file = open(directory+"\\"+name+".txt",'a+',encoding="utf-8")
                file.write(str(txt))
                os.remove(path+name+".jpg") # removing the image after it converted to text

#           if file.endswith(".jpg"):
        print("Image Conversion completed")

createImageText()
```

Compression function
Huffman compression

```
def compress(self):
                    filename, file_extension = os.path.splitext(self.path)
                    output_path = filename

                    with open(self.path, 'r+',encoding="utf8") as file, open(output_path, 'wb',) as output:
                            text = file.read()
```

```
                text = text.rstrip()

                frequency = self.make_frequency_dict(text)
                self.make_heap(frequency)
                self.merge_nodes()
                self.make_codes()

                encoded_text = self.get_encoded_text(text)
                padded_encoded_text = self.pad_encoded_text(encoded_text)

                b = self.get_byte_array(padded_encoded_text)
                output.write(bytes(b))

        print("Compressed")
        return output_path
```

LZW compression

```
for symbol in data:
    string_plus_symbol = string + symbol # get input symbol.
    if string_plus_symbol in dictionary:
        string = string_plus_symbol
    else:
        compressed_data.append(dictionary[string])
        if(len(dictionary) <= maximum_table_size):
            dictionary[string_plus_symbol] = dictionary_size
            dictionary_size += 1
        string = symbol

if string in dictionary:
    compressed_data.append(dictionary[string])

# storing the compressed string into a file (byte-wise).
out = input_file.split(".")[0]
output_file = open(out + ".hc", "wb")
for data in compressed_data:
    output_file.write(pack('>H',int(data)))

output_file.close()
file.close()
```

Reconstruction Function

```
def createImageText():
    dirs = len(os.walk(path).__next__()[1])
    for i in range(1,dirs+1):
        img = np.zeros([3300,2550,3],dtype=np.uint8)
        img.fill(255)
        cv2.imwrite(outpath + str(i) + '.jpeg',img)
        image = Image.open(outpath + str(i) + '.jpeg')
        filelistall = path + str(i) + '/Text/'
        dirs2 = os.listdir(filelistall)
        for item in dirs2:
            textfile=open(filelistall+item, "r", encoding="utf8")
            contents =textfile.read()
            font = ImageFont.truetype('path//nyala.ttf', 40)
            f, e = os.path.splitext(os.path.basename(filelistall+item))
            x, y = [x.strip() for x in f.split(',')]
            x2 = int(x)
            y2 = int(y)
            image = Image.open(outpath + str(i) + '.jpeg')
            d = ImageDraw.Draw(image)
            location = (x2, y2)
            text_color = (100, 100, 200)
            d.text(location, contents, font= font, fill=text_color)
            #image.paste(croped, (x2, y2))
            image.save(outpath + str(i) + '.jpeg', quality=100)
createImageText()
def createImage():
    dirs = len(os.walk(path).__next__()[1])
    for i in range(1,dirs+1):
        image = Image.open(outpath + str(i) + '.jpeg')
        filelistall = path + str(i) + '/Images/'
        size2 = len([f for f in os.listdir(filelistall)if os.path.isfile(os.path.join(filelistall, f))])
        if size2==0:
            print("");
        else:
            dirs2 = os.listdir(filelistall)
            for item in dirs2:
                croped = Image.open(filelistall+item)
                f, e = os.path.splitext(os.path.basename(filelistall+item))
                x, y = [x.strip() for x in f.split(',')]
                x2 = int(x)
                y2 = int(y)
                image = Image.open(outpath + str(i) + '.jpeg')
                image.paste(croped, (x2, y2))
                image.save(outpath + str(i) + '.jpeg', quality=100)
createImage()
```