A Mathematical Expression Detection and Segmentation Module for Enhanced Document Layout Analysis

Jake Bruce

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Supervisor: Prof. Lynn Abbott, Ph.D.
Co-Supervisors: Prof. Jason Xuan, Ph.D. and Prof. Michael Hsiao, Ph.D.

in January 2014
Blacksburg, VA

Keywords: document layout analysis, optical character recognition, mathematical expression detection and segmentation, document image, type-specific layout analysis

Abstract

Various document layout analysis techniques are employed in order to enhance the accuracy of optical character recognition (OCR) in document images. Type-specific document layout analysis involves localizing and segmenting specific zones in an image so that they may be recognized by specialized OCR modules. Zones of interest include titles, headers/footers, paragraphs, images, mathematical expressions, chemical equations, musical notations, tables, circuit diagrams, among others. False positive/negative detections, oversegmentations, and undersegmentations made during the detection and segmentation stage will confuse a specialized OCR system and thus may result in garbled, incoherent output. In this work a mathematical expression detection and segmentation (MEDS) module is implemented and then thoroughly evaluated. The module is fully integrated with the open source OCR software, Tesseract, and is designed to function as a component of it. Evaluation is carried out on freely available public domain images so that future and existing techniques may be objectively compared.

Acknowledgments

I would like to thank Dr. Abbott for his helpful insights, my labmates Sherin Fathy, Ahmed Ibrahim, Amira Youssef, Mahmoud Sobhy, and Sherin Ghannam for keeping me company, teaching me a thing or two about their rich culture, and letting me try some great foods. Last but not least I'd like to express my gratitude for my family and their unconditional support.

Table of contents

1 Introduction 1
1.1 Enhancing Information Accessibility 1
1.2 Introduction to OCR and Document Analysis: A Brief History 5
1.3 Google Books Initiative 8
1.4 Contributions of this Thesis 10
1.5 Organization of Thesis 11

2 Literature Review 13
2.1 The Beginnings of OCR 13
2.1.1 Fixed-font 13
2.1.2 Omnifont 15
2.2 Pattern Recognition Techniques in OCR 16
2.2.1 Text Line Finding 17
2.2.2 Character Feature Extraction 21
2.2.3 Character Classification 28
2.2.4 Detection of Merged or Broken Characters 30
2.2.5 Word Recognition and Linguistic Analysis 31
2.3 Document Layout Analysis Techniques 31
2.3.1 Introduction to Document Layout Analysis 31
2.3.2 Preprocessing 35
2.3.3 Document Structure Analysis 37

3 Method 77
3.1 Introduction 77
3.1.1 Purpose 77
3.1.2 Problem Statement and Project Scope 78
3.1.3 Definitions and Acronyms 79
3.1.4 Tesseract Document Layout Analysis Framework Overview 79
3.1.5 Overview 81
3.2 System Overview 81
3.3 System Architechture 81
3.4 Component Design 84
3.4.1 Groundtruth Dataset Generation 84
3.4.2 MEDS Module 90
3.4.3 Evaluation Module 126

4 Experimental Results 135
4.1 Detector Parameter Selection and Training 135
4.2 Final Evaluation 137

5 Conclusion and Future Work 148

Bibliography 150

1 Introduction

Basically, our goal is to organize the world's information and to make it universally accessible and useful.

Larry Page – Co-founder of Google

1.1 Enhancing Information Accessibility

Never, since the invention of the printing press, has society seen such a radical change in its means of information distribution. Armed with powerful search engines roaming the vast expanse of the World Wide Web, nearly everyone in the world has, at their very fingertips, access to archives full of information. This enhanced information accessibility is having profound implications for society and could lead to a fruitful age of enlightenment.

The global effects of high speed Internet access are seen daily as hundreds of millions browse for information/multimedia, look up map directions, interact through email/social networks/video games, shop remotely, video chat, etc. Corporations like Google, Microsoft, Facebook, eBay, and Amazon continue building and extending the capacity of their server farms as the growth of user demand shows no signs of slowing down. By mid-2012, it was reported that nearly an eighth of the world's population was on the popular social networking site, Facebook [1]. As such figures continue to grow, studies are showing that technology is even affecting the manner in which we think and behave at the most fundamental levels. Whether or not the long-term effects of this relatively nascent medium of interaction prove to be largely positive or negative remains to be seen. One remaining certainty, however, is that continuing innovation is, for better or for worse, altering the manner in which we live out our daily lives.

It was Benjamin Franklin who once said that “genius without education is like silver in the mine.” One would be hard-pressed in arguing that, throughout history, all people have been able to realize their full potential to succeed and make a difference in the world. If that were true, many would argue that our knowledge would, by now, have long since surpassed its current state. In fact it was just under five hundred years ago, that Europeans were finally emerging from an age of intellectual darkness which had lasted for roughly a millennium. If we look back to the spread of knowledge throughout written history, starting from the earliest true writing systems developed in ancient Egypt/Mesopotamia circa 3000 BC to the origins of philosophy, math, science, and theater in ancient Greece, all the way to the birth of the “modern era” which

culminated itself in the scientific revolution of the sixteenth century AD, we notice a general trend of small bursts of knowledge spreading repeatedly, each time with greater strength than before, each one improving upon on its predecessor. Sir Isaac Newton exemplified this trait of humanity with his statement that “if I have seen further, it is by standing on the shoulders of giants.”

Although much of what defines us from a cultural perspective may indeed be passed from generation to generation through word of mouth, our tremendous advancements in math, science, art, and literature since the dawn of the modern era can be largely attributed to Johannes Gutenberg's invention of the printing press, which made mass distribution of books possible in Late-Medieval Europe. Prior to this key event in history, the stage was set in Europe for an age of scientific inquiry and revelation when the religious leader, Thomas Aquinas, embraced the separation between the purely theological and purely scientific schools of thought. Also of vital importance was the translation and recurrence of ancient Greek writings which had been studied and further developed by Arabic scholars. The first universities built in Medieval Europe were initially centered around classical Greek and religious studies and helped to lead Europe out of its age of darkness. This collaborative environment of scholastic endeavor helped set the framework for an age of enlightenment which would move humanity a step forward. Archaic ideas such as bloodletting were soon supplanted by discoveries leading to modern medicine and the commonly held geocentric model of our earth was replaced by a heliocentric one. Major breakthroughs were made in every field to foster the spread of knowledge which took society to where it is today. Without this ideal of scientific thinking combined with the means to distribute information, society would have never seen such tremendous improvements.

Moving forward to the present day, society has recently made technological breakthroughs which make the world's knowledge and information more accessible than ever before. In fact, many have suggested that the widely used search engine, Google, will go down in history as rivaling in importance with Gutenberg's printing press. It was only about a decade and a half ago that two Stanford Ph.D. students decided that they would like to take a shot at downloading and categorizing the entire internet. These two graduate students are of course the founders of Google [2], a now successful multinational corporation which, during the late nineties, left its search engine competitors far behind. Google is unique in that its employees facilitate a diverse range of interesting projects ranging from cataloging the human genome,

building autonomous vehicles, developing smart homes of the future, to developing augmented reality eye glasses, among many others. It is, however, in Google's core mission of finding ways to make the world's information “more universally accessible and useful,” that the company has had its greatest impact on the world at large. It was in keeping with this mission that, in 2005, in collaboration with HP Labs and the Information Science and Research Institute at UNLV, Google revived and open sourced an optical character recognition engine that had been developed as a Ph.D. project for HP Labs between 1985 to 1995. Although optical character recognition (OCR), the autonomous conversion of printed documents into digital formats, is a very mature area of research [3], development in this area continues in order to increase recognition support for the broad spectrum of languages, formats, and subject matter of printed documents. HP's OCR engine, named “Tesseract,” had proven itself as one of the industry's leading engines during UNLV's Fourth Annual Test of OCR Accuracy [4]. Eventually, however, HP subsequently went out of the OCR business, leaving the software to basically collect dust for about a decade.

Meanwhile, by around 2004, Google had begun its Google Books Initiative [5], a large-scale library digitization project. This initiative began with the lofty goal of digitizing all of the world's printed documents such that they may be indexed and searched online. By around 2005, Google hired Ray Smith, the former lead developer of Tesseract, to return to his long-abandoned, yet ground-breaking, Ph.D. work and also brought Tesseract into the open source domain. In so doing, Google helped to spur further research interest into efficient and accurate document recognition[1]. In the roughly eight years since the project was revived, support has been added for recognition of over fifty languages. Advanced page layout analysis techniques have been implemented in order to detect various types of documents ranging from novels, magazines, newspapers, images, textbooks, sheet music, etc. Language and script detection modules have also been implemented in order to autonomously determine what processing should be carried out for any given world document [6]. If Google's endeavor is successful, then the resulting implications to society will be extraordinary, possibly similar to the impact that Arabic scholars had on Europe when sharing and translating ancient Greek literature. If Google is successful in the autonomous digitization and recognition of any printed document regardless of its origin, then it will not be long before information from all of the world's documents become instantly

[1] The term, recognition, is herein used to describe a machine's extraction of a document's contents. This requires both the document page layout analysis as well as algorithms which subsequently convert the page layout contents into a machine-understandable form. The field of document layout analysis is further discussed in Section 1.2.

accessible in every language and to everyone around the world. Such a development would certainly speed up the world's already significant progress toward an era of far greater enlightenment and wisdom than has yet been seen.

The autonomous recognition of all printed documents would not only expedite the global advancement of knowledge and wisdom, but would also have tremendous implications toward every individual in society. Such a breakthrough would be especially significant toward the endeavor of Assistive Technology. With many devices being developed and studies being carried out on ways to enhance human computer interaction (HCI) for visually or physically handicapped individuals, digital access to all printed documents could make finding information, not only more convenient, but also possible for many who would not otherwise have access. Global autonomous document recognition could also help open the doors toward breaking down language barriers in information accessibility.

As research and development continues to enhance the accurate translation of discourse between various languages [7], the successful recognition of printed documents could eventually allow them to be machine-translated according to the language preference of a given user. With instant access to all of the world's information, regardless of its language or origin, at one's disposal, collaboration and learning among individuals across the world will be significantly enhanced. All people in the world regardless of their language preference, geographical location, and physical ability will have access to the world's stores of knowledge, and the opportunity to have a profound impact on society through the medium of the World Wide Web. Enhanced document analysis and recognition capabilities will make a significant contribution toward this end. The following section will discuss the background as well as some of the fundamental problems faced in the fields of document analysis and recognition.

1.2 Introduction to OCR and Document Analysis: A Brief History

From Herbert Shantz's *The History of OCR* [3], it is clear that the OCR of printed documents has been studied extensively over the last century. In one of the earliest OCR patents [8] (Figure 1), a mechanical apparatus was used to measure the incidence of light reflected back from a printed character when illuminated through a set of character templates. A character detection would occur when the light emitted from the template overlapped the character (assumed to be in dark print) sufficiently to prevent light from being reflected upon the medium. Despite requiring a significant amount of human intervention to ensure proper alignment and being largely inefficient

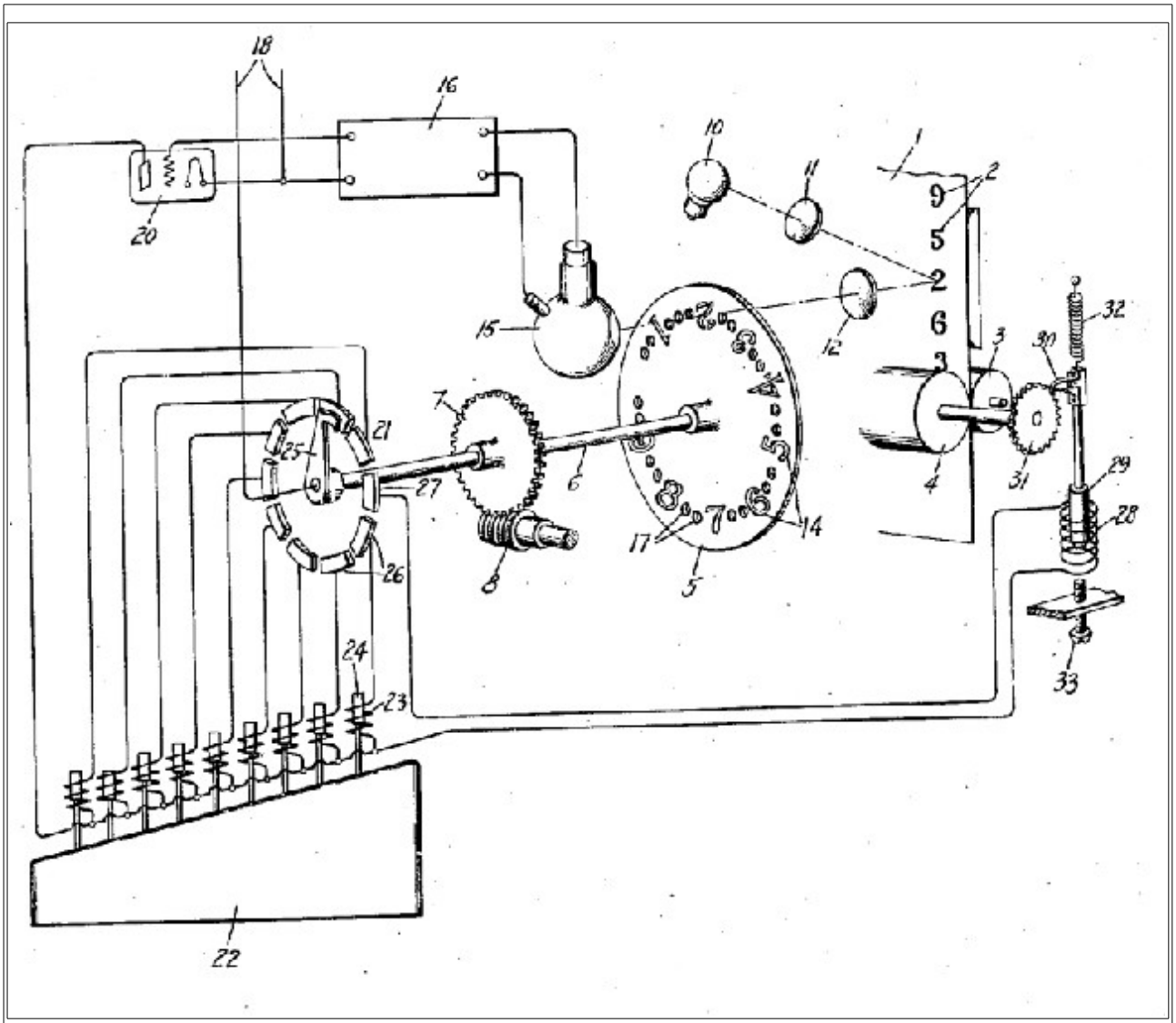


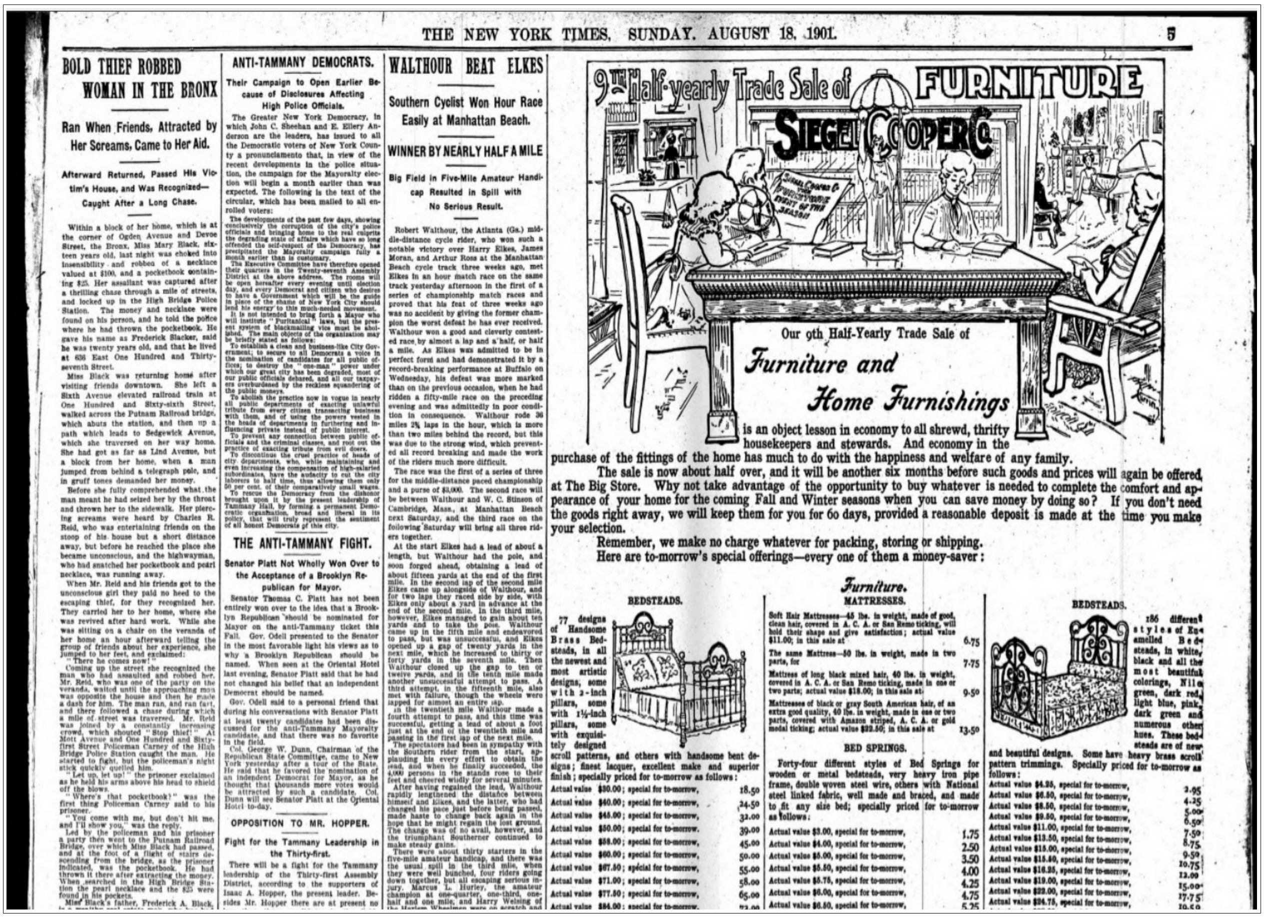
Figure 1: An illustration taken from the 1933 patent from Paul W. Handel, a former employee of General Electric, entitled “Statistical Machine” [8]. This is one of the earliest OCR devices ever invented.

at best, the fundamental ideas which motivated this early initiative are seen repeatedly throughout the century, and even now, albeit on a much larger scale.

Although some of the first commercial OCR systems were released during the 1950's, their applicability was limited in that, by and large, they were only capable of handling a single font type with very strict rules on character spacing. It was not until the mid-late 1970's, with the invention of both the charge-coupled device (CCD) flatbed scanner and the “Kurzweil Reading Machine” [9] that it became possible for a computer to read a variety of documents with reasonable accuracy. Although the training process for a particular font would take several hours and multi-column page layouts or images had to be specified by the user manually, Kurzweil's software showed significant improvement over the state-of-the-art technologies of the time.

In the 1980's, a company called Calera Recognition Systems [10] introduced an omnifont system that could read pages containing a mixture of fonts while also locating pictures and columns of text without any user intervention or extra training. The progress of the state-of-the-art in document recognition will be further discussed in the Chapter 2 Literature Review. More recent commercial OCR systems such as ABBYY FineReader [11], OmniPage Professional [12], and Readiris [13], are all quite accurate, not only in recognizing individual words or characters, but also in understanding and reproducing document layout structure. A magazine or newspaper page may, for instance, contain an intricate heading structure followed by multiple columns of text, pull-out quotes, in-set images, and/or graphs as demonstrated in the historical New York Times article shown in Figure 2 [14].

In order to understand and recognize content of such a document, it is essential to first carry out document layout analysis techniques which will determine how the document is partitioned. The text will be recognized with an understanding of where the columns of text are, which portions of text indicate headings or quotes, and which segments correspond to images, tables, captions, etc. If the text is not partitioned appropriately prior to recognition then the textual output will become unpredictable. With columns, paragraphs, or other structures merged together incorrectly, the text will lose much of its intended meaning and become far less readable to the human eye. For these reasons, sophisticated page layout analysis algorithms are of the utmost importance, not only for document recognition accuracy, but also in ensuring that the generated output is formatted correctly.


THE NEW YORK TIMES, SUNDAY, AUGUST 18, 1901.

BOLD THIEF ROBBED WOMAN IN THE BRONX

Ran When Friends, Attracted by Her Screams, Came to Her Aid.

Afterward Returned, Passed His Victim's House, and Was Recognized—Caught After a Long Chase.

ANTI-TAMMANY DEMOCRATS.

Their Campaign to Open Earlier Because of Disclosures Affecting High Police Officials.

THE ANTI-TAMMANY FIGHT.

Senator Platt Not Wholly Won Over to the Acceptance of a Brooklyn Republican for Mayor.

OPPOSITION TO MR. HOPPER.

Fight for the Tammany Leadership in the Thirty-first.

WALTHOUR BEAT ELKES

Southern Cyclist Won Hour Race Easily at Manhattan Beach.

WINNER BY NEARLY HALF A MILE

Big Field in Five-Mile Amateur Handicap Resulted in Spill with No Serious Result.

9th Half-yearly Trade Sale of FURNITURE

SIEGEL COOPER CO

Our 9th Half-Yearly Trade Sale of

Furniture and Home Furnishings

is an object lesson in economy to all shrewd, thrifty housekeepers and stewards. And economy in the purchase of the fittings of the home has much to do with the happiness and welfare of any family.

The sale is now about half over, and it will be another six months before such goods and prices will again be offered at The Big Store. Why not take advantage of the opportunity to buy whatever is needed to complete the comfort and appearance of your home for the coming Fall and Winter seasons when you can save money by doing so? If you don't need the goods right away, we will keep them for you for 60 days, provided a reasonable deposit is made at the time you make your selection.

Remember, we make no charge whatever for packing, storing or shipping.

Here are to-morrow's special offerings—every one of them a money-saver:

Furniture.

BEDSTEADS. MATTRESSES. BED SPRINGS. BEDSTEADS.

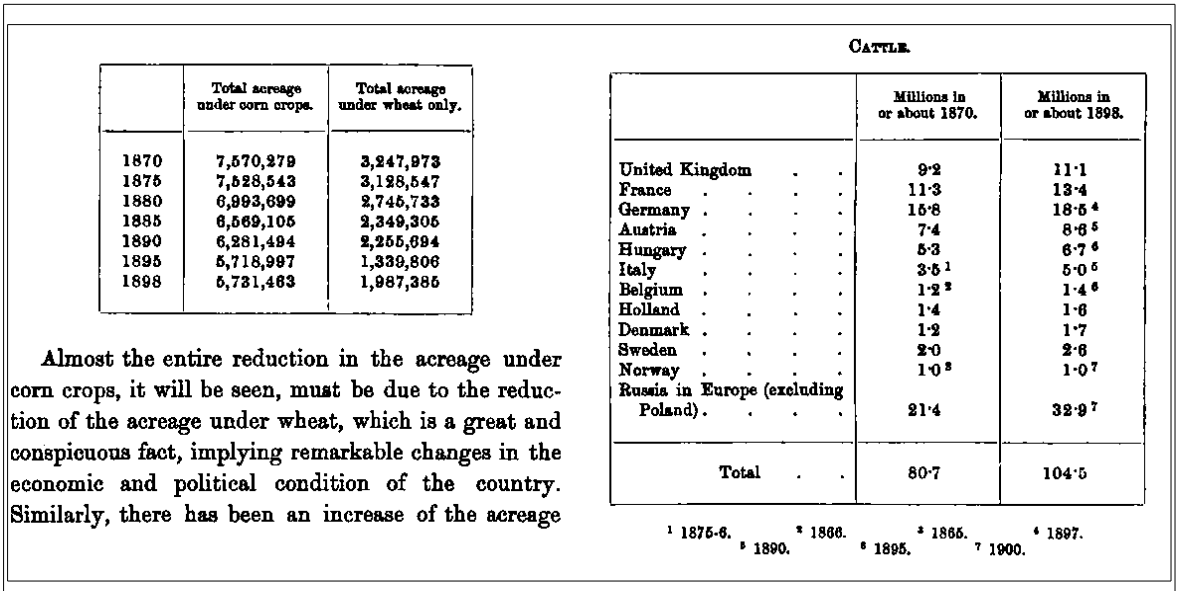
Figure 2: This excerpt from a 1901 New York Times article [14] was optically recognized by ABBY Fine Reader 8. The "New York Times" heading at the very top, the "Furniture and Home Furnishings" label embedded in the illustration, and the layout of the three columns at the bottom right were all incorrectly recognized by the commercial system.

Although most publishers keep digital copies of their more recent documents, there is also great demand for older documents which, unless they are digitized, will largely become forgotten by society. This would be unfortunate in that it is often surprising how pertinent older information and ideas can be. For companies such as Google who would like to make the world's information more readily available and accessible as well as to the Assistive Technology community, this is of the utmost importance. For this reason, a standard OCR output format called hOCR, which embeds OCR output within well-defined and widely available HTML and CSS structures has been put into place [15]. In order to ensure the quality of textual output generated by OCR for the wide variety of possible document layout structures, sophisticated document layout techniques are critical.

1.3 Google Books Initiative

There are various languages, dialects, and page layout formats for which Google's Tesseract software is being developed. Among them are mathematical equations, tables, graphs, and other figures which can be found in any standard science or math text book. While Smith's original work was optimized solely for the recognition of English newspaper formats, Google's continued efforts are aimed at recognizing page formats from a much broader scope [5]. Much of Google's ideas regarding document recognition are essentially in their infancy, and have a long way to go before being fully realized. Although an experimental equation detector has been added to the Tesseract software, its results, although showing significant promise, have been tested to have fairly limited accuracy. A table detector implemented by Google has also been tested on some sample images [16] (Figure 3) to show that, it too, could use significant improvement (Figure 4). Notice that, in the left-most table in Figure 4, the software failed to indicate the years as either belonging to the table or the normal text. They were simply disregarded. Also, the software was unable to determine where exactly the table boundaries are (which should be labeled green). In the right-most table, notice that although a better job was done, while the bottom portion of the text consists of footnotes, it is therein incorrectly labeled as part of the table. Also, the second line of all column labels are not recognized as part of the table when they clearly should be.

The problem of efficiently and accurately detecting equations, tables, graphs, and other figures for the broad spectrum of possible document types is certainly no easy one to solve. Although from a human's perspective, this problem may seem trivial, programming a machine to sum up a document with the same accuracy as the human eye proves to be a daunting task, as will be further discussed in the literature review chapter of this paper. As the inventors of Google continue to work toward their dream of creating an online “Library of Alexandria,” there is significant progress to be made before such a large-scale endeavor can be fully realized. The Google Book Search initiative has opened up many avenues for future research in document understanding and recognition, of which, this project is certainly one of the many to come.



	Total acreage under corn crops.	Total acreage under wheat only.
1870	7,570,279	3,247,973
1875	7,528,543	3,128,547
1880	6,993,699	2,745,733
1885	6,569,105	2,349,305
1890	6,281,494	2,255,694
1895	5,718,997	1,339,806
1898	5,731,463	1,987,385

Almost the entire reduction in the acreage under corn crops, it will be seen, must be due to the reduction of the acreage under wheat, which is a great and conspicuous fact, implying remarkable changes in the economic and political condition of the country. Similarly, there has been an increase of the acreage

CATTLE.

	Millions in or about 1870.	Millions in or about 1898.
United Kingdom	9·2	11·1
France	11·3	13·4
Germany	15·8	18·5 [4]
Austria	7·4	8·6 [5]
Hungary	5·3	6·7 [6]
Italy	3·5 [1]	5·0 [5]
Belgium	1·2 [2]	1·4 [6]
Holland	1·4	1·6
Denmark	1·2	1·7
Sweden	2·0	2·6
Norway	1·0 [3]	1·0 [7]
Russia in Europe (excluding Poland)	21·4	32·9 [7]
Total	80·7	104·5

[1] 1875-6. [2] 1866. [3] 1865. [4] 1897. [5] 1890. [6] 1895. [7] 1900.

Figure 3: The above text includes excerpts from two different pages taken from a scan of Sir Robert Griffin's Stastics textbook (circa 1913) [16]. On the left is a table followed by a paragraph of text, while on the right is a larger table. These images were extracted from a PDF which was made available online by Google.

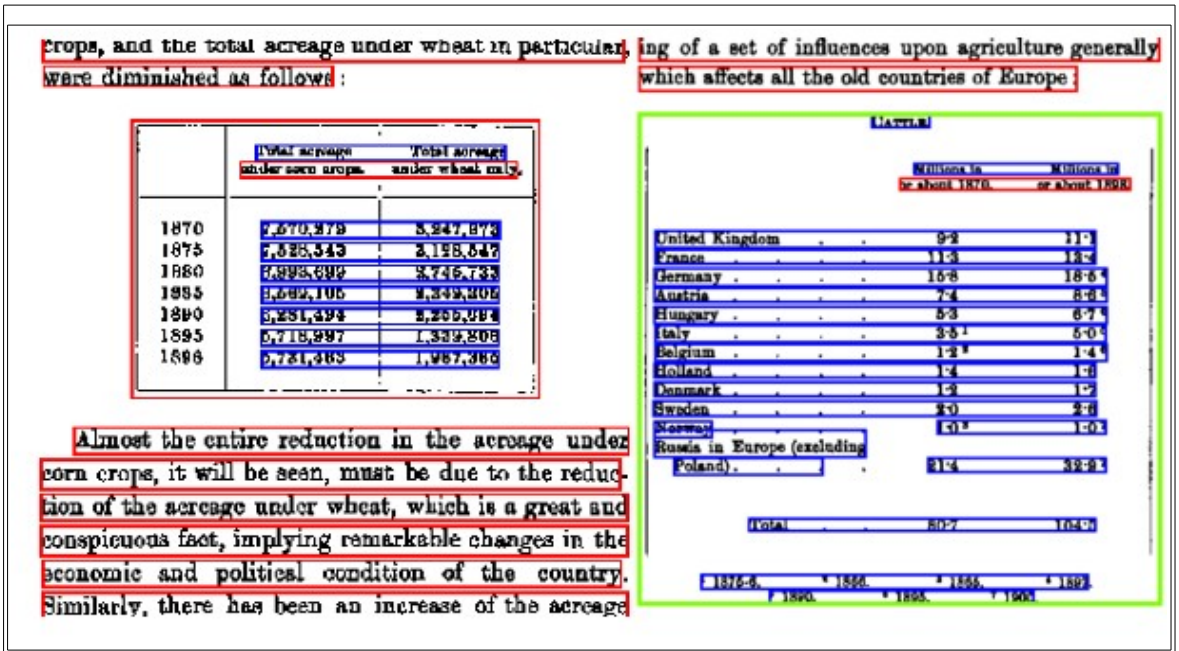


Figure 4: Above is the text from Figure 3 after having been labeled by Tesseract's table detection software. The text within the blue rectangles was identified as belonging to a table while the text within red rectangles was not. The green rectangle should encompass the entire table figure. As can be seen there are both false negatives and false positives.

1.4 Contributions of this Thesis

This thesis introduces a novel approach to mathematical expression detection and segmentation (MEDS) during the document layout analysis stage of OCR. The focus of this thesis is toward enhancing the OCR quality of printed scientific documents. The motivation for MEDS is illustrated by Figure 5. From Figure 5, it is clear that, when presented with mathematical expressions as input, an OCR system trained specifically for English will result in garbled output. With reliable MEDS, it becomes possible to prevent this mangled output from occurring, and also allows existing equation recognition algorithms, which have been extensively studied in the literature, to be provided a properly segmented input. While state-of-the-art mathematical symbol and structure recognition engines have been shown to attain near perfect accuracy on properly detected and segmented mathematical regions [17], their highly favorable results operate under the assumption that MEDS has already been carried out either automatically or manually with perfect accuracy. For the mathematical recognition studies observed in the literature, either manual or semi-automated techniques are used in order to properly isolate all mathematical expressions from normal text prior to any training or evaluation. The goal of this thesis is to produce and evaluate a purely automated system which carries out this functionality as a component within a larger document layout analysis framework, Tesseract [18].

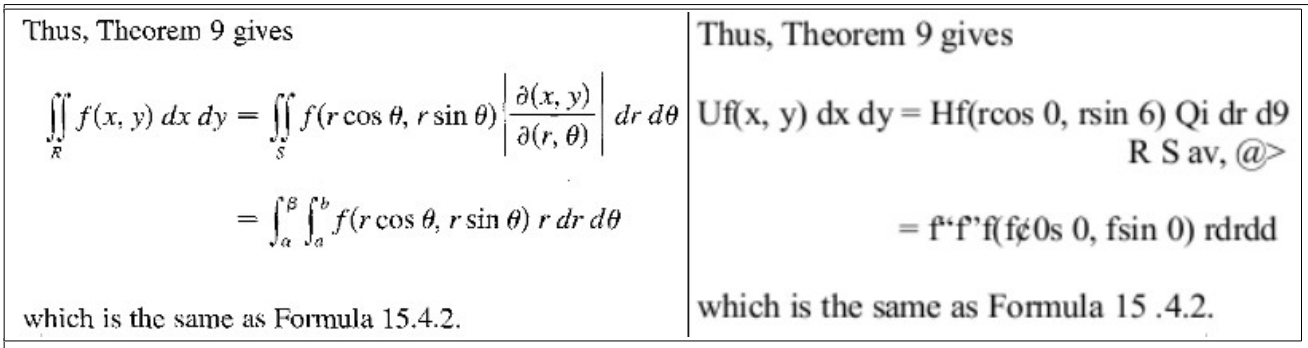


Figure 5: Example of OCR results on text excerpt. On the left is an example of text that was scanned at 300 dpi from a calculus text book. To the right is the output generated by the leading open source OCR engine, Tesseract.

By utilizing and interfacing with the existing data structures and algorithms present within Google's open source OCR engine, Tesseract, much of the more well-studied areas of OCR / document analysis research are surpassed so that a study of the relevant problem of MEDS can be explored in much greater detail than would be possible otherwise. As the Tesseract software, much like commercial state-of-the-art

systems, is capable of partitioning a document into columns, paragraphs, headings, etc., the software implemented in this work searches Tesseract's resulting partitions in order to detect regions of interest. Greater document understanding is accomplished through recognition of a variety of relevant features, many of which have yet to have been explored in existing research. Relevant features are subsequently combined into a binary classifier in order to detect regions of interest. These regions are then fed into a segmentation module whose aim is to properly combine the detection areas into properly segmented regions. The primary contributions of this work are briefly summarized below:

- A freely available ground-truth dataset of manually segmented mathematical expressions, taken from 75 randomly selected pages from 4 scientific/mathematical text books. Publications were chosen from the public domain so that the dataset can be made freely available online for the objective comparison with future or existing research endeavors.
- A novel evaluation framework which takes pixel-accurate measurements of a MEDS module's true/false positive rate, precision, false discovery rate, accuracy, specificity, and negative predictive value. Measurements are also taken for the oversegmentations, and undersegmentations made on detected regions. This framework and all of the data is freely available to help in facilitating the objective comparison of existing or future MEDS modules.
- Development of a MEDS module which is fully integrated with Tesseract's layout analysis framework. The developed MEDS module is designed so that various combinations of detection and segmentation techniques can be easily experimented with through compile-time polymorphism.

1.5 Organization of Thesis

The work to be discussed in this thesis is aimed toward moving the world a step closer to realizing some of the lofty goals set by Google's engineers and scientists. Chapter 2 presents a review of existing document analysis techniques with extra emphasis on those involving mathematical/scientific documents. Although there are a wide variety of problems which need to be tackled in the area of document recognition, the primary focus is on enhancing equation detection accuracy through the use of feature recognition and a support vector machine (SVM) classifier. Chapter 3, the method section, discusses the ground truth generation procedure, feature recognition algorithms, classification technique, and result evaluation. Chapter 4, the

results section, will involve a discussion of all results and their significance. Chapter 5, the conclusion, summarizes important points and discusses recommendations for future work.

2 Literature Review

“We are like dwarfs sitting on the shoulders of giants. We see more, and things that are more distant, than they did, not because our sight is superior or because we are taller than they, but because they raise us up, and by their great stature add to ours.”

John Salisbury

2.1 The Beginnings of OCR

2.1.1 Fixed-font

Over the past century, Assistive Technology has been a primary incentive for OCR research and development. While machine understanding was initially the most commercially viable domain for OCR, several reading devices for the blind have been implemented over the years. In 1914, one of the earliest reading devices, the Optophone [19], could allow blind individuals to understand printed text without relying on Braille. The device projected light upon a character of interest, focusing the light's reflection upon a selenium photosensor. A sound with a frequency corresponding to the reflected light would then be emitted to alert the reader of the current character. A blind individual trained to use such a device, however, could only expect to read at a mere one word per minute.

While there were some OCR patents released in subsequent years [20][8], it was not until the late forties and early fifties that there was any commercial development in the OCR industry. In 1949, RCA engineers were working on an OCR system which used an early text-to-speech synthesis technology to read individual characters out loud [21]. This system required the user to move a “eye” (a cathode ray tube) across the letters of interest. The rays were then reflected upon a photosensor connected to a complex processing unit (Figure 5). The project, however, was discontinued prior to completion because it was not judged to be commercially viable.

In 1953, David Shepard patented an OCR system, “Gismo”, which could read all 26 fixed-font letters of the English alphabet, understand musical notation, and comprehend Morse Code [22]. Shepard founded Intelligent Machines Research Corporation (IMRC) and released the world's very first commercial OCR systems. Credit card reading, although now carried out through magnetic strip recognition, was one of the first commercially successful applications of OCR. The Farrington B numeric font,

still widely used on the front of credit cards to this day, was invented by Shepard in order to minimize recognition errors.

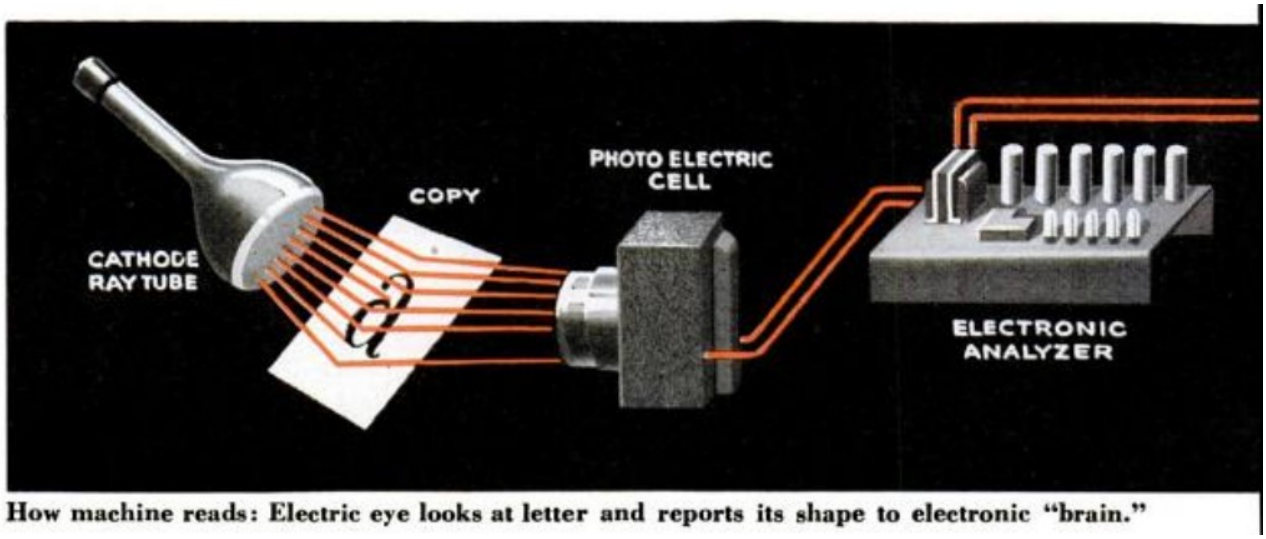


Figure 6: An image of RCA's 1949 OCR system taken from an issue of Popular Science [21]. The system was discontinued prior to completion due to its high costs .

IBM utilized Shepard's patents over subsequent years while also improving the accuracy of fixed-font OCR. The IBM 1408 Optical Character Reader [23] was packaged with the IBM 1401 Data Processing System (Figure 6) in 1960. The entire system, which included printer, optical reader, central processing unit, magnetic storage, etc., was sold for $146,600 [24], a price tag which, if sold by today's standards, would amount to over a million dollars. The IBM 1418 Optical Character Reader could only handle the ten numeric characters, the dash symbol, and the lozenge symbol. A later model, however, the IBM 1428, was alphanumeric. The alphanumeric reader could be programmed to read several document layout types assuming that they were printed in the correct font and format. Recognizable documents included premium notices, charge sales invoices, operations and route slips, payroll and dividend checks, and mail orders [25]. Throughout the 1960's, fixed-font OCR continued to be utilized and improved upon due to its usefulness in a variety of industrial applications. Some of the devices from this era are, in fact, still used even to this day for applications such as mail sorting and banking.

Although commercial OCR systems from the 1960's and early 1970's were primitive by today's standards, they were quite successful during their time as illustrated in Figure 7 [26]. Maintenance costs for word processing, an expensive resource at the time, could be reduced significantly with ordinary typewriters used for drafting and their OCR results used for final editing [27]. Fixed font OCR, although

primitive, indeed proved to meet most of the requirements set by industry. For purposes of Assistive Technology, however, it was of little to no use. The blind or visually impaired community needed an optical reader to understand not only OCR-specific fonts and layouts, but a wide variety of printed documents including books, newspapers, magazines, text books, etc., just as the idea of OCR originated primarily for the purpose of Assistive Technology, some of its most important breakthroughs were driven by this same incentive.

Figure 7 The IBM 1401 System (Optical Character Reader not shown here). From left to right, the punch card reader/writer, mainframe, printer, and magnetic tape units [26].

2.1.2 Omnifont

A major commercial breakthrough in the field of OCR came with the introduction of Ray Kurzweil's Reading Machine in 1976 [9]. Up until this time, all OCR systems were tailored to a specific font, or perhaps a specific set of fonts. This font limitation can be attributed to the template matching algorithms commonly used at the time, which would compare each incoming character image to a library of bit-mapped images. Although recognition of a larger set of fonts can be made possible through the addition of more templates into the library, too many templates would cause the processing speed of each character to decrease significantly. Although it would be ideal to have a set of fonts which could encompass all possibilities in the template library, this would prove unfeasible as there would be such a wide range of possibilities.

Omnifont recognition is characterized primarily by its use of sophisticated feature extraction techniques. As opposed to the brute force character-by-character template matching algorithms utilized in earlier systems, feature extraction enables recognition of characters irrespective of the font or typeset they are in. These techniques find properties which are relatively invariant for the same character with respect to the kinds of changes that occur across different typestyles. These properties can often include line segments (vectors), concavities, and loops. For example, the properties of a standard capital "B" include two loops on top of one another. Although the number “8” has this same feature, it does not have a straight edge on the left side as does the “B”. Furthermore, it is often that the two characters can also be disambiguated based on contextual analysis. For instance, if a character with the two vertically adjacent loops is detected at the beginning of a word, this character is far more likely to be a letter than a number.

The Kurzweil Reading Machine used feature extraction and could be trained on any number of fonts. Once the system was trained on a given font (a process taking several hours), the knowledge would be stored on disk so that retraining would no longer be required. The system could be trained to handle up to nine fonts simultaneously [10]. If the page contained pictures or multiple columns, the user would be required to specify their locations manually. While sophisticated techniques have been developed to address the problems of document analysis, the following subsection section will focus on work which has been done to prevent any retraining from being required on new fonts. With the enhancements in processing speed and more abundant memory attributed to the advent of microprocessors, it became possible to implement much more intelligent systems utilizing complex pattern recognition approaches, as will be discussed in the following section.

2.2 Pattern Recognition Techniques in OCR

As with all pattern recognition applications, in OCR some combination of feature selection, extraction, and classification is essential. In general, a statistical classifier will observe the features of its input and, based upon those features, choose the optimal class label to which the input should be associated. For a given problem, there are often many combinations of features and classifiers from which acceptable results may be obtained. The choice of classifier and feature set is largely application dependent, and, as of yet, no “one fits all solution” has been found. For OCR there are many such combinations which have been proven to yield near perfect results. This, of course, is to be expected, in that OCR is one of the most historically well-studied areas

in the field of Pattern Recognition. Not only are pattern recognition techniques fundamental to character-by-character classification, but they are also essential for the detection of merged or broken characters, text lines, word recognition and linguistic analysis, and, as will be discussed in Section 2.3, document layout analysis. While a broad overview of all techniques utilized for OCR would be outside of the scope of this thesis, some of the most fundamental and important ones will briefly be discussed.

2.2.1 Text Line Finding

Character and word classification algorithms typically operate under the assumption that the unknown text to be recognized is already in the fully upright position. This is an unrealistic assumption given the many possible angles of skew with which the text may have been scanned. Skewed text, as illustrated in Figure 8 [28], is commonly encountered by most OCR systems. Assuming that page layout analysis has already extracted all of the columns and text blocks, it is then necessary to recognize angle of skew for each block. This is essential, not only so that characters may be rotated to their upright positions prior to classification, but also to prevent words and characters in vertically adjacent rows from becoming mangled inappropriately. Individual character classification algorithms will often utilize a character's positional information within a row as a distinguishing feature. As illustrated by Figure 9 [29], there is much information about a character which can be derived from where it's top, middle, and bottom portions reside within a row. In order to have access to such information, accurate text line finding algorithms are essential. Some of the most important techniques are briefly discussed.

Horizontal Projection Profile. One of the most straightforward methods for determining the skew angle of a document image uses horizontal projection profiles. When the horizontal projection profile is applied to an M x N pixel image, a column vector of size M x 1 is obtained. Elements of this column vector are the sum of pixel values in each row of the image [30]. The contents of this vector are at maximum amplitude and frequency when the text is skewed at zero degrees since the number of co-linear black pixels is maximized in this condition. One way in which the horizontal projection profile can be utilized is by rotating the input image through a range of angles while calculating the projection profile for each one [31]. Each projection profile is then compared to determine which one has the maximum amplitude and frequency.

Although much work has been done in order to optimize this approach, there are still more efficient and accurate methods which can be utilized [28].

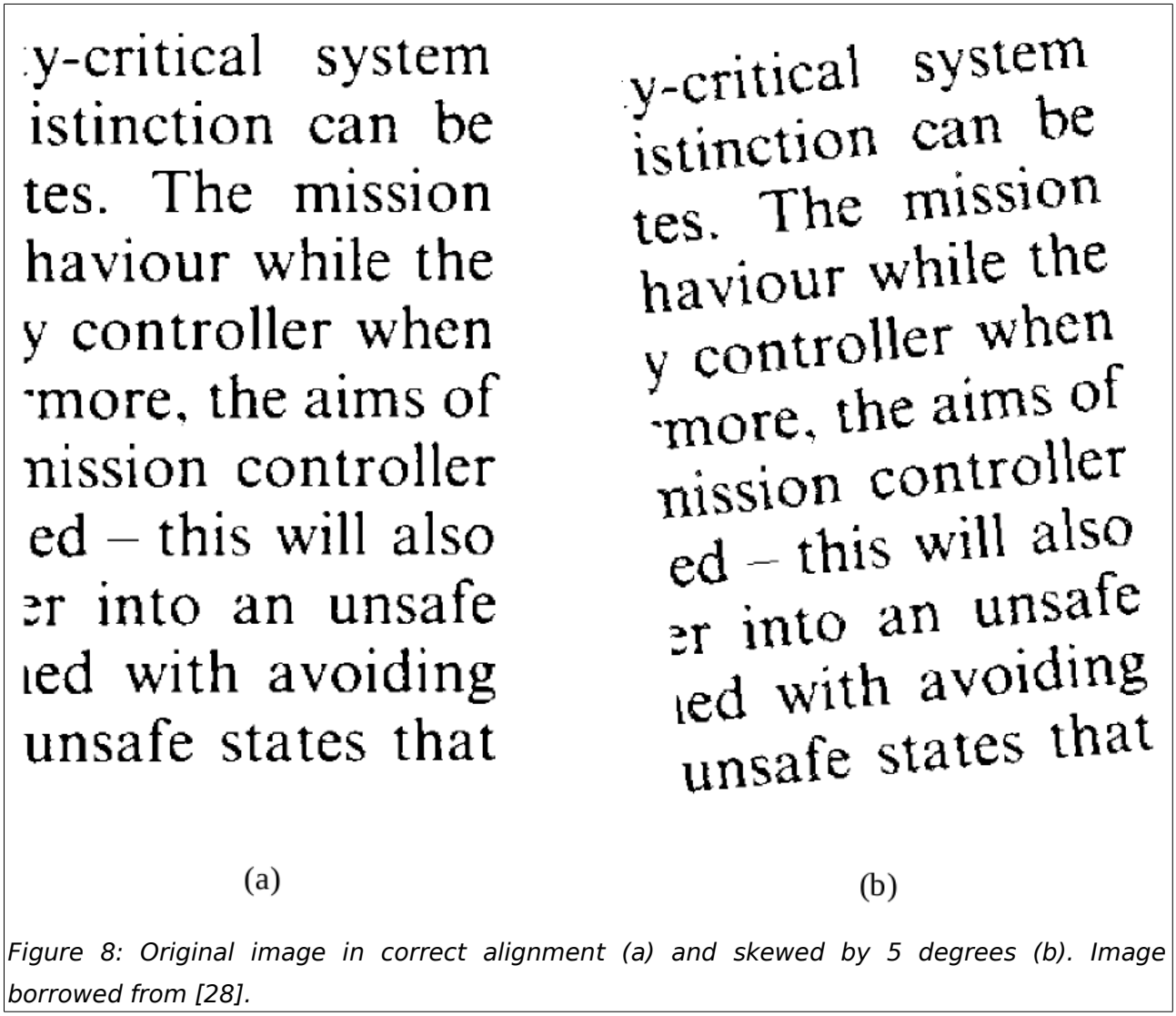


Figure 8: Original image in correct alignment (a) and skewed by 5 degrees (b). Image borrowed from [28].

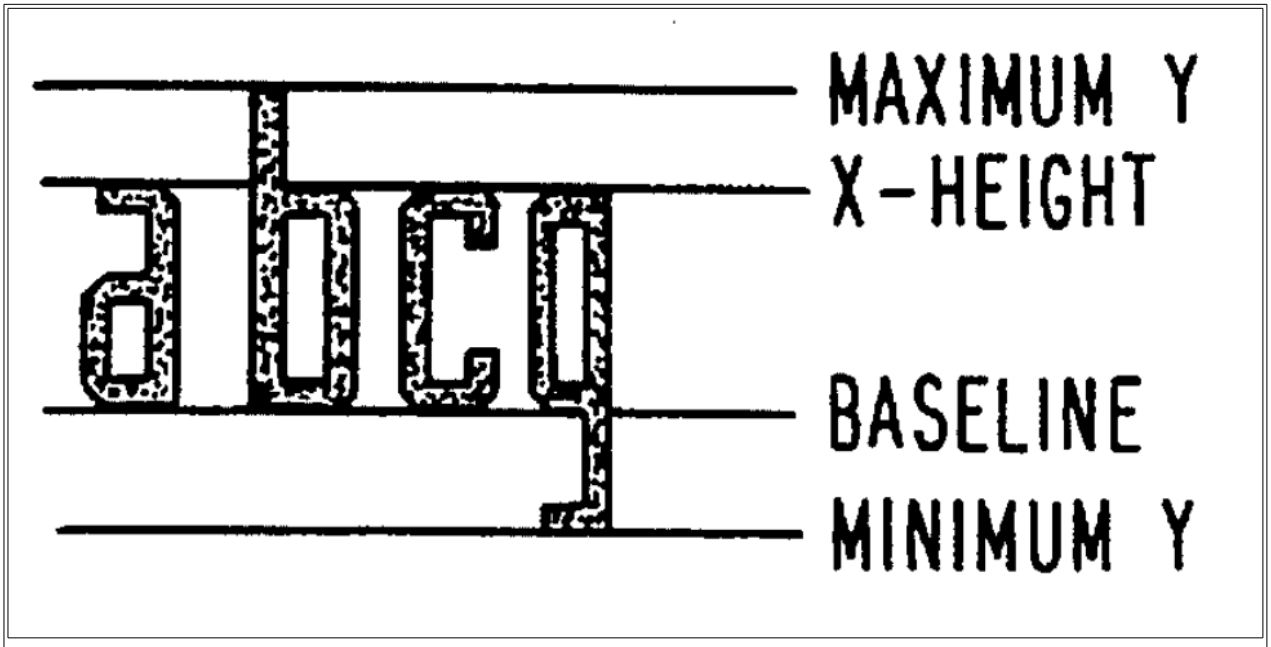


Figure 9: For printed text, a given character often has a precise position within the text line which can be useful for classification purposes [29].

Hough Transform. The Hough transform, a well known feature extraction technique in computer vision, can be utilized in order to detect, not only the skew angle of a document image, but any mathematically tractable shape of interest. This technique, when applied to 2D images, will take a series of (x, y) coordinates (for the case of document images this will likely correspond to groups of connected pixels) and transform them into a new coordinate space. While the coordinate space will vary depending upon the desired shape to be detected, for straight lines the x and y coordinates will be converted to the (ρ, θ) coordinate space using the following equation:

$$\rho = x\cos(\theta) + y\sin(\theta)$$

where ρ is the distance of the (x, y) point from the origin (usually at the upper left-hand corner of the image), and θ varies between -90° and 90°. The (ρ, θ) parameter uniquely represents a given line in the image by specifying its perpendicular angle and distance with respect to the origin as shown in Figure 9.

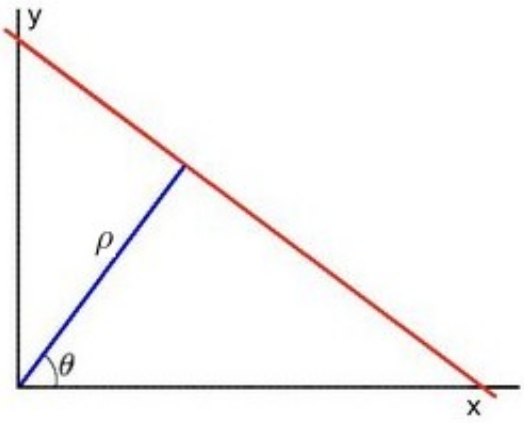


Figure 10: Mapping from (x,y) to Hough space

For each chosen (x, y) coordinate within the image, the Hough transform algorithm will calculate the ρ values corresponding to some subset of the possible θ values between -90° and 90°. There can be an infinite set of lines going through a given point, thus the amount of lines required per point depends upon the desired accuracy of the system as well as desired overall computational speed. The set of chosen lines per point, each represented in Hough parameter space (ρ, θ), is represented by an accumulator array [32], each entry of which corresponds to a unique line in the image. Each time that a line is found to go through an (x, y) point of interest, its corresponding entry in the accumulator array is incremented. When the process is completed, the accumulator array entries with the highest increments will correspond to the lines which intersect the most points in the image.

For OCR purposes, lines of text may be found within the image based upon this operation. The (x, y) coordinates of interest typically correspond to the centroids of connected components (groups of connected foreground pixels which often correspond to individual characters). When several parallel lines are found to have very high entries in the accumulator array, this will often mean that the page was scanned at the skew angle corresponding to these lines, and that they are likely to represent individual lines of text within the document.

Geometric Distribution of Connected Components. The Hough Transform has been utilized in various techniques to achieve accurate skew results. For a more complete survey of past techniques the reader is referred to [28]. These techniques, for the most part, vary, not by their use of the Hough Transform, but by their method for determining connected components which are of interest and most likely to

correspond to rows of text. In [33], Smith utilizes an efficient and simple algorithm which, unlike previous methods, finds lines of text independently of the page's skew. The connected components of the image are extracted and filtered such that the remaining components are most likely to represent a body of text. The connected components are then sorted based on their positions in ascending order from left to right and iterates through them. Each connected component is added to a row of text to which it is most likely to belong based on vertical overlap. If no such row exists then it is created. Based upon which connected components are added to which rows, a running average is kept on the slope of the text rows. This process is continued in an iterative fashion until all connected components have been associated with rows. This algorithm has been found to achieve reasonably accurate results while proving to be more efficient than corresponding Hough Transform based algorithms.

Curved Text Line Detection. Even when text lines are accurately found, it is often the case that the lines will need to be fitted to the text more precisely due to scanning artifacts which may give the text a curved appearance as depicted in Figure 10. Among the techniques utilized for this problem are quadratic or cubic spline modeling via least square fitting techniques [34] as well as active contour tracing via snakes [35]. Smoothing techniques are often applied in order to simplify the input for curved line detection. The optimal technique to be applied largely depends upon the type of document fed into the OCR system. Thus document understanding at early stages in the OCR process is of great importance in achieving accurate results. For a more complete account of text line detection in various documents, the reader is referred to [36].

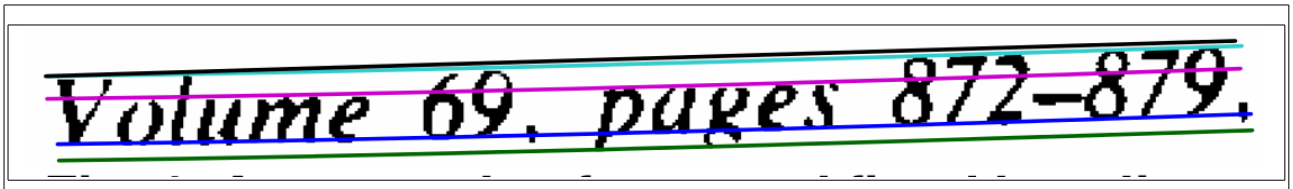


Figure 11: An example of skewed and slightly curved text borrowed from [34]. Close inspection shows that the cyan/gray line is curved relative to the straight black line above it.

2.2.2 Character Feature Extraction

The problem of feature extraction for optical character recognition, although a difficult task, has been extensively studied in the literature. Techniques vary based upon their application, with handwritten recognition often requiring different

techniques from printed character recognition. As with the rest of this thesis, the focus here will be on techniques pertaining to printed character recognition. Techniques utilized by Google's OCR System, Tesseract, will be emphasized and discussed primarily since they are used within this thesis project.

Edge Extraction. After text lines have been located as discussed in the previous section, the next task for a typical OCR system will be to perform some image processing operations on the input in order to make features more easily and efficiently recognizable. Tesseract [29] utilizes a novel edge operator which can take advantage of grayscale values if they are available to achieve robust character segmentation results. Text and non-text can often also be distinguished based on contextual evidence as well as using basic height/width filters. Furthermore, the edge extraction algorithm will inherently filter out a significant amount of noise since it will disregard any portions of the image which do not form closed loops. The term “closed loop” is used here to describe a contour which, after being followed a certain amount of time will return to its starting position.

Also of importance is preserving the relationships between the inner and outer portions of characters. Take, for instance, the character “o” depicted at the left on Figure 12 [37]. Since the edge detector will find the inner and outer portions of this character as separate, simple data structures must be implemented which store the relationships among overlapping edges. In Tesseract, a 2D bucket sorting technique is utilized in order to store all of the inner portions of characters as enclosures or “holes” within them. The results of edge extraction are stored in chain code format as illustrated by Figure 13 [38].

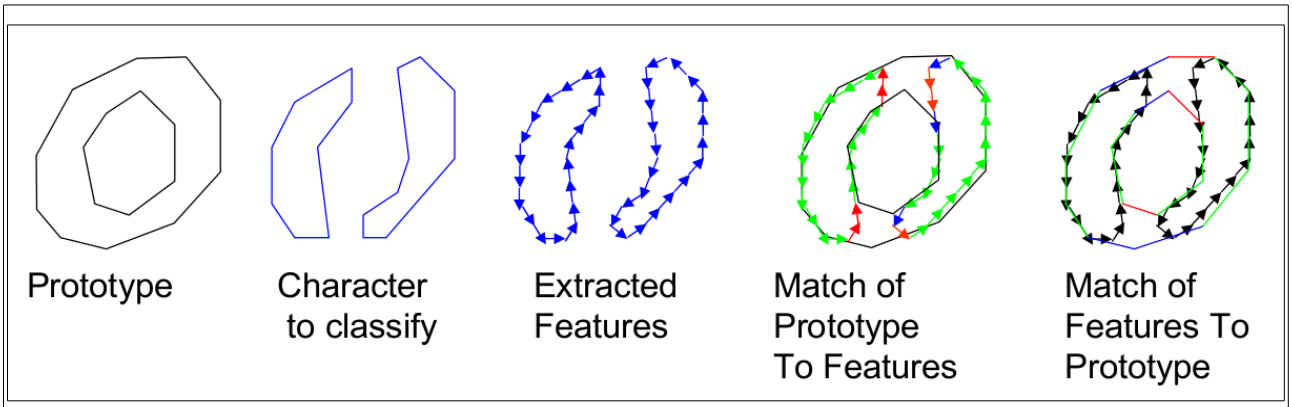


Figure 12: Comparison of a prototype for “o” to an unknown character by template matching [37].

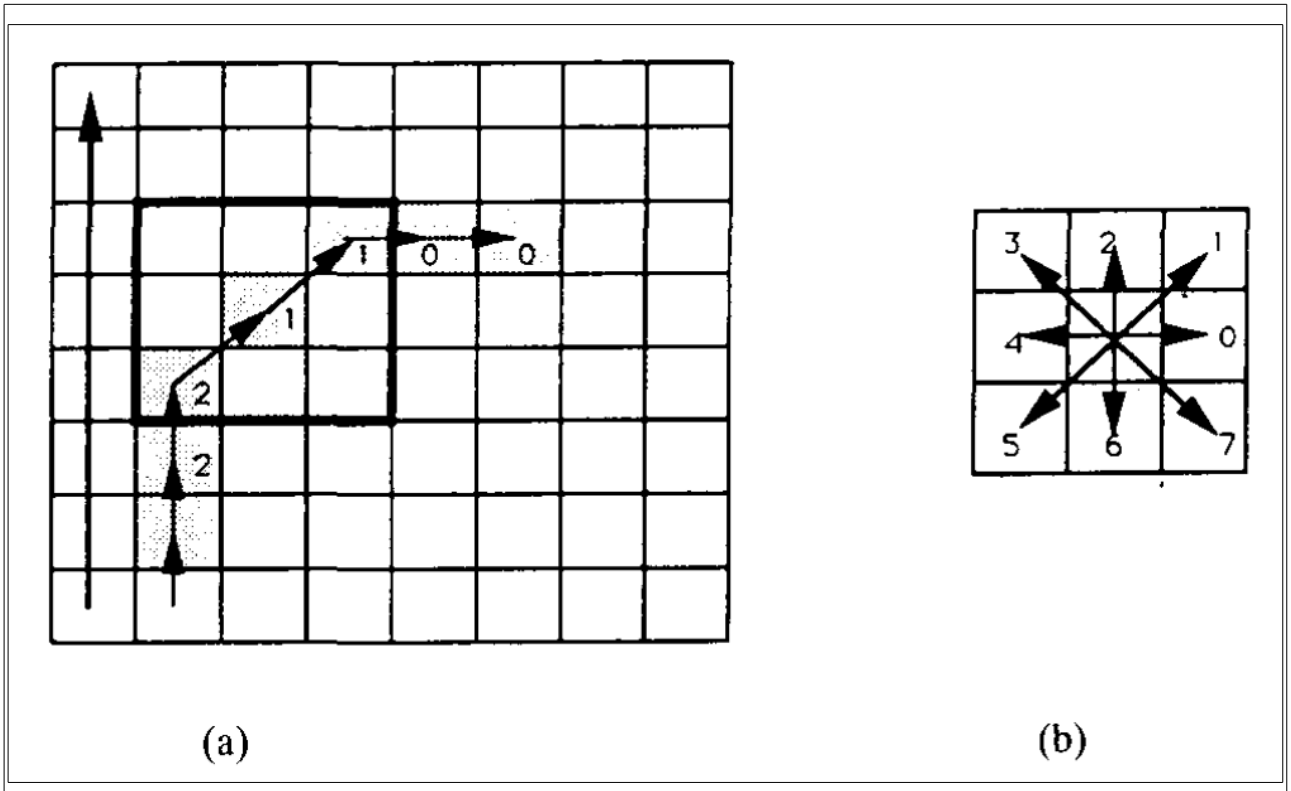


Figure 13: (a) Example of chain-code. (b) Chain-code directions [38].

Polygonal Approximation. In Tesseract, the process of polygonal approximation is required in order to optimize the efficiency and accuracy of subsequent feature extraction techniques. Polygonal approximation of a character image, if done effectively, results in an output whose data is neither too fine or course for purposes of feature extraction [39]. It becomes easier to detect global convexes and concavities as well as character enclosures, which are very important features.

The process of polygonal approximation utilized by Tesseract analyzes the chain code output of the edge extractor in order to locate simplifications which can be made, which will enhance the robustness of subsequent feature extraction techniques. The process begins by first breaking up the character into directional segments, separated by 90° or two subsequent 45° transitions [29]. The second stage involves further analysis of these segments and subsequent approximations being made between the end points of each segment. The process is repeated iteratively until certain criterion are met.

Normalization and Template Matching. After polygonal approximation and prior to feature extraction, normalization is applied to the input in order to eliminate some of the complexities which may come about from various font differences. For an in depth discussion on normalization techniques the reader is referred to Chapter 3 of

[39]. Normalization is very important in accounting for character font transformations which may occur in terms of size, perspective, and rotation with respect to the features of prototypes used in training the system. Normalization techniques can, in general, be separated into categories using either linear or nonlinear methods. While linear methods account for affine transformations often found in printed characters, nonlinear techniques are generally geared more towards handwritten character recognition wherein much more drastic variation is to be expected.

Normalization can be performed either before or after feature extraction. If done after feature extraction, then the process is carried out within the feature space rather than directly on the character's pixels. In the case of Tesseract, normalization is carried out on the feature space of the character's polygonal approximation, which can be viewed as a vector of 3D features, the dimensions of which are simply x position, y position, and direction within the range of $[0, 2\pi]$ [40]. Figure 14 gives an example of how Tesseract will normalize the features of unknown characters while matching them to those of character prototypes.

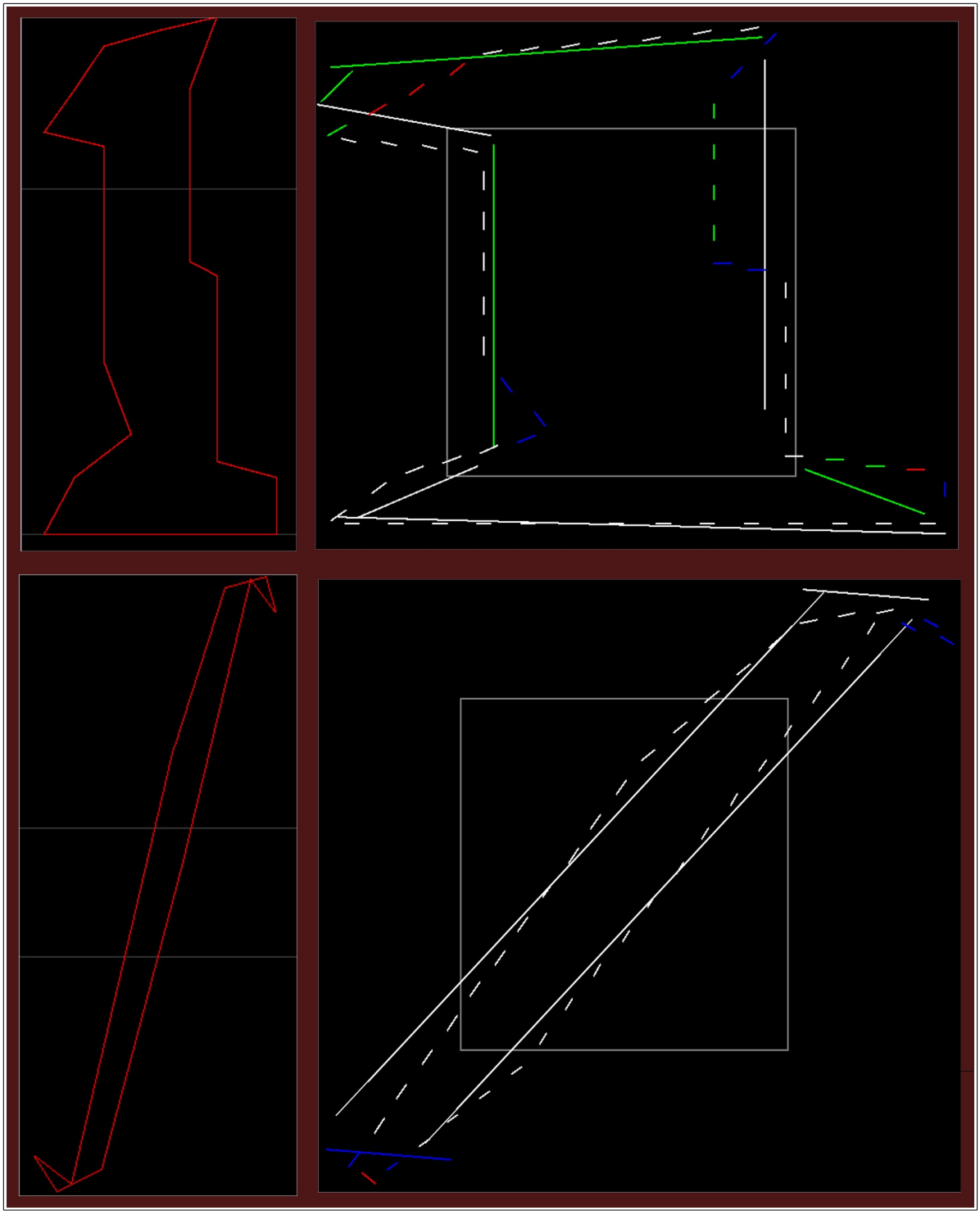


Figure 14: (top) The polygonal approximation features of a "1" followed by those same features after normalization with respect to the prototype of "1" to the right. (bottom) Features of an "integral," a character for which there is currently no valid template in Tesseract. To the right is the integral after normalization with respect to the prototype of "/". For both normalized pictures, the solid lines represent the prototypes while the dotted lines represent the normalized unknown character. Lines are colored from best to worst match: white, green, red, blue. These images were taken from Tesseract's debugger.

As illustrated by Figure 13, Tesseract normalizes a feature vector by each character prototype to which it is compared. For instance, assume that the character “8” is fed into the system. Based upon a coarse shape analysis of the character a subset of the the total prototypes may be chosen as potential candidates. For instance “B” may be chosen since it has two enclosures, and “0” may also be chosen based upon its convex top and bottom regions. Assuming that only these characters are chosen as candidates, the feature vector for “8” will be subsequently normalized based upon both of these prototypes prior to the respective template matching. The process of normalization begins by isotropically scaling the bounding box to a fixed height and width. The feature vector is then centered and scaled anisotropically based upon the second moments of the prototype to which it is being compared [40]. Moment-based character normalization has been studied extensively in the literature dating back to even before the advent of microprocessors. For some examples of in-depth studies the reader is referred to [41] [42].

During Tesseract's classifier training, the training data is automatically grouped into clusters based upon certain important features. These feature clusters are then utilized during classification to reduce computation time with very little loss in accuracy. The five most important features utilized by Tesseract will be herein briefly discussed.

Concavities. One of the most important features in character recognition are concavities. By definition, a concavity is part of an outline which does not lie on its convex hull (the smallest convex region enclosing the outline as illustrated by Figure 15 [43]). In Tesseract, a concavities are characterized by the direction of their hull line, their centroid [29], shape, skew, and area.

Figure 15: Red outlines represent convex hulls of the white regions [43].

Functional Closures. Character closures are common features which can be useful in distinguishing characters regardless of their font. For instance the “e” and “o” characters will both always consist of a single closure when properly drawn or printed. The term functional closure is useful when a character's closure may be slightly degraded somehow, such that there there may be an unintended opening. In Tesseract [29], each concavity is tested for functional closure. Based open the location of the concavity within the character (i.e. upward facing, downward facing, etc.) a threshold is assigned for the maximum character to concavity width ratio expected for a functional closure. If the ratio is below the appropriate threshold then a functional closure will be detected.

Axes. Tesseract defines axis features only on characters for which no concavities or closures are detected. Characters including commas, periods, quotations, etc., fall under this category. The axis feature measures a character's length to width ratio. The length of a character is determined by finding a point on the outline whose distance from the character's centroid is maximum. The vector going from the point to the centroid is said to be the character's major axis. The character's width is then calculated as the sum of the maximum perpendicular distances from the major axis to the character outline on either side of the axis. The major axis length to character width ratio can be useful in disambiguating commas, periods, quotes, etc.

Lines. As illustrated by Figure 13, lines are useful features in template matching. Line features are only used by Tesseract for unknown characters which closely match more than one of the prototypes, as measured with concavities, closures, and axes [29]. The degree to which a line in the unknown character matches a line in a prototype is measured based upon the normalized position of the center of the line, its quantized direction, and its scaled length.

Symmetry and Detection of Italicized Characters. Vertical as well as horizontal symmetry can be a very useful measure in discriminating certain characters. For instance, the character “C” and “G”, “j” and “/”, “j” and “]”, “T” and “1”, etc. can often be disambiguated through their respective measurements of symmetry. The main difficulty in symmetry measurement is not in measuring the degree of symmetry about an axis, but rather in locating the axis of interest. While the problem is trivial for vertical text (simply drawing a vertical line through the center will suffice), italicized text is much more difficult since the axis is rotated slightly and may be difficult to locate. Tesseract utilizes two methods to determine a character's axis of

symmetry. Once this axis is found it is then easy to determine whether or not the character is italicized.

The first method used by Tesseract searches the character's outline for a vector which passes from the bottom to the top half of the character's bounding box. The direction of this vector may be a good indication for the direction of the axis of symmetry. For round characters such as “o” and “e” and those which contain vertical lines such as “H” and “p”, this method is useful. However, for angular characters such as “X” or “8”, a valid result is not produced. The second method finds the rightmost point on the outline then calculates the most clockwise line which can be drawn through this point, without intersecting any other point on the outline. This operation is repeated on the leftmost point of the outline as well. The line which was least clockwise from the vertical becomes the axis of symmetry.

After the axis of symmetry found, the outline is searched around the axis for points of reflection. Symmetry testing is commenced at a point where the axis intersects the outline and works in opposite directions simultaneously. The points are tested for being in the same locality of the point on the opposite side of the axis. Symmetry is only measured for certain character candidates and typically only in one direction (either vertical or horizontal).

2.2.3 Character Classification

The line finding, edge extraction, polygonal approximation, and feature extraction techniques discussed thus far would be of little to no value without an effective classifier. In pattern recognition, a classifier will take a set of feature measurements as input and, using these measurements, choose from a finite set of classes, the class to which the unknown input is mostly likely to belong. In the case of OCR, the classes will often correspond to individual characters. Tesseract employs two separate classifiers: one is termed the static classifier while the other is the adaptive classifier. In order to save computational time, a class pruner is utilized first to narrow down the number of candidate classes for an unknown character.

Tesseract Class Pruner. In the first stage of classification, Tesseract will employ its class pruner in order to reduce the number of potential candidates to which an unknown character is to be compared. The class pruner uses a fixed quantized version of the 3D feature space wherein each of the 3 dimensions (x, y, θ) are quantized into 24 cells. After the unknown character's features are quantized, they are indexed to the quantized feature space in order to obtain a set of classes which allow

the given features. The number of feature hits for each class is summed and the best few matching classes are then fed into the next stage of classification [40].

Tesseract Static Classifier. Both Tesseract's adaptive and static classifiers are unique when compared to more standard techniques in that they operate on a variable number of features. While standard classifiers such as neural networks, support vector machines, etc., will work in a feature space of fixed dimension, Tesseract has a variable number of features for each class of interest. The classifer can be regarded as an optimized K-Nearest-Neighbor (KNN) classifier where the character class, k, with minimum distance from the unknown character is computed as follows:

$$\min(k)\frac{1}{M+J_k}\left(\sum_{l,i}\left(x_{il}-\mu_{ijk}\right)^2+\sum_{j,i}\left(x_{il}-\mu_{ijk}\right)^2\right)$$

where the variables are as follows: i is the current feature dimension (either x position, y position, or θ); j is the cluster; k is the character class; and l is the unknown's feature. x_{il} is the feature dimension (either x position, y position, or θ) of the unknown character x's feature at index l. M is the total number of feature vectors in the unknown character, x (this varies depending upon the character of interest). J_k is the total number of character clusters which the training set was divided into. μ_{ijk} is the mean feature value for the i^{th} feature, j^{th} cluster, and k^{th} class calculated during training.

While the left-most summation measures the distance between each feature dimension and its corresponding average clustered prototype value, the right-most summation measures the distance between the average value in each cluster to the corresponding feature dimension. The result of these summations is then divided by the total number of features in the unknown character and training set. A key advantage to this approach is its symmetry. The nearest matching features between both the unknown and prototype and the prototype and the unknown are effectively found. Say, for instance that the unknown character is “e” and the prototype to which it is compared is “c”. Since most of the features in “c” are allowed by “e”, it becomes possible that the “e” will be misclassified as “c” if only the distance between the “e” and “c” is computed. When the distance between the “c” and “e” is added into the classification, the lack of crossbar in the “c” will incur a penalty, thus lowering the risk of misclassification.

Tesseract Adaptive Classifier. After word recognition, as will be briefly discussed in section 2.2.5, a second pass is made by Tesseract's classifier. This time the classification is considered to be adaptive in that it utilizes the extra information obtained after word recognition in order to better train the classifier to the current font. After word recognition is carried out, there may be several characters which can be disambiguated and thus used to better train the classifier on the second pass and increase accuracy. The adaptive classifier is essentially the same as the static one except that it applies a different type of normalization to the unknown character prior to comparing it to the prototype. While, for the static classifier, the centroid of the unknown character is centered in the feature space and then scaled anisotropically to normalize the second moments of the outlines, the adaptive classifier will normalize the unknown by centering the horizontal centroid of the outline and scaling isotropically to normalize the x-height of the character. This normalization retains font differences, which, at this stage of OCR, is very important [40].

2.2.4 Detection of Merged or Broken Characters

While some of the first OCR systems would only recognize each individual character independently, more sophisticated systems such as Tesseract, Omnipage, and Abby Fine Reader, ReadIris, etc., [11][12][13] analyze inter-character relationships in order to increase their systems' robustness in the presence of noise. In Tesseract, while the results of word recognition (described in Section 2.2.5) are found to be unsatisfactory, a character merger/segmenter module is utilized in order to test the word on new potential character candidates in areas with low character recognition confidence. The merger/segmenter module will locate concave vertices of a questionable character's polygonal approximation and attempt to separate the character in those locations to test for a possible merged character as illustrated by Figure 16. Likewise, potentially broken characters are attached to their neighbor and tested if their combined width is within an acceptable range.

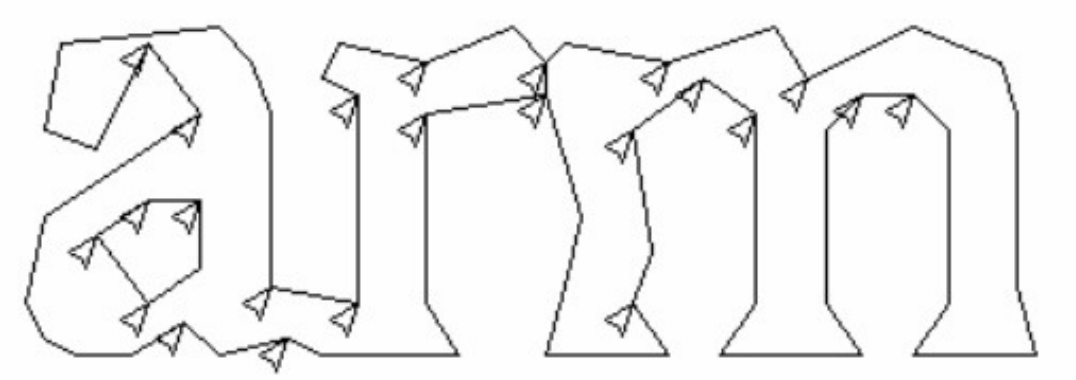


Figure 16: Example of merged letters with candidate chop points (denoted by triangles) [37].

2.2.5 Word Recognition and Linguistic Analysis

Individual words within Tesseract are detected based upon the distribution of space between characters found on a text line. Characters which are within an appropriate horizontal distance parallel to the text line are considered to be within the same words, while groups of such characters are considered to be separate words. The word recognition module looks up candidate words in a dictionary to make sure they are valid. This information is also vital to detecting broken or merged characters and training the adaptive character classifier.

Some basic linguistic information can be important for increasing accuracy. For instance, in Tesseract's English word recognition module, numeric characters are not allowed to exist in alphabetic words, uppercase characters cannot follow lower case ones, and the only punctuation allowed within a word are apostrophes. Markov methods are also very useful in OCR due to spelling conventions (such as u following q) and the need for words to be pronounceable (i.e., g is unlikely to follow j). By modeling each individual character as a possible state and each character occurence as the next element in a Markov chain, it is possible to use a transition matrix (whose width and height are 26, the number of characters in the English alphabet) to help in selecting a word's next character [29]. While it is possible to make choices based upon multiple characters, the transition matrix for making a choice based upon the previous m - 1 characters would require a transition matrix of size $26^{(m\text{-}1)}$x$26^{(m\text{-}1)}$. Rather than using large values for m, Tesseract employs dictionary methods.

Strings of characters can be reduced into words which are either in a dictionary or can be generated through the use of various production rules [44]. For each string of characters a set of candidate words are derived using the dictionary. The word which has the highest overall rating based upon the recognition confidence of its individual characters is chosen. The word recognition result can then be utilized in order to boost the adaptive character classifier's accuracy since certain characters which had low confidence in the static classifier may now be confirmed.

2.3 Document Layout Analysis Techniques

2.3.1 Introduction to Document Layout Analysis

The improvements made in the field of commercial OCR throughout the 80's and early 90's are primarily attributed to enhanced processor and digitizer technologies rather than to improved classification techniques for individual patterns

[45]. By the early 90's there had been significant progress already made toward the study of OCR and pattern recognition techniques which are still largely in use to this day. A significant amount of the more recent progress made in the state-of-the-art has been due to improvements in document layout analysis and understanding as opposed to the much more mature character-by-character feature extraction and classification algorithms.

While inflexible hardwired classification engines once dominated the market for OCR, the computational advancements of the 70's and 80's allowed for more intelligent systems to take hold. While systems became robust against multiple fonts, merged/broken characters, and document skew as discussed in the previous section, the need also arose for systems which could recognize pages from a wide variety of document types. While an OCR system may be predominantly exposed to documents like newspapers, magazines, letters, etc., it is also often necessary to process such “special” documents as electronic circuit diagrams, envelopes, checks, tax return forms, music notations, etc. [46].

The importance of document layout analysis techniques is made apparent in the presence of both the former and latter document types as they may contain complex backgrounds, lines with drop-caps, mathematical formulas, various symbols, imagery, tables, graphs, multiple columns, titles, headings/subheadings, etc. Therefore, it becomes important, not only to recognize the individual words and characters, but to also interpret and preserve the layout and spatial context of a document's components. Such details as spatial context and document structure are vital in conveying a document's message as it is intended to be perceived, as well as for understanding how exactly the document needs to be processed in order to achieve the optimal recognition accuracy. Figure 17 [47] shows two examples of document images with complex layouts.

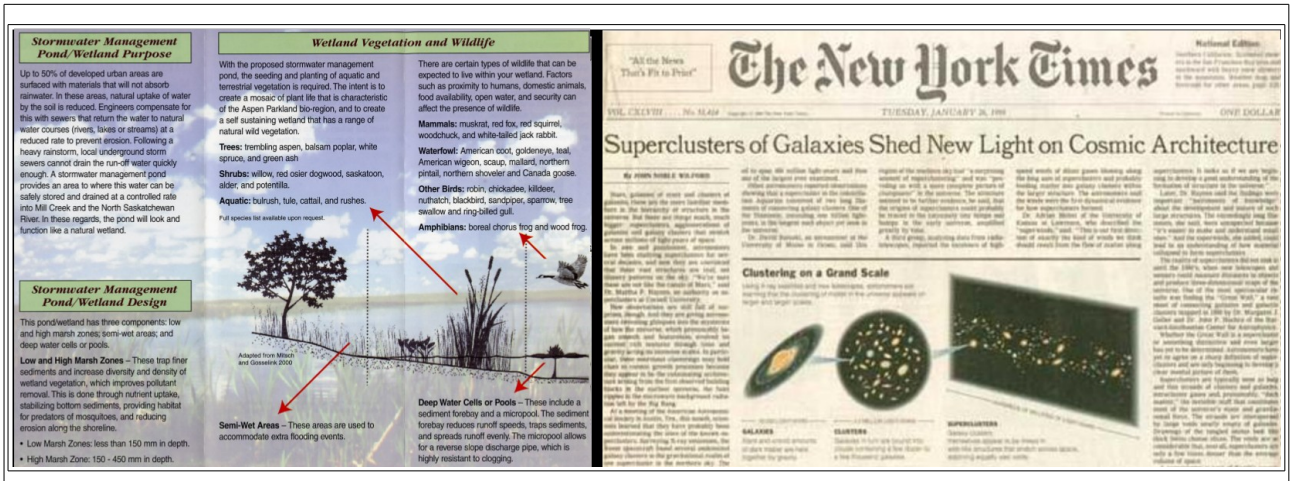


Figure 17: Two document images with complex layouts [47]. On the left, an intricate background as well as complex column structure is observed. On the right is a newspaper article with a complex layout of titles and columns along with imagery and captions. Both document images will require sophisticated image processing and layout analysis techniques in order to achieve both OCR accuracy as well as efficient data storage and indexing within computer systems.

Document layout analysis is a very important design component for any OCR system and has been extensively studied [46][47][48][49][50][51][52][53]. Not only is document layout analysis often essential for obtaining correct OCR results, it can also provide the means for computer systems to use logical information such as titles, footers, authors, captions, abstracts, page numbers, etc., to more efficiently store and index a document image's information [54]. This contextual information is also essential for Assistive Technology purposes, in enabling blind individuals to have an understanding of the same spatial and logical cues afforded by the document's visual layout [55]. This section will discuss how the field of document analysis is divided into various sub-problems by existing literature and then compare and contrast various techniques which address these problems. The problem of document analysis can be broadly divided into its five most important interdependent components: image preprocessing, document structure analysis, document content representation, training set development, and finally performance evaluation as illustrated by Figure 18 [47]. After a brief overview of what each stage entails along with some introduction of terminology, various techniques found in the literature for each stage will be discussed.

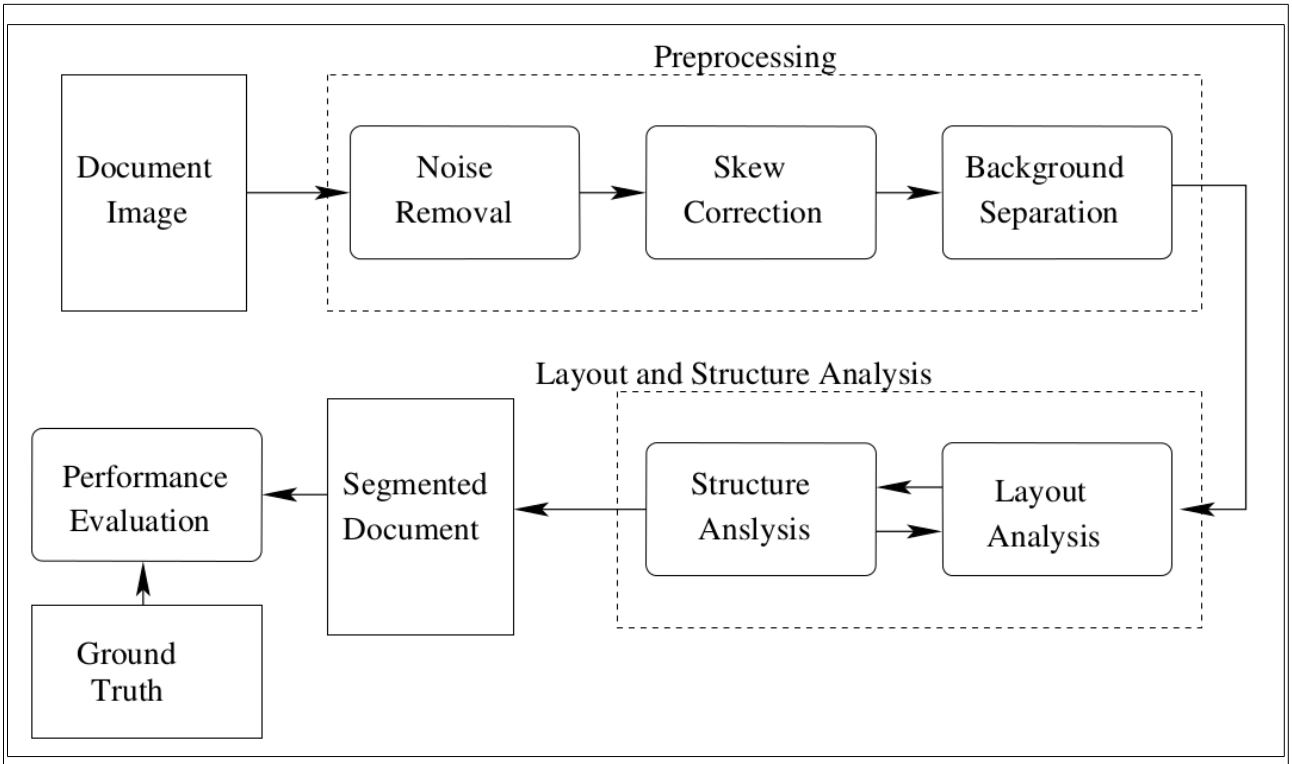


Figure 18: The five most important inter-dependent components of document analysis involve preprocessing, document structure analysis, document content representation (not illustrated here), training set development (ground truth), and performance evaluation. Each module is described as interdependent because the performance of the overall system really depends on each component. For instance, if preprocessing is not effective, then structure analysis will likely fail. If the document content representation is not consistent, then ground truth and performance evaluation will yield insignificant results. Diagram borrowed from [47].

Firstly, the most common problems addressed by image preprocessing (Section 2.3.1) in document analysis involve noise removal, separation of background and foreground regions, and skew correction. Secondly, after any necessary preprocessing is carried out on the document image, the modified image is fed into the system's document structure analysis module. From a broad perspective, document structure analysis involves first extracting the document's geometric structure and then mapping that structure into a valid logical one which can be understood by computer systems. A document is thus considered as having both a physical (geometric-based) structure and a logical (content-based) structure. Thus document structure analysis is commonly divided into two distinct phases: physical layout analysis and logical layout analysis. Each distinct phase of document structure analysis will be further discussed in Section 2.3.2.

Thirdly, an important question which must be asked prior to the design of any structural layout system, is how exactly a given document should be represented by a computer internally. This brings about the problem of document content representation (Section 2.3.3) which also has been extensively addressed by the literature. A standard format for all documents is of course desired, however the problem of finding a standard format which could accommodate all possible document image layouts has been no trivial one. A variant of SGML or HTML is commonly employed, however the tradeoffs between different representations as well as grammars will be briefly reviewed. Fourthly, given the knowledge of how a document should be represented internally, an important question is what training sets and groundtruths are available for evaluating performance (Section 2.3.4). Since building a training set manually is an extremely expensive and time consuming process, the ideas of creating synthetic training data and/or training a system to automatically generate training data from documents while requiring little to no human correction, have been explored. Finally the issue of performance evaluation has been explored through various techniques which will be discussed in Section 2.3.5.

2.3.2 Preprocessing

The most common problems addressed by image processing involve noise removal, separation of background and foreground regions, and skew correction. Since skew correction was already covered in great detail by Section 2.2.1, it will not be further discussed in this section. Noise removal in image processing is a well studied field and advanced techniques have been developed to cope with white noise, salt and pepper noise, quantization artifacts, etc. Such noise sources are often compensated for by using techniques such as median filtering, dithering, low pass filtering, etc. [52]. An in depth overview of noise reduction methods in image processing can be found in [56]. For purposes of document layout analysis, one of the more important noise removal tasks involves the detection and filtering of half-tones. This discussion will be followed by a brief overview of preprocessing tasks for background and foreground separation.

Noise Removal: Dealing with Half-tones

Halftones, as illustrated by Figure 19 [57], utilize variably sized or spaced dots in order to create the optical illusion of an infinite range of colors while, in actuality, only printing a limited amount. Half-tones are utilized by color and grayscale printers in order to reproduce imagery while requiring few colors of ink. Figure 19, for instance, creates the illusion of grayscale while only requiring black dots. When scanned at high resolution, the halftones in a document become a significant noise artifact, as an image's connect components clearly should not be divided into such small dots for document analysis purposes. Halftones can be detected through the use of various filtering techniques [58], whose accuracy often depends upon the dot sizes and spaces of the halftone in question. Once detected, the halftones can be converted into continuous grayscale by applying an appropriate low-pass filter to smooth out all of the dots, followed by a sharpening technique which will reduce the blur.

Figure 19: An example of a halftone image [57]. Notice that, when looking at the image from a distance, the illusion is created that the image is in grayscale, when, in fact, it is actually printed with only black dots of varying sizes.

Background and Foreground Separation

Although the problem of foreground detection is often very simple in the case of the most typical black text on white background, the problem becomes much more complex when faced with intricate backgrounds which are overlayed with text in varying color, size, and font as depicted in Figure 20. In the former case it is possible

to use thresholding techniques like Otsu's method [59]. An alternative method which could work for varying background and foreground color schemes would be to find the outline of characters through edge detection [29]. In the presence of complex backgrounds, however, more sophisticated background and foreground separation techniques may be required. A common approach is to compute statistical properties of image patches and assign them as either foreground or background using a trained classifier such as a neural network [47]. Through a combination of edge detection and a trained classifier it becomes possible to detect foreground text of varying colors on a complex background with a certain degree of confidence as demonstrated in Figure 20.

Figure 20: (Left) Part of a document image with complex background. (Right) The same image with foreground separated from background [47].

2.3.3 Document Structure Analysis

A primary component of any document analysis system is the document structure analysis stage itself. As previously indicated in Figure 18, however, the steps of preprocessing, document content representation, training set development, and performance evaluation also play a crucial role. In this section the term “document

structure analysis" is used to refer to the broad class of both physical and logical document structure analysis methods which will be explored in this section. In general, physical layout analysis techniques are one of the first steps of an OCR system and will initially divide the document image into areas perceived as text and non-text, as well as splitting multi-column text into columns [18]. In this literature review an important distinction between physical and logical layout analysis techniques is made such that, while logical layout analysis techniques make final classification decisions on blocks, physical layout analysis techniques extract and evaluate the geometry of blocks without necessarily reaching any final conclusions on their syntactic meaning. While the physical layout analysis stage looks for geometric patterns, the logical layout analysis stage will utilize this and other information in order to infer a document's meaning from a syntactic perspective (i.e., the type of document and the location and functional purpose of its "zones" which may include titles, headers, footers, math equations, imagery, etc.). This logical understanding is very important both for indexing and storage purposes as well as for Assistive Technology applications as previously mentioned.

A document analysis system must be able to understand not only how a document can best be partitioned into its logical sections, but also the role that physical geometry plays in conveying information effectively. It is important for a document recognition system to move back and forth between physical and logical analysis in an intelligent manner which may vary significantly depending upon the aspects of what is being recognized. This concept is illustrated by the bidirectional arrows seen in Figure 18. Although it is possible for a very specific physical layout to match to only a single logical structure (i.e. in the case of a very complex and unique form), there is never a guaranteed one-to-one mapping between any physical and logical layout or vice versa. In creating a system that can generalize to a wide variety of document structures while minimizing overfit, it is thus important not to make assumptions too early based solely upon geometric information. A system may require to make "fuzzy" decisions which, in later steps, can be further refined to reach an appropriate solution. For instance, if text is found to be centered within a column this could open up many distinct possibilities based upon the contents of the text itself as well as its context within the entire page. It could, for instance, be the title of a new subsection, new chapter, a mathematical formula, a quote, image caption, or any number of other possibilities. Thus, while an understanding of the geometric structure of a block of text is important, there is more information required in order to

understand the block's logical structure. If an OCR algorithm yields results with low enough confidence then various alternatives can be tested (i.e. for mathematical formulas, musical notation, other languages, etc.).

Document Physical Structure Analysis

Physical layout analysis, an essential step for all OCR and document analysis systems, localizes individual blocks of text and imagery while leaving assignment of logical meaning of these blocks as well as final classification of text/nontext regions to later stages in processing which will be discussed in the Document Logical Structure Analysis Section. Methods for physical layout analysis fall into roughly three categories: top-down, bottom-up, and hybrid, each of which will be discussed in turn by this section. An important distinction between algorithms involves the types of physical layouts which they can handle. The following three types of physical layout patterns are commonly defined: these include Manhattan, rectangular, and arbitrary layouts [60]. A document's Manhattan layout can be viewed as the document divided into a grid, which may be horizontally or vertically split recursively into smaller components in any given region. For a Manhattan layout, if a region overlaps another then it must be entirely covered by that region (i.e., there is no partial overlap). Rectangular layouts consist of several rectangles arbitrarily spaced apart or which could be partially overlapping. Arbitrary layouts, on the other hand, are formed by unconstrained polygonal shapes as demonstrated by Figure 21 [61].

Top-down physical layout analysis techniques recursively segment the document into smaller rectangles which are expected to correspond with image, column, paragraph, or other text block boundaries [54]. Bottom-up techniques, on the other hand, analyze individual pixels or connected components, recursively merging them together into larger regions. While bottom-up techniques can handle arbitrary physical layouts, top-down methods are constrained to only handling rectangular regions. A disadvantage of bottom-up techniques, however, is that they may result in over-fragmented regions. For instance, a bottom-up technique will be more likely to properly segment small structures like individual paragraphs of text than to properly segment entire columns. Due to these trade-offs it is often that hybrid techniques, which combine top-down and bottom-up ideas, are employed [62]. Starting with top-down methods, variants of each broad category of physical layout analysis will be reviewed by this section.

Figure 21: An example of an arbitrary physical layout wherein blocks of text are fit to the shape of a pie chart. A layout analysis system should ideally be able to segment text blocks into the appropriate shape, which sometimes may be more complicated than simple rectangular layouts. For this figure, a document layout analysis system which can only handle rectangles would be insufficient, and would likely result in a mangled output. Image borrowed from [61].

Top-Down Physical Structure Analysis

The first physical layout analysis technique to be reviewed here is the “top-down” method. Top-down strategies segment blocks based upon interpretations of the document from a high level (i.e., by first looking at a representation of the entire document and recursively splitting it into smaller components). Top-down strategies will then typically attempt to verify each segmentation by visiting each node down the to the terminals (the lowest levels, corresponding to individual connected components or pixels) [63]. For documents having a complex layout, top-down methods are often more robust but slower than bottom-up ones. Typical bottom-up algorithms are faster, but can be less reliable since they may greedily over-segment blocks without regard to all of the available contextual information.

X-Y Cut Algorithm. The X-Y cut algorithm [64] is a top-down approach which has been utilized extensively over the past several decades. The technique analyzes

vertical and horizontal projection profiles of the image to find regions of low pixel density, often termed as “valleys” [46][47]. Assuming that the document has a white background and Manhattan layout, its X and Y valleys are likely to correspond with horizontal and vertical text block boundaries respectively. For instance, these could be divisions between paragraphs and columns. The X-Y cut algorithm will start with the horizontal and vertical projection profiles of the entire image and use the largest valley (or valleys) in either direction as the first splitting point. After having made the first split(s), the algorithm will then recursively make further splits within each sub-region using the same methodology. The document's physical layout is represented by an X-Y tree data structure wherein each node represents a split region. If the algorithm is correct, then the terminal nodes of the tree will correspond to the individual text blocks. Once the terminal nodes have been located, the algorithm will backtrack through the tree structure to ensure that the physical structure is appropriate based upon some preconceived notions of expected document structure. A possible result of the algorithm is illustrated by Figure 22. In order for the X-Y cut algorithm to work correctly, it is vital that the document first has its skew corrected. If, for instance, the horizontal projection profile is taken for a document that has been rotated by several degrees, then many of the “valleys” will not be found correctly and thus the algorithm will fail.

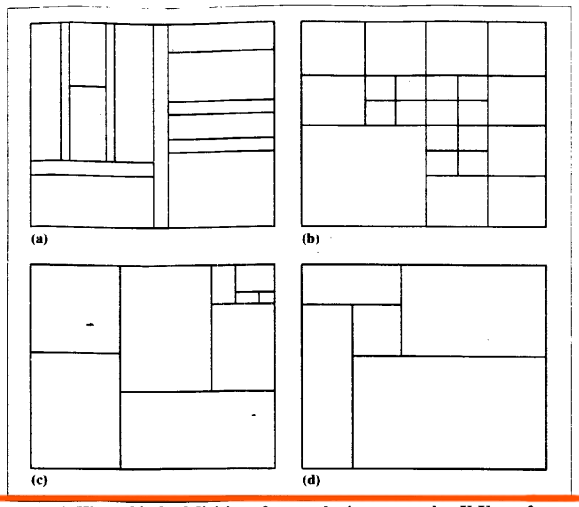


Figure 2. Hierarchical subdivision of rectangles into rectangles: X-Y tree for page segmentation (a); Quad tree for comparing or combining several images (b); K-D tree for fast search (c); example of tiling with rectangular blocks that cannot be obtained by successive horizontal and vertical subdivisions only (d).

to the various components to construe a valid graphic "sentence."

Although we use compiler tools developed primarily for formal languages, the syntactic analysis of document images exhibits many of the difficulties of parsing natural language. Layout conventions may be insufficient to identify *every* document component. For instance, text lines with equations buried in them may radically alter the expected line spacing. We must therefore ensure that minor deviations have only local effects. Furthermore, the grammar for a modern programming language is established from the start, while document grammars must be inferred indirectly, as later discussed. (A never-ending task: Journals frequently put on new faces.)

Block grammars. The document grammar for a specific journal consists of a set of block grammars. Each block grammar subdivides a block horizontally or vertically into a set of subblocks. The net result of applying the entire document grammar is therefore a subdivision of the page into *nested rectangular blocks*. Such a subdivision can be represented efficiently in a data structure called the X-Y tree[6] (Figure 2). The block grammars themselves are also organized in the form of a tree: The block grammar to be used to subdivide each block is determined recursively by the results of the parse at the level above.

Syntactic attributes

A (horizontal/vertical) *block profile* is a binary string that contains a zero for each horizontal or vertical scanline that contains only white pixels; otherwise it is a one.

A *black atom* is a maximal all-one substring. It is the smallest indivisible partition of the current block profile. A *white atom* is an all-zero substring.

A *black molecule* is a sequence of black and white atoms followed by a black atom. A *white molecule* is a white atom that separates two black molecules.

An *entity* is a molecule that has been assigned a *class label* (title, authors, figure caption). It may depend on an ordering relationship.

This approach effectively transforms the difficult two-dimensional segmentation into a set of manageable one-dimensional segmentation problems.

The syntactic formalism is theoretically well understood, and sophisticated software is available for lexical analysis and parsing of strings of symbols. Each block grammar is therefore implemented as a conventional string grammar that operates on a binary string called a *block profile*. The block profile is the thresholded vertical or horizontal projection of the black areas within the block. Zeros in the block profile correspond to white spaces that extend all the way across the block and are therefore good candidates for the locations of subdivisions.

Representing the structure of an entire page in terms of block grammars simplifies matters considerably. But each block grammar itself is a complex structure. It must accommodate many alternative configurations. For instance, to divide the title block from the byline block, the block grammar must provide for a varying number of title lines and bylines, and for changes in spacing caused by the ascenders and descenders of the letters. To simplify the design process, each block grammar is constructed in several stages, in terms of syntactic attributes extracted from profile features.[7]

Syntactic attributes. The first stage of a block grammar operates on the ones and zeros of the block profile. Strings of ones or zeros are called *atoms*. Atoms are divided into classes according to their length. A string of alternating black and white atoms is a *molecule*. The class of a molecule depends on the number and kind of atoms it contains. Finally, molecules are transformed into *entities* depending on the order of their appearance. The words *atom*, *molecule*, and *entity* were chosen because they are not specific to a particular publication or subdivision. (See the sidebar on syntactic attributes.)

The syntactic attributes that determine the parse are the size and number of atoms within an entity, and the number and order of permissible occurrences of entities on a page. Table 1 shows the expected variation in the horizontal profile of a page fragment that includes the title and byline. The assignment of symbols into larger units is accomplished by rewriting rules or *productions*. These

Figure 22: A possible result of the X-Y Cut algorithm on a page taken from [54]. Here the entire page is cut vertically (red) and then each sub-region is cut horizontally (green). The splitting order from this point becomes rather complex but is color coded as follows: orange, yellow, blue, and pink. Notice that a single node may have more than two children, which is the case for sections with multiple paragraphs, columns, etc.

Run-Length Smoothing Algorithm (RLSA). It is typically unnecessary to perform processing on all pixels of the document image. For the top-down algorithms previously described, which use either maximal white space rectangles or projection profiles, the document image is usually reduced in size during a preprocessing stage. By reducing the size and complexity of the input image, both the efficiency and accuracy can be enhanced assuming that only insignificant data is reduced. For instance, when detecting entire columns of text, the spacing between individual characters, words, and lines is unnecessary. One way to reduce the amount of data is to use a run-length smoothing algorithm (RLSA) [65] which will be discussed further in the Bottom-up Physical Structure Analysis section. This method can merge characters into words, words into text lines, and text lines into paragraphs by "smearing" the text to join characters into blobs. This is done by inspecting white spaces between foreground pixels and, if their width is below some threshold, setting them to black.

Template Techniques. "Template" techniques which have been observed in the literature [66][67], are labeled as top-down even though they often rely on a combination of both logical and physical document structure analysis [50]. These methods require a significant amount of knowledge about the expected document structure on which they are trained and may not generalize well to new types of documents. An effective way in which document structure can be described is through the use of a Form Description Language (FDL) [66]. The basic concept of FDL is that both the logical and physical structures of a document can be described in terms of a set of rectangular regions. The FDL specifies how a document should be processed based upon various aspects of its physical layout. Systems which utilize an FDL typically operate on a limited assortment of document types, thus its use is very application specific.

Dengel et al. present a technique which they call "Discriminating Attribute Values in uncertain Object Sets (DAVOS) [67]. By "object sets", Dengel et al. are referring to sets of regions on a document image along with their appropriate logical labels. The attributes (geometric features) of these objects may not be limited to single values but could cover a range of possible values and are thus considered as "uncertain." The DAVOS system analyzes business letters and builds a decision tree where each level corresponds with an increasing level of document type specificity. The terminals on the tree specify the entire logical layout of the document. Just as with FDL, the ability of the DAVOS system to generalize to new document types is

limited. DAVOS was only tested on business letters and was evaluated against a bottom-up technique (which utilized merging of connected components) and shown to have similar but “more balanced” results (i.e. logical labeling errors were more distributed among the various labels).

Bottom-Up Physical Structure Analysis

While top-down approaches start with the complete document image, repeatedly splitting it into smaller regions, bottom-up approaches carry out the inverse operation. Starting with the document image's primitives (i.e. individual pixels, connected components, words, etc., depending upon the application) bottom-up techniques repeatedly merge smaller regions into larger ones. While allowing more flexibility over top-down techniques, bottom-up techniques often result in greedy over-segmentation of regions. Bottom-up physical layout analysis techniques all utilize connected component analysis and may also make use of Veronoi diagrams, run length smoothing, mathematical morphology, neural networks [68], as well as communication theory (Document Image Decoding) [69]. This section will briefly review work that has been done for bottom-up techniques, starting with a discussion of connected component analysis.

Connected Component Analysis. As discussed previously, connected components are sets of foreground pixels such that a four or eight-connected path exists between every pixel pair in the the set. While text usually consists of connected components with a relatively consistent size and spacing, graphics generally tend to consist of larger connected components with more sparsely distributed positions. By analyzing these spatial properties of connected components, it becomes possible to identify and group text and graphics separately. Connected component generation involves grouping all four or eight-connected foreground pixels together in the document image. The components are then grouped based upon their bounding box location. The output of a connected component (cc) generation algorithm is a list of cc's where each entry contains the bounding box coordinates, shape of the region, number of black pixels, an image of the region itself, etc. The cc's are typically sorted by their bounding box position, and can be then filtered based upon height and width to determine regions more likely to be text vs those which are more likely to be graphics [46].

An example of a bottom-up physical analysis technique is Bixler et al.'s text extraction algorithm [70]. Bixler demonstrates his algorithm by extracting and recognizing the text from a map as shown in Figure 23. His technique first uses a

standard recursive (stack-based) flood fill algorithm in order to find the connected components [71]. After finding an initial starting foreground pixel, the flood fill algorithm can be described simply as follows: (1) If the current pixel is not foreground then return, (2) Set the current pixel to a replacement color in order to mark it as processed, store it in memory (3) Recurse to the function in each direction in turn (4) Return from the function. The aforementioned algorithm is then repeated for each unmarked foreground pixel of the image in turn, until all connected components are found and all marked pixels grouped into their constituent connected components are stored in memory, along with their bounding boxes, and any other relevant information.

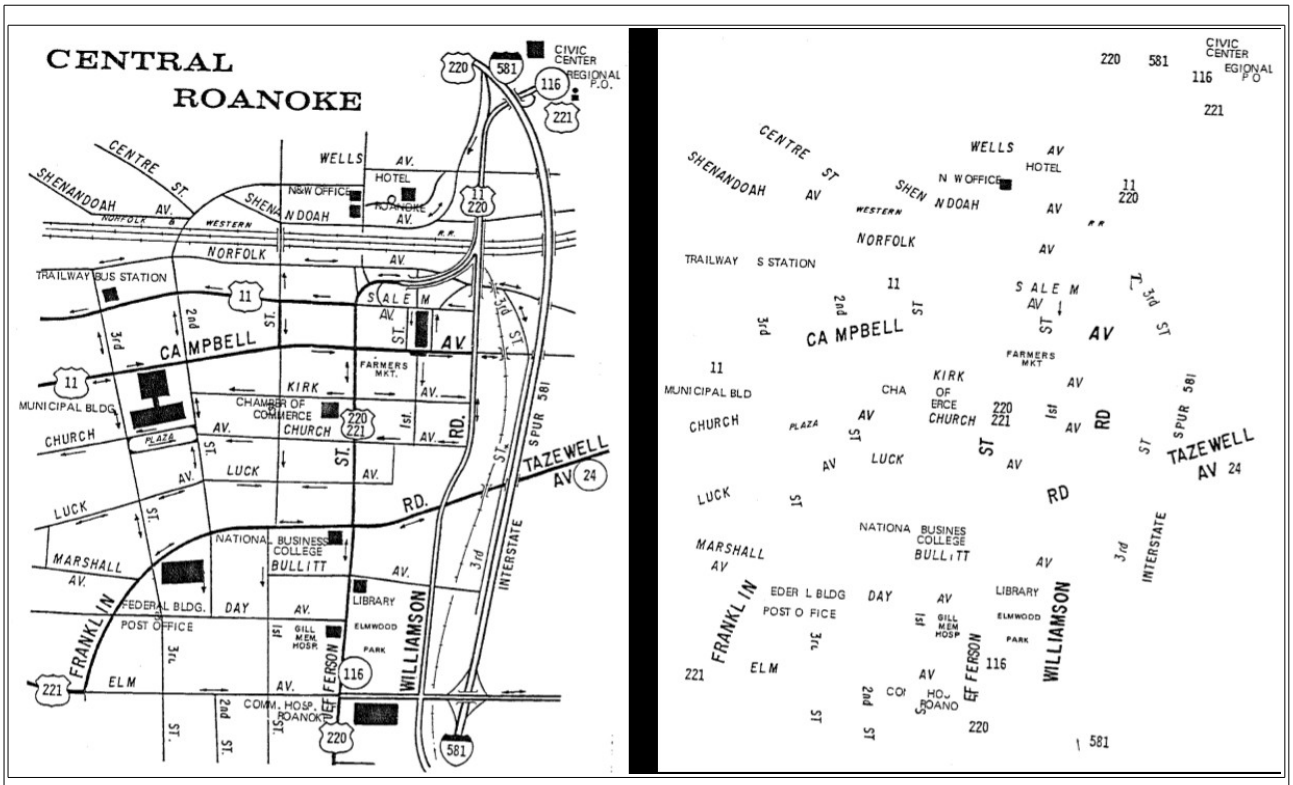


Figure 23: On the left is a map, and on the right is the map's extracted text. Notice there are some dependencies where the foreground text was confused with the imagery of the map. For instance one of the "f's" in the word "Post Office" is missing because it overlaps with a road [70].

With the connected components found, Bixler then determines which ones are text and which are graphics based on a simple height and width thresholding technique. Once the components have been segmented into text and graphics, those identified as graphics are subtracted from the image to leave only the text. The resolution of the image is then reduced based upon the size of the character

components. A connected component tracking algorithm is then utilized in order to find words which could be potentially in any direction (i.e. vertical, diagonal, horizontal, etc.). The algorithm scans the reduced document image from left to right, top to bottom looking for a starting connected component, then does a nearest neighbor search in each direction to find the closest character. Information about the spacing and direction between the first two characters is then utilized to track the location of the next character until entire words are detected. The procedure is repeated for each unique starting point until all words are found. The technique achieved near perfect results for a complex map, with only those words which significantly overlapped graphics being missed.

Document Spectrum Analysis ("Docstrum"). The Document Spectrum (Docstrum) proposed by O'Gorman [72], is a representation of a document which describes global structure features and can be useful for page analysis. The technique takes the document's connected components and utilizes a k-nearest-neighbor clustering technique in order to segment the document into words, text lines, paragraphs, etc. The algorithm recognizes five nearest neighbors for each connected component, where closeness is measured by Euclidean distance in the image. Each nearest neighbor pair is described by a 2-tuple, (d, θ), which is the distance and angle between the centroids of the two connected components. The "Docstrum" is the plot of (d, θ) for all nearest neighbor pairs in the image as illustrated by Figure 24. The text's spacing between characters and words as well as the line angles can be estimated by summing up the distance and angle values in the docstrum plot. The distances and angles are converted to respective histogram representations. The nearest neighbor angle histogram is smoothed and the peak found. The angle of the peak value gives a rough estimate of the text line orientation. This rough estimate is then used to determine intra-line and inter-line spacing by analyzing two histograms of the nearest-neighbor distance values. The first histogram is for intra-line spacing and filters out all distance values that are not within a tolerable range of the textline orientation estimate. This histogram thus represents the distribution of inter-character and word spacing within each text line. The second distance histogram filters out all values outside of a tolerable range of the textline orientation estimate's perpendicular. This histogram, therefore, represents the distribution of the document's inter-line spacing.

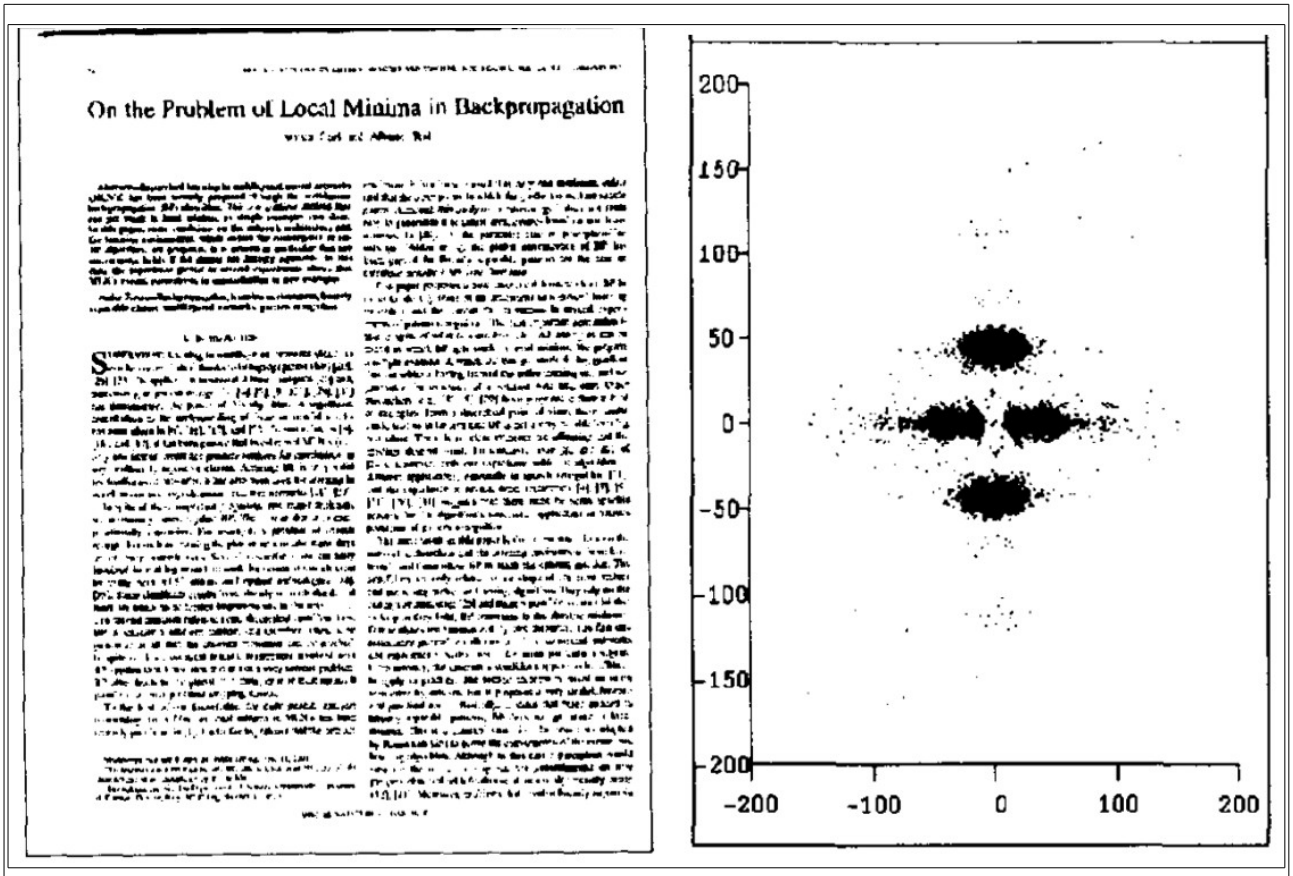


Figure 24: On the left is a document image and on the right is its corresponding Document Spectrum representation [72].

Nearest neighbors on each line are merged into words and then a regression fit is made to the centroids of the words in order to locate text lines. A straight line is fitted to the centroids in each group by minimizing the sum of square errors between centroids and the line. From these text lines a final estimate is made of the page's skew. An issue with this method is that text line descenders and noise could reduce the accuracy of the initial estimate and cause problems with reaching the right conclusions. It is important to have the correct threshold values and to smooth the histograms appropriately in order to get successful results. After the text lines are estimated, larger structures (like paragraphs or other text blocks) are then detected. The blocking technique examines pairs of text lines to determine whether or not they meet certain criteria to be considered part of the same text block. If the two lines are approximately parallel, close enough in perpendicular distance, and/or horizontally overlap to some degree then they are said to meet the criteria of belonging to the same block.

One of the benefits of this algorithm is that it does not assume that each component of the document has the same skew angle. Thus it is possible to indiscriminately segment lines and/or blocks of text in any direction. This may be useful for a variety of circumstances including analysis of magazines or journals with sporadically appearing vertical text, scans of several credit cards or business cards each on the same page but at arbitrary angles, maps with text overlayed over imagery in arbitrary directions, etc. The technique was tested on hundreds of scanned journal pages, however no comprehensive performance evaluation is given.

Voronoi Diagram. Given a set of points and a subset of these points called sites (or generators), the Voronoi diagram is the partition of the entire set into convex cells, such that each cell is the region consisting of all points that are closer to a particular site than to any other. Voronoi diagrams are among the most fundamental and well-studied objects in computational geometry [73]. An *ordinary* Voronoi diagram, as illustrated by Figure 25 [74], is one which uses Euclidean distance as its metric and can be described as the set of Voronoi regions which correspond to the convex shapes created by the partition.

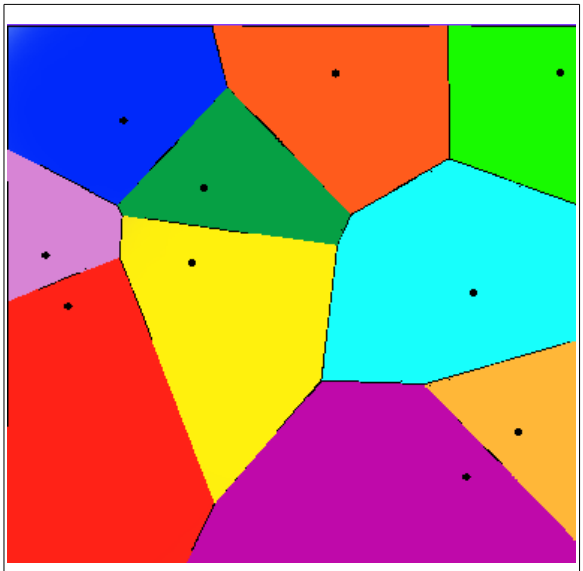


Figure 25: Illustration of an ordinary Voronoi diagram [74].

An *area* Voronoi diagram is a generalization of the *ordinary* Voronoi diagram depicted by Figure 25 which uses the Euclidean distance between the areas of connected components as a metric rather than the distance between points. The *area* Voronoi diagram for a document image can be found by the following procedure: (1)

Sub-sample every connected component in the image such that all that remains is a subset of the points on the outer edge; (2) generate an *ordinary* Voronoi diagram using this subset of points in the image; (3) remove all edges of the Voronoi diagram which both belong to points of the same connected component. This process is illustrated by Figure 26 [75].

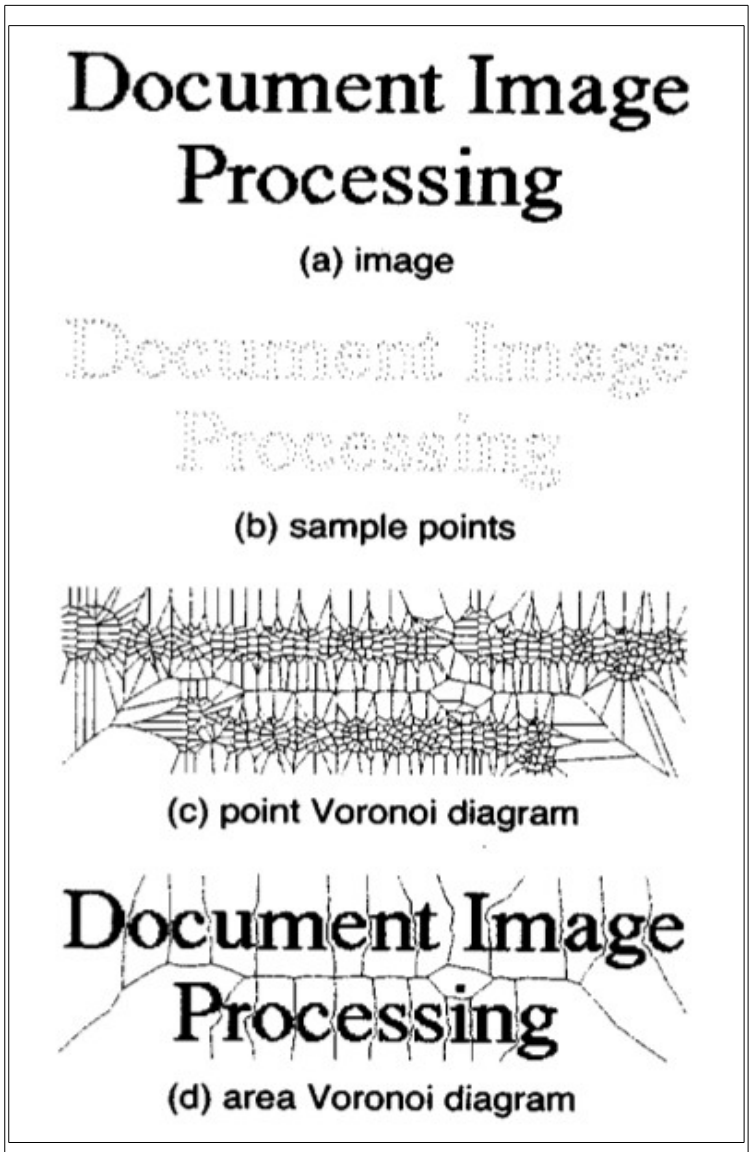


Figure 26: The process of segmenting an image into an area Voronoi diagram as illustrated by Kise et al. [75].

Kise et al. formulate the problem of physical page as that of determining which edges of a document's *area* Voronoi diagram best represent the boundaries of document components. By analyzing various features in the document image,

superfluous edges of the area Voronoi diagram can be removed, thereby leaving only the edges corresponding to document boundaries. Superfluous edges would, for instance, correspond to the space between characters, words, text lines, etc., when a division of the page into separate paragraphs, columns, imagery, title, etc., is required. For each edge, all of its line segments are evaluated in order to determine the minimum distance between the two points on the connected component which were used to generate the Voronoi line segments in the first place. If this minimum distance is below a given threshold for any line segment of the edge, then the entire edge removed. Likewise, the area of connected components are divided by these edges are compared and if the distance between the connected components is small enough in relation to the area ratio of the two connected components, then the corresponding edge is removed.

Kise et al. evaluate their algorithm on 16 document images at two resolutions, 90 DPI and 300 DPI, having a non-Manhattan layout each at 4 different skew angles to test for robustness (thus a total of 128 with non-Manhattan layout when counting the resolutions and skew). In order to test the applicability with Manhattan layouts, the algorithm was also evaluated on 98 images from the University of Washington database (UW1) all at 300 DPI. In evaluating the algorithm on these datasets, the percentage of the “body” text, “auxiliary” text, and “non-text” document zones which were over and under fragmented is evaluated respectively. The algorithm performed best on the body text of the non-Manhattan documents scanned at higher resolution where only 2.1% of zones were over-fragmented and only .4% under-fragmented. The algorithm faired poorly for the segmentation of non-text zones of all document types, but especially poorly for Manhattan documents where it resulted in a 98% over-fragmentation rate.

Run Length Smearing Algorithm (RLSA). Proposed originally in 1974 by Johnston [76] in order to separate text blocks from graphics, the Run Length Smearing Algorithm (RLSA) has been frequently used to obtain basic features for document analysis [46]. RLSA, in its most basic form, transforms a binary image as follows: (1) For each background pixel, if the number of neighboring foreground pixels is above a certain threshold, then the pixel is changed to foreground; (2) all foreground pixels are left unchanged. When applied horizontally or vertically to the rows or columns of an image respectively, RLSA has the effect of linking together neighboring background pixels that are separated by a number of pixels below the given threshold (illustrated by Figure 27 [65]). With an appropriate choice of threshold, it is possible for the linked

areas to correspond with separate document zones. The threshold is typically set based upon the character height, gap between words, and interline spacing [46].

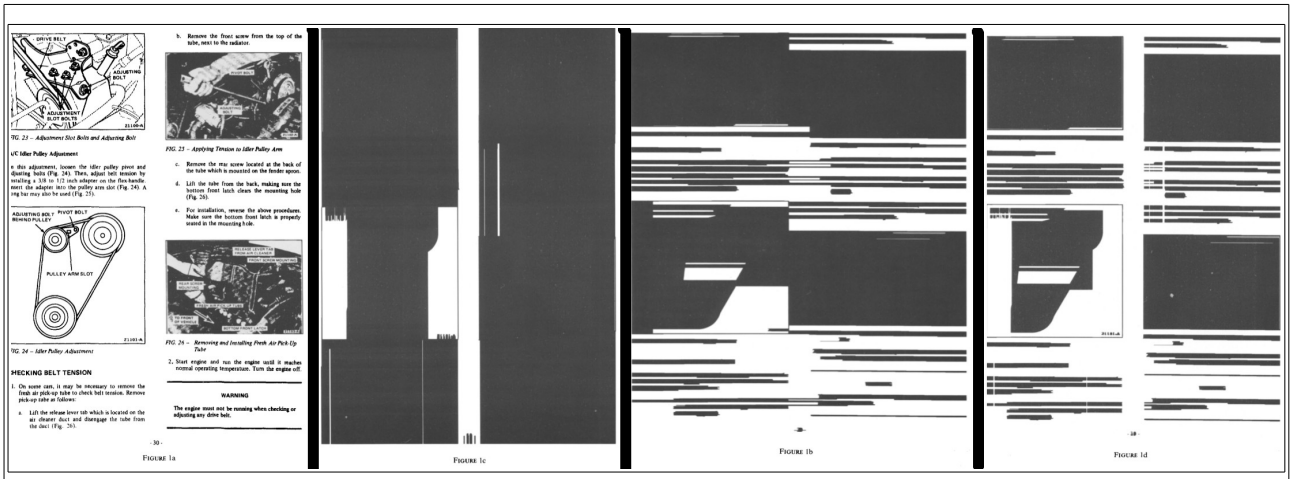


Figure 27: From left to right: (1) The original image. (2) Vertically run length smeared image. (3) Horizontally run length smeared image. (4) The logical AND of (2) and (3). [65]

Multiresolution Morophology. Bloomberg [77] discusses an approach to document image analysis which uses morphological operations at multiple resolutions. For an in-depth overview of morphological image processing (including definitions for terms such as structuring element, dilation, erosion, opening, and closing) the reader is referred to [78]. While connected component based techniques are effective on pages with only characters, they can exhibit practically unbound time and memory requirements when presented with pages consisting of halftones, graphics, and/or handwritten notes. While, in connected component analysis, a region's shape is primarily dictated by the configuration of its “ON” pixels, Bloomberg considers a region's shape based upon relationships between adjacent “ON” and “OFF” pixels and then considers texture to be the statistical distribution of such shapes in an image. Bloomberg describes a morphological image processing operation called the “generalized opening,” based upon the hit-or-miss transformation [79], which is useful for localizing shapes and textures of interest within an image. The rationale behind carrying out operations at multiple resolutions is that a single document image typically will contain shapes and textures of various sizes. While it may be more advantageous to process smaller regions at a higher resolution, larger regions generally require only a course view (lower resolution). Multiresolution image processing exploits such size differences such that regions can be processed in their most appropriate resolutions. Bloomberg describes a solution to the problem of half-

tone segmentation which closes the image (performs a dilation followed by an erosion) with a large structural element, followed by an opening (erosion followed by dilation) in order to only keep the half-tone regions while removing the text ones. Since using large structural elements on a high resolution image ends up being very costly, Bloomberg instead carries out the same functionality by carrying out a cascade of openings and closings using a 2x2 structural element while subsampling the image in between these operations.

Bloomberg further goes on to illustrate how multiresolution morphology can be used for the detection of italicized words (and also a separate technique for detecting bold words). To detect italics, Bloomberg looks for edges inclined at about 12° from the vertical. A 6x13 structural element is used which consists of 4 “OFF” pixels on top of one another aligned at an approximately 12° angle followed by 4 “ON” pixels in the same configuration. The aforementioned pixel sequences are separated by “don’t care” pixels in between and to the sides as shown in Figure 28.

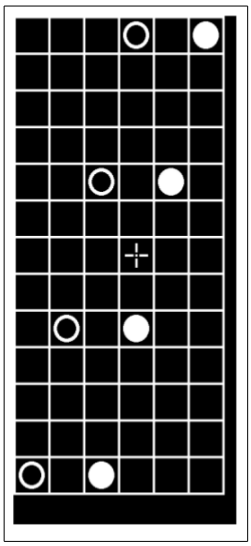


Figure 28: 6x13 structural element used by Bloomberg [77] in detecting italics. The open circles on the left represent “OFF” pixels, the closed circles represent “ON” pixels, and the empty squares are “don’t care” pixels.

As illustrated by Figure 29 (a), the “generalized opening” operation (hit or miss transform followed by dilation) is carried out on the image using the structure element shown in Figure 28 in order to find which regions of the text contain italics. The resulting “ON” pixels of the generalized opening are considered the “seed” pixels for italicized words. A closing operation followed by an opening as carried out on the seed

pixels in order to, respectively, merge the correct seed pixels, and then get rid of noise. After performing a small vertical dilation on the result, the final seed image, Figure 29 (b), is ready. Next, a word mask, Figure 29 (c), is created by sub-sampling the image by a factor of 4 and using a cascade of openings and closings as was done for the half-tone segmentation problem, followed by a small horizontal dilation. The final italics selection mask, Figure 29 (d), is then created by keeping only those words in the word mask which overlap the seed pixels in the final seed image, thus resulting in a mask which only keeps the italicized words in the image.

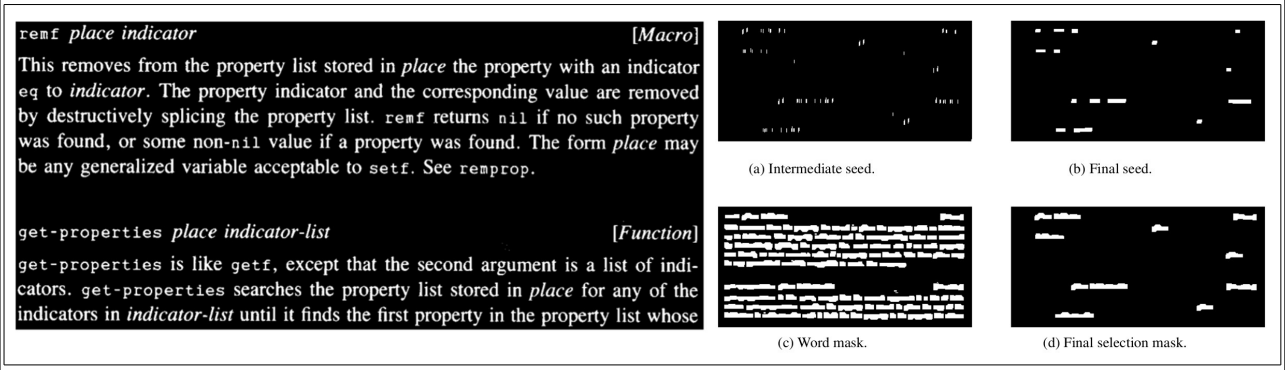


Figure 29: Bloomberg's italics detection [77]. Original text is on the left and on the right is (a) the intermediate seed image, (b) the final seed image, (c) the word mask, and (d) the final italic word selection mask for the image.

Hybrid Physical Structure Analysis

Hybrid structure analysis can simply be regarded as any mixture of the top-down and bottom-up approaches previously described. Several of the hybrid segmentation algorithms found in the literature utilize a combination of "splitting and merging" strategies [62] [80] [81]. Such algorithms will begin by carrying out a top-down methodology in order to first split the page into regions which appear homogeneous, usually based upon horizontal or vertical projection profile measurements. Liu et al. [80], for instance, utilize an algorithm which operates as follows: If a region is found to be in-homogeneous based upon certain critieria then it is split into 4 rectangular sub-regions using an adaptive thresholding technique to choose the line positions. If two regions within the previously split region are then found to be homogeneous based upon the same criteria then they are merged. This is continued recursively until there are no more splits and merges to be made. There are also various hybrid techniques which operate based upon measuring the space

between text areas, referred to as “maximal white space calculation.” Such techniques such as those of Okamoto et al. [82] and Bruel [83] are derived from Baird’s “Shape-directed Cover Algorithm” [84]. This subsection will first discuss Baird’s shape directed cover algorithm as well as a newer algorithm which is in the same spirit. Two notable open source systems which perform document layout analysis are Tesseract [18] and OCR-Opus [85]. Both systems utilize a combination of bottom-up and top-down physical layout analysis techniques and will briefly be discussed as well.

Shape-directed Cover Algorithm. In an attempt to combine the strengths of top-down and bottom-up methods (i.e. faster run time for the more greedy bottom up methods but more global knowledge for top-down), Baird et al. [84] proposed a “global-to-local” strategy which first finds the rectangular coordinates of all foreground connected components and then finds all of the maximal white space rectangles surrounding them. A white space rectangle is considered maximal if it contains only white pixels and cannot be further expanded while staying entirely white. The white space rectangles are then sorted into a binary tree structure where the right-most white space rectangles are at the root, and the left-most are the leaves. Multi-way branches which occur when there is more than one maximal white space rectangle at a given X coordinate, are handled using singly linked lists as entries in the binary tree. Unlike most of the previous top-down physical layout analysis research, Baird focuses intently on algorithmic complexity. When he denotes the number of maximal white space rectangles as m and the number of foreground rectangles as n, he found his algorithmic complexity for sorting the white space rectangles to be $O(n \log n + m)$.

Once the rectangles are sorted, a subset of these rectangles denoted as the “cover set” is chosen. Any regions of the image not covered by the union of this cover set will define the segmented text blocks. In order to speed up processing time, the rectangles in the cover set are chosen in a greedy fashion using the high level information available in the binary tree. In terms of processing speed, this can prove advantageous over the X-Y Cut algorithm which uses extensive backtracking. The cover space is chosen based upon domain specific information. For instance, in Manhattan layouts, the white space rectangles between columns will typically have a high (but not too high) aspect ratio. Baird et al. thus assigns shape scores to the rectangles in order to favor the most significant and choose the cover space based upon these scores. Experiments were run on over 100 Manhattan layouts which included typewritten and printed pages from letters, magazines, books, journals, and newspapers, which included complex layouts consisting of headers, footers,

embedded mathematical equations, graphs, multiple columns, etc. The authors reported near perfect results for large column structures but would observe errors for smaller blocks of text especially in the presence of noise.

White space cover algorithm by Breuel. Breuel presents a variation of Baird's white space analysis algorithm which is simpler to implement (requires less than 100 lines of Java code) [83]. The algorithm starts by picking one of the black rectangles, called the “pivot”, toward the center of the image. Since the maximal white rectangle cannot contain the pivot, there are now four distinct possibilities for the maximal rectangle's location: above, below, to the right, and to the left of the pivot. Each sub-rectangle is then evaluated using a quality measure to determine which is most likely to contain the maximal rectangle. After the sub-rectangles and their respective quality measures are inserted into a priority queue, the above steps are repeated. This process continues until a fully white-space region is detected. The rectangle corresponding to this region is the optimal solution. The results of this algorithm were described as favorable when run on the same dataset as Baird (the UW3 Database [86]), with no errors observed on 223 pages. An in-depth evaluation, however, was not provided. Figure 30 illustrates a more recent technique called “the White Space Cuts Algorithm,” [87] which combines Baird and Bruel’s approaches.

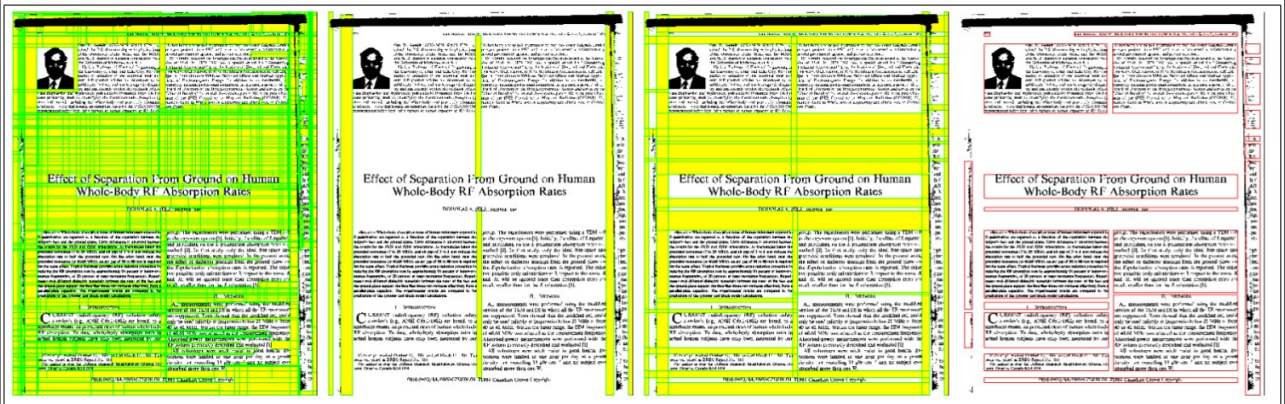


Figure: 30. An example image illustrating different steps of the whitespace-cuts algorithm [87]. Left to right: whitespace cover of the page background, extracted vertical separators and borders, extracted horizontal separators, extracted page segments.

OCRopus Open Source OCR System. Bruel, who was discussed previously for his novel variation of Baird’s white space cover technique, is the project lead for OCRopus, an open source Google-sponsored project which addresses various problems in Document Analysis through the use of large scale machine learning. All of the project’s

modules are written in Python and the project emphasizes modularity, easy extensibility, and reuse. OCRopus is aimed at both the research community as well as large scale commercial document conversions [85]. The system includes overridable modules including, but not limited to, noise removal, skew detection, text/image segmentation, layout analysis, textline recognition, optical character recognition, and statistical language modeling. For more details on the architecture as well as algorithms implemented by this system, the reader is referred to [88].

Tesseract Layout Analysis Module. Tesseract, another Google-sponsored project as described earlier, utilizes a hybrid page layout analysis algorithm [18] which starts by utilizing bottom-up techniques in order to locate “tab-stops” on each text line. These “tab-stops” can represent the left and right edges of columns at that particular vertical location of the page. Each left and right tab stop is connected to form a “column partition,” a vertical slice of a column at the given text line. A “column partition set” is the group of all “column partitions” at a vertical position of the page (i.e. stretching from the left of the page to the right). The column partition sets are then iterated in order to derive the column structure that makes the most sense for the entire page. Once the column structure of the page has been derived, this structure is applied in a top-down fashion in order to derive the page’s reading order.

Document Logical Structure Analysis

While the physical layout analysis step of a document analysis system generally divides an image into areas of text and non-text while determining an initial estimate of the page’s basic columnar structure, the logical layout analysis step will further investigate the resulting structure in order to determine where splits or merges may need to be made based upon the perceived syntactic meaning of the document’s components. The document analysis system may then iteratively transition back and forth from physical to logical analysis based upon further document understanding until some criteria is met. Once all of the document’s components have been fully classified and segmented, the result of the logical analysis will be an increased understanding of the page’s components. The page may, for instance, be composed of the chapter name at the very top, a page number to the top right, and several columns of text. The columns may consist of imagery, half tones, mathematical equations, block quotations, as well as various other components.

In the literature, document logical layout analysis research can be most broadly split into three main categories: (1) type-specific detection, (2) zone classification, and

(3) page classification [89]. Type-specific logical layout analysis techniques, which are the primary focus of this thesis, emphasize the use of separate algorithms to detect possible components (i.e., text, image, math, half-tone, chemical equation, etc.) and make no assumptions about whether or not these components have already been correctly segmented. The input to a type-specific logical analysis algorithm may, for instance, consist of a single block which should really be logically separated into multiple separate blocks, and/or various blocks which actually need to be merged. Zone classification techniques, on the other hand, assume that the document has already been properly segmented into type specific zones and all that needs to be determined is what these types are. Such techniques will extract features from the zones and then use these in order to directly classify the zone type as one of a finite set of types (i.e., normal text, mathematics, imagery, etc.). Page classification layout analysis research is geared toward classifying an entire page based upon the type of its content. A page may, for instance, be categorized as a title page, table of contents, appendix, glossary, regular page, etc.

Although page classification is an important component of document understanding, such a region-wide generalization should only be made after an in-depth analysis of the page is carried out to gain an understanding of all of its zones. It would not make sense to segment the page into logical zones based upon an ill-fitted estimate of what contents the page is expected to have. Likewise, zone classification may be an important step in logical layout analysis but it misses the important point that no physical layout analysis technique done prior to logical analysis is perfect. A full understanding of the under and over segmentations made by the initial physical layout analysis algorithm may rely upon the type of content in question. If, for instance, a table element is detected by the logical analysis module, it may make the most sense to then check for oversegmentations that could have been made by the physical analysis module. All of the elements that are clearly part of a larger table should be merged such that the table is then segmented appropriately. Likewise, there may be a block of text which has actually been undersegmented by the physical analysis module. An example of an undersegmentation is seen when there are inline mathematical expressions within a paragraph of text. While the entire paragraph may have been correctly segmented by the physical layout module, it is very important for the mathematical expressions within it to be segmented from the normal text in order to avoid subsequent recognition errors. In such an instance, it is then the logical layout analysis module’s job to detect candidate mathematical expression regions and then

utilize the appropriate physical layout techniques necessary to segment them from the normal text.

Since proper physical segmentation and page classification are dependent upon the type of content in question, type-specific detection is the primary focus of this thesis. After a brief overview of some of the page and zone classification techniques found in the literature, type-specific detection techniques, the focal point of this work, will be discussed in greater detail. The focus will be primarily on type-specific detection of mathematical expressions. Detection of other zones such as tables, logos, and music scores will also be briefly discussed.

Page Classification

The motivation for page classification research is two-fold. Firstly, it is important in facilitating faster document processing. If a page's specific type can be known then the corresponding type-specific layout analysis techniques may be employed to reduce processing time. Secondly it is important to facilitate faster indexing of page types. For instance if the title page is known, it will be very fast to do a document wide search to find the author of the work. When a user is searching for a specific document, knowing the class under which the document resides may allow for a quicker and more fruitful query experience. Page classification techniques found in the literature utilize a wide range of feature and classifier types in order to categorize an entire page. Although some methods utilize the output of a commercial OCR engine [90], the majority of techniques only use features taken directly from the document image [91] [92]. Page classification can be carried out at various stages in the document layout analysis process. The stage at which a final decision regarding the page type is made may vary based upon the type of document fed into the system, its physical and logical layout, etc. For an in-depth overview of how the problem of page classification has been previously approached, the reader is referred to [93].

Zone Classification

Zone classification techniques are employed in order to logically label regions independently of physical segmentation. Such techniques operate under the assumption that whatever physical layout technique was carried out prior to logical analysis has already properly segmented the page into logically independent zones (i.e., normal text, equation, table, image, etc.). In the literature these techniques vary based upon feature extraction methods, classification techniques, and the number of zone types to choose from. While earlier works [94] [95] may only distinguish between

2-3 zone types, more recent work [96] [97] is observed to distinguish a wider variety of zone types (i.e., 9-10).

The work done in [95] employs features based on the zone's spatial distribution of pixels to train a binary decision tree classifier that distinguishes text from non-text. Fan et al. [94] use pixel density features to first segment text from non-text and then use a “pixel connectivity histogram” in order to classify nontext as either photographic imagery or vectorized graphics. The pixel connectivity histogram takes into account every foreground pixel of the given zone, measures the number of other foreground pixels connected to it (the connectivity measurement), and gives the number of such foreground pixels that fall under each connectivity measurement found. The classification algorithm utilized was not specified by this work. Wang et al. [96] use a 25 dimensional vector composed of run-length, spatial, and background features in order to train an optimized decision tree to classify a zone as one of nine zone types. Abd-Almageed et al. [97] extract features of the zones based on the run-length and spatial distribution of foreground pixels. The partial least squares algorithm is then carried out on these features in order to reduce their dimensionality. A novel hybrid classification approach which combines the benefits of a one-against-all classification scheme with those of a one-against-one scheme is used to determine the zone type from the reduced feature space. An SVM is used as the underlying binary classifier. Zones are classified into one of 10 logical types (chemical drawing, small text and symbols, drawing, halftone, logo or seal, map, math, ruling, table and large text). Both [96] and [97] are evaluated on the University of Washington III (UWIII) data-set [86].

Type-specific Classification

Type specific classification techniques make no assumption about the accuracy of any physical segmentation carried out prior to logical analysis. Thus a type specific technique will, not only detect the type of a given region, but also choose what further physical layout analysis may be required in order to ensure that the given region is properly segmented (i.e., table regions, for instance, may require a different segmentation technique than what would be required to segment normal text regions). The primary focus for this thesis is in type-specific classification: specifically, the proper detection and segmentation of mathematical equations. After an in depth overview of mathematical equation detection and segmentation techniques found in the literature, type-specific classification of other types will also be very briefly discussed.

Mathematical Expression Detection

Only a dozen independent studies related to the type-specific segmentation of mathematical expressions from document images were observed in the literature [98][99][100][101][102][103][104][105][106][107][108][109]. Although the research of math detection in document images may be largely uncharted, it is no mystery that there are a wide variety of mathematical expressions prevalent in text books, journals, and technical papers. Such documents are often desired to be viewed through portable devices, desktop computers and/or screen reading software for general convenience as well as assistive technology purposes. Even for “digitally-born” PDF documents, it is rare that mathematical regions of text end up being properly viewable by most software. For scanned documents this problem is even worse. Most commercial OCR modules employed typically have no understanding of mathematical expressions and become confused by their presence. A common result of the presence of mathematical expressions during OCR is garbled output, not only of the mathematical expression regions, but even nearby normal text regions which would otherwise be recognized correctly. One of the early motivations for mathematical expression detection was, not necessarily to properly recognize these regions, but rather to ensure that their presence does not reduce the accuracy of normal commercial OCR software [104].

While there has been a relatively small amount of work found in the literature geared toward the logical and physical segmentation of mathematical regions, the OCR of these regions is a relatively mature field of study [110][17]. In mathematical OCR, however, it is typically assumed that all regions are perfectly segmented prior to recognition, either manually or automatically. In the literature, these regions are most often manually segmented prior to evaluation such that the recognition problem is evaluated independently of the segmentation problem. The only commercial mathematical OCR software found in the literature to date, “Infty” [111], implements an expression detection module, however no detailed evaluation of it is provided [100]. Their most thorough evaluations are thus carried out on regions which were manually segmented beforehand. Since the type-specific logical and physical layout analysis of mathematical expressions is a largely uncharted area of study while mathematical OCR has been studied extensively, the primary focus of this thesis is on layout analysis rather than recognition. While mathematical OCR is certainly a very important processing step, it is outside of the scope of this work. For a recent in-depth literature survey on mathematical OCR the reader is referred to [17].

Literature Review

Before discussing the existing literature on expression detection, it is important to first specify some common notation and practices. Mathematical expressions found in printed text can either be located on a separate line from normal text or be mixed in with the text. Expressions falling under the first category are commonly termed as either displayed/isolated by the literature while those which fall under the second are termed as embbeded/inline. Expression detection techniques most often operate in several steps referred to as passes. The first pass often consists of locating initial expression candidates referred to as seed regions. These regions are then either removed or grown in further passes based upon various heuristics. The term, “digital-born document” refers to a document which was created directly from a computer rather than being scanned in. The dozen independent studies found in the literature each consist of either one or more conference or journal papers and will be briefly reviewed in the chronological order of their first publication.

Lee et al. [99] (1995). While this study is primarily geared towards mathematical expression recognition, the system also included an expression detection module. Bayes decision rules are employed in order to locate displayed expression regions. If a displayed region is detected then the entire line on which it resides is labeled as an expression region. No detailed analysis of the detection accuracy is provided. Similar work was also later carried out in [112].

Inoue et al. [100] (1998). An early study carried out by the authors of the “Infty” commercial OCR software, this work explains the software's underlying expression segmentation module. The module recognizes normal text and segments it from mathematical text in the same step by using information obtained from a commercial OCR engine along with a dynamic programming algorithm. Experiments are run on 50 pages of Japanese text, of which detection errors are reported to have occurred on every page. No thorough evaluation is provided.

Fateman [101] (1999). The technique described operates in three separate passes and includes an interactive system which allows the user to manually correct any segmentation errors which may have been made. During the first pass, each connected component in the image is separated into one of two “bags”: one for normal text, the other for mathematical text. The math bag initially contains all italicized letters, Roman digits, punctuation, special symbols, and horizontal lines. These are considered as the seed regions. The second pass will then group the math-bag components into zones and, according to horizontal and vertical proximity, grow the seed regions by relabeling nearby text components as belonging to the math bag.

On the third pass, remaining punctuation connected components in the math bag that are still isolated are moved to the text bag. Remaining isolated Greek letters, Roman numbers, etc. are kept in the math bag. Text components that are close in proximity to the math bag, and could be considered as math such as “sin”, “cos”, etc., are moved to the math bag. Finally, the results are then shown to a human for interactive editing and correction.

Toumit et al. [102] (1999). A specialized top-down physical segmentation technique operating on the entire image with image reductions to segment math regions is briefly described but no specific details are provided. Displayed expressions are located under the assumption that they are always centered and on their own line. No further specific details are given. Embedded expressions are located by first finding special characters (i.e., “+”, “=”, “>”, etc.) and propagating around these using rules specific to the given symbol. Various concepts and heuristics are defined and utilized for mathematical expression detection, primarily: atomic structures are single mathematical symbols; composite structures are logical groupings of atomic structures; implicit structures have no graphical representation (i.e., the multiplication operator when representing the multiplication of a with b as simply ab). To represent mathematics, a tree structure is used which allows each node to have more than two children. It is argued that mathematics is not inherently a binary recursive data structure, but simply a recursive one. While $a+b+c$ can be represented by a binary tree, matrices, integrals, and vectors cannot necessarily be represented in this way. No detailed evaluation was carried out in this work.

Garain et al. [103] (2000). Four relavent works from Garain and his advisor, Chaudhuri, will be herein briefly reviewed. Garain's earliest technique first segments all text lines (this includes those of displayed expressions) by measuring horizontal projection profiles and denoting the boundary between two lines as local minima of these profiles. Next, each text line is separated into its constituent connected components. The mean and standard deviation is calculated for the distance of the bottom of the text line to the bottom of each connect component. Since math expressions may contain elements whose distance from the baseline varies more than normal text, this metric can, in some cases, be very helpful. If the standard deviation is above a predefined threshold then the line is expected to contain an expression. If such a line is also observed to be vertically separated from normal text lines significantly then it is labeled as a displayed expression region.

Embedded regions are then found by first looking at remaining normal text lines to find mathematical characters (i.e. “+”, “=”, etc.). When such characters are found, they are considered as seeds for the expression region and are recursively merged with their neighbors based upon the following criteria: (1) if the seed region is just a binary operator then the immediate left and right “words” are merged with the seed operator, (2) “words” adjacent to the seed region on the immediate left and right are merged if they contain one or more mathematical symbols, superscripts/subscripts, single dots/ellipsis, or numbers. Their detection algorithm is tested on 120 pages containing a total of 140 mathematical expression zones. Of these 120 pages, 20 are taken from the UWIII dataset. 132/140 expression zones were properly detected, with eight of them being entirely missed and three entirely false detections. No partially correct detections are presented in the results (i.e., either the detection is completely correct or completely wrong based upon their evaluation technique).

In 2003 a morphological technique was proposed by Chowdhury [113] in order to segment displayed expression regions from all other regions in the text. A morphological approach is first carried out to segment table, text, and graphic zones. Further segmentation is then carried out on the result in order to find the displayed expression regions. Such regions are categorized into the three basic types: those which contain large horizontal lines, those which contain a long vertical separator (as is often seen in matrices or determinants) and all others. In order to segment the regions which fall under the “all others” category, the page is first closed with a horizontal structuring element in order to locate the text lines. Subscripts and superscripts (localized based on proximity and size relationship) are associated with their text lines. Features such as number of subscripts and superscripts, vertical overlaps, presence of tall symbols, and horizontal positioning of connected components are used by a decision tree classifier to locate the displayed regions. Similar techniques are used to localize regions containing horizontal or vertical lines. The techniques are evaluated on a set of 197 images. While roughly 97% of the displayed regions were reported to have been correctly segmented, no measurement was given of false positives. The technique was tested on embedded expression regions and found to have a 68% true positive rate. Again, no indication was given of false positive rate.

In 2004, Garain and Chaudhuri propose a technique for segmenting embedded expressions from document images [114]. n-grams are utilized in order to spot sentences output from a commercial OCR that are likely to contain embedded

expressions. Sentences containing phrases like “such that”, “note that”, “denote”, etc., were shown to have a higher probability of containing embedded expressions than those which did not. A dataset containing 400 scanned pages of scientific documents including various science books, journals, conference proceedings, etc., is utilized by this study for evaluation [115]. Words recognized by the commercial OCR are evaluated as possible embedded expression candidates based upon the probability of their sentence to contain embedded expressions based on n-grams, the commercial OCR's confidence rating for constituent letters within the word in question, italic/bold/normal type style detection, inter-character spacing within the word in question, and variance of the bottom y coordinates for the constituent symbols within the word in question.

Experiments were carried out on the 400 scanned pages from the dataset which contained over 3000 embedded expressions. Evaluation includes a count of the true positives, false negatives, and false positives. Also included in the evaluation are partially recognized regions. The evaluation scheme strives to combine all of these measurements into one single metric/score. Partial recognitions are weighted based upon how many components were supposed to be identified as math in the region vs how many were actually identified[2]. The false negative count is weighted by zero for some reason, and thus does not factor into the score. Based upon their scoring technique, ranging from 0 to 1, an average score of .963 was obtained for all 400 pages.

In 2009, Gerain experiments with methods for detecting both displayed and embedded expressions in document images [116]. Gerain approaches the problem by first extracting features for displayed and then embedded expressions from the document image and then experimenting with various averaging techniques on the respective features in order to see which has the best discrimination power for the given problem. Features are first extracted in order to classify an entire line of text as either being a displayed expression or not. In this study, the tendency of displayed expressions to also contain normal text seperators which don't belong to the expression (i.e., commas, periods, phrases like “and”, “therefore”, etc.) is not accounted for. The features used to detect displayed expressions include a measurement of the vertical space above and below the text line in question, the vertical scatter of the bottom y coordinates of the connected components on the line

[2] This may prove problematic since components with a large number of pixels will be weighted just the same as components that are very small. For this reason, pixel-accurate methods of evaluation are utilized in this work, as will be explained in a later section.

in question, the pixel height of the text line in question relative to the average pixel height of all the lines on the page, and the number of mathematical symbols on the text line. Each of these features is normalized to a value between 0 and 1 by using the following exponential expression, $1 - e^{-x}$, where x is the feature value. The exponential allows for slight changes in the quantities being measured to have a large impact on the feature values.

After text lines are labeled as either displayed or normal, embedded expressions are then sought out for the remaining normal text lines. Individual words within sentences are classified as either displayed or normal based upon the following features. As in the previous study, linguistics is incorporated in order to detect sentences which are likely to contain embedded expressions. Other features incorporated for embedded expression detection include the commercial OCR confidence rating of each word of the sentence, the typestyle of the given word (i.e., italic, bold, etc.), the scatteredness of the connected components within the word about the textline, and the average horizontal gap between characters within the the word in question. After normalizing these features, they are used to classify every word of a sentence as either normal text or embedded expression text[3]. The features for displayed text lines are each combined into a single scalar value through one of the following averaging techniques: arithmetic mean, geometric mean, harmonic mean, and weighted mean. Lines are chosen as displayed text or not by comparing the resulting feature value to a scalar feature value found empirically which varies depending upon the averaging technique used. A very similar approach is used to detect words segmented by the commercial OCR engine that are embedded expressions.

Experiments are carried out on 200 scanned pages. 150 of the pages were taken from Garrain's corpus [115] and the other 50 were taken from the INFTY database [117] which only contains manually segmented displayed expression images. Training to determine thresholds is carried out on 50 of the images, and evaluation carried out on the other 150. Tests using the weighted average method wherein the weights are determined through a gradient descent algorithm showed the best results. Using their specialized efficiency metric which takes into account false

[3] Garrain relies on commercial OCR engines to segment his words within sentences, prior to classification. He makes no corrections to improper segmentations made by the commercial OCR. Thus in the situation where a word is improperly segmented by the commercial OCR as containing part of an expression and part of a normal text all in one block, Garrain's algorithm will always either result in false positives or false negatives.

negative, false positives, true positives, and partial recognitions, [114] a score of 87% is achieved for embedded expression extraction while a score of 88% is achieved for displayed expression extraction.

Kacem et al. [104] (2001). As illustrated by Figure 31, the primary motivation of this work is not to properly segment all mathematical regions for mathematical recognition purposes, but rather to only segment those regions which may interfere with a normal commercial OCR engine (i.e., that could result in errors). The proposed technique identifies various mathematical symbols in the document without the use of any commercial OCR engine. These symbols include product, summation, integrals, roots, fraction bars, large brackets (i.e., that surround horizontally overlapping expressions on different lines), small delimiters (i.e., normal paranthesis/brackets), and binary operators such as plus, subtraction and equals. These symbols are identified based on a measurement of their connected component's aspect ratio, area, and pixel density. Using a training set which contains various appearances of these symbols in printed text, a histogram is created for all measurements in order to know their distribution. Based upon these histograms, upper and lower bounds are set on these measurements for each symbol. When identifying a new connected component, a label is assigned to it based upon the intersection of all three measurements. Rather than doing an immediate binary label based upon this information, a “membership degree” is assigned to the connected component for each of the possible symbol types. If any of the membership degrees are within the upper and lower bounds, the symbol type with the highest degree is assigned to the connected component. This method was evaluated on a test database of 460 mathematical symbols and 95.3% of connected components were found to be well-labeled. It was not indicated whether or not there were false positives.

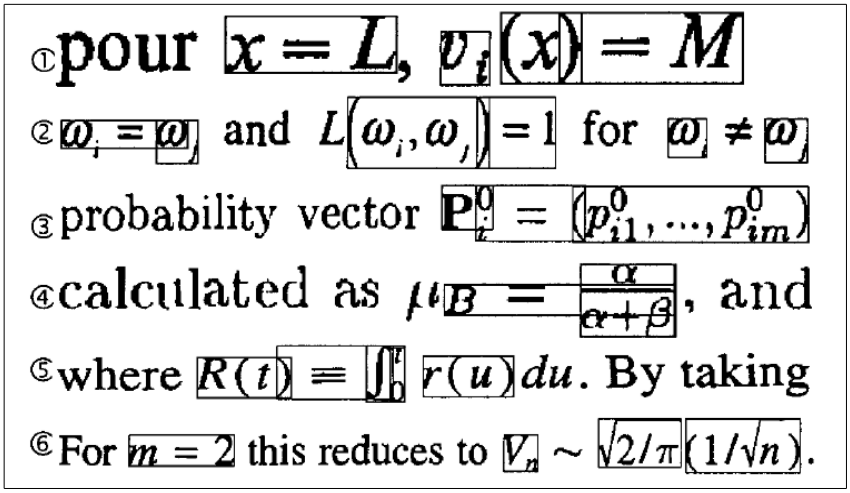


Figure 31: Regions detected by the method of [104] *as belonging to expressions are shown above bounded by rectangles. Note that, although most of the regions are over-segmented with various symbols being missed altogether, subtraction of the above labeled regions will result in improved accuracy for most commercial OCR engines which would otherwise be confused by the presence of the various mathematical expressions.*

Once the connected components are labeled as indicated above, the text lines are determined by grouping all the connected components based upon proximity. The specific algorithm used is not specified. Math symbols found within text lines are often used as heuristics to dictate whether or not lines should be merged. For instance, in the case of a large fraction bar, it is clear that there should be both a numerator and denominator. This may require vertically merging part of the two text lines together. Once the lines are extracted, their aspect ratios and position in relation to other lines is measured in order to determine whether or not they are likely to be a displayed equation. Further measurements are then made on each connected component's vertical position within its corresponding line and its height in relation to the average connected component height for the line. For every line, all of the connected components are labeled as one of the following topography features: overflowing, ascending, descending, centered, high, or deep as illustrated by Figure 32.

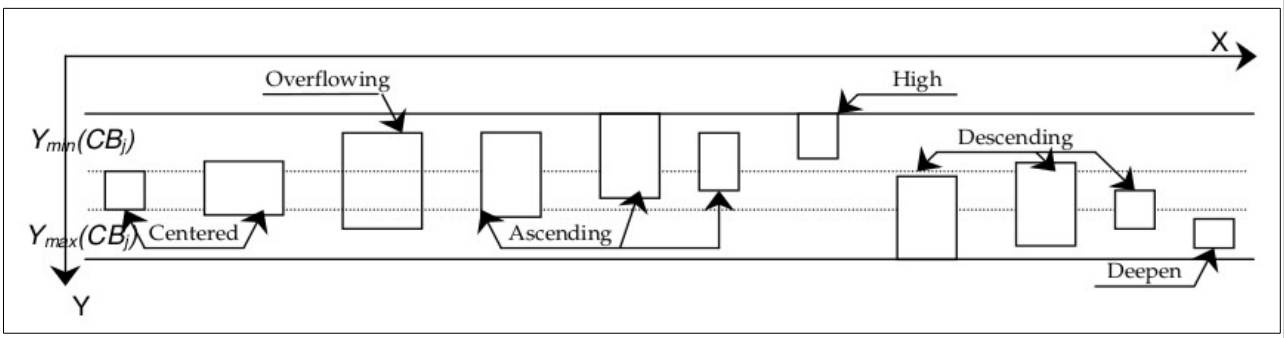
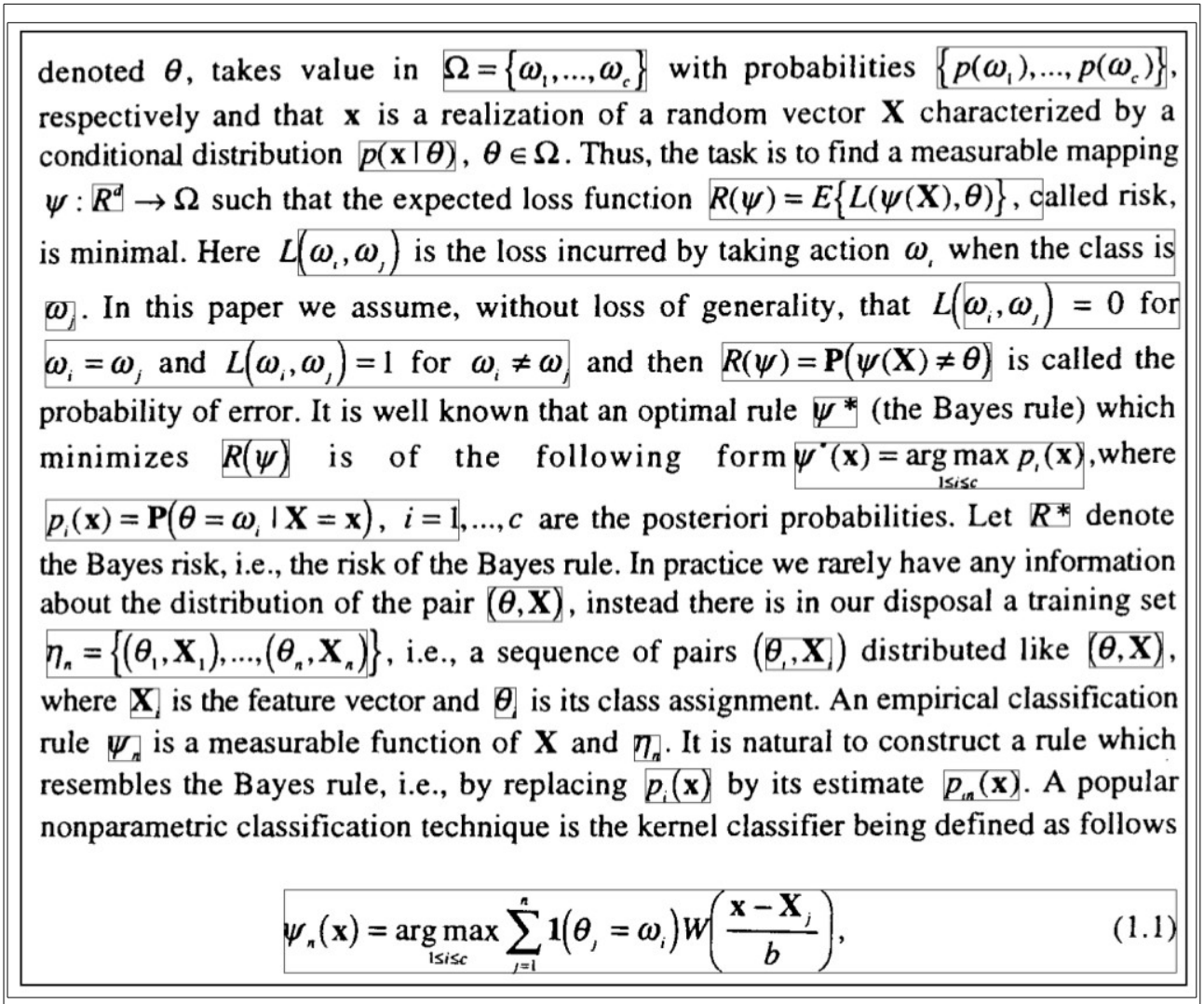


Figure 32: A connected component's possible topography features on text line j based upon vertical location in reference to line j's upper and lower central bands [104].

The topography features are used to help determine the type of a symbol. For instance subscripts would be descending or deep, while superscripts would be ascending or high. Radicals would be overflowing, and fraction bars would be centered. Subscripts and superscripts are found by comparing the relative size and position of two adjacent connected components. Training is carried out on these measurements using a histogram approach similar to the one described for math symbol identification. Next, rule-based context is propagated based upon the specific math symbols in question (for instance, each connected component inside a radical symbol). Since summations, products, or integrals are often accompanied by limits, these are sought out above and below such symbols.

The technique was evaluated on 100 pages with roughly 93% of the equations reported as being properly segmented. While the technique was reported to be fairly reliable for extracting displayed expression regions, it faced problems with embedded expressions. Greek or italic symbols which should have been labeled as expressions were often ignored as illustrated by Figure 33.



denoted θ, takes value in $\Omega = \{\omega_1, ..., \omega_c\}$ with probabilities $\{p(\omega_1), ..., p(\omega_c)\}$, respectively and that $\mathbf{x}$ is a realization of a random vector $\mathbf{X}$ characterized by a conditional distribution $p(\mathbf{x} \mid \theta)$, $\theta \in \Omega$. Thus, the task is to find a measurable mapping $\psi : R^d \to \Omega$ such that the expected loss function $R(\psi) = E\{L(\psi(\mathbf{X}), \theta)\}$, called risk, is minimal. Here $L(\omega_i, \omega_j)$ is the loss incurred by taking action ω_i when the class is ω_j. In this paper we assume, without loss of generality, that $L(\omega_i, \omega_j) = 0$ for $\omega_i = \omega_j$ and $L(\omega_i, \omega_j) = 1$ for $\omega_i \neq \omega_j$ and then $R(\psi) = \mathbf{P}(\psi(\mathbf{X}) \neq \theta)$ is called the probability of error. It is well known that an optimal rule ψ^* (the Bayes rule) which minimizes $R(\psi)$ is of the following form $\psi^*(\mathbf{x}) = \arg\max_{1 \le i \le c} p_i(\mathbf{x})$, where $p_i(\mathbf{x}) = \mathbf{P}(\theta = \omega_i \mid \mathbf{X} = \mathbf{x})$, $i = 1, ..., c$ are the posteriori probabilities. Let R^* denote the Bayes risk, i.e., the risk of the Bayes rule. In practice we rarely have any information about the distribution of the pair $(\theta, \mathbf{X})$, instead there is in our disposal a training set $\eta_n = \{(\theta_1, \mathbf{X}_1), ..., (\theta_n, \mathbf{X}_n)\}$, i.e., a sequence of pairs $(\theta_i, \mathbf{X}_i)$ distributed like $(\theta, \mathbf{X})$, where $\mathbf{X}_i$ is the feature vector and θ_i is its class assignment. An empirical classification rule ψ_n is a measurable function of $\mathbf{X}$ and η_n. It is natural to construct a rule which resembles the Bayes rule, i.e., by replacing $p_i(\mathbf{x})$ by its estimate $p_{in}(\mathbf{x})$. A popular nonparametric classification technique is the kernel classifier being defined as follows

$$\psi_n(\mathbf{x}) = \arg\max_{1 \le i \le c} \sum_{j=1}^{n} \mathbf{1}(\theta_j = \omega_i) W\left(\frac{\mathbf{x} - \mathbf{X}_j}{b}\right), \qquad (1.1)$$

Figure *33:* A result of Kacem's expression segmentation technique [104]. Note that the theta symbol is only segmented for the cases when context propagation dictates that it should be. When it is by itself it is missed entirely, while when it is wrapped in a parenthesis or has a subscript it is segmented.

Jin et al. [105] (2003). Isolated expressions are extracted based on a Parzen window classifier and embedded expressions are extracted based on 2-D structure analysis and various heuristics. The technique is evaluated on a dataset consisting of 93 pages from technical journals. 10% of the pages in this set are used for training and the other 90% for evaluation. The results are reported as favorable, however no thorough evaluation or specific results are provided.

Drake and Baird [106] (2005). Drake and Baird utilize a technique based upon Kise's bottom-up Area Veronoi Diagram-based physical segmentation method [75]. For each line of text, the Area Veronoi Diagram is calculated and then each vertex and

edge is classified as either normal text or mathematical. All of the results for the line are then combined in order to classify the line as normal text or as a displayed expression. A strength of this technique is that the Veronoi diagram is invariant to skew of the page. Thus an expression could be detected by this technique regardless of its angle in reference to the rest of the page. The input to both training and evaluation are images of isolated text lines which were cut out of page images manually or synthesized in isolation using Latex. The dataset contains roughly 4,400 connected components labeled as math, with about half used for training and the other half used for testing. The Veronoi diagram's edges were also labeled in the dataset, with roughly 4,000 used for both training and testing purposes respectively. From reviewing the confusion matrices provided it was found that the true positive rate for math connected component detection was 88% and false positive rate was ~7%. The algorithm was not tested on any lines that contained a mixture of math and normal text.

Tian et al. [107] (2005). Tian et al. propose a technique aimed at segmenting both displayed and embedded expressions. Displayed expressions are found by calculating the average y distance of the center of all connected components on a line from the line's center. If this measure is above an empirically determined threshold, then the line is declared as a displayed expression candidate. To confirm whether or not the candidate is truly a displayed expression region, the line's connected components are run through a recognizer specifically designed from mathematical symbols. If any mathematical symbols are found then the text line is confirmed to be a displayed expression region. Embedded formulas are found by analyzing the spatial orientations of connected components on the text line, recognizing mathematical symbols, and employing propagation rules based upon these symbols. The technique is evaluated on more than 100 pages of technical documents to achieve a true positive rate of 95.19% for displayed expressions and 90.12% for embedded expressions. False positive rates are not reported.

Yamazaki et al. [108] (2011). Yamazaki et al. describe a technique which they have integrated into OCRopus [85]. The technique only detects displayed expressions and uses features very similar to those proposed by Garain. Also included are the following features: standard deviation of symbol aspect ratio within a text line, and left indentation measurement. Rather than using the averaging techniques employed by Garain, a SVM is used. The system is tested on an unspecified number of pages

containing 542 displayed expressions, of which, 531 are identified correctly. No further evaluation is provided.

Lin and Baker et al. [109] (2012). X. Lin from Peking University and J. Baker from University of Birmingham collaborated in developing novel techniques for expression segmentation in digitally-born PDF documents [118][119]. In 2012, Lin proposed a technique for segmenting embedded expressions in digitally-born PDF documents (the displayed expressions segmentation is not evaluated in this work). Since the documents used in this study are digitally-born it is assumed that all typesetting information and text is available within the PDF's on which the experiments are run. In order to detect embedded expressions, text lines are evaluated each in turn (no OCR is required). First the words on the text line are segmented by using an adaptive thresholding technique on the PDF's image. A histogram is created which gives the frequency of horizontal gap lengths throughout the line. The second most frequent gap length is used for determining the word gap threshold (the first most frequently occurring gap is typically the distance between individual characters within words). Characters such as parenthesis, equals signs, sums, etc., are segmented as words regardless of their left and right horizontal gaps, assuming that their unicode is available within in the file.

Once the words are segmented, 12 features are calculated for each individual word. These include 7 geometric layout features, 3 character features, and 2 context features. The geometric layout features include the variance of font size of the symbols within a word based on the PDF's typesetting information, variance of the y-coordinates of the symbols, variance of inter-character gap, variance of the bounding box width and height, a measure of the degree to which all the symbols in the word correspond to the same language (i.e., English or Non-English), and percentage of English characters found within the word. The character features include the amount of mathematical characters in the word, recognition result of the leftmost character, and recognition result of the rightmost character. The recognition result of the left most and right most characters give an indication as to whether or not the words in between or to the left or right are mathematical. For instance, if the right-most character of the word is an “=” then it can be inferred that whatever is directly to the left must also be mathematical. Context features include result of the right most symbol of the previous word (word to the left), and type of the left-most symbol of the next word (word to the right). Continuing with the “=” example, if the right-most symbol of the word to the left is an “=” sign, then it can be inferred that the current

word is in some form mathematical in that it is part of the equation. All of the aforementioned features are normalized to some value between [-1,1]. The features are then fed into a binary SVM classifier which has been trained on labeled datasets.

Experiments are carried out on 50 journal papers and 5 mathematical text books. 2 pages from each paper and 20 pages from each textbook are randomly selected, thus experiments are carried out on 200 pages in total. The 200 pages are divided into 5 equal subsets and 5-fold cross-validation is carried out. In each round a single subset is used for evaluation and the 4 others are used for training. This is repeated 5 times such that all 5 subsets are evaluated in this manner. The precision (positive predictive value) and recall (true positive rate) measurements are made for each of the 5 evaluations and then averaged to get the final result: 86.94% precision and 84.29% recall[4].

Also in 2012, Lin et al. proposed a new technique for the evaluation of expression segmentation methods [120]. A new evaluation metric is proposed which takes into account oversegmentations, false positives, merges, etc. Weights of various error types can be set based on specific application scenarios by changing parameters of the evaluation tool implemented. For instance, in document information retrieval of a math equation, a false negative should typically be weighted much higher than a false positive. Either area-based evaluation or symbol-based evaluation is offered but no pixel-level accuracy is achieved.

The dataset and groundtruth are claimed to be publicly available but truly are not since the documents used are not in the public domain. The dataset has 194 digitally generated PDF pages. In total, 400 document pages were carefully selected with an aim to be statistically representative of a wide variety of documents. Sources for these documents range from conference proceedings, journals, books, and reports. For each source document at least 1 and at most 8 pages are selected and added to the dataset. Documents are selected with publication years ranging from 1977 to 2010. Domain topics include mathematics, computer science, biology, and physics. 65% of the document pages are single column and the remainder are multicolumn. PDFs are also included that are generated by different PDF-writers (i.e. AFPL Ghostscript, Acrobat Distiller, Acrobat PDFWriter, ESP Ghostscript, GNU Ghostscript, Miktex PdfTex, etc.). The number of displayed and embedded formulas in each page is counted and selected so that there is a wide variety of both counts.

[4] It should be noted that this segmentation technique suffers similar problems to Garain's 2009 approach, in that the initial word segmentation is never fixed, regardless of how incorrect the adaptive thresholding technique may be.

Lin et al. Lin et al. [118] collaborated with Baker et al. with the goal of improving mathematical expression segmentation accuracy for digitally-born PDF documents. In this work, it is argued that improper initial physical segmentation of text lines that contain math causes significant problems in formula identification. The authors describe various cases of commonly mis-segmented mathematical expressions separated into three basic categories. The first category describes a text line containing expressions that are oversegmented into two vertically overlapping lines (occurs with fractions, sums/integrals when upper/lower bounds are present, etc.). The second category describes matrices and other grid-like expressions which results in similar difficulties. The third category describes a single expression which is covered by multiple lines. This occurs when an expression on the left side of an equation is set equal to multiple expressions where each subsequent expression after the first is covered by a new line. It is argued that, for identification and recognition purposes, it is best that each new line of such an expression is merged during physical segmentation as opposed to keeping each expression on the right of the equals as a separate segment.

To address these problems, a learning-based text-line merging technique is utilized. The technique utilizes one classifier to find the first two categories of mis-segmented expressions and a second classifier for the third category. First the text body is segmented from the header/footer regions. The text lines and columns are then found using projection profile cuts. Next, for each line, the decision is made as to whether or not the current line should be merged with the next line, based on the first two improperly segmented categories. For this purpose, several features are utilized. Features include vertical space between the text lines, the relative horizontal width of the textlines, the difference of indentations between the two text lines, ratio of average textline character widths and heights, ratio of main font sizes used in the lines (for digitally-born PDF's the font sizes are typically available), 2 features describing existence of fraction signs, the existence of a large operator in either line, features describing whether or not the text line ends with a binary operator, and if the lower line ends with a formula index. Then some of the aforementioned features are employed just to describe the individual textlines themselves rather than the relation between two consecutive ones.

The classification task is separated into two stages: the first aims at properly segmenting all individual expressions and the second aims at merging single expressions that span multiple lines. When training for both of these stages,

performance is compared on 7 machine learning algorithms: SVM, MLP, Decision Tree, Random Forest, Bayesian Network, Bootstrap Aggregating (Bagging), and Adaboost. Bagging and Adaboost were reported to obtain the best performance during training for the first and second stage of segmentation, respectively, and were thus adopted for evaluation. The technique is evaluated on 600 document pages. 100 of these pages are used for training while the other 500 are used for evaluation. Precision and recall are reported on both stages for 100 of the the images tested, and then for the remaining 400 accuracy is only reported. The precision, recall, and accuracy were, for the most part, all reported to be above 90%. False positive rates are not reported.

Baker et al. Baker et al. [119] introduced a tool, Maxtract [121] which uses projection profile cutting to segment the mathematical expressions but is reported to not be very accurate. An improved segmentation technique is described and its effects on the accuracy of Maxtract are reported. A histogram-based approach is described for line segmentation. The approach is to first extract all of the connected components on the page, and then determine initial lines based on grouping the connected components based on vertical proximity. A histogram is then constructed for the entire page which captures the horizontal distance between each adjacent component on each text line. Two local minimums are commonly observed in a similar location on the histograms of their pages. These local minimums are used to represent the minimum and maximum distance expected of a “principal” text line. Principal text lines are those that would correspond to either normal lines or the main lines in big math expressions (for instance, ones that contain summations, integrals, etc). The “non-principal” lines are the ones which may correspond to limits or upper and lower bounds, and thus may need to be merged with their nearest principal line. Such lines tend to be more sparsely distributed horizontally than the principal lines. Any characters having a distance observed outside of the aforementioned range for “principal” lines is considered a candidate for being part of a “non-principal” line.

A second pass is then used to correct any lines which may have been mistakenly labeled as non-principal, the heights of the non-principal lines are compared with the height of their next line. If the maximum connected component height of the non-principal line is greater than that of the principal line divided by some threshold then the non-principal line is re-labeled as principal. The threshold value is determined empirically on a small sample set. A third pass then checks that the lines that were considered as principal truly are principal. This is done by making sure that all of the principal lines have a height greater than the maximum height of

the non-principal lines. The resulting non-principal lines are then merged with their adjacent horizontally overlapping principal lines. If a non-principal line has no adjacent horizontally overlapping principal line then it will be converted to a principal line.

The technique is evaluated on 200 pages comprising a mixture of technical journals and text books. 96.9% accuracy is reported. The technique was then further manually evaluated on a larger dataset of 1000 pages from more than 60 mathematical papers and an accuracy of 98.6% was reported. The most common error was reported as that of incorrectly classifying a non-principal line as principal. This occurs when the horizontal distance between characters on a non-principal line is similar to that of the principal line. The authors then carry out a further experiment which integrates the aforementioned segmentation technique into their Maxtract software. Note that the aforementioned technique does not necessarily go so far as to logically segment the entire mathematical expression regions, it only serves to ease the physical segmentation task prior to logical labeling. Unfortunately this also results in some errors which wouldn't occur with normal segmentation techniques. For instance, footer regions were sometimes mistaken for non-principal lines and incorrectly merged with their preceding line. A technique similar to that reported in the earlier literature [109] is then employed after the aforementioned segmentation step to identify mathematical zones.

The modified Maxtract software is evaluated on two datasets. The first dataset has 184 document pages and the second has only 10 pages. On the first dataset, the math expression identification technique was reported to have, for displayed expressions, a true positive rate of 73.18%, 7.85% false positive rate, and 1.26% false negative rate. 6.56% of the regions were reported as being oversegmented while 12.41% of the region were reported as undersegmented. No results were reported for embedded expressions in the first dataset however. In the second smaller dataset which contains only ten images, results were reported for both displayed and embedded expressions. While the isolated expressions have a true positive rate of 78.85% and false positive rate of 1.92%, the embedded expression only have a true positive rate of 35.6% and false positive rate of 26.08%, and thus appear to require significant improvement.

Detection of Other Zone Types

While type-specific detection of mathematical expressions is the primary focus in this work, there are many other aspects to type-specific layout analysis that also need to be worked on. These may include, for instance, the segmentation of tables [122],

musical scores [123], chemical equations [124], circuit diagrams [125], etc. Although these will not be studied for this work, they remain as important challenges in the field.

3 Method

3.1 Introduction

3.1.1 Purpose

The goal of this project is to enhance the quality of document layout analysis and OCR for printed (non-handwritten) technical/scientific public domain documents which may contain displayed/inline equations, matrices, illustrations, graphs, etc. Automated processing of printed documents requires both physical and logical layout analysis techniques to be employed in order to segment and classify zones of interest for correct processing. After physical layout analysis is carried out, regions corresponding to illustrations, plain text, musical notation, and mathematical formulas all must be classified so that they can be processed correctly. The application of this project is in automated processing of digitized public domain documents (or non public domain documents with author/publisher consent and due legal permission). Automated document processing has seen widespread use in industrial settings (i.e., automated processing of bank notes or postage envelopes) as well as for Assistive Technology in aiding the blind and/or visually impaired in their accessibility to information. Document layout analysis and OCR has also found use in the “Google Books Initiative” whose founders have envisioned a veritable online “Library of Alexandria” from which all of the world's knowledge could be acquired.

This project aims to achieve its goal by developing and evaluating a mathematical expression detection and segmentation (MEDS) module fully integrated with Google's existing document layout analysis software [18], compare this module's accuracy to that of a default implementation provided with the software, and to evaluate performance under a wide variety of inputs. Since a significant problem observed in the existing literature is a lack of objective performance comparison among MEDS modules, this work is tested on a dataset of public domain documents that will be made available to others. The groundtruth dataset, MEDS implementation, and evaluation tools are made publicly available [126] in the hopes that the performance of the current technique may more easily and objectively be compared to previous and/or future techniques.

While the general layout analysis framework of Tesseract is utilized for this work, a new MEDS module implementation overrides Tesseract’s default one through run-time polymorphism. The performance of Tesseract’s document layout analysis

framework will be evaluated both under the default MEDS module and the newly implemented one, the results compared, and experiments carried out on printed text from a variety of sources in order to gain insight into how best to increase accuracy over a wide variety of documents while avoiding the problems of overtraining to the largest extent possible.

3.1.2 Problem Statement and Project Scope

Currently, the commercial OCR system best suitable in handling scientific documents is Masakazu Suzuki's “Infty Reader” which can accurately recognize a wide variety of complex mathematical equations as well as matrices, assuming that they are first properly isolated from other, non-math, regions of text. While some of the system's reported shortcomings are attributed to the merging or breaking of characters during image scanning, many of the system's errors observed in practice were caused to the improper isolation of the math regions from non-math regions. The Infty Reader system isolates regions of interest based solely upon analysis of the output of a commercial OCR system, Abby Fine Reader. Regions which appear to be “junk output” are deemed as candidates for math recognition. Infty Reader effectively sidesteps the problem of physical and logical layout analysis, relying solely upon whatever physical and logical layout analysis is performed by the proprietary Abby Fine Reader system utilized. The system's degree of document understanding prior to recognition, therefore, is entirely at the mercy of false positive recognition and/or layout analysis errors made by Abby Fine Reader, which, in and of itself, was not even designed with the layout analysis of scientific/mathematical documents in mind.

Meanwhile, the “Google Books Initiative” project has spurred a great amount of interest in the automated processing of a wide variety of documents ranging from ancient texts, magazines, articles, to scientific/mathematical textbooks, dissertations, etc. in over fifty languages. An experimental equation detector was implemented as part of Google’s 2011 release of their open source OCR engine, Tesseract. Upon performing a preliminary evaluation of the equation detector on several pages of a public domain calculus text book [127] it was found that fewer than a fourth of the equation zones were fully segmented. Of all of these fully segmented equation zones, none of them were without at least some false positive pixels and/or under/over-segmentations. *The overall problem statement of this project can thus be described as designing and evaluating a new MEDS module which can detect and segment math expressions correctly on a range of document types*. In the literature, the problem of layout analysis for documents with formulas has either been that of properly detecting

and then segmenting the regions for math recognition or to detect all of the regions so they can be discarded and thus not hinder normal OCR output. It is important to stress that the former of these two problems is the one being addressed in this work.

3.1.3 Definitions and Acronyms

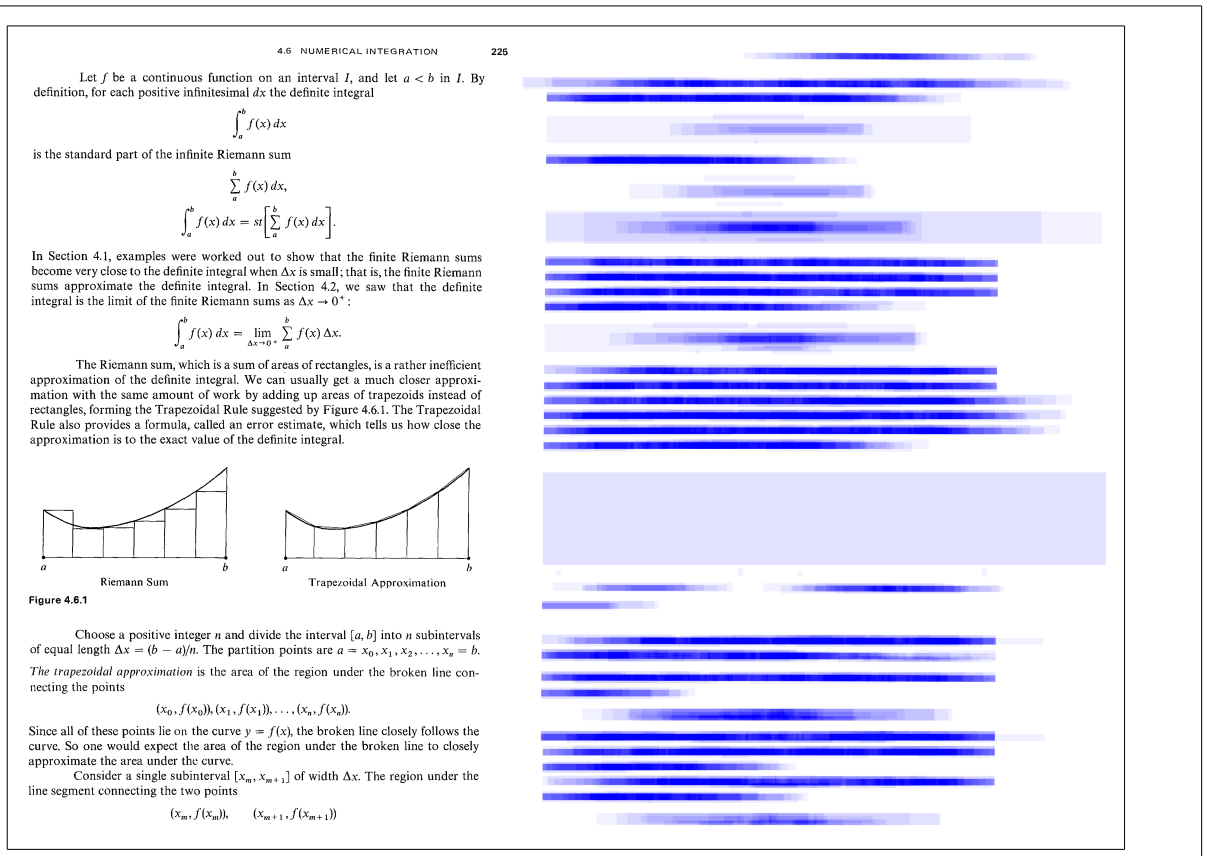
Various acronyms and terms which will be used throughout the remainder of this work are italicized and briefly explained in this section. The problem being addressed in this work is that of both detecting and properly segmenting regions of mathematical text in a document image from those of non-mathematical text, so that overall document recognition accuracy may be improved. The problem is herein given the title of mathematical expression detection and segmentation (MEDS). The *MEDS* module implemented in this work operates as a component of Tesseract's larger document layout analysis system which is still under development by Google [18]. The overall program operates in two major phases: detection and then segmentation. The initial connected components found during detection, referred to as the *seed regions*, are then *merged* into surrounding regions based upon various heuristics during the segmentation phase.

3.1.4 Tesseract Document Layout Analysis Framework Overview

Since the MEDS module implemented in this work is fully integrated with and utilized as a component of Google's open source document layout analysis and OCR software, Tesseract, this section gives a brief overview of some of the layout analysis software's inner workings and introduces some associated terminology. While the bulk of Tesseract's layout analysis software is geared toward physical layout analysis (i.e., segmenting columns of text, filtering out noise, and segmenting image regions) some logical layout analysis for detecting math equations and table regions is also observed. The MEDS module implemented in this work overrides Tesseract's default equation detection implementation through run-time polymorphism. Run-time polymorphism is used to facilitate performance comparisons of the new and default modules.

Prior to initiating the MEDS module, Tesseract's layout analysis system segments image regions from normal text on the page and also filters out noise. The page is then divided into regions referred to as Column Partitions. Each Column Partition (CP) represents a region of text which should be physically and logically segmented from its neighboring regions (i.e., an individual row of text within a column,

a displayed expression, etc.). These regions are initially segmented through a projection blurring technique as illustrated by Figure 34 which blurs all connected components in the direction of their nearest neighbor. Each blurred region in Figure 34 is thus effectively treated as an initial CP by Tesseract's physical layout analysis system pending further processing. A Column Partition Set (CPset) is a division of a horizontal slice of the page into column partitions at a given vertical location. Upon completion of Tesseract's document layout analysis the page is represented as a list of CPSets, where each entry of the list represents all of the text at a horizontal slice of the page. While a page with only one column will consist of a list of one element CPsets, a more complex page layout may consist of a title (one element CPSet) followed by three columns of text (several three element CPSets). In Figure 34, for instance, while much of the page consists of one element CPSets, the CPSet corresponding to the image captions on the lower half of the page and the heading at the top of the page would both ideally consist of two elements. More information on the algorithms used by Tesseract to determine which CPSets are the best fit for a page layout can be found in [18].



4.6 NUMERICAL INTEGRATION 225

Let f be a continuous function on an interval I, and let $a < b$ in I. By definition, for each positive infinitesimal dx the definite integral

$$\int_a^b f(x)\,dx$$

is the standard part of the infinite Riemann sum

$$\sum_a^b f(x)\,dx,$$

$$\int_a^b f(x)\,dx = st\left[\sum_a^b f(x)\,dx\right].$$

In Section 4.1, examples were worked out to show that the finite Riemann sums become very close to the definite integral when Δx is small; that is, the finite Riemann sums approximate the definite integral. In Section 4.2, we saw that the definite integral is the limit of the finite Riemann sums as $\Delta x \to 0^+$:

$$\int_a^b f(x)\,dx = \lim_{\Delta x \to 0^+} \sum_a^b f(x)\,\Delta x.$$

The Riemann sum, which is a sum of areas of rectangles, is a rather inefficient approximation of the definite integral. We can usually get a much closer approximation with the same amount of work by adding up areas of trapezoids instead of rectangles, forming the Trapezoidal Rule suggested by Figure 4.6.1. The Trapezoidal Rule also provides a formula, called an error estimate, which tells us how close the approximation is to the exact value of the definite integral.

Figure 4.6.1

Choose a positive integer n and divide the interval $[a, b]$ into n subintervals of equal length $\Delta x = (b - a)/n$. The partition points are $a = x_0, x_1, x_2, \ldots, x_n = b$.

The trapezoidal approximation is the area of the region under the broken line connecting the points

$$(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n)).$$

Since all of these points lie on the curve $y = f(x)$, the broken line closely follows the curve. So one would expect the area of the region under the broken line to closely approximate the area under the curve.

Consider a single subinterval $[x_m, x_{m+1}]$ of width Δx. The region under the line segment connecting the two points

$$(x_m, f(x_m)), \qquad (x_{m+1}, f(x_{m+1}))$$

Figure 34: On the left is a document image and on the right is debug output from Tesseract showing how that image is blurred during physical segmentation.

3.1.5 Overview

The method section of this thesis is organized as follows. The System Overview section gives a general top-down description of the system and then the System Architecture section discusses how the main modules function together in a meaningful way. The Component Design section then gives in depth details on how all of the modules are designed and what data structures and algorithms are used. The Component Design section is then followed by a brief conclusion which gives some ideas for future work.

3.2 System Overview

The MEDS software component described here is designed such that it may be used in coordination with other components to produce a full-fledged document layout analysis system. While the document layout analysis system, as a whole, is geared toward the proper segmentation and detection of all zones (i.e. normal text, image, halftones, mathematical expressions, musical notation, logos, chemical equations, etc.) in an arbitrary printed document image, the MEDS subsystem is geared toward only the proper detection and segmentation of mathematical expression regions. This subsystem is fully integrated with an existing layout analysis system, Tesseract [18], and its results compared to those of a default MEDS component supplied with the software. The subsystem is then evaluated on a ground truth data set which includes 75 images all taken from public domain texts. The overall system is divided into three primary components: groundtruth generation, evaluation technique, and MEDS implementation. The evaluation method objectively gauges performance by calculating true positive rate, precision, accuracy, false positive rate, false discovery rate, specificity, and negative predictive value all down to pixel-level [87]. The dataset, evaluation tools and groundtruth generation tools are made freely open to the public [126] in hopes that they may be useful for the objective comparison of the subsystem's performance to any future or existing techniques.

3.3 System Architechture

As mentioned previously, the system consists of three primary components: the ground truth generation module, mathematical expression detection and segmentation (MEDS) module, and the evaluation module. Together, these three modules effectively comprise a test-driven development environment wherein MEDS modules may be interchanged and evaluated against one another for objective performance comparison. All MEDS modules which can be evaluated by this system

are fully integrated with Tesseract's document layout analysis software [18], and are instantiated by overriding Tesseract's EquationDetectBase class [128]. Tesseract utilizes a hybrid physical layout analysis approach to locate initial CP candidates on the page which are then corrected through further type-specific document layout analysis techniques (i.e., segmentation algorithms for table, music, math, etc.). While Tesseract provides a default MEDS module, a preliminary evaluation of the module's accuracy demonstrated in Section 3.4.3 shows a need for improvement. Figures 35 and 36 illustrate how the groundtruth generation, MEDS, and evaluation modules collaborate in order to foster a test-driven development environment for the enhancement of MEDS accuracy.

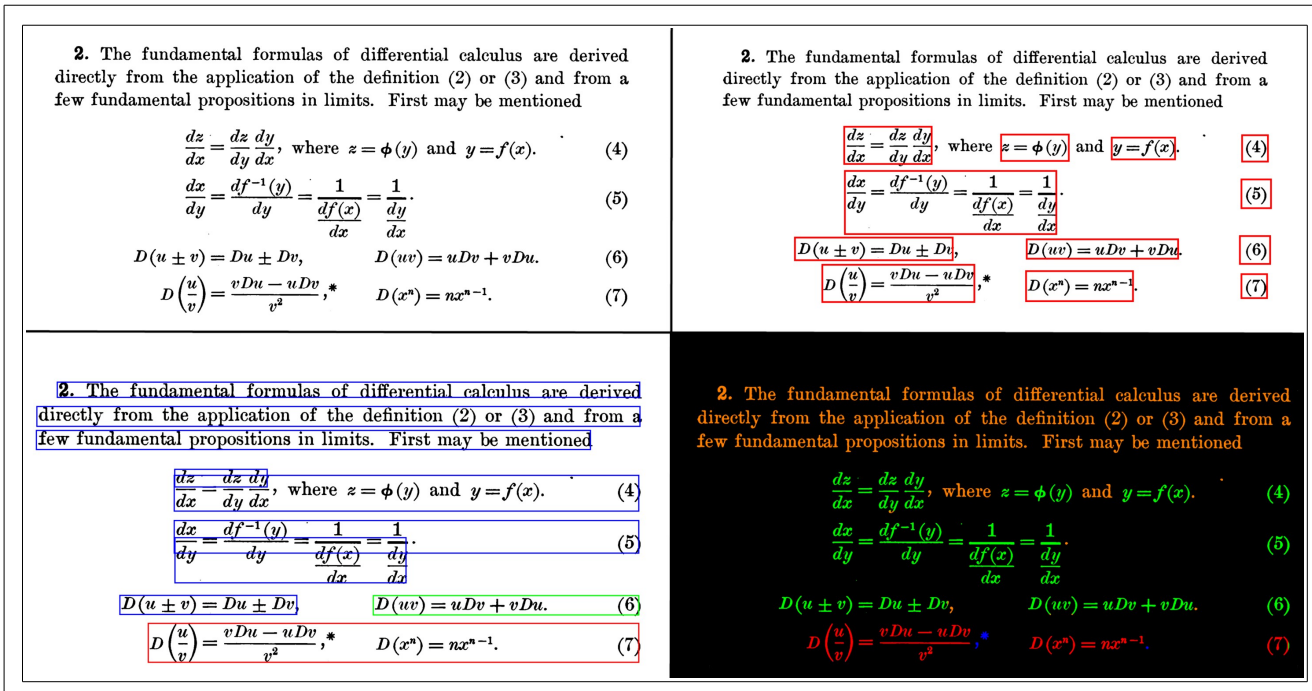


Figure 35 (Top-left) Portion of input image. (Top-right) Manually generated groundtruth. Segments outlined with red rectangles are individual displayed expressions. (Bottom-left) Output of Tesseract's default MEDS module. Segments outlined with red rectangles are individual displayed expressions, segments outlined with green rectangles are individual embedded expressions, and segments outlined with blue rectangles are normal text. (Bottom-right) Pixel-accurate evaluation results for displayed expressions found in Tesseract's default module after comparing it to the groundtruth. False negative pixels are colored green, false positive pixels are colored blue, true positive pixels are colored red, and true negatives are orange. All background pixels are colored black. Notice that the true positive region at the bottom is undersegmented (a single region should correspond to the three regions).The commas and asterisk are false positives since they are not logically part of the mathematical expressions.

The system is first fed multiple document image pages (in formats such as .png) from which the groundtruth is manually generated. The groundtruth will contain the bounding box coordinates for all displayed and embedded math expressions (and may also contain the coordinates of all displayed expression labels) as further explained in Section 3.4. Both the input document image and the groundtruth are fed into the evaluation module which subsequently triggers Tesseract's document layout analysis software as illustrated in Figure 36. The MEDS module to be evaluated is embedded in Tesseract's layout analysis software such that it is called after initial CPSet estimates have been made through Tesseract's hybrid physical segmentation technique [18]. Either Tesseract's default MEDS module can be evaluated or any new MEDS module may override Tesseract's default one so that it can be evaluated. Once the layout analysis software is finished being run on the document image (or multiple images if desired), the results of the MEDS module are evaluated against the groundtruth to obtain various evaluation metrics as specified in Section 3.4.3. The resulting metrics for different MEDS modules evaluated on the same input data may then be objectively compared.

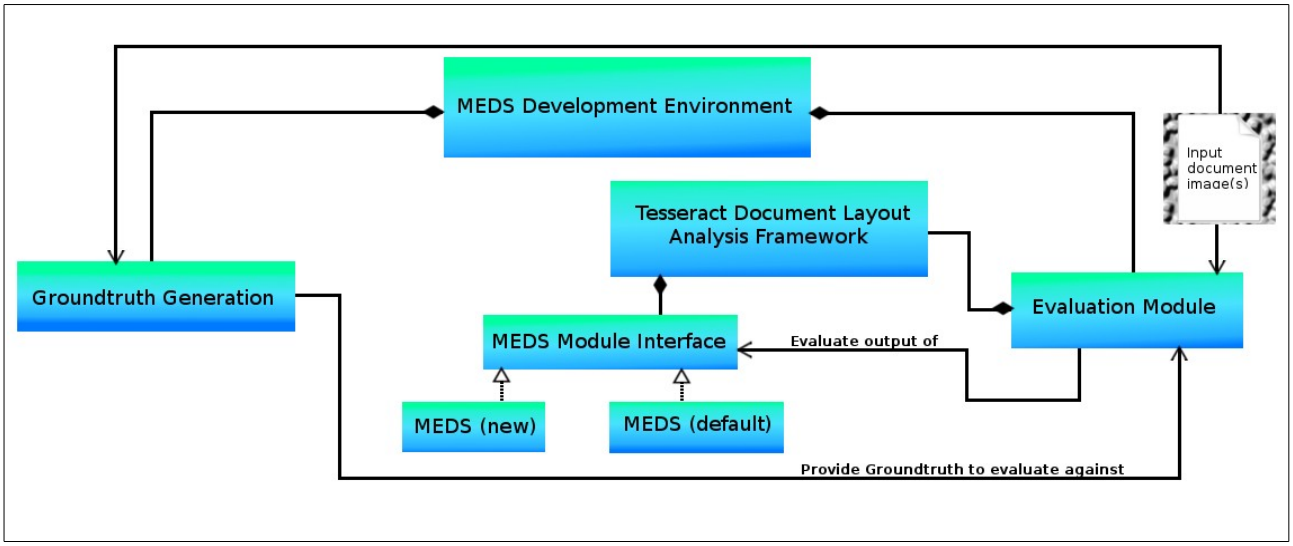


Figure 36 High-level UML-based system architecture overview.

The MEDS module implementation is divided into two primary components: detection and segmentation. The detection subsystem uses a trained binary classifier to predict whether each connected component of the image is math or non-math. The binary classifier takes as input a feature vector found from the feature extractor as illustrated in Figure 37. Once trained, the classifier can carry out a prediction on new

data assuming the input data is a feature vector generated from the same feature extractor it was trained with. The segmentation module uses various heuristics to then merge detected math regions with neighboring ones (i.e., a + operator should have both left and right operands, a fraction bar should have upper and lower operands, etc.). Compile-time polymorphism is utilized here for both the Detector and Segmentor so that different MEDS modules can be effectively interchanged for comparison and testing purposes without any significant performance overhead. The detection module consists of training, feature extraction, and binary classifier implementations. The Segmentor currently uses no supervised training and operates purely on heuristic analysis, however the use of supervised training for this stage is considered a goal for future work.

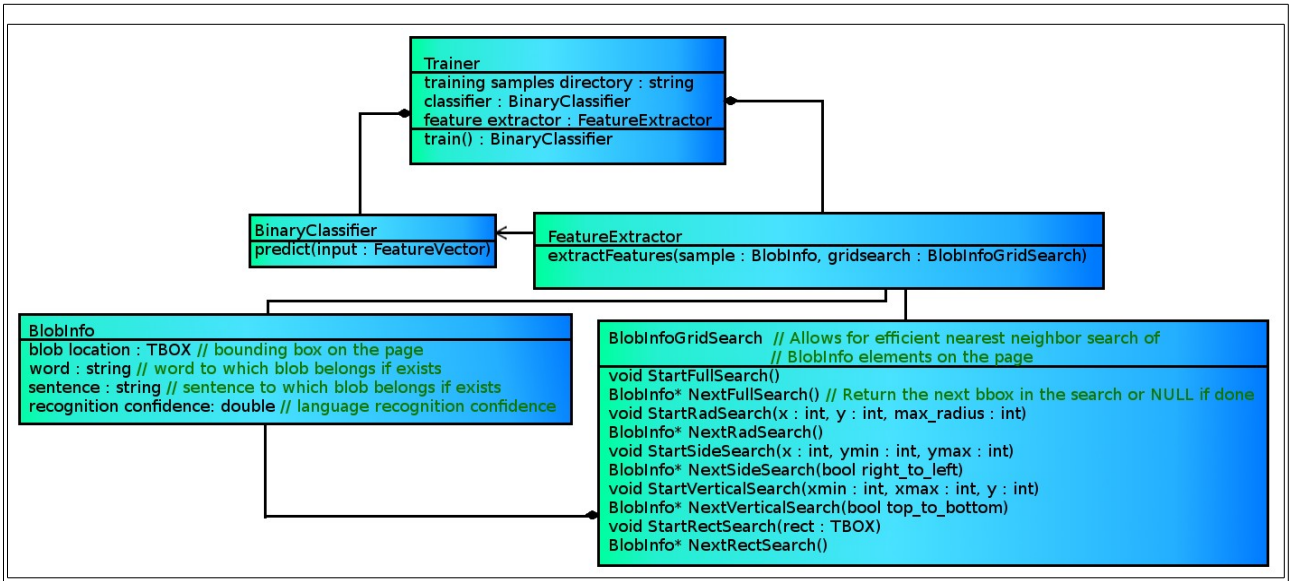


Figure 37 UML diagram to depict the trainer, classifier, and feature extractor interfaces used in the detection module (and also the data BlobInfoGrid data from which features are extracted to be explained in Section 3.4). Compile-time polymorphism is utilized in order to facilitate experimentation with and comparison of various combinations of training, classification and feature extraction.

3.4 Component Design

The design of the three primary components: groundtruth dataset generation, MEDS, and the performance evaluation are each discussed in this section.

3.4.1 Groundtruth Dataset Generation

In designing the MEDS module it is important to first have a proper understanding the problem domain. Mathematical recognition modules require that

their input be properly segmented a priori in order to obtain good accuracy. The definition of “properly segmented” often depends on the type of mathematical expression being analyzed as well as its context. In this work, a groundtruth dataset is manually generated to define the correct segmentations of mathematical regions in a set of 75 images extracted at random pages from the five text books shown in Table 1 (all of which are in the public domain).

Table 1 The textbooks utilized in manually generating the groundtruth dataset for this study

Textbook	Total Pages Used
E. Bidwell, *Advanced Calculus.* (1911) [127]	30
A. S. Kompaneyets, *Theoretical Physics.* (1961) [129]	15
A. C. Lunn, *The Differential Equations of Dynamics.* (1909) [130]	15
D. Sloughter, *Difference Equations to Differential Equations: An Introduction to Calculus.* (2000) [131]	15

In generating the groundtruth dataset, three types of math expressions are defined: displayed expressions, embedded/inline expressions, and (optionally) displayed expression labels. A displayed expression is any expression which resides on its own line of text separated from normal non-math text whereas an embedded/inline expression is one which resides as a part of a normal text line. The displayed expression labels are numbers, letters, or symbols that are used to label and refer to a displayed expression. A displayed expression may, for instance, reside on its own line and then to either the right or left may have a separate label such as (1), (a), etc. The label may then refer back to that expression within the text. Although the labels were manually segmented during the groundtruth generation, the current work will only strive to segment displayed and embedded regions, with the segmentation of displayed labels being left as a goal for future work.

In order to manually segment the expressions, a Qt Graphical User Interface (GUI) implemented in a previous undergraduate independent study [132] was tweaked so that boxed regions of an image can be manually specified, assigned a type (displayed, embedded, or label) and then printed to a file. The GUI used to manually generate the groundtruth dataset is shown in Figure 38.

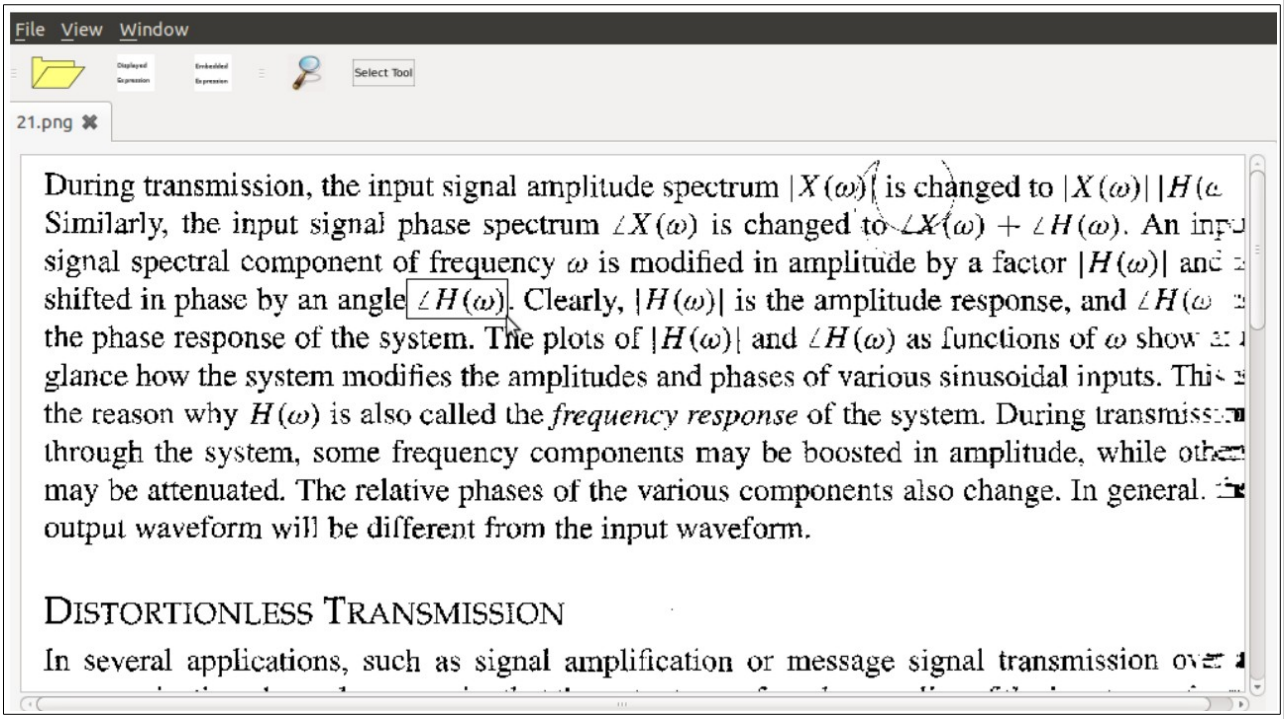
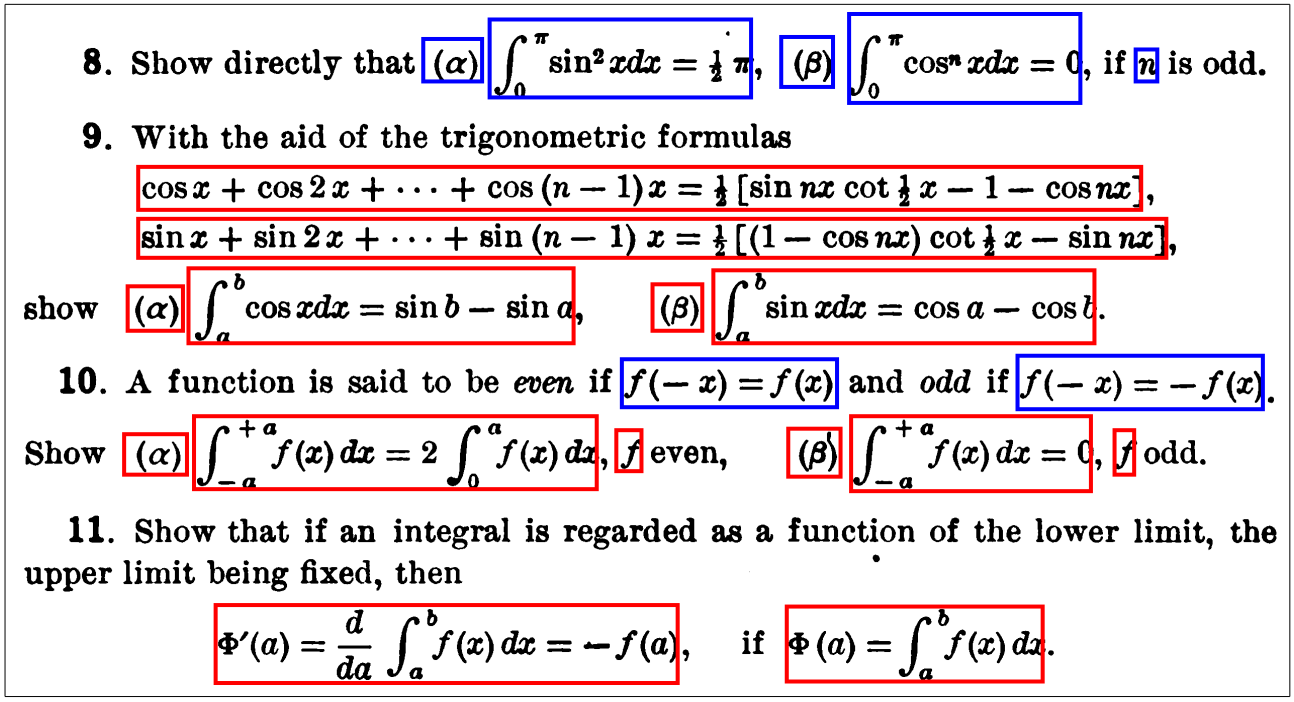


Figure 38 A screenshot of the GUI used to manually generate the groundtruth dataset used for this study.

In manually generating the groundtruth dataset, certain conventions were adopted in order to ensure that the dataset is consistent. In some instances it becomes unclear as to whether a mathematical expressions should be considered displayed or whether it should be considered embedded. In Figure 39, for instance, each of the mathematical expressions could possibly pass as being considered displayed since they comprise most of the text lines, with the lines being spatially separated more than in most normal text. The convention adopted in this work is that, if math expressions appear as part of a line with normal sentence structure and are not intentionally moved down to a separate line, then these expressions are considered embedded. If the expression is moved down a line from a normal sentence then it is called displayed, even if it still obeys normal sentence structure conventions.



8. Show directly that $(\alpha)\ \int_0^{\pi} \sin^2 x\,dx = \tfrac{1}{2}\pi$, $(\beta)\ \int_0^{\pi} \cos^n x\,dx = 0$, if n is odd.

9. With the aid of the trigonometric formulas

$$\cos x + \cos 2x + \cdots + \cos(n-1)x = \tfrac{1}{2}[\sin nx \cot \tfrac{1}{2}x - 1 - \cos nx],$$

$$\sin x + \sin 2x + \cdots + \sin(n-1)x = \tfrac{1}{2}[(1 - \cos nx)\cot \tfrac{1}{2}x - \sin nx],$$

show $(\alpha)\ \int_a^b \cos x\,dx = \sin b - \sin a$, $(\beta)\ \int_a^b \sin x\,dx = \cos a - \cos b$.

10. A function is said to be *even* if $f(-x) = f(x)$ and *odd* if $f(-x) = -f(x)$.

Show $(\alpha)\ \int_{-a}^{+a} f(x)\,dx = 2\int_0^a f(x)\,dx$, f even, $(\beta)\ \int_{-a}^{+a} f(x)\,dx = 0$, f odd.

11. Show that if an integral is regarded as a function of the lower limit, the upper limit being fixed, then

$$\Phi'(a) = \frac{d}{da}\int_a^b f(x)\,dx = -f(a), \quad \text{if } \Phi(a) = \int_a^b f(x)\,dx.$$

Figure 39 Groundtruth dataset segmentation of part of a page taken from [127]. Segments that are colored red are considered displayed while those which are blue are considered embedded. The choice of which regions are displayed vs. embedded is made based on the convention that all expressions that are part of a normal sentence structure and not placed on their own line are embedded, whereas all other expressions are displayed.

Segmenting Numbers. Another rule adopted in this work is that numbers should be labeled as math or non-math based upon their context. While quantities can be interpreted as mathematical since they inherently involve multiplication, such entities as section numbers, parts of section headings, and dates/years. should not be regarded as mathematical. False detection of such cases harms overall document analysis performance in that it may result in the improper interpretation of the document's contents.

Extending Displayed Expressions to New Lines. Since the current manual groundtruth dataset generation procedure can only segment rectangular regions, expressions cannot extend to new lines unless the resulting segment would be a rectangle. Extending the groundtruth generation procedure to allow for isothetic blocks of text is left as an idea for future work. This problem is illustrated by Figures 40 and 41.

$$\begin{aligned}\Delta f &= f(a+h, b+k) - f(a, b)\\ &= hf'_x(a+\theta_1 h, b) + kf'_y(a+h, b+\theta_2 k)\\ &= hf''_x(a+\theta h, b+\theta k) + kf''_y(a+\theta h, b+\theta k)\\ &= hf''_x(a, b) + kf''_y(a, b) + \zeta_1 h + \zeta_2 k,\end{aligned}$$

where the θ's are proper fractions, the ζ's infinitesimals.

To prove the first form, add and subtract $f(a+h, b)$; then

$$\begin{aligned}\Delta f &= [f(a+h, b) - f(a, b)] + [f(a+h, b+k) - f(a+h, b)]\\ &= hf'_x(a+\theta_1 h, b) + kf'_y(a+h, b+\theta_2 k)\end{aligned}$$

Figure 40 Result of existing groundtruth dataset segmentation technique. Notice that the uppermost region is over-segmented (i.e. the set of expressions should correspond to one entity but here they correspond to two so that the comma is not incorrectly considered part of the expression and segmented regions cannot be isothetic). Red segments are displayed while blue ones are embedded.

$$\begin{aligned}\Delta f &= f(a+h, b+k) - f(a, b)\\ &= hf'_x(a+\theta_1 h, b) + kf'_y(a+h, b+\theta_2 k)\\ &= hf''_x(a+\theta h, b+\theta k) + kf''_y(a+\theta h, b+\theta k)\\ &= hf''_x(a, b) + kf''_y(a, b) + \zeta_1 h + \zeta_2 k,\end{aligned}$$

where the θ's are proper fractions, the ζ's infinitesimals.

To prove the first form, add and subtract $f(a+h, b)$; then

$$\begin{aligned}\Delta f &= [f(a+h, b) - f(a, b)] + [f(a+h, b+k) - f(a+h, b)]\\ &= hf'_x(a+\theta_1 h, b) + kf'_y(a+h, b+\theta_2 k)\end{aligned}$$

Figure 41 The correct segmentation which is not currently implemented in the existing groundtruth generation technique. Notice that the top expression region is now properly segmented as one entity even though the comma causes the region to be a more complex isothetic shape than a simple rectangle. Red segments are displayed regions while blue ones are embedded.

Expressions Embedded in Images. Mathematical expressions embedded within images or other non-normal text as illustrated in Figure 42, are considered displayed expressions based upon the convention adopted by this work.

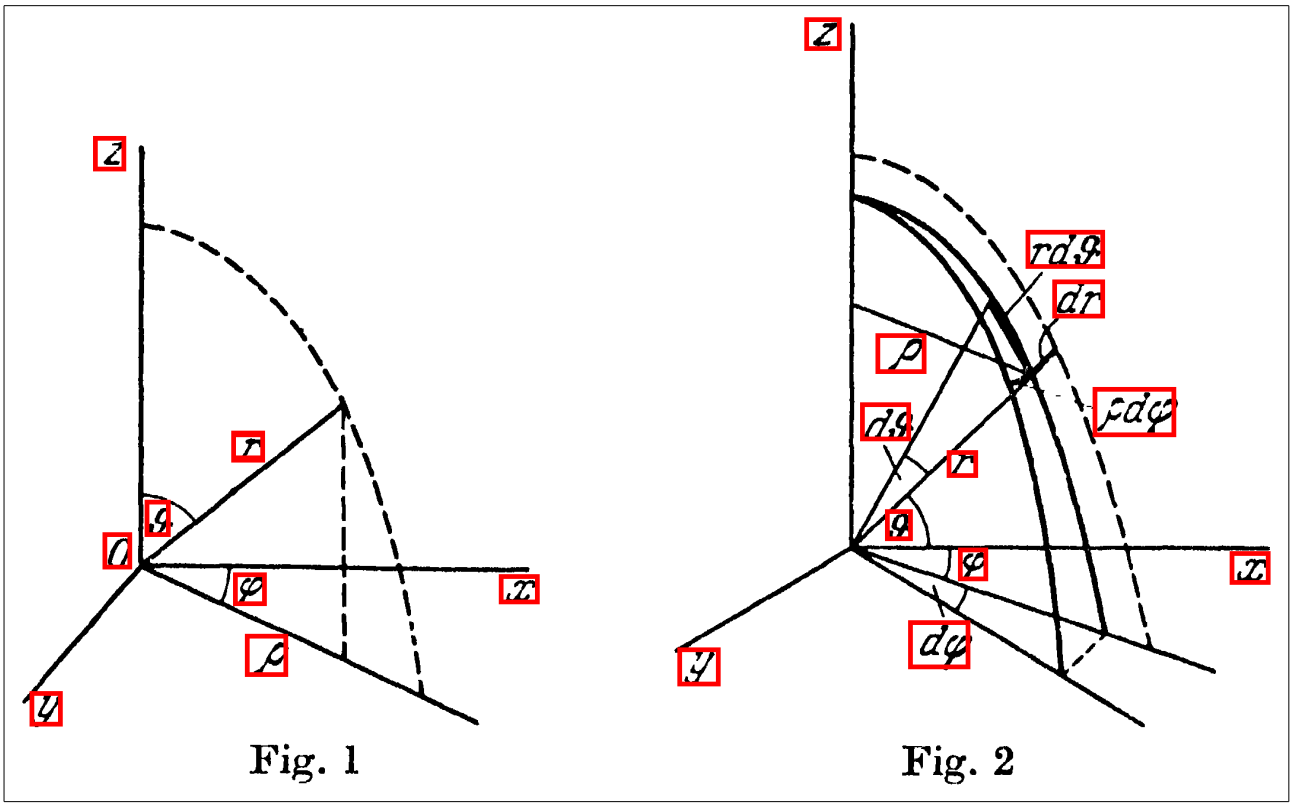


Figure 42 A segment of an image taken from the manually generated groundtruth dataset from the text book [127]. All segments here are segmented as displayed expressions.

Expressions Separated by Commas. Expressions separated by commas are only joined as a single element if there is an ellipsis prior to the last element as illustrated by Figure 43. This decision is made because any group of expressions that includes an ellipsis can mathematically be interpreted as a series, which, itself, is a single expression. When lists of expressions do not include an ellipsis they are not merged and the commas are interpreted as part of the sentence wherein the expressions are embedded as opposed to being part of the expression.

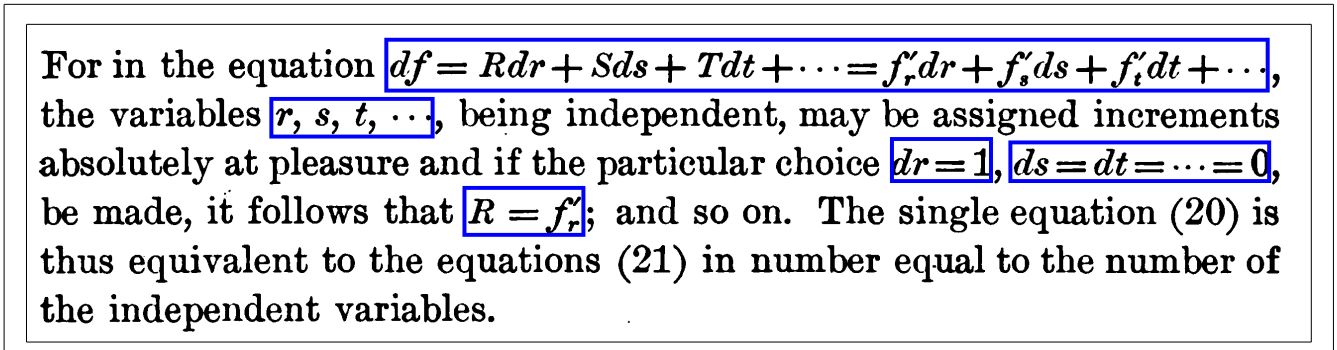

For in the equation $df = Rdr + Sds + Tdt + \cdots = f'_r dr + f'_s ds + f'_t dt + \cdots$, the variables $r, s, t, \cdots$, being independent, may be assigned increments absolutely at pleasure and if the particular choice $dr = 1$, $ds = dt = \cdots = 0$, be made, it follows that $R = f'_r$; and so on. The single equation (20) is thus equivalent to the equations (21) in number equal to the number of the independent variables.

Figure 43 A segment of the groundtruth dataset taken from [127]. All blue segmented regions are considered embedded expressions. Notice that on the second line of text, the comma which follows the ellipsis is functionally part of the sentence in which the expression resides, whereas the $r, s, t, \ldots$ can be interpreted as a single mathematical series and is thus segmented as one region.

3.4.2 MEDS Module

The MEDS module operates by first detecting regions of interest on the page which, based upon a trained classifier, are considered strongly likely to be mathematical in nature. Segmentation is then carried out using rationale based upon the segmentation rules discussed in the groundtruth generation section. An idea for iteratively searching for missed regions based on segmentation results is left as a goal for future work. This would require either a math symbol recognizer or using a Hausdorff distance metric to find other symbols on the page matching to those which were found during segmentation. The process would then iteratively continue until no more new math symbols are found on the page. Due to time constraints this is kept as an idea for future work. Once the detection and segmentation steps are complete, the final step ideally involves searching for displayed expression labels (i.e., the number, letter, or other symbol which is used to refer back to a displayed expression). The label, assuming it is recognizable by the language OCR engine, would then be found in the recognized text and mapped to its location on the image. Again, due to time constraints the searching for displayed expression labels is not implemented in this work, but kept as an idea for future work.

Detection Subsystem

The design starts in a bottom up fashion and makes no assumptions about the correctness or incorrectness of how CP's were formed by the Tesseract framework from which it is instantiated. The module does, however, utilize the results of

language-specific OCR (while the default language is English, other languages may also be employed) in order to quickly rule out most normal regions of text from possibly being mathematically oriented. A goal in the design of this module is that it should still work all the same even if the underlying Tesseract framework from which it is instantiated were to be altered.

The input to the MEDS module is a grid of CP objects (CPGrid) and the list of CPsets determined to best represent the entire page by the hybrid layout analysis technique described in [18]. The grid template container class is used within the Tesseract framework for fast neighborhood access to bounding box classes as illustrated in Figure 44. The grid provided to the MEDS module has, as its contained object, CP objects determined through the previous hybrid layout analysis. Since one of the design goals for this module is to ensure the MEDS results are as independent of previous hybrid layout analysis as possible, the CPGrid is first converted to a BlobGrid. While the CPGrid allows for nearest neighbor access among the CP's, the BlobGrid allows for nearest neighbor access among all of the connected components[5] in the image as illustrated in Figure 45.

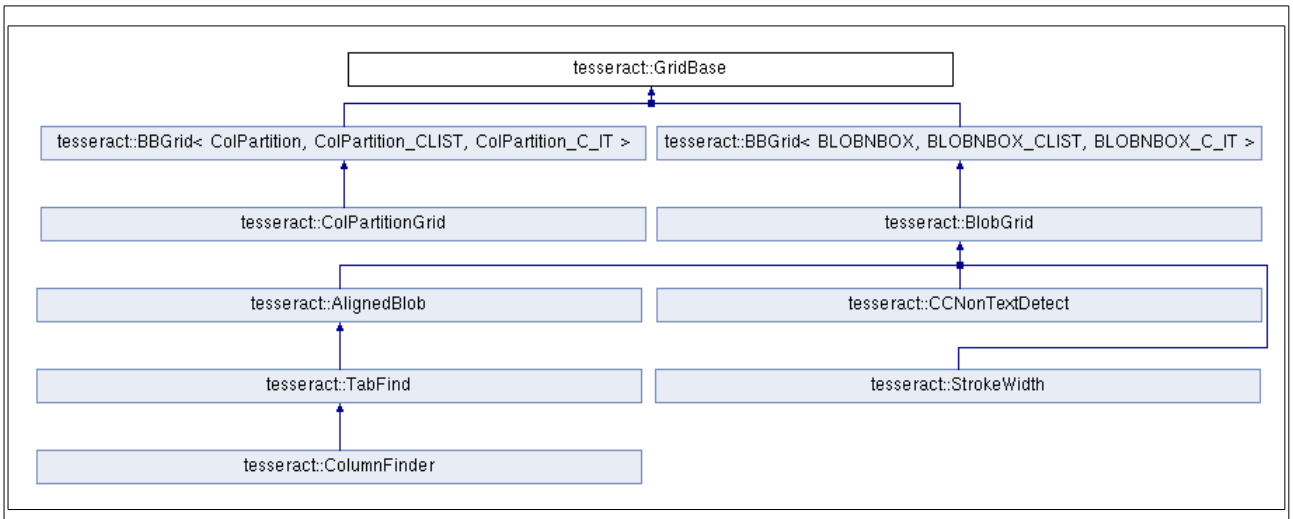
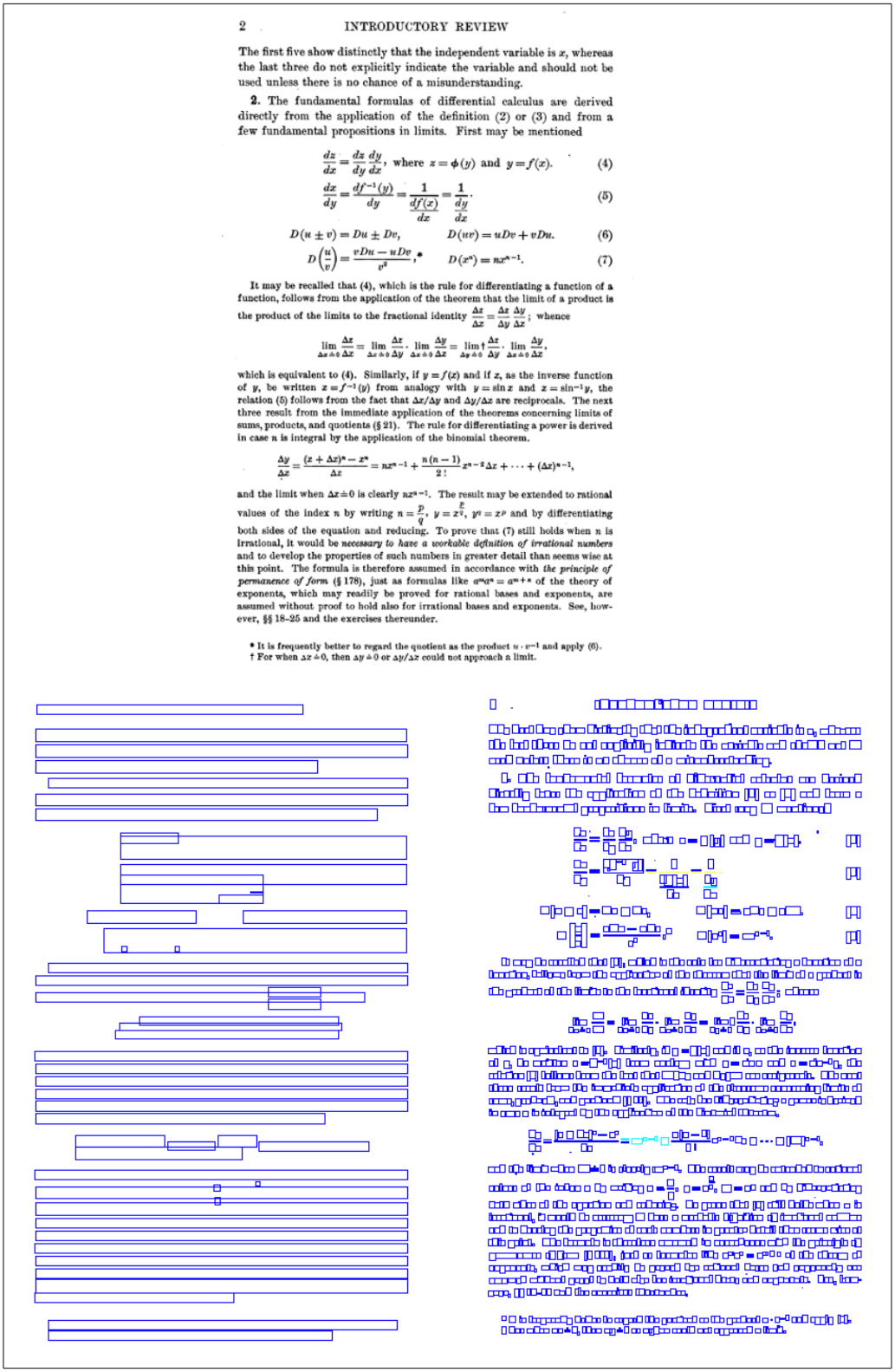


Figure 44 The GridBase datastructure is used extensively within the Tesseract framework to facilitate fast nearest neighbor access among various components on the image. The above image is a Doxygen-generated inheritance diagram showing many of the different classes which are derived from the GridBase class. Being a template container class, it's core functionality of nearest neighbor search can be utilized for any number of data structures ranging from CP's, recognized blobs (BLOBNBOX), unrecognized blobs (C_BLOBS), and has been utilized to build a custom grid data structure in this project.

[5] The terms “connected component”, “blob”, and “character” are used interchangeably in this work to describe either a single group of connected pixels or a character recognized by Tesseract which may consist of one or more groups of connected pixels as is the case for characters like “i” or “=”.

While the BlobGrid is desirable for proper understanding of mathematical expression regions, it contains no understanding of normal text regions. Easily recognizable symbols which consist of multiple connected components (i.e. “i” and “=”) are kept as separate blobs within the grid. Running Tesseract's language specific OCR on these separated symbols proved to be largely inaccurate. For instance an “=” symbol run through Tesseract's character recognizer would consist of a recognition being made for each horizontal line in the “=”. Surprisingly, in one trial the individual horizontal lines were often not recognized as dashes but instead as “j”'s. Similar problems were observed on various other characters. For instance periods were often mis-recognized as “o”'s.

In order to mitigate such problems, a new GridBase data structure, the BlobInfoGrid, is implemented in this work, which contains a combination of the information in the BlobGrid and information attained from running OCR on each previously determined column partition on the page. The grid's objects contain information on both the symbols which were recognized during OCR and those which were not. If information is available from OCR for a given symbol, then the recognized word to which the symbol belongs as well as its confidence rating are stored within the object. The object also contains the symbol's bounding box and the sentence to which it belongs (if applicable). Since, during the recognition stage, some blobs may be improperly merged into symbols, a second pass is made by the MEDS module in order to detect all blobs belonging to invalid words, and improperly merged blobs are thus split into separate objects in order to facilitate proper analysis of potential mathematical expression regions as illustrated in Figure 46.



2 INTRODUCTORY REVIEW

The first five show distinctly that the independent variable is x, whereas the last three do not explicitly indicate the variable and should not be used unless there is no chance of a misunderstanding.

2. The fundamental formulas of differential calculus are derived directly from the application of the definition (2) or (3) and from a few fundamental propositions in limits. First may be mentioned

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}, \text{ where } z = \phi(y) \text{ and } y = f(x). \tag{4}$$

$$\frac{dx}{dy} = \frac{df^{-1}(y)}{dy} = \frac{1}{\frac{df(x)}{dx}} = \frac{1}{\frac{dy}{dx}}. \tag{5}$$

$$D(u \pm v) = Du \pm Dv, \qquad D(uv) = uDv + vDu. \tag{6}$$

$$D\left(\frac{u}{v}\right) = \frac{vDu - uDv}{v^2},^{*} \qquad D(x^n) = nx^{n-1}. \tag{7}$$

It may be recalled that (4), which is the rule for differentiating a function of a function, follows from the application of the theorem that the limit of a product is the product of the limits to the fractional identity $\frac{\Delta z}{\Delta x} = \frac{\Delta z}{\Delta y}\frac{\Delta y}{\Delta x}$; whence

$$\lim_{\Delta x \doteq 0}\frac{\Delta z}{\Delta x} = \lim_{\Delta x \doteq 0}\frac{\Delta z}{\Delta y}\cdot\lim_{\Delta x \doteq 0}\frac{\Delta y}{\Delta x} = \lim_{\Delta y \doteq 0}\frac{\Delta z}{\Delta y}\cdot\lim_{\Delta x \doteq 0}\frac{\Delta y}{\Delta x},$$

which is equivalent to (4). Similarly, if $y = f(x)$ and if x, as the inverse function of y, be written $x = f^{-1}(y)$ from analogy with $y = \sin x$ and $x = \sin^{-1} y$, the relation (5) follows from the fact that $\Delta x/\Delta y$ and $\Delta y/\Delta x$ are reciprocals. The next three result from the immediate application of the theorems concerning limits of sums, products, and quotients (§ 21). The rule for differentiating a power is derived in case n is integral by the application of the binomial theorem.

$$\frac{\Delta y}{\Delta x} = \frac{(x+\Delta x)^n - x^n}{\Delta x} = nx^{n-1} + \frac{n(n-1)}{2!}x^{n-2}\Delta x + \cdots + (\Delta x)^{n-1},$$

and the limit when $\Delta x \doteq 0$ is clearly nx^{n-1}. The result may be extended to rational values of the index n by writing $n = \frac{p}{q}$, $y = x^{\frac{p}{q}}$, $y^q = x^p$ and by differentiating both sides of the equation and reducing. To prove that (7) still holds when n is irrational, it would be *necessary to have a workable definition of irrational numbers* and to develop the properties of such numbers in greater detail than seems wise at this point. The formula is therefore assumed in accordance with *the principle of permanence of form* (§ 178), just as formulas like $a^m a^n = a^{m+n}$ of the theory of exponents, which may readily be proved for rational bases and exponents, are assumed without proof to hold also for irrational bases and exponents. See, however, §§ 18–25 and the exercises thereunder.

* It is frequently better to regard the quotient as the product $u \cdot v^{-1}$ and apply (6).
† For when $\Delta x \doteq 0$, then $\Delta y \doteq 0$ or $\Delta y/\Delta x$ could not approach a limit.

Figure 45 (Top center) The input image. (Bottom-left) The CPGrid of the image which is the input to the MEDS module. (Bottom-right) The result of converting the CPGrid back into a BlobGrid. Each rectangle on the image represents a blob. Blobs colored cyan are ones for which the hybrid analysis was unable to determine whether the blob represents part of text or part of an image. Yellow blobs have been labeled as “vertical text”.

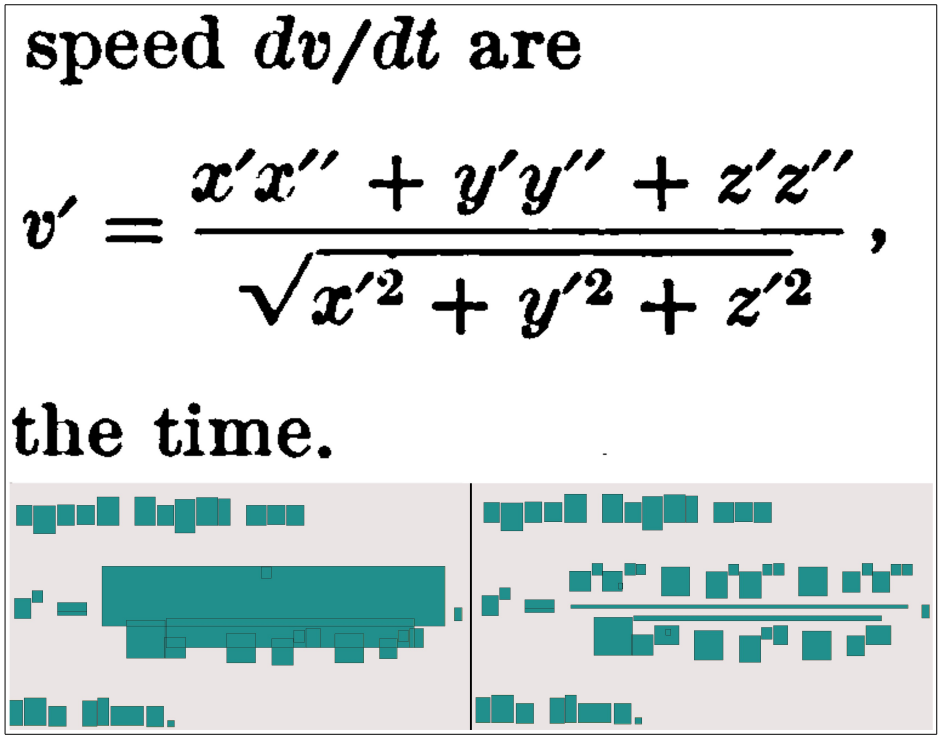


Figure 46 Both the bottom left and bottom right images correspond to the mathematical expression region shown in the upper image. The left image shows the symbols remaining after OCR by Tesseract. Notice that many of the symbols were ruled out as junk or improperly merged to their nearest neighbors. On the right is the same expression region after the new MEDS module noticed that the blobs in the region did not belong to valid words, split them back up, and reinserted them into the grid for proper analysis.

The new grid data structure, called the BlobInfoGrid is used as input to all subsequent feature extraction, classification, and segmentation techniques. The detection module utilizes a supervised machine learning approach in order to predict which non-noise connected components on a page are most likely to belong with printed mathematical expressions. Detected regions are considered as the initial seeds to mathematical expression segmentation carried out in the second stage. The primary motivation is not necessarily to maximize detection accuracy on the groundtruth data set, but rather to allow for generalized prediction accuracy on unforeseen pages. While perfect accuracy is not expected, it is important that at least a single component for each mathematical expression zone is detected at this stage since these components may then be merged with their nearest neighbors in a later step for proper segmentation. Thus, for regions wherein a single component has already been detected, false negatives may be of very little importance. False

positives, however, will likely be impossible to account for in later stages without harming MEDS accuracy. The goal of the detection step is thus to detect at least a single connected component of each expression zone while minimizing false positives. The remainder of each detected zone can then be combined with its neighbors using various heuristics during the segmentation stage of the MEDS module.

A problem observed with the resulting BlobInfoGrid structure is that certain mathematical characters would, in some instances, be missing. Tesseract's framework had, in fact, discarded the characters as noise prior to running the MEDS module or performing any sort of recognition. Although this problem occurs rarely, it certainly occurs often enough to be a significant problem. Since resolving such an issue would require modification to the Tesseract framework external to the MEDS module it is outside of the scope for the current work. An idea for future work however, is to modify Tesseract's framework to be less harsh on discarding perceived noise prior to running any math detection/segmentation modules.

There are two primary components to the detection module: (1) training and (2) prediction. The training step extracts chosen features from a small subset of the groundtruth, runs and evaluates the binary classifier on these features multiple times in order to learn classifier and feature specific parameters. These parameters are then later used by the classifier to make decisions about unforeseen data during prediction. Accuracy is evaluated by testing the prediction accuracy on the non-training subset the groundtruth dataset, and then subsequently evaluated through visual inspection on random unforeseen data which are completely unrelated to the groundtruth.

Training and Classification

This section discusses various classifier optimization and design techniques which have been considered, discusses how training is carried out in the Detection subsystem, then discusses the SVM binary classifier used for this implementation.

Classifier Design and Model Selection Techniques Considered

Cross validation. A labeled training set is randomly divided into m disjoint sets of equal size n/m, where n is the total number of labels in the training set. The classifier is trained m times, each time with a different set held out as a validation set. The average of the m validation errors is considered as the classifier's performance. By adjusting parameters so as to minimize this error, it is hoped that the classifier's accuracy will generalize better to new data.

Jackknife. Train the classifier *n* times (*n* is the total number of labels in the set), each time using the entire training set from which a different single training point has been deleted. Each resulting trained classifier is then tested on the single deleted point and overall accuracy is estimated as the mean of all test results. Jackknife is especially good for comparing two classifiers, to see if the difference in accuracy between them is statistically significant. Jacknife is essentially the same as leave-one-out cross-validation.

Bootstrap. Samples of any size less than that of the training set are randomly pulled from the labeled training set with replacement (i.e., the same label may be used multiple times). The classifier is trained on all of the samples and tested on the others. The average of all of the accuracies is measured. Classifier parameters can be adjusted during training in order to minimize the average error.

Bootstrap with Aggregation (Bagging). Results of multiple classifiers trained on the bootstrap dataset are pooled together in some fashion to get the final result during prediction. The multiple classifiers are typically all the same type (i.e. SVM, neural net, decision tree, etc) but have different parameters due to being trained on different sets. The results of all of the classifiers are typically combined through a voting mechanism.

Boosting. Multiple weak learners (classifiers with accuracy only slightly better than chance) are combined in order to achieve arbitrarily high accuracy on the training set. A subset of labels of some size less than the total number of labels is randomly selected without replacement and the first classifier is trained on this subset. A second training set is then selected so that half of the patterns in it are correctly classified by the first classifier then half are incorrectly classified by it. Boosting techniques vary based upon how this second and subsequent training subsets are chosen. An example boosting technique in order to train three classifiers described in [133] involves flipping a fair coin to decide between the following:

(1) Select remaining labels from the dataset (not in the already extracted subset) presenting them one by one to the existing classifier until it misclassifies one of them. The misclassified label is added to the new subset.

(2) Add a pattern that the first classifier classifies correctly.

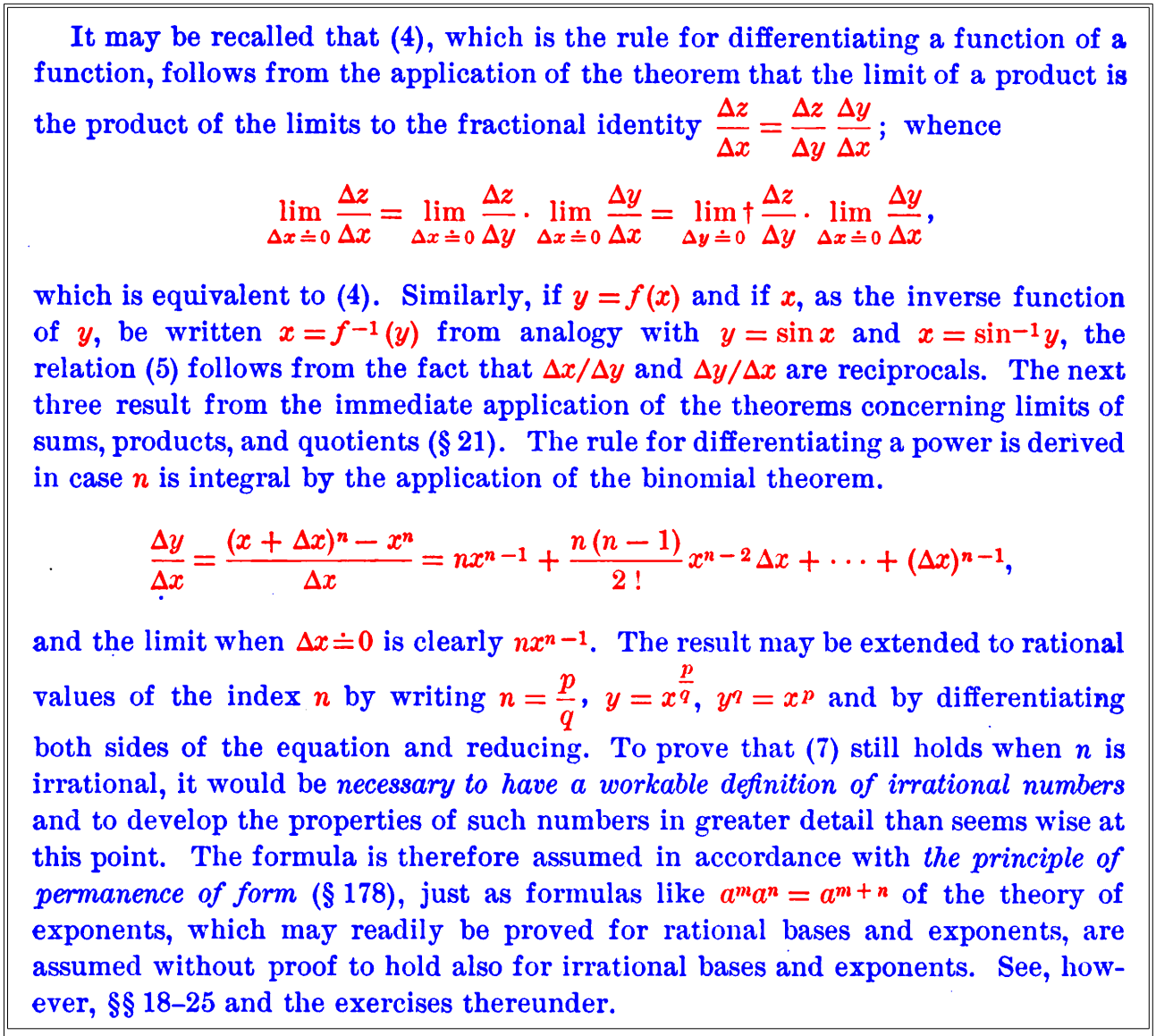
This is continued until no more patterns can be added in this manner. Thus half of the patterns in the new subset are correctly classified by the first classifier and the other half are not. The second classifier is then trained on this new dataset. A third dataset is then found by presenting remaining labels (not in either of the first two

sets) to the first two classifiers. If the two classifiers disagree, then the label is added to the third dataset, otherwise it is ignored. Finally the third classifier is trained on the third dataset.

Adaptive Boosting (AdaBoost). Each label receives a weight that determines its importance for training new classifiers. If a label is accurately classified, then its chance of being used again in subsequent classifiers is reduced, while if it is not accurately classified its chance of being used again is raised. The algorithm starts by assigning a uniform weight to each label in the training set. On each iteration a subset is drawn according to the weights (initially at random since they are uniform) and a classifier is trained on the subset. Next the weights of misclassified labels are increased and correctly classified label weights are decreased. Labels chosen based upon these new weights are then used to train the next classifier and the process is repeated until the desired number of classifiers are trained. During prediction, the weighted output of all classifiers are combined and the final classifier decision is made using the sigmoid function on this summed output.

Training Process

Training samples are first automatically generated by running the feature extractor on all BlobInfo objects on the grid. Each BlobInfo object then stores a feature vector of floating point values, each element of which is normalized between [0,1]. For each BlobInfo element on the grid, a training sample is created which consists of the blob's feature vector, it's binary label (math/non-math), and the blob's bounding box on the image. A label is automatically assigned to each blob by comparing its bounding box to those of bounding boxes in the manually generated groundtruth described in Section 3.4.1. Any blob that intersects with a groundtruth box is labeled as math while all others are labeled as non-math in their corresponding samples as illustrated in Figure 47.



It may be recalled that (4), which is the rule for differentiating a function of a function, follows from the application of the theorem that the limit of a product is the product of the limits to the fractional identity $\frac{\Delta z}{\Delta x} = \frac{\Delta z}{\Delta y}\frac{\Delta y}{\Delta x}$; whence

$$\lim_{\Delta x \doteq 0} \frac{\Delta z}{\Delta x} = \lim_{\Delta x \doteq 0} \frac{\Delta z}{\Delta y} \cdot \lim_{\Delta x \doteq 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta y \doteq 0} \dagger \frac{\Delta z}{\Delta y} \cdot \lim_{\Delta x \doteq 0} \frac{\Delta y}{\Delta x},$$

which is equivalent to (4). Similarly, if $y = f(x)$ and if x, as the inverse function of y, be written $x = f^{-1}(y)$ from analogy with $y = \sin x$ and $x = \sin^{-1} y$, the relation (5) follows from the fact that $\Delta x/\Delta y$ and $\Delta y/\Delta x$ are reciprocals. The next three result from the immediate application of the theorems concerning limits of sums, products, and quotients (§ 21). The rule for differentiating a power is derived in case n is integral by the application of the binomial theorem.

$$\frac{\Delta y}{\Delta x} = \frac{(x + \Delta x)^n - x^n}{\Delta x} = nx^{n-1} + \frac{n(n-1)}{2!} x^{n-2}\Delta x + \cdots + (\Delta x)^{n-1},$$

and the limit when $\Delta x \doteq 0$ is clearly nx^{n-1}. The result may be extended to rational values of the index n by writing $n = \frac{p}{q}$, $y = x^{\frac{p}{q}}$, $y^q = x^p$ and by differentiating both sides of the equation and reducing. To prove that (7) still holds when n is irrational, it would be *necessary to have a workable definition of irrational numbers* and to develop the properties of such numbers in greater detail than seems wise at this point. The formula is therefore assumed in accordance with *the principle of permanence of form* (§ 178), just as formulas like $a^m a^n = a^{m+n}$ of the theory of exponents, which may readily be proved for rational bases and exponents, are assumed without proof to hold also for irrational bases and exponents. See, however, §§ 18–25 and the exercises thereunder.

Figure 47 Above is a page from [127] part of an image used in the training set. After the BlobInfoGrid was generated for this image, each blob in the grid was automatically assigned a binary label (math or non-math) based upon the blob's location in reference to any entry within the page's manually generated groundtruth. For debugging purposes, the foreground of blobs which are labeled by the groundtruth as math were automatically colored red while those which were not were colored blue.

Once each blob is assigned to a label, the labels are fed into whatever training module is being utilized. The DLib Machine Learning Library [134] is used extensively in this work for training and binary classification purposes. The library includes several versions of the Support Vector Machine classifier (SVM), Multilayer Perceptron (MLP), and Bayesian Networks and also includes a cross validator implementation useful for optimal parameter selection.

Training Data Selection. With a groundtruth only available for 75 images, there are not very many options for selecting training pages. In this work, 15 randomly selected pages from one text book [127] are used for training purposes. Since it is desired to prove that the classification techniques outlined here can generalize well it was decided to not use data from more than one book, especially since there are currently only five books to choose from in the entire groundtruth dataset. Since the 15 pages used for training belong to the same book from which 30 of the total groundtruth images belong (the other four books each have 15 randomly selected images which were added to the dataset), testing is carried out both with and without the other 15 pages from the same book. If accuracy is significantly higher for the dataset with the 15 pages from the same book then low generalization and overtraining will become a major concern.

Binary Classification

For purposes of detecting mathematical connected components, a Support Vector Machine (SVM) is utilized in this work. Each character or connected component of the image is first assigned a normalized feature vector by the Feature Extraction sub-module to be described in the next section. This feature vector is then fed into a SVM binary classifier in order to determine whether the component is math or non-math. The SVM classification technique, first proposed in 1992 by Vapnik et al. [135], non-linearly maps a feature vector to a higher dimensional space where a linear decision surface is constructed. During training the SVM finds a hyperplane in the higher dimensional feature space with maximal margin between the vectors of the two classes as illustrated in Figure 48 [136]. The optimal hyperplane is constructed using the support vectors. The support vectors are a subset of the training samples which are closest to the decision plane while also maximally separating the two classes (labeled as -1 and +1 as shown in Figure 48).

Determining this optimal hyperplane involves first choosing the non-linear kernel function which will map the input feature vectors into a higher dimensional feature space suitable for the SVM's application. Although the dimensional space of the transformed feature vector is potentially infinite after the kernel is applied, little computational complexity is added since the optimal hyperplane algorithm (see Appendix A.1 in [137] for the mathematical details) uses the scalar results of inner products from the increased feature space rather than carrying out any of its computations in that space. The kernel function chosen must satisfy Mercer's Condition, meaning that any resulting matrix from applying the kernel to all of the

feature vectors must be guaranteed to always be positive semi-definite (i.e. $\boldsymbol{a}M\boldsymbol{a}^T >= 0$ for all $\boldsymbol{a}$ where M is the resulting matrix and $\boldsymbol{a}$ is any $1 \times N$ vector, and where N is the number of rows in the matrix). Mercer's Condition guarantees that a higher dimensional feature space does indeed exist for any possible set of feature vectors to be kernelized (see p. 283 of [137] for the definition of Mercer's Condition). Standard kernel techniques such as the Radial Basis Function (RBF), Polynomial, and Linear have all been proven to satisfy Mercer's Condition.

While, for some situations, a linear decision boundary may be possible with the unaltered input feature space, the RBF (Gaussian) kernel is often cited as the most standard kernel function for this task, and is used for the SVM in this work. The RBF kernel replaces the inner product of the feature vector with the following operation repeated over every combination of values in the vector during the quadratic hyperplane optimization algorithm:

$$K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0$$

The value for gamma is one that needs to be fine-tuned through one of the aforementioned parameter selection techniques. For purposes of this work, cross validation is used in order to fine-tune the γ parameter as the C parameter explained in the following paragraph.

While the original SVM algorithm proposed in 1992 was implemented for the restricted case where the training data can be separated without any errors, the technique was extended in 1995 [137] to work on training data on which some labeling errors are to be expected. The resulting SVM formulation, often referred to as the soft margin, C formulation, or the C-Support Vector Classifier (C-SVC), has become the most widely used in practice and is illustrated in Figure 48. The C-SVC introduces a slack variable, ξ, for each vector which quantifies margin error (i.e. deviation from the expected minimum hyperplane distance based upon the support vectors). Each error, ξ, is added up to give a metric for the total amount of margin error for the given hyperplane. The constant, C, is a parameter set by the user which specifies how much total margin error can be tolerated while still achieving the optimal hyperplane. This parameter, like γ, is chosen through cross-validation in this work. Through cross-validation, the combination of γ and C which gives the best results on the training data can be chosen.

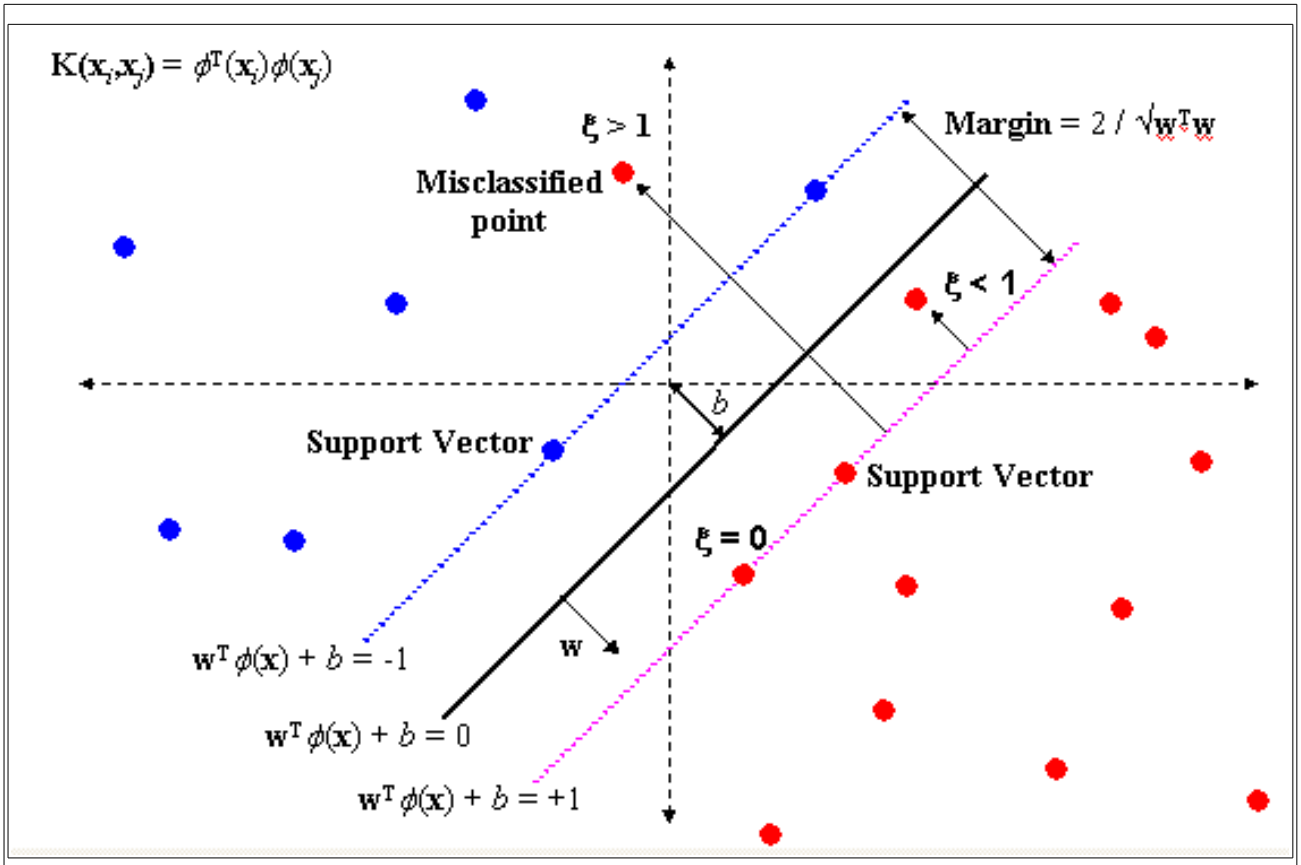


Figure 48 Illustration of the soft-margin, C formulation (C-SVC) of the SVM binary classifier [136]. The w vector and b give the optimal hyperplane within the higher dimensional feature space specified by the kernel function ϕ. The ξ values are deviation measurements of each sample from the hyperplane's expected minimal margin specified by the support vectors. The soft-margin SVM's C parameter specifies the total deviation that can be tolerated.

Fine Tuning of SVM Parameters

As previously mentioned, the parameters C and γ need to be fine-tuned during training. Among parameter selection techniques that were considered for this purpose are cross-validation, jack-knife (leave one out cross-validation), bootstrap, and boosting as described at the beginning of this section. A potential problem with the bootstrap technique for parameter selection is that the entire training set may or may not be used since samples are drawn with replacement. Depending upon the nature of the training set and possibly random chance this may or may not prove to be an effective mechanism for parameter selection. Bootstrapping with Aggregation (bagging) involves combining the decisions of several classifiers trained on different bootstrapped datasets during prediction. Bagging has been observed to increase the accuracy of unstable classifiers (i.e. classifiers for which small changes in the training set lead to significant changes in classifier output) [133]. Utilizing an ensemble of

SVM's through bagging has been shown to greatly increase classifier performance for handwritten digit recognition [138], and is thus considered as a potential avenue for experimentation. Like bagging, boosting also operates by combining the output of an ensemble of classifiers. Boosting, however is considered most effective for training an ensemble of weak learners so that their combined decision is highly accurate. Since the SVM is most certainly not a weak learner, the effectiveness of the boosting technique may be limited for the current approach.

Cross validation is a useful parameter selection technique which has often been observed to yield favorable results in practice. While 10-fold cross-validation has been observed to give favorable performance in many scenarios, the decision of how much to divide the training data is application dependent. If $m = n$ fold cross validation is carried out where the training set size is n (i.e. leave one out cross validation/jackknife), the classifier yielding the best performance in cross-validation is more likely to have too high of a variance, be too sensitive to and over-fit the training set and thus may not generalize well to new data. If too small of an m value is chosen, on the other hand, the resulting classifier may not be sensitive enough to both the training set and new data, have too much bias, and would be under-fit to the training data.

In this work, the classifier parameters are fine-tuned using 10-fold cross-validation. The 10-fold cross-validation model selection procedure is carried out by running cross-validation on the training set with possible combinations of the model parameters C and γ and then choosing the parameters that yield the best average results. To enhance numerical stability during training, all training samples are each subtracted by the training sample mean (for each individual feature value in each sample) and then divided by the training sample standard deviation prior to training being carried out. While this simple operation had no effect on classifier accuracy, it was observed to significantly speed up the training procedure from taking several hours (or even days) to taking less than or slightly more than one hour when training with over 30,000 samples and using a separate CPU core for each cross-validation. As recommended in [139], an initial starting point for C and γ parameters is found through a coarse parameter selection technique which runs the cross-validation initially at a low value for both parameters (1e-3 and 1e-7 for C and γ respectively) and then exponentially increments each one in turn up to a maximum value (1,000 and 100 are the empirically chosen values used here for C and γ respectively). A total of 10 possibilities are tested on each parameter (thus a total of 100 10-fold cross

validations). Whichever (C, γ) pair yields the highest sum of sensitivity (true positive rate) and specificity (true negative rate) is the pair that is selected as the starting point for subsequent finer-tuned parameter selection. The final parameters are then found by feeding them into the *Bound Optimization by Quadratic Approximation* (BOBYQA) algorithm [140], a C++ implementation of which is conveniently included in the D-Lib Machine Learning Library [134]. As with the coarse parameter selection, the finer-tuned BOBYQA parameter selection finds the (C, γ) pair which maximize the sum of sensitivity and specificity.

Feature Extraction

While the D-Lib Machine Learning Library [134] is utilized for both the training and classification steps in this work, all of the feature extraction steps were implemented in-house. To achieve desired results in binary classification it is of the utmost importance that the features extracted for each character be highly descriptive at distinguishing math from non-math. For this work, the features implemented can be categorized as either “spatial” or “recognition-based”. While spatial features describe a character's spatial relationship with regard to its surrounding characters, recognition-based features are any features that can be gleaned from OCR results. Both the spatial and recognition-based features implemented in this work are described in this section. All feature values that are not already scaled, are scaled between 0 and 1 using the normalization mapping of $1 - e^{-x}$, where x is the un-scaled feature value. The rationale behind using this normalization technique is for slight deviations in the quantities being measured to result in a significant change to the feature as recommended in [116].

Spatial Features

Number of Horizontally or Vertically Aligned Characters. In many mathematical equations there are seen elements which essentially “cover” multiple adjacent elements that are either horizontally or vertically adjacent depending upon the context as illustrated by Figures 49 and 50. If the center of a horizontally adjacent character is within the vertical bounds of the current character's bounding box, then that character is “covered” by the current one as shown in Figure 49. This also applies when a vertically adjacent character's center is within the horizontal bounds of the current character as shown in Figure 50.

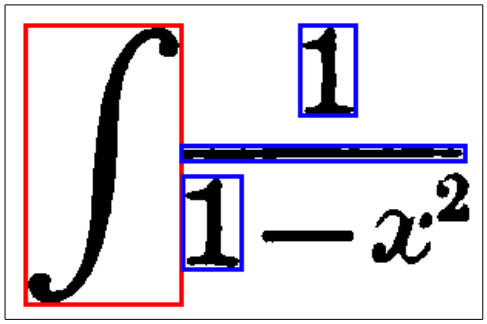


Figure 49 When the integral symbol is analyzed in the above expression, it is measured to have three horizontally adjacent vertically overlapping elements to its right. The red and blue bounding boxes were drawn manually a for illustration purposes, where the red box "covers" the blue boxes.

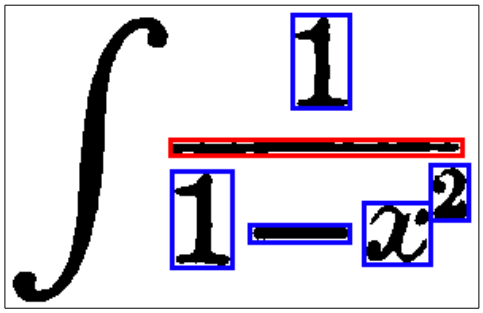
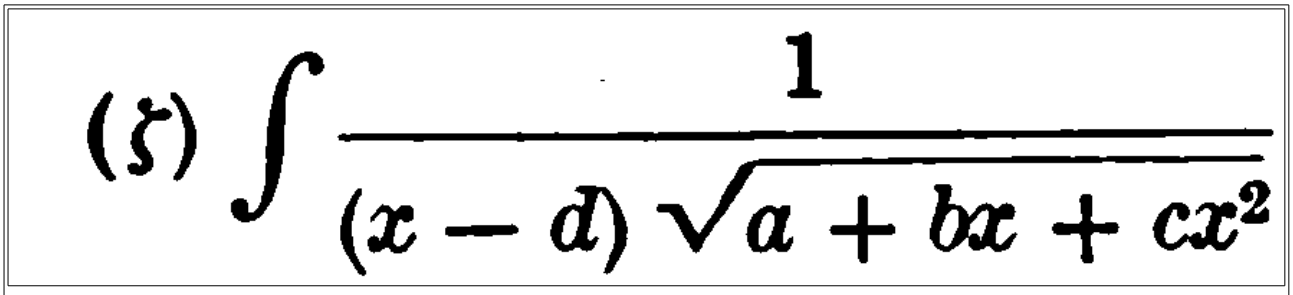


Figure 50 When the fraction bar in the above expression is analyzed it is found to have one vertically adjacent horizontally overlapping element above and four below. The red and blue bounding boxes were manually added for illustration purposes, where the red box "covers" the blue boxes.

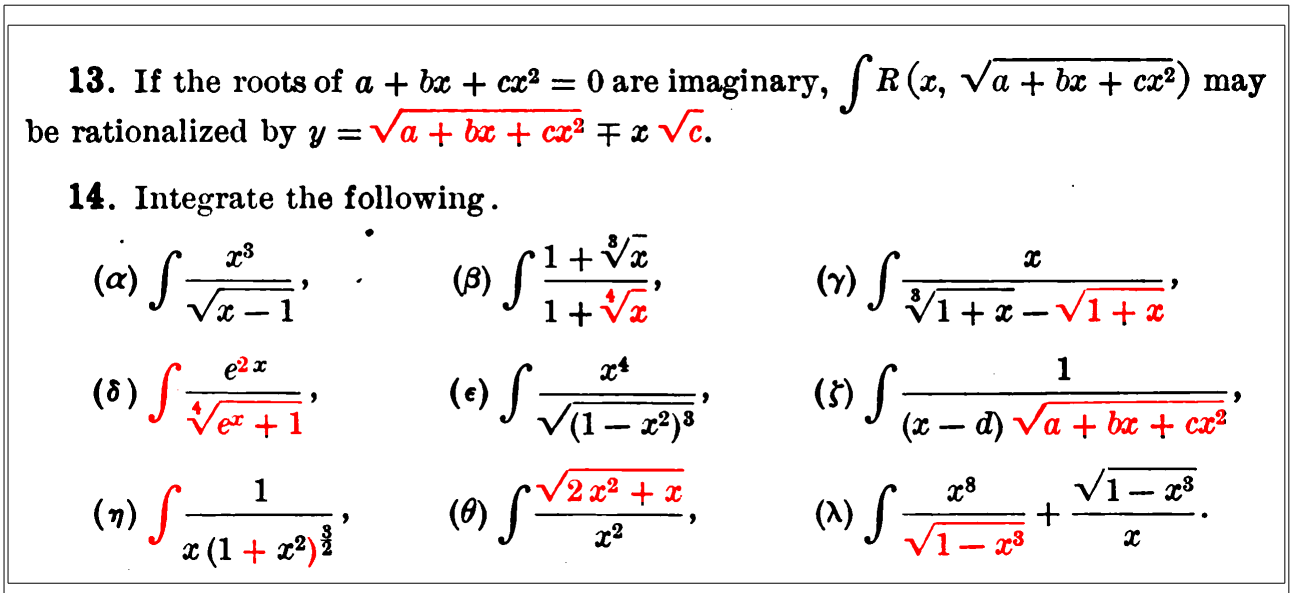
For a character to be considered adjacent however, it has been empirically found that it must be within half the character's height if looking for horizontally adjacent neighbors and half the character's width if looking for vertically adjacent neighbors. Each character is assigned a number of elements greater than or equal to zero to which it "covers," based upon the definition specified here, as a feature.

Number of Completely Nested Characters. Similar to the above feature, however less often observed, are symbols which have multiple characters effectively nested inside of their bounding box. This has primarily been observed for square roots and is not very often observed otherwise in practice. If the number of characters completely inside of the current character is greater than one, then the feature may be useful as illustrated for the square root shown in Figure 51. Figure 52 demonstrates results of the nested character feature on a training image.



$$(\zeta) \int \frac{1}{(x-d)\sqrt{a+bx+cx^2}}$$

Figure 51 The square root in the above expression contains 8 elements nested within it.



13. If the roots of $a + bx + cx^2 = 0$ are imaginary, $\int R\left(x, \sqrt{a+bx+cx^2}\right)$ may be rationalized by $y = \sqrt{a+bx+cx^2} \mp x\sqrt{c}$.

14. Integrate the following.

$$(\alpha) \int \frac{x^3}{\sqrt{x-1}}, \qquad (\beta) \int \frac{1+\sqrt[3]{x}}{1+\sqrt[4]{x}}, \qquad (\gamma) \int \frac{x}{\sqrt[3]{1+x}-\sqrt{1+x}},$$

$$(\delta) \int \frac{e^{2x}}{\sqrt[4]{e^x+1}}, \qquad (\epsilon) \int \frac{x^4}{\sqrt{(1-x^2)^3}}, \qquad (\zeta) \int \frac{1}{(x-d)\sqrt{a+bx+cx^2}},$$

$$(\eta) \int \frac{1}{x(1+x^2)^{\frac{3}{2}}}, \qquad (\theta) \int \frac{\sqrt{2x^2+x}}{x^2}, \qquad (\lambda) \int \frac{x^8}{\sqrt{1-x^3}} + \frac{\sqrt{1-x^3}}{x}.$$

Figure 52 Results of the nested character feature on a training image demonstrated through foreground region coloring. The red blobs are inside the bounding box of a nested blob. The square-root next to (α) was filtered out by Tesseract as noise prior to running the MEDS module. The other three missed square roots were broken into two blobs and thus have no nested characters. The integrals, parenthesis, and plus sign were all mis-recognized by Tesseract as containing more than one character.

Subscripts or Superscripts. The presence of superscripts and subscripts can often, but not always, infer presence of mathematical notation. An example of a situation where mathematics cannot be inferred from superscripts and subscripts is the presence of footnotes. Used as a single binary feature in conjunction with multiple other features, the presence of super and subscripts can be informative, however. A character is assigned four binary features (1 if true 0 otherwise) based upon whether it has a superscript, has a subscript, is a superscript, or is a subscript. A character has a superscript if a horizontally adjacent character to the right vertically overlaps at least to some extent but also has a bottom that is around or above the character's

vertical center. Likewise, a character has a subscript if a horizontally adjacent character to the right which vertically overlaps at least to some extent has a top that is around or below the current character's vertical center. A character is a super/subscript if its neighbor which is found to have a subscript assigns it as its own super/subscript.

As demonstrated by Figures 53-55, the superscript/subscript feature can be informative but also misleading in some circumstances. In Figure 53, for instance, a fraction bar is seen as the superscript of part of an "=" symbol. Punctuation often meets the previously mentioned requirements of being a subscript. In order to prevent false detection of punctuation as subscripts, the feature extraction technique makes sure that, for normal text, the last character is non-punctuation during subscript detection. Furthermore when detecting a superscript or subscript the current blob must be the rightmost on its word whereas the neighboring blob (the potential superscript/subscript) must be the left-most blob on its word. When a blob belongs to a mis-recognized word, as shown in Figures 53-55, this precaution is of little help however. An area threshold is also employed in order to prevent noise from being mistaken for sub/superscripts.

$$\frac{dy}{dx} = \frac{R\sin\tau + (x-a)}{R\cos\tau - (y-y_0)}, \qquad \left(\frac{dy}{dx}\right)_{a,\,y_0} = \tan\tau = f'(a),$$

$$\frac{d^2y}{dx^2} = \frac{[R\cos\tau - (y-y_0)]^2 + [R\sin\tau + (x-a)]^2}{[R\cos\tau - (y-y_0)]^3}, \qquad \left(\frac{d^2y}{dx^2}\right)_{a,\,y_0} = \frac{1}{R\cos^3\tau},$$

and $$y = y_0 + (x-a)f'(a) + \tfrac{1}{2}(x-a)^2\frac{1}{R\cos^3\tau} + \cdots$$

Figure 53 Result of sub/superscript detection on a training image depicted by foreground region coloring in order to illustrate the feature's reliability. The red blobs were detected to have a sub/superscript, the green blobs are superscripts, and the blue blobs are subscripts. Due to the page's spacing, the d in d^2y was not found to have a square since the bottom of the 2 is above d's center. The large parenthesis were also found to have sub/superscripts based upon the criteria and since it is not known that they are themselves punctuation due to improper recognition.

Figure 54 The word "Simpson's" was mis-recognized by Tesseract, resulting in the apostrophe being mistaken for a super-script.

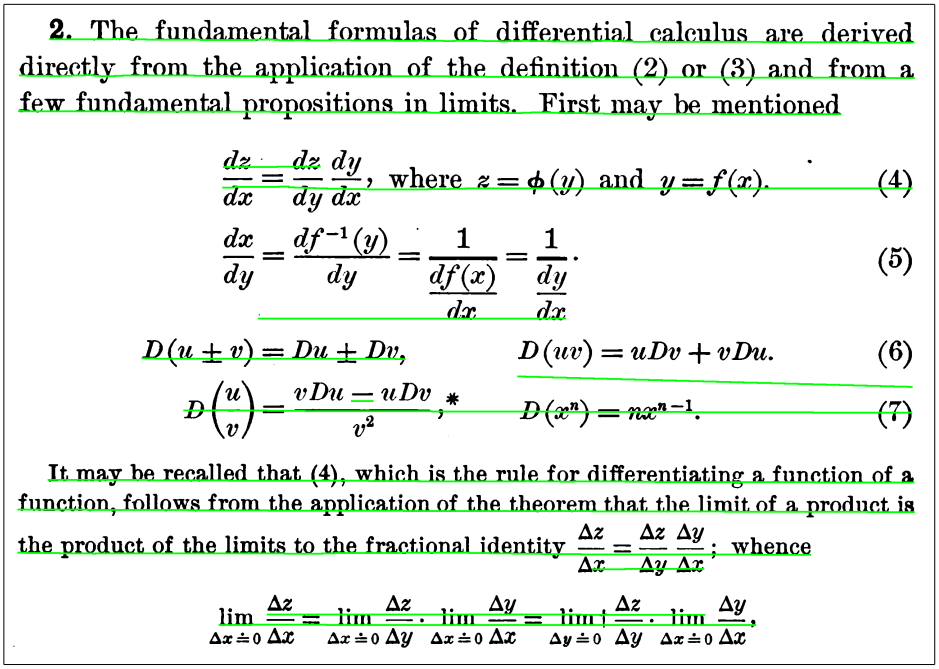
Figure 55 The word, "right-" was mis-recognized by Tesseract, resulting in the bottom of the "g" being mistaken for a subscript, the left part of the "h" mistaken as a superscript, and the "-" being mistaken for a subscript of the right part of the "h" and the "t" which were improperly combined into a single character.

Character Height. Mathematical characters like integral signs, exponents, square roots, etc. are observed to have heights which differ from the height of normal text on a page. A distinguishing feature of some mathematical regions is thus their difference from the average normal text height. Normal text is defined here as any word for which a valid match is found in Tesseract API's dictionary as described in [29]. The height of all characters or connected components belonging to valid words is averaged over the image to give the average normal text height. The character feature is then measured for each character as h/h_μ where h is the character height and h_μ is the average normal text height on a page. If there is no normal text found on the page then the character height feature is simply h.

Character Width to Height Ratio. The width to height ratio feature is primarily utilized in helping to detect fraction bars. Just as with the character height feature, the average width/height ratio is first taken for all normal text on a page. The width/height feature is then measured as r/r_μ where r is the width/height ratio of the character being measured and r_μ is the average width/height ratio for normal text on

the page. If there is no normal text on the page the width/height ratio is simply r and a flag is set on a separate binary feature to denote that there exists no normal text on the page. This binary flag is set in order to prevent the classifier from being confused by the new measurements it may receive for pages without normal text. In this work, however, all pages tested will have at least some normal text.

Vertical Distance Above Row Baseline (VDARB). Fraction numerators, fraction bars, and exponents in embedded expressions are positioned above the baseline wherein normal text is expected to reside. The baseline for each row of text found by Tesseract is computed by fitting a quadratic spline to groups of blobs using a least squares technique as described in [34]. The fitted baseline can be useful for detecting outliers on normal text lines, however it loses its meaning for non-normal text lines like displayed expressions where the fitted baseline is often incorrectly computed as shown in Figure 56. It was deemed that a row must therefore contain at least one valid word in order for this feature to be meaningful. If a character resides on a row which contains at least one valid word, then the difference between that character's bottom-y coordinate to the y coordinate of the row's baseline at the character's x position is computed. Since normal characters like the character “p” often have their bottom residing well below the baseline, all characters with a negative distance from the baseline are assigned to 0 for this feature, unless the top of the character is below the baseline as well, in which case the absolute value of the distance is used. The feature is then subtracted by the average vertical baseline distance for the normal text on the given row, divided by the row's height, and then normalized between [0,1] exponentially. If the character does not reside on a row with at least one valid word then this feature is fixed to 0.



2. The fundamental formulas of differential calculus are derived directly from the application of the definition (2) or (3) and from a few fundamental propositions in limits. First may be mentioned

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}, \text{ where } z = \phi(y) \text{ and } y = f(x). \tag{4}$$

$$\frac{dx}{dy} = \frac{df^{-1}(y)}{dy} = \frac{1}{\frac{df(x)}{dx}} = \frac{1}{\frac{dy}{dx}}. \tag{5}$$

$$D(u \pm v) = Du \pm Dv, \qquad D(uv) = uDv + vDu. \tag{6}$$

$$D\left(\frac{u}{v}\right) = \frac{vDu - uDv}{v^2}, \qquad D(x^n) = nx^{n-1}. \tag{7}$$

It may be recalled that (4), which is the rule for differentiating a function of a function, follows from the application of the theorem that the limit of a product is the product of the limits to the fractional identity $\frac{\Delta z}{\Delta x} = \frac{\Delta z}{\Delta y}\frac{\Delta y}{\Delta x}$; whence

$$\lim_{\Delta x \doteq 0} \frac{\Delta z}{\Delta x} = \lim_{\Delta x \doteq 0} \frac{\Delta z}{\Delta y} \cdot \lim_{\Delta x \doteq 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta y \doteq 0} \frac{\Delta z}{\Delta y} \cdot \lim_{\Delta x \doteq 0} \frac{\Delta y}{\Delta x},$$

Figure 56 The baselines found during Tesseract's layout analysis were automatically drawn on several images from [127] in order to gauge the effectiveness of the VDARB feature. From the above example it is can be seen that, while Tesseract's quadratic spline line detection algorithm [34] is highly effective on normal text, results are somewhat unpredictable in the presence of mathematical expressions.

Count of Stacked Characters at Character Position. Mathematical regions may have a two-dimensional layout that is more complex than what is observed in normal text regions. If a character is observed to be part of a non-existent or non-valid word after language recognition, then a vertical search is done above and below that character at its horizontal position to count the total chain of adjacent nearest neighbors that also belong to invalid or non-existent words. A neighbor is considered adjacent only if its distance from the current character is less than or equal to half of the initial character's height (height of the character at the position being measured). The nearest neighbor search looks up first until all characters which meet the aforementioned criteria are found and then does the same by looking down. The total number of stacked characters does not include the current character itself because the exponential normalization technique used (as described in [116]) yields much better separation this way. The method is illustrated by Figure 57.

$$\frac{d^2y}{dx^2} = \frac{[R\cos\tau - (y - y_0)]^2 + [R\sin\tau + (x - a)]^2}{[R\cos\tau - (y - y_0)]^3}$$

Figure 57 Fractions are an example of mathematical notation that is often two dimension-ally more complex than is normal language text. For the $\frac{d^2y}{dx^2}$, both of the d symbols have 2 as their stacked count features. The minus signs and exponents are the exceptions as they are assigned 0. The fraction bars are also assigned 0 because their height is very low so that no nearest neighbors can be vertically adjacent.

Examples of feature values found using this technique are shown in Figures 58 and 59. Words which are considered valid by Tesseract's dictionary method are discarded for purposes of this feature in order to avoid false positives. Unfortunately, this causes several blobs which should have a stacked feature of 1 or 2 to be fixed at 0 as can be observed in Figure 58. The dx at the bottom left of Figure 58, for instance, is considered to be a valid word by Tesseract and thus has a stacked feature fixed at 0. This also occurs for the $[R$, cos, $(y$, and $(x$. The sin is a somewhat unusual circumstance because the word was misrecognized and improperly segmented into three blobs by Tesseract: the “si”, the dot on the i, and the “n”. The “si” has one stacked feature above it because the dot on the “i” is mistaken for a separate entity. The “si”, “n”, as well as the τ however are at a distance from the fraction bar greater than half of their heights and are thus not seen as adjacent to it. The $+$, on the other hand is seen to have two adjacent elements below it: the fraction bar and the closing bracket. More results of this technique are demonstrated in Figure 59. Although by no means perfect, this feature can give a good indication of the the “geometric complexity” of a mathematical expression region as described in [115].

$$\frac{d^2y}{dx^2} = \frac{[R\cos\tau - (y - y_0)]^2 + [R\sin\tau + (x - a)]^2}{[R\cos\tau - (y - y_0)]^3}$$

Figure 58 Results of using the described stacked feature algorithm on the same expression shown in Figure 53. Dark blobs have a stacked feature of 0, red blobs have a stacked feature of 1, and green blobs have a stacked feature of 2.

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}, \text{ where } z = \phi(y) \text{ and } y = f(x). \quad (4)$$

$$\frac{dx}{dy} = \frac{df^{-1}(y)}{dy} = \frac{1}{\frac{df(x)}{dx}} = \frac{1}{\frac{dy}{dx}}. \quad (5)$$

$$D(u \pm v) = Du \pm Dv, \qquad D(uv) = uDv + vDu. \quad (6)$$

$$D\left(\frac{u}{v}\right) = \frac{vDu - uDv}{v^2}, \qquad D(x^n) = nx^{n-1}. \quad (7)$$

It may be recalled that (4), which is the rule for differentiating a function of a function, follows from the application of the theorem that the limit of a product is the product of the limits to the fractional identity $\frac{\Delta z}{\Delta x} = \frac{\Delta z}{\Delta y}\frac{\Delta y}{\Delta x}$; whence

$$\lim_{\Delta x \doteq 0}\frac{\Delta z}{\Delta x} = \lim_{\Delta x \doteq 0}\frac{\Delta z}{\Delta y}\cdot\lim_{\Delta x \doteq 0}\frac{\Delta y}{\Delta x} = \lim_{\Delta y \doteq 0}\frac{\Delta z}{\Delta y}\cdot\lim_{\Delta x \doteq 0}\frac{\Delta y}{\Delta x},$$

which is equivalent to (4). Similarly, if $y = f(x)$ and if x, as the inverse function of y, be written $x = f^{-1}(y)$ from analogy with $y = \sin x$ and $x = \sin^{-1}y$, the relation (5) follows from the fact that $\Delta x/\Delta y$ and $\Delta y/\Delta x$ are reciprocals. The next three result from the immediate application of the theorems concerning limits of sums, products, and quotients (§ 21). The rule for differentiating a power is derived in case n is integral by the application of the binomial theorem.

$$\frac{\Delta y}{\Delta x} = \frac{(x + \Delta x)^n - x^n}{\Delta x} = nx^{n-1} + \frac{n(n-1)}{2!}x^{n-2}\Delta x + \cdots + (\Delta x)^{n-1},$$

and the limit when $\Delta x \doteq 0$ is clearly nx^{n-1}. The result may be extended to rational values of the index n by writing $n = \frac{p}{q}$, $y = x^{\frac{p}{q}}$, $y^q = x^p$ and by differentiating

Figure 59 Results of using the described stacked feature algorithm. Dark blobs have a stacked feature of 0, red blobs have a stacked feature of 1, green blobs have a stacked feature of 2, and blue blobs have a stacked feature of 3 or more.

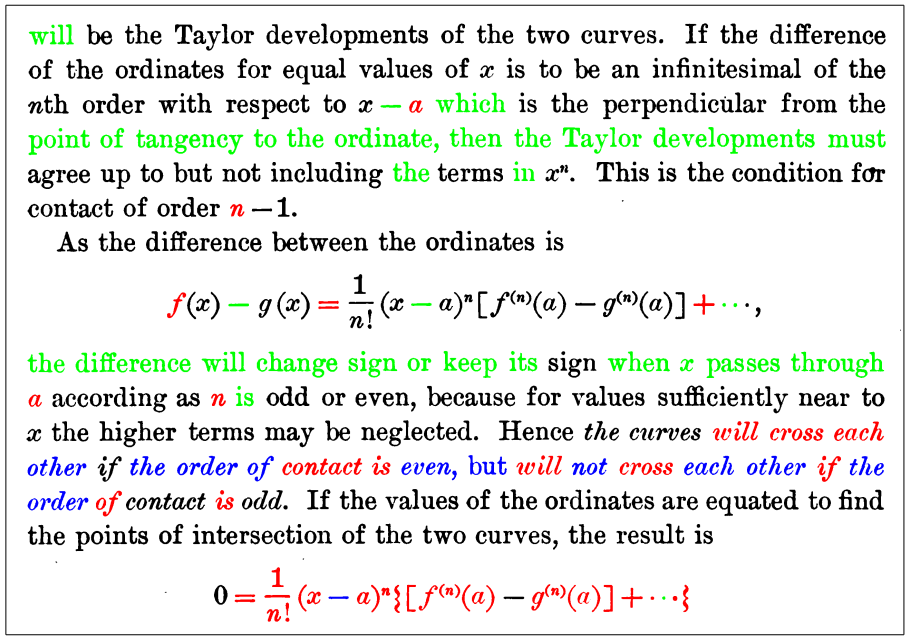
Recognition-based Features

Recognized Math Symbols or Words. The language[6] Tesseract OCR utilized in this work allows for the instant detection of some basic mathematical characters like “<”, “>”, “+”, “-”, “/”, “%”, etc. While using OCR trained specifically for

[6] Here the language classification result indicates the result of a classifier that was trained for a particular language. Although in the context of this work English is all that is tested, testing of existing techniques in various languages is of interest for future studies.

mathematics would increase accuracy significantly and allow for the detection of more complex symbols like integrals, greek letters, etc., training Tesseract for new symbols is a very time consuming task that is kept as an idea for future work. For purposes of this work a finite list of math words has been generated. If a character belongs to a word or symbol on the list then its corresponding recognized math symbol feature is set to 1, otherwise it is 0.

Italicized or Bold Text. Single italicized or bold characters among normal text have often been observed to correspond to mathematical variables. Whether or not the text is math often depends upon the context of the sentence to which they belong. A helpful feature in further distinguishing math from non-math in a sentence is linguistic analysis. If a sentence has n-grams that have been measured to extensively belong to mathematical sentences then the likelihood of bold or italicized characters in that sentence being mathematical increases. The n-gram feature used in this work is discussed in a following section. Tesseract utilizes a technique described in [29] which detects bold and/or italicized text. Unfortunately, however Tesseract's assignment of bold/italics was found to be rather unstable as illustrated in Figure 60. Since italics appear to be much more accurate than bold assignments, only italics are utilized as a feature for purposes of this work. If a blob belongs to an italic word the feature is assigned 1 otherwise it is fixed to 0.



will be the Taylor developments of the two curves. If the difference of the ordinates for equal values of x is to be an infinitesimal of the nth order with respect to $x-a$ which is the perpendicular from the point of tangency to the ordinate, then the Taylor developments must agree up to but not including the terms in x^n. This is the condition for contact of order $n-1$.

As the difference between the ordinates is

$$f(x)-g(x)=\frac{1}{n!}(x-a)^n[f^{(n)}(a)-g^{(n)}(a)]+\cdots,$$

the difference will change sign or keep its sign when x passes through a according as n is odd or even, because for values sufficiently near to x the higher terms may be neglected. Hence *the curves will cross each other if the order of contact is even*, but *will not cross each other if the order of contact is odd*. If the values of the ordinates are equated to find the points of intersection of the two curves, the result is

$$0=\frac{1}{n!}(x-a)^n\{[f^{(n)}(a)-g^{(n)}(a)]+\cdots\}$$

Figure 60 Results of Tesseract's italic and bold text assignment. Blobs colored red were detected as italic, green as bold, and blue as both bold and italic.

Method

OCR Confidence Rating. After language-specific OCR is carried out, normal text can be largely distinguished from abnormal text based upon the OCR confidence rating assigned to each recognized character. This feature primarily serves to rule out normal text from potentially mathematical text, and does not necessarily say anything about whether or not a character should be considered mathematical. After carrying out recognition on a character, Tesseract assigns to it a rating which specifies how confident the OCR engine was in making its final decision. The rating which Tesseract assigns to a character is a negative number which approaches zero for high confidence but may be as low as -20 for characters recognized with extremely low confidence. The confidence feature is computed as c/c_μ where c is the current character's confidence rating and c_μ is the average character confidence rating on the page for characters which are part of valid words. If there are no valid words on the page then the feature is simply $-c$ and a separate feature, which indicates whether or not a valid word is found on the page is set to 1 where it would normally be 0.

Linguistic Analysis (n-grams). Since the Tesseract OCR utilized in this work can recognize normal text with near perfect accuracy, linguistic analysis can be performed in order locate sentences of recognized text which are likely to contain mathematical notation. In a 2005 project by Garain and Chaudhuri [115], an in-depth statistical study was carried out on 297 document pages from books, journals, and exam papers and 103 synthetically generated document pages (from Microsoft Word and TEX). Among several other in-depth analyses, a linguistic analysis was carried out. Linguistic analysis of sentences revealed that a word-level n-gram model could be of great help in categorizing sentences into one of two categories: namely with or without mathematical expressions. In the study, uni-grams, bi-grams, and tri-grams are extracted for 870 sentences containing math and 2,655 not containing math. The n-grams are ranked separately for each category based upon frequency of occurrence. The top 150 n-grams for each class are used to generate an “n-gram Profile” for their respective class.

In Garain's work, a classification technique utilized on 877 new test sentences involved first finding the n-grams for that sentence and counting the number of the found n-grams that exist in the math and non-math n-gram Profiles respectively. If more of the sentence's n-grams were observed in the math category then the sentence is categorized as math, otherwise if more n-grams were observed in the non-math category the sentence is categorized as non-math. If the sentence has equal amounts of math and non-math n-grams then it is considered indeterminate. Test

results showed that accuracy increased for sentences with more n-grams. Accuracy ranged from 90.2% for sentences with up to 50 n-grams to 98.9% for sentences with up to 150 n-grams.

In this work, n-grams are ranked by frequency of occurrence from the result of Tesseract's OCR on 7 of the 15 pages which were taken from [127] and used for training. The rationale behind not using all 15 pages to generate n-gram Profiles is to avoid over-training of the classifier to this particular text, and, as previously mentioned, only pages from a single text were chosen for training since only 5 texts are currently available in the entire dataset. A sentence is, in this work, defined as a sequence of words starting with a valid word having a capitalized first letter and ending with either a “.” or a “?” (“!” is not used as a sentence ending due to the presence of factorials). Sentences recognized by Tesseract are first automatically separated into math and non-math by determining each sentence's region on the image and comparing that region to the manually generated groundtruth. If a sentence overlaps any groundtruth region then it is labeled as math otherwise it is labeled as non-math. Next, n-gram profiles are generated for both the math and non-math sentences. These profiles, including uni-grams, bi-grams, and tri-grams, are ranked by frequency of occurrence, and are each placed in their own text file. The matching n-grams in the non-math profile are then subtracted from matching n-grams in the math profile so that the math profile gives the frequency of occurrence of the n-grams most relevant to math sentences. If there are more math n-grams than non-math n-grams then the count of matching non-math n-grams are upweighted by the ratio math/non-math word ratio during subtraction. The updated frequencies are then used to re-rank the math n-gram profile in descending order of updated frequency. The updated math n-gram profile is then utilized in this work to generate an n-gram-based feature for all characters belonging to a given sentence. After a brief discussion of how individual characters are assigned to sentences within the context of this work, the method used to assign the n-gram feature (i.e. probability that a sentence contains math based on n-grams) to a sentence is discussed. Also the resulting n-gram profiles for the limited dataset will briefly be covered.

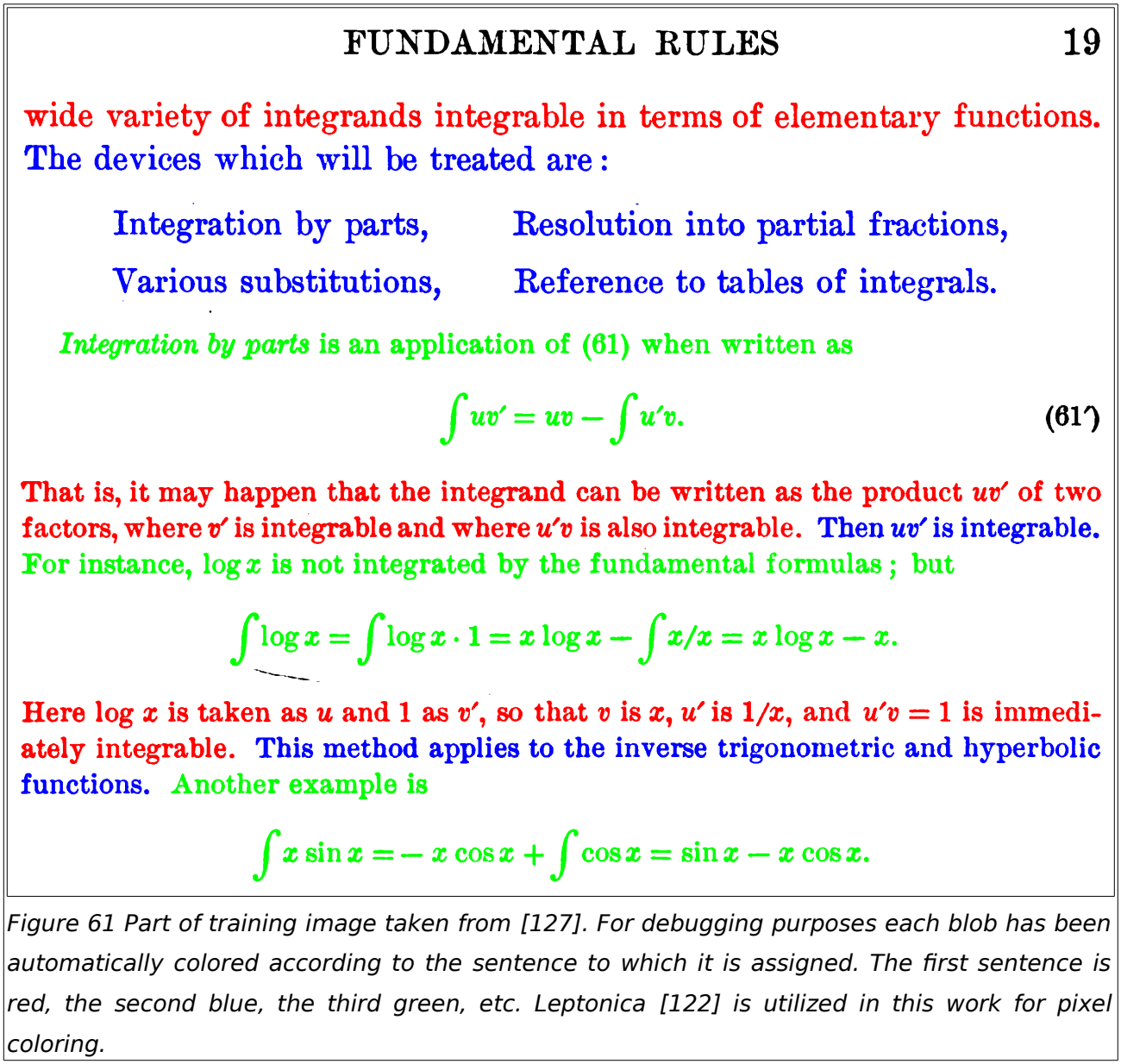
Assigning Blobs to Sentences. In order to ensure that the right n-gram feature is assigned to each blob within a sentence, it is important that each blob is physically assigned to the correct recognized sentence to which it logically belongs. Although tesseract does not store a mapping from the individual blob to the physical row of text to which it belongs, it does inherently store a row of text which points

indirectly to all of the characters that reside on that line. The results of Tesseract's page recognition are stored in a top-down fashion, starting with a small set of *blocks* each of which contains one or more *rows* with the *rows* each containing one or more *words* which each contain one or more individual *blobs*. During preparation of the BlobInfoGrid, tesseract's OCR is carried out on the entire page for which the BlobInfoGrid is being created. The result of Tesseract's OCR is a “page result” object which points to the resulting blocks, rows, words, and blobs structured in the top-down manner previously mentioned. In order to assign individual blobs to sentences, it is necessary that the blobs also have access to the word and row to which they belong which is not the case for the output of Tesseract. Since bidirectional access is not given by Tesseract's page results, some very simple but convenient data structures are implemented in this work. In order to give the BlobInfo objects access to the word and row to which they belong, a “WordInfo” object pointer is assigned to each BlobInfo object. This WordInfo object is created for each word recognized by Tesseract and contains pointers to all BlobInfo objects which reside within it, a pointer to the row to which the word belongs, a pointer to the word result from Tesseract, and sentence start and ending flags which are only true if the given word is found to reside on a sentence boundary. Likewise a “RowInfo” object pointer is assigned to each WordInfo object. The RowInfo object contains pointers to all WordInfo objects contained in it, the corresponding row result from Tesseract, a convenience function for concatenating the recognized text of all words on the row, and also stores any other convenient information: for instance the average distance of each of the row's blob's from the row's baseline if that is needed for feature extraction. BlobInfo objects for which no Tesseract recognition results are obtained are assigned to a NULL WordInfo pointer. Although pointer access is both to and from each RowInfo object and its WordInfo objects, the WordInfo objects are owned by the RowInfo object in which they reside and are deallocated upon the RowInfo object's destruction. Each of the RowInfo objects are stored in a vector belonging to the BlobInfoGrid.

Sentences are first found by iterating through the RowInfo vector in order to find the words which signify sentence boundaries. The convention adopted here is that only valid capitalized words, i.e., words which have been deemed as “valid” by Tesseract's API and consist of a capitalized letter immediately followed by a lowercase one, can be considered as sentence start boundaries. Likewise only valid words ending with a period or question mark can be considered as sentence ending boundaries. The exclamation point is not used as a sentence boundary in this work because factorial

symbols have been observed to cause otherwise valid sentences to end prematurely. Once the first start boundary word is detected, the subsequent words are checked for end boundaries. With the detection of an end boundary, the algorithm then seeks start boundaries again for subsequent words repeating the previously mentioned pattern until the last word on the page is reached. If the last word is reached and an end boundary is still being sought then the final word is flagged as the sentence's end boundary regardless of its content. Each time an end boundary of a sentence is found, a Sentence object is instantiated to store the indices of its row and word boundaries and the recognized text in the sentence. The Sentence object also stores the bounding box coordinates of each of its rows as well as the n-grams found during feature extraction. Each Sentence object is appended to a vector owned by the BlobInfoGrid.

The BlobInfo objects belonging to sentences are then each assigned an index corresponding to the Sentence object to which they belong. In order to assign the BlobInfo objects to their sentences the RowInfo vector is again iterated with the row and word indices of each WordInfo object in each row being checked against the corresponding row and word boundary indices for all of the Sentence objects. If the current WordInfo object is found to belong to a sentence then all of the BlobInfo objects which it points to, i.e., which were found to belong to the word during Tesseract page recognition, are assigned to that same sentence. Determining which sentence the WordInfo object belongs to involves iterating through each sentence object and comparing the WordInfo object's word and row indices to the corresponding sentence boundary indices as follows. If the row index is less than the Sentence start row or greater than the sentence end row then word is not part of the sentence, otherwise if blob's row index is in between the sentence's start and end row boundaries then the word is assigned to that sentence. If the word is on a sentence row boundary then the decision requires also comparing the word index as follows: if the current sentence starts and ends on the same row then the word's index must be >= the sentence start boundary and <= the sentence end boundary. If the sentence starts and ends on different rows and the word is on the starting row then its word index must be >= the sentence's start word boundary but the end word boundary does not matter. Likewise when the sentence starts and ends on different rows and the word is on the ending row of the sentence then the word's index must be <= the sentence's end boundary but the start boundary does not matter. Results of this blob sentence assignment technique are illustrated in Figure 61.



FUNDAMENTAL RULES 19

wide variety of integrands integrable in terms of elementary functions. The devices which will be treated are:

Integration by parts, Resolution into partial fractions,
Various substitutions, Reference to tables of integrals.

Integration by parts is an application of (61) when written as

$$\int uv' = uv - \int u'v. \qquad (61')$$

That is, it may happen that the integrand can be written as the product uv' of two factors, where v' is integrable and where $u'v$ is also integrable. Then uv' is integrable. For instance, $\log x$ is not integrated by the fundamental formulas; but

$$\int \log x = \int \log x \cdot 1 = x \log x - \int x/x = x \log x - x.$$

Here $\log x$ is taken as u and 1 as v', so that v is x, u' is $1/x$, and $u'v = 1$ is immediately integrable. This method applies to the inverse trigonometric and hyperbolic functions. Another example is

$$\int x \sin x = -x \cos x + \int \cos x = \sin x - x \cos x.$$

Figure 61 Part of training image taken from [127]. For debugging purposes each blob has been automatically colored according to the sentence to which it is assigned. The first sentence is red, the second blue, the third green, etc. Leptonica [122] is utilized in this work for pixel coloring.

The Tesseract recognition results for the sentences shown in Figure 61 are as shown below. It is observed that Tesseract often mistakes the lowercase “w” for a capital one and of course the mathematical text primarily results in junk output except for the u' and v' variables which are recognized perfectly in several cases.

Sentence 1: Wide variety of integrands integrable in terms of elementary functions.=
Sentence 2: The devices which Will be treated are:
Integration by parts, Resolution into partial fractions,
Various substitutions, Reference to tables of integrals.

Sentence 3: Integration by parts is an application of (61) when written as
fun’ = uv — fu’v.
Sentence 4: That is, it may happen that the integrand can be written as the product uv’ of two factors, where v’ is integrable and where u’v is also integrable.
Sentence 5: Then uv’ is integrable.
Sentence 6: For instance, logo: is not integrated by the fundamental formulas ; but floga:=flog2:-lzxloga:-—-fa:/:c=a:loga:-9:.
Sentence 7: Here log 3; is taken as u and 1 as 1:’, so that v is 2:, u’ is 1/9:, and u’v = 1 is immedi- A
ately integrable.
Sentence 8: This method applies to the inverse trigonometric and hyperbolic functions.
Sentence 9: Another example is
fa: sina: =— 3: cosa; +fcos2: :- sina: — cc cosx.

Although some junk output is assigned a sentence with the current technique, a largely useful n-gram Profile is made assuming that there is enough sentence content to analyze.

N-*gram Profile.* After feeding 7 of the 15 available training images from [127] into the n-gram Profile generator developed in this work, some interesting results were obtained as shown in the below table. Of a total of a meager 75 math sentences and 34 non-math sentences the top 20 math n-grams found after subtracting matching non-math n-grams (each matching non-math n-gram is weighted by the math to non-math total word count ratio during subtraction) are shown in Table 2.

Table 2 N-gram Profile automatically generated from 7 of the 15 training images used in this work. 75 math and 34 non-math sentences were used to generate this profile. All matching non-math n-grams are subtracted from the math n-gram counts after being weighted by the math/non-math word ratio to result in the above profile.

Tri-grams	Counts	Bi-grams	Counts	Uni-grams	Counts
area under the	4	the function	10	=	47
under the curve	4	at the	7	+	18
equal to the	4	of a	6	f	16
mass of the	4	the area	5.6532	function	10.9596
be written as	3	the mass	5.6532	approach	9
written as the	3	the curve	5	y	8
limit of the	3	the density	5	area	7.6532
the rod to	3	equal to	5	sin	7
of the rod	3	it is	4.3064	rod	7
continuous at the	3	and if	4	density	7
is not continuous	3	be written	4	mass	6.6532
not continuous at	3	the product	4	values	6
exactly equal to	2	under the	4	value	6
than the original	2	if the	4	written	5
integration by parts	2	written as	4	between	5
as the product	2	the rod	4	variable	4
a function f	2	that the	3.3064	case	4
is integrable and	2	values of	3	product	4
it is necessary	2	function f	3	integrable	4
it is clear	2	integral of	3	intervals	4

As may be expected for such a limited amount of data, a lot of the n-grams are largely specific to the particular document in which they are found. For instance the word “mass” and “rod” are seen a significant amount of times for this limited amount of sentences and are largely specific to the particular context in which the words are

used. tri-grams like “it is clear” or “it is necessary” however may be useful for a wider range of documents. Adding more data to the groundtruth dataset would make the coverage of these n-gram profiles much more powerful. For purposes of this study, a short list of empirically determined stop-words is manually generated to specify uni-grams which should likely be removed from the profile like “a”, “the”, “and”, etc.

N-gram Feature Assignment. All blobs belonging to a given sentence are each assigned their own uni-gram, bi-gram, and tri-gram features. The n-grams are first extracted from the sentence. The features each start at zero and are incremented by the count of all matching n-grams to the n-gram Profile. The features are then scaled to an interval from [0,5], with 0 being the lowest possible n-gram feature and 5 being the highest. The feature is then normalized to [0,1] using $1 - e^{-x}$ on the scaled feature. The manner in which the scaling is carried out is decided empirically and depends upon the nature of the n-gram Profile. If the highest counted object on the profile has a count less than or equal to 5, then the total n-gram feature for that sentence is kept as the original count but capped at 5. If the highest counted object on the profile has a count greater than 5 then the feature is divided by 10, an empirically chosen constant, and capped to 5 as the upper bound.

Segmentation Subsystem

The segmentation subsystem takes as its input all regions that were recognized as math by the detection subsystem previously explained (the seed regions), and merges them with their neighbors to find all of the logical math zones on the image with as few under-segmentations, over-segmentations, missed regions, and false positives as possible. It is also decided within this subsystem whether an expression should be labeled as displayed or embedded. The resulting math regions can then be fed directly into a recognition module, assuming that the segmentation system has made the proper decisions.

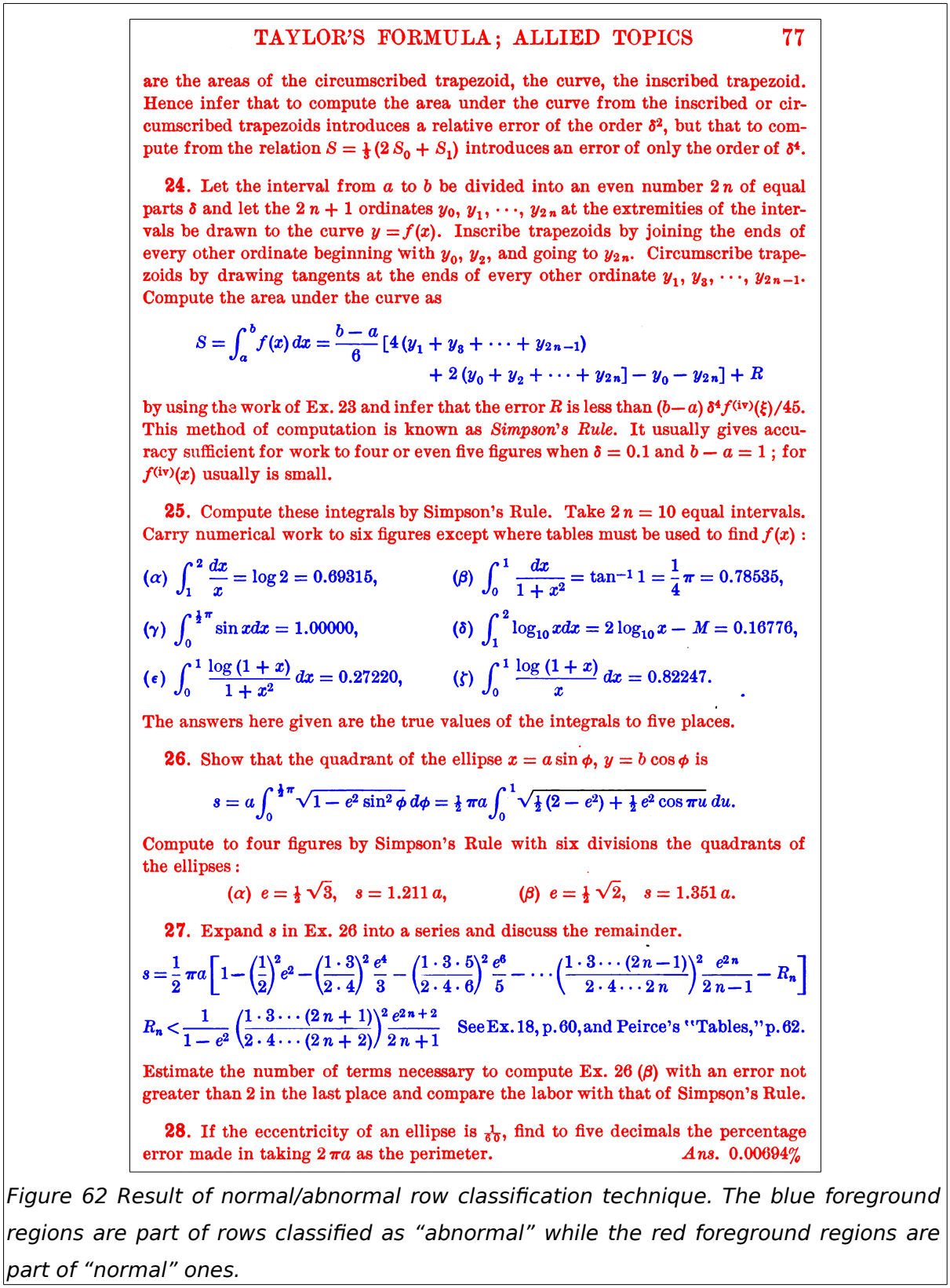
This step does not include a classifier and relies upon various heuristics in order to make the appropriate decisions. Although no supervised learning is utilized in this work, it would not be difficult to extend this module to handle supervised learning. Training would involve assigning each blob the directions to which it should be merged. For instance the fraction bar in the following expression $\frac{a+b}{c}$ should be merged with its nearest neighbor above and below while $+$ should be merged with its nearest neighbor left, right, and below. Four binary classifiers could be trained, each one representing one of the following merge procedures: merge with nearest neighbor left, right, above, and below. If any of the the four classifier outputs are 1 then the

corresponding merge operation would be carried out. If a seed region “covers” multiple regions either horizontally or vertically as previously explained in the spatial feature extraction section of this work entitled *Number of Horizontally or Vertically Aligned Characters*, then all of the adjacent neighbors covered by the seed region in the appropriate direction would be merged rather than just the nearest neighbor. If a blob is already merged in a given direction then no action would be required during the prediction stage. Although supervised learning of these four binary classifiers and choosing the best features for them would be a productive avenue for experimentation, it is kept as an idea for future work and is here replaced by a simpler unsupervised heuristic approach to be described in this section.

During the segmentation process, a seed region can be merged in any combination of the following directions: left, right, up, or down. The decision as to which directions are appropriate for merging are based on various heuristics which are enumerated in this section.

Classify as Displayed or Embedded. Each math blob is first classified as either displayed or embedded. The classification technique employed here is simple: if a blob belongs to a row that is deemed as “normal” then it is classified as embedded otherwise it is classified as displayed. Normal rows have a good overall recognition confidence and have a predictable vertical spacing. The width in normal text rows is also predictable up to the last row of a paragraph which is expected to be less than or equal to the width of its preceding rows. Specifically, the number of valid words is counted on each individual row and then the mean and standard deviation of the valid word count per row is calculated (only rows containing at least one valid word are used in this measurement). Two passes are carried out in order to determine whether a row should be considered “normal” or “abnormal” (abnormal rows end up being considered as candidates for containing displayed expressions in this work). On the first pass, rows are classified as “abnormal” if their valid word count negatively deviates from the average by more than twice the standard deviation. On the second pass, the average and standard deviation of the vertical space between rows are then calculated (the top row is not included in this calculation since it is often a header). If the vertical space above a row previously classified as “abnormal” is below the standard deviation, then the row is considered to be a paragraph ending and assigned back to “normal”. An example of results for this procedure is illustrated by Figure 62. An improved classification technique for normal and abnormal rows would enhance

both detection and segmentation accuracy significantly but is kept as a goal for future work.



TAYLOR'S FORMULA; ALLIED TOPICS 77

are the areas of the circumscribed trapezoid, the curve, the inscribed trapezoid. Hence infer that to compute the area under the curve from the inscribed or circumscribed trapezoids introduces a relative error of the order δ^2, but that to compute from the relation $S = \frac{1}{3}(2S_0 + S_1)$ introduces an error of only the order of δ^4.

24. Let the interval from a to b be divided into an even number $2n$ of equal parts δ and let the $2n+1$ ordinates $y_0, y_1, \cdots, y_{2n}$ at the extremities of the intervals be drawn to the curve $y = f(x)$. Inscribe trapezoids by joining the ends of every other ordinate beginning with y_0, y_2, and going to y_{2n}. Circumscribe trapezoids by drawing tangents at the ends of every other ordinate $y_1, y_3, \cdots, y_{2n-1}$. Compute the area under the curve as

$$S = \int_a^b f(x)\,dx = \frac{b-a}{6}[4(y_1 + y_3 + \cdots + y_{2n-1}) + 2(y_0 + y_2 + \cdots + y_{2n}] - y_0 - y_{2n}] + R$$

by using the work of Ex. 23 and infer that the error R is less than $(b-a)\,\delta^4 f^{(iv)}(\xi)/45$. This method of computation is known as *Simpson's Rule*. It usually gives accuracy sufficient for work to four or even five figures when $\delta = 0.1$ and $b - a = 1$; for $f^{(iv)}(x)$ usually is small.

25. Compute these integrals by Simpson's Rule. Take $2n = 10$ equal intervals. Carry numerical work to six figures except where tables must be used to find $f(x)$:

(α) $\int_1^2 \frac{dx}{x} = \log 2 = 0.69315$, (β) $\int_0^1 \frac{dx}{1+x^2} = \tan^{-1} 1 = \frac{1}{4}\pi = 0.78535$,

(γ) $\int_0^{\frac{1}{2}\pi} \sin x\,dx = 1.00000$, (δ) $\int_1^2 \log_{10} x\,dx = 2\log_{10} x - M = 0.16776$,

(ϵ) $\int_0^1 \frac{\log(1+x)}{1+x^2}\,dx = 0.27220$, (ζ) $\int_0^1 \frac{\log(1+x)}{x}\,dx = 0.82247$.

The answers here given are the true values of the integrals to five places.

26. Show that the quadrant of the ellipse $x = a\sin\phi$, $y = b\cos\phi$ is

$$s = a\int_0^{\frac{1}{2}\pi} \sqrt{1 - e^2\sin^2\phi}\,d\phi = \tfrac{1}{2}\pi a \int_0^1 \sqrt{\tfrac{1}{2}(2-e^2) + \tfrac{1}{2}e^2\cos\pi u}\,du.$$

Compute to four figures by Simpson's Rule with six divisions the quadrants of the ellipses:

(α) $e = \frac{1}{2}\sqrt{3}$, $s = 1.211\,a$, (β) $e = \frac{1}{2}\sqrt{2}$, $s = 1.351\,a$.

27. Expand s in Ex. 26 into a series and discuss the remainder.

$$s = \frac{1}{2}\pi a\left[1 - \left(\frac{1}{2}\right)^2 e^2 - \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 \frac{e^4}{3} - \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)^2 \frac{e^6}{5} - \cdots - \left(\frac{1\cdot 3\cdots(2n-1)}{2\cdot 4\cdots 2n}\right)^2 \frac{e^{2n}}{2n-1} - R_n\right]$$

$$R_n < \frac{1}{1-e^2}\left(\frac{1\cdot 3\cdots(2n+1)}{2\cdot 4\cdots(2n+2)}\right)^2 \frac{e^{2n+2}}{2n+1}$$ See Ex. 18, p. 60, and Peirce's "Tables," p. 62.

Estimate the number of terms necessary to compute Ex. 26 (β) with an error not greater than 2 in the last place and compare the labor with that of Simpson's Rule.

28. If the eccentricity of an ellipse is $\frac{1}{60}$, find to five decimals the percentage error made in taking $2\pi a$ as the perimeter. *Ans.* 0.00694%

Figure 62 Result of normal/abnormal row classification technique. The blue foreground regions are part of rows classified as "abnormal" while the red foreground regions are part of "normal" ones.

Method

Segmentation Algorithm. The segmentation algorithm works by iterating through all of the BlobInfo elements in the grid, skipping them unless they are mathematical and haven't already been processed. When an unprocessed mathematical region is found, it is first flagged as processed in order to ensure it will only be processed once. A Merge Decision is made for each of the four directions as specified in the Merge Decision subsection. The Merge Operation (as specified in the Merge Operation subsection) is then carried out on whichever of the four directions were decided for merging by the Merge Decision. The aforementioned operation is recursively repeated for every merged blob until all merged blobs are processed and no more merges are deemed necessary by the Merge Decision. The algorithm then continues iterating the BlobInfo elements, repeating the aforementioned procedure for each BlobInfo element. The final math zones are then set to the bounding box represented by the top left and bottom right points of each resulting segmentation.

Data structures Utilized. Each BlobInfo element contains a Merge object which specifies what merge operations are to be carried out. The Merge object contains an initially NULL pointer for each direction (left, right, up, and down). If a merge operation in the corresponding direction is not to be carried out then the pointer will remain NULL, otherwise it will point to the merged BlobInfo element. The Merge object also contains a flag which specifies whether or not the corresponding BlobInfo element has already been processed so that each will only be processed once. Each Merge object also contains a pointer to the bounding box which corresponds to the entire segmentation to which it's BlobInfo element belongs. Whenever a merge operation is carried out, this segment is modified if the new object merged is outside of the bounds of it's segment's bounding box.

Merge Decision. For each unprocessed mathematical BlobInfo element, a merge decision must be made for the four directions. The factors which underlie the merge decision are dependent upon whether the merge direction is up/down or right/left.

Vertical Merges. Fraction bars should typically be merged both up and down with the elements that they “cover” as explained earlier in the Feature Extraction subsection. Also of interest here are limits and intervals wherein characters below and/or above need to be merged as illustrated by Figure 63.

$$J_n(x) = \frac{x^n}{2^n} \sum_0^\infty \frac{(-1)^i x^{2i}}{2^{2i} i!\,(n+i)!} = \frac{x^n}{2^n n!} I_n(\tfrac{1}{4}x^2)$$

$$\lim_{h \to 0} \frac{R(h)}{h} = \lim_{h \to 0} \frac{f(c+h) - T(c+h)}{h}$$

Figure 63 Both of the above expressions were taken from the training set. The top comes from [129] and the bottom from [131]. The summations, limits, and fractions illustrate the need for vertical merging of fraction and non-fraction elements alike.

Stacked elements at a position which satisfy the rules in the Stacked element feature section are merged up/down. The features found during feature extraction are useful here. If an element has a stack count greater than one, the stacked neighbors are all immediately merged.

Once any tall element or stack of elements is merged, the entire merged region in question immediately grows in size. Each element being merged is immediately assigned to the box which represents the entire segment which it has joined and required to grow the segment if it is outside of its bounds. The size of the segment to which a blob belongs plays a role in merging new blobs when dealing with displayed expression regions. If the current blob has been classified as part of a displayed expression, belongs to a segment which vertically “covers” multiple objects adjacent to it (the entire segmentation), and none of the adjacent blobs are separators (periods, commas, phrases like “such that” “therefore” “thus” etc), then all of the covered blobs are immediately merged. In Figure 64, for instance, the entire merged region would start with just the |A|, then its right would be modified as the equivalent operator is merged (as described in the following horizontal merging subsection), then top and bottom bounds would be modified as the large vertical bar is merged, etc.

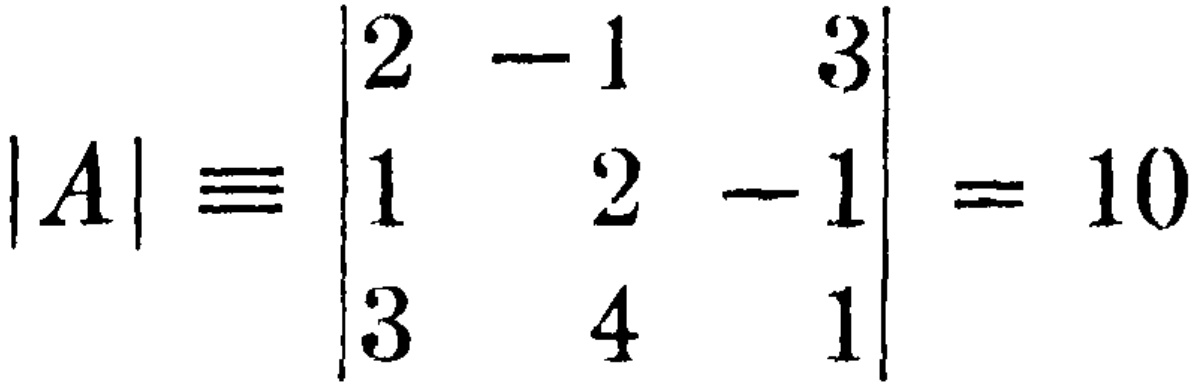


Figure 64 An expression taken from [127] which includes a determinate. Determinates and matrices include multiple vertically stacked elements which must be merged into a single math segment.

Horizontal Merges. As explained in the previous section, the vertical “coverage” feature is made useful in order to horizontally merge multiple vertically overlapping elements as shown in Figure 64. For displayed expressions, the rule employed here is to continue merging right until either a significant space gap is found (empirically decided as more than twice the maximum gap observed in the current segmentation) or a separator is detected. For embedded expressions the merging is more conservative. If the seed is a known binary operator then a merge takes place both to the left and right. Merging to the right is obvious if the nearest adjacent neighbor is math, otherwise it becomes tricky. This work will rely largely on the detector's accuracy for embedded expressions. If the nearest neighbor to the right is part of an invalid word but was not classified as math, then merging will only take place if either the current blob is a known operator or if the adjacent blob is close enough to the current one (an empirically chosen distance of less than half the width of the current blob has been chosen for this task).

Merge Operation. Whenever a merge operation is carried out the pointer in the corresponding direction will be set to the merged element, and then the merged element's corresponding pointer in the opposite direction will be set as well. Thus if a right merge is carried out then the current blob's right pointer is set to the blob on the right while the merged blob's left pointer is set to the current blob. This operation ensures that the blobs are logically linked to each other in each direction. As previously mentioned, each blob has access to the bounding box which gives the

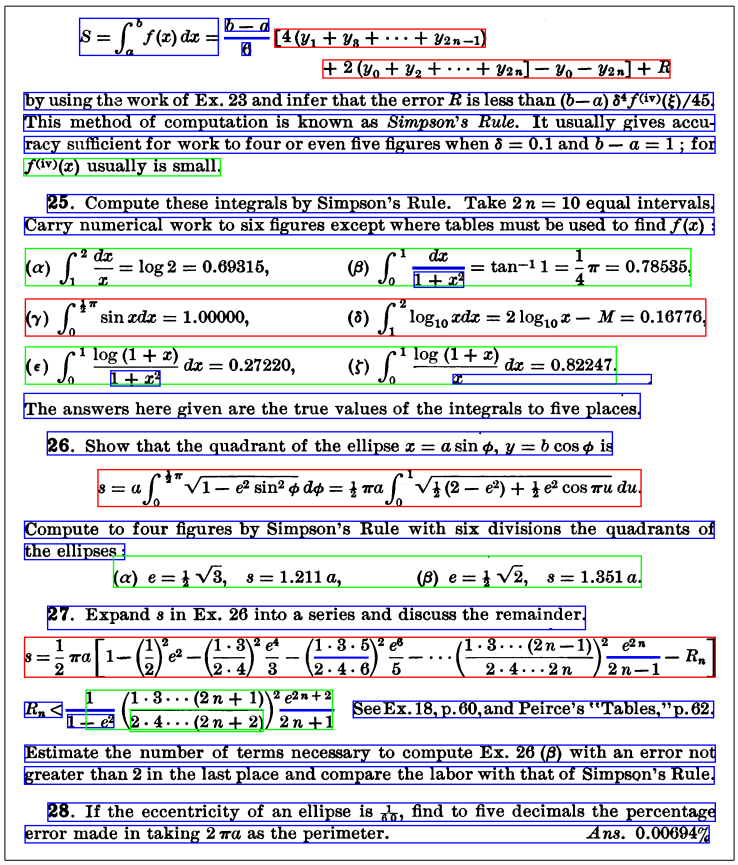
entire region currently under segmentation. If a merged blob is outside of this boundary, then the boundary is modified to fit the blob during the merging process.

Final Output Preparation. Once all the math segmentations have been prepared, the column partitions found by Tesseract are modified only in places wherein mathematical partitions have been found (i.e. if a paragraph contains embedded mathematical expressions, the regions corresponding to these expressions should be understood as part of the paragraph however segmented from the normal text).

3.4.3 Evaluation Module

A significant problem observed in previous work for math detection and segmentation has been that of objectively comparing the performance of one technique to another. Difficulties in objective comparison of different works are a result of each author using their own private datasets and evaluation techniques. In this work the evaluation code as well as the dataset are made publicly available [126] in order to encourage objective performance comparison of current, new, and existing MEDS techniques. This section describes the design and functionality of this work's pixel accurate evaluation module.

The evaluation module used in this work is designed to evaluate both the math regions found by Tesseract's default MEDS module [141] as well as the math regions found by any newly implemented MEDS module which overrides Tesseract's default one. The output of Tesseract's default equation detector is automatically evaluated by first writing the results to a “box file” which contains the left, bottom, top, and right coordinates for each detected region as illustrated in Figure 65 and also coloring the corresponding pixels in the image based on the result type of each box file entry as illustrated in Figure 66.



$$S = \int_a^b f(x)\,dx = \frac{b-a}{6}\left[4\,(y_1 + y_3 + \cdots + y_{2n-1}) + 2\,(y_0 + y_2 + \cdots + y_{2n}] - y_0 - y_{2n}] + R\right.$$

by using the work of Ex. 23 and infer that the error R is less than $(b-a)\,\delta^4 f^{(iv)}(\xi)/45$. This method of computation is known as *Simpson's Rule.* It usually gives accuracy sufficient for work to four or even five figures when $\delta = 0.1$ and $b - a = 1$; for $f^{(iv)}(x)$ usually is small.

25. Compute these integrals by Simpson's Rule. Take $2n = 10$ equal intervals. Carry numerical work to six figures except where tables must be used to find $f(x)$:

$(\alpha)\ \int_1^2 \frac{dx}{x} = \log 2 = 0.69315,$ $(\beta)\ \int_0^1 \frac{dx}{1+x^2} = \tan^{-1} 1 = \frac{1}{4}\pi = 0.78535,$

$(\gamma)\ \int_0^{\frac{1}{2}\pi} \sin x dx = 1.00000,$ $(\delta)\ \int_1^2 \log_{10} x dx = 2\log_{10} x - M = 0.16776,$

$(\epsilon)\ \int_0^1 \frac{\log(1+x)}{1+x^2}\,dx = 0.27220,$ $(\zeta)\ \int_0^1 \frac{\log(1+x)}{x}\,dx = 0.82247.$

The answers here given are the true values of the integrals to five places.

26. Show that the quadrant of the ellipse $x = a\sin\phi$, $y = b\cos\phi$ is

$$s = a\int_0^{\frac{1}{2}\pi} \sqrt{1 - e^2\sin^2\phi}\,d\phi = \tfrac{1}{2}\pi a \int_0^1 \sqrt{\tfrac{1}{2}(2-e^2) + \tfrac{1}{2}e^2\cos\pi u}\,du.$$

Compute to four figures by Simpson's Rule with six divisions the quadrants of the ellipses:

$(\alpha)\ e = \tfrac{1}{2}\sqrt{3},\quad s = 1.211\,a,$ $(\beta)\ e = \tfrac{1}{2}\sqrt{2},\quad s = 1.351\,a.$

27. Expand s in Ex. 26 into a series and discuss the remainder.

$$s = \frac{1}{2}\pi a\left[1 - \left(\frac{1}{2}\right)^2 e^2 - \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 \frac{e^4}{3} - \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)^2 \frac{e^6}{5} - \cdots \left(\frac{1\cdot 3\cdots(2n-1)}{2\cdot 4\cdots 2n}\right)^2 \frac{e^{2n}}{2n-1} - R_n\right]$$

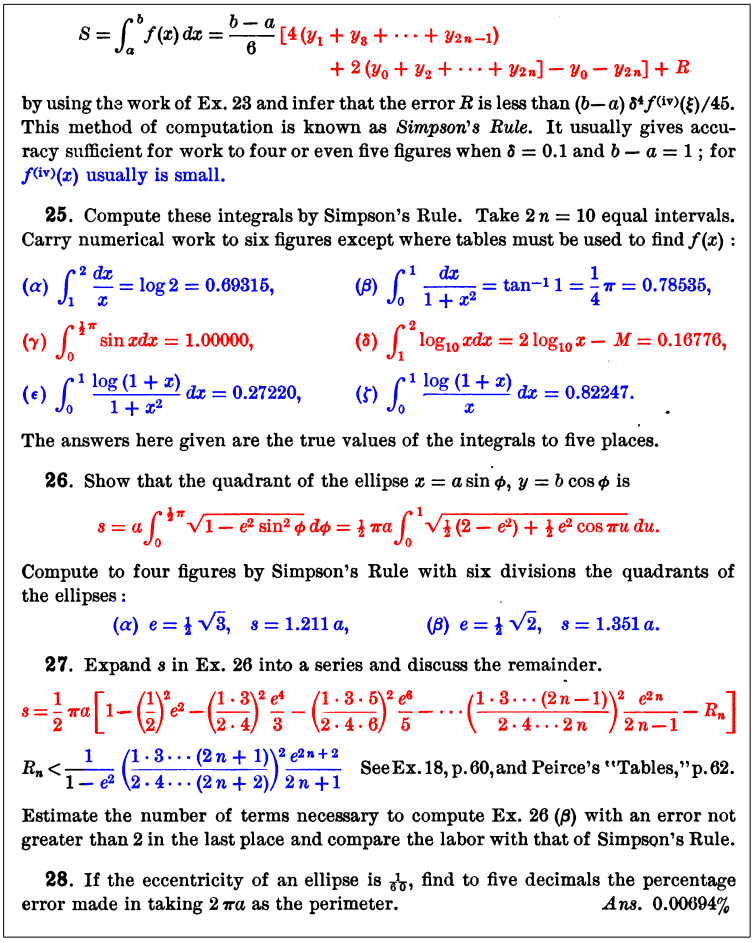
$R_n < \frac{1}{1-e^2}\left(\frac{1\cdot 3\cdots(2n+1)}{2\cdot 4\cdots(2n+2)}\right)^2 \frac{e^{2n+2}}{2n+1}$ See Ex. 18, p. 60, and Peirce's "Tables," p. 62.

Estimate the number of terms necessary to compute Ex. 26 (β) with an error not greater than 2 in the last place and compare the labor with that of Simpson's Rule.

28. If the eccentricity of an ellipse is $\frac{1}{10}$, find to five decimals the percentage error made in taking $2\pi a$ as the perimeter. *Ans.* 0.00694%

Figure 65 Debug output from Tesseract's default MEDS module. Red regions were classified as displayed expressions, green regions classified as embedded, and blue regions are non-math.

Method



$$S = \int_a^b f(x)\,dx = \frac{b-a}{6}[4(y_1 + y_3 + \cdots + y_{2n-1}) + 2(y_0 + y_2 + \cdots + y_{2n}] - y_0 - y_{2n}] + R$$

by using the work of Ex. 23 and infer that the error R is less than $(b-a)\,\delta^4 f^{(iv)}(\xi)/45$. This method of computation is known as *Simpson's Rule*. It usually gives accuracy sufficient for work to four or even five figures when $\delta = 0.1$ and $b - a = 1$; for $f^{(iv)}(x)$ usually is small.

25. Compute these integrals by Simpson's Rule. Take $2n = 10$ equal intervals. Carry numerical work to six figures except where tables must be used to find $f(x)$:

$(\alpha)\ \int_1^2 \frac{dx}{x} = \log 2 = 0.69315,$ $(\beta)\ \int_0^1 \frac{dx}{1+x^2} = \tan^{-1} 1 = \frac{1}{4}\pi = 0.78535,$

$(\gamma)\ \int_0^{\frac{1}{2}\pi} \sin x\,dx = 1.00000,$ $(\delta)\ \int_1^2 \log_{10} x\,dx = 2\log_{10} x - M = 0.16776,$

$(\epsilon)\ \int_0^1 \frac{\log(1+x)}{1+x^2}\,dx = 0.27220,$ $(\zeta)\ \int_0^1 \frac{\log(1+x)}{x}\,dx = 0.82247.$

The answers here given are the true values of the integrals to five places.

26. Show that the quadrant of the ellipse $x = a\sin\phi$, $y = b\cos\phi$ is

$$s = a\int_0^{\frac{1}{2}\pi} \sqrt{1 - e^2\sin^2\phi}\,d\phi = \tfrac{1}{2}\pi a \int_0^1 \sqrt{\tfrac{1}{2}(2-e^2) + \tfrac{1}{2}e^2\cos\pi u}\,du.$$

Compute to four figures by Simpson's Rule with six divisions the quadrants of the ellipses:

$(\alpha)\ e = \frac{1}{2}\sqrt{3},\quad s = 1.211\,a,$ $(\beta)\ e = \frac{1}{2}\sqrt{2},\quad s = 1.351\,a.$

27. Expand s in Ex. 26 into a series and discuss the remainder.

$$s = \frac{1}{2}\pi a\left[1 - \left(\frac{1}{2}\right)^2 e^2 - \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 \frac{e^4}{3} - \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)^2 \frac{e^6}{5} - \cdots \left(\frac{1\cdot 3\cdots(2n-1)}{2\cdot 4\cdots 2n}\right)^2 \frac{e^{2n}}{2n-1} - R_n\right]$$

$$R_n < \frac{1}{1-e^2}\left(\frac{1\cdot 3\cdots(2n+1)}{2\cdot 4\cdots(2n+2)}\right)^2 \frac{e^{2n+2}}{2n+1}$$

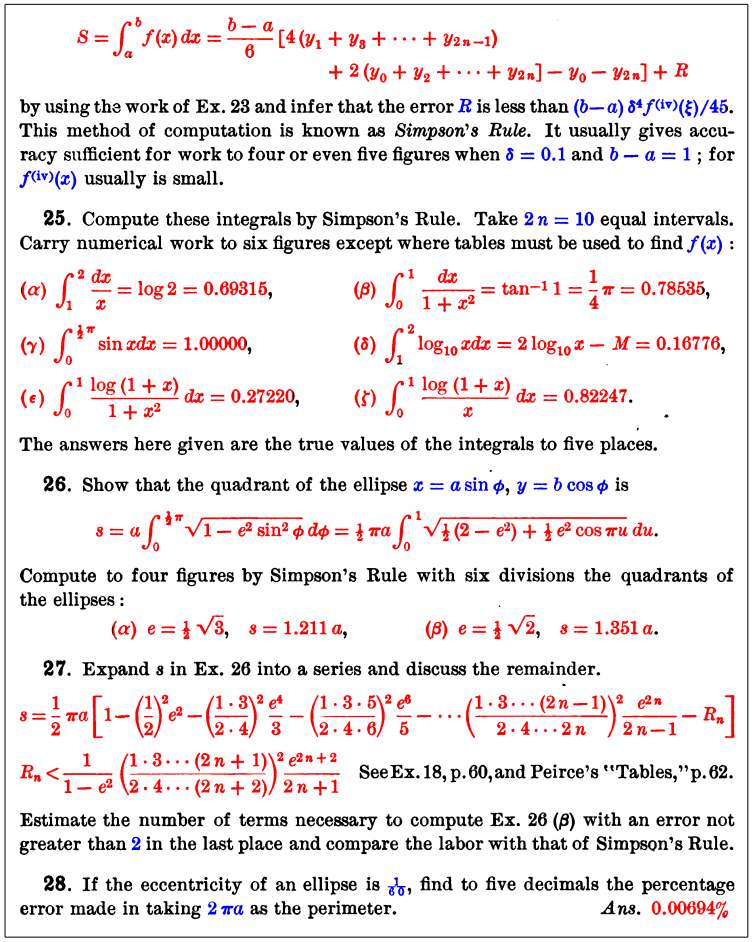
See Ex. 18, p. 60, and Peirce's "Tables," p. 62.

Estimate the number of terms necessary to compute Ex. 26 (β) with an error not greater than 2 in the last place and compare the labor with that of Simpson's Rule.

28. If the eccentricity of an ellipse is $\frac{1}{60}$, find to five decimals the percentage error made in taking $2\pi a$ as the perimeter. *Ans.* 0.00694%

Figure 66 The same image as shown in Figure 53 except with the foreground regions colored for the math expression bounding boxes found by Tesseract's default MEDS module. The blue regions here were labeled as part of an embedded expression region while the red regions were labeled as part of a displayed expression region.

The foreground regions are then also colored for the rectangles in the corresponding image's manually generated groundtruth data as illustrated in Figure 67.



$$S = \int_a^b f(x)\,dx = \frac{b-a}{6}[4(y_1 + y_3 + \cdots + y_{2n-1}) + 2(y_0 + y_2 + \cdots + y_{2n}] - y_0 - y_{2n}] + R$$

by using the work of Ex. 23 and infer that the error R is less than $(b-a)\,\delta^4 f^{(iv)}(\xi)/45$. This method of computation is known as *Simpson's Rule*. It usually gives accuracy sufficient for work to four or even five figures when $\delta = 0.1$ and $b - a = 1$; for $f^{(iv)}(x)$ usually is small.

25. Compute these integrals by Simpson's Rule. Take $2n = 10$ equal intervals. Carry numerical work to six figures except where tables must be used to find $f(x)$:

(α) $\int_1^2 \frac{dx}{x} = \log 2 = 0.69315$, ($\beta$) $\int_0^1 \frac{dx}{1+x^2} = \tan^{-1} 1 = \frac{1}{4}\pi = 0.78535$,

(γ) $\int_0^{\frac{1}{2}\pi} \sin x\,dx = 1.00000$, ($\delta$) $\int_1^2 \log_{10} x\,dx = 2\log_{10} x - M = 0.16776$,

(ϵ) $\int_0^1 \frac{\log(1+x)}{1+x^2}\,dx = 0.27220$, ($\zeta$) $\int_0^1 \frac{\log(1+x)}{x}\,dx = 0.82247$.

The answers here given are the true values of the integrals to five places.

26. Show that the quadrant of the ellipse $x = a\sin\phi$, $y = b\cos\phi$ is

$$s = a\int_0^{\frac{1}{2}\pi} \sqrt{1 - e^2\sin^2\phi}\,d\phi = \tfrac{1}{2}\pi a \int_0^1 \sqrt{\tfrac{1}{2}(2-e^2) + \tfrac{1}{2}e^2\cos\pi u}\,du.$$

Compute to four figures by Simpson's Rule with six divisions the quadrants of the ellipses:

(α) $e = \frac{1}{2}\sqrt{3}$, $s = 1.211\,a$, (β) $e = \frac{1}{2}\sqrt{2}$, $s = 1.351\,a$.

27. Expand s in Ex. 26 into a series and discuss the remainder.

$$s = \frac{1}{2}\pi a\left[1 - \left(\frac{1}{2}\right)^2 e^2 - \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 \frac{e^4}{3} - \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)^2 \frac{e^6}{5} - \cdots \left(\frac{1\cdot 3\cdots(2n-1)}{2\cdot 4\cdots 2n}\right)^2 \frac{e^{2n}}{2n-1} - R_n\right]$$

$$R_n < \frac{1}{1-e^2}\left(\frac{1\cdot 3\cdots(2n+1)}{2\cdot 4\cdots(2n+2)}\right)^2 \frac{e^{2n+2}}{2n+1}$$

See Ex. 18, p. 60, and Peirce's "Tables," p. 62.

Estimate the number of terms necessary to compute Ex. 26 (β) with an error not greater than 2 in the last place and compare the labor with that of Simpson's Rule.

28. If the eccentricity of an ellipse is $\frac{1}{10}$, find to five decimals the percentage error made in taking $2\pi a$ as the perimeter. *Ans.* 0.00694%

Figure 67 The same image as shown in Figures 53 and 54 but with the foreground regions of the bounding boxes from the manually generated groundtruth automatically colored using the same convention as in Figure 54.

Figures 66 and 67 illustrate the hypothesis and groundtruth (correct) results respectively which are compared in this module in order to evaluate the correctness of the hypothesis. The term hypothesis is used by this section to refer to any MEDS results for a type of expression in a single image whereas the term groundtruth refers to the expected/correct segmentation results for a type of expression in a single image. Figures 66 and 67 are thus essentially representing two groundtruth/hypothesis pairs since both embedded and displayed expressions are shown. A single groundtruth/hypothesis pair is used to evaluate just one expression type (either displayed, embedded or optionally displayed expression labels). Both the groundtruth and hypothesis are represented by an image/file pair. The image is as shown in Figures 66 and 67 (except with only one color being observed) and allows for pixel accurate evaluation while the file gives the bounding boxes of all segmented regions. While the image allows for pixel-accurate comparisons, the file allows for the detection of over-

segmentations and under-segmentations. Over-segmentations occur when a single region in the groundtruth is incorrectly divided into multiple regions by the hypothesis, while under-segmentations occur when multiple regions in the groundtruth are incorrectly merged into one region by the hypothesis.

Pixel-by-pixel comparison of the foreground pixels of the groundtruth and hypothesis images allow for the pixel accurate calculation of True Positive Rate (TPR), Positive Predictive Value (PPV), Accuracy (ACC), False Positive Rate (FPR), False Discovery Rate (FDR), True Negative Rate (TNR), and Negative Predictive Value (NPV). All 7 of the aforementioned pixel accurate metrics are defined in Table 3 with the following notation:

Positives (P). *P* pixels are the foreground pixels in the groundtruth that are of the color being observed (i.e. red if evaluating displayed expressions and blue if evaluating embedded expressions).

Negatives (N). *N* pixels are the total foreground pixels in the groundtruth that are not of the color being observed (either black or the color of a different expression type which is not currently being evaluated).

True Positive Pixels (TP). *TP* pixels that are colored in the groundtruth and are also colored in the hypothesis. *TP* are thus pixels that are correctly labeled in the hypothesis.

False Negative Pixels (FN). *FN* pixels are colored in the groundtruth but not in the hypothesis. The *FN* and *TP* pixels should add up to the total positive pixels in the groundtruth (*P*).

False Positive Pixels (FP). *FP* pixels are not colored in the groundtruth but are colored in the hypothesis.

True Negative Pixels (TN)*: TN* pixels are not colored in the hypothesis and are also not colored in the groundtruth. The sum of the *TN* and *FP* pixels should be equal to the total negative pixels in the groundtruth (*N*).

Table 3 The seven pixel-accurate metrics which are found to measure the validity of a hypothesis in comparison to a groundtruth along with measurements of oversegmentations and undersegmentations.

Metric	Pseudonyms	Definition
TPR	Recall/Sensitivity/Hit Rate	TP/(TP+FN) = TP/P
PPV	Precision	TP/(TP+FP)
ACC	Accuracy	(TP+TN)/(TP+FN+TN+FP) = (TP+TN)/(P+N)
FPR	Fallout	FP/(FP+TN) = FP/N
FDR	False Discovery Rate	FP/(FP+TP)
TNR	Specificity	TN/(FP+TN) = TN/N
NPV	Negative Predictive Value	TN/(TN+FN)

In order to calculate the metrics shown in Table 3 and to count the oversegmentations and undersegmentations, a bipartite graph data structure is utilized [87]. The bipartite graph data structure consists of two graphs, one representing the hypothesis and the other representing the groundtruth. For each graph, the vertices are first added, each one representing a segmented region. Edges between the groundtruth and hypothesis graphs are then made to represent the intersection of pixels between them. If a vertex is unmatched by the other image (i.e. a segmented region in one graph has no overlapping region in the other) then it will have no edges. Vertices may also have multiple edges if more than one region in the other graph intersects the one in the current graph.

Once the bipartite graph structure is initialized with all of its vertices and edges, it then becomes rather easy to measure over-segmentations, under-segmentations, entirely missed regions, and entirely false positive regions. Over-segmentations occur when one vertex in the groundtruth maps to many in the hypothesis and under-segmentations occur when a single hypothesis vertex maps to multiple groundtruth vertices. A region is entirely missed when a groundtruth vertex has no edges and a region is entirely false positive when a hypothesis vertex has no edges. For each hypothesis region, the number of overlapping groundtruth pixels gives the true positives. These true positives aggregated over the entire image and then divided by the total positives in the groundtruth then gives the TPR. Similar calculations are then carried out for the remaining six metrics and aggregated for the entire image to yield the final metrics. The metrics are then written to a file in the following format:

Method

// region-wide statistics:
[# correctly segemented regions] / [total # regions]
[# regions completely missed (fn)]
[# regions completely wrongly detected (fp)]

// stats on oversegmentations and undersegmentations:
[# oversegmented regions]
[# total oversegmentations for all regions]
[# avg oversegmentations per oversegmented groundtruth region]
[# undersegmented regions]
[# total undersegmentations for all regions]
[# avg undersegmentations for undersegmented hypothesis region]

// pixel counts:
[# total foreground pix (tp+fp+tn+fn)]
[# total positively detected pix (tp+fp)]
[# total negatively detected pix (tn+fn)]
[# total true positive pix (tp)]
[# total false negative pix (fn)]
[# total true negative pix (tn)]
[# total false positive pix (fp)]

// metrics based on pixel counts (all between 0 and 1)
[TPR/Recall/Sensitivity/Hit_Rate = tp/(tp+fn)]
[Precision/Positive_Predictive_Value = tp/(tp+fp)]
[Accuracy = (tp+tn)/(tp+fn+tn+fp)]
[FPR/Fallout = fp/(fp+tn)]
[False_Discovery_Rate = fp/(fp+tp)]
[TNR/Specificity = tn/(fp+tn)]
[Negative_Predictive_Value = tn/(tn+fn)]

In the above format, the statistics on over-segmentations and under-segmentation requires some explanation. The number of over-segmented regions gives the number of vertices in the groundtruth which have more than one edge in the hypothesis. The total over-segmentations for all regions gives the total number of edges aggregated over each over-segmented groundtruth region. The average over-segmentations per over-segmented groundtruth region gives the average number of edges that an over-segmented groundtruth vertex has over the entire groundtruth graph (i.e., the average *severity* of an oversegmentation). This is effectively a measure of how badly split a typical over-segmented region is. The under-segmentation statistics are very similar to the over-segmentation ones. Under-segmentation, however, is found when a hypothesis vertex has more than one edge pointing to the groundtruth. To illustrate the usefulness of the evaluation module, evaluation of the default Tesseract MEDS module was carried out on the input image shown in Figures 65-67 yielded the metrics shown in Tables 4-7 for displayed expressions:

Table 4 Region-wide statistics for Tesseract default equation detector.

Region-wide Statistics	**Measurement**
# correctly segemented regions] / [total # regions]	4/16
# regions completely missed (fn)	10
# regions completely wrongly detected (fp)	0

Table 5 Over/under-segmentation statistics for Tesseract default equation detector.

Over/under-segmentation Statistics	**Measurement**
# oversegmented regions	0
# total oversegmentations for all regions	0
# avg oversegmentations per oversegmented groundtruth region	0
# undersegmented regions	1
# total undersegmentations for all regions	2
# avg undersegmentations for undersegmented hypothesis region	2

Table 6 Pixel count statistics for Tesseract default equation detector.

Pixel Count Statistics	**Measurement**
# total foreground pix (tp+fp+tn+fn)	1,157,429
# total positively detected pix (tp+fp)	140,878
# total negatively detected pix (tn+fn)	1,016,551
# total true positive pix (tp)	140,096
# total false negative pix (fn)	130,031
# total true negative pix (tn)	886,520
# total false positive pix (fp)	782

Table 7 Pixel accurate metrics for Tesseract's default equation detector.

Pixel-accurate Evaluation Statistics	**Measurement**
TPR/Recall/Sensitivity/Hit_Rate = tp/(tp+fn)	0.518630
Precision/Positive_Predictive_Value = tp/(tp+fp)	0.994449
Accuracy = (tp+tn)/(tp+fn+tn+fp)	0.886980
FPR/Fallout = fp/(fp+tn)	0.000881
False_Discovery_Rate = fp/(fp+tp)	0.005551
TNR/Specificity = tn/(fp+tn)	0.999119
Negative_Predictive_Value = tn/(tn+fn)	0.872086

4 Experimental Results

The math expressions of 75 images were manually extracted and placed into “box files” which contain the image index, type of expression, and bounding box coordinates as discussed in **Section 3.4.1**. Of these images and their corresponding box files, 15 of them were used to train the math expression detector while the other 60 were used to then evaluate it. The following section first discusses the results of the detector parameter selection and cross validation training technique as described in **Section 3.4.2**'s *Fine Tuning of SVM Parameters*. The results of Parameter Selection and cross validation on the 15 training images are then followed by a presentation of and discussion for all final evaluation results which were carried out on the remaining 60 images used in this work.

4.1 Detector Parameter Selection and Training

The D-Lib Machine Learning Library [134] was utilized in this work to train four separate SVM classifiers, each of which uses the RBF kernel. While all four of the classifiers are trained using the same procedure, they are done so on different combinations of features extracted from each sample of the image, where a sample is an individual element in the image's custom grid data structure as illustrated in *Figure 46* of **Section 3.4.2**. The classifiers are named based upon the SVM kernel used for training along with the name of the feature extractor combination employed. Each feature extractor was simply named F_Ext (for feature extractor) followed by an identifier. The feature extractor which extracts all of the features described in **Section 3.4.2** is named F_Ext1. The full list of features used by F_Ext1 is shown in Table 8. The remaining three feature extractors use a subset of the F_Ext1 features as shown in Table 9.

Table 8 All of the features extracted in this work. The feature extractor named F_Ext1 uses all 22 features while the other three extractors tested use a subset of these.

Shorthand Name	**Feature Description**
rhabc	Rightward horizontally adjacent blobs covered
uvabc	Upward vertically adjacent blobs covered
dvabc	Downard vertically adjacent blobs covered
cn	Number of completely nested characters
has_sup	Has a superscript
has_sub	Has a subscript
is_sup	Is a superscript
is_sub	Is a subscript
h	Blob height
whr	Blob width/height ratio
vdarb	Vertical distance above row baseline
cosbabp	Count of stacked blobs at blob position
imw	Is blob in math word
is_italic	Italicized text
ocr_conf	OCR confidence rating
unigram	Unigram Feature
bigram	Bigram Feature
trigram	Trigram Feature
in_valid_row	Blob belongs to row with normal text (at least one valid word)
in_valid_word	Blob belongs to normal text
bad_page	Page doesn't have normal text
stop_word	Blob belongs to stop word

Table 9 The four classifiers which were trained and tested in this work along with the features on which they were trained.

Classifier Name	**Feature Combination**
RBFSVM_F_Ext1	All Features
RBFSVM_F_Ext2	No “in_valid_word” feature
RBFSVM_F_Ext3	No “in_valid_row” feature or n-gram features
RBFSVM_F_Ext4	No italics feature

As discussed in **Section 3.4.2**'s *Fine Tuning of SVM Parameters,* a coarse to fine-grained parameter selection technique was carried out in order to determine what SVM parameters gave the best 10-fold cross-validation results overall. This technique was carried out on all four of the classifiers shown in Table 9 and yielded the results shown in Table 10. In addition to the four RBF kernel SVM's tested, a linear SVM was tested using the F_Ext1 features but could not achieve a true positive rate above 75% during any cross validation and was thus the discarded in favor of the RBF kernel.

Table 10 Each classifiers' optimal parameter combination, TPR, and TNR found through coarse to fine grained parameter selection using repeated 10-fold cross-validation.

Classifier Name	Optimal (C,γ)	TPR	TNR
RBFSVM_F_Ext1	(123.88, 0.83020)	88.79%	97.87%
RBFSVM_F_Ext2	(133.34, 0.81802)	88.75%	97.86%
RBFSVM_F_Ext3	(7.0551, 7.65815)	87.54%	97.44%
RBFSVM_F_Ext4	(16.513, 2.66315)	89.79%	97.50%

4.2 Final Evaluation

While the parameter selection and training described in the previous section was carried out on 15 images from the set of 75 images used in this work, the final evaluation of the trained classifiers as well as the segmentation technique is carried out on the remaining 60 images as was shown in Table 1 of **Section 3.4.1**. The 15 images used for training were taken from Bidwell's Advanced Calculus (1911) [127] . The remaining 60 images are separated into test sets, each containing 15 images from a separate book as illustrated in Table 11.

Table 11 The four tests which were carried out. Fifteen pages of the corresponding textbook was used in each test.

Test Name	**Textbook from which 15 Pages are Used for Testing**
Test1	D. Sloughter, Difference Equations to Differential Equations: An Introduction to Calculus (2000) [131]
Test2	E. Bidwell, Advanced Calculus (1911) [127]
Test3	A. C. Lunn, The Differential Equations of Dynamics (1909) [130]
Test4	A. S. Kompaneyets, Theoretical Physics (1961) [129]

The second test (Test2) shown in Table 11 is carried out on different pages taken from the same textbook which was used for training. Although performance was generally observed to be slightly better on the same textbook on which the training was carried out, overtraining is not a major concern since the results do not significantly differ between the datasets. The average evaluation results for each of the four detectors shown in Table 10, each averaged over all four tests shown in Table 11, are given in Table 12 and illustrated in Figure 68. The classifier names are here replaced by the term MEDS (Mathematical Expression Detection and Segmentation) followed by the corresponding number of the classifier used. Thus MEDS1 corresponds to RBFSVM_F_Ext1, MEDS2 corresponds with RBFSVM_F_Ext2, etc. Although segmentation is not being carried out yet at this stage, the MEDS modules that are tested use the same detectors that are evaluated here.

Table 12 Results of detection averaged over all four tests.

Classifier	**TPR**	**FPR**	**ACC**	**TNR**	**PPV**	**FDR**	**NPV**
MEDS1	82.77%	12.61%	87.36%	87.38%	63.94%	36.06%	94.03%
MEDS2	82.78%	12.57%	87.39%	87.43%	63.93%	36.06%	94.05%
MEDS3	83.12%	22.22%	80.88%	77.77%	59.46%	40.54%	89.15%
MEDS4	81.36%	10.35%	88.35%	89.65%	63.97%	36.03%	95.04%

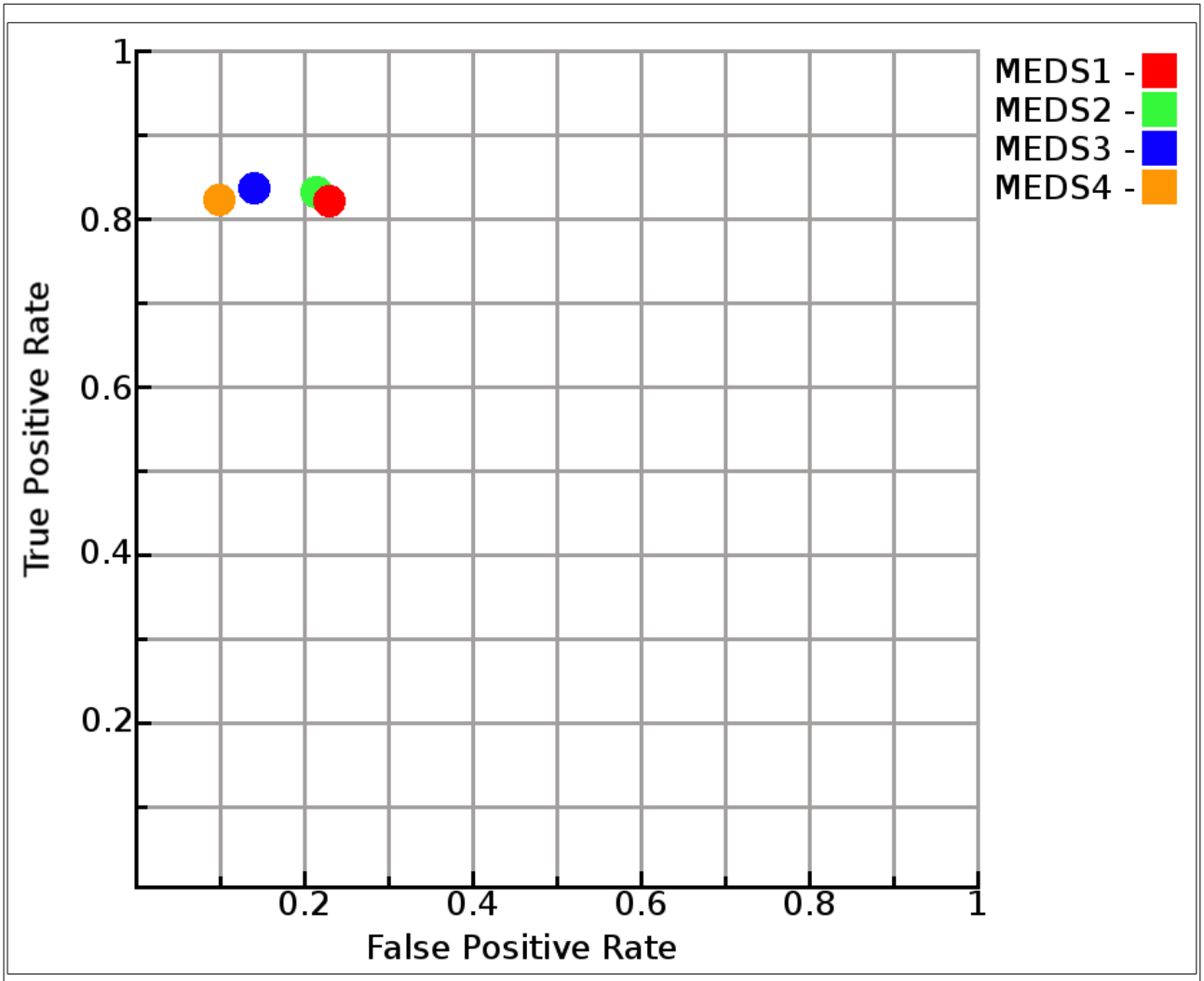


Figure 68: Graphical depiction of the overall average detection results on the four classifiers which were tested. The classifier trained without the italic feature (MEDS4) is shown to give the lowest false positive detection rate.

After an analysis of the detection evaluation results, it was observed that many of the false positive detections were small parts of valid words, or even stop-words like “the”, “at”, “and”, etc. In an attempt to mitigate such false positive recognition a post-processing step was employed after the detection which removes all blobs detected as math that are within stop-words. If blob detected as math is observed in a valid, non-math word that isn't a stop-word, then the ratio of math blobs to total blobs in that word has to be above an empirically chosen threshold of .6. The detection results after the post-processing step are shown in Table 13 and illustrated in Figure 69.

Table 13 Detection results after post-processing step is carried out to filter out obvious false positives. This also causes a slight decrease in true positive rate, but still results in an increase in overall accuracy.

Classifier	TPR	FPR	ACC	TNR	PPV	FDR	NPV
MEDS1	80.40%	9.285%	89.53%	90.71%	69.97%	30.03%	95.07%
MEDS2	80.43%	9.255%	89.55%	90.74%	70.06%	29.94%	95.09%
MEDS3	81.56%	15.89%	85.31%	84.11%	66.09%	33.91%	94.40%
MEDS4	79.82%	7.676%	90.32%	92.32%	70.61%	29.39%	95.01%

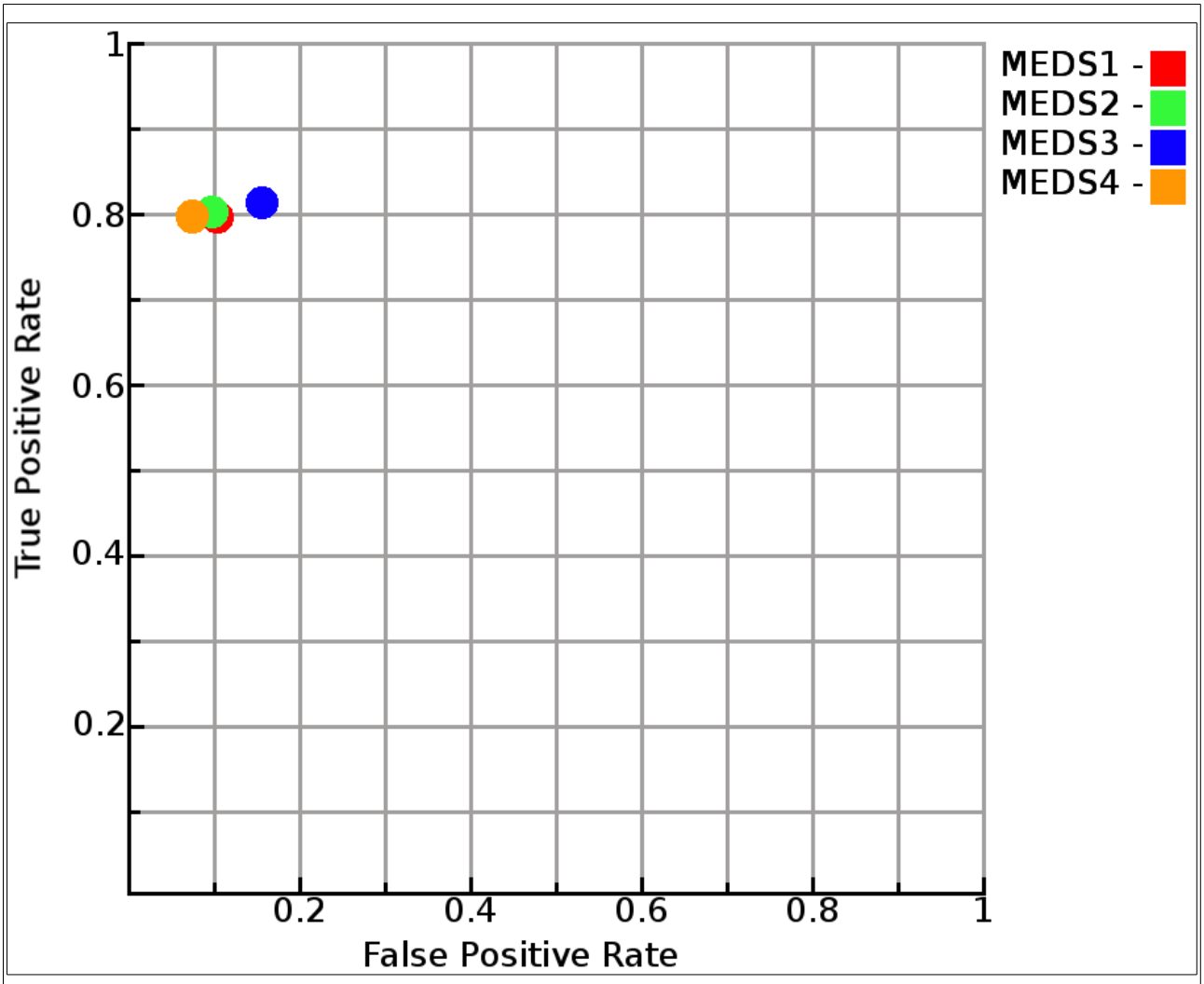


Figure 69 Graphical representation of the results shown in Table 9.

After the detection and post processing was evaluated, the final results after the segmentation algorithm described in **Section 3.4.2**. The recursive segmentation

algorithm employed here was observed to be very successful at minimizing the occurrence of oversegmentations and undersegmentations in the results. However, the algorithm was also observed to become rather time-consuming as the number of blobs in a segment increases. Each time a segment's size is increased, all of the blobs in that segment are required by the algorithm to be updated and rechecked for possible merges in all four directions. Although the current implementation shows significant potential for efficiency improvements, these are kept as ideas for future work due to time constraints. The final results after the detection, post-processing, and segmentation are carried out are given in Tables 14, 15, and 16 and illustrated by Figure 70. While Table 14 gives the pixel accurate metrics, Tables 15 and 16 give the region-wide statistics. The metrics from evaluating Tesseract's default equation detector are also included.

Table 14 Final pixel-accurate results of detection, post-processing, and segmentation.

MEDS	**TPR**	**FPR**	**ACC**	**TNR**	**PPV**	**FDR**	**NPV**
MEDS1	90.21%	15.32%	86.78%	84.68%	59.31%	40.69%	97.72%
MEDS2	90.26	15.35%	86.76%	84.65%	59.28%	40.72%	97.72%
MEDS3	91.99%	22.32%	82.15%	77.68%	54.85%	45.15%	97.52%
MEDS4	90.17%	13.40%	87.98%	86.60%	60.68%	39.32%	97.82%
Tesseract	34.14%	3.685%	86.50%	96.31%	62.18%	19.49%	87.13%

Experimental Results

Table 15 Region segmentation statistics for each MEDS module tested averaged over all four test sets. Avg. Overseg/Underseg refers to the average number of over/undersegmentationed regions per page. The severity is the average degree to which each such region is over/undersegmented (i.e., how many regions an oversegmented groundtruth region is split into by the hypothesis image).

MEDS	Avg Overseg.	Overseg. Severity	Avg. Underseg.	Unserseg. Severity
MEDS1	9.88	3.75	1.32	4.16
MEDS2	9.90	3.75	1.32	4.16
MEDS3	10.28	3.78	1.37	4.06
MEDS4	9.90	3.63	1.28	4.09
Tesseract	0.27	0.27	2.42	2.27

Table 16 Region-wide statistics for each MEDS module tested, averaged over all four test sets. The Correct Segmentation % is the ratio of groundtruth regions that had no overlapping false positive hypothesis pixels. Completely missed % is the ratio of groundtruth regions that had no overlapping true positive hypothesis pixels. The average falsely detected count is the average number of regions per page which have no true positive pixels.

MEDS	Correct Segmentation %	Completely Missed %	Avg. Falsely Detected Count
MEDS1	72.19%	11.28%	25.80
MEDS2	72.34%	11.14%	25.90
MEDS3	75.58%	9.595%	37.03
MEDS4	71.39%	11.26%	22.35
Tesseract	10.35%	76.67%	0.200

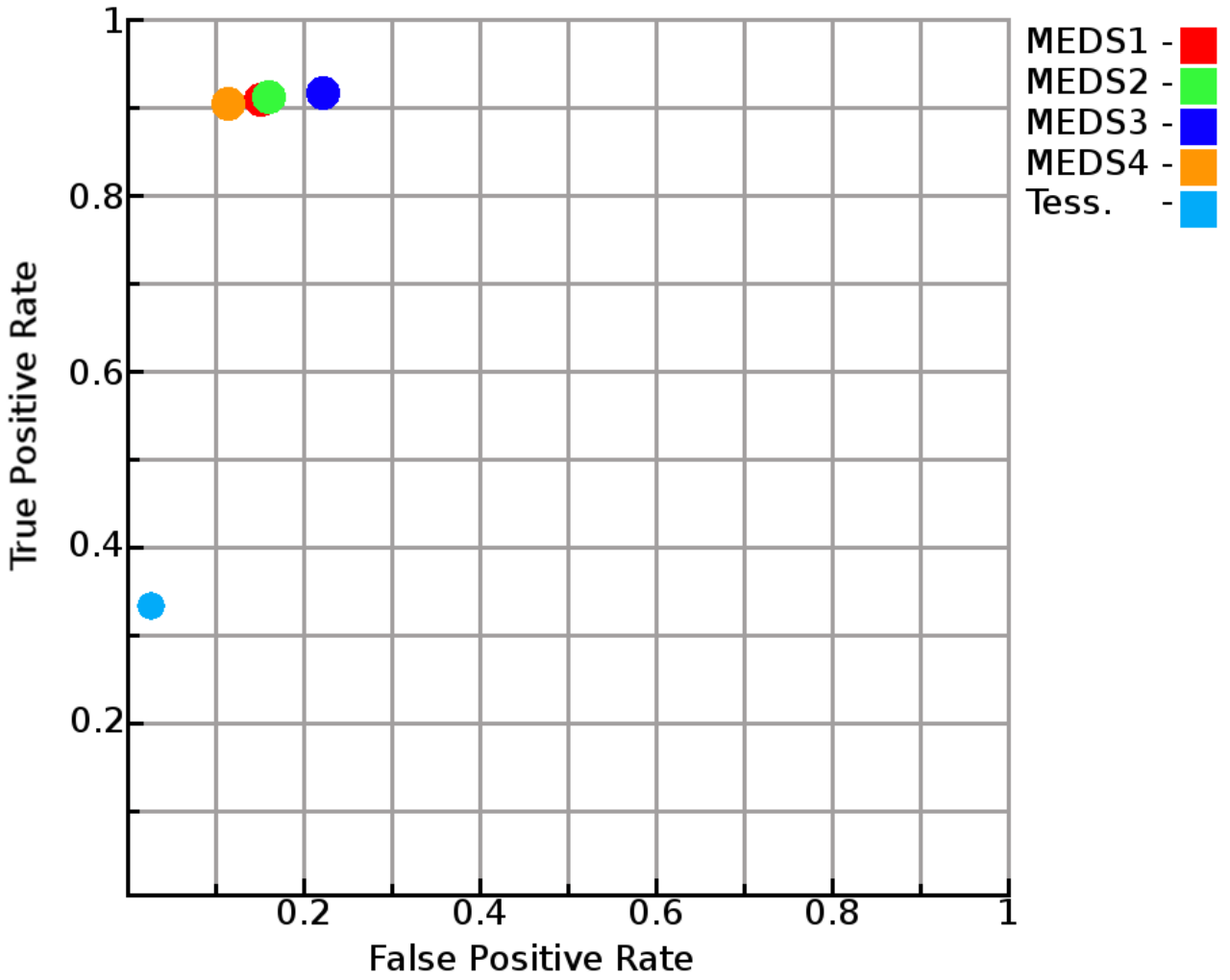


Figure 70 Graphical representation of the results shown in Table 10.

As illustrated by Table 14 and Figure 70, false detections are worsened significantly by the segmentation stage since more false regions are often improperly merged together. The true positive rate, however, is increased by nearly 10% during segmentation. The importance of minimizing false positive detections while maintaining an *acceptable* true positive rate is thus emphasized. Since nearly 10 oversegmented regions were observed on average per page, with an average severity of around 3 oversegmenations per segmented region, the segmentation module is far from perfect. Problems with undersegmentations can often be attributed to separators like commas, periods, or phrases like “or” and “such that” being improperly merged to a region. The results, however, are satisfactory for the scope of this current work. After giving the numerical statistics for the evaluation, a more intuitive explanation of the results is demonstrated by Figures 71-72. These images were automatically

generated during evaluation in order to keep track of each pixel as it is evaluated and to help avoid duplicate pixel counts. Each foreground pixel in the binarized image can be, upon evaluation, counted as a true positive, false positive, true negative, or false negative. The pixels are color-coded as shown in Table 17. The rest of these images can be viewed at [126].

Table 17 Pixel color codes used to keep track of pixels during evaluation

Pixel Type	**Color Code**
True Positive	Red
False Positive	Blue
True Negative	Orange
False Negative	Green

The following images are some of the final results from the MEDS4 detector/segmentor with pixels color-coded as shown in Table 17. MEDS4 was observed to have the highest accuracy. As mentioned previously, this particular dector/segmentor differs from the others in that Tesseract's italics feature is not used. This feature was observed not to be particularly accurate, having many false positives. It is likely that the inconsistency of this feature may have confused the SVM classifier during training. Other potential problems will be addressed in the conclusion/future work section of this thesis. While the Tesseract equation detector results have a very similar accuracy to the detector/segmentor implemented in this work, it has a highly different specificity and sensitivity as can be seen in Figure 70. The Tesseract equation detector succeeds in having a very false positive rate but unfortunately has a true positive rate that could be argued as too low for practical purposes given the evaluation results depending upon the intended application of course. The precision, however, is slightly higher than the MEDS4 precision. The significant amount of false positive detections and segmentations made by MEDS4 leaves room for improvement. Some of the aspects that need improvement will be discussed in the conclusion/future work section of this thesis.

Section 3.4 *Differentiation of Compositions of Functions* 3

where we have used the continuity of g at c to ascertain that s goes to 0 as h goes to 0. Putting (3.4.2), (3.4.3) and (3.4.4) together, we now have

$$(f \circ g)'(c) = f'(g(c))g'(c), \tag{3.4.5}$$

which is our desired result.

Chain Rule If f and g are differentiable, then

$$(f \circ g)'(x) = f'(g(x))g'(x). \tag{3.4.6}$$

Example Suppose $h(x) = (1 + x^2)^{10}$. Then $h(x) = f \circ g(x)$ where $g(x) = 1 + x^2$ and $f(x) = x^{10}$. Now

$$g'(x) = 2x$$

and

$$f'(x) = 10x^9,$$

so

$$h'(x) = (f \circ g)'(x) = f'(g(x))g'(x) = f'(1 + x^2)(2x) = 10(1 + x^2)^9(2x) = 20x(1 + x^2)^9.$$

Note that the preceding example is a particular case of the following general example. If g is a differentiable function, $n \neq 0$ is an integer, and $h(x) = (g(x))^n$, then $h(x) = f \circ g(x)$ where $f(x) = x^n$. Then we have

$$f'(x) = nx^{n-1},$$

and so

$$h'(x) = (f \circ g)'(x) = f'(g(x))g'(x) = n(g(x))^{n-1}g'(x).$$

That is,

$$\frac{d}{dx}(g(x)^n) = n(g(x))^{n-1}g'(x). \tag{3.4.7}$$

Example To illustrate the previous comments,

$$\frac{d}{dx}(3x - 2)^6 = 6(3x - 2)^5 \frac{d}{dx}(3x - 2) = 6(3x - 2)^5(3) = 18(3x - 2)^5.$$

Example For another illustration, if

$$f(x) = \frac{3}{(x^3 + 4)^5},$$

then

$$f'(x) = (-5)(3)(x^3 + 4)^{-6} \frac{d}{dx}(x^3 + 4) = -15(x^3 + 4)^{-6}(3x^2) = -\frac{45x^2}{(x^3 + 4)^6}.$$

Figure 71 MEDS4 final results on an image taken from [131].

while in the final state the atom went to the ground state and a quantum appeared in the field.

Let us divide the Hamiltonian of the system into two terms: $\hat{\mathscr{H}} = \hat{\mathscr{H}}^{(0)} + \hat{\mathscr{H}}^{(1)}$, where $\hat{\mathscr{H}}^{(0)}$ corresponds to the separated atom and field while $\hat{\mathscr{H}}^{(1)}$ describes the interaction. We then deduce a general formula for the transition probability, and apply it to a radiation. We shall therefore call $\hat{\mathscr{H}}^{(0)}$ the Hamiltonian of the unperturbed system, and regard $\hat{\mathscr{H}}^{(1)}$ as a small perturbation causing the transition. The eigenfunctions and eigenvalues of the operator $\hat{\mathscr{H}}^{(0)}$ are determined from the equation

$$-\frac{h}{i}\frac{\partial \psi_m^{(0)}}{\partial t} = \hat{\mathscr{H}}^{(0)}\psi_m^{(0)}. \tag{34.1}$$

Allowing for perturbation, the wave function satisfies the equation

$$-\frac{h}{i}\frac{\partial \psi}{\partial t} = (\hat{\mathscr{H}}^{(0)} + \hat{\mathscr{H}}^{(1)})\psi. \tag{34.2}$$

Considering that $\hat{\mathscr{H}}^{(1)}$ is a small perturbation, we represent the wave function in the form

$$\psi = \psi_i^{(0)} + \psi^{(1)}, \tag{34.3}$$

the "product" $\hat{\mathscr{H}}^{(1)}\psi^{(1)}$ will be neglected as being of the second order. Then, for $\psi^{(1)}$ we obtain the nonhomogeneous equation

$$-\frac{h}{i}\frac{\partial \psi^{(1)}}{\partial t} - \hat{\mathscr{H}}^{(0)}\psi^{(1)} = \hat{\mathscr{H}}^{(1)}\psi_i^{(0)}. \tag{34.4}$$

We shall look for $\psi^{(1)}$ in the form of an eigenfunction expansion of the operator $\psi^{(0)}$:

$$\psi^{(1)} = \sum_m c_m(t)\,\psi_m^{(0)}, \tag{34.5}$$

each of the functions $\psi_m^{(0)}$ satisfying the homogeneous equation (34.1). Substituting the series (34.5) in the nonhomogeneous equation and using the indicated property of the function $\psi_m^{(0)}$, we arrive at the following equality:

$$-\frac{h}{i}\sum_m \frac{\partial c_m}{\partial t}\psi_m^{(0)} = \hat{\mathscr{H}}^{(1)}\psi_i^{(0)}. \tag{34.6}$$

The coefficients c_m can be determined therefrom by taking advantage of the orthogonality property of the eigenfunctions $\psi_m^{(0)}$ (30.6). For this it is necessary to multiply both sides of (34.6) by $\psi_n^{(0)*}$ and integrate over a volume. Then only the term $-\frac{h}{i}\frac{\partial c_n}{\partial t}$ remains on the left, while on the right a certain integral is obtained which is characteristic of the perturbation method set out here:

Figure 72 MEDS4 results on an image taken from [129].

In order to illustrate the degree to which over-training to the specific textbook from which the 15 training images were taken is a concern, the average results for each individual dataset from the best performing detector/segmentor, MEDS4, is shown in Table 18. These are the results of detection and do not include any post-processing or segmentation. As was shown previously in Table 11, the Test2 images are taken from the same textbook as were the training images. Thus a significant improvement seen in the results of Test2 may be an indication that overtraining is a concern.

Table 18 Average Detection results for MEDS4 on each individual test.

Test	**TPR**	**FPR**	**ACC**	**TNR**	**PPV**	**FDR**	**NPV**
Test1	80.45%	17.70%	81.26%	82.29%	56.02%	43.98%	92.91%
Test2	83.36%	6.50%	91.36%	93.50%	71.71%	28.29%	95.40%
Test3	81.73%	10.10%	89.08%	89.90%	61.38%	38.62%	95.03%
Test4	79.90%	7.077%	91.72%	92.92%	66.78%	33.22%	96.82%

From Table 18, it can be argued that, although there may be a small degree of overtraining as indicated by the higher precision and true positive rate observed for Test2, the overtraining is not a major concern since the numbers are not very drastic. Test4 which is entirely unrelated to the dataset used for training (in fact the textbook used in Test4 was published more than 40 years after the one for Test2!) even has a slightly higher accuracy measurement than Test2.

Although the results of MEDS1 and MEDS2 (without the valid_word feature) are very similar, a relatively significant change is observed for MEDS3 and MEDS4 results. As indicated in Table 9, MEDS3 does not utilize the n-gram features or the valid_row feature, while MEDS4 simply discards the italics feature. Although the current n-gram Profile is generated only from a small amount of mathematical regions in the training set and may not be statistically useful in a larger sense, the drastic increase in false positive rate from MEDS1 to MEDS3 may indicate that the n-gram feature combined with the valid_row feature prevents a significant amount of false detections from taking place. If a more statistically significant n-gram profile were to be generated from a larger dataset and then applied to this work, it may reduce false detections even more greatly.

5 Conclusion and Future Work

The detection/segmentation technique utilized in this work can increase OCR accuracy in document images by allowing for a higher degree of document understanding prior to recognition. In order for mathematical regions to be properly recognized during OCR and not mangled with normal language text it is important that mathematical expression regions are detected and then properly segmented from their surroundings. The evaluation technique utilized in this work counts the true positive, false positive, true negative, and false negative pixels after detection and segmentation is carried out in order to get a highly accurate and objective understanding of performance. The count of oversegmentations, undersegmentations, falsely detected segmentations, and falsely missed segmentations on a page can also give a useful indication of performance. The mathematical detection and segmentation module implemented in this work has potential for significant improvement and also gives very favorable results overall.

There are many aspects of the mathematical detection and segmentation module which, if improved, could make the results even more favorable. Firstly, the feature extraction and segmentation code could be optimized for speed. During segmentation, some of the methods from the feature extractor are used repeatedly, and since these methods have not been optimized the speed of the program is reduced significantly. This is especially true for larger segmentations where, each time the segment's size is increased, all of the blob's inside of it need to be re-evaluated for potential merges in all directions. The segmentation algorithm currently implemented also needs to be modified for detecting separators and mathematical words to further enhance the accuracy.

Perhaps the most important future work item for both increasing true positive rate and decreasing false positive rate, would be ensuring the proper identification of “abnormal” rows of text which are more likely to contain one or more displayed expressions. While the current method can be satisfactory in some instances it is often wrong. Implementing an accurate abnormal row detector is outside of the scope of this work, but would improve the usefulness of several of the features which are extracted only for abnormal rows. When a row is misclassified as abnormal, it can result in an entire row of normal text being improperly segmented as a mathematical region. A separate SVM for detecting abnormal rows may be a possible avenue for future work. Header and footer rows need to have been found either prior to mathematical

expression detection/segmentation being carried out or as part of the overall process. Knowing that a row is a header, and not part of normal sentence structure is important. Math may occur in the header however, so it is still important that headers are taken into consideration. The current implementation simply assumes that the first row is the header. While this is often the case, it has been observed that the first row or couple of rows may be noise, in which cases both the header and first row of normal text are improperly seen as abnormal.

Another important future work item is to generate more data. The current amount of pages, 75, is a very small amount of data, and makes it difficult to get a truly objective understanding of the results. Getting data which is more statistically significant is also important. Only 5 textbooks have been utilized in this work, which although satisfactory for a small test set, are not as representative as would be desired for practical applications. Adding a math symbol recognizer to the MEDS module would be extremely useful. If math symbols could be recognized then it would be possible to find regions that may have been missed by the detection phase during segmentation. For any missed regions detected, the segmentation step could then be repeated until no more missed regions are found. Support for detecting displayed expression labels is also kept as an idea for future work. Detecting labels which refer to displayed expressions would increase overall document understanding.

Improved italics and bold detection would also be extremely useful in making the detector more robust. The italics and bold detection implemented as part of Tesseract was used in feature extraction for this work but was found to not be very accurate and was thus not used to train the final classifier. Implementing italic/bold detection from scratch is outside of the scope of this work but would be very useful.

[1] K. Wilcox and A. Stephen. "Are Close Friends the Enemy? Online Social Networks, Self-Esteem, and Self-Control," *Forthcoming Columbia Business School Research Paper No. 12-57*, Date posted: October 3, 2012.

[2] D. A. Vise and M. Malsee, *The Google Story*. New York City: Dell Publishing, 2005.

[3] H. F. Shantz, *The History of OCR, Optical Character Recognition*. Manchester Center: Recognition Technologies Users Association, 1982.

[4] S. V. Rice, F. R. Jenkins, and T. A. Nartker, "The Fourth Annual Test of OCR Accuracy," Technical Report 95-03, Information Science Research Institute, University of Nevada, Las Vegas, July 1995.

[5] L. Vincent. "Google Book Search: Document Understanding on a Massive Scale," *International Conference on Document Analysis (ICDAR)*, 2007, pp. 819 - 823.

[6] R. Unnikrishnan, and R. Smith. "Combined Script and Page Orientation Estimation Using the Tesseract OCR Engine," *International Workshop of Multilingual OCR*, 25th July 2009, Barcelona, Spain.

[7] Z. Huang, M. Cmejrek, and B. Zhou. "Soft Syntactic Constraints for Hierarchical Phrase-Based Translation Using Latent Syntactic Distributions," *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010, pp. 138–147.

[8] P. W. Handel, "Statistical Machine," United States Patent Office. 1,915,993, Jun. 27, 1933.

[9] A. Kleiner, and R. Kurzweil, "A Description of the Kurzweil Reading Machine and a Status Report on Its Testing and Dissemination," *Bulletin of Prosthetics Research*, vol. 27, no. 10, Spring 1977, pp. 72-81.

[10] M. Bokser, "Omnidocument Technologies," *Proceedings of the IEEE*, vol. 80, no. 7, July 1992, pp. 1066-1078.

[11] ABBYY FineReader, "ABBYY FineReader for Personal Use," Internet: http://www.nuance.com/for-business/by-product/omnipage/professional/index.htm, Date Accessed: 2013.

[12] Nuance Inc., "OmniPage Professional," Internet: :http://www.nuance.com/for-business/by-product/omnipage/professional/index.htm, Date Accessed: 2013.

[13] Iris Products and Technologies, "Introducing the New Readiris 14," Internet: http://www.irislink.com/c2-2115-189/Readiris-14--OCR-Software--Scan--Convert---Manage-your-Documents-.aspx, Date Accessed: 2013.

Bibliography

[14] Contributor: Bob Stein (uploader to http://archive.org), "New York Times August September 1901 Collection," Internet: http://archive.org/download/NewYorkTimesAugSept1901Collection/New_York_Times_August_September_1901_Part_7_text.pdf, Date Accessed: 2013.

[15] T. M. Breuel, and U. Kaiserslautern. "The hOCR Microformat for OCR Workflow and Results," *Ninth International Conference on Document Analysis and Recognition (ICDAR)*, 2007, pp. 1063 - 1067.

[16] R. Griffin, *Statistics*. London: Macmillon and Co., 1913, pp. 121-122.

[17] R. Zanibbi and D. Blostein, "Recognition and Retrieval of Mathematical Expressions," *IJDAR*, vol. 15, no. 4, December 2012, pp. 331-357.

[18] W. R. Smith. "Hybrid Page Layout Analysis via Tab-Stop Detection," *Proceedings of the 10th International Conference on Document Analysis and Recognition*, 2009, pp. 241-245.

[19] F. d'Albe. "On a Type-Reading Optophone," *Proc. Roy. Soc., Lond.*, 1914, pp. 373-375.

[20] G. Tauschek, "Reading Machine," United States Patent Office. 2,663,758, Dec. 22, 1953.

[21] M. Martin, "Reading Machine Speaks Out Loud," *Popular Science*, vol. 154, no. 2, Feb 1949, pp. 125-127.

[22] D. Shepard, "Apparatus for Reading," United States Patent Office. 2,663,758, Dec. 22, 1953.

[23] J. Leimer. "Design Factors in the Development of an Optical Character Recognition Machine," *IRE Transaction on Information Theory*, 1962, pp. 167-171.

[24] M. H. Weik, "A Fourth Survey of Domestic Electronic Digital Computing Systems," Internet: http://ed-thelen.org/comp-hist/BRL64-i.html#IBM-1401, Date Accessed: 2013..

[25] IBM, "IBM Systems Reference Library," 1964.

[26] The IBM 1401 Demo Lab and Restoration Project Computer History Museum, "IBM 1401 Archive Pics," Internet: http://ibm-1401.info/IBM1401_ArchivePics.html, Date Accessed: 2014.

[27] L. O. Eikvil, "Optical Character Recognition," Norwegian Computing Center, 1993.

[28] J. J. Hull, and S. L. Taylor. "Document Image Skew Detection: Survey and Annotated Bibliography," *World Scientific*, 1998, pp. 40-64.

[29] R. Smith, "Apparatus and Method for Use in Image Processing," United States Patent Office. 5,583,949, Dec 10, 1996.

[30] S. Li, Q. Shen, and J. Sun, "Skew Detection Using Wavelet Decomposition and Projection Profile Analysis," *Pattern Recognition Letters*, vol. 28, no. 5, Jan 2007, pp. 555-562.

[31] W. Postl. "Detection of Linear Oblique Structures and Skew Scan in Digitized Documents," *Proc. 6th Int. Conf. Pattern Recognition*, 1986, pp. 687-689.

[32] S. N. Srihari, and V. Govindaraju, "Analysis of Textual Images Using the Hough Transform," *Machine Vision and Applications*, vol. 2, no. 1, Jan 1989, pp. 141-153.

[33] R. Smith, "A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation," *Proc. of the 3rd Int. Conf. on Document Analysis and Recognition*, vol. 2, no. 1, Jan 1995, pp. 1145-1148.

[34] R. Smith. "An Overview of the Tesseract OCR Engine," *Proc. Int. Conf. Document Anal. Recognit.*, 2007, pp. 629-633.

[35] S. S. Bukhari, F. Shafait, and T. M. Breuel. "Coupled Snakelet Model for Curled Textline Segmentation of Camera-Captured Document Images," *Proc. 10th Int. Conf. on Document Analysis and Recognition*, 2009, pp. 33-53.

[36] L. Likforman-Sulem, A. Zahour, and B. Taconet, "Text Line Segmentation of Historical Documents: A Survey," *International Journal of Document Analysis and Recognition*, vol. 9, no. 2, Jan 2007, pp. 123-138.

[37] R. Smith. "Tesseract OCR Engine: What It Is, Where It Came From, Where It Is Going," *OSCON*, 2007.

[38] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical Review of OCR Research and Development," *Proceedings of the IEEE*, vol. 80, no. 7, Jul 1992, pp. 1029-1058.

[39] S. Mori, N. Hirobumi, and Y. Hiromitsu, *Optical Character Recognition*. New York: Wiley & Sons, Inc., 1999, pp. 193-367.

[40] R. Smith, "History of the Tesseract OCR Engine: What Worked and What Didn't. How to Build a World-Class OCR Engine in Less Than 20 Years," *SPIE-IS&T*, vol. 8658, no. 2, Feb 2013, p. 12.

[41] F. L. Alt, "Digital Pattern Recognition by Moments," *Journal of Associated Computing Machinery*, vol. 9, no. 1, Jan 1962, pp. 240-258.

[42] R. Casey. "Moment Normalization of Hand Printed Characters," *IBM Journal of Research and Development*, 1970, pp. 548-557.

[43] AForge.NET, "Blobs Processing," Internet: http://www.aforgenet.com/framework/features/blobs_processing.html, Date Accessed: 2013.

[44] M. D. McIlroy, "Development of a Spelling List," *IEEE Trans on Communications*, vol. 30, no. 1, Jan 1982, pp. 91-99.

[45] G. Nagy, "At the Frontiers of OCR," *Proc. IEEE*, vol. 80, no. 2, Jul 1992, pp. 1093-1100.

[46] Y. Y. Tang, C. D. Yan, and C. Y. Suen, "Document Processing for Automatic Knowledge Acquisition," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 1, Feb 1994, pp. 3-21.

[47] A. M. Namboodiri, and A. Jain. "Document Structure and Layout Analysis," *Advances in Pattern Recognition*, Springer-Verlag, London 2007.

[48] M. Nadler, "A Survey of Document Segmentation and Coding Techniques," *Computer Vision Image Graphics Process*, vol. 28, no. 2, Nov 1984, pp. 240-262.

[49] R. Haralick. "Document Image Understanding: Geometric and Logical Layout," *Proc. IEEE Conf. Computer Vision and Pattern Recognition. Seattle*, 1994, pp. 385-390.

[50] Y. Y. Tang, S. W. Lee, and C. Y. Suen, "Automatic Document Processing: a Survey," *Pattern Recognition*, vol. 29, no. 12, Dec 1996, pp. 1931-1952.

[51] S. Mao, A. Rosenfeld, and T. Kanungo, "Document Structure Analysis Algorithms: a Literature Survey," *Document Recognition and Retrieval*, vol. 5010, no. 10, Jan 2003, pp. 197-207.

[52] N. Chen, and D. Blostein, "A Survey of Document Image Classification: Problem Statement, Classifier Architecture and Performance Evaluation," *International Journal on Document Analysis and Recognition*, vol. 10, no. 1, May 2007, pp. 1-16.

[53] S. Marinai. "Introduction to Document Analysis and Recognition," *Machine Learning in Document Analysis and Recognition, Berlin, Germany: Springer*, 2008, pp. 1-20.

[54] G. Nagy, S. Seth, and M. Viswanathan, "A Prototype Document Image Analysis System for Technical Journals," *Computer*, vol. 25, no. 1, Jan 1992, pp. 10-22.

[55] Y. N. Elglaly, F. Quek, T. Smith-Jackson, and D. Dhillon. "Touch-Screens Are Not Tangible: Fusing Tangible Interaction With Touch Glass in Readers for the Blind," *ACM International Conference on Tangible, Embedded and Embodied Interaction (TEI)*, 2013, pp. 245-252.

[56] F. Goudail, *Statistical Image Processing for Noisy Images*. New York: Klewer Academic Plenum Publishers, 2004.

[57] R. Miller, "Ink-Jet Basics," Internet: http://www.thetonesystem.com/inkjet_basics.html, Date Accessed: 2013.

Bibliography

[58] O. G. Guleryuz, "A Multiresolutional Algorithm for Halftone Detection," *Proc. SPIE Image and Video Communications and Processing*, vol. 5685, no. 1, Jan 2005, pp. 1098-1105.

[59] N. Otsu, "A Threshold Selection Method From Gray-Level Histograms," *IEEE Trans. Sys. Man. Cyber*, vol. 9, no. 1, Jan 1979, pp. 62-66.

[60] B. Xie, and G. Agam, "Boosting Based Text and Non-Text Region Classification," *Document Recognition and Retrieval Proc. SPIE*, vol. XVIII, no. 1, Jan 2011, pp. 1-9.

[61] A. Gourdol, "CSS3 Regions: Rich Page Layout With HTML And CSS3," Internet: , Date Accessed: 2013.

[62] T. Pavlidis, and J. Zhou, "Page Segmentation and Classification," *Graphical Models and Image Processing*, vol. 54, no. 1, Jan 1992, pp. 484-496.

[63] H. S. Baird, H. Bunke, and P. S. Wang. "Background Structure in Document Images," *Document Image Analysis, World Scientific*, 1994, pp. 17-34.

[64] G. Nagy, and S. Seth. "Hierarchical Representation of Optically Scanned Documents," *Proc. of the 17th Conf. on Pattern Recognition*, 1984, pp. 347-349.

[65] F. Wahl, K. Wong, and R. Casey, "Block Segmentation and Text Extraction in Mixed Text/Image Documents," *Graphical Models and Image Processing*, vol. 20, no. 1, Jan 1982, pp. 375-390.

[66] J. Higashino, H. Fujisawa, Y. Nakano, and M. Ejiri. "A Knowledge-Based Segmentation Method for Document Understanding," *Proc. 8th Int. Conf. on Pattern Recognition*, 1986.

[67] A. Dengel, and F. Dubiel, "Computer Understanding of Document Structure," *International Journal of Imaging Systems and Technology*, vol. 7, no. 1, Jan 1996, pp. 271-278.

[68] A.K. Jain, Y. Zhong, "Page Segmentation using Texture Analysis," *Pattern Recognition*, vol. 29, no. 5, May 1996, pp. 743–770.

[69] T. Tokuyasu and P. A. Chou, "Turbo Recognition: A Statistical Approach to Layout Analysis," *In Proceedings of the SPIE*, vol. 4307, no. 1, Jan San Jose, CA, 2001, pp. 123-129.

[70] J. P. Bixler. "Tracking Text in Mixed-Mode Document," *Proc. ACM Conference on Document Processing System*, 1998, pp. 177-185.

[71] T. Pavlidis, "Algorithms for Graphics and Image Processing," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 63, no. 8, Jan 1983, p. 395.

[72] L. O'Gorman, "The Document Spectrum for Page Layout Analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, Nov 1993, pp. 162-173.

[73] S. Arya, T. Malamotos, and D. M. Mount. "Space-Efficient Approximate Voronoi Diagrams," *Proc. 34th ACM Sympos. Theory Comput.*, 2002, pp. 721-730.

[74] Wikipedia, "Voronoi Diagram," Internet: http://en.wikipedia.org/wiki/Voronoi_diagram, Date Accessed: 2013.

[75] K. Kise, A. Sata, and M. Iwata, "Segmentation of Page Images Using the Area Voronoi Diagram," *Computer Vision and Image Understanding*, vol. 70, no. 3, Jun 1998, pp. 370-382.

[76] E. G. Johnston, "Printed Text Discrimination," *Computer Graphics and Image Processing*, vol. 3, no. 1, Mar 1974, pp. 83-89.

[77] D. S. Bloomberg. "Multiresolution Morphological Approach to Document Image Analysis," *1st International Conference of Document Analysis and Recognition*, 1991, pp. 963-971.

[78] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River NJ: Prentice Hall, 2008, pp. 627-688.

[79] J. Serra, *Image Analysis and Mathematical Morphology*. New-York: Academic Press, 1982.

[80] J. Liu, Y. Tang, Q. He, and C. Suen. "Adaptive Document Segmentation and Geometric Relation Labeling: Algorithms and Experimental Results," *Proc. 13th Int'l Conf. Pattern Recognition*, 1996, pp. 763-767.

[81] Esposito, D. Malerba, and G. Semeraro. "A Knowledge-Based Approach to the Layout Analysis," *Proc. Third Int'l Conf. Document Analysis and Recognition*, 1995, pp. 466-471.

[82] M. Okamoto and M. Takahashi. "A Hybrid Page Segmentation Method," *Proc. 2nd Int. Conf. on Document Analysis and Recognition*, 1993, pp. 743-748.

[83] T.M. Breuel. "Two Geometric Algorithms for Layout Analysis," *Proceedings of the 5th International Workshop on Document Analysis Systems V*, 2002, pp. 188–199.

[84] H.S Baird, S.E Jones, and S.J Fortune. "Image Segmentation by Shape-Directed Covers," *Proceedings 10th International Conference on Pattern Recognition*, June, 1990, pp. 820–825.

[85] T. M. Breuel. "The OCRopus Open Source OCR System," *Proc. IS&T/SPIE 20th Annu. Symp.*, pp. 1 -15, 2008.

[86] I. Phillips, B. Chanda, and R. Haralick, "UW-III English/Technical Document Image Database," Internet: http://www.science.uva.nl/research/dlia/datasets/uwash3.html, Date Accessed: 2013.

[87] F. Shafait. "Geometric Layout Analysis of Scanned Documents," *PhD thesis, University of Kaiserslautern*, 2008.

[88] F. Shafait. "Document Image Analysis with OCRopus," *Proc. IEEE Int'l Multitopic Conf.*, 2009.

[89] W. Abd Almageed, M. Agrawal, W. Seo, and D. Doermann. "Document-zone Classification using Partial Least Squares and Hybrid Classifiers," *Proc. Int'l Conf. on Patt. Reco.*, 2008, pp. 1-4.

[90] F. Cesarini, M. Lastri, S. Marinai, and G. Soda. "Encoding of Modified X-Y Trees for Document Classification," *Int. Conf. on Document Analysis and Recognition (ICDAR)*, 2001, pp. 1131–1135.

[91] C. Shin and D. Doermann. "Classification of Document Page Images Based on Visual Similarity of Layout Structures," *Proc. SPIE Conf. Document Recognition and Retrieval VII*, 2000, pp. 182-190.

[92] F. Esposito, D. Malerba, and G. Semeraro, "Classification in Noisy Environments using a Distance Measure Between Structural Symbolic Descriptions," *IEEE Trans. on PAMI*, vol. 14, no. 3, March 1992, pp. 390–402.

[93] N. Chen and D. Blostein, "A Survey of Document Image Classification: Problem Statement, Classifier Architecture and Performance Evaluation," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 10, no. 1, June 2007, pp. 1-16.

[94] K. Fan and L. Wang, "Classification of Document Blocks Using Density Feature and Connectivity Histogram," *Pattern Recognition Letters*, vol. 16, no. 9, September 1995, pp. 955-962.

[95] D. Chetverikov, J. Liang, J. Komuves, and R. Haralick. "Zone Classification Using Texture Features," *Proc. 13th Int'l Conf. Pattern Recognition*, 1996.

[96] Y. Wang, R. Haralick and I. Phillips, "Document Zone Content Classification and its Performance Evaluation," *Pattern Recognition*, vol. 39, no. 1, Jan 2006, pp. 57 -73.

[97] W. Abd Almageed, M. Agrawal, W. Seo, and D. Doermann. "Document-zone classification using partial least squares and hybrid classifiers," *Proc. Int'l Conf. on Patt. Reco.*, 2008, pp. 1-4.

[98] M. Okamoto and A. Miyazawa, *An Experimental Implementation of a Document Recognition System for Papers Containing Mathematical Expressions*. Berlin: Springer, 1992.

[99] Hsi-Jian Lee and Jiumn-Shine Wang. "Design of a Mathematical Expression Recognition System," *Proceedings of the Third International Conference on Document Analysis and Recognition*, Aug 1995, pp. 1084-1087.

[100] K. Inoue, R. Miyazaki and M. Suzuki. "Optical Recognition of Printed Mathematical Documents," *Proc. Third Asian Technology Conf. Math.*, 1998, pp. 280-289.

[101] R. J. Fateman. "How to Find Mathematics on a Scanned Page," *Proc. SPIE 3967*, 1999.

[102] J. Y. Toumit, S. Garcia-Salicetti, and H. Emtoz. "A Hierarchical and Recursive Model of Mathematical Expressions for Automatic Reading of Mathematical Documents," *Proceedings of Fifth International Conference on Document Analysis and Recognition (ICDAR)*, 1999, pp. 119–122.

[103] B.B. Chaudhuri and U. Garain. "An Approach for Recognition and Interpretation of Mathematical Expressions in Printed Document," *Pattern Analysis and Applications, vol. 3*, 2000, pp. 120-131.

[104] A. Kacem, A. Belaid, B. M. Ahmed, "Automatic Extraction of Printed Mathematical Formulas using Fuzzy Logic and Propagation of Context," *International Journal of Document Analysis and Recognition*, vol. 4, no. 2, December 2001, pp. 97-108.

[105] J. Jin, X. Han and Q. Wang. "Mathematical Formulas Extraction," *Proc. of the 7th Int'l Conf. Document Analysis and Recognition (ICDAR), Edinburgh, Scotland*, 2003, pp. 1138-1141.

[106] D.M. Drake, H.S. Baird. "Distinguishing Mathematics Notation from English Text Using Computational Geometry," *Proceedings of the International Conference on Document Analysis and Recognition*, 2005, pp. 1270–1274.

[107] X. D. Tian, W. Z. Sun, M. H. Ha. "Research on Optical Formulas Extraction," *Proceedings of the 4th International Conference On Machine Learning and Cybernetics*, August 2005, pp. 4886-4890.

[108] S Yamazaki, F Furukori. "Embedding a Mathematical OCR Module into OCRopus," *International Conference on Document Analysis and Recognition*, 2011, pp. 880 - 884.

[109] X. Lin, L. Gao, Z. Tang, X. Hu, and X. Lin. "Identification of Embedded Mathematical Formulas in PDF Documents using SVM," *Proc of Document Recognition and Retrieval XIX*, 2012.

[110] K. Chan and D. Yeung, "Mathematical Expression Recognition: A Survey," *Int',l J. Document Analysis and Recognition*, vol. 3, no. 1, August 2000, pp. 3-15.

[111] C. D. Malon, S. Uchida, and M. Suzuki, "Mathematical symbol recognition with support vector machines," *Pattern Recognition Letters*, vol. 29, no. 9, July 2008, pp. 1326–1332.

[112] Hsi-Jian Lee and Jiumn-Shine Wang, "Design of a Mathematical Expression Understanding System," *Pattern Recognition Letters*, vol. 18, no. 3, March 1997, pp. 289-298.

[113] Chowdhury, S.P., Mandal,S., Das, A.K., Chanda, B.. "Automated Segmentation of Math-zones from Document Images," *7th International Conference on Document Analysis and Recognition vol. 2*, 2003, pp. 755–759.

[114] U Garain, BB Chaudhuri, and A. R. Chaudhuri. "Identification of Embedded Mathematical Expressions in Scanned Documents," *Int. Conf. Pattern Recognition*, 2004, pp. 384-387.

[115] U. Garain and B.B. Chaudhuri, "A Corpus for OCR of Printed Mathematical Expressions," *Int'l. Journal of Document Analysis and Recognition (IJDAR)*, vol. 7, no. 4, September 2005, pp. 241-259.

[116] U. Garain. "Identification of Mathematical Expressions in Document Images," *Proc. Int'l Conf. on Document Analysis and Recognition*, 2009, p. 1340.

[117] S. Uchida, A. Nomura, and M. Suzuki, "Quantitative Analysis of Mathematical Documents," *IJDAR*, vol. 7, no. 4, June 2005, pp. 211-218.

[118] X Lin, L Gao, Z Tang, J Baker, M Alkalai, V Sorge. "A Text Line Detection Method for Mathematical Formula Recognition," *12th International Conference on Document Analysis and Recognition*, 2013, pp. 339-343.

[119] M Alkalai, J B Baker, V Sorge, X Lin. "Improving Formula Analysis with Line and Mathematics Identificatio," *12th International Conference on Document Analysis and Recognition*, 2013, pp. 334-338.

[120] X Lin., L Gao, Z Tang, X Lin and X Hu. "Performance Evaluation of Mathematical Formula Identification," *10th International Workshop on Document Analysis Systems*, 2012, pp. 287-291 .

[121] J Baker, "Maxtract," Internet: http://www.cs.bham.ac.uk/research/groupings/reasoning/sdag/maxtract.php, Date Accessed: 2013.

[122] D. Bloomberg, "Leptonica ," Internet: http://www.leptonica.com/, Date Accessed: 2013.

[123] Marinai and Nesi. "Projection Based Segmentation of Musical Sheets," *ICDAR, Bangalore, India*, 1999.

Bibliography

[124] J Baker, AP Sexton, V Sorge. "Comparing Approaches to Mathematical Document Analysis from PDF," *ICDAR*, 2011.

[125] SS Bukhari, F Shafait, and TM Breuel. "Document Image Segmentation Using Discriminative Learning Over Connected Components," *9th IAPR International Workshop on Document Analysis Systems*, 2010, pp. 183-190.

[126] J. Bruce, "Isagoge," Internet: http://sourceforge.net/projects/projectisagoge/, Date Accessed: 2013.

[127] E Bidwell, *Advanced Calculus*. Boston: The Athenum Press: Gin and Company Proprieters, 1911.

[128] Z Liu, "EquationDetectBase.h," Internet: https://code.google.com/p/tesseract-ocr/source/browse/trunk/textord/equationdetectbase.h?r=842, Date Accessed: 2013.

[129] A. S Kompaneyets, *Theoretical Physics*. Osmania University: Foreign Languages Publishing House, 1961.

[130] A. C. Lunn, *The Differential Equations of Dynamics*. Lancaster, PA: The New Era Printing Company, 1909.

[131] D. Sloughter, *Difference Equations to Differential Equations: An Introduction to Calculus*. Furman University, Greenville, SC: Creative Commons, 2000.

[132] J Grieves, "Open Scan and Read," Internet: http://sourceforge.net/projects/osr/, Date Accessed: 2008.

[133] R. Duda, P. Hart, D. Stork, *Pattern Classification (Second Edition)*. New York: A Wiley-Interscience Publication, 2001.

[134] K Davis, "DLib C++ Library," Internet: http://dlib.net, Date Accessed: 2013.

[135] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers," *5th Annual ACM Workshop on COLT, Pittsburgh, PA*, 1992.

[136] M. Varma, "The Standard SVM Formulation," Internet: http://research.microsoft.com/en-us/um/people/manik/projects/trade-off/svm.html, Date Accessed: 2013.

[137] C. Cortes and V. Vapnik, "Support Vector Networks," *Machine Learning*, vol. 20, no. 3, September 1995, 273-297.

[138] H. Kim, S. Pang, H. Je, D. Kim, S.Y. Bang, "Support Vector Machine Ensemble with Bagging," *Lecture Notes in Computer Science*, vol. 2388, no. , Feb 2002, 397-408.

[139] C.W. Hsu. "A Practical Guide to Support Vector Classification," *Department of Computer Science, Tech. rep. National Taiwan University*, 2003.

[140] Powell, MJD. "The BOBYQA Algorithm for Bound Constrained Optimization Without Derivatives," *Technical Report NA2009/06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge*, 2009.

[141] Z Liu, "equationdetect.h," Internet: https://code.google.com/p/tesseract-ocr/source/browse/trunk/ccmain/equationdetect.h?r=840, Date Accessed: 2013.