

# Cross-comparison of Digital and Digitized Physical Evidence

John Erik Rekdal



Master's Thesis  
Master of Science in Information Security  
30 ECTS  
Department of Computer Science and Media Technology  
Gjøvik University College, 2014

Avdeling for  
informatikk og medieteknikk  
Høgskolen i Gjøvik  
Postboks 191  
2802 Gjøvik

Department of Computer Science  
and Media Technology  
Gjøvik University College  
Box 191  
N-2802 Gjøvik  
Norway

## Abstract

In forensics, investigations comprise diverse types of evidence. For example digital evidence in form of electronic documents and physical evidence, e.g. printed paper documents. One major challenge is to efficiently and accurately link digital evidence and physical evidence together. In particular, a computational method is needed to deal with the huge amount of data available in a forensic investigation and to reduce the time spent on linking and analyzing the different types of evidence.

The thesis aims to improve the efficiency and effectiveness of this process by using computational methods such as plain text search (String search), approximate string matching and OCR (optical character recognition), and to incorporate these in a proof-of-concept tool. The tool is used for an experimental setup for testing of linking accuracy between similar and dissimilar documents. A dataset was created and used for testing, based on feedback from Økokrim<sup>1</sup>. The thesis seeks to answer how OCR affects evidence linking, characteristics of a forensics dataset, characteristics that enables linking and how it is possible to increase efficiency in evidence linking.

The proof-of-concept tool, contains five methods for comparison, four text comparison methods; Levenshtein distance, Word frequency, Cosine similarity and W-shingles. And one image-to-image comparison; a pixel-to-pixel similarity. It uses Optical Character Recognition for text generation from scanned documents. Text extraction from digital documents are done through Java libraries.

The results shows that W-shingles is the best performing algorithm for matching documents in this setting, and that text sanitation does not have any practical influence on W-shingles, whereas it does increase the matching accuracy for the remaining methods. Characteristics that enables evidence linking was found to be shingles and frequency of unique words used in Cosine similarity. Characteristics in a dataset consisting of DOCX documents are bold font style, and the combination of font size 11 and font type Calibri, which is the default combination for Microsoft Word<sup>®</sup> 2007, 2010 and 2013. The efficiency and accuracy of OCR can be increased by using ensemble voting and decreasing runtime. As for OCR error rate in a forensics environment it is a nonissue since it is not used to recreate evidence, but for matching and locating evidence.

---

<sup>1</sup>The Norwegian National Authority for Investigation and Prosecution of Economic and Environmental Crime

## Preface

Thanks to my supervisors Katrin Franke and Slobodan Petrovic. And a thanks to my fellow students, especially Pieter Ruthven, André Nordbø, Espen Didriksen and Andrii Shalaginov throughout the two years of the master studies for good help, feedback and camaraderie. I would also like to thank Odd Tore Bøe for getting me out of the man cave and provide some good food, company and discussions in between the battles.

## Contents

<b>Abstract</b> . . . . .	<b>ii</b>
<b>Preface</b> . . . . .	<b>iii</b>
<b>Contents</b> . . . . .	<b>iv</b>
<b>List of Figures</b> . . . . .	<b>vi</b>
<b>List of Tables</b> . . . . .	<b>viii</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
1.1 Keywords . . . . .	1
1.2 Covered topics . . . . .	1
1.3 Research questions . . . . .	1
1.4 Justification, Motivation and Benefits . . . . .	2
1.5 Limitations . . . . .	2
1.6 Contribution . . . . .	3
1.7 Thesis Structure . . . . .	3
<b>2 Related work</b> . . . . .	<b>4</b>
2.1 Digital Forensics and Digital Evidence . . . . .	4
2.2 Optical Character Recognition . . . . .	4
2.3 Hashing and Fuzzy hashing . . . . .	12
2.4 Dataset and characteristics . . . . .	13
2.5 Text comparison and evidence linking . . . . .	14
<b>3 Methodologies</b> . . . . .	<b>17</b>
3.0.1 Experimental design . . . . .	17
3.0.2 Literature review . . . . .	18
3.1 OCRopus . . . . .	18
3.2 The dataset . . . . .	20
3.3 Data preprocessing . . . . .	21
3.4 Levenshtein distance . . . . .	22
3.5 Cosine similarity . . . . .	23
3.6 W-shingling . . . . .	25
3.7 Word frequency . . . . .	26
3.8 Pixel comparison . . . . .	27
3.9 Fuzzy hashing . . . . .	29
<b>4 Experimental design and results</b> . . . . .	<b>30</b>
4.1 Experiment setup . . . . .	30
4.2 Results . . . . .	37
4.2.1 Analysis of the dataset . . . . .	37
4.2.2 Analysis of the experiment data . . . . .	40

4.2.3	Document-to-image . . . . .	40
4.2.4	Image-to-image . . . . .	41
4.2.5	Pixel comparison . . . . .	42
4.2.6	Similarity score . . . . .	43
4.2.7	Result comparison . . . . .	43
4.2.8	Runtime . . . . .	43
<b>5</b>	<b>Discussion, Conclusion and Future work . . . . .</b>	<b>44</b>
5.1	Discussion . . . . .	44
5.2	Conclusion . . . . .	51
5.2.1	Theoretical implications . . . . .	52
5.2.2	Practical implications . . . . .	52
5.3	Future work . . . . .	53
	<b>Bibliography . . . . .</b>	<b>55</b>
<b>A</b>	<b>Appendix . . . . .</b>	<b>60</b>
A.1	Application . . . . .	60
A.2	Source code . . . . .	68
A.2.1	OCRMethods.java . . . . .	70
A.2.2	FileInfo.java . . . . .	87
A.2.3	OCRUI.java . . . . .	92
A.2.4	TestRun.java . . . . .	100
A.3	Extended Results . . . . .	103
A.3.1	Document to Image . . . . .	103
A.3.2	Image to Image . . . . .	103
A.3.3	Pixel comparison . . . . .	104
A.3.4	One-to-one comparison . . . . .	104
A.4	Dataset . . . . .	107
A.5	Additional information . . . . .	112
A.5.1	OCRopus testpage . . . . .	112
A.5.2	Transcript of Q&A session with Økokrim . . . . .	113

## List of Figures

1	Manual forensic process . . . . .	3
2	Simple conversion of a test document with FreeOCR . . . . .	5
3	Human interaction algorithm . . . . .	8
4	Algorithm for OCR accuracy . . . . .	9
5	ssdeep fuzzy hash comparison . . . . .	13
6	General overview of the methodology for the experiment . . . . .	17
7	Modularity of OCRopus . . . . .	19
8	OCRopus' components . . . . .	20
9	Vector comparison for "aa aa" and "aa bb" . . . . .	23
10	Vector comparison for "aa" and "aa aa aa aa aa aa" . . . . .	24
11	Vector comparison for "aa aa" and "bb bb" . . . . .	24
12	Intersection of two sets. . . . .	25
13	Union of two sets. . . . .	26
14	Sampling of picture . . . . .	27
15	Preliminary pixel results . . . . .	28
16	One-to-many comparison . . . . .	32
17	One-to-one comparison . . . . .	33
18	Noise reduction . . . . .	33
19	OCR output cleanup . . . . .	34
20	Single thread OCR conversion . . . . .	35
21	Multi thread OCR conversion . . . . .	35
22	Tool operation . . . . .	36
23	The most recurring fonts in the dataset. . . . .	37
24	The most recurring font sizes in the dataset. . . . .	38
25	The most frequently recurring combination of fonts and sizes in the dataset. . . . .	39
26	Occurrences of different types of characteristics present in the dataset. . . . .	39
27	Matching accuracy, Document-to-Image . . . . .	40
28	Matching accuracy, Image-to-Image . . . . .	41
29	Pixel comparison score . . . . .	42
30	Embedded picture . . . . .	46
31	OCR noise . . . . .	47
32	The start screen for the application. . . . .	61
33	Choosing document folder. . . . .	62
34	Choosing document folder. . . . .	63
35	Optional keyword sort, with feedback when sorting is done. . . . .	64
36	Choosing a single picture for matching. . . . .	65

37	Choosing multiple pictures for matching. . . . .	66
38	Tool, analysis screen . . . . .	67
39	All the libraries used in the application. . . . .	69
40	OCROPUS test page . . . . .	112



## List of Tables

1	Specification for the virtual machine . . . . .	30
2	Similarity score, without text sanitation. . . . .	43
3	Similarity score, with text sanitation. . . . .	43
4	Accuracies of all texts comparison methods for all combinations of settings. . . . .	43
5	Runtime for the experiments . . . . .	43
6	Results without text sanitation, document to image . . . . .	103
7	Results with text sanitation, document to image . . . . .	103
8	Results with text sanitation, image to image . . . . .	103
9	Results with text sanitation, image to image . . . . .	103
10	Results for the pixel comparison method . . . . .	104
11	Levenshtein corpora score . . . . .	104
12	Most occurring fonts . . . . .	107
13	Most occurring font sizes . . . . .	108
14	Other characteristics . . . . .	108
15	Fonts and sizes . . . . .	109

# 1 Introduction

In this chapter an introduction is given on the subject and the challenges faced in this field. It continues with the research questions, then justification, motivation and importance, and finally the scientific contribution.

## 1.1 Keywords

Digital forensics, digital evidence, physical evidence, evidence linkage, approximate string matching, OCR.

## 1.2 Covered topics

The scope of the thesis is within the field of digital forensics. The main goal is to find methods to increase the efficiency and effectiveness of linking the digitized version of physical evidence found at a crime scene to digital evidence stored on a computer or other digital devices.

In investigations today, it can be a problem with overwhelming amounts of electronically stored information (ESI). And, a way to make the job of correlating digital evidence easier for investigators would be a welcome addition [1].

The thesis covers many areas, such as text search (string search), optical character recognition (OCR, which is a method to transform text from a scanned document/picture into text) and edit distance (approximate string matching) which is a method for comparing the content difference between two strings.

## 1.3 Research questions

In this thesis the following research questions are defined:

- What is an acceptable error rate for OCR in a forensics setting?
- What are characteristics of a realistic dataset in forensic research, development and testing?
- Which of the characteristics are significant and enable evidence linking?
- In which way is it possible to increase efficiency and effectiveness in evidence linking?

## 1.4 Justification, Motivation and Benefits

The methods applied here would help the people investigating cases to do their work more efficiently. Less time would be used for the linking and processing of documents, and an effect of this could be a shorter time from start of investigations until the case goes to court. With some countries putting a time limit on investigations [2] and changing priority of cases [3], the use of electronically stored information (ESI) and the management of this needs to take less time [4]. Figure 1 shows a normal forensics process where the investigator has found a physical document and manually has to search through large amounts of data using a keyword search A.5.2 [5], and manually inspect each document matching the search criteria. Depending on the search criteria and methods used this can be quite time consuming. A survey by Symantec [1], across ten countries in Europe, Middle East and Africa, asked lawyers about the effect of ESI on their legal proceedings.

All of the 5000 lawyers surveyed, confessed to:

"losing a case or legal matter in the past two years due to limitations in the technology used to process Electronically Stored Information" [1].

Furthermore:

"60 percent of respondents admitted they struggled with the amount of information that had to be searched; 29 percent complained that they did not have enough time to conduct thorough investigations; and 24 percent said they lacked sufficiently sophisticated e-discovery technology to fulfill requests effectively.

When asked how this might be alleviated, 57 percent specifically called for "improvements to search technology used to identify, preserve and process ESI" [1].

However, 98% of the lawyers said that digital evidence identified during the investigation had been vital to the success of legal matters. And 91% saying ESI are important or critical in their daily work.

Benefits for finding better matching methods are that the investigators could focus more on finding the digital evidence and solving the crime, instead of being slowed down by bad applications and lack of proper tools for the job. The authors of [4] discuss the time consumption tied to management issues, and the lack of proper tools for analyzing digital evidence, resulting in a situation where evidence end up being put on a shelf for weeks or months before being reviewed.

## 1.5 Limitations

The thesis does not measure the OCR accuracy of OCRopus, neither does it compare OCRopus to another OCR engine. The dataset contains no spreadsheets (Excel files) or presentations (Powerpoint files), neither does it contain files from Autocad® or other specialized software.

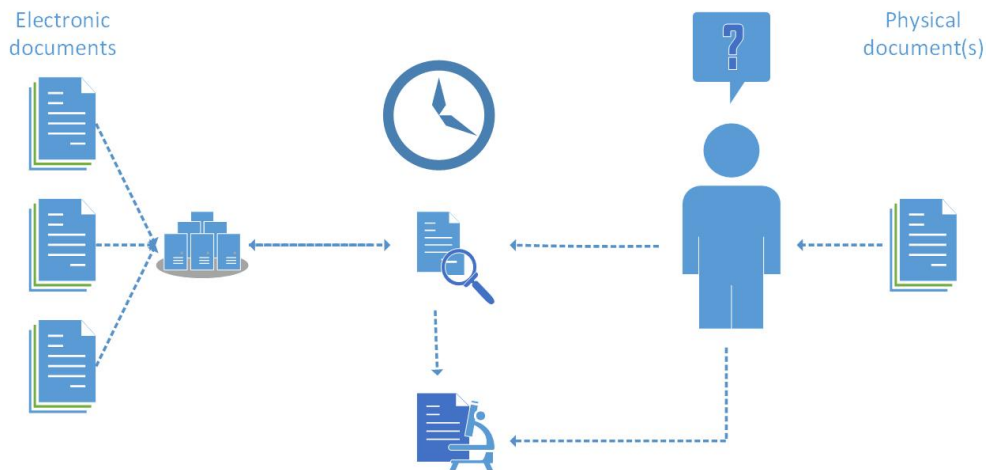


Figure 1: Manual forensic process, where the investigator finds the physical document, reads it, search for documents containing keywords from the evidence and manually checks all the matches.

## 1.6 Contribution

The thesis seeks to find methods for increasing efficiency and accuracy when dealing with the linkage of digitized physical evidence and electronic evidence. This will be done through the implementation of different methods in a proof-of-concept tool for use in the experiment. The proof-of-concept tool contains five different comparison methods, and is able to link documents with a 100% accuracy for some methods. It is used in this thesis to measure which method is most accurate in linking the evidence. By giving each of the methods implemented an accuracy score, a recommendation on which methods to use, and what characteristics enables evidence linking are answered. This could be valuable information for software developers developing forensic tools and might give an idea for new features to implement in an already existing tool.

A test dataset was created, based on descriptions from Økokrim on which type of documents they most frequently encounter. The dataset consists of 100 documents with a wide variety of characteristics, ranging from bold characters, colors, tables to full size images. All in the Microsoft Word<sup>®</sup> format, DOCX and PDF.

## 1.7 Thesis Structure

In Chapter 2 an overview of the research and work done in the field of the thesis is given. Chapter 3 describes the different methodologies used in the experiments and the expected outcome. Chapter 4 describes the experiment setup, how it works and the input for the proof-of-concept tool, as well as the results from the experiments. Lastly, Chapter 5 contains the discussion, the theoretical and practical implications of the results, the conclusion and suggested future work.

## 2 Related work

This chapter covers relevant and related work done in the areas important for this thesis. Each of the different aspects gets its own paragraph with research being done and a description of how it is used, to give the reader an overview of the field.

### 2.1 Digital Forensics and Digital Evidence

Since the terms Digital Forensics and Digital Evidence are used in thesis is important to define what these terms mean. Below are two definitions for these terms.

**Digital Forensics** is defined in [6][page 16] as:

"The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations"

**Digital Evidence** is defined in [7][page 7] as:

*"any data stored or transmitted using a computer that support or refute a theory of how an offense occurred or that address critical elements of the offense such as intent or alibi"*

Where data refers to information in various formats, such as; text, images, audio, and video.

### 2.2 Optical Character Recognition

Optical character recognition is a technology that allows us to convert text from different types of documents, such as images and scanned documents. The source for these documents can be books [8]<sup>1</sup>, newspapers [9], handwriting [10] and research papers or other documents [11][12]. Figure 2 presents a simple conversion of the document on the left side to the text on the right side.

---

<sup>1</sup><http://books.google.com/googlebooks/about/>

<sup>2</sup>[www.freeocr.net](http://www.freeocr.net)

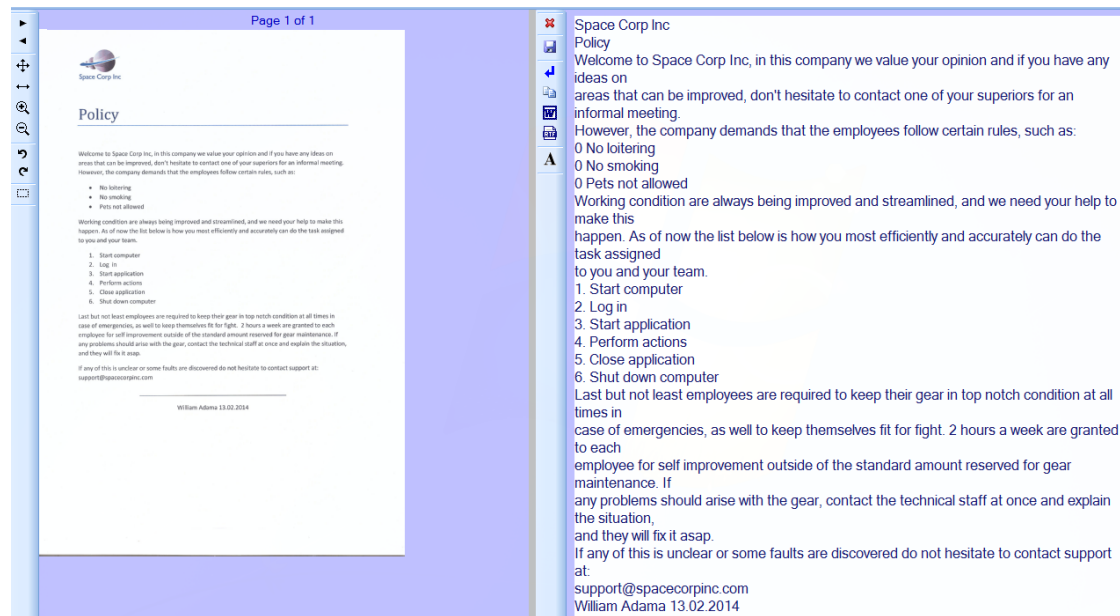


Figure 2: Simple conversion of a test document with FreeOCR <sup>2</sup>

In OCR, three main phases are used; pre-processing, character recognition, and post-processing. The subtasks for each main phase are given below [16][17]:

### Pre-processing

- Detect resolution, rescale
- Binarization
- De-skewing and denoising
- Page layout analysis/Segmentation/zoning
- Detection of lines and words

### Character recognition

- Feature extraction
- Classification based on different algorithms

### Post-processing

- Dictionaries

OCR is used in many different areas, such as the Google book project as already mentioned, but also, automatic number plate recognition [13] and to read and check passport information [14].

The authors of [15] states:

"In general, these applications tend to have two characteristics: (1) they are able to parse individual symbols from images of documents and (2) they can classify these images to reproduce text output. To be useful, OCR software must be both accurate and fast."

The authors of [18] explains how to establish confidence in digital forensics and evidence found by the utilized methods. Investigators might be confident that the tools and methods they use will produce a reliable result, but they might struggle to establish the same confidence on a scientific basis. In some fields of forensic sciences an error rate is used to describe the probability of false negatives, false positives and other inaccuracies when comparing two samples, but for digital forensics the use of statistical error rates can be misleading or inappropriate [18].

One of the main differences between digital forensics and other forensics disciplines is that digital forensics tries to find multiple artifacts that can tie a suspect to certain actions. Many of the other disciplines try to match two samples to establish if it is from the same source or the same substance, or identify the substance [19]. For instance when comparing DNA and trying to match these, one would have a statistical error rate, but in digital forensics, the methods to get to the evidence is often a series of methods and tools which all might introduce a small chance of systematical error. A systematical error is an error that can be introduced through improper implementation of algorithms or inherent errors in the algorithm itself etc., as opposed to random errors which are based on the natural process, and the inability to perfectly measure them.

To mitigate some of these errors in digital forensics, the authors pose three questions:

1. Are the techniques (e.g., hashing algorithms or string searching) used to process the evidence valid science?
2. Are the implementations of the techniques (e.g., software or hardware tools) correct and appropriate for the environment where they are used?
3. Are the results of the tools interpreted correctly? [18]

In question 1 it is assumed the authors by "valid science" mean scientifically sound and reliable.

The solution to these questions can be many, such as; tool testing, dual tool verification, training, guidelines and documentation are used to mitigate errors. One of the criteria used in the Daubert Standard <sup>3</sup> for admittance of evidence is the known or potential error rate [20]. However the other criteria state the evidence can be emitted if the technique or theory can and has been tested, if it has been peer reviewed, there are standards controlling the operation and if it has acceptance in the scientific community.

---

<sup>3</sup>Standard for admittance of digital evidence based on scientific principles.

The authors of [21] managed to get accuracy ratings of 99% with a voting system between different OCR engines, this is also supported by the paper [22] stating it is a simple and very effective method. The accuracy achieved meets the minimum 99% requirement set forth by the Meeting of the Experts on Digital Preservation [23] and should give us an indication on what percentage level is achievable and a baseline for future comparison.

One trend that is recurring in papers on this subject is the use of voting ensembles, where the classification is done with different classifiers and the output from each of these are voted on. The classifiers used can be k-nearest neighbors (kNN) and Bayes nets, and not necessarily about combining complete OCR systems for voting. The ensemble option is described and utilized with success in [21][22][24][25][26] and [27] where the general consensus is that the accuracy increases when using voting ensembles. In Figure 3 an algorithm with an ensemble voting is presented, the output from different OCR engines are compared. If they are similar they are accepted, if they are different they are sent of to an error estimator. This estimator is based on a artificial neural network, where a "suspect" word is calculated and sent of to a clustering phase where the average of the cluster is presented to a human operator for verification. In Figure 4 an algorithm for ensemble voting between different classifiers are shown. After the preprocessing the OCR conversion is done with neutral networks and kNN, and the accuracy achieved with this system were 99.3%.



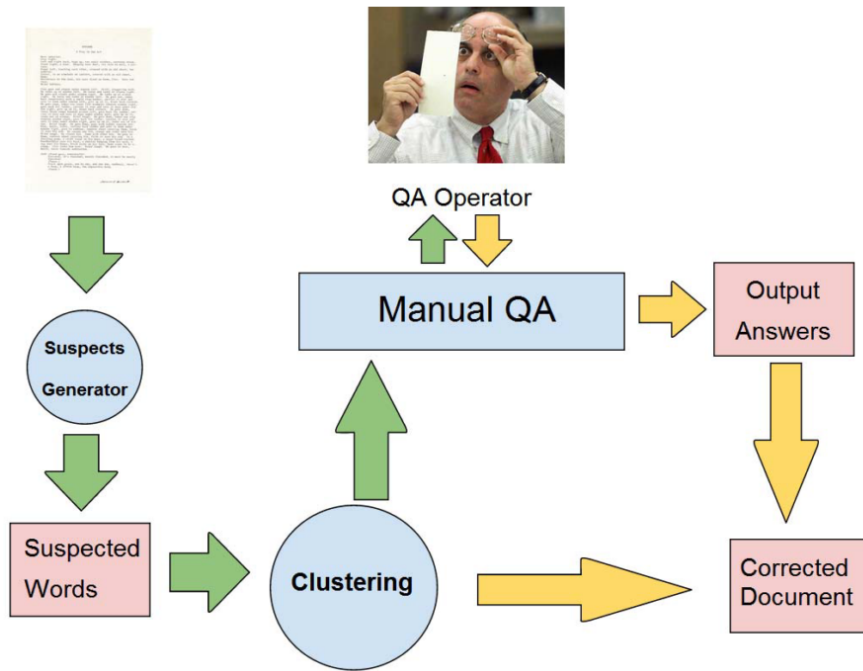


Figure 3: Proposed algorithm for increasing OCR accuracy, by using ensembles and human interaction. Picture from [21].

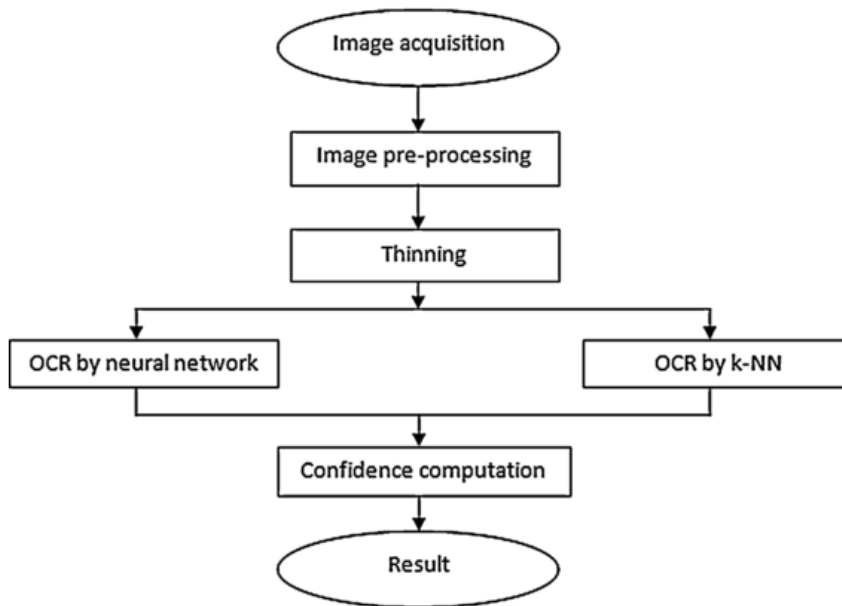


Figure 4: Algorithm for increasing OCR accuracy using voting ensemble. Picture from [24]

To further improve OCR results, the authors of the article [9] study how OCR software performs on newspapers. They determine factors that effect OCR accuracy, how to measure this and how to increase the accuracy. Even though the paper addresses the specific use for newspapers the concepts are representative for other documents as well. In the experiment, they use the already built in dictionary in the OCR software and then added their own specialized dictionaries with local names etc. The best results were obtained by utilizing the built in dictionary, second, no dictionaries and third, both dictionaries. However, it seems the experiment with both dictionaries obtained a poor result, because of an implementation fault, since others managed to get a better accuracy with both dictionaries. These secondary dictionaries were specialized as in geographic location names, which would not normally be found in a dictionary. For the default dictionary they received results ranging from 71%-92.02%. They did not agree upon what was an acceptable threshold, but for historic newspapers the consensus was that the categories: good, average and bad OCR should be used. Defined as:

Good OCR accuracy = 98-99% accurate (1-2% of OCR incorrect)

Average OCR accuracy = 90-98 % accurate (2-10% of OCR incorrect)

Poor OCR accuracy = below 90 % accurate (more than 10% of OCR incorrect) [9]

The methods used to try to increase accuracy were dictionaries, greyscale vs. bi-tonal files, image optimization software and correcting OCR text manually and using a confusion matrix and language modeling, this was not tested. The conclusion is that the best method for increasing accuracy is to manually correct the mistakes of the machine, in the paper this was done through

crowd sourcing, however this is too time consuming. It is also understood from the paper that specialized dictionaries can increase the accuracy when OCR is used on documents that contain a lot of words and abbreviations not found in a normal dictionary.

With 99% accuracy, this still amounts to 30 errors on a page of 3000 characters, which must be corrected by a human operator, or the documents are rejected and operator entry is used instead [28]. When humans are needed to check the output, the authors in the paper [21] have a solution where they managed to decrease the cost in time of using humans with 50% and 83% depending on the quality of the documents. In Figure 3 this algorithm is shown, the misclassified words are presented to the human for manual correction.

One of the huge problems with OCR is the use of different alphabets. The authors of [29] used OCR on Bulgarian documents written in Cyrillic with some Latin symbols in the text. A small subset of German documents were used as well for comparison. The main problem here is that some of the Cyrillic letters and words get recognized as Latin letters and words, decreasing the accuracy drastically. By using two dictionaries they managed to increase the accuracy of these documents with mixed languages, but not to the same accuracy score achievable from a single language/alphabet document.

The authors of [30] try to combat what they call alphabet confusion error, when one characters in one alphabet gets recognized as a character from another, by adding more dictionaries and using Levenshtein distance to find the best candidates, which were then used as input into algorithms to improve the accuracy by taking symbol confusion statistic and sentence context into account. These post-processing methods greatly decreased the alphabet confusion error and normal symbol recognition error. By using a dictionary tailored to the language we use OCR on, and another one to help for optimal character recognition (like æ,ø,å or Cyrillic characters), the accuracy will increase.

The authors of [28] identifies four potential sources for improvement.

- Improved image processing;
- Adaptation of the classifier to the current document;
- Multi-character recognition;
- Increased use of context. [28]

**Image processing:**

In this category they discuss degraded copies, which make the OCR process more difficult. Improved image processing can help alleviate errors on the defects originating from old copiers, printers and tabloid presses etc. Techniques discussed are better binarization and de-skewing.

**Adaptation:**

The authors here discuss how the OCR can adapt to the document where for instance the document is essentially made up of just one font. A single-font classifier is more accurate and faster than a multi-font classifier. One reason for this is that a multi-font classifier needs to distinguish between the letter O and the number 0 in many different fonts which might lead to errors. This can be avoided by automatically identifying the font or adapting the classifier parameters, such as choosing only the appropriate characters in the current font. Font identification requires a lot of data on every font, so the authors believe a more promising path is classifier adaptation.

To further expand on this, the authors discuss training the classifier on the most common documents, which is also supported by [31], where the OCR system learns character models directly from the document. They identify words that they have high confidence in, have been recognized correctly, and use these words for training of the classifier, resulting in an error reduction of 34.1% in character errors.

**Multi-character recognition:**

Instead of recognizing single characters it might be better to recognize images of larger units such as complete words. Half of all the words in a normal English text consist of common words<sup>4</sup> such as; "the, a, an, to, from". When these words are discovered they can be used for training the classifier (word recognition leads to adaptation) to recognize the shape of those specific letters in those words. This in turn helps to recognize the remaining words. One problem with this is when identical glyphs are used for the same symbols, for instance O and 0 or 1 and l. To solve this, context has to be taken into account.

**Linguistic context:**

Here the authors discuss OCR systems choosing n-grams like 'ing' over 'lng' or for dictionaries 'bolt' rather than 'holt'. By using word frequencies we could improve recognition. In addition, they also discuss specialized dictionaries. But, to bring this to a new level, OCR needs to utilize semantics and syntax. The example the authors give here is:

"For instance, "leek tar fail, tar lent", while lexically acceptable, is clearly ungrammatical." [28]

To avoid meaningless yet grammatically correct sentences like the one above, the OCR system can use semantic analysis. However, the efficiency of this is based upon that the text in question does not have nonsensical words, ungrammatical phrases, sentences that defy semantic interpretation and unexpected constructs, which is more common in technical material than a novel. This high-level linguistic context is something humans use as well when presented with an unfamiliar handwriting [28].

The authors believe a combination of multiple techniques such as those mentioned above is worth pursuing for improvement of OCR.

---

<sup>4</sup>Called stop words in information retrieval

The authors of [27] compare different methods for character recognition and problems still in the field. There are several classification methods, and the paper divides them into four categories; Statistical methods, Artificial Neural networks, Kernel methods and multiple classifier combination. Some of the methods discussed are k-nearest neighbor (kNN), artificial neural networks (ANNs) and support vector machines (SVMs). In the performance comparison between the different categories, the authors stress that it is difficult to get a fair comparison because many classifiers can be tweaked and thus their performance can be affected by human factors. The number of steps involved in character recognition (pre-processing, feature extraction, classification) does not make it any easier. For further testing, the authors recommend using open source code for every processing step so researchers can compare each step fairly.

### 2.3 Hashing and Fuzzy hashing

Prevalent hashing such as MD5, is used to identify two identical files and create bad and good hash sets for use in digital forensics. These sets are used to exclude files from a disk image which are not important to the investigation. An example of this can be files belonging to the operating system or known software packages such as Microsoft Office<sup>®</sup> (as long as the files are not modified). For bad hash sets, known malware is hashed and the value is compared to the files in the disk image to potentially identify malware, backdoors, cracker tools etc. present on the compromised system.

As opposed to normal hashing, fuzzy hashing is used to check if two files are similar and not identical. Normal hash algorithms are not suited, because the hash value from these two similar files will be vastly different. Instead the author of [32] introduces a new technique by combining traditional hashes to construct a new algorithm that can be used to identify similar files, like modified versions of a document with inserted, deleted and substituted material. This hash does not adhere to the avalanche effect<sup>5</sup> [32][33], for the whole output, so similar files would create similar hashes, which can be compared and given a similarity score.

"Context triggered piecewise hashing is a powerful new method for computer forensics. It will enable examiners to associate files that previously would have been lost in vast quantities of data that now make up an investigation. By creating associations between files that are homologous but not identical, investigators will be able to quickly find relevant pieces of material in new investigations." [32]

To test this, the author made an application named ssdeep, containing the fuzzy hash algorithm, and a document containing parts of the Gettysburg Address by Abraham Lincoln. The hash signature for this document was recorded, and the document changed by editing the font and font size, two paragraphs were added at the beginning of the document, one of them in a different color and other insertions and deletions. At the end of the document "I AM THE LIZARD KING" was added. In Figure 5 the comparison between the two documents is presented, resulting in a 57% match when run through ssdeep.

---

<sup>5</sup>A small change in the plaintext or the key, will result in a huge change in the cipher text

```
$ ssdeep -b gettysburg-address.doc > sigs.txt
$ ssdeep -bm sigs.txt gettysburg-modified.doc
gettysburg-modified.doc matches gettysburg-address.doc (57)
```

Figure 5: Result from comparing two similar files in the application ssdeep. Picture from [32].

## 2.4 Dataset and characteristics

The authors of [34] discuss the importance of proper datasets for OCR testing, and they mention three types of datasets, offline handwriting, online handwriting and machine printed text. Of these three, only the latter one is of interest to this thesis. The dataset in question i.e. the CDROM data set developed was by the University of Washington [35]. The authors states such a dataset should include documents in each of the world's major languages and scripts. However, they concentrated on the English language and Roman script. For document types they propose:

- Articles: journals, proceedings, books, etc.;
- Business letters and memorandums;
- Newspapers/magazines;
- Maps: street maps: terrain maps, etc.;
- Forms;
- Manuscripts;
- Engineering CAD/CAM drawings;
- Advertisements. [35]

For each of these types of documents, several instances are needed in various formats and quality. To describe each document they propose to include language, script, font information, zone definition (size and location of zones) and ground truth for each document. And lastly, degradation models to simulate coffee stains and degradation which comes from the use of photocopiers etc.

The authors of [36] discuss the need for a forensics corpora, and the creation of such data sets for a fair comparison of forensics tools. They have created several data sets such as; disk images, memory images, network packets and files available for download [37]. However they are not alone in making such datasets. The CFReDS Project [38] and the Forensics Wiki [39] have several datasets available for diverse purposes such as string search, memory images and filecarving etc. For email the Enron dataset is often used [40][41]. This dataset contains the emails of 150 former Enron employees all in text form with headers, subject field etc. and no attachments. Common characteristics in forensics datasets are the need for them to be documented, i.e. the most important features of the datasets need to be documented. The amount of documentation required depends on the dataset. In an investigation scenario of an image, documentation of the hash value of the image and where the evidence is located might suffice, but in a string search dataset an exact disk address for each string is needed.

## 2.5 Text comparison and evidence linking

To clarify what is meant by evidence linking the author pose two scenarios; One where the goal is to link two documents together. Here, characteristics such as content, hash values etc. are important. For the second scenario; The goal is to link a document to a person or a user. Metadata such as timestamps and access rights are important. In this thesis, we look at scenario one, and scenario two can be a continuation, when the document has been located. However, a mix between the two is used today by law enforcement, through the use of digital forensics tools, such as EnCase, The Sleuth Kit, AccessData Forensics Toolkit and X-Ways, which all offer string matching, i.e. keyword search as a method for locating files [42].

### **Text retrieval:**

Correcting misspelled words has been a problem for a long time, and perhaps the oldest potential application for approximate string matching, dating at least back to 1928 [43]. Approximate string matching has been used since the sixties to deal with the problem and 80% of the errors could be corrected with just one operation [44]. In this thesis, the approximate string matching is used in information retrieval, i.e. finding relevant information in a text, where string matching is one of the basic tools.

For string matching there are several available options. One of them is edit distance, i.e. the minimum number of edit operations needed to transform one string into the other [45]. This is often called the Levenshtein distance, but there are others as well, such as Damerau edit distance. To be able to explain the different operations properly some definitions are needed. A sequence of operations is represented on the form  $S(x,y) = t$ , where  $x$  and  $y$  are different strings, and  $t$  is a positive integer which represent the total cost of the operations [46]. Characters are denoted as  $a$  and  $b$ . The amount of changes (cost) can then be used to calculate the distance between two strings.

Edit operation in use are:

- Insertions:  $S(x,a)$  inserting the letter  $a$ .
- Deletion:  $S(a,x)$  deleting the letter  $a$ .
- Substitution:  $S(a,b)$  substituting  $a$  by  $b$ .
- Transposition:  $S(ab,ba)$  swap the adjacent letters  $a$  and  $b$ .

Some popular distances used are:

**Levenshtein distance:**

The edit operations in Levenshtein distance are; insertion, deletion and substitution. Each edit operation of a single character usually has a cost of 1 [47].

**Damerau edit distance:**

Identical to Levenshtein but adds one additional operation of transposing two adjacent characters. These characters must be adjacent both before and after the operation [44].

**Term frequency:**

Term frequency is the number of times a certain word occurs in a text and it can be used to compare documents. In [48] the term frequency was used to classify different documents into categories based on the most recurring word in each document. They utilized the "bag-of-words" representation for documents where each word is treated as a feature, and they observed that term frequency was a superior method for smaller feature sets. In [49] they use word frequency to compare two corpora, using keywords to differentiate one corpora from another. For each of the corpora a word frequency list is created and the two word frequency lists are compared and given a score.

**Jaccard coefficient:**

This index is used to compare the similarity between two sets. The Jaccard coefficient can be used in information retrieval such as web search. Here, the query word is compared to the index or the keywords in a document and the best matching result is returned. In [50], the authors use the Jaccard coefficient in combination with other methods to compare the similarity between keywords, and it got results close to a 100%. In [51], they use Jaccard coefficient to measure the similarity for a word and the misspelled version of that word, by comparing each letter of the word. Here, each letter can switch position and still be identified as the same word. The consensus is that the Jaccard coefficient is suitable for word similarity measurement.



**Cosine value:**

This measurement is used to compare the similarity between two strings, by calculation the Cosine value of an angle between two vectors, a measurement of how similar they are is given. In [52] the authors used cosine similarity as a baseline for comparing semantic similarity of texts in two corpora. Another use for cosine similarity is shown in [53], where the authors use cosine similarity to detect obfuscated malware. They compare the malware by extracting the instruction frequency count for each malware sample and run these two vectors through the calculation.

**W-shingles:**

This measurement divides text up into sets of unique "shingles" (a set of subsequences of tokens) and calculated their resemblance using set theory [54]. In [55] the author used Patricia tree<sup>6</sup> and W-shingles algorithms to identify plagiarized documents in a dataset. The dataset was generated by making one original dataset consisting of the 10 first HTML pages returned from 900 web searches. The plagiarized document dataset was created by taking passages from the same 10 return results and mix it up. The result of this comparison was a percentage value between the expected similarity and similarity calculated by the two algorithms. The best result obtained was from W-shingles with difference of 4.13%. The conclusion being that both algorithms can be used to detect plagiarism (similarity) in a given document, but that W-shingles yields a better result.

---

<sup>6</sup>A radix tree where any parent with only one child is merged with its parent.

### 3 Methodologies

This chapter describes the different methodologies used in this research for the research questions. A discussion about different datasets and their flaws and merits are given and whether they are applicable for this thesis and experiment. An overview of the methods is given, with a description of expected output and feasibility.

To answer the research questions:

- What is an acceptable error rate for OCR in a forensics setting?
- What are characteristics of a realistic dataset in forensic research, development and testing?
- Which of the characteristics are significant and enable evidence linking?
- In which way is it possible to increase efficiency and effectiveness in evidence linking?

The authors performed a literature study, to get a better understanding and to see what the general consensus in the field is. To produce results, the author used an experimental design, based around a created dataset and the developed proof-of-concept tool, containing the different text comparison methods, image comparison method and the OCR conversion by OCRopus. The general overview of the methodology for the experiment can be seen in Figure 6.

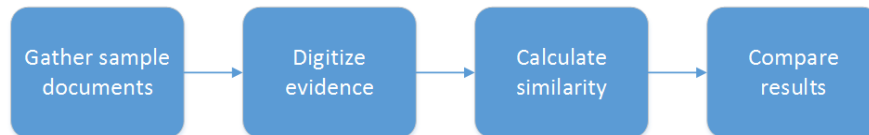


Figure 6: General overview of the methodology for the experiment, with its four phases.

#### 3.0.1 Experimental design

In this thesis we use experimental design in order to establish a cause-and-effect relationship, between the independent and dependent variables, where independent are the different methods, and dependent is the matching accuracy. By tuning the input for the methods we can determine if this affects the output i.e. the matching accuracy for each of the methods. Based on this, we are able to observe which configurations yields the best results. The authors of the book [56][page 226] state:

"Because we have not only observed but also *manipulated* the situation, we have used an experimental design"

This backs up our selection of this method, since we are testing many different configurations of variables in the experiments.

### 3.0.2 Literature review

We use a between-study literature analysis in this thesis, which comprise of contrasting and comparing information from two or more sources [57]. Information compared, are methods and results. But, the paper states that every component of a paper should be compared to every component from the other paper(s). However, not all of the components in each paper are of interest for this thesis. By using multiple sources we allow ourself to combine the information from various sources and increase the understanding of the field, hence, giving us the ability to better answer the research questions.

## 3.1 OCRopus

OCROpus was chosen because of its open source and ease of getting the output into another application. By using an open source application we can inspect the source code and get an understanding of what is going on, and/or modify the code if need be. This was not needed, but parts of the code were scrutinized for better understanding of some option triggers. The use of open source tool for testing is supported by [27] where they stress that tools cannot be compared fairly if the implementation cannot be looked at.

The OCROpus application is built using a nearest neighbor classifier, a standard multilayer perceptron, a support vector machine classifier and a Hidden Markov Model line recognizer [11][58]. The application also utilizes ensembles (classifier combination), language modeling for specific languages [58], which adhere to the methods for OCR described in Chapter 2. The language model is based on weighted finite state transducers (WFSTs), which can utilize n-grams, dictionaries and regular expressions. The best model however is a combination of a dictionary based model and a statistical character based model. The WFST models appear to be suited for multi-script and multi-language recognition as well [58]. Another important aspect of this is that they can be composed and used in a modular fashion to enable OCROpus to adapt quickly to new languages and document types (see Figure 7), which makes OCROpus quite versatile and increases its potential for adaptation to other languages than just English.

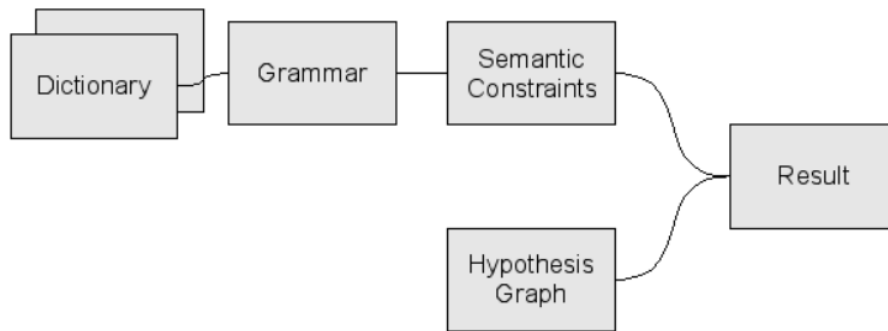


Figure 7: "Language models based on finite state transducers can be composed modularly from dictionaries, n-grams, grammatical patterns, and semantic patterns." Picture and text from [11].

OCROPUS has three main stages, see Figure 8.

- Physical layout analysis is responsible for identifying text columns, text blocks, text lines, and reading order.
- Text line recognition is responsible for recognizing the text contained within each line (note that lines can be vertical or right-to-left) and representing possible recognition alternatives as a hypothesis graph.
- Statistical language modeling integrates alternative recognition hypotheses with prior knowledge about language, vocabulary, grammar, and the domain of the document. [11]

The input to this application is the images files selected in the proof-of-concept tool, and the output from OCROPUS is processed by different methods described further down in this chapter.

OCROPUS does all of the usual OCR steps of preprocessing mentioned in 2.2 such as; cleanup, layout analysis, text-image segmentation, column finding, text line modeling etc. For a full view and in depth explanation of all of these steps, see paper [11].

Limitations as described on the homepage for OCROPUS [59].

- Performance on multi-column documents
- Performance on documents containing images

Another limiting factor is the runtime of OCROPUS, but this is discussed in Chapter 4.1

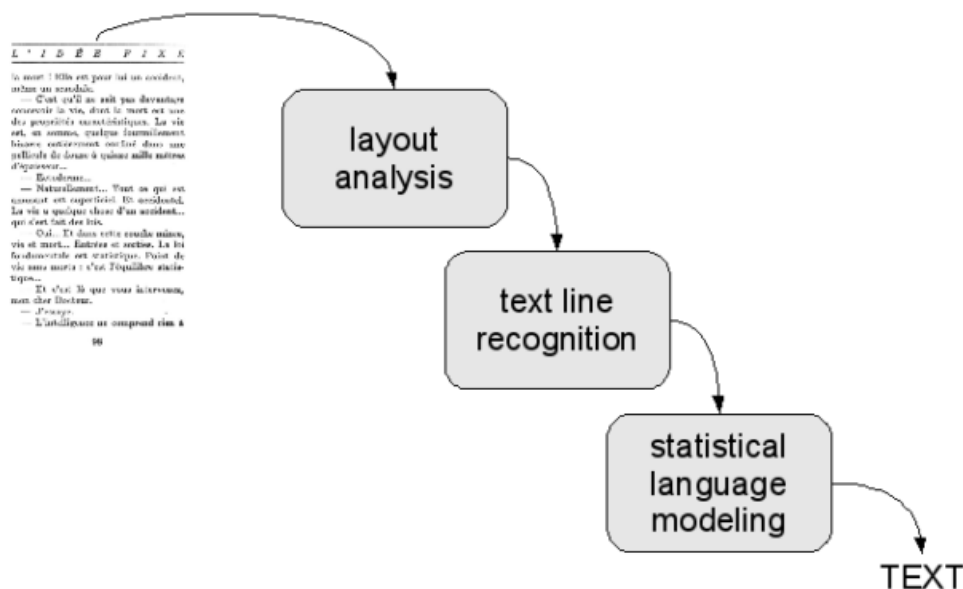


Figure 8: A rough flow diagram of OCRopus, which consists of three major components: layout analysis, text line recognition, and statistical language modeling. Picture from [11].

### 3.2 The dataset

There are several datasets available for testing, but none of them fulfills the requirements needed for this experiment. The considered datasets for the experiment are discussed in 2.4, but after discussions with Økokrim they have recommended focusing on Microsoft Office<sup>®</sup> file formats (DOC, DOCX) and PDF documents. Since these are the most common file types they encounter. Which makes sense since Microsoft<sup>®</sup> has a leading market share for office suites [60]. Moreover, most of these datasets are only text files, so they do not contain any tables or pictures etc. To get these constructs, the filetypes; DOCX and PDF is used to be able to include more than just text.

The dataset created for this experiment, includes documents such as; invoices, school tasks, business strategies, job postings etc., for a total of 100 documents. The reason for this low amount of documents are the runtime of the experiment, the dataset had to be in a feasible size to keep the run time down. To reflect the diversity of digital evidence the documents are not from any particular field. The data is gathered through a web search for DOCX documents. The documents have been screened and sanitized to ensure they do not contain any information that could identify individuals or companies. Any similarity to real companies or persons is purely coincidental. Some documents have been created by the author for the initial testing and are included in the dataset. The documents in the dataset contain tables, lists, figures, charts, images, different fonts and font sizes. The dataset contains no audio or video files.

The characteristics looked at in the dataset are numbers of documents with:

- Fonts
- Font sizes
- Most used font and sizes
- Graphs
- Pictures
- Tables
- Lists
- Bold
- Underline
- Italic
- Color (colored character or colored background)

### **3.3 Data preprocessing**

To ensure that all documents are compared in a fair way, they all need to be one page long, because of limitations in the proof-of-concept tool. It only supports one-to-one image comparison for pixels. However, we can choose to do OCR on a longer document. All the other methods will work, except that the pixel comparison will only compare the first scanned image and the first page of document.

All documents are converted to PDF and scanned. When the application is executed, the PDF files are converted to black and white images at 300 DPI as per the recommendation from [23]. This is done to reduce noise from different colors etc. The documents are scanned using the Microsoft Windows 8<sup>®</sup> built in scan application (Windows Fax and Scan) and a HP 4800 series scanner and the setting "Black & White" as described in [61]. For the conversion of the digital documents to images, the tool uses a library called PDFbox [62]. This library converts images to a Java BufferedImage of the type BINARY which is documented here [63] to give RGB values of [0,0,0] and [255,255,255] only, which in turn converts to the sRGB value of -1 and -16777216. These values are consistent for the converted documents, but some discrepancies are observed in the scanned documents.

### 3.4 Levenshtein distance

As stated in [47] the Levenshtein distance uses three operations; deletions, insertions and substitutions, to get the edit distance between two strings denoted as  $a$  and  $b$  in Equation 3.1[45].

$$\text{Lev}_{a,b}(i, j) = \min \begin{cases} \text{Lev}_{a,b}[i - 1, j] + \text{the cost of deleting } a_i \\ \text{Lev}_{a,b}[i, j - 1] + \text{the cost of inserting } b_j \\ \text{Lev}_{a,b}[i - 1, j - 1] + \text{the cost of replacing } a_i \text{ with } b_j \end{cases} \quad (3.1)$$

For example, the transformation of "color" to "colossus" gives a distance of 4, since it takes a minimum of four operations to do this.

1. color -> colos (substitution of "r" with "s")
2. colos -> coloss (insertion of "s")
3. coloss -> colossu (insertion of "u")
4. colossu -> colossus (insertion of "s")

The output from this algorithm is a number between 0 for a perfect match and is at most the length of the longest string for a total mismatch. Where the strings:

1. "aaaaa" and "aaaaa" yields a result of 0
2. "aaaaa" and "bbbbbbbbbb" yields a result of 10
3. "aaaaa" and "bababababa" yields a result of 5

Levenshtein distance was chosen because of its long track record in related research and it is easy to implement and understand.

### 3.5 Cosine similarity

The cosine similarity (See Equation 3.2[53], where  $x$  and  $y$  denote the two vectors) measures the angle between two vectors and gives an output between 0 and 1. Where 0 equals no similarity and 1 equals that the strings are similar (not identical). This method can never yield a negative result, since none of the frequencies can be negative. One major drawback with this method is that two vectors containing exactly the same terms but different frequency will yield a result of 1 since the angle between these two vectors will be zero, see Figure 10.

$$\text{Cos}(\theta) = \frac{\sum_{i=1}^n x_i \cdot y_i}{\sqrt{\sum_{i=1}^n x_i^2} \cdot \sqrt{\sum_{i=1}^n y_i^2}} \quad (3.2)$$

Examples:

1. "This is a test" and "This is a test" yields 1.0
2. "aa" and "aa aa aa aa aa aa" yields 1.0. See Figure 10.
3. "aa aa" and "bb bb" yields 0.0. See Figure 11.
4. "aa aa" and "aa bb" yields 0.7. See Figure 9.

Cosine similarity was chosen because of its ease of implementation and understanding, as well the promising results it shows for text comparison as seen in [53].

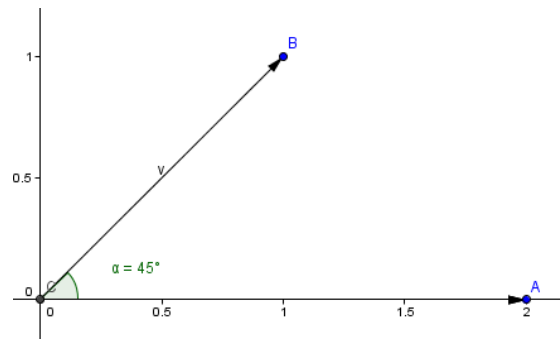


Figure 9: Vector comparison for "aa aa" and "aa bb" Where the cos value of  $45^\circ$  is 0.7





Figure 10: Vector comparison for "aa" and "aa aa aa aa aa aa" Where the cos value of  $0^\circ$  is 1.0

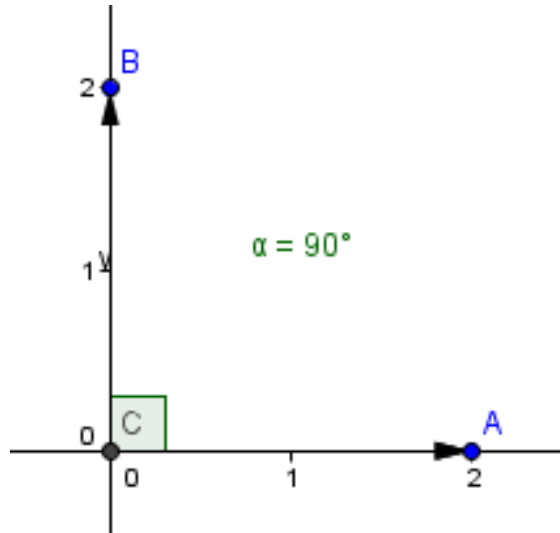


Figure 11: Vector comparison for "aa aa" and "bb bb" Where the cos value of  $90^\circ$  is 0.0

### 3.6 W-shingling

In this method the strings are divided up into sequences of tokens. These tokens can be lines, words, or letters, but they need to be countable. By associating a set of subsequences  $S$  of tokens to every document  $D$  one can create a bag of shingles (subsequences contained in  $D$ ) for each document or its  $W$ -shingling  $S(D, w)$  where  $w$  is the size of the shingles [54]. The 4-shingling of (an,iris,is,an,iris,is,an,iris) are; {(an,iris,is,an), (iris,is,an,iris), (is,an,iris,is), (an,iris,is,an), (iris,is,an,iris)} After these  $W$ -shingles are created for each document, they are compared using Equation 3.3 [50] [54], where  $A$  and  $B$  denotes two documents and  $r$  the resemblance score.

$$r_w(A, B) = \frac{|S(A, w) \cap S(B, w)|}{|S(A, w) \cup S(B, w)|} \quad (3.3)$$

The value returned is given by taking the intersection (see Figure 12) of the two sets divided by the union (see Figure 13) of the sets and is between 0 and 1. A 0 means no similarity and a 1 means identical. This equation is also called the Jaccard coefficient [50].

Example output with a  $w$  value of 3

1. "an iris is an iris is an iris" and "an iris is an iris is an iris" yields 1.0
2. "aaaaaa" and "bbbbbb" yields 0.0
3. "aa aa" and "bb bb" yields 0.0
4. "aa aa" and "aa bb" yields 0.7
5. "aaaaaa" and "aaabbb" yields 0.25

In the implementation used, the  $w$  is set to 3 and the strings are divided by characters and not words e.g. the string "an iris is an iris" gives {(an )(n i)( ir)(iri)(ris)}.  $W$ -shingles was chosen based on the recommendation by Økokrim (see A.5.2), its ease of understanding and implementation and promising results as seen in [55].

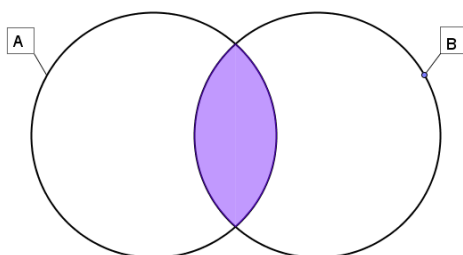


Figure 12: Intersection of two sets.

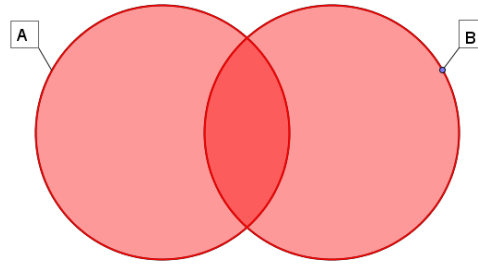


Figure 13: Union of two sets.

### 3.7 Word frequency

In this method, the word frequency of each document is calculated and compared. The word frequencies for document *A* are put into a vector, and the word frequencies for document *B* are put into another vector. To make these vectors comparable they need to contain the same information and same length, to ensure this the terms lacking from vector *A* are added to vector *B* with a frequency of 0 and vice versa. See below for the vector transformation.

$$\begin{aligned} A[a,5|b,3|c,1|d,2] &\rightarrow A[a,5|b,3|c,1|d,2|e,0|f,0] \\ B[a,3|e,2|f,7] &\rightarrow B[a,3|b,0|c,0|d,0|e,2|f,7] \end{aligned}$$

The two vectors are iterated through and the absolute value of the difference between the entries in the vectors are calculated. See Equation 3.4 where  $x$  denote vector *A* and  $y$  vector *B* and  $n$  the number of unique words.

$$\text{Score} = \sum_{i=1}^n |x_i - y_i| \quad (3.4)$$

The output is the sum of the difference in word frequencies. The output will range from 0 which means no difference to a number depending on the number of different terms in each document. For instance, a document *A* with [a,10] and *B* with [b,10] will yield 20 as output. One potential problem with this methods is that it does not take word order into account. For the binary strings 1001 and 0011, they would appear to be the same (assuming each bit is processed as a word), but the meaning/value of the two are highly different. It was chosen because it is a commonly used method for text comparison [48] [49], and it is easy to implement a version of this in the tool.

### 3.8 Pixel comparison

This method rescales the two input images to 300x300 pixels, where the RGB average value of 25 areas of 30x30 pixels each (see Figure 14) are calculated and put into two 25x3 feature vectors. The Euclidian distance between the regions in the vectors are calculated (see Equation 3.5) and accumulated and yield a result between 0 and 11041 where closer to 0 is more similar and closer to 11041 is more different.

$$\text{Score} = 25 * \sqrt{(r1 - r2)^2 + (g1 - g2)^2 + (b1 - b2)^2} \quad (3.5)$$

In Equation 3.5,  $r1$  and  $r2$  denotes the average red value from the two images,  $g1$  and  $g2$  the average green value,  $b1$  and  $b2$  the average blue value.

To minimize errors such as colors not being scanned correctly and converted correctly, all images are converted to black and white images. However, this will also remove all color information (uniqueness) that could make it easier for this algorithm to identify similar areas.



Figure 14: Sampling of picture for average pixel calculation. Picture from [64].

In Figure 15, a small comparison of some preliminary test data is presented. The first column is the picture converted from the original PDF document, the second column is the same picture compared with itself, the remaining columns are the scanned versions of said documents compared against the converted original. The algorithm identifies three out of four pictures correctly. The high accuracy result here is misleading since the input for the methods here are only the few images displayed in the figure, and the results here are not representative for the methods performance on the real dataset.

A problem with this method however is that in documents with a lot of uniformly distributed text/images, the pixel average of these documents will be very similar and can yield lower scores to documents where the content is different, which happens in row 1, column 3 in Figure 15 where another image has been ranked as the best match, but, one can easily see that these are not the same documents.

This pixel comparison method was chosen because it is fairly easy to both understand and modify. It is based on the code from [64]. By not using libraries for this comparison method, the author has full insight into the workings of the algorithm.










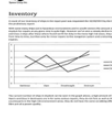














Original	Same picture	Scanned documents			
	 0.0	 169	 237	 294	 462
	 0.0	 204	 221	 316	 490
	 0.0	 150	 277	 303	 398
	 0.0	 330	 609	 753	 793

Figure 15: Preliminary results of a pixel comparison between different images/documents, using the implemented pixel-to-pixel algorithm.

### 3.9 Fuzzy hashing

Fuzzy hashing was considered as a method to be implemented in the proof-of-concept tool, but preliminary tests showed that documents scanned to PDF files, and run through the application ssdeep discussed in 2.3 yielded results below the built in matching threshold in ssdeep. We assume this has to do with the fact that the ssdeep application takes the files bit-for-bit. A PDF file created by a scanner software would be quite different from a PDF file created from Adobe Acrobat<sup>®</sup> or Microsoft Office<sup>®</sup>, even though the files both contain the same content. Since no matches could be found using this tool, the implementation of this method was dropped. Moreover, it would have added unnecessary complexity to the tool as more file operations and writing to the file system would be required.

## 4 Experimental design and results

This chapter contains the description of the experiment and how it is set up, the results from the conducted experiments and the characteristics of the dataset.

### 4.1 Experiment setup

The proof-of-concept tool is developed on a virtual machine hosted in VmWare running in Windows 7. The specifications and tools used for the experiment in the virtual machine are listed in Table 1. The reason for running the proof-of-concept tool in Ubuntu is the system requirement for OCRopus [59]. OCRopus utilizes python libraries and the Linux package manager 'apt' for installing the required dependencies. The Java language is the one the author is proficient in, and therefore the natural choice for development. The proof-of-concept tool is based on an early version developed by the author for the subject "Computational Forensics" at Gjøvik University College. It only implemented Levenshtein distance, and a one-to-one comparison of text at that time.

Table 1: Specification for the virtual machine

Hardware/software	Specification
CPU	i7-720QM 1.6Ghz(2.8Ghz)
RAM	4GB
OS	Ubuntu 12.04
IDE	Netbeans 7.3
Language	Java (JDK7)
OCR software	OCRopus 0.7

The input for the tool is the path to a folder containing the digital documents, an optional keyword to screen for, and file paths to the image files of the scanned document(s). For automatic testing of many documents the tool only accepts one page documents, for longer documents, the input has to be fed manually for each document. The output from the automatic tool in the experiment are several comma separated text files, containing the best matches for each document for each method. This data has to be examined and manually analyzed. The result of this analysis yields a score on how accurate each of the methods are to identify the correct document.

In Figure 22 the general and current workings of the tool is shown. As mentioned above, the input to the tool is a folder with digital documents, shown on the left hand side, and a folder with image files from the scanned documents shown on the right hand side. When the program executes, the digital documents are converted to image files and linked to the filename of the converted document, to keep track of the connection between the files. The scanned documents are converted to text by OCRopus and run through a regular expression method (see Figure 19) to remove the excess column of text seen in Figure 20 on the left hand side. Since it is only the text on the right hand side that is interesting. All the text (for both inputs) is converted to lower case, since this does not change the content of the texts. These two texts are then sent into the four different text comparison algorithms, while the pixel comparison gets the image from the converted digital document and the image of the scanned document as an input. The output and matching result are displayed in the tool (see Figure 38 in the appendix), but modifications can be done in the code to write it to file which was done for the experiment.

For the experiment, five different configurations were used:

1. Document to Image, without text sanitation.
2. Document to Image, with text sanitation.
3. Image to Image, without text sanitation.
4. Image to Image, with text sanitation.
5. Pixel to Pixel comparison.

Where "Document to Image" denote the comparison of extracted text of the documents through the use of Java libraries, and the OCR extracted text from the scanned document.

"Image to Image" denote the comparison between the OCR generated text from the converted image from the original document and the OCR generated text from the scanned document.

"Pixel to Pixel" denote the pixel comparison between the converted image from the original document and the image from the scanned documents.

Text sanitation is used for all methods except pixel-to-pixel comparison, where there are no text to sanitize. The text sanitation method is shown in Figure 18, and it removes all non-alphanumeric characters, single digits, single letters, and replaces double spaces with single spaces to counteract some of the noise generated by the OCR conversion.



All the text generated through OCR from the scanned documents are compared against all of the text extracted from the original document and its OCR generated text from the converted image, a one-to-many comparison. See Figure 16, where  $n$  denote the number of original documents, and  $m$  denote the number of scanned documents. Each scanned document are compared against all the original documents, as well as its converted image, using all five methods with and without text sanitation.

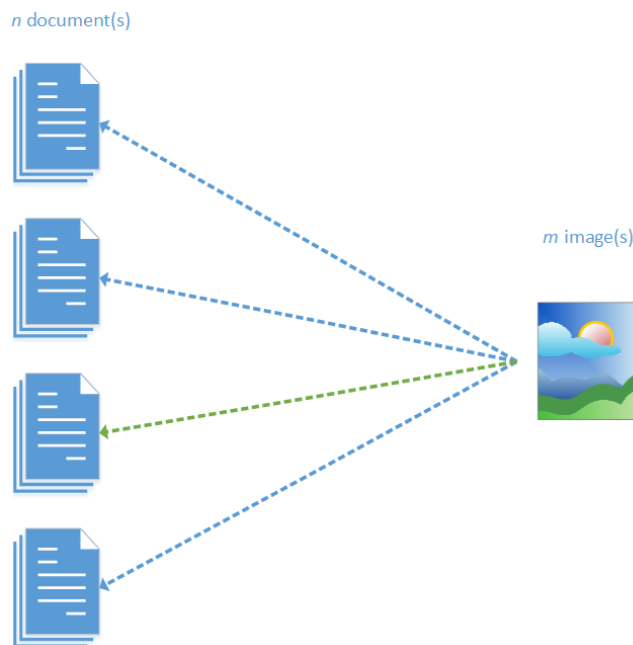


Figure 16: Each single image and OCR generated text are compared against all of the text from the original documents as well as the converted image. Green indicates the best match.

In addition, a one-to-one comparison between the documents with the same configuration is performed. The original document/converted document compared to its scanned version see Figure 17. Where  $n$  denote the number of documents/images. By doing this a score for the different configurations are given. This score reflects the potential viability of the methods used. The results from this run are used to calculate a score shown in Equation 4.1 where  $x$  denotes the Levenshtein distance between the two texts,  $y$  denotes the number of characters in the original, extracted text by either libraries or OCR from the converted image and  $n$  denotes the number of documents in the corpora. This results in a score where, closer to zero means less edits needed, i.e. the texts are more similar, and a score further from zero, the opposite, more dissimilarity.

$$\text{Score} = \frac{\sum_{i=1}^n \frac{x_i}{y_i}}{n} \quad (4.1)$$



Figure 17: Each single image and OCR generated text are compared against the original documents and its converted image.

```
//Methods for removing excess information in the strings from OCR input
private String stringCleaner(String input){
    //Removes everything that is not alphanumeric and spaces
    String newstr = input.replaceAll("[^A-Za-z0-9 ]", " ");
    //Removes single digits
    newstr= newstr.replaceAll("(?!\\S)\\d(?!\\S)", "");
    //Removes single letters
    newstr= newstr.replaceAll("(?!\\S)\\w(?!\\S)", "");
    //Replaces double space with a single space
    newstr = newstr.replaceAll("\\s{2,}", " ");
    return newstr;
}
```

Figure 18: Code for removing noise before the text comparison methods, through the use of regular expressions

```

//Used to sanitize the output from OCRopus
public String Regex(String input){
    try{
        String regex01 = "(....)(/)(\\d*)(/)(\\d*.*)(\\.)(...)"
            + "(\\.)(...)(\\s)(.)(\\s)";
        Pattern p = Pattern.compile("(....)(/)(\\d*)(/)(\\d*.*)"
            + "(\\.)(...)(\\.)(...)(\\s)(.)(\\s)");
        Matcher m = p.matcher(input);
        if (m.find()) {
            //System.out.println("MATCH FOUND");
            //System.out.println(m.group(0));
        }

        input = input.substring(input.indexOf(m.group()));
        input = input.replaceAll(regex01, "");
    } catch(Exception e){
        System.out.println("Failure in Regex: " + e);
    }
    return input;
}
}

```

Figure 19: Code for removing excess output from OCRopus, as seen in Figure 20

There are some limitations to the tool. Because of the libraries used, it can only do pixel comparison of PDF documents, that is, only PDF documents can be converted to images by the proof-of-concept tool. For all methods to work, all the test documents will need to be converted to the PDF file format. The author did not find any free to use libraries for conversion of the Microsoft Office<sup>®</sup> file format DOCX to images, but there are some libraries that can be purchased<sup>1</sup>. Implementing this would make the application capable of handling these types of file formats as well, and is something that is required if it is to be used.

One major drawback of the tool is that for single documents with multiple pages, it will not be able to do a pixel comparison because of limitations in the code and the libraries used to implement the functionality. It will run all of the text comparison algorithms, but the pixel comparison methods will only run on the first page of the document. With the use of other document-to-image converter libraries, this should not be a problem, and result in a solution were a one-to-one comparison for each of the pages and scanned pages of the same document are measured. And at the end take the average value of these numbers for a matching score. Another way to do this would be to divide the original document up into several one page documents and then run the application, however this is highly impractical and defeats the purpose of the application.

<sup>1</sup><http://www.qoppa.com/wordconvert/>

Another limiting factor is the total time it takes for OCRopus to yield a text output from a single page. The way OCRopus is implemented and configured in the proof-of-concept tool, the process takes close to three minutes per page, which is unfeasible if the dataset is huge. OCRopus can be configured for multi core/thread support. This support was enabled to reduce the conversion time of image to text, but this resulted in a garbled output as seen in Figure 20. Where the output from the single thread conversion is shown and in the red rectangle, with the image names in order. In Figure 21 the image names are not in an ascending order, which results in the output being in the wrong order compared to the original document (see Figure 40 in the appendix). If this is limited to just the virtual machine is unknown. No other mentioning of this problem was found in OCRopus' help section. This is a problem because of the implementation. The proof-of-concept tool extracts the text straight from the shell, and not from the created text files from OCRopus. By looking closely on Figures 20 and 21, one can observed that images with the same name contains the same text. This implication is discussed in future work 5.3.

```
temp/0001/010001 bin.png : BOOK REFIEH
temp/0001/010002 bin.png : A Mine of Information on Libraries
temp/0001/010003 bin.png : Jaideep Sharma
temp/0001/010004 bin.png : (Department of Library and Information Science, Kurukshetra University, Kurukshetra)
temp/0001/010005 bin.png : Mohamed Taher, 2002, Libraries in India's National Developmental Perspective: A Saga of Fifty Years
temp/0001/010006 bin.png : since Independence, New Delhi, Concept Publishing Co, 569 pp, Rs.900/-
temp/0001/010007 bin.png : The importance of libraries is evident in the oft-quoted
temp/0001/010008 bin.png : phrase, library is the heart of an institution'. It is charged
temp/0001/010009 bin.png : with the responsibility of storing, organising, disseminating,
temp/0001/01000a bin.png : preserving and passing on to future generations the cultural
temp/0001/01000b bin.png : and intellectual wealth of the society. It serves an important
temp/0001/01000c bin.png : function in teaching, research and satisfying other information
temp/0001/01000d bin.png : needs of the people. In spite of this we do not find a library
temp/0001/01000e bin.png : culture in our country, people waiting to get a seat in the
temp/0001/01000f bin.png : library, lectures, for story-telling sessions or exhibitions being
```

Figure 20: OCR conversion with a single thread, results in a correct order in the shell output. Also shown here on the left hand side is the excess information OCRopus creates.

```
temp/0001/010003 bin.png : Jaideep Sharma
temp/0001/010001 bin.png : BOOK REFIEH
temp/0001/010002 bin.png : A Mine of Information on Libraries
temp/0001/010006 bin.png : since Independence, New Delhi, Concept Publishing Co, 569 pp, Rs.900/-
temp/0001/010007 bin.png : The importance of libraries is evident in the oft-quoted
temp/0001/010004 bin.png : (Department of Library and Information Science, Kurukshetra University, Kurukshetra)
temp/0001/010005 bin.png : Mohamed Taher, 2002, Libraries in India's National Developmental Perspective: A Saga of Fifty Years
temp/0001/010008 bin.png : phrase, library is the heart of an institution'. It is charged
temp/0001/010009 bin.png : with the responsibility of storing, organising, disseminating,
temp/0001/01000a bin.png : preserving and passing on to future generations the cultural
temp/0001/01000b bin.png : and intellectual wealth of the society. It serves an important
temp/0001/01000c bin.png : function in teaching, research and satisfying other information
temp/0001/01000d bin.png : needs of the people. In spite of this we do not find a library
temp/0001/01000e bin.png : culture in our country, people waiting to get a seat in the
temp/0001/01000f bin.png : library, lectures, for story-telling sessions or exhibitions being
```

Figure 21: OCR conversion with four threads, results in a wrong order in the shell output.

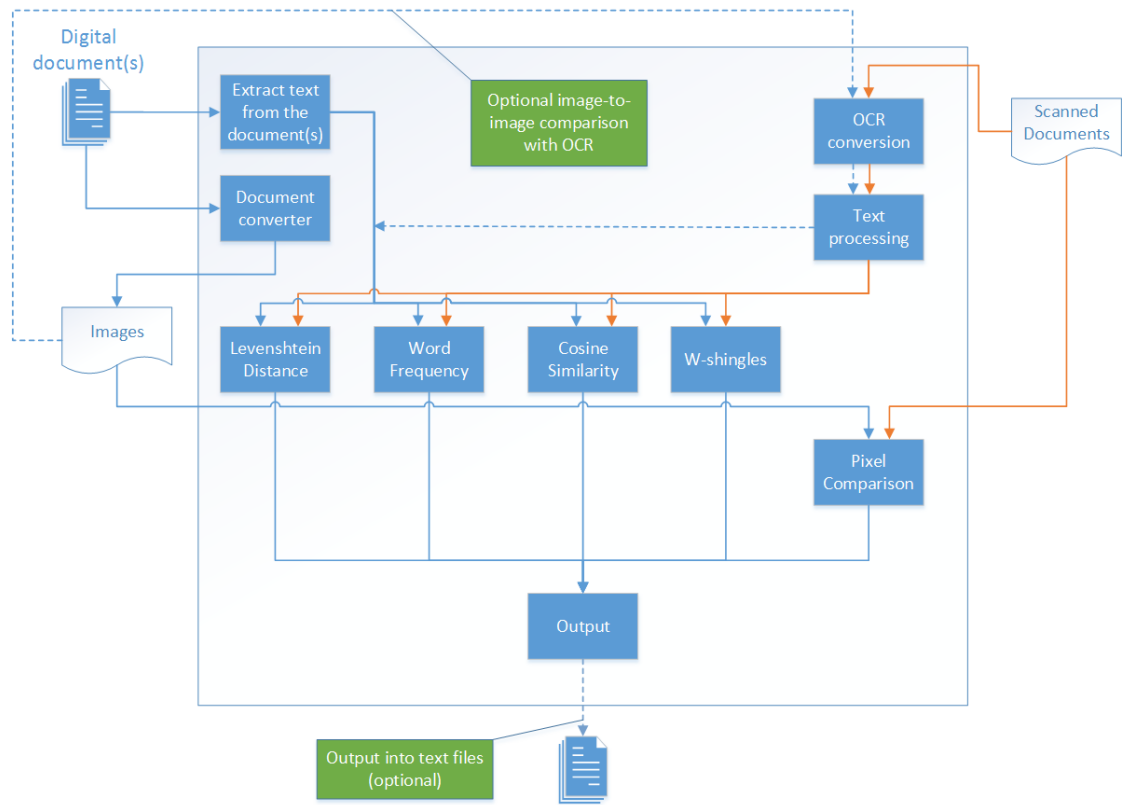


Figure 22: Flowchart of how the tool operates, objects outside of the square are files on the file system. Objects inside the square are operations in the application.

## 4.2 Results

In this section the obtained results are presented. Only the most frequently occurring characteristics from the dataset is presented. The full results are contained in the appendix A.4. The accuracy results for the methods are presented side-by-side for each configuration for an easier comparison.

### 4.2.1 Analysis of the dataset

The dataset consists of 100 documents of great variety, containing graphs, pictures and lists etc. Many of the documents contain several of these items, as well a combination of different font and font sizes.

The main characteristics for the created dataset are shown in the following figures. They include the occurrences of font type, font size, and the combinations of these two. As seen in Figure 23 the font Calibri is the one that occurs the most in the dataset, and it is used in 51 out of the 100 documents. See Table 13 in the Appendix for the full list of font occurrences.

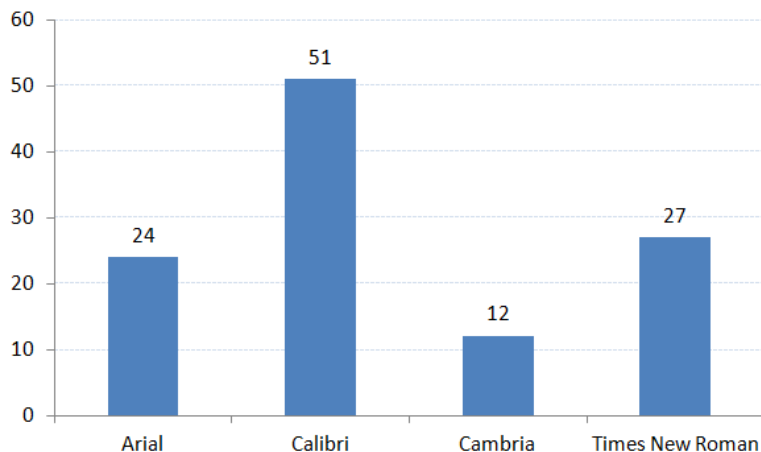


Figure 23: The most recurring fonts in the dataset.

The most frequently occurring font size in the dataset is 12pt. as seen in Figure 24, which is the default font size for Times New Roman which is the default font in Microsoft Word<sup>®</sup> 2003. The font size 11, is the second most used font size and is the default for Microsoft Word<sup>®</sup> 2007, 2010 and 2013. See Table 12 in the appendix for the full list of font size occurrences.

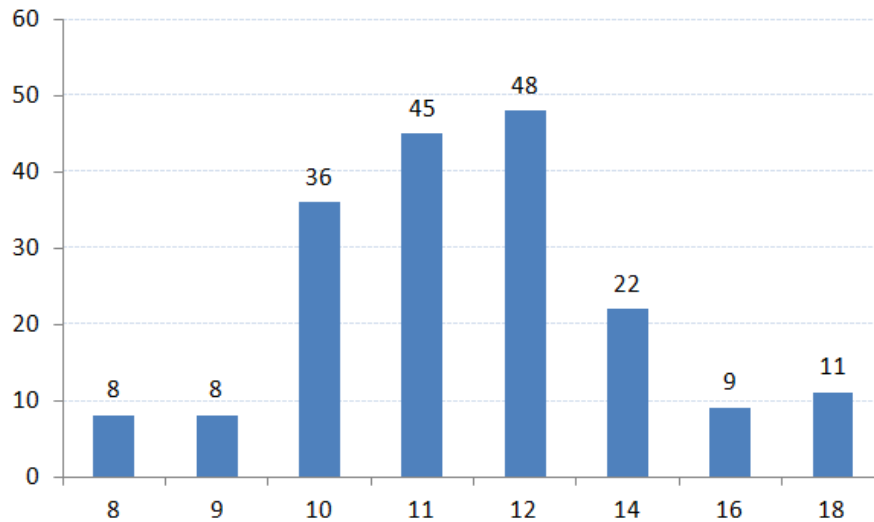


Figure 24: The most recurring font sizes in the dataset.

The most frequently occurring font and size is Calibri 11, (see Figure 25) which is the default font and size for Microsoft Word<sup>®</sup> 2007, 2010 and 2013. See Table 15 in the appendix for the full list of combinations of font and font size occurrences.

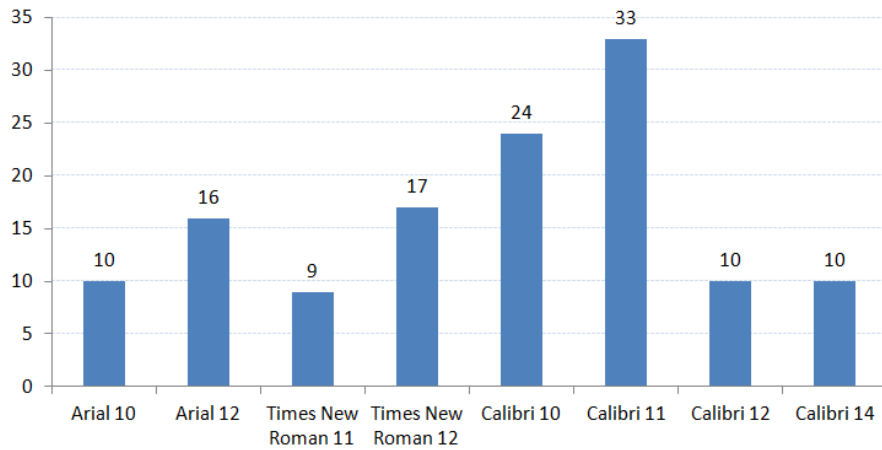


Figure 25: The most frequently recurring combination of fonts and sizes in the dataset.

Other frequencies of characteristics can be seen in Figure 26. Where 74 documents contains bold text, 49 contains colored characters or characters with a colored background, and 45 of the documents contains lists see Table 14 in the Appendix for a full lists.

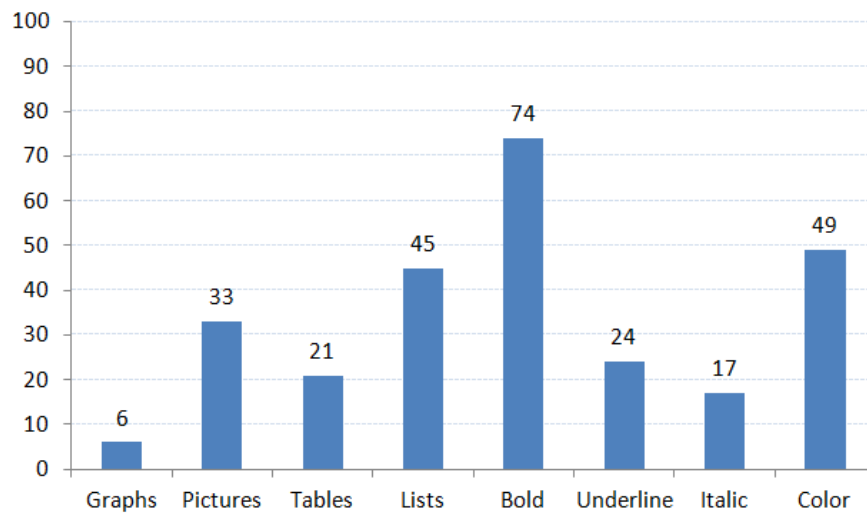


Figure 26: Occurrences of different types of characteristics present in the dataset.



#### 4.2.2 Analysis of the experiment data

The results are presented in bar charts, where each blue bar represent the percentage of documents that are identified correctly without text sanitation and red bars represents the percentage of documents identified correctly with text sanitation.

#### 4.2.3 Document-to-image

The following results are obtained from the comparison of text generated from the scanned document through OCR and the extracted text from the original documents.

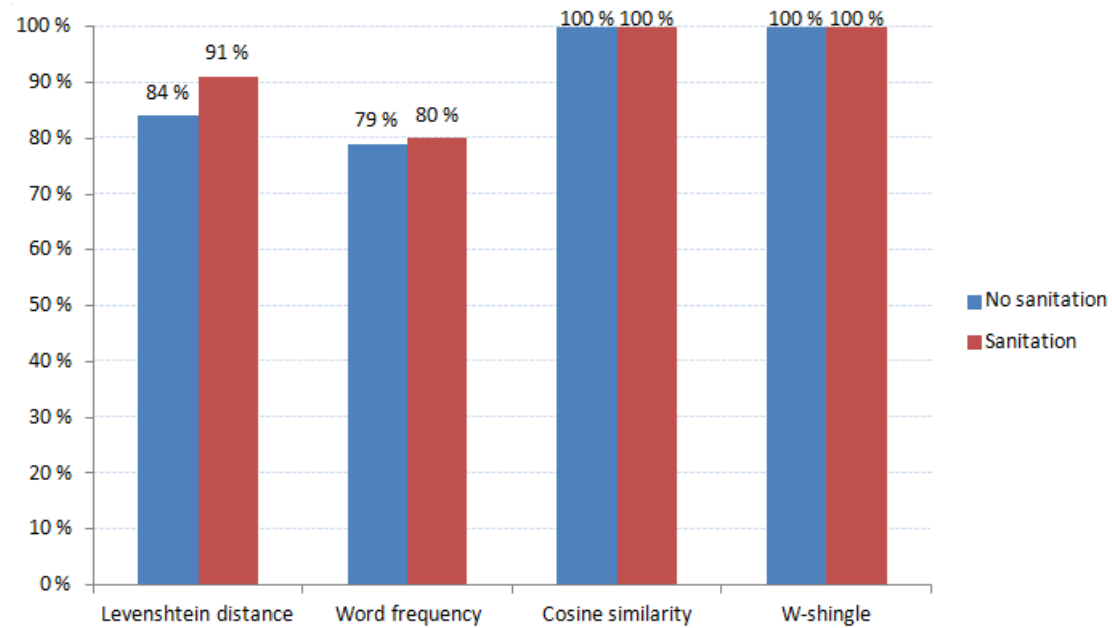


Figure 27: Matching accuracy for the different text comparison methods with and without text sanitation.

In Figure 27 a small increase of 7% in matching accuracy for Levenshtein and 1% for Word frequency when using text sanitation are displayed.

#### 4.2.4 Image-to-image

The following results are obtained from the comparison of text generated from the scanned document and the converted original document, through OCR.

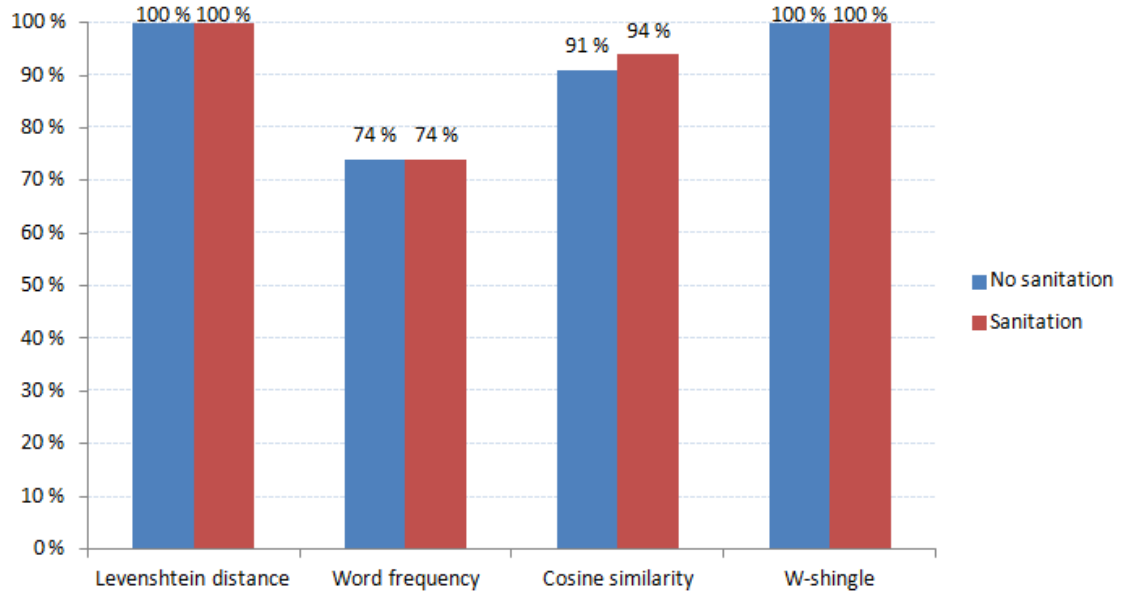


Figure 28: Matching accuracy for the different text methods with and without text sanitation.

In Figure 28 Levenshtein distance reaches up to a 100% accuracy when comparing OCR generated text from the scanned document and the OCR generated text from a converted document, whereas Word frequency and Cosine similarity drops a few percent. With text sanitation a small increase for Cosine similarity is observed, and the accuracy for the remaining methods stays at the same level.

#### 4.2.5 Pixel comparison

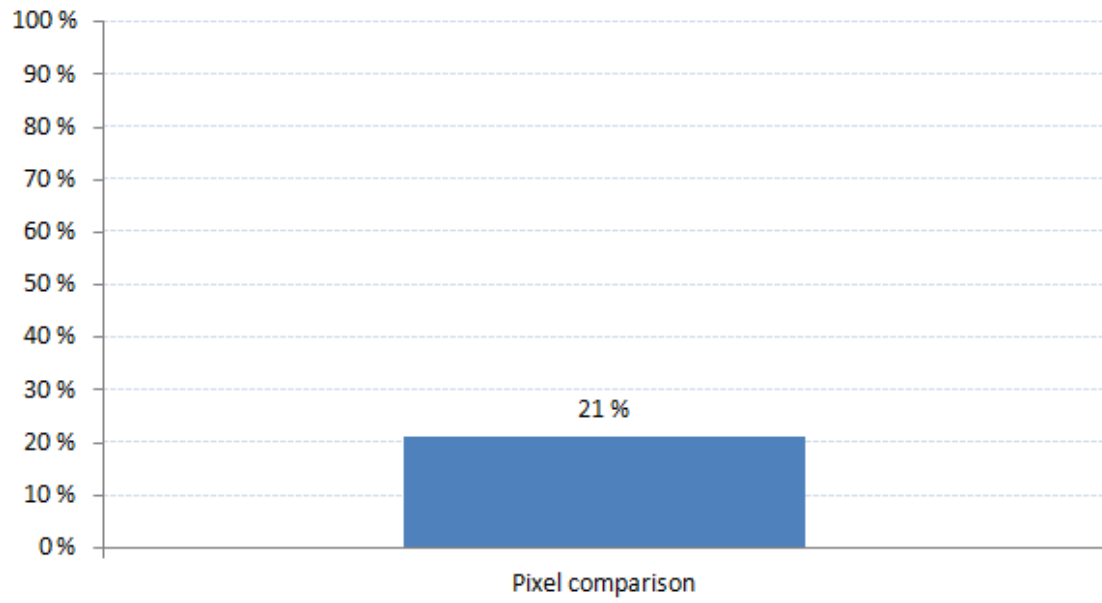


Figure 29: Matching accuracy for the pixel comparison method.

In Figure 29 the pixel-to-pixel comparison methods manages to identify 21% documents correctly based on the best match.

#### 4.2.6 Similarity score

The scores here were obtained through the process described in Section 4.1, where a one-to-one comparison between the documents with different configurations are calculated and normalized. "Document" denote the text extracted from the original document. "OCR original text" denote the text generated from OCR on the image from the converted PDF document. Both are compared to the OCR generated text from the scanned documents.

Table 2: Similarity score, without text sanitation.

Text source	Score
Document	0.31
OCR original text	0.18

Table 3: Similarity score, with text sanitation.

Text source	Score
Document	0.20
OCR original text	0.16

#### 4.2.7 Result comparison

Table 4: Accuracies of all texts comparison methods for all combinations of settings.

Configuration/Methods	Levenshtein	Word freq.	Cosine sim.	W-shingles	Average
Doc. to Img.	84%	79%	100%	100%	90.75%
Doc. to Img. Sanitized	91%	80%	100%	100%	92.75%
Img. to Img.	100%	74%	91%	100%	91.25%
Img. to Img. Sanitized	100%	74%	94%	100%	92.0%
Average per method	93.75%	76.75%	96.25%	100%	

#### 4.2.8 Runtime

The runtime was measured using a timer. This timer is started when the execute button is pushed (see Figure 38 in the appendix), and terminates when the all the conversions and calculations are done.

Table 5: Runtime for the experiments

Experiment	Timing
Image to image (200 images, one-to-one)	8h 43m 8s
Image to image (200 images, one-to-many)	6h 2s
Document to Image (100 images, one-to-one)	4h 11m 31s
Document to image (100 images, one-to-many)	6h 32m 46s
Document to image (1 image, one-to-many)	4m 26s

## 5 Discussion, Conclusion and Future work

This chapter contains the discussion about the results and their significance, the conclusion based on these findings and, theoretical and practical implications, and lastly future work.

### 5.1 Discussion

As described in Chapter 2, the term good accuracy is from 99% and upwards, which was the threshold given by the Meeting of the Experts on Digital Preservation [23]. In Table 11 in the appendix the number of changes to transform the OCR scanned text into the extracted text from the document for each document without sanitation is shown. The number of changes between two documents ranges from 28 to 1999 in the test dataset, and the average is 374 changes per document. The average number of characters per document in this dataset is 1725. Resulting in a 21.7% dissimilarity for each document on average or, 78.3% similarity, below the proposed limit. However, the OCR conversion of these documents are not supposed to be used as a substitute for the real document (electronic or physical) as evidence, just as a help to locate the original ones, but the following is worth mentioning.

A document is categorized in one of two categories; original or a duplicate. In Federal Rules of Evidence, Rule 1002 [67] states that an original recording, photograph or writing must be presented, before a secondary source can be admitted. If the original evidence is unavailable, a duplicate can be admitted in lieu of the original as long as there is a satisfactory reason.

The definition of evidence types in Rule 1001 [68] states:

- (a) A “writing” consists of letters, words, numbers, or their equivalent set down in any form.
- (b) A “recording” consists of letters, words, numbers, or their equivalent recorded in any manner.
- (c) A “photograph” means a photographic image or its equivalent stored in any form.
- (d) An “original” of a writing or recording means the writing or recording itself or any counterpart intended to have the same effect by the person who executed or issued it. For electronically stored information, “original” means any printout - or other output readable by sight - if it accurately reflects the information. An “original” of a photograph includes the negative or a print from it. [68]

The primary evidence i.e. the original source, is admissible without objection. It can still be challenged as to if it is the best evidence available.

## Duplicates

When an original source cannot be admitted, and a copy must be used instead, provided that the reason for this is adequately explained, it is called a duplicate and is defined in Rule 1001 [68] as:

(e) A “duplicate” means a counterpart produced by a mechanical, photographic, chemical, electronic, or other equivalent process or technique that accurately reproduces the original.[68].

In lieu of the original source, Rule 1003 [69] states:

"A duplicate is admissible to the same extent as the original unless a genuine question is raised about the original's authenticity or the circumstances make it unfair to admit the duplicate." [69]

Rule 1001 paragraph 3 & 4 [68] states that evidence reproduced with high accuracy and virtually no possibility of error can be given the status of original evidence, such as computer printouts and carbon paper copies<sup>1</sup>.

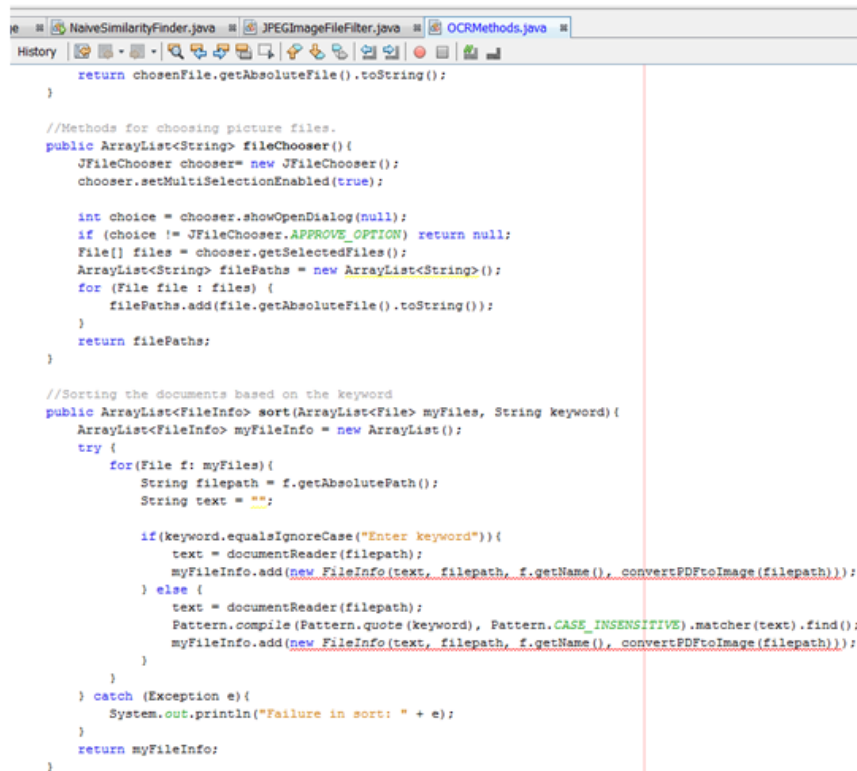
A common factor for the ten highest ranking matches is that they consist mainly of text, two documents has small images. No tables, and main font and font sizes are Calibri, Times New Roman, Arial and 11 and 12 in sizes. For the ten lowest ranked documents a common theme is that they all contain a picture which adds a lot of extra text when converted, see Figure 30, where in this case the original text is 505 character long, but the OCR conversion is 1787 character long. For identifying the digital document with text comparison, this is not a good thing. But, if the goal was to make this document searchable so you could search in all of the text, the OCR does its job.

The second scenario is where a picture adds a lot of extra noise, as seen in Figure 31. Here the logo has been translated into ",42 -w", the logo name has been translated correctly, but the header and the line beneath is unintelligible. The table add some noise, and the picture adds a lot of excess characters. The original OCR output before run through the text sanitation is 633 characters, and after the text sanitation the OCR text is cut down to 351 characters, which is a decrease of 44.5%. The original text is cut down to 221 from 254 which is a decrease of 13%. So, we remove a lot of noise from the text output from the images, but retain most of the original text.

However, since the proof-of-concept tool and OCR in this instance is used to locate the original digital document, the accuracy does not have such a high impact except, on the results for the linking. In addition, when you get a potential match with the application, the match would need to be verified by an operator, to ensure it is the correct document. One implication of the evidence rule is that if the physical document you find is tarnished in some way that makes it difficult to read, a fresh copy could be printed from the original document and presented in its place.

---

<sup>1</sup>The laws discussed here apply to the US.



```

return chosenFile.getAbsolutePath().toString();
}

//Methods for choosing picture files.
public ArrayList<String> fileChooser() {
    JFileChooser chooser= new JFileChooser();
    chooser.setMultiSelectionEnabled(true);

    int choice = chooser.showOpenDialog(null);
    if (choice != JFileChooser.APPROVE_OPTION) return null;
    File[] files = chooser.getSelectedFiles();
    ArrayList<String> filePaths = new ArrayList<String>();
    for (File file : files) {
        filePaths.add(file.getAbsolutePath().toString());
    }
    return filePaths;
}

//Sorting the documents based on the keyword
public ArrayList<FileInfo> sort(ArrayList<File> myFiles, String keyword) {
    ArrayList<FileInfo> myFileInfo = new ArrayList();
    try {
        for(File f: myFiles){
            String filepath = f.getAbsolutePath();
            String text = "";

            if(keyword.equalsIgnoreCase("Enter keyword")){
                text = documentReader(filepath);
                myFileInfo.add(new FileInfo(text, filepath, f.getName(), convertPDFtoImage(filepath)));
            } else {
                text = documentReader(filepath);
                Pattern.compile(Pattern.quote(keyword), Pattern.CASE_INSENSITIVE).matcher(text).find();
                myFileInfo.add(new FileInfo(text, filepath, f.getName(), convertPDFtoImage(filepath)));
            }
        }
    } catch (Exception e){
        System.out.println("Failure in sort: " + e);
    }
    return myFileInfo;
}

```

This snippet of code handles the input when choosing a folder for files, as well as sorting the files based on a keyword supplied by the operator.

```

public String convertPictureToText(ArrayList<String> pictureFilePaths){
    String[] input = new String[3];
    input[0] = "/bin/bash";
    input[1] = "-c";
    input[2] = "ocropus -lbin ";
    for(int i = 0; i < pictureFilePaths.size(); i++){
        input[2] += pictureFilePaths.get(i).toString() + " ";
    }
}

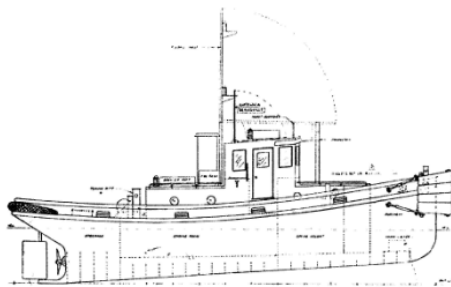
```

Figure 30: Document with an embedded picture with a lot of text, which adds extra text in the OCR conversion.



### Technical details

Name	Daedalus
Type	Scout
Class	I
Length	2000m
Width	100m
Weight	20000 ton
Cargo space	1500m3
Crew	1450
Fuel	20000000 liters
Power	10 GW
Speed	250 Knots
Engine	Antimatter drive Nibbler 2000
Weapon systems	N/A



First part of the string	Second part of the string
,Ae	0(
-w	;N
Space Corp Inc	A),,re---
riofoil	-]le-
1 oxrk mir-nl	.;,L.2M--Ae==-,H-----
4 w-wAAAAAMciA LaL	
WCu11t	e71
Name	za==a----].
Type	Y'r
	alg agLaagsgsaeaseeea 2MMrPwmrr-
Class	v'
length	gessAg asees
Width	--r''
Weight	--...f., 4-g 1
Cargo space	i,;-----
Crew	++,.
Fuel	=wwwwwwre
Power	
Speed	] eww-A s-
Engine	]ilests.
Weapon systems	L-4-'t;-----t''' - (
	LSAS-
	'aa...L..L.A-
	=-]
	n]
	----
Daedalus	==w-]
Scout	..
2000m	]----'r; --
100m	]
20000 ton	]--2
1500m3	L-----
1450	., '.
20000000 liters	
10 GW	
250 Knots	
Antimatter drive Nibbler 2000	
N/A	

Figure 31: The OCR results of the document to the left. The second column of text is the continuation of the text on the right beneath the first column, to make one long string.



As seen in Section 4.2.1 the font Calibri is the most recurring one in the dataset, and it is used in 51 out of the 100 documents. Calibri is the default font in Microsoft Office® 2007, 2010 and 2013, and as such, no big surprise to see it rank this high.

For the font sizes, 12 is the most frequent. Font size 12 is the default size for Times New Roman, which is the default font in Word 2003. Next in line is font size 11, and is the default size used in Microsoft Word® 2007, 2010 and 2013. Only three occurrences separates these two results, and had the dataset included Excel (XLSX) files, the occurrences of font size 11, would most likely been more preminent.

The prominent occurrences of Calibri 11, does not come as a big surprise either since Calibri 11 is the default font and font size combination for Microsoft Word® 2007, 2010 and 2013.

Other characteristics emerging, are bold letters, colored background or colored letters, lists and pictures (here graphs that are not made by the graph tool in Word count as a picture). This is true for this dataset, and the distribution of characteristics in Figure 26 might be different if the dataset had included documents with several pages. This is only speculation from the author, but we assume that the number of documents with tables, pictures, and color would increase. This is also tied to the type of document used to create the dataset. Had the dataset been created with only text files, there would have been no tables, pictures, or graphs etc., only text.

As to how realistic this dataset is, it is difficult to give an answer, since there are such a variety of datasets and file types available. The dataset fulfills some of the criteria put forth in Section 2.4. However, not all of the criteria are applicable to this dataset. Characteristics that should be added are more file types and documents with degradation, such as; coffee spills, water damage, crumbled together etc. The most likely characteristic for this dataset that should be changed is the length of each document.

The Levenshtein distance results are at an 84% accuracy, see Figure 27 without text sanitation. Which means that, the excess information added by OCR conversion makes noise that affects the matching accuracy. With text sanitation the accuracy improves with 7%, so the removal of the excess noise improves the results for Levenshtein distance.

By comparing the text generated with OCR from all documents and images, as seen in Figure 28, Levenshtein distance increased to 100% accuracy, while Word frequency and Cosine similarity drops, W-shingles stays at a 100% throughout all configurations.

The Word frequency method manages to identify 79% of the documents correctly without text sanitation see Figure 27. However, for this method the text sanitation did not have such a good effect where we got a 1% increase. As seen in Figure 28 the result drops when comparing image-to-image with OCR generated text. This drop in accuracy could be the results of the OCR conversion generating too much noise on both sides, and the removal of this potential noise did not

have any impact. To improve these results for Word frequency, a different way to calculate the scores than the one implemented could be used, one example of such a calculation is the one used in [49].

For Cosine similarity and W-shingles they both score a 100% without text sanitation, as seen in Figure 27 and a 100 % with text sanitation when using document-to-image comparison. In the results from the image-to-image comparison in Figure 27, we see that Cosine similarity falls behind without text sanitation. Again, this might be because of the noise introduced by the OCR conversion. However, we see a small increase in the accuracy when the text sanitation method is utilized. The W-shingle method keeps itself at a steady 100% for all the combinations.

The results for the pixel comparison method are shown in Figure 29, and the accuracy here is quite low at 21%. The poor results of the pixel comparison can be a result of the limitations discussed in Section 3.8.

Using string sanitation resulted in an increase of 8% on the best matching documents (7% for Levenshtein and 1% for Word frequency), and it did not affect the top matching algorithms at all when comparing documents to OCR generated text from the scanned documents. With the comparison of image-to-image we see a small increase of 3% for Cosine similarity when utilizing string sanitation. In total, the difference between the worst and best result for different configurations are 2% with document to image comparison without sanitation at 90.75% and document-to-image comparison with sanitation at 92.75%. Whereas, image-to-image comparison with and without sanitation scores 92% and 91.25% shown in Table 4.

These results are not consistent with the similarity score given in Tables 2 and 3. According to these numbers the configuration which should have scored the highest is image-to-image comparison with text sanitation at a score of 0.16, but it is 0.75% behind document-to-image comparison with text sanitation at a score of 0.20 and 92.75%. However, if we look at the results for just Levenshtein distance the similarity score matches up with the percentage. This is not a big surprise since Levenshtein distance was used to calculate both of these scores.

Based on the results for this dataset (see Table 4) the characteristics that enables evidence linking most accurately are "shingles" used in W-shingles and the frequency of unique words used in Cosine similarity on a document-to-image comparison with text sanitation. However, the author stress that this is true for the dataset used in the thesis, and might not be true for other datasets.

By looking at the scores in tables 2 and 3 we can see that OCR generated text from the original document, compared to the OCR generated text from the scanned documents with text sanitation yields the best score. By comparing OCR generated text from two images, we circumvent the problem about text extraction and excess text discussed in 5.1. Since both images contains the same information, they are more likely to create the same noise. However, as seen in column 5 in Table 4 the impact of image-to-image comparison versus document-to-image with and without

text sanitation is minimal, the maximum difference is 2.5%.

The run time of the tool while converting image to text is slow, as seen in Table 5. The runtime for each experiment is several hours, which contradicts the statement in [15].

"To be useful, OCR software must be both accurate and fast."

This application and its implementation of OCRopus is not fast enough when running the comparison with a 100 documents. However, if we want to only locate one document, and its best matches, the tool is useful. One single image takes approximate 4 minutes and 20 seconds to be processed and matched. The reason for this, is that the conversion of all 100 PDFs to image files takes additional time. To reduce the time and make the tool and experiments more efficient would be to find a faster way to convert the documents to images, and a faster OCR engine. Or, by dropping the pixel-to-pixel comparison as well as the image-to-image comparison with OCR. The time for each single image would drop down again to around 2 minutes. Moreover, the removal of excess methods can also help reduce the processing time. There are also the possibility to scan directly to PDF and run the OCR engine in Adobe Acrobat<sup>®</sup> on these documents, and then use the proof-of-concept tool to extract the text from these documents for comparison to the corpora. The discrepancies in the run times can be the result of some of the experiments being run in batches of 25 and 25 documents, and not all in one go. Each of these batches would add a new preprocessing stage with the document to image conversion, and some extra time has accumulated in the end results. However, the difference between 200 images, one-to-many and 100 images, one-to-many with only 30 minutes extra for double the amount of images we can not explain. Better timers should be used and the machine running the experiment should be in a controlled environment for the duration of the experiments.

As discussed previously in Chapter 2 ensemble voting is a good way to increase the accuracy of OCR engines. For the results here, the ensemble voting is not a good idea. Since the four other methods would decrease the overall similarity score for a document compared to *W*-shingle's 100% matching accuracy. The implementation of ensemble voting and the removal or addition of methods are a part of the future work, see Section 5.3.

## 5.2 Conclusion

### General

The methods chosen for this thesis (Experimental design and literature review) yielded tangible results, but it should be mentioned that a qualitative method with interviews with law enforcement investigators or a quantitative method with questionnaires about file types, problems encountered when using forensics software etc., would have added more weight to the Justification, Motivation and Benefits parts of the thesis.

### What is an acceptable error rate for OCR in a forensics setting?

As discussed the acceptable error rate for OCR in a forensics setting is really not that important since it is only used for matching. But the higher the accuracy of the OCR engine the more likely we are to get an accurate match. The best way to increase this accuracy is by using voting ensembles on the classifier level of the OCR system, or character for character between two OCR engines. The proposed level for a good OCR system is at 99%.

### What are characteristics of a realistic dataset in forensic research, development and testing?

As seen, the characteristics varies quite a lot, but for this specific dataset the most prominent characteristics are the font Calibri, font size 12, and combination of font and font size, Calibri 11. As well as the occurrences of bold text, color, lists and pictures.

### Which of the characteristics are significant and enable evidence linking?

The characteristics that enables linking most accurate are "shingles" used by W-shingles and unique word frequency and the angle between the two vectors used in Cosine similarity. Cosine similarity does not take order into account so the order does not really matter for the identification. However documents could be crafted that would identify as a match even though the order is completely different, just as long the word frequencies are the same. W-shingles on the other hand does take order into account and is the best algorithm for matching on this dataset for all configurations.

### In which way is it possible to increase efficiency and effectiveness in evidence linking?

We made sure that the input for the comparison methods were as similar as possible. Either by comparing image-to-image with OCR, or document-to-image with and without text sanitation. In addition the run times for the OCR engine needs to be reduced. The author believes that the proof-of-concept tool can help investigators locate evidence faster if the proposed changes are made.

### 5.2.1 Theoretical implications

As seen from the results presented, one could in theory drop all other methods except, W-shingles and still get a 100% linking accuracy, but we need to keep in mind the "No Free Lunch Theorem" and that these result might not be the same for other datasets. As discussed in Chapter 2 ensemble voting can be used to increase OCR accuracy, but the results here show that there is no need for this in this setting. Had W-shingles not scored a 100% for all configurations this could have been a way to increase the matching accuracy. However, it must be said that ensemble voting could still be used for the conversion if it was between two OCR engines, which in turn could improve the overall accuracy of each method. It should also be mentioned that we are aware that the implementation and utilization of some methods completely ignore semantics, word order and important relationships in sentence structures, such as dependencies between words.

### 5.2.2 Practical implications

As discussed earlier the runtime of these experiments are too long. A faster OCR engine is needed, or tweaking of the current one needs to be done. The runtime for matching a single document is acceptable, but the runtime becomes infeasible when comparing several documents at a time. In effect, the conversion of documents into images should be dropped, as well as the pixel-to-pixel comparison which has the lowest accuracy score overall, or be replaced by a better image recognition algorithm. The Word frequency and Levenshtein distance methods can be dropped as well to reduce complexity of the source code. For the experimental part, image-to-image comparison should be dropped, unless a faster OCR engine is found or the multi thread options is enabled and the output ordered. A practical implication of implementing ensemble voting on OCR level would most likely result in an even longer runtime unless both of the OCR engines are fast.

### 5.3 Future work

The main areas that needs an improvement are:

#### OCR engine

For this proof-of-concept tool to be usable the total matching time needs to be reduced. This can be done through tweaking OCRpus, training OCRopus on the dataset or replacing OCRopus as the OCR engine. Tests with multi thread support should be done as well. If this are to be implemented in the proof-of-concept tool, the raw text generated by OCRopus<sup>TM</sup> can be processed and sorted to get the correct order, before removing the excess text.

It is worth mentioning that many scanners have the option to scan to a PDF document instead of an image file. By using Adobe Acrobat<sup>®</sup>, this document can then be made searchable and the text can be extracted using Java libraries (as done in the proof-of-concept tool). Adobe Acrobat<sup>®</sup> supports batch jobs, which speeds up this process [65], a server side solution for automation of this process exists as well [66]. By using these we could circumvent the slow processing of OCRopus and only use the text comparison features of the proof-of-concept tool.

#### Dataset

A more realistic dataset should be created. To get access to a proper dataset from a law enforcement agency and base a new dataset on this would be a promising start. In addition the dataset should include documents that are tarnished, either by coffee spills, crumbled together and straightened out, wet and dried and partially shredded etc. This would give the methods and OCR engine an additional challenge, and it is not unlikely to encounter documents like this on a crime scene. A combination of file types should be added to the dataset as well, spread sheets, images, PDFs, DOC, DOCX, txt files and emails to make it less uniform and more realistic. The dataset should also contain modified files from the original one, so matching accuracy for similar documents can be tested. To make the dataset more interesting, documents created and modified by several users could be added, so the investigators can take the investigation one step further by looking at the metadata of the documents.

#### Proof-of-concept tool

If this tool is to be used, the GUI needs to be cleaned up, as well as the source code, and better libraries must be found (for file conversion so file naming can be controlled), depending on which methods are kept and which are dropped. To make it even more usable the application should be able to run on a Windows machine, information on that can be found here: [70]. If this is not possible a different OCR engine should be considered. Support for documents with multiple pages, and the automatic conversion and extraction should be added, but this requires tremendous modifications in the source code.

**Methods**

More text comparison methods can be found and tested to see how they handle this dataset. As shown, the text extraction methods used for the original documents does not manage to extract the text in the embedded images as the OCR engines does. One might look into dropping all text comparison algorithms, and only look at image recognition algorithms for the matching, since this would circumvent the problem with a difference in the two documents. The W-shingles algorithm should be further tested, and the  $w$  value should be varied to see which one yield best results. The computational complexity of the methods, and the comparison in the proof-of-concept tool should be discussed as well.

**Experiment**

The matching accuracy between two similar documents should be tested. The dataset should include an original document, and a modified version of this, with either text added or removed. To see if it manages to match the scanned document to a modified digital document. Modifications here can be: added or removed paragraphs, pictures and graphs etc. In addition, the timing of the experiments should be better, and one way would be to have separate timers for image conversion, similarity calculations, OCR conversion and total runtime.

## Bibliography

- [1] Symantec. 2010. Survey reveals poor availability of digital evidence brings legal process to a halt across emea. [http://www.symantec.com/en/uk/about/news/release/article.jsp?prid=20100907\\_01](http://www.symantec.com/en/uk/about/news/release/article.jsp?prid=20100907_01). Visited 23.05.2014.
- [2] Izenberg, D. 2010. A-g sets time limits for criminal investigations. <http://www.jpost.com/Israel/A-G-sets-time-limits-for-criminal-investigations>. Visited 23.05.2014.
- [3] Riksadvokaten. 2013. Mål og prioriteringer for straffesaksbehandlingen i 2013 — politiet og statsadvokatene. <http://riksadvokaten.no/filestore/Dokumenter/2013/Rundskrivnrifor2013-Mlogpriorforstraffesaksbehandlingen.pdf>. Visited 21.05.2014.
- [4] Garfinkel, S. L. 2010. Digital forensics research: The next 10 years. *Digital Investigation*, 7, Supplement(0), S64 – S73. The Proceedings of the Tenth Annual {DFRWS} Conference.
- [5] Rekdal, J. E. John rekdal - cross comparison of digital and digitized physical evidence. <https://kepler.hig.no/share.cgi?ssid=0F6ma0t>. Visited 16.05.14.
- [6] Palmer, G. 2001. A road map for digital forensic research. In *First Digital Forensic Research Workshop, Utica, New York*, 27–30.
- [7] Casey, E. 2011. *Digital evidence and computer crime: forensic science, computers and the internet*. Academic press.
- [8] Breuel, T. 2009. Recent progress on the ocropus ocr system. In *Proceedings of the International Workshop on Multilingual OCR*, 2. ACM.
- [9] Holley, R. March/April 2009. How good can it get? analysing and improving ocr accuracy in large scale historic newspaper digitisation programs. *D-Lib Magazine*, 15(3/4).
- [10] Broda, B. & Piasecki, M. 2007. Correction of medical handwriting ocr based on semantic similarity. In *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, Yin, H., Tino, P., Corchado, E., Byrne, W., & Yao, X., eds, volume 4881 of *Lecture Notes in Computer Science*, 437–446. Springer Berlin Heidelberg.
- [11] Breuel, n. M. 2008. The ocropus open source ocr system. volume 6815, 68150F–68150F–15.
- [12] Shafait, F. 2009. Document image analysis with ocropus. In *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*, 1–6.
- [13] Du, S., Ibrahim, M., Shehata, M., & Badawy, W. Feb 2013. Automatic license plate recognition (alpr): A state-of-the-art review. *Circuits and Systems for Video Technology, IEEE Transactions on*, 23(2), 311–325.



- [14] Bessmeltsev, V., Bulushev, E., & Goloshevsky, N. High-speed ocr algorithm for portable passport readers.
- [15] Kindseth, J., Peterson, M., Khole, M., & Gogte, A. Character recognition using machine learning techniques.
- [16] Eikvil, L. Ocr - optical character recognition". <http://www.nr.no/~eikvil/OCR.pdf>. Visited 16.05.14.
- [17] Guyon, I., Haralick, R. M., Hull, J. J., & Phillips, I. T. 1997. Data sets for ocr and document image understanding research. *Handbook of character recognition and document image analysis*, 779–799.
- [18] Scientific Working Group on Digital Evidence. 2013. Swgde establishing confidence in digital forensic results by error mitigation analysis. <https://www.swgde.org/documents/CurrentDocuments/2013-09-14SWGDEErrorMitigationAnalysisV1-3>. Visited 23.05.2014.
- [19] Carrier, B., Spafford, E. H., et al. 2003. Getting physical with the digital investigation process. *International Journal of digital evidence*, 2(2), 1–20.
- [20] Cornell University Law School. Daubert standard. [http://www.law.cornell.edu/wex/daubert\\_standard](http://www.law.cornell.edu/wex/daubert_standard). Visited 23.05.2014.
- [21] Abdulkader, A. & Casey, M. 2009. Low cost correction of ocr errors using learning in a multi-engine environment. In *Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on*, 576–580.
- [22] Rahman, A. & Fairhurst, M. 2003. Multiple classifier decision combination strategies for character recognition: A review. *Document Analysis and Recognition*, 5(4), 166–194.
- [23] Russell, J. C. March 12 2004. Report on the meeting of experts on digital preservation. U.S. Government Printing Office, Washington, DC.
- [24] Matei, O., Pop, P., & Vălean, H. 2013. Optical character recognition in real environments using neural networks and k-nearest neighbor. *Applied Intelligence*, 39(4), 739–748.
- [25] Mao, J. 1998. A case study on bagging, boosting, and basic ensembles of neural networks for ocr. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 3, 1828–1833. IEEE.
- [26] William B. Lund, Douglas J. Kennard, E. K. R. 2013. Why multiple document image binarizations improve ocr. *Proceedings of the 2nd International Workshop on Historical Document Imaging and Processing*.
- [27] Liu, C.-L. & Fujisawa, H. 2005. Classification and learning for character recognition: comparison of methods and remaining problems. In *Int. Workshop on Neural Networks and Learning in Document Analysis and Recognition*. Citeseer.

- [28] Rice, S. V., Nagy, G. L., & Nartker, T. A. 1999. *Optical Character Recognition: An Illustrated Guide to the Frontier*. Kluwer Academic Publishers, Norwell, MA, USA.
- [29] Mihov, S., Schulz, K., Ringlstetter, C., Dojchinova, V., Nakova, V., Kalpakchieva, K., Gerasimov, O., Gotscharek, A., & Gercke, C. 2005. A corpus for comparative evaluation of ocr software and postcorrection techniques. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, 162–166 Vol. 1.
- [30] Ringlstetter, C., Schulz, K., Mihov, S., & Louka, K. 2005. The same is not the same - postcorrection of alphabet confusion errors in mixed-alphabet ocr recognition. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, 406–410 Vol. 1.
- [31] Kae, A., Huang, G. B., Doersch, C., & Learned-Miller, E. G. 2010. Improving state-of-the-art ocr through high-precision document-specific modeling. In *CVPR*, 1935–1942. IEEE.
- [32] Kornblum, J. September 2006. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3(Supplement).
- [33] Mandal, A. K. & Tiwari, M. A. October 2012. Analysis of avalanche effect in plaintext of des using binary codes. *International Journal of Emerging Trends & Technology in Computer Science*, 1(Issue 3).
- [34] Guyon, I., Haralick, R. M., Hull, J. J., & Phillips, I. T. 1997. Data sets for ocr and document image understanding research. *Handbook of character recognition and document image analysis*, 779–799.
- [35] Phillips, I., Chen, S., & Haralick, R. Oct 1993. Cd-rom document database standard. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, 478–483.
- [36] Garfinkel, S., Farrell, P., Roussev, V., & Dinolt, G. 2009. Bringing science to digital forensics with standardized forensic corpora. *digital investigation*, 6, S2–S11.
- [37] Garfinkel, S., Farrell, P., Roussev, V., & Dinolt, G. 2014. Digital corpora producing the digital body. <http://digitalcorpora.org/>. Visited 23.05.2014.
- [38] NIST. 2013. The cfreds project. <http://www.cfreds.nist.gov/>. Visited 23.05.2014.
- [39] Forensics Wiki. 2014. Forensic corpora. [http://www.forensicswiki.org/wiki/Forensic\\_corpora](http://www.forensicswiki.org/wiki/Forensic_corpora). Visited 23.05.2014.
- [40] EDRM. New edrm enron email data set. <http://www.edrm.net/resources/data-sets/edrm-enron-email-data-set>. Visited 23.05.14.
- [41] Project, C. Enron email dataset. <http://www.cs.cmu.edu/~./enron/>. Visited 23.05.2014.
- [42] Puzyriov, R. 2013. Digital forensics tool testing. 32. Limited access.

- [43] Masters, H. V. 1928. A study of spelling errors: A critical analysis of spelling errors occurring in words commonly used in writing and frequently misspelled. *The Phi Delta Kappan*, 11(2), pp. 39–41. <http://jstor.org> Requires registration.
- [44] Damerau, F. J. March 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3), 171–176.
- [45] Ristad, E. S. & Yianilos, P. N. 1998. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5), 522–532.
- [46] Navarro, G. 1999. A guided tour to approximate string matching. *ACM Computing Surveys*, 33, 2001.
- [47] Levenshtein, V. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10, 707.
- [48] Azam, N. & Yao, J. 2012. Comparison of term frequency and document frequency based feature selection metrics in text categorization. *Expert Systems with Applications*, 39(5), 4760–4768.
- [49] Rayson, P. & Garside, R. 2000. Comparing corpora using frequency profiling. In *Proceedings of the workshop on Comparing Corpora*, 1–6. Association for Computational Linguistics.
- [50] Singthongchai, J. & Niwattanakul, S. 2013. A method for measuring keywords similarity by applying jaccard's, n-gram and vector space. *Lecture Notes on Information Theory Vol*, 1(4).
- [51] Niwattanakul, S., Singthongchai, J., Ekkachai, Naenudorn, Supachanun, & Wanapu. 2013. Using of jaccard coefficient for keywords similarity. *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, 1.
- [52] Mihalcea, R., Corley, C., & Strapparava, C. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, 775–780.
- [53] Karnik, A., Goswami, S., & Guha, R. 2007. Detecting obfuscated viruses using cosine similarity analysis. In *Modelling & Simulation, 2007. AMS'07. First Asia International Conference on*, 165–170. IEEE.
- [54] Broder, A. Z. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, 21–29. IEEE.
- [55] Pereira, A. & Ziviani, N. 2003. Syntactic similarity of web documents. In *Web Congress, 2003. Proceedings. First Latin American*, 194–200. IEEE.
- [56] Leedy, P. D. & Ormrod, J. E. 2012. *Practical Research: Planning and Design. Tenth Edition*. Pearson.
- [57] Onwuegbuzie, Anthony J., N. L. L. & Collins, K. M. 2012. Qualitative analysis techniques for the review of the literature. *Qualitative Report* 17.

- [58] Breuel, T. 2009. Recent progress on the ocropus ocr system. In *Proceedings of the International Workshop on Multilingual OCR, MOCR '09*, 2:1–2:10, New York, NY, USA. ACM.
- [59] Breuel, T. M. ocropus the ocropus(tm) open source document analysis and ocr system. <https://code.google.com/p/ocropus/>. Visited 23.05.2014.
- [60] Trefis Team. 2013. An overview why microsoft's worth \$42. <http://www.forbes.com/sites/greatspeculations/2013/01/09/an-overview-why-microsofts-worth-42/>. Visited 23.05.2014.
- [61] Hewlett Packard. Hp scanjet 4800 series photo scanner. <http://www.hp.com/ctg/Manual/c00437630.pdf>. Visited 23.05.2014.
- [62] The Apache Software Foundation. Apache pdfbox - a java pdf library. <http://pdfbox.apache.org/>. Visited 23.05.2014.
- [63] Oracle. Class bufferedimage. <http://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html>. Visited 23.05.2014.
- [64] Santos, R. How do i compare two images to see if they are equal? <http://www.lac.inpe.br/JIPCookbook/6050-howto-compareimages.jsp>. Visited 23.05.2014.
- [65] Adobe. Batch ocr using acrobat professional. [http://blogs.adobe.com/acrolaw/2005/10/batch\\_ocr\\_using\\_1/](http://blogs.adobe.com/acrolaw/2005/10/batch_ocr_using_1/). Visited 23.05.2014.
- [66] Adobe. Adobe livecycle pdf generator es4. [http://www.adobe.com/content/dam/Adobe/en/products/livecycle/pdfs/pdf\\_generator.pdf](http://www.adobe.com/content/dam/Adobe/en/products/livecycle/pdfs/pdf_generator.pdf). Visited 23.05.2014.
- [67] Cornell University Law School. Rule 1002. requirement of the original. [http://www.law.cornell.edu/rules/fre/rule\\_1002](http://www.law.cornell.edu/rules/fre/rule_1002). Visited 19.05.14.
- [68] Cornell University Law School. Rule 1001. definitions that apply to this article. [http://www.law.cornell.edu/rules/fre/rule\\_1001](http://www.law.cornell.edu/rules/fre/rule_1001). Visited 19.05.14.
- [69] Cornell University Law School. Rule 1003. admissibility of duplicates. [http://www.law.cornell.edu/rules/fre/rule\\_1003](http://www.law.cornell.edu/rules/fre/rule_1003). Visited 19.05.14.
- [70] Google Groups. ocropus in windows. [https://groups.google.com/forum/#!topic/ocropus/HJ\\_kEsxHue0](https://groups.google.com/forum/#!topic/ocropus/HJ_kEsxHue0). Visited 21.05.14.

## A Appendix

### A.1 Application

When the application is executed the start screen in Figure 32 is displayed. And it is in here all the operations takes place. To start the process of document linking, push "Choose folder" button as shown in 33, browse to the folder containing your original documents and hit ok, see Figure 34. Next we have the option to screen the documents for keywords as shown in Figure 35, this was not done for the experiments. The next mandatory step is to choose a picture 36, or several pictures 37, remember to check of the multiple pictures box. For the whole process to start, push the execute button and wait for the results 38. When the results are calculated you have the ability to sort the results by using the buttons in the middle, and then use the arrow buttons to browse the different documents and their score within that method.

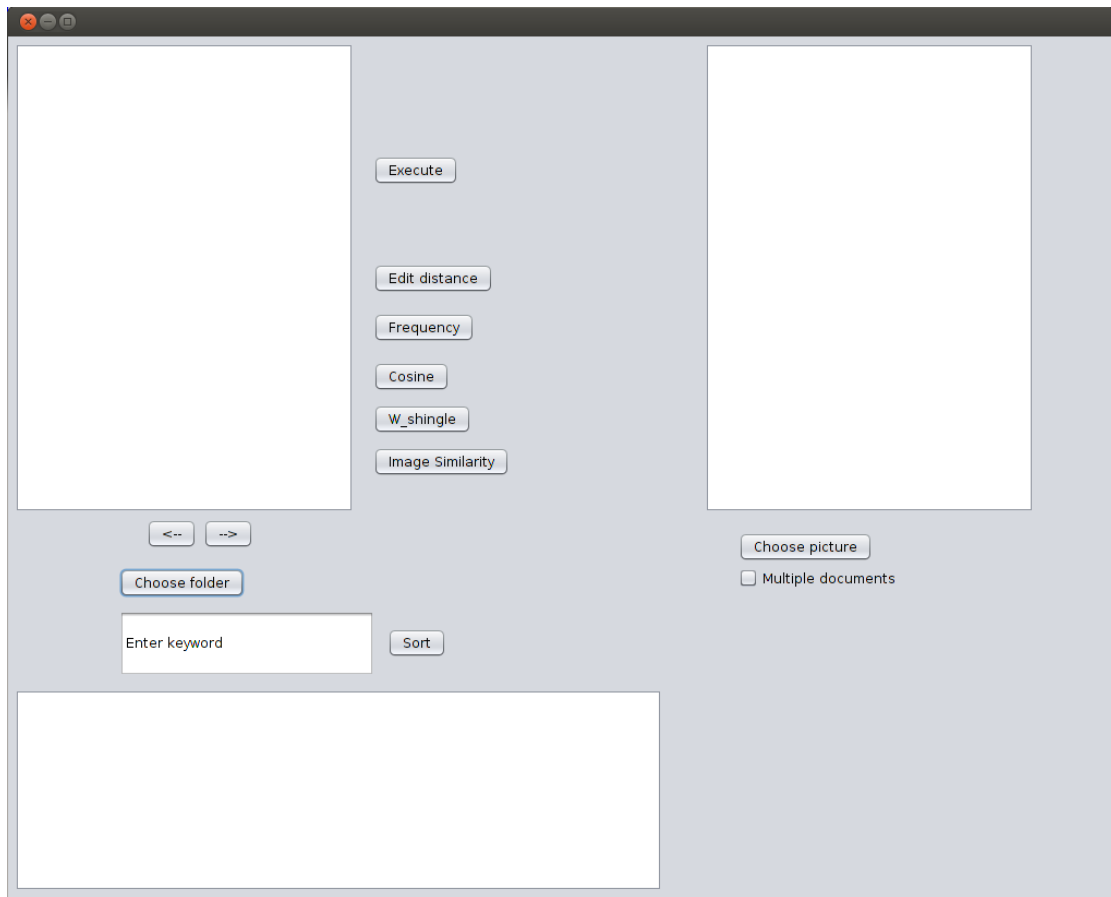


Figure 32: The start screen for the application.

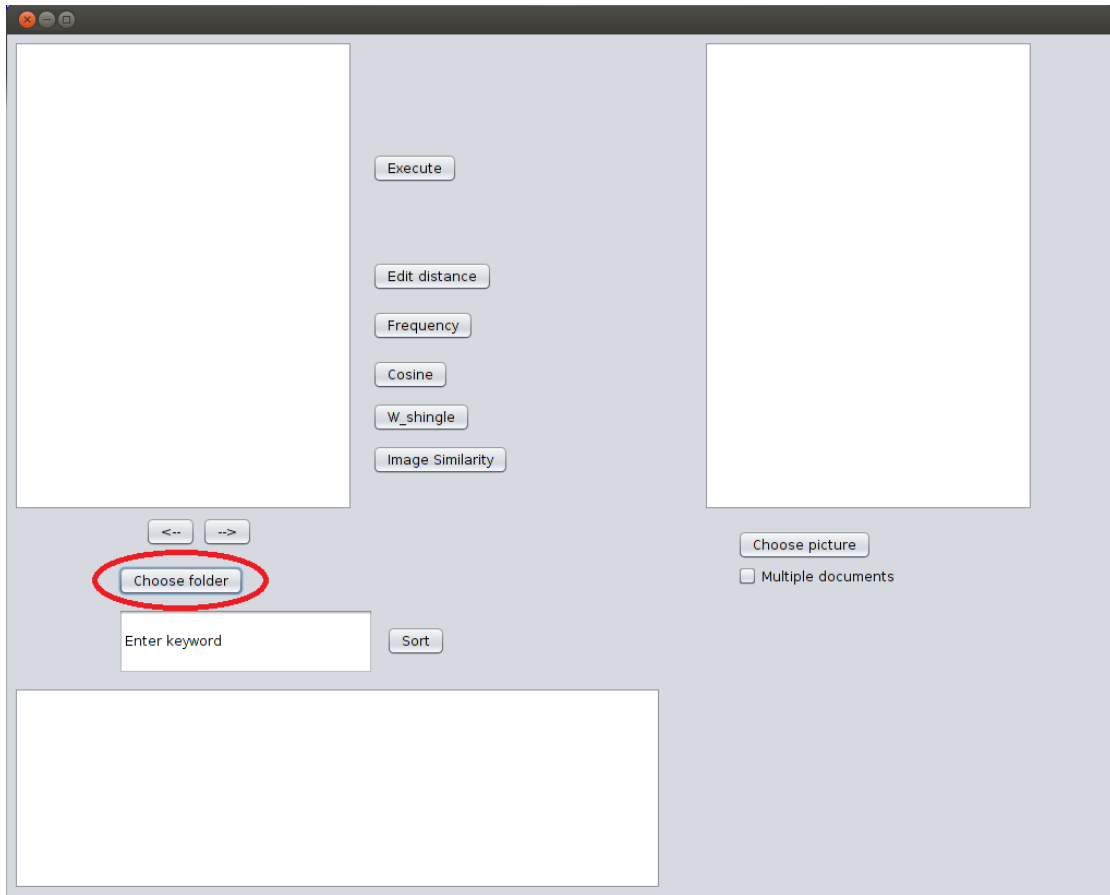


Figure 33: Choosing document folder.

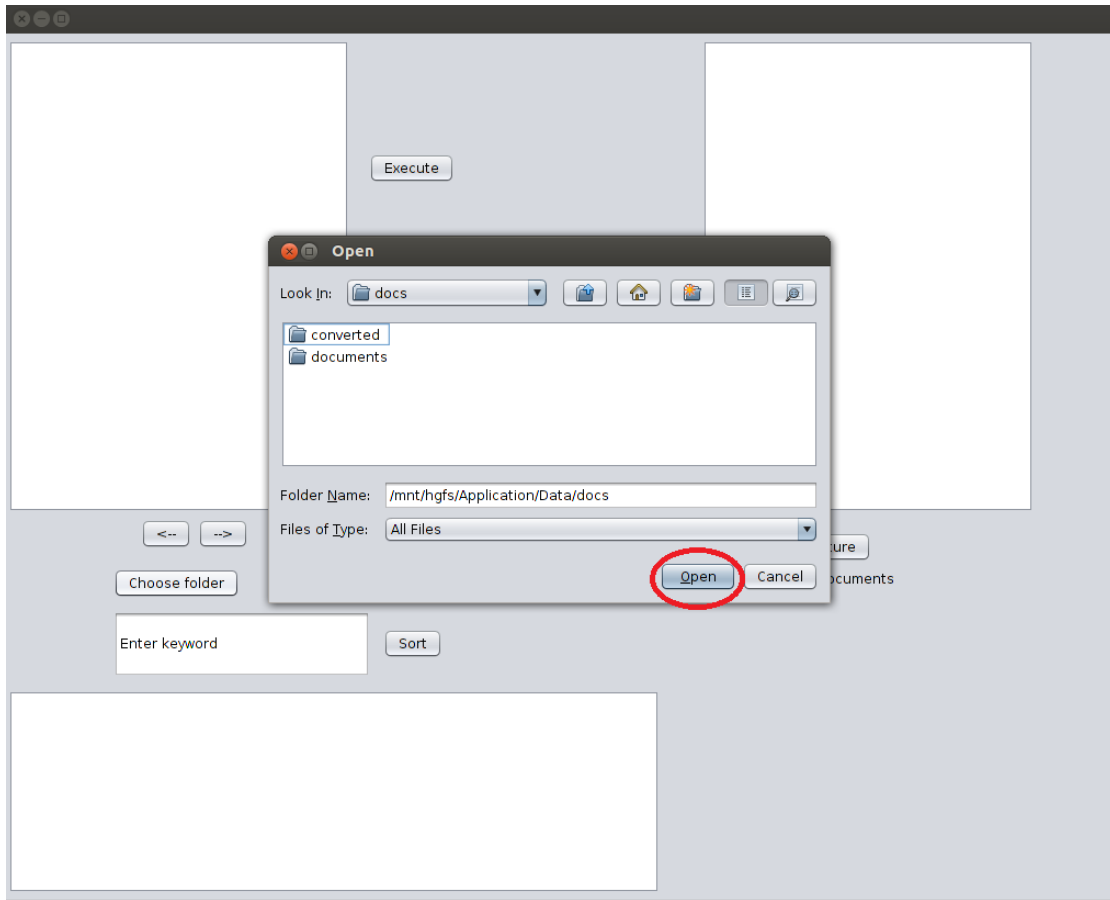


Figure 34: Choosing document folder.



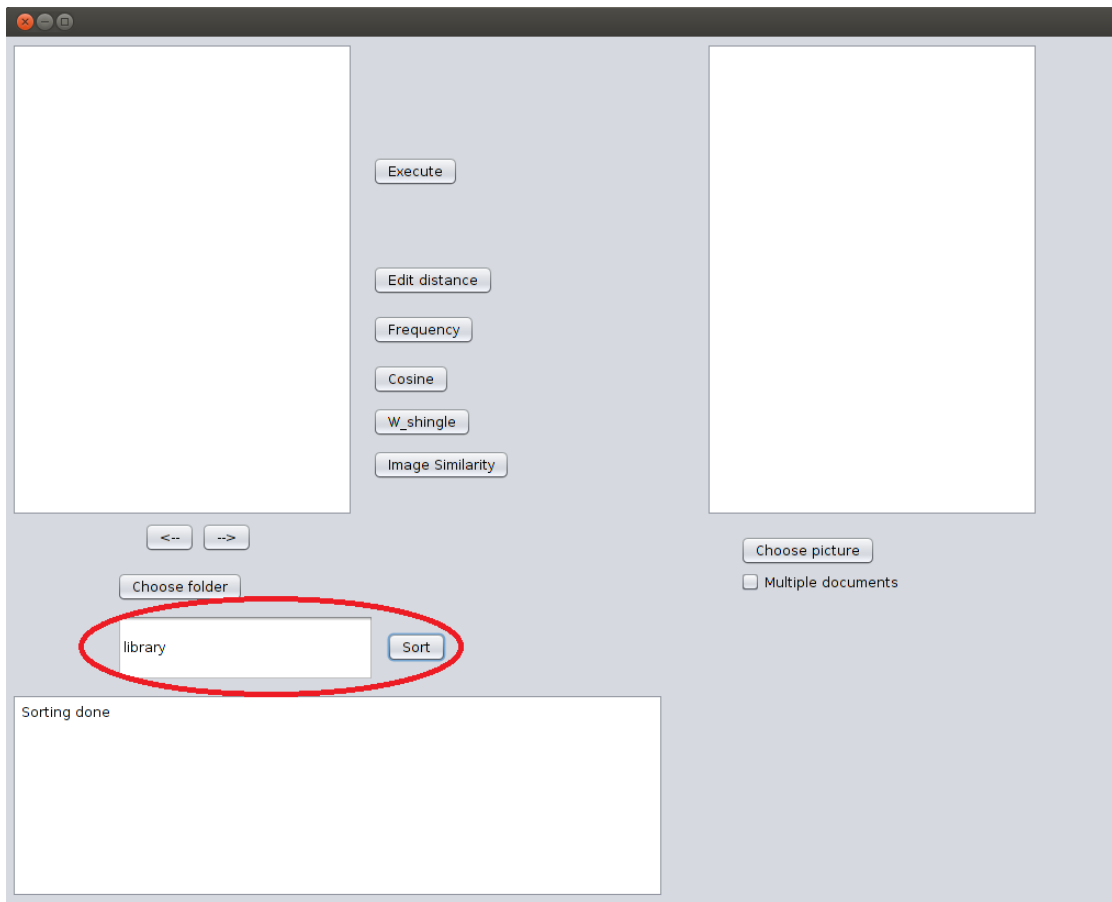


Figure 35: Optional keyword sort, with feedback when sorting is done.

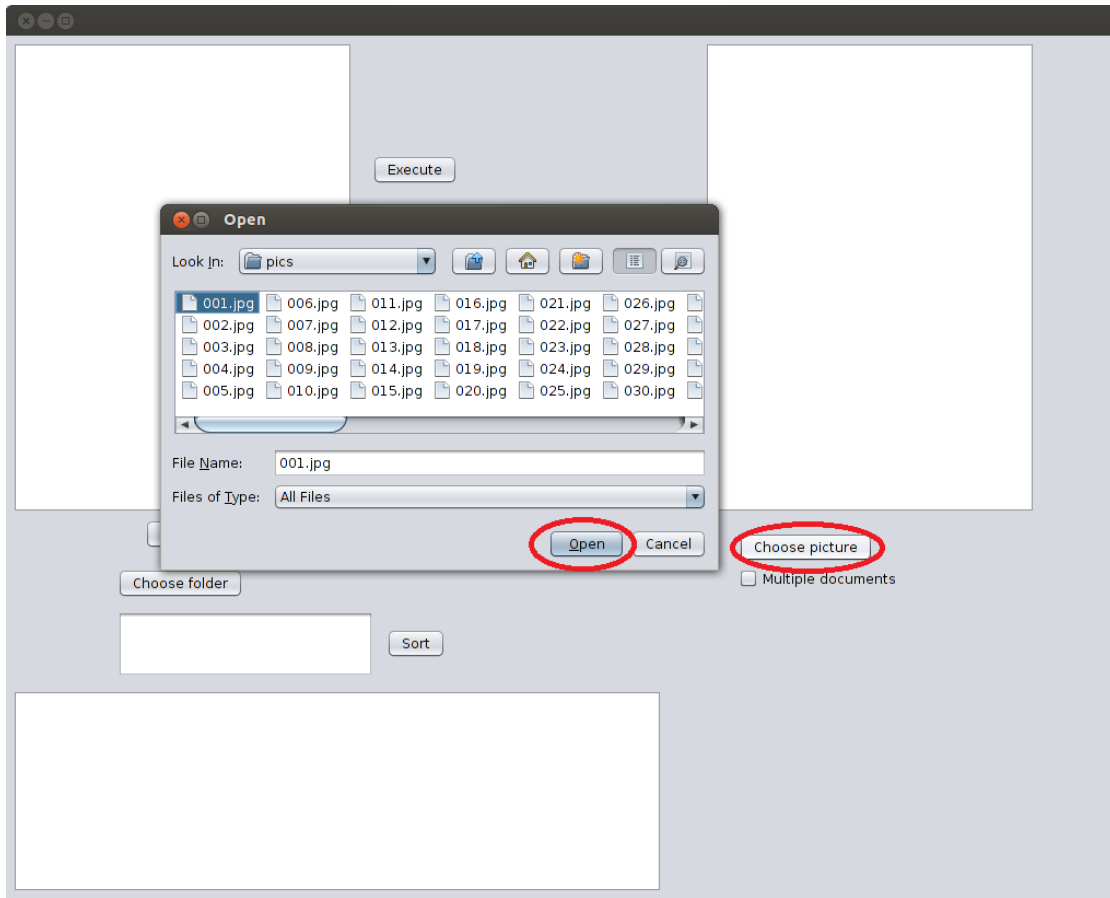


Figure 36: Choosing a single picture for matching.

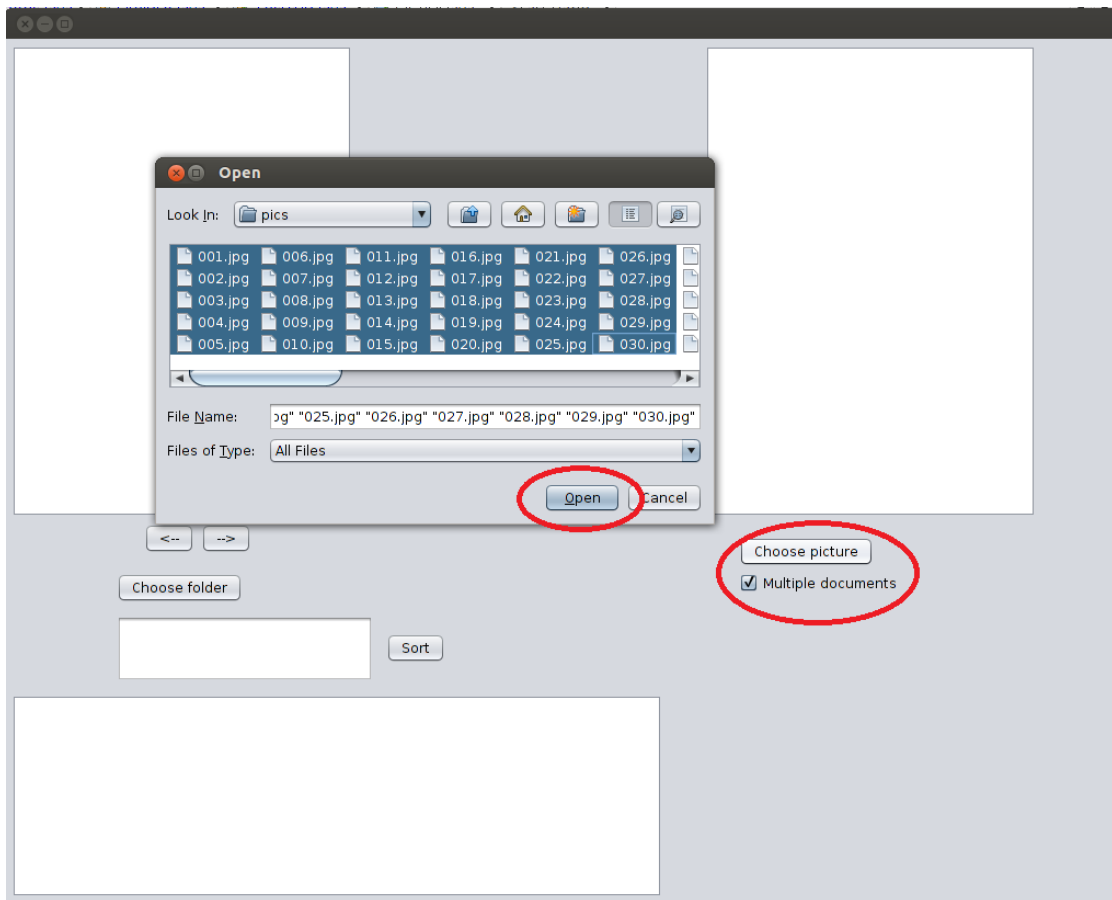


Figure 37: Choosing multiple pictures for matching.

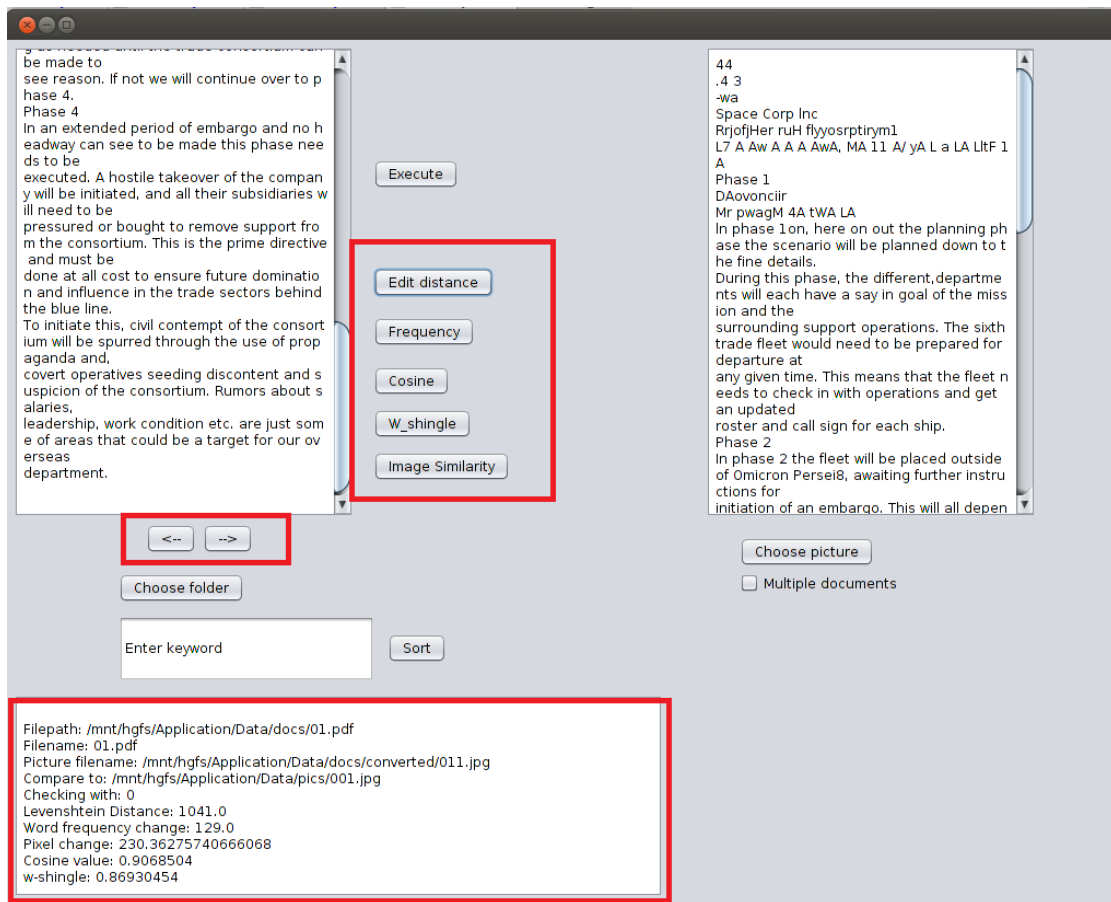


Figure 38: The result after execution With the possibility to sort on hits for each methods, and the ability to browse through all the documents. The original text is displayed on the left hand side, while the OCR text is displayed on the right hand side. At the bottom of the screen the filename and matched document is shown, together with the scores for all the methods

## A.2 Source code

The source code of the application is divided into four classes;

- OCRMethods.java
- OCRUI.java
- FileInfo.java
- Testrun.java

OCRMethods.java is the class which contains all the methods for calculations, file handling, conversion of documents, text manipulation, and execution. See A.2.1 for detailed code.

FileInfo.java is the class for the object that contains all the information about the different documents. It contains the path to the document, the name of the converted picture and which picture it is compare to which yielded the highest match. See A.2.2 for detailed code.

OCRUI.java contains the automatically generated code by Netbeans from the drag and drop creation of the GUI, and is not really that interesting. However, the file do contain some snippets of code, for instances executing of methods when a button is pushed. See A.2.3 for detailed code.

Testrun.java contains a lot of test code, to debug faulty methods and for an easier access to methods, without having to run the whole application. It contains the same methods as OCRMethods.java

The application utilizes many different libraries for file conversion etc, and to be able to run this application these need to be installed as well. See Figure 39 for the full list of libraries used.

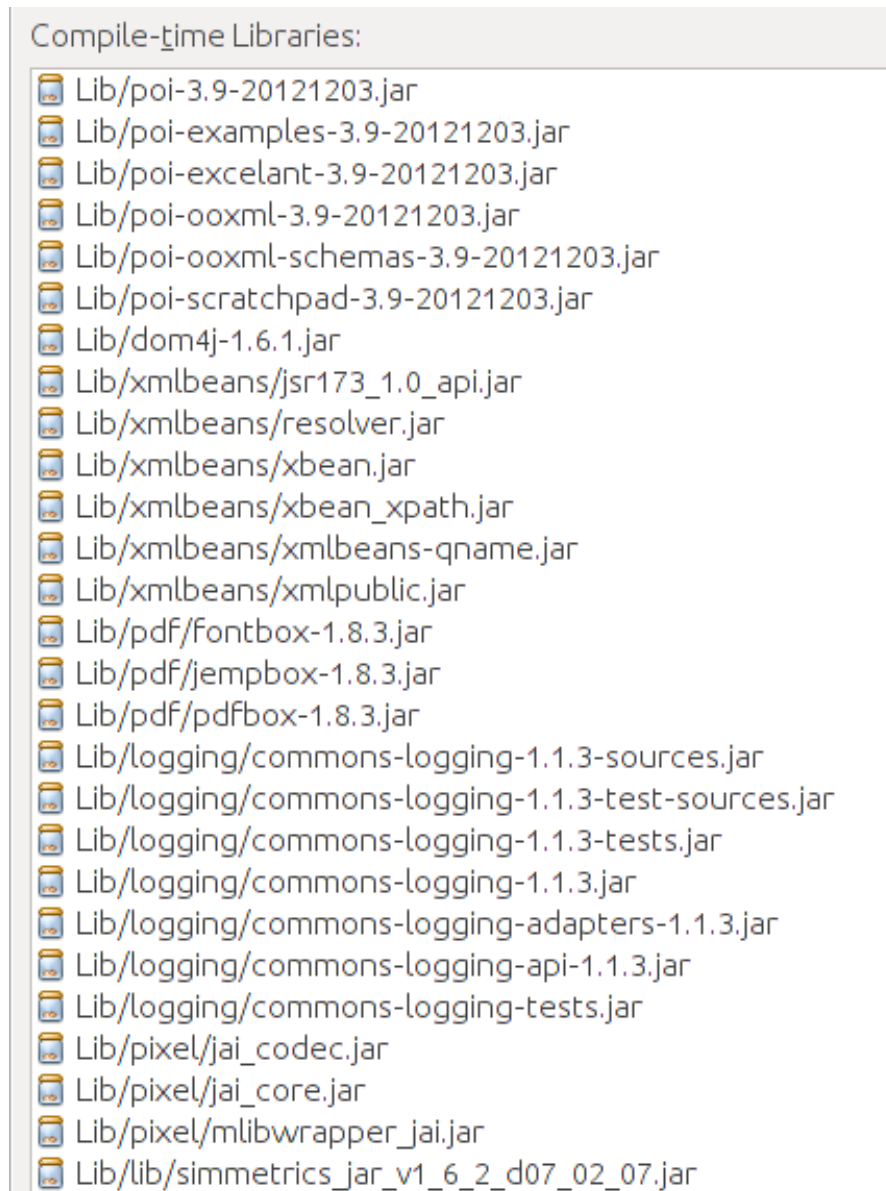


Figure 39: All the libraries used in the application.

### A.2.1 OCRMethods.java

Listing A.1: Library imports used in OCRMethods.java

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;
import java.awt.image.renderable.ParameterBlock;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.imageio.ImageIO;
import javax.media.jai.InterpolationNearest;
import javax.media.jai.JAI;
import javax.media.jai.iterator.RandomIter;
import javax.media.jai.iterator.RandomIterFactory;
import javax.swing.JFileChooser;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.util.PDFImageWriter;
import org.apache.pdfbox.util.PDFTextStripper;
import org.apache.poi.hwpf.HWPFDocument;
import org.apache.poi.hwpf.extractor.WordExtractor;
import org.apache.poi.xwpf.extractor.XWPFFWordExtractor;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import org.apache.poi.openxml4j.opc.OPCPackage;
import org.apache.poi.xssf.extractor.XSSFExcelExtractor;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
```

Listing A.2: Global variables in OCRMethods.java

```
private int counter = 1;
private Color[][] signature;
// The base size of the images.
private static final int baseSize = 300;
```

Listing A.3: Converter from \*.PDF to \*.png files in OCRMethods.java

```

public String convertPDFtoImage(String filePath){
    String IMAGE_FORMAT = "jpg";
    int DEFAULT_IMAGE_RESOLUTION = 300;
    String newFilePath = "";
    String outputPrefix = "";
    if(counter < 10){
        outputPrefix = "/mnt/hgfs/Application/Data/docs/converted/0" + counter;
        newFilePath = "/mnt/hgfs/Application/Data/docs/converted/0" + counter
        + "1.jpg";
        counter++;
    } else {
        outputPrefix = "/mnt/hgfs/Application/Data/docs/converted/" + counter;
        newFilePath = "/mnt/hgfs/Application/Data/docs/converted/" + counter
        + "1.jpg";
        counter++;
    }

    PDDocument document = null;
    try{
        document = PDDocument.load(filePath);
    } catch(Exception e){
        System.out.println("Failure in convertPDFtoImage: " + e);
    }

    PDFImageWriter imageWriter = new PDFImageWriter();
    try {
        //Replaced Integer.MAX_VALUE with the second 1 to
        //only have the code convert the first page.
        boolean success = imageWriter.writeImage(document, IMAGE_FORMAT, null, 1,
        document.close());
        if (!success) {
            //System.err.println("Error: no writer found for image format '"
            //+ IMAGE_FORMAT + "'");
            System.exit(1);
        }
    } catch(Exception e){
        System.out.println("Failure in imageWriter: " + e);
    }

    return newFilePath;
}

```



Listing A.4: OCR conversion code

```
public String convertPictureToText(ArrayList<String> pictureFilePaths){
    String[] input = new String[3];
    input[0] = "/bin/bash";
    input[1] = "-c";
    input[2] = "ocropus-nlbin ";
    for(int i = 0; i < pictureFilePaths.size(); i++){
        input[2] += pictureFilePaths.get(i).toString() + " ";
    }

    input[2] += "-o temp";

    try{
        Run(new String[]{" /bin/bash", "-c", "rm -rf temp"});
    } catch(Exception e){
        System.out.println("Removing folder: " + e);
    }
    Run(input);
    Run(new String[]{" /bin/bash", "-c", "ocropus-gpageseg 'temp/?????.bin.png'"
+ " --minscale 00"});
    String output = Run(new String[]{" /bin/bash", "-c", "ocropus-rpred"
+ " 'temp/????/???????.bin.png'"});
    return output;
}
```

Listing A.5: Code to run shell commands in Bash

```

public String Run(String[] cmd) {
    StringBuffer theRun = null;
    try {
        System.out.println("Command passed: " + cmd[2]);
        Process process = Runtime.getRuntime().exec(cmd);
        BufferedReader reader = new BufferedReader
            (new InputStreamReader(process.getInputStream()));
        int read;
        char[] buffer = new char[4096];
        StringBuffer output = new StringBuffer();
        while ((read = reader.read(buffer)) > 0) {
            theRun = output.append(buffer, 0, read);
        }
        reader.close();
        process.waitFor();

    } catch (IOException | InterruptedException e) {
        System.out.println("Failure in Run(): " + e);
    }

    if(theRun.toString() == null){
        return "The string is empty";
    }
    return theRun.toString().trim();
}

```

Listing A.6: Directory chooser for the document folder

```

public String directoryChooser(){
    JFileChooser chooser= new JFileChooser();
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    int choice = chooser.showOpenDialog(null);
    if (choice != JFileChooser.APPROVE_OPTION) return null;
    File chosenFile = chooser.getSelectedFile();
    return chosenFile.getAbsolutePath().toString();
}

```

Listing A.7: Method for sorting documents based on keyword

```

//This methods sort documentms based on a keyword. It also adds all the
//needed information about each document into the document object.
public ArrayList<FileInfo> sort(ArrayList<File> myFiles, String keyword){
    ArrayList<FileInfo> myFileInfo = new ArrayList();
    try {
        for(File f: myFiles){
            String filepath = f.getAbsolutePath();
            String text = "";

            if(keyword.equalsIgnoreCase("Enter keyword")){
                String imageName = convertPDFtoImage(filepath);

                //Comment this out when using image-to-image comparison
                text = documentReader(filepath);
                String imageName = convertPDFtoImage(filepath);

                /* This block enables image-to-image comparison with ocr
                ArrayList<String> orgimage = new ArrayList<>();
                orgimage.add(imageName);
                text = Regex(convertPictureToText(orgimage));
                */

                myFileInfo.add(new FileInfo(text, filepath,
                    f.getName(), imageName));
            } else {
                text = documentReader(filepath);
                Pattern.compile(Pattern.quote(keyword),
                    Pattern.CASE_INSENSITIVE).matcher(text).find();
                myFileInfo.add(new FileInfo(text, filepath, f.getName(),
                    convertPDFtoImage(filepath)));
            }
        }
    } catch (Exception e){
        System.out.println("Failure in sort: " + e);
    }
    return myFileInfo;
}

```

Listing A.8: Method for choosing picturefiles

```

public ArrayList<String> fileChooser(){
    JFileChooser chooser= new JFileChooser();
    chooser.setMultiSelectionEnabled(true);

    int choice = chooser.showOpenDialog(null);
    if (choice != JFileChooser.APPROVE_OPTION) return null;
    File[] files = chooser.getSelectedFiles();
    ArrayList<String> filePaths = new ArrayList<String>();
    for (File file : files) {
        filePaths.add(file.getAbsolutePath().toString());
    }
    return filePaths;
}

```

Listing A.9: Sorts the FileInfo objects based on the comparison method

```

public ArrayList sortFiles(ArrayList<FileInfo> dataArray){
    Collections.sort(dataArray);
    return dataArray;
}

```

Listing A.10: Method for calling the correct text extractor

```

public String documentReader(String filePath){
    String text = "";
    try{
        if (filePath.endsWith(".pdf")){
            text = readPDF(filePath);
        } else if (filePath.endsWith(".doc")){
            text = readDOC(filePath);
        } else if (filePath.endsWith(".docx")){
            text = readDOCX(filePath);
        } else if (filePath.endsWith(".xlsx")){
            text = readXLSX(filePath);
        } else {
            text = readText(filePath);
        }
    } catch (Exception e) {
        System.out.println("Fault in documentReader: " + e );
    }
    return text;
}

```

Listing A.11: Method for reading text files

```

public String readText(String filePath) throws Exception{
    BufferedReader br = null;
    String everything = "";
    try{
        br = new BufferedReader(new FileReader(filePath));
        StringBuilder sb = new StringBuilder();
        String line = br.readLine();

        while (line != null) {
            sb.append(line);
            sb.append("\n");
            line = br.readLine();
        }
        everything = sb.toString();
    }catch(IOException e){
        System.out.println("Failure in readText: " + e);
    }
    finally {
        br.close();
        return everything;
    }
}

```

Listing A.12: Method for extracting text from DOC files

```

public String readDOC(String filePath){
    File file = null;
    WordExtractor extractor = null ;
    String text = "";
    try {
        file = new File(filePath);
        FileInputStream fis = new FileInputStream(file.getAbsolutePath());
        HWPFDocument document = new HWPFDocument(fis);
        extractor = new WordExtractor(document);
        String [] fileData = extractor.getParagraphText();
        for (String fileData1 : fileData) {
            if (fileData1 != null) {
                text += fileData1;
            }
        }
    }
    catch(Exception e){
        System.out.println("Failure in readDOC: " + e);
    }
    return text;
}

```

Listing A.13: Method for extracting text from PDF files

```
public String readPDF(String filePath){
    PDDocument doc = null;
    String text = "";
    try {
        doc = PDDocument.load(filePath);
        PDFTextStripper myStripper = new PDFTextStripper();
        text = myStripper.getText(doc);
    } catch (Exception e) {
        System.out.println(e);
    } finally {
        if(doc != null ){
            try{
                doc.close();
            } catch (Exception e){
                System.out.println("Failure in readPDF: " + e);
            }
        }
    }
    return text;
}
```

Listing A.14: Method for extracting text from DOCX files

```
public String readDOCX(String filePath){
    XWPFDocument docx = null;
    String text = "";

    try {
        docx = new XWPFDocument(OPCPackage.openOrCreate(new File(filePath)));
    } catch (Exception e){
        System.out.println("Failure in readDOCX" + e);
    }
    XWPFWordExtractor wx = new XWPFWordExtractor(docx);
    text = wx.getText();

    return text;
}
```

Listing A.15: Method for extracting text from XLSX files

```

public String readXLSX(String filePath){
    XSSFWorkbook xlsx = null;
    String text = "";
    try {
        xlsx = new XSSFWorkbook(OPCPackage.openOrCreate(new File(filePath)));
        XSSFExcelExtractor ex = new XSSFExcelExtractor(xlsx);
        text = ex.getText();
    } catch(Exception e){
        System.out.println("Failure in readXLSX" + e);
    }
    return text;
}

```

Listing A.16: Levenshtein Distance calculation

```

//Code modified from http://stackoverflow.com/questions/95110/similarity-string-comparison-in-java
public int computeLevenshteinDistance(String s1, String s2){
    //Remove/add comment to enable/disable string sanitazion
    //s1 = s1.toLowerCase();
    //s2 = s2.toLowerCase();
    s1 = stringCleaner(s1.toLowerCase());
    s2 = stringCleaner(s2.toLowerCase());

    int[] costs = new int[s2.length() + 1];
    for(int i=0; i <= s1.length(); i++){
        int lastValue = i;
        for (int j = 0; j <= s2.length(); j++){
            if (i == 0){
                costs[j] = j;
            }else{
                if(j > 0){
                    int newValue = costs[j-1];
                    if(s1.charAt(i-1) != s2.charAt(j-1)){
                        //Minimun cost of the three operations
                        newValue = Math.min(Math.min(newValue,
                            lastValue), costs[j]) + 1;
                        costs[j-1] = lastValue;
                        lastValue = newValue;
                    }
                }
            }
        }
        if (i > 0){
            costs[s2.length()] = lastValue;
        }
    }
    return costs[s2.length()];
}

```

Listing A.17: Methods for creating a HashMap

```
private HashMap<String , Integer> frequencyMap(String input){
    HashMap<String , Integer> map = new HashMap<>();
    String[] split = input.split(" ");
    for (String s : split) {
        if (!map.containsKey(s.trim())) {
            map.put(s.trim(), new Integer("1"));
        } else {
            map.put(s.trim(), map.get(s.trim()) + new Integer(1));
        }
    }
    return map;
}
```

Listing A.18: Method for string sanitazion

```
private String stringCleaner(String input){
    //Removes everything that is not alphanumeric and spaces
    String newstr = input.replaceAll("[^A-Za-z0-9 ]", " ");
    //Removes single digits
    newstr= newstr.replaceAll("(?<!\S)\d(?!\S)", "");
    //Removes single letters
    newstr= newstr.replaceAll("(?<!\S)\w(?!\S)", "");
    //Replaces double space with a single space
    newstr = newstr.replaceAll("\s{2,}", " ");
    return newstr;
}
```

Listing A.19: Method for removing excess output from OCRopus

```
public String Regex(String input){
    try{
        String regex01 = "(...)(/)(\\d*)(/)(\\d*.*)(\\.)(...)
+ "(\\.)(...)\\s)(.)\\s)";
        Pattern p = Pattern.compile("(...)(/)(\\d*)(/)(\\d*.*
+ "(\\.)(...)\\.)(...)\\s)(.)\\s)");
        Matcher m = p.matcher(input);
        //The if statement below must be here for the sanitazion to work.
        //Also used for bugfixing.
        if (m.find()) {
            //System.out.println("MATCH FOUND");
            //System.out.println(m.group(0));
        }

        input = input.substring(input.indexOf(m.group()));
        input = input.replaceAll(regex01, "");
    } catch(Exception e){
        System.out.println("Failure in Regex: " + e);
    }
    return input;
}
```



Listing A.20: Method for calculating word frequency

```

public int countWordFrequency(String inputOrg, String inputOcr){
    String [] stringSplitOrg = inputOrg.toLowerCase().split(" ");
    String [] stringSplitOcr = inputOcr.toLowerCase().split(" ");

    ArrayList<String> cleanedOrgArray = new ArrayList<>();
    ArrayList<String> cleanedOcrArray = new ArrayList<>();

    String cleanedStringOrg = "";
    String cleanedStringOcr = "";

    //Adding the split string to a sanitized string array,
    //and creating a new sanitized string
    for(String s: stringSplitOrg){
        if(!stringCleaner(s).equalsIgnoreCase("") ||
            !stringCleaner(s).equalsIgnoreCase(" ")){
            cleanedOrgArray.add(s);
            cleanedStringOrg += s.trim() + " ";
        }
    }

    for(String s: stringSplitOcr){
        if(!stringCleaner(s).equalsIgnoreCase("") ||
            !stringCleaner(s).equalsIgnoreCase(" ")){
            cleanedOcrArray.add(s);
            cleanedStringOcr += s.trim() + " ";
        }
    }

    //Add/remove comments to enable/disable string sanitazion
    HashMap org = frequencyMap(cleanedStringOrg);
    HashMap ocr = frequencyMap(cleanedStringOcr);
    //HashMap org = frequencyMap(inputOrg.toLowerCase());
    //HashMap ocr = frequencyMap(inputOcr.toLowerCase());

    //Copying the missing keys from each set into the other set
    //with a value of 0 for both sets.
    for(int i = 0; i < cleanedOrgArray.size(); i++){
        if(!ocr.containsKey(cleanedOrgArray.get(i).trim())){
            ocr.put(cleanedOrgArray.get(i).trim(), 0);
        }
    }

    for(int i = 0; i < cleanedOcrArray.size(); i++){
        if(!org.containsKey(cleanedOcrArray.get(i).trim())){
            org.put(cleanedOcrArray.get(i).trim(), 0);
        }
    }
}

```

```
int changes = 0;

//Calculating difference
for(Object o: org.keySet()){
    changes += Math.abs((int)org.get(o) - (int)ocr.get(o));
}
return changes;
}
```

Listing A.21: Method for calculating cosine similarity

```

//Code modified from https://github.com/Simmetrics/simmetrics/blob/master/
src/uk/ac/shef/wit/simmetrics/similaritymetrics/CosineSimilarity.java

public float getCosineSimilarity(String inputOrg, String inputOcr) {

    //Remove/add comment to enable/disable string sanitazion/
    //String [] stringSplitOrg = inputOrg.toLowerCase().split(" ");
    //String [] stringSplitOcr = inputOcr.toLowerCase().split(" ");
    String [] stringSplitOrg = stringCleaner(inputOrg.toLowerCase()).split(" ");
    String [] stringSplitOcr = stringCleaner(inputOcr.toLowerCase()).split(" ");

    ArrayList<String> cleanedOrgArray = new ArrayList<>();
    ArrayList<String> cleanedOcrArray = new ArrayList<>();
    for(String s: stringSplitOrg){
        if(!stringCleaner(s).equalsIgnoreCase("") ||
        !stringCleaner(s).equalsIgnoreCase(" ")){
            if(s.length() != 1){
                cleanedOrgArray.add(s);
            }
        }
    }
    for(String s: stringSplitOcr){
        if(!stringCleaner(s).equalsIgnoreCase("") ||
        !stringCleaner(s).equalsIgnoreCase(" ")){
            if(s.length() != 1){
                cleanedOcrArray.add(s);
            }
        }
    }

    final Set<String> allTokens = new HashSet<>();
    allTokens.addAll(cleanedOrgArray);
    final int termsInString1 = allTokens.size();
    final Set<String> secondStringTokens = new HashSet<>();
    secondStringTokens.addAll(cleanedOcrArray);
    final int termsInString2 = secondStringTokens.size();

    //Combining sets
    allTokens.addAll(secondStringTokens);
    final int commonTerms =
        (termsInString1 + termsInString2) - allTokens.size();

    return (float) (commonTerms) / ((float) (Math.pow((float) termsInString1,
    0.5f) * Math.pow((float) termsInString2, 0.5f)));

}

```

Listing A.22: Method for making w-shingles

```
//Code modified from https://github.com/commoncrawl/commoncrawl-crawler/
blob/master/src/org/commoncrawl/util/Shingle.java

public Set<String> getShingles(String line) {
    int CHAR_GRAM_LENGTH = 3;
    //Add/remove comments to enable/disable string sanitazion
    //String clean = line;
    String clean = stringCleaner(line);
    HashSet<String> shingles = new HashSet<>();
    for (int i = 0; i < clean.length() - CHAR_GRAM_LENGTH + 1; i++) {
        //Extract the ngrams.
        String shingle = clean.toString().substring(i, i +
            CHAR_GRAM_LENGTH).toLowerCase();
        shingles.add(shingle);
    }
    return shingles;
}
}
```

Listing A.23: Method for calculating the Jaccard Index used by W-shingles

```
public float getJaccardIndex(Set<String> shinglesA, Set<String> shinglesB) {
    Set<String> intersection = new HashSet<>(shinglesA);
    intersection.retainAll(shinglesB);
    Set<String> union = new HashSet<>(shinglesA);
    union.addAll(shinglesB);

    float neumerator = intersection.size();
    float denominator = union.size();
    return neumerator / denominator;
}
}
```

Listing A.24: Method for calculating image signature

```
//Code modified from http://www.lac.inpe.br/JIPCookbook/
6050-howto-compareimages.jsp

private Color[][] calcSignature(RenderedImage i){
    // Get memory for the signature.
    Color[][] sig = new Color[5][5];
    // For each of the 25 signature values average the pixels around it.
    // Note that the coordinate of the central pixel is in proportions.
    float[] prop = new float[]
    {1f / 10f, 3f / 10f, 5f / 10f, 7f / 10f, 9f / 10f};
    for (int x = 0; x < 5; x++){
        for (int y = 0; y < 5; y++){
            sig[x][y] = averageAround(i, prop[x], prop[y]);
        }
    }
    return sig;
}
}
```

Listing A.25: Method for averaging the pixel values around a central point and return the average as an instance of Color. The point coordinates are proportional to the image.

```
private Color averageAround(RenderedImage i, double px, double py){
    // Get an iterator for the image.
    RandomIter iterator = RandomIterFactory.create(i, null);
    // Get memory for a pixel and for the accumulator.
    double[] pixel = new double[3];
    double[] accum = new double[3];
    // The size of the sampling area.
    int sampleSize = 15;
    int numPixels = 0;
    // Sample the pixels.
    for (double x = px * baseSize - sampleSize; x < px *
        baseSize + sampleSize; x++){
        for (double y = py * baseSize - sampleSize; y < py *
            baseSize + sampleSize; y++){
            iterator.getPixel((int) x, (int) y, pixel);
            accum[0] += pixel[0];
            accum[1] += pixel[1];
            accum[2] += pixel[2];
            pixel[2]).getRGB());
            numPixels++;
        }
    }
    // Average the accumulated values.
    accum[0] /= numPixels;
    accum[1] /= numPixels;
    accum[2] /= numPixels;
    return new Color((int) accum[0], (int) accum[1], (int) accum[2]);
}
```

Listing A.26: Method for calculating the distance between the signatures of an image and the reference one. The signatures for the image passed as the parameter are calculated inside the method.

```
private double calcDistance(RenderedImage other){
    // Calculate the signature for that image.
    Color[][] sigOther = calcSignature(other);
    // There are several ways to calculate distances between two vectors ,
    // we will calculate the sum of the distances between the RGB values of
    // pixels in the same positions.
    double dist = 0;
    for (int x = 0; x < 5; x++){
        for (int y = 0; y < 5; y++){
            int r1 = signature[x][y].getRed();
            int g1 = signature[x][y].getGreen();
            int b1 = signature[x][y].getBlue();
            int r2 = sigOther[x][y].getRed();
            int g2 = sigOther[x][y].getGreen();
            int b2 = sigOther[x][y].getBlue();
            double tempDist = Math.sqrt((r1 - r2) * (r1 - r2) + (g1 - g2)
                * (g1 - g2) + (b1 - b2) * (b1 - b2));
            dist += tempDist;
        }
    }
    return dist;
}
```

Listing A.27: Method for calculating the similarity between two pictures

```
public double calculateImageDistance(String originalFilePath ,
String pictureFilePath){
    double distance = 0;
    try{
        RenderedImage ref = rescale(ImageIO.read(new File(pictureFilePath)));
        //Calculate the signature vector for the reference.
        signature = calcSignature(ref);
        //Filepath to the converted image from the original document
        File myOrg = new File(originalFilePath);
        //For each image, calculate its signature and its distance from the
        //reference signature.
        RenderedImage rother = rescale(ImageIO.read(myOrg));
        distance = calcDistance(rother);
    } catch(Exception e){
        System.out.println("Failure in calculateDistance: " +e);
    }
    return distance;
}
```

Listing A.28: Method for rescaling images to 300x300

```

private RenderedImage rescale(RenderedImage i){
    float scaleW = ((float) baseSize) / i.getWidth();
    float scaleH = ((float) baseSize) / i.getHeight();
    // Scales the original image
    ParameterBlock pb = new ParameterBlock();
    pb.addSource(i);
    pb.add(scaleW);
    pb.add(scaleH);
    pb.add(0.0F);
    pb.add(0.0F);
    pb.add(new InterpolationNearest());

    return JAI.create("scale", pb);
}

```

Listing A.29: Method for writing data to a textfile

```

public void writeToFile(String filename, String text){
    BufferedWriter writer = null;
    try{
        writer = new BufferedWriter(new FileWriter(filename, true));
        writer.write(text);
    }
    catch(IOException e){
        System.out.println("Failure in writeToFile: " + e);
    }
    finally{
        try{
            if (writer != null){
                writer.close();
            }
        }
        catch (IOException e){
            System.out.println("Failure in writeToFile: " + e);
        }
    }
}

```

Listing A.30: Method for writing info to file for several objects

```

public void writeInformation(ArrayList<FileInfo> info, String filename){
    for(FileInfo f: info){
        writeToFile(filename, f.toStringMod());
    }
}

```

## A.2.2 FileInfo.java

Listing A.31: The FileInfo.java code with constructor getters and setters. And additional methods for sorting cloning and print output label

```

/**
 * Class containing all the info about the different documents in the document
 * folder and the best matching image file from the scanned documents,
 * and the scores from the different methods.
 */
public class FileInfo implements Comparable<FileInfo> {
    String text;
    String filepath;
    String filename;
    String imageFileName;
    String comparedTo;
    //The value sortBy is used by the application to
    //be able to sort the documents on the best matches
    //for each different method.
    int sortBy;
    double wordFrequency;
    double editDistance;
    float cosine;
    double pixelDistance;
    float w_shingle;

    //Constructor
    public FileInfo(String text, String filepath,
        String filename, String imageFileName) {
        this.text = text;
        this.filepath = filepath;
        this.filename = filename;
        //this.changePercentage = 100;
        this.wordFrequency = 100;
        this.pixelDistance = 100;
        this.w_shingle = 0;
        this.sortBy = 0;
        this.imageFileName = imageFileName;
    }

    //Get and set methods
    public String getImageFileName() {
        return imageFileName;
    }

    public void setImageFileName(String imageFileName) {

```



```
        this.imageFileName = imageFileName;
    }

    public void setCosine(float cosine) {
        this.cosine = cosine;
    }

    public void setSortBy(int sortBy) {
        this.sortBy = sortBy;
    }

    public String getComparedTo() {
        return comparedTo;
    }

    public void setComparedTo(String comparedTo) {
        this.comparedTo = comparedTo;
    }

    public double getPixelDistance() {
        return pixelDistance;
    }

    public void setPixelDistance(double pixelDistance) {
        this.pixelDistance = pixelDistance;
    }

    public double getWordFrequency() {
        return wordFrequency;
    }

    public void setWordFrequency(double wordFrequency) {
        this.wordFrequency = wordFrequency;
    }

    public double getEditDistance() {
        return editDistance;
    }

    public void setEditDistance(double editdistance) {
        this.editDistance = editdistance;
    }

    public void setW_shingle(float w_shingle) {
        this.w_shingle = w_shingle;
    }

    public String getText() {
        return text;
    }
}
```

```
}

public void setText(String text) {
    this.text = text;
}

public String getFilepath() {
    return filepath;
}

public void setFilepath(String filepath) {
    this.filepath = filepath;
}

public String getFilename() {
    return filename;
}

public void setFilename(String filename) {
    this.filename = filename;
}

public int getSortBy() {
    return sortBy;
}

public float getCosine() {
    return cosine;
}

public float getW_shingle() {
    return w_shingle;
}

//Method for cloning the objects , used to generate output for the experiment
@Override
public FileInfo clone(){
    FileInfo myNewFile = new FileInfo(getText(),
        getFilepath(), getFilename(), getImageFileName());
    myNewFile.setComparedTo(getComparedTo());
    myNewFile.setSortBy(getSortBy());
    myNewFile.setWordFrequency(getWordFrequency());
    myNewFile.setEditDistance(getEditDistance());
    myNewFile.setCosine(getCosine());
    myNewFile.setPixelDistance(getPixelDistance());
    myNewFile.setW_shingle(getW_shingle());
}
```

```
        return myNewFile;
    }

//This method is used to sort instances of this class in an array.
@Override
public int compareTo(FileInfo o) {
    if(sortBy == 0){
        //editDistance
        if(editDistance < o.editDistance){
            return -1;
        } else if (editDistance > o.editDistance) {
            return 1;
        } else {
            return 0;
        }
    }
    } else if (sortBy == 1){
        if(wordFrequency < o.wordFrequency){
            return -1;
        } else if (wordFrequency > o.wordFrequency) {
            return 1;
        } else {
            return 0;
        }
    }
    }else if(sortBy == 2){
        if(pixelDistance < o.pixelDistance){
            return -1;
        } else if (pixelDistance > o.pixelDistance) {
            return 1;
        } else {
            return 0;
        }
    }
    } else if(sortBy == 3){
        if(cosine > o.cosine){
            return -1;
        } else if (cosine < o.cosine) {
            return 1;
        } else {
            return 0;
        }
    }
    } else if(sortBy == 4){
        if(w_shingle > o.w_shingle){
            return -1;
        } else if (w_shingle < o.w_shingle) {
            return 1;
        } else {
            return 0;
        }
    }
    }
    return 0;
}
```

```

}

//toString method used in the GUI
@Override
public String toString() {
    return "\n\rFilepath: " + filepath + "\n\rFilename: " + filename
        + "\n\rPicture filename: " + imageFileName
        + "\n\rCompare to: " + comparedTo
        + "\n\rChecking with: " + sortBy
        //+ "\n\rNumber of characters in original text: " + text.length()
        + "\n\rLevenshtein Distance: " + editDistance
        //+ "\n\rBest Word frequency match in: " + bestFrequencyMatch
        + "\n\rWord frequency change: " + (wordFrequency)
        //+ "\n\rBest Pixel match in " + bestPixelMatch
        + "\n\rPixel change: " + pixelDistance
        + "\n\rCosine value: " + cosine
        + "\n\rw-shingle: " + w_shingle ;
}

//toString method used to create csv files for the experiment
public String toStringMod() {
    return "\r\n"+ filepath
        + "," + imageFileName
        + "," + comparedTo
        + "," + sortBy
        //+ "\n\rNumber of characters in original text: " + text.length()
        + "," + editDistance
        //+ "\n\rBest Word frequency match in: " + bestFrequencyMatch
        + "," + (wordFrequency)
        //+ "\n\rBest Pixel match in " + bestPixelMatch
        + "," + pixelDistance
        + "," + cosine
        + "," + w_shingle ;
}
}
}

```

### A.2.3 OCRUI.java

Listing A.32: Executed code for the Execute button

```

jButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e){
        //Checks if a picture has been chosen for conversion
        if (pictureFilePath.equals("")){
            pictureFilePath = myOCR.fileChooser();
        }

        //Gets the optional keyword for sorting
        String keyword = jTextField1.getText();

        myFileInfo = myOCR.sort(myFiles, keyword);

        //For timing the experiment
        long start = System.currentTimeMillis();
        try{
            //Checking to see if multiple documents are chosen.
            if (!jCheckBox1.isSelected()){
                String input = myOCR.convertPictureToText(pictureFilePath);
                String output = myOCR.Regex(input);
                scannedText.setText(output);

                //For each FileInfo object (one for each document)
                //Adds the calculated distances
                for (FileInfo f: myFileInfo){
                    String original = f.getText();
                    double distance =
                    myOCR.computeLevenshteinDistance(original, output);
                    double change = myOCR.countWordFrequency(original, output);
                    float cosine = myOCR.getCosineSimilarity(original, output);
                    f.setComparedTo(pictureFilePath.get(0).toString());
                    f.setWordFrequency(change);
                    f.setEditDistance(distance);
                    f.setCosine(cosine);
                    f.setW_shingle(myOCR.getJaccardIndex(
                    myOCR.getShingles(original), myOCR.getShingles(output)));
                    f.setPixelDistance(myOCR.calculateImageDistance(
                    f.getImageFileName(), pictureFilePath.get(0).toString()));
                }
            } else {
                //For multiple documents.
                for (String s: pictureFilePath){
                    ArrayList<String> myTemp = new ArrayList<>();
                    myTemp.add(s.toString());
                    String input = myOCR.convertPictureToText(myTemp);
                    String output = myOCR.Regex(input);
                    scannedText.setText(output);
                }
            }
        } catch (Exception e) {
            //Handle exception
        }
    }
}

```

```

for(FileInfo f: myFileInfo){
    String original = f.getText();
    double distance = myOCR.computeLevenshteinDistance(
        original, output);
    double change = myOCR.countWordFrequency(
        original, output);
    float cosine = myOCR.getCosineSimilarity(
        original, output);
    f.setComparedTo(s.toString());
    f.setWordFrequency(change);
    f.setEditDistance(distance);
    f.setCosine(cosine);
    f.setW_shingle(myOCR.getJaccardIndex(
        myOCR.getShingles(original), myOCR.getShingles(output)));
    f.setPixelDistance(myOCR.calculateImageDistance(
        f.getImageFileName(), s.toString()));
}
//The following if statement is here for the experiments part
//To clone the files and add them do arrays for later output.
if(true){
    //Sorting on Levenshtein distance
    for(FileInfo f: myFileInfo){
        f.sortBy = 0;
    }

    myFileInfo = myOCR.sortFiles(myFileInfo);
    myMethod0.add(myFileInfo.get(0).clone());
    myMethod5.add(myFileInfo.get(1).clone());

    //Sorting on Word frequency
    for(FileInfo f: myFileInfo){
        f.sortBy = 1;
    }
    myFileInfo = myOCR.sortFiles(myFileInfo);
    myMethod1.add(myFileInfo.get(0).clone());
    myMethod6.add(myFileInfo.get(1).clone());

    //Sorting on Pixel distance
    for(FileInfo f: myFileInfo){
        f.sortBy = 2;
    }
    myFileInfo = myOCR.sortFiles(myFileInfo);
    myMethod2.add(myFileInfo.get(0).clone());
    myMethod7.add(myFileInfo.get(1).clone());

    //Sorting on Cosine similarity
    for(FileInfo f: myFileInfo){
        f.sortBy = 3;
    }
}

```

```

    }
    myFileInfo = myOCR.sortFiles(myFileInfo);
    myMethod3.add(myFileInfo.get(0).clone());
    myMethod8.add(myFileInfo.get(1).clone());

    //Sorting on W-shingle score
    for(FileInfo f: myFileInfo){
        f.sortBy = 4;
    }
    myFileInfo = myOCR.sortFiles(myFileInfo);
    myMethod4.add(myFileInfo.get(0).clone());
    myMethod9.add(myFileInfo.get(1).clone());
}
}
}
} catch (Exception ex){
    System.out.println("Failure in Execute: " + ex);
}

System.out.println((System.currentTimeMillis() - start) / 1000);
myFileInfo = myOCR.sortFiles(myFileInfo);

//Writes the best match for each document info
//to cvs files for each method.
myOCR.writeInformation(myMethod0,"00-Levenshtein.txt");
myOCR.writeInformation(myMethod1,"01-WordFrequency.txt");
myOCR.writeInformation(myMethod2,"02-Pixel.txt");
myOCR.writeInformation(myMethod3,"03-Cosine.txt");
myOCR.writeInformation(myMethod4,"04-wshingle.txt");

//Writes the second best match for each document info
//to cvs files for each method.
myOCR.writeInformation(myMethod5,"05-Levenshtein.txt");
myOCR.writeInformation(myMethod6,"06-WordFrequency.txt");
myOCR.writeInformation(myMethod7,"07-Pixel.txt");
myOCR.writeInformation(myMethod8,"08-Cosine.txt");
myOCR.writeInformation(myMethod9,"09-wshingle.txt");

}
});

```

Listing A.33: Executed code for the Edit distance button

```

jButton5.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        for(FileInfo f: myFileInfo){
            //System.out.println(f.getFilename() + " vs." + pictureFilePath.get(0));

            f.sortBy = 0;
        }
        //Sorting the files based on change Levenshtein distance
        myFileInfo = myOCR.sortFiles(myFileInfo);
        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(0).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(0).toString());

    }
});

```

Listing A.34: Executed code for the Frequency button

```

jButton6.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        for(FileInfo f: myFileInfo){
            //System.out.println(f.getFilename() + " vs." + pictureFilePath.get(0));

            f.sortBy = 1;
        }

        //Sorting the files based on Word frequency
        myFileInfo = myOCR.sortFiles(myFileInfo);
        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(0).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(0).toString());

    }
});

```



Listing A.35: Executed code for the Cosine button

```

jButton7.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        for(FileInfo f: myFileInfo){
            //System.out.println(f.getFilename() + " vs." + pictureFilePath.get(0));

            f.sortBy = 3;
        }

        //Sorting the files based on change percentage
        myFileInfo = myOCR.sortFiles(myFileInfo);
        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(0).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(0).toString());

    }
});

```

Listing A.36: Executed code for the W\_shingle button label

```

jButton10.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        for(FileInfo f: myFileInfo){
            //System.out.println(f.getFilename() + " vs." + pictureFilePath.get(0));

            f.sortBy = 4;
        }

        //Sorting the files based on change percentage
        myFileInfo = myOCR.sortFiles(myFileInfo);
        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(0).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(0).toString());

    }
});

```

Listing A.37: Executed code for the Image Similarity button

```

jButton11.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        for(FileInfo f: myFileInfo){
            //System.out.println(f.getFilename() + " vs." + pictureFilePath.get(0));

            f.sortBy = 2;
        }

        //Sorting the files based on change percentage
        myFileInfo = myOCR.sortFiles(myFileInfo);
        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(0).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(0).toString());

    }
});

```

Listing A.38: Executed code for the "arrow left" button

```

jButton8.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        index--;
        if(index < 0){
            index = myFileInfo.size() -1;
        } else if(index >= myFileInfo.size()){
            index = index % myFileInfo.size();
        }

        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(index).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(index).toString());
        //System.out.println(index);

    }
});

```

Listing A.39: Executed code for the "arrow right" button

```

jButton9.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        index ++;
        index = index % myFileInfo.size();
        //Displaying the text from the best matching file
        originalText.setText(myFileInfo.get(index).getText());
        //Displaying info about the best matching file
        commandText.setText(myFileInfo.get(index).toString());
        //System.out.println(index);
    }
});

```

Listing A.40: Executed code for the Choose folder button

```

jButton2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //Choosing a folder
        documentDirectoryPath = myOCR.directoryChooser();

        File f = new File(documentDirectoryPath);
        //Makes a list of all the files/folders in that folder
        ArrayList<File> fileListing = new ArrayList<File>(Arrays.asList(f.listFiles()));

        //Adds only files to the actual list
        myFiles.clear();
        for(File fi: fileListing){
            if(fi.isFile()){
                myFiles.add(fi);
            }
        }
    }
});

```

Listing A.41: Executed code for the Choose picture button

```

jButton3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //Chooses a picturefile to run OCR on.
        pictureFilePath = myOCR.fileChooser();
    }
});

```

Listing A.42: Executed code for the Sort button

```
jButton4.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        //Check to see if a document folder has been chosen.  
        if(documentDirectoryPath.equals("")){  
            documentDirectoryPath = myOCR.directoryChooser();  
        }  
        String keyword = jTextField1.getText();  
        //Sort documents based on keyword  
        myFileInfo = myOCR.sort(myFiles, keyword);  
  
        commandText.setText("Sorting done");  
    }  
});
```

### A.2.4 TestRun.java

Listing A.43: Code for getting the Levenshtein distance between the document and the OCR from the matching scanned document.

```

ArrayList<String> pictureFilePath;
ArrayList<String> documentFilePath;

Testrun myTestRun = new Testrun ();

try{
    documentFilePath = myTestRun.fileChooser ();
    pictureFilePath = myTestRun.fileChooser ();
    for(int i = 0 ; i < pictureFilePath.size (); i++){
        ArrayList<String> myTemp = new ArrayList <>();
        myTemp.add(pictureFilePath.get(i).toString ());

        String original =
myTestRun.documentReader(documentFilePath.get(i).toString ());
String ocr = myTestRun.convertPictureToText(myTemp);
String cleaned = myTestRun.Regex(ocr);
double distance =
myTestRun.computeLevenshteinDistance(original , cleaned);
double change = (distance / original.length()) * 100;
String output = documentFilePath.get(i).toString () + ","
        + pictureFilePath.get(i).toString () + "," + change
        + "," + distance + "," + original.length() + "\r\n";
myTestRun.writeToFile ("percentage.txt", output);

    }

} catch (Exception e){
    System.out.println ("Failure in test code" + e);
}
}

```

Listing A.44: Code for a one-to-one image-to-image comparison with OCR.

```

ArrayList<String> pictureFilePath;
ArrayList<String> documentFilePath;
ArrayList<File> myFiles = new ArrayList();
ArrayList<FileInfo> myFileInfo = new ArrayList();
Testrun myTestRun = new Testrun();

try{
    documentFilePath = myTestRun.fileChooser();
    pictureFilePath = myTestRun.fileChooser();

    //Image to image cleaned/noclean
    System.out.println(documentFilePath.size());
    System.out.println(pictureFilePath.size());
    for(int i = 0 ; i < pictureFilePath.size(); i++){
        ArrayList<String> myTemp01 = new ArrayList<>();
        myTemp01.add(documentFilePath.get(i).toString());
        ArrayList<String> myTemp02 = new ArrayList<>();
        myTemp02.add(pictureFilePath.get(i).toString());

        String imageORG = myTestRun.convertPictureToText(myTemp01);
        String regex01 = myTestRun.Regex(imageORG);

        String imageOCR = myTestRun.convertPictureToText(myTemp02);
        String regex02 = myTestRun.Regex(imageOCR);

        String cleaned01 = myTestRun.stringCleaner(regex01);
        String cleaned02 = myTestRun.stringCleaner(regex02);

        int levnoclean = myTestRun.computeLevenshteinDistance(
            regex01, regex02);
        int levclean = myTestRun.computeLevenshteinDistance(
            cleaned01, cleaned02);

        int wordnoclean = myTestRun.countWordFrequency(
            regex01, regex02);
        int wordclean = myTestRun.countWordFrequency(
            cleaned01, cleaned02);

        double cosinenoclean = myTestRun.getCosineSimilarity(regex01, regex02);
        double cosineclean = myTestRun.getCosineSimilarity(cleaned01, cleaned02);

        Set wnoclean01 = myTestRun.getShingles(regex01);
        Set wclean01 = myTestRun.getShingles(regex02);

```

```
float shingles01 = myTestRun.getJaccardIndex(wnoclean01, wclean01);

Set wnoclean02 = myTestRun.getShingles(cleaned01);
Set wclean02 = myTestRun.getShingles(cleaned02);

float shingles02 = myTestRun.getJaccardIndex(wnoclean02, wclean02);

String output = myTemp01.get(0).toString() + "," +
myTemp02.get(0).toString() + "," + levnoclean + "," +
levclean + "," + wordnoclean + "," + wordclean + "," +
cosinenoclean + "," + cosineclean + "," + shingles01 + "," +
shingles02 + "," + regex01.length() + "," +
regex02.length() + "," + cleaned01.length() + "," +
cleaned02.length() + "\r\n";

myTestRun.writeToFile("imagetoimage.txt", output);
}
```

### A.3 Extended Results

#### A.3.1 Document to Image

The following results are from obtained through the comparison of text extracted from the scanned document through OCR and the extracted text from the original documents.

Table 6: Results without text sanitation, document to image

Method	Accuracy
Levenshtein	84%
Word frequency	79%
Cosine similarity	100%
W-shingle	100%

Table 7: Results with text sanitation, document to image

Method	Accuracy
Levenshtein	91%
Word frequency	80%
Cosine similarity	100%
W-shingle	100%

#### A.3.2 Image to Image

The following results are from obtained through the comparison of text extracted from the scanned document and the converted original document, through OCR.

Table 8: Results with text sanitation, image to image

Method	Accuracy
Levenshtein	100%
Word frequency	74%
Cosine similarity	91%
W-shingle	100%

Table 9: Results with text sanitation, image to image

Method	Accuracy
Levenshtein	91%
Word frequency	80%
Cosine similarity	100%
W-shingle	100%



### A.3.3 Pixel comparison

Results from the pixel comparison methods.

Table 10: Results for the pixel comparison method

Method	Accuracy
Pixel comparison	21%

### A.3.4 One-to-one comparison

Table 11: List of Levenshtein distance between the OCR conversion of the scanned document and the original document. No text sanitation.

Document name	Number of changes
01.pdf	158
02.pdf	688
03.pdf	88
04.pdf	223
05.pdf	519
06.pdf	87
07.pdf	95
08.pdf	1191
09.pdf	130
10.pdf	90
11.pdf	226
12.pdf	369
13.pdf	299
14.pdf	201
15.pdf	263
16.pdf	309
17.pdf	137
18.pdf	30
19.pdf	230
20.pdf	82
21.pdf	100
22.pdf	327
23.pdf	90
24.pdf	32
25.pdf	1526
26.pdf	287
27.pdf	127
28.pdf	66
29.pdf	101
30.pdf	714
31.pdf	57
32.pdf	921
33.pdf	1088

34.pdf	190
35.pdf	374
36.pdf	891
37.pdf	101
38.pdf	110
39.pdf	364
40.pdf	460
41.pdf	162
42.pdf	338
43.pdf	105
44.pdf	97
45.pdf	358
46.pdf	615
47.pdf	1860
48.pdf	320
49.pdf	552
50.pdf	399
51.pdf	478
52.pdf	580
53.pdf	278
54.pdf	350
55.pdf	94
56.pdf	220
57.pdf	70
58.pdf	183
59.pdf	506
60.pdf	310
61.pdf	305
62.pdf	282
63.pdf	168
64.pdf	225
65.pdf	133
66.pdf	108
67.pdf	696
68.pdf	72
69.pdf	206
70.pdf	819
71.pdf	88
72.pdf	1999
73.pdf	197
74.pdf	371
75.pdf	204
76.pdf	206
77.pdf	235
78.pdf	100
79.pdf	28

80.pdf	535
81.pdf	312
82.pdf	138
83.pdf	473
84.pdf	245
85.pdf	137
86.pdf	102
87.pdf	694
88.pdf	166
89.pdf	149
90.pdf	240
91.pdf	303
92.pdf	442
93.pdf	143
94.pdf	576
95.pdf	711
96.pdf	326
97.pdf	1164
98.pdf	829
99.pdf	1473
100.pdf	570

## A.4 Dataset

All this data shows number of documents out of the 100 that contains the characteristics. The total number of occurrences will exceed a 100 because one document can have several instances of the same font but in different font size and different fonts.

Table 12: Most occurring fonts

Font type	# of occurrences
American Typewriter	1
Arial	24
Book Antiqua	1
Calibri	51
Cambria	12
Courier New	4
Franklin Gothic Book	12
Georgia	1
Helvetica-Bold	1
Hobo Std	1
Informal Roman	1
Lucida Sans	2
Monotype Corsiva	1
Palatino	1
Tahoma	3
Times New Roman	27
Trebuchet MS	1
Verdana	2

Table 13: Most occurring font sizes

Font size	# of occurrences
6	1
6.5	1
7	1
8	8
8.5	1
9	8
9.5	1
10	36
10.5	1
11	45
12	48
13	3
14	22
15	1
15.5	1
16	9
18	11
20	4
22	2
24	2
26	6
36	2

Table 14: Other characteristics

Characteristics	# of occurrences
Graphs	6
Pictures	33
Tables	21
Lists	45
Bold	74
Underline	24
Italic	17
Color	49
Average word count	249
Average number of characters	1725

Table 15: Fonts and sizes

<b>Font and size</b>	<b># of occurrences</b>
Arial 6	1
Arial 8	2
Arial 8.5	1
Arial 9	4
Arial 9.5	1
Arial 10	10
Arial 11	3
Arial 12	16
Arial 13	1
Arial 14	3
Arial 15	1
Arial 14	1
Arial 16	2
Arial 18	1
Arial 20	2
Arial 24	1
Times New Roman 8	1
Times New Roman 9	1
Times New Roman 10	2
Times New Roman 11	9
Times New Roman 12	17
Times New Roman 13	2
Times New Roman 14	5
Times New Roman 16	3
Times New Roman 18	3
Times New Roman 22	1
Calibri 8	4
Calibri 9	3
Calibri 10	24
Calibri 11	33
Calibri 12	10
Calibri 14	10
Calibi 16	3
Calibri 18	5
Calibri 20	2
Calibri 22	1
Helvetica-Bold 14	1
Monotype Corsiva 14	1
Courier New 10	1
Courier New 20	1
Courier New 12	3
Courier New 14	1
Courier New 18	2
Courier New 36	1

Informal Roman 24	1
Georgia 12	1
American Typewriter 36	1
American Typewriter 12	1
American Typewriter 15.5	1
Tahoma 10.5	1
Tahoma 8	1
Tahome 10	2
Cambria (Body) 14	1
Cambria 16	3
Cambria (Headings) 26	7
Book Antiqua (Headings) 32	1
Verdana 7	1
Verdana 6.5	1
Verdana 9	2
Verdana 10	1
Verdana 12	1
Trebuchet MS 14	1
Trebuchet MS 11	1
Franklin Gothic Book 14	1
Lucida Sans 10	1
Hobo Std 18	1
Palatino 12	1





## A.5 Additional information

### A.5.1 OCRopus testpage

---



---

## *BOOK REVIEW*

---



---

### A Mine of Information on Libraries

Jaideep Sharma

*(Department of Library and Information Science, Kurukshetra University, Kurukshetra)*

---

**Mohamed Taher, 2002, Libraries in India's National Developmental Perspective: A Saga of Fifty Years since Independence, New Delhi, Concept Publishing Co, 569 pp, Rs. 900/-**

---

The importance of libraries is evident in the oft-quoted phrase, 'library is the heart of an institution'. It is charged with the responsibility of storing, organising, disseminating, preserving and passing on to future generations the cultural and intellectual wealth of the society. It serves an important function in teaching, research and satisfying other information needs of the people. In spite of this we do not find a library culture in our country, people waiting to get a seat in the library, lectures, for story-telling sessions or exhibitions being frequently organized in the libraries. We measure the growth of our libraries by the number of the members, the days of opening, and the number of users visiting the library in the year. Other figures are also quoted which are at times exaggerated. Why is it so? How can we improve it? A look into the past helps us to evaluate our plans and working, judge our performance, find out our faults and the scope for improvement in the system and plan for the future.

There have been attempts made by individuals, towards presenting history of libraries and librarianship in India. We have books, articles and theses published earlier, but the book under review is a very earnest attempt in this direction done, in a different way, adding a valuable source to the area of study. It is based on an extensive review of the literature, use pattern of libraries and present position of libraries by collecting data from librarians as well as users and incorporating their views, observations, suggestions etc. It presents a lot of anecdotes in Ranganathan style that makes the work interesting.

Dr. Mohammed Taher has undertaken a gigantic task to assess the role of libraries in the national development, and that of the nation in the development of libraries, in the 50 years of independence. The presentation of the subject has been spread over nine chapters. After very clearly stating the purpose and scope of the book he presents a background to the book in chapter one which discusses the history of libraries, along with social and intellectual histories. The history of book, importance of encyclopaedia and its growth and role through the times form a pertinent discussion of library development. It also includes a definition of information and its role in development. The chapter closes with a discussion of the survey undertaken for the study.

Chapter 2 is devoted to library history and historiography (writing of history) in India. It presents a

review of literature on the subject. The author ascribes the lack of history of libraries to its lying scattered as bits and pieces in an incoherent and disjointed manner. He informs that it was only in the medieval period that a historical consciousness grew in the society. The rest of the chapter focuses on the history of script writing and book promotion programmes. There is discussion on the literature of library science and other areas as well as individuals as source material on the history of libraries.

Chapters 3 and 4 are devoted to library history in the pre and post independent eras, respectively. Discussions on social movements, how they formed, succeeded or failed follows library movements in different states of India and the West. The influence of movements in the West on those in India has also been discussed. The next part of the chapter is devoted to the role of books libraries, friends of libraries and the government. Information networks and programmes, e.g. INFLIBNET and NISSAT, have also been dealt in brief. The chapters also provide data regarding growth in book production in different states annually, the number of libraries in different states as well as the number of books received under the National Library Act.

The fifth chapter is devoted to the analysis and growth of library movement based on the response to the questionnaire distributed to librarians and users. Comments and suggestions given by the users and librarians have also been included. There is a chapter devoted to the preservation and maintenance techniques used by libraries and finally a concluding chapter on libraries of the future. Annual state library bills have been appended as a useful source of information. There is a comprehensive bibliography classified under subheads at the end.

The book provides a mine of data and information. However, some data are quite old e.g. the entry on courses in library science in Kurukshetra University mentions only diploma course. The university is providing all courses including PhD. Treatment given to modern libraries, networks and systems is meagre. However, the work presents an exhaustive coverage of the topic otherwise not available elsewhere. It would go a long way in providing an impetus to the planning and improving the library and information system of the country. □

### A.5.2 Transcript of Q&A session with Økokrim

This is an extract from a transcript from the Q&A part of a presentation the author did at Økokrim the 14.03.14 [5]. Only the relevant pieces of feedback are presented here.

09:00 into the video

**Thomas (Økokrim):** Could also be of interest of preprocessing, like if you have a lot of logos and stuff, then, maybe the pixel one should have sort of a higher weighting factor, if you have a lot of text, then you should maybe look at shingles. Shingles seems to be the leading tool at the moment to compare text.

12:05 into the video

**Thomas (Økokrim):** When it comes to the last question I will turn the word over to Egil, I think it's, more like you will see later when we walk around, that we provide access to the digital data for the investigators. It's basically up to them at the moment if they have some paper documents they want to find in the digital evidence, and I guess then you mostly use search words?

**Egil (Økokrim):** search words, yes.

**Me:** So keyword search, so you import a folder with documents into the tool you use and just search for well "fraud" or "money".

**Egil (Økokrim):** Yeah, try to find similar data and documents. And it's also, when you find physical evidence, it's dated and it's important for us to go into the electronic information and find the same document and look when it's created, because that is often different than the , than the using searching words by word.

**Thomas (Økokrim):** Get the correct date from an electronic document, was created , then you see that they backdated the document. Like the original digital document was created in word 2010 and it is dated 1998.

**Thomas (Økokrim):** We have a small project ourselves, where we will be using shingles to look for.. when we start an investigation , we typically get some papers from someone, and then the idea is to scan that and OCR treat it, and put it into the, and look for it using shingles.