A Watermark Recognition System

Final Report: An Approach to Matching Similar Watermarks

Group 18B

Diana Banță, Sydney Kho, Anna Lantink,
Alexandru Marin, Vladimir Petkov

Client: Dr. Martin Skrodzki

Coach: Dr. Zhengjun Yue

Teaching Assistant: Ana Bălțăreţu

Computer Science and Engineering

Technische Universiteit Delft

The Netherlands

June 25, 2023

Preface

This report is the product of the second year Software Project for Computer Science and Engineering at the Technical University of Delft. To complete the Software Project, the group chose to create a system that could match watermarks on similarity. The goal of this project was to make a prototype system that could find similarities between watermarks. Ideally, this system will be expanded in the future. This report details the process of producing this system. Specifically, the following report covers the process of designing the product, outlining the requirements, as well as researching, implementing, and evaluating the system. This report seeks to provide a general overview of all the steps, so in-depth technical descriptions are not provided. The sources[1] used are cited, so for further information these can be referred to.

We would like to thank our client, Dr. Martin Skrodzki, our coach, Dr. Zhengjun Yue, and our teaching assistant, Ana Bălţăreţu, for their continual support throughout this project. They have provided us with useful feedback that allowed us to keep this project organized and on track. We would also like to thank Drs. Regina Hoffmann for her guidance in writing this report. Additionally, we would like to acknowledge that ChatGPT[2] was used occasionally in this project. For more information on the usage of ChatGPT, please refer to Appendix 1. Finally, we would like to thank those at the German Museum of Books and Writing and the Bernstein Project for their hard work in digitizing their large archive of watermarks. Without these watermarks, we would not have been able to build our system as effectively as we have. As a group, we are very grateful for the opportunity to work on such an engaging project, and we feel that we have significantly benefited from this experience.

[1] The IEEE referencing style has been used.
[2] https://openai.com/blog/chatgpt

Summary

Watermarks are invaluable tools for historians and researchers. Based on a document's watermark, it can be determined where and when it was created [1]. However, to identify a watermark, a specialist must be contacted who will manually search for matching watermarks in their archive. This can take a significant amount of time, therefore this process would benefit greatly from automation.

This report aims to discuss the process of creating a system for finding similar watermarks. To do so, the problem was first analyzed by researching the history and use of watermarks. Based on the discoveries made, constraints are put on the problem to make it more feasible. For example, all watermark images must contain a watermark, and be cropped around the watermark. Existing research and technologies are also taken into account to further guide the problem definition. Based on this clearer definition of the problem in addition to an interview with the client, concrete requirements were created for the final product.

Based on the requirements, a design was put together for a system that can find similar watermarks, consisting of harmonization, feature extraction, and similarity matching. Harmonization involved isolating the watermark by using many different image processing techniques such as morphological operations [2], wavelet denoising [3] [4], and Sauvola thresholding [5]. Feature extraction involved extracting useful information from the isolated watermarks using SIFT [6], Hu moments [7], and Zernike moments [8], and finally similarity matching would use this information to score how similar any pair of watermarks is.

A graphical user interface was also designed to allow the user to interact with the system and to make it easier to use. This gives the option of using the system either manually, or automatically through the command line. The manual pipeline resulted in better results, as the user was able to compensate for mistakes made by the system. To avoid recalculating the features of known watermarks with every query, a database also has to be built. This can also be done automatically, but the user interface can be used to manually build the database for better accuracy.

Ethical implications of the system were discussed, as well as ways to mitigate them. Specifically, the ethics of the dataset of watermarks was discussed, as well as potential ways the system can be abused. Overall, it was determined that these concerns were sufficiently mitigated for this project.

Next, the system was evaluated. This was done by running the pipeline and looking at the ranking it produced, where a result is counted as correct if a corresponding image is in the top 10% of the dataset. Accuracy for the automatic pipeline with the automatic database pipeline was 42%, and with the manual database 41%. Accuracy for the manual pipeline with the automatic database was 45%, and with the manual database 53%. As expected, incorporating user interaction with the manual pipeline and database improved scores. It was also found that improving the quality of watermarks in the dataset improve accuracy results to 82%.

Overall, the system works sufficiently, especially given the expected use case of building the database manually and having the user interact with the pipeline. Nonetheless, it is also clear there is still room for improvement. The existing process for similarity matching is somewhat arbitrary and could be optimized by using a machine learning model. The harmonization for images of the original watermark also rarely works perfectly without help. Finally, the system could be further extended with optional goals that were not achieved, such as segmenting the watermarks into parts that can be assigned semantic meaning.

Contents

1 Introduction

Sometimes the greatest secrets hide exactly where one never expects them to be. With a piece of paper, one may focus on the words or pictures but may miss something nearly invisible: its watermark. Watermarks are patterns, embedded in paper, that can only be seen when shining a light from a specific angle. Historically, watermarks were used by manufacturers to identify their paper, a type of insignia [1]. Analyzing a document's watermark can provide details about the place and time that a document was produced. For this reason, watermarks can provide invaluable information for historians and researchers. To identify a watermark, a specialist must be contacted who will then manually comb through archives searching for matching artifacts. This can take a significant amount of time, which is the problem that this project seeks to solve.

The goal of this report is to discuss the design, implementation, and evaluation of a system that, when given a watermark as input, can return similar watermarks. The overall design of the system is as follows. The watermark image will be processed such that it can be isolated from its surroundings. After the watermark outline is isolated, it will be compared to a database of other watermarks. The most similar ones to the input will then be returned. Since the product may be used by researchers with little technical knowledge, it is necessary that the product is simplistic and easy to use. After the design and implementation of the system are covered, the system will be evaluated on its accuracy.

Several constraints are considered to ensure that this project can be completed in the given time. One constraint is that input images are expected to fulfill certain properties. For example, the watermark images should be cropped around the watermark, and watermarks should not be obscured by text. Another constraint on the system is that the database is not expected to have security.

The structure of this report is as follows. In Chapter 2, the problem of finding similar watermarks will be analyzed. Chapter 3 will outline the requirements for the system, which will be followed by the design of the watermark similarity system in Chapter 4. Chapters 5 and 6 discuss how the system has been implemented. Specifically, Chapter 5 explains how watermark similarity is calculated, and Chapter 6 shows how the user can interface with the system. The ethical implications of this system are analyzed in Chapter 7. Chapter 8 discusses the accuracy of finding similar watermarks with this system, as well as its limitations. Finally, Chapters 9 and 10 conclude this report and recommend improvements that can be made.

2 Analysis of Challenges in Matching Similar Watermarks

In this section, the problem of watermark similarity will be analyzed. In Section 2.1 the context behind watermark similarity is explained, and in Section 2.2 the challenges of watermark analysis are discussed. After defining the problem, a solution for finding similar watermarks is proposed in Section 2.3. The constraints on the system are explicitly defined in Section 2.4. Additionally, existing solutions for watermark similarity are reviewed in Section 2.5, and finally, in Section 2.6 this project's stakeholders are mentioned.

2.1 Context for Watermarks

In order to analyze watermarks, it should first be understood what watermarks are and what their purpose is. Watermarks are small images that are embedded in paper and are used to identify the paper's manufacturers [1]. They are primarily found in historical documents since modern paper is no longer produced with watermarks. Watermarks can be seen by shining light through the paper from a certain angle. For this project, watermark images have been provided by the German Museum of Books and Writing, who have digitized and uploaded them for The Bernstein Project [9]. These digitized watermarks can be categorized into two types. The first is "untraced" watermarks, which are scans taken directly from the watermarked paper (Fig. 1). The second is "traced" watermarks, which are scans of tracings of watermarks (Fig. 2). Evidently, the tracings of watermarks are often much clearer than the direct scans.

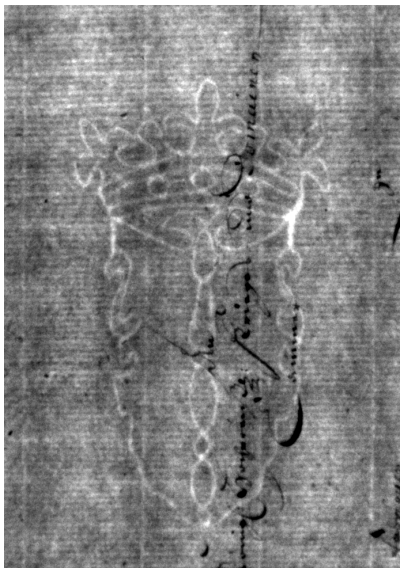

Figure 1: An 'untraced' watermark.

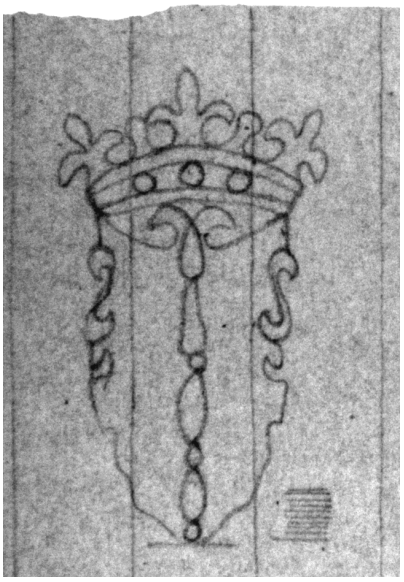

Figure 2: A traced watermark.

2.2 Challenges in Watermark Analysis

Since watermarks can provide important information on the location or time that a piece of paper was produced, they can be invaluable for researchers. When a researcher comes across a historical document with a watermark, they can analyze a watermark to understand the document's origins and context. To get this information researchers must contact experts, who must manually go through an

archive of watermarks. This can take days if not weeks. Producing a system to automatically find similar watermarks could bring this time down to mere minutes. With such a system, a researcher could quickly find similar watermarks and thus be able to get more clarity and detail on their watermark.

2.3 A Solution for Finding Similar Watermarks

The aim of the project is to develop a tool that, given an image of a watermark, outputs similar watermark images. This tool will focus on a novel approach to matching similar watermarks by utilizing traditional image processing techniques instead of machine learning. The approach would make it easier to expand the system's watermark database.

Ideally, this tool will be expanded further into a more complex system. The end goal is to provide a tool for the German Museum of Books and Writing that will allow a user to upload a watermark and compare it with watermarks that are already in the database of the museum.

2.4 Constraints on the Input Watermarks

Applying constraints on the expected input will make this project more feasible. Many of the digitized images appear surrounded by borders (Fig. 4), with non-watermark artifacts on the image (Fig. 3), or with several watermarks appearing on one piece of paper (Fig. 4). In untraced watermarks, text can heavily obscure watermarks (Fig. 5). All of these cases complicate the process of computationally analyzing the image. To make this computation more reliable it is first assumed that all input watermarks are cropped tightly around the watermark. This is done to eliminate borders and non-watermark text from the images. Second, text or writing should not obstruct the watermark. Third, the watermark itself should not be blurred or too small and be relatively easy to identify from the background. Finally, input images must contain exactly one watermark.

2.5 Existing Solutions and Research

Before designing the system, it is useful to analyze research and projects that address a similar problem. There has been much research on aspects of watermark analysis. However, previous research relies heavily on machine learning, while this project utilizes a more classical image processing-based

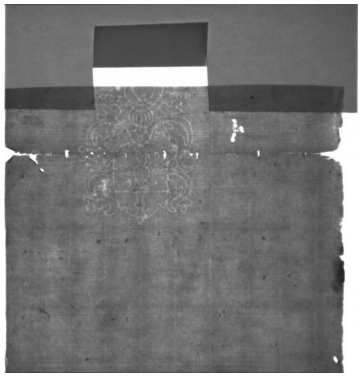

Figure 3: An image with paper artifacts.

Figure 4: An image with rulers and with two watermarks on the paper.

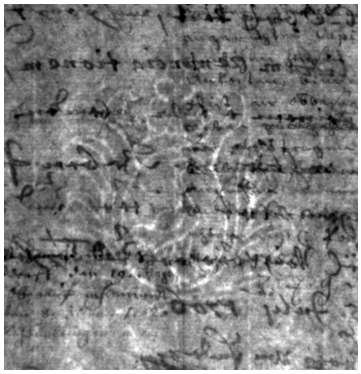

Figure 5: A watermark covered by text.

approach. Thus, no research has been found that approaches the problem in the same way as this project. For example, Shen et. al [10] propose a watermark recognition solution that matches watermarks using a neural network. With this solution, it would be necessary to retrain the neural network each time the database were to expand, taking a significant amount of time and processing power. Since the database for this project will be expanded continually, retraining a neural network each time is not feasible. Alternately, Hiary [11] has done significant research on vectorizing and denoising watermarks, but has not extended his research to similarity matching. The aforementioned studies, although not exactly like this project, are useful for providing ideas for approaching this project. In particular, Hiary's technique of first denoising an image and then extracting features [11] has been useful in designing the system.

Research on general image processing techniques unrelated to watermarks can be useful as well. Convolutional neural networks have been found effective in reducing noise [12], and may be effective in clarifying watermark outlines. Combined wavelets can be used to remove unwanted lines [4]. Zernike moments can extract information about shapes [8], which can be used to describe watermarks. Although none of these directly answer the problem of watermark similarity, they may be useful as building blocks in finding a solution.

2.6 Stakeholders

The stakeholders of this project include the client, Dr. Martin Skrodzki, as well as the developers. Potential future users of the tool are also stakeholders, such as the German Museum of Books and Writings, watermark researchers, and historians. Another stakeholder is the TU Delft, since the client is a member of the university, and the project may be further developed by the TU.

3 Requirements of the Watermark Similarity System

Outlining requirements ensures that key components of the product are agreed upon between the developers and the client. They also ensure that the project stays on track. Requirements Engineering was done in the first two weeks of the project with the client to clarify the project's expectations. Requirements have been split into functional, discussed in Section 3.1, and non-functional, discussed in Section 3.2. They have been further formatted using the MoSCoW model [13], which categorizes them as *Must-haves*, *Should-haves*, *Could-haves*, and *Won't-haves*. To see the completion status of the requirements, and the way the work was distributed, refer to Appendix 2 and 3 respectively.

3.1 Functional Requirements

In this section, the functional requirements will be outlined. Functional requirements involve the actions a product can execute. They have been categorized into must-haves (Section 3.1.1), should-haves (Section 3.1.2), could-haves (Section 3.1.3), and won't-haves (Section 3.1.4) based on the priority level of the requirement.

3.1.1 Functional Must-Haves

1. As a user, I must be able to input an image of a watermark into the system.
2. As a user, I must be able to input the number of similar watermarks that I want to receive as a ranked list from the existing database.
3. As a user, I must receive as output a ranking of the top n most similar watermarks to my input.
4. As a user, for each similar watermark that is output, I must see its similarity percentage as well, where the similarity percentage is the measure of similarity between the input and the image.
5. As the client, I expect that the system must work for watermarks that are mirrored, rotated, scaled, sheared, and clipped.
6. As the client, I expect that the database must be extended on a rolling basis (after a user provides a watermark it gets stored in the database for future reference).

3.1.2 Functional Should-Haves

7. As a user, I should be able to input my watermark into the web-based GUI and run the system on that input.
8. As a user, I should be able to see the output of similar watermarks and percentages through the web-based GUI.
9. As a user, the web-based GUI I interact with should be simple, with clearly labeled buttons and not too much text cluttering the screen.
10. As a user, I should be able to see the result after harmonization so that it can be cleaned up. I should then be able to input this into the rest of the pipeline.
11. As a user, I should have the option to crop the input image before the pipeline is run through the GUI.
12. As a user, I should have the option to augment the strategy used for harmonization by adjusting hyperparameters and strategies through the GUI.
13. As a user, I should be able to state whether the watermark I put in is traced or untraced and have the option to invert the colors.

3.1.3 Functional Could-Haves

14. As the user, I could choose that the system segments the input watermark into elements.
15. As the user, I could input several watermarks, and the system outputs common segmented elements between the watermarks.
16. As a user, I could assign semantic meaning to the returned segmented watermark by inputting a set of tags through the GUI and this would then be stored in the database.
17. As a user, I could have the option to edit the intermediary harmonized watermark output through a tool in the GUI that allows me to add and remove lines in the image.
18. As a user, I could have the option to have my input image cropped automatically by the software.
19. As a user, if the system hasn't given me back relevant results, I could mark the watermark as unresolved and it would join a public section of all such unresolved watermarks.

3.1.4 Functional Won't-Haves

20. As the client, I expect that the system won't make use of or output contextual data for each watermark, for example, the location of the press, or the date of the watermark.
21. As the client, I expect that the system won't make use of or output any image's (Exif[3]) metadata.
22. As the user, I expect that the system won't accept images that are not in non-image formats (e.g. .mp3, .mp4, .pdf, etc.).
23. As the user, I expect that the system won't accept images that are not in grayscale.

3.2 Non-Functional Requirements

In this section, the non-functional requirements will be outlined. Non-functional requirements include all elements related to what the product is or uses. These, too, have been categorized into must-haves (Section 3.2.1), should-haves (Section 3.2.2), could-haves (Section 3.2.3), and won't-haves (Section 3.2.4).

3.2.1 Non-Functional Must-Haves

1. The system must be accessible through the command line.
2. The system must be written in Python 3.
3. The system must use JavaScript for a web application.
4. The system must use pickle (.pkl) files to store the dataset.
5. The system must be able to run on MacOS, Windows, and Linux.
6. The system must be able to support only one user at a time.

3.2.2 Non-Functional Should-Haves

7. The system should be accessible through a web-based GUI.
8. The web-based GUI will run locally.
9. The system should be testable, maintainable, and well-documented.
10. The code should be portable and extendable.

[3]Exchangeable image file format - a standard that specifies formats for images, sound, and ancillary tags used by digital cameras (including smartphones), scanners and other systems handling image and sound files recorded by digital cameras.

3.2.3 Non-Functional Could-Haves

11. The system could have a German/Italian/Dutch version.

3.2.4 Non-Functional Won't-Haves

12. As the client, I expect that the system won't implement security with regard to images uploaded to the system or the dataset.
13. As the client, I expect that the system won't be deployed on a public server.
14. As the client, I expect that the system won't use a local database but will load the data from a pre-stored pickle file.

The above points outline the goals that the watermark system seeks to accomplish. The design, implementation, and evaluation are all created based on the requirements.

4 Watermark System Design

After outlining the problem the project seeks to solve, a design can now be created. First, an overview of the watermark similarity system's design is given in Section 4.1. Then a further explanation is provided on the design of the three main components of the watermark similarity pipeline: harmonization in Section 4.2, feature extraction in Section 4.3, and similarity matching in Section 4.4. Finally, the design for the user interface is shown in Section 4.5.

4.1 Overview of the Design for Analyzing Watermarks

There are two main components to this watermark similarity system. One is the similarity matching pipeline, and the other is the user interface (UI). The pipeline takes the input image and tries to extract useful information about the watermark. It is also responsible for similarity calculations between the input watermark and the comparison watermarks. The user interface allows the user to interact with the similarity matching pipeline, as well as build the database of comparison watermarks. When interfacing with the pipeline, the user can input data to the pipeline, see similar images that are output, and even customize parts of the pipeline. Refer to Section 4.5 for more detail on the user interface.

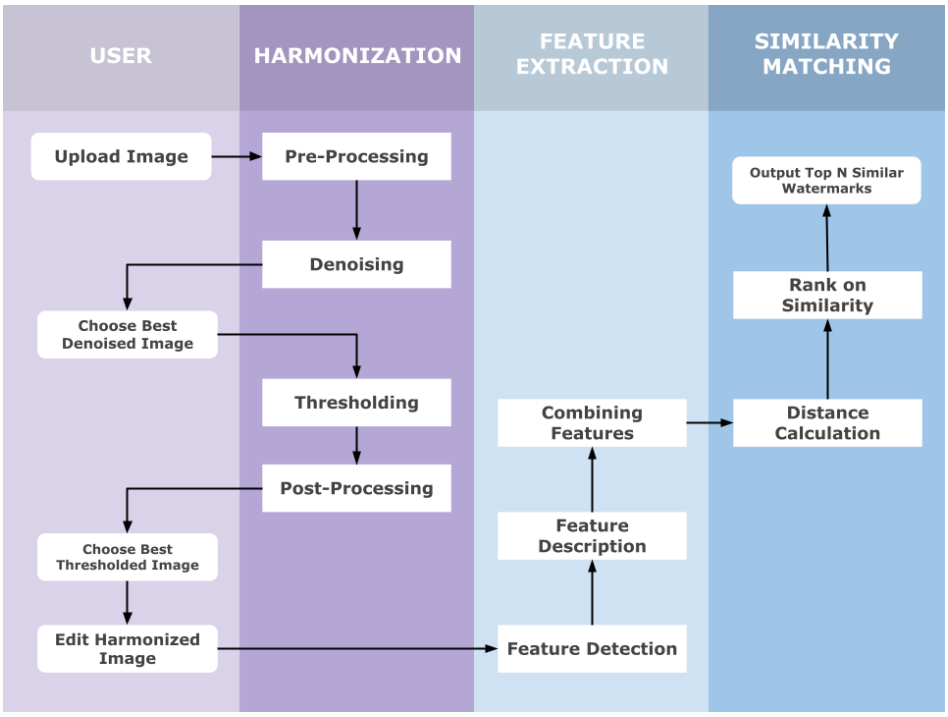

Figure 6: A high-level diagram of the watermark similarity pipeline.

The pipeline has been split up into three core steps: harmonization, feature extraction, and similarity matching. Each of these categories has sub-steps, which are explained further in sections 4.2, 4.3, and 4.4. In the harmonization step, the input image is processed in order to isolate the watermark as best as possible. The feature extraction step involves representing this isolated watermark as a vector. Finally, similarity matching compares a representation of the input watermark to a database of other watermarks and ranks which are the most similar. A diagram of the watermark similarity pipeline can be seen in Fig. 6.

4.2 Harmonizing Watermarks Design

The harmonization step takes the input image and extracts the watermark from it. As explained by Shen et. al [10], this is a complex problem due to its cross-modal nature. Some watermark images are untraced, whilst other watermark images are traced, making a single uniform approach difficult.

Harmonization has been split up into four parts: pre-processing, denoising, thresholding, and post-processing. Pre-processing enhances the image and the watermark. For example, by sharpening the image. Then the image is denoised, which removes blemishes and lines that don't relate to the watermark. Line removal in particular is necessary because of vertical or horizontal lines that often obstruct the watermark, as can be seen in Fig. 2 and 1. Thresholding turns the image from grayscale into black and white, a process called binarization. Binarization isolates the foreground, which is the watermark, from the background. After binarization, thresholding also strengthens watermark lines. Finally, post-processing tries to eliminate any leftover noise from the binarized image. This four-step approach is inspired by the "bottom-up" approach suggested by Hiary [11]. Their proposed pipeline first enhances the watermark, then removes any noise caused by the paper. Each step in the harmonization process seeks to make it easier to separate the watermark outline from its surroundings. The ideal harmonized image has just the watermark outline and no noise.

4.3 Design for Extracting Features from Watermarks

The feature extraction step seeks to represent a processed watermark image as a vector so that two images can be compared. Following Hiary's bottom-up approach [11], the extraction comes after image harmonization. Feature extraction has two steps. The first is feature detection, which detects which points in an image are interesting, like for example corners. It is also possible to skip feature detection and consider the whole image as interesting. The second step is feature description, which seeks to represent these interest points as vectors. After extracting features it is possible to combine features into an all-encompassing descriptor, which can describe a watermark more holistically.

For this system, feature extraction can allow watermarks to be described in a more useful and space-efficient way than if they were just a regular image. Utilizing features that describe the shape of a watermark makes it easier to narrow down which watermarks are generally similar. This project also seeks to utilize features that are invariant to changes in scale, rotation, shearing, clipping, or translation. These invariant features allow the system to account for similar watermarks having variations in their scans.

4.4 Design for Matching Watermarks on Similarity

Similarity matching is the process of taking two vectors and then calculating their similarity, typically using a type of distance measure. In this system, the vectors represent the features of a watermark. When matching similarity, the feature vector of a watermark will be compared to the database that contains feature vectors of other watermarks. Then the most similar vectors, and thereby the most similar watermarks, are output from the system.

When matching watermarks, it is important to consider how the watermarks appear. As described by Müller [1], watermarks were produced by bent wires. These wires would be replaced as they deteriorate over time, meaning that the same type of watermark would appear slightly different on different papers. This is significant for similarity matching because near-identical watermarks, but not

completely identical, still must be considered the same. It is therefore necessary to choose a metric for calculating similarity that accounts for this.

4.5 Design for the User Interface

There are two components of the system that require user interaction: the similarity matching pipeline, and building the database. For the pipeline, the user inputs their watermarks and gets similar watermarks. The database building allows the user to create a database of images that input watermarks are compared against. For each of these components, there is an automatic and a manual option.

The automatic option exists for users that want to run the component quickly, with less user interaction, and with less accuracy. It is accessible through the console and has default values that are used for processing the single or multiple input watermarks. There is some customization possible, like indicating if a watermark is traced and what the path to the watermark is, but overall these options have very minimal user input. For the automatic database building, the user can input the path to the images, the name of the database, and if the images are traced. After some time the database file is generated with no further input. For similarity matching, the user can specify the image path and if the image is traced. Then after some time, the user sees in the console the image paths of the most similar images, with the corresponding similarity scores.

The manual option exists for those users that have more time to work with the system and want better accuracy. This option will be presented through a graphical user interface (GUI), which involves more user interaction. First, there is the GUI for similarity matching which contains an input page (Fig. 7), a processing page (Fig. 8), and an output page (Fig. 9). With the input screen, the user can upload their image, indicate if the image is traced, and how many similar images should be output. The processing screen appears whenever the images are loading. The output screen shows then the input image, the result of the harmonization on the image, as well as all of the similar images. The number of similar images that are output corresponds to the number the user has input. The names and similarity scores of the output images can also be seen.

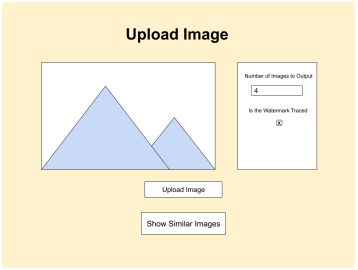

Figure 7: The input screen for similarity matching.

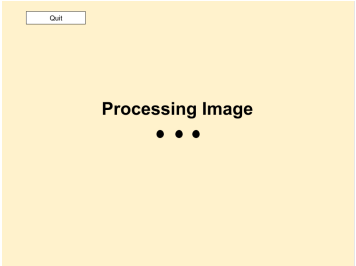

Figure 8: The processing screen.

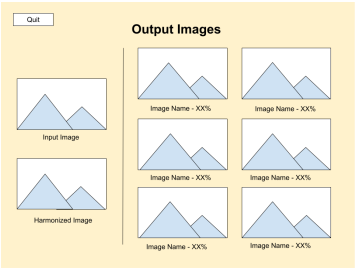

Figure 9: The output screen for similarity matching.

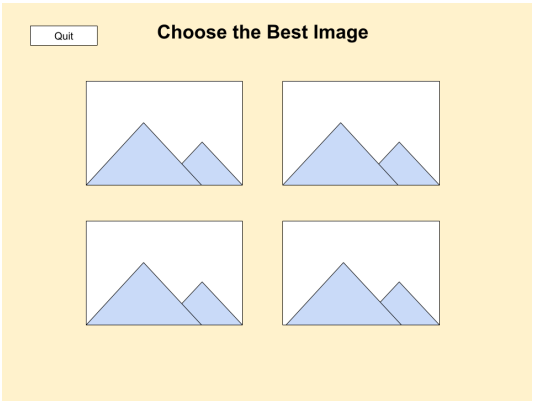

Figure 10: The screen for choosing the best harmonization option.

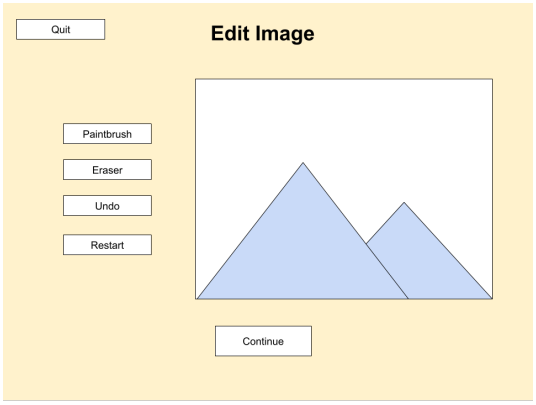

Figure 11: The screen for editing watermark images.

Additionally, the similarity-matching component has screens for pipeline customization. One type of screen allows the user to choose the best image from a series of denoised and thresholded intermediate results. An example of this can be seen in Fig. 10. The aim of this step is to improve the clarity of the harmonized watermark in a customized manner. Another screen enables the user to observe the result of the harmonization pipeline, and add lines or erase noise that appears in the image (Fig. 11). The idea is that incorporating user interaction will produce a more clear outline of the watermark.

Second, there is a GUI for building the database to be used. This component iterates over each image from a specified directory and gives the option to customize the harmonization, similar to the GUI for similarity matching. Then after all the watermarks have had their features extracted, they are appended to a database that is saved. This interface has an input page (Fig. 13), where the user can input the path to the file containing the images to process. For each image, there is a screen (Fig. 12) that allows the user to see the image, and specify if it is traced or not. Finally, the edit-image screens seen in Fig. 10 and 11 are included additionally, so that each image in the database can be harmonized manually as well as possible.

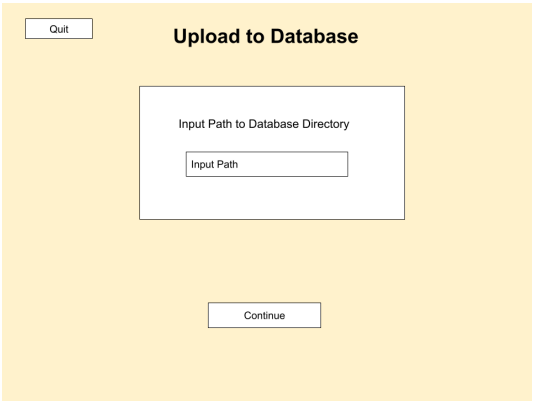

Figure 12: The input screen when building the database.

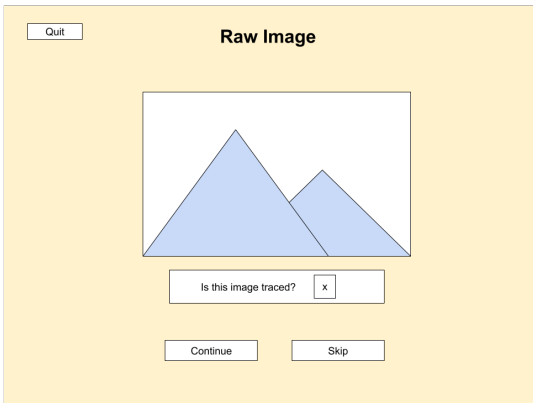

Figure 13: The screen for seeing a database image before it is processed.

5 Watermark Pipeline Implementation

This chapter will discuss how each aspect of the watermark similarity pipeline has been implemented. Much of the basic structure of the watermark pipeline has already been described in Chapter 4. This chapter will not reiterate the basic structure, but will instead elaborate on the specific algorithms and techniques used to implement each step of the pipeline. Note that the implementation of the user interface is excluded here, it can be found in section 6.

First, the harmonization of watermarks will be discussed. Harmonization has been split into two parts, one for traced watermarks in Section 5.1, and one for untraced watermarks in Section 5.2. The goal of each of these two parts is the same, to begin with an input image and output an image with the watermark outline isolated. However, they were split into two because traced and untraced watermarks are very different in how they are harmonized.

After harmonization, the implementation of feature extraction is discussed in Section 5.3. Finally, in Section 5.4 the calculations for watermark similarity matching are explained. In these sections, calculations are made with certain default hyperparameters. Specifications of the hyperparameters used can be found in Appendix 4.

5.1 Implementing Harmonization of Traced Watermarks

Traced watermarks (Fig. 14) tend to have better-defined outlines, making them easier to distinguish than their untraced counterparts. However, there are still challenges when harmonizing a traced watermark. When removing noise and, particularly, lines there is a risk of altering the quality of the image. Moreover, images of tracings often contain annotations that must be removed but are often difficult to distinguish from the watermark properly.

The harmonization process is divided into four phases: pre-processing and denoising are two phases and are both discussed in Section 5.1.1. The third phase is thresholding (Section 5.1.2), and the fourth is post-processing (Section 5.1.3). For an overview of the purpose of each phase, refer to Section 4.2. After processing the image with these phases, the harmonized image consists of the watermark outline in white, with a black background.

Figure 14: Traced watermark to process.

Figure 15: Traced watermark after line removal.

5.1.1 Pre-processing and Denoising Traced Watermarks

The first phase of the harmonization procedure involves pre-processing the image. This involves enhancing the contrast and removing the unnecessary lines that may appear. Below, an overview of the pre-processing steps is given.

In the provided dataset, the images were taken by placing the watermark paper on a bright surface. It is therefore common to have significant contrast between the image borders and center, which can make it more difficult to process the image. To address this, the first pre-processing step involves identifying and replacing these bright areas with the color of the paper. As a result, the borders of the image will appear to be more uniform.

Next, any non-watermark lines are removed from the image. Although this step is typically part of denoising, for traced watermarks it works optimally as a pre-processing step. Line removal can be difficult because the watermarks also contain lines, and these should be preserved. To eliminate only non-watermark lines, the combined wavelet - Fourier analysis approach presented by Münch [4] was used, and it will be further detailed in Section 5.2.1. For the automatic pipeline only the vertical lines are removed (Fig. 15), while for the GUI pipeline, the user can choose to remove either the vertical, horizontal, both, or none of the lines present in the image.

After the lines have been removed, the next step is to apply contrast stretching to adjust the image intensities. This process creates more contrast in the image by stretching the image intensities, which results in the watermark standing out more from the background (Fig. 16) [14].

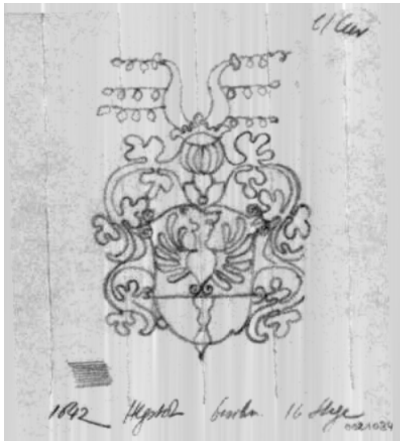

Figure 16: Traced watermark after contrast stretching.

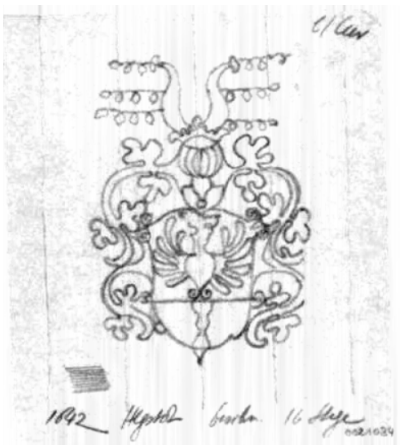

Figure 17: Traced watermark after shadow removal.

Lastly, shadows are removed from the image. This is done because the contrast stretching amplifies all color differences, including shadows, so these should be removed since they are not part of the watermark. Shadow removal is done by isolating the background of the image, subtracting it from the original image, and normalizing the result to restore the original range of values (Fig. 17).

The second phase of the harmonization procedure is denoising. For traced watermarks, this consists of applying a Gaussian blur to blur any remaining noise. This phase helps enhance the quality of noisy images, in order to optimize thresholding results.

5.1.2 Thresholding Traced Watermarks

Thresholding seeks to convert the image into black and white, with the watermark outline appearing in white. For images with low noise levels, Sauvola thresholding is used. This is a local thresholding method, meaning that the threshold value is chosen based on the surrounding region [5]. For high-noise images, local thresholding cannot be used since neighboring regions are obstructed by noise. Instead, a global threshold is used, which takes into account the whole image when choosing a threshold value.

Following binarization, morphological closing and dilation are performed to connect lines that may have been eroded in the process. Morphological closing works by repeatedly dilating first and then eroding the image [2]. The result of thresholding a low-noise image can be seen in Fig. 18.

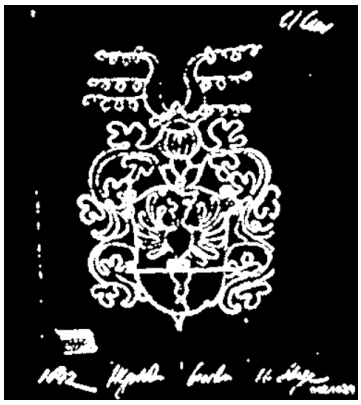

Figure 18: Traced watermark with low noise levels after thresholding.

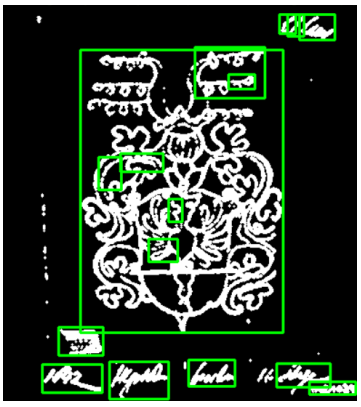

Figure 19: Traced watermark before grouping together the overlapped regions.

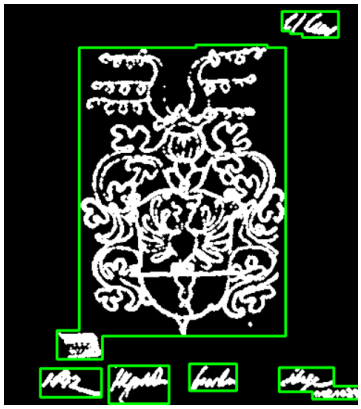

Figure 20: Traced watermark after grouping together the overlapping regions.

5.1.3 Post-processing Traced Watermarks

After thresholding, non-watermark artifacts may still remain in the image, as can be seen in Fig. 18. The post-processing phase seeks to remove these regions so that only the watermark remains. First, connected pixels in the thresholded image are grouped into regions [15]. These regions are filtered on area, length, width, position within the image, and overlap with other regions. An overview of filtering criteria is discussed below.

The first filtering stage removes regions with very small areas, as well as regions that are too high or too wide. These are removed since it is assumed that they are not part of the watermark. Moreover, regions with centroids that are outliers are filtered, since these will be far from the watermark. After filtering the image is cropped to be bounded around the watermark.

The next stage finds which regions overlap each other (Fig. 19), and these overlapping regions are grouped together (Fig. 20). These overlapping regions are also filtered, removing any that are too small, or far away from other overlapping regions. The result after the post-processing phase can be seen in Fig. 21.

Figure 21: Traced watermark after post-processing.

5.2 Implementing Harmonization of Untraced Watermarks

Like with harmonizing traced watermarks, harmonizing untraced watermarks seeks to isolate the watermark outline. Untraced watermarks (Fig. 22) appear most frequently in the dataset. However, they also vary highly in quality, clarity, size, and occlusion. Denoising them thus becomes much harder. This pipeline harmonizes untraced watermarks most effectively when they are sharp, large, and clearly visible to the human eye. Note also that harmonizing untraced watermarks typically performs worse than traced.

As with traced watermarks, the harmonization of untraced watermarks is split into 4 phases. Preprocessing and denoising are discussed in Section 5.2.1. Section 5.2.2 covers thresholding and post-processing. For an overview of the meaning of each phase refer to Section 4.2.

Figure 22: Untraced watermark to process.

Figure 23: Watermark after sharpening.

Figure 24: Watermark after wavelet denoising.

5.2.1 Pre-processing and Denoising of Untraced Watermarks

The first phase consists of pre-processing. For untraced watermarks, this phase simply inverts the watermark image, so that the watermark outline appears in black, and the background in white. Note that the images shown in this section are not inverted, to make them easier to compare to the original visually.

The second phase, denoising, aims to produce a clear distinction between the watermark and the background making non-watermark artifacts less distinct. This process involves 5 steps.

The first step seeks to increase the contrast and level of detail in the image. After receiving the input watermark, it is sharpened, which emphasizes the lines in the image. Specifically, a Gaussian unsharp mask is used, which blurs the image using a Gaussian kernel, and subtracts from the input image the blurred image [16]. The results can be seen in Fig. 23.

The second step is to remove horizontal and vertical lines in the image. In the watermark creation process, lines were produced in the paper by the wires used for the hanging of the paper when drying [1]. To remove them without affecting the watermark, wavelet denoising is used. This technique works by decomposing the image into different frequency bands [3]. High-frequency bands tend to represent more repetitive patterns, like non-watermark lines. Low-frequency bands tend to contain more unique parts of the image, such as the watermark's lines. Therefore, removing higher frequency bands also removes unwanted lines. This technique was used in combination with discrete Fourier transforms to construct an image without non-watermark lines (Fig. 24) [17].

The third step is to enhance the contrast, through contrast stretching. As described in Section 5.1.1, contrast enhancement creates a greater difference between the foreground and the background by stretching image intensities [14].

The fourth step is to remove or blur non-watermark details. To accomplish this Block-matching and 3D Filtering (BM3D) is used[4]. BM3D, in essence, attempts to identify where the edges of an image are, and where the background of an image is. Then, it blurs the background but preserves the watermark (Fig. 25). This is accomplished by grouping similar image fragments, and applying a three-dimensional linear transformation [18].

The final step is applying another edge-preserving filter. Specifically, the Kuwahara filter[5] is used to further blur the background, while maintaining the outline of the watermark (Fig. 26). To accomplish this, each pixel is assigned a new intensity value, being the mean of the most homogenous part of the pixel's neighborhood [19]. However, this filter occasionally mistakes textured backgrounds as edges, so watermarks with a clear background work best. For example, cropping the image to the watermark often leads to better results.

5.2.2 Thresholding and Post-Processing Untraced Watermarks

Afterdenoising the image, it is necessary to binarize it. This isolates the watermark from the background by converting the image into black and white. This is described further in Section 4.2. The thresholding phase involves 2 steps. Following the thresholding phase, the image is post-processed.

The first step in thresholding is to binarize the image. The denoised image may still exhibit inconsistencies in illumination and contrast across the image, which binarization should account for.

[4] https://pypi.org/project/bm3d/

[5] https://pypi.org/project/pykuwahara/

Figure 25: Watermark after BM3D.

Figure 26: Watermark after Kuwahara filtering.

Therefore, a local thresholding method like Sauvola thresholding is used [6]. As described in Section 4.2, this technique calculates threshold values based on the neighborhood of a pixel [5].

The second step in thresholding is cleaning up the watermark outline by closing gaps in the watermark outline and removing small pieces of leftover noise. As in Section 5.1.2, morphological closing is used to accomplish this [2]. For this operation, a morphological cross of size 3x3 is used for 3 iterations.

Finally, after thresholding, the image is post-processed. This is done by doing a connected component analysis. Doing so groups connected pixels into regions and then filters out regions that have a size below a certain threshold value. Post-processing of untraced watermarks is a slightly simpler version of the post-processing of traced watermarks, seen in Section 5.1.3. The watermark image after thresholding and post-processing can be seen in Fig. 27.

Figure 27: Watermark after thresholding and post-processing.

[6] https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_niblack_sauvola.html

5.3 Implementing Feature Extraction

Encoding the harmonized image into a series of numbers, a process called feature extraction makes it easier to calculate their similarity. For this project, techniques were researched that enabled the features to be invariant to scale, translation, and rotation. These are important, since the similar watermarks may be different sizes on the paper, they may be in different parts of the image, or they may be rotated in the image. The system should also match mirrored and clipped images. Mirrored watermarks are common, as the mirrored version of a watermark is found on the other side of the paper. Clipped watermarks are watermarks where part of the watermark is missing. They can be difficult to match due to the information loss but it is still possible.

This need for flexibility led to the decision to use 3 different feature extraction methods that will be combined: SIFT (Section 5.3.1), Hu moments (Section 5.3.2), and Zernike moments (Section 5.3.3). Section 5.3.4 elaborates on how these different methods were implemented in the code. For information on how these moments are combined refer to Section 5.4.2.

5.3.1 Scale Invariant Feature Transform

The Scale Invariant Feature Transform (SIFT) finds and describes interesting points in an image. It detects interest points using the difference-of-Gaussian, determining the scale and orientation, and finally describing the region surrounding it [6]. This process makes the resulting interest points scale- and rotation-invariant. Because the points are localized, it does not matter if the original image is mirrored or clipped, as SIFT will still find the same interest points. SIFT matches similar images by comparing each interest point in an image to every interest point in another image. Unfortunately, with clipped images there is less of the watermark to work with, so fewer interest points can be matched. SIFT is ideal for matching images with many details, such as text.

5.3.2 Hu Moments

Hu moments describe how pixels in an image are spread out by deriving scale-, translation-, and rotation-invariant moments from central moments. This results in seven numbers that fully describe the shape of the watermark. The seventh number even allows for detecting mirroring based on the sign, as an image and its mirrored version will have opposite signs [7]. The only requirement this feature misses is accounting for clipping. Clipped images may result in completely different values because of differing shapes. Still, for matching watermarks that are not clipped, the Hu moments are very useful.

5.3.3 Zernike Moments

Zernike moments offer a different way to derive moments that is more relevant for matching images on their shape. Similar to Hu moments, they describe the way pixels in an image are spread out. However, rather than being derived from the central moments, Zernike moments calculate moments based on different Zernike polynomials [20] [21]. This feature is rotation-invariant, but not scale- or translation-invariant. Despite this, it is by far the best at describing the shape of a watermark. The relatively small variations in scale and translation are therefore considered acceptable.

5.3.4 Implementation of Feature Extraction

To make experimentation with different feature extraction methods easier, the strategy design pattern was used. The strategy design pattern allows for a strategy to be provided when running the feature extraction, rather than hard-coding only a single option. An overview of the strategies that are available can be seen in Fig. 28. Many different feature extraction methods were experimented with, which means some are unused in the final product, and therefore not discussed in this report. For the sake of future extensions, they are left in the code.

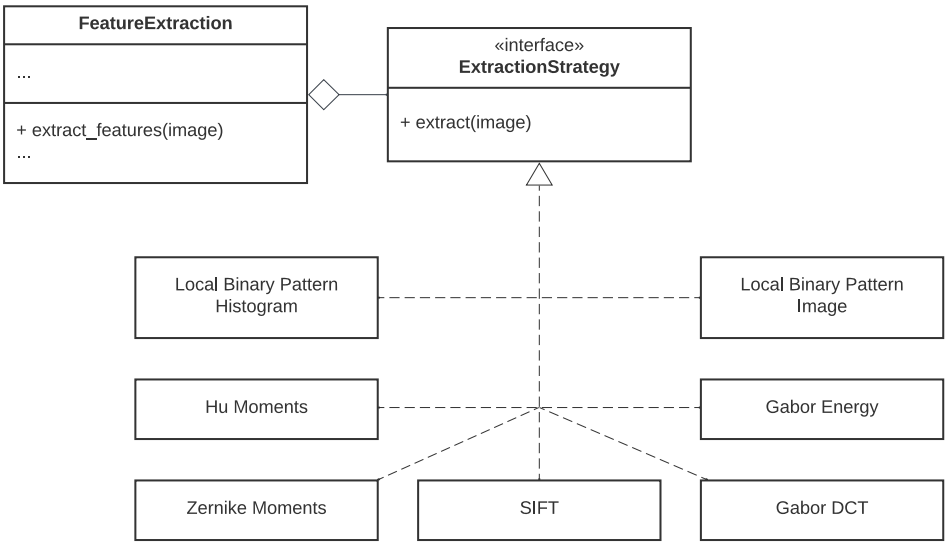

Figure 28: Overview of implemented strategies.

5.4 Implementing a Similarity Matcher

The set of features from feature extraction need to be compared to determine which features, and therefore which watermarks, are similar. To do so, a similarity measure needs to be chosen for each of them (Section 5.4.1) and they need to be combined into a single "score" (Section 5.4.2). The geometric mean of the resulting similarity measures is used for this score, but to improve results, further fine-tuning could be achieved using machine learning, which is discussed in the Recommendations, Chapter 10.

5.4.1 Getting similarity scores from features

The first feature is SIFT descriptors. As mentioned before, interest points in one image need to be matched to points in another. OpenCV[7] provides multiple functions for doing so, and the one that was chosen here is `knnMatch()`[8]. This method returns the `k` best matches where distance is defined by the Euclidean distance. The two best matches are found and Lowe's ratio test is applied, where it is checked whether the best match is sufficiently different from the second best. This finds good matches by making the assumption that a good match will stand out from the other matches [6]. The final score is an integer number of good matches.

[7] https://pypi.org/project/opencv-python/

[8] https://docs.opencv.org/4.x/db/d39/classcv_1_1DescriptorMatcher.html

The second feature is Hu moments. Each of the Hu moments varies massively in magnitude. Some are typically in the range of 10 to 30, while others never get larger than 10^{-18}. To account for this, the base 10 logarithm is applied to the absolute value of each Hu moment, while preserving the sign. This preserves the relative magnitude while bringing all Hu moments into the same range of values. Now the distance between them is the Manhattan distance between the vectors. The Manhattan distance was chosen based on OpenCV's implementation of `matchShapes()`[9] which also uses Hu moments to compare shapes.

Finally, the Zernike moments are compared in a similar way to the Hu moments. Zernike moments do not have the same issue as Hu moments, as the values are generally in the same range. Because of this, they can be compared directly using the Manhattan distance.

5.4.2 Combining the similarity scores

To combine the similarity measures into one score, each one is converted into a percentage, and their geometric mean is taken. The geometric mean is used because it emphasizes small numbers, so if one score is low, the overall score is also low. The percentages are based on the maximum values that were found for the number of SIFT matches, the Hu moment distance, and the Zernike moment distance respectively. The geometric mean was chosen over the arithmetic mean because it gave better results. This already gave decent results, however, it quickly became clear that combining the features optimally would be very time-consuming by hand, which is the reason a machine learning approach was attempted as well (Section 10).

These three steps, harmonization, feature extraction, and similarity matching, make up the core of this project. They are the logic of the system, making it possible to determine what is similar and what is not.

[9] https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html

6 User Interface Implementation

With a functioning pipeline in place, it was decided there should be a user interface to make interaction with the system easier. This section will explain how the system can run automatically by building the database and running the pipeline without interaction, as well as how results can be improved by using the system in the browser and building the database manually.

First, Section 6.1 discusses how the pipeline can be run automatically. Then the method for building the database automatically will be given in Section 6.2. In Section 6.3 there is an overview of how the browser can be used to run the pipeline manually. Finally, Section 6.4 explains of how the database can be created manually.

6.1 Running the Pipeline Automatically

Using the command line, the pipeline can be run automatically. This means there is no user interaction, which typically makes the results worse. Running it automatically may be useful if a large set of data needs to be processed quickly. To do so, default values were manually experimented with and decided on to achieve the best results. More details on the system's accuracy are discussed later in Section 8.3. The input image is harmonized and features are extracted, then the features are compared to the images in the database to return a ranked list.

Specifications on how the system can be run can be found in the README of this project's associated GitLab repository[10]. The automatic pipeline takes three main arguments. The first argument is the input path, which is the path of the input watermark to be matched to. The second argument is the path to the database containing the comparison watermarks. The third argument is a flag that, when input, tells the system that the watermark is traced.

6.2 Building the Database Automatically

Before the system can be used, the database has to be built. When the database is built automatically, each image in a provided folder is harmonized and has its features extracted. Building the database automatically uses default values instead of user interaction. This leads to lower accuracy but faster runtime. In the database, the features and the path to the original image are saved for future use. Because the path to the original image is used, it is important that they are available to the system, for example by being placed in the system's working directory.

Running the system is specified in the README of this project's associated repository. The automatic database builder takes two arguments. The first is the path of the folder containing the images that are to be processed. The second is the name of the database that will have the processed images appended to it.

6.3 Finding Similar Watermarks with the Graphical User Interface

If the user wants to find similar watermarks manually, through the graphical user interface, they must run the application. How this is done is specified in the README, found in this project's GitLab repository. Once run, the start screen of the application can be accessed from `http://localhost:5000/`.

[10] https://gitlab.ewi.tudelft.nl/cse2000-software-project/2022-2023-q4/ta-cluster/cluster-18/sp-18b/cs-for-the-humanities-watermarks-18b/

For a detailed explanation of each scene's design, refer to Section 4.5. To give an overview, in the start scene (Fig. 29) users upload their watermark image and input if it is traced, which database they want to use for comparison, and the number of similar images they want to get as output.

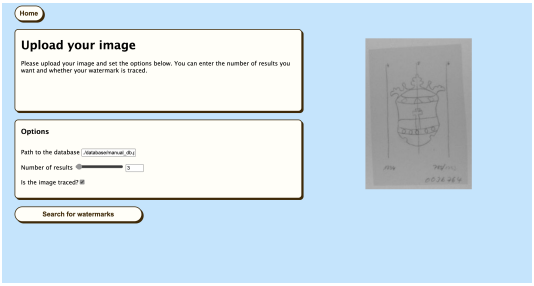

Figure 29: The input screen.

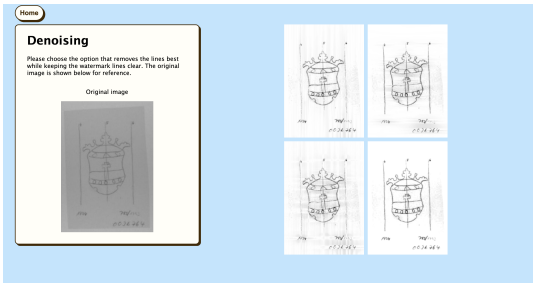

Figure 30: The denoise screen.

After pressing the submit button, the user is presented with a series of denoised images (Fig. 30), and they must choose the best option. Once their choice is clicked, they proceed to the threshold page (Fig. 31). Here, six thresholded images are presented, and the user must choose the best. After the thresholded image is chosen, the user can edit the image (Fig. 32). On this screen, an image is displayed of the raw input image overlayed with the harmonized image in green, to aid users that may have forgotten what the original watermark looks like. To edit the image, the user can paint in lines, erase noise, undo their changes, or restart from the base harmonized image. Once this process is complete, the user waits for similar images to load (Fig. 33).

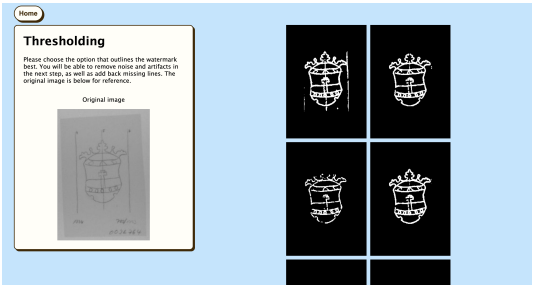

Figure 31: The threshold screen.

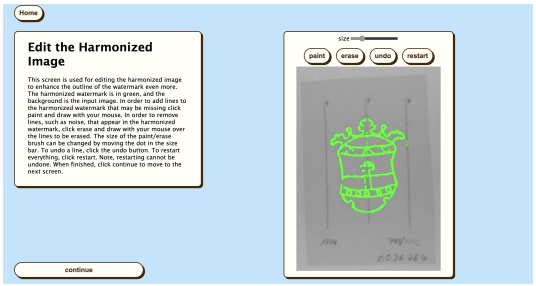

Figure 32: The edit screen.

Once done, the output page is shown (Fig. 34), with the input picture, the harmonized picture, as well as the output watermarks. Each output watermark has its image name, its rank, and its similarity percentage displayed. To see how the similarity percentage is calculated refer to Section 5.4. The number of watermarks output corresponds to the number input by the user. It is also possible for a user to generate ten more similar watermarks by choosing a button at the bottom of the page.

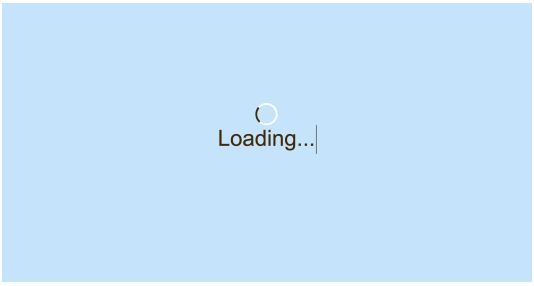

Figure 33: The loading screen

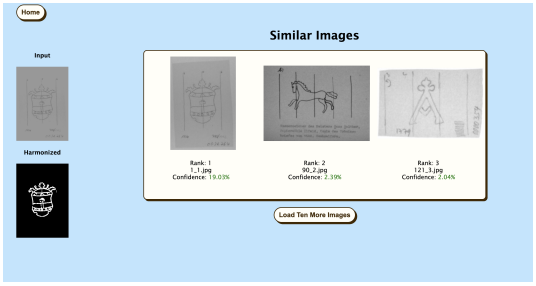

Figure 34: The output screen

6.4 Building the Database Manually

To achieve the most accurate possible results, users are encouraged to build the database manually. Building the database manually ensures that the watermark outline is more distinct. Users must first run the GUI, which is described in the README of this project's repository. The start screen for building the database can be accessed with `http://localhost:5000/build_database`.

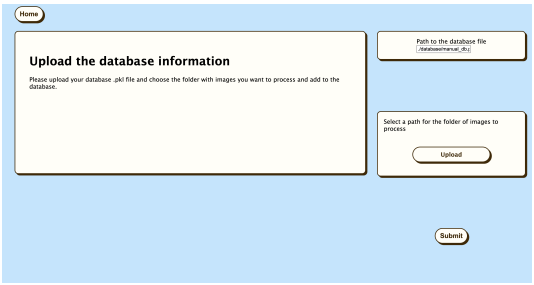

Figure 35: The start screen for building the database through the GUI.

Figure 36: The intermediary screen that shows a database image before processing.

In the first scene, the user must upload the directory of images to be processed for the database. The path to the database file to append to must also be specified. Then the user is taken through each image in the directory. For each image, an intermediary screen is displayed that shows the image and allows the user to specify if it is traced. After this screen, the user goes through the denoising, thresholding, and editing screens outlined in Section 6.3. After one image is finished editing, the intermediary screen shows a new image in the directory. Once the user has gone through all images, the data is appended to the specified path. If the process is interrupted, the data is not stored.

7 Ethical Implications

Considering the ethics of a project allows the project to accomplish its goal in a safe and responsible manner. There are two main ethical concerns that must be considered when designing our watermark recognition system. The first regards the dataset of images this project uses. Specifically, concerns regarding the origins of the dataset will be discussed in Section 7.1, as well as how these concerns can be mitigated. The second ethical concern is potential ways that the watermark similarity system can be abused, which is examined in Section 7.2. Ways that abuse can be mitigated will also be considered.

7.1 Ethics of the Watermark Data

First, the data being used in this project will be reviewed in an ethical light. The data consists of images of watermarks, from documents that are hundreds of years old. All data has been provided in its digitized form by the German Museum of Books and Writing for this project. Due to their age, the documents can also no longer be copyrighted. However, it is still possible that using legally acquired data can pose ethical challenges. For example, these dataset images may be images of personal documents or private correspondences. It is known that the data used for this project was part of a private collection of documents that was acquired by the German National Museum of Books and Writing. However, it is not known where precisely each document came from. It is therefore also not known if the owner of each document gave their consent for it to be published or used. It would be an ethical problem to use data that its owner did not agree to distribute. Therefore, the assumption is made that the museum has acquired these documents in an ethically responsible manner. Since the museum is a trusted and reliable organization, this is a relatively safe assumption to make.

To mitigate concerns regarding the ethics of the dataset of images used, the most straightforward solution would be to find a new dataset. Ideally, this dataset would be one wherein each document has received consent to be used for the purpose of this project. However, this would not necessarily be a realistic solution given the sheer quantity of watermarks required to have an extensive and useful database. Furthermore, authentic watermarked documents are historical in nature, so finding an owner of a document, or their descendant, may prove challenging. It must therefore be accepted that there will inevitably be a trade-off between the reliability of this dataset, and the size of the dataset. As mentioned above, this project's data comes from a reliable organization and is assumed to have been acquired ethically.

7.2 Potential for Abusing the System

Now potential methods of abusing this system will be analyzed. This watermark similarity system allows any user to add any image to the dataset of watermarks without any restrictions. This is an ethical concern, because it could result in sensitive or private data can be stored and circulated by the system. This could result in users getting access to data not intended for them. However, since this system has not been deployed, the lack of security is not a major concern. If a user were to upload a problematic document, the database would only be accessible locally and the issue would not spread to other users. In other words, it would be contained. However, if this system were ever to be deployed, then sensitive data could be circulated to other users, which would be a significant ethical problem.

There are several things that could be done to mitigate unwanted data being added to the database. One step that has already been taken, is designing the system to only run locally. In this case, if any

malicious or sensitive data could not leave the confines of the machine running the system. However, this would not be an acceptable solution if the product were to be deployed. Another, more robust, step to mitigate this concern would be to require permission to upload images to the system's dataset. By adding permissions, it would be ensured that only trusted individuals could add to the system, significantly mitigating the risk of utilizing unethical data. A similar solution would be to have a trusted party, such as a member of the museum, go through all uploaded data to ensure that it is appropriate.

8 Discussion

After implementing the watermark similarity pipeline, its effectiveness in solving the problem must be evaluated. To evaluate the pipeline, the dataset was split and labeled which is explained in Section 8.1. Each form of the pipeline, automatic and manual, is evaluated in sections 8.3 and 8.4, respectively. Since there are two ways of generating databases, automatically and manually, each pipeline evaluates each database. After the pipeline evaluations are discussed, Section 8.5 gives a brief evaluation of running the system on a clearer dataset. The usability of the GUI is evaluated in Section 8.6. An overview of the testing of the system is given in Section 8.7. Finally, the limitations of the system are discussed in Section 8.8.

8.1 Splitting and Labeling the Dataset

From the provided dataset, 500 images have been taken and used for developing and evaluating the watermark recognition system. The selection of images was done randomly, by arbitrarily choosing a watermark and including two to five images of that watermark in the dataset. Then each picture was labeled with a distinct tag, following the format `X_Y.jpg` or `X_Y.png`. In this structure, `X` represents the watermark id within the dataset, while `Y` corresponds to the id of the instance of that watermark. So two images of the same watermark may be called `12_2.jpg` and `12_3.jpg`, as seen in Figs. 37 and 38. In the end, 500 images with 151 distinct watermarks were acquired.

The dataset was randomly divided into 85% for training and 15% for evaluation. The training set was used in implementing the pipeline and creating the database of comparison watermarks. The evaluation set was used for assessing the performance of the system in identifying and returning similar watermarks to a given image.

To assess evaluation, two databases are used, one generated automatically and one - manually. To generate the automatic database, the training set was run through a database-building script, as described in Section 6.2. To generate the manual database, all training data was run through the database-building GUI, as described in Section 6.4. Using the GUI, each training image was harmonized with optimal values, and edited, so that watermarks would be isolated as well as possible.

Figure 37: An image of a watermark of a queen, with label 12_2.png

Figure 38: An image of a watermark of a queen, with label 12_3.png

		Traced	Untraced	Overall
Automatic Pipeline	Automatic Database	44%	40%	41%
	Manual Database	61%	35%	41%
Manual Pipeline	Automatic Database	61%	40%	45%
	Manual Database	61%	51%	53%

Table 1: Results for the Evaluation

8.2 Calculating Accuracy and Evaluation Data

To calculate the accuracy of the system, the number of evaluation images with a match found is divided by the total number of images. Accuracy is calculated for traced evaluation images, untraced evaluation images, and all evaluation images. A match is found if at least one similar watermark is found in the top 10% of returned similar watermarks. So for a dataset of 500 images, this would be the top 50 images.

To access the raw data, including all dataset images and full evaluation data, refer to Appendix 5.

8.3 Evaluation of the Automatic Pipeline

First, the automatic pipeline will be evaluated. The evaluation is performed by programmatically running the automatic pipeline on each image from the evaluation set, one at a time. The accuracy is calculated as stated in Section 8.2.

The results for running the automatic pipeline's evaluation can be seen in Table 1. Evaluating the automatic pipeline with the automatically generated database led to some of the lowest results. This was expected since all of the images were processed with default values. This results in no user interaction in the pipeline, which means that very few images will have the watermark successfully isolated.

To compare, when the manual database was used with the automatic pipeline, the accuracy of traced watermarks improved while untraced watermarks' accuracy worsened. The worsening of untraced watermarks is interesting in particular because the manual database contains better harmonized watermarks than the automatic one. The reasoning for these results could be that the automatic pipeline tends to not remove enough noise in untraced watermarks. When matching to the dataset, the noise leads to incorrect matches. On the other hand, traced watermarks have most noise removed, so the accuracy of the matches would be improved.

Another representation of the evaluation data can be seen in Figures 39 and 40. These histograms show the rank at which correct similar watermark images are found for the evaluation image. As can be seen, the automatic database and the manual database perform similarly. Most of the similar images are found after the top 50 images.

8.4 Evaluation of the Manual Pipeline

The manual pipeline is what users are expected to interact with most often, so evaluating it gives a realistic idea of how the pipeline will perform. The GUI made a programmatic evaluation of the pipeline excessively difficult. Instead, each evaluation image was manually uploaded to the GUI and

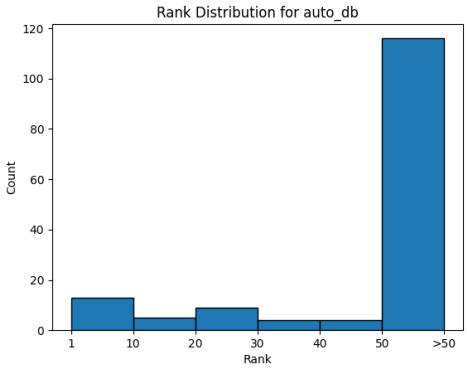

Figure 39: A histogram representing the rank at which similar watermarks are found for evaluation images for the automatically generated database.

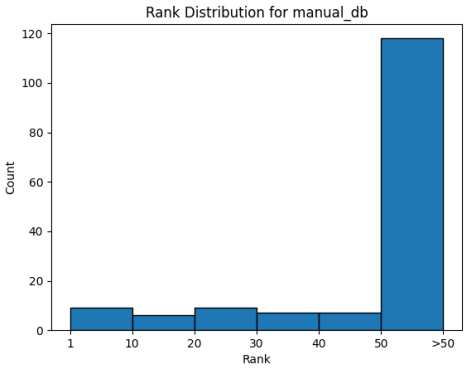

Figure 40: A histogram representing the rank at which similar watermarks are found for evaluation images for the manually generated database.

edited using the GUI to clarify the watermark outline. Then the 50 most similar watermarks were output and searched through to see if an actual similar image was found. Note that this data does not have histograms due to the difficulty in manually collecting that data.

The results can be found in Table 1. As can be seen, the manual pipeline performed better than the automatic pipeline. The combination of the manual pipeline with the manual database performed the best out of all evaluations. It was observed that whenever similar images were found with this combination, they typically were found within the top 20. It was expected that the manual pipeline with the manual database would perform the best because the manual options incorporate user interaction and customization. User interaction typically makes the watermark outlines clearer and thus improves the results of finding similar features.

8.5 Evaluation of a Dataset of Clearer Images

It's clear from the evaluations in sections 8.3 and 8.4 that the accuracy scores for the system could be better. To see where the problem may be, the dataset was analyzed further. Upon reviewing the dataset, it became clear that several images contained watermarks that appear indistinguishable, such as Figure 41.

Unclear watermarks were present in the dataset due to the random nature of its generation. To test the

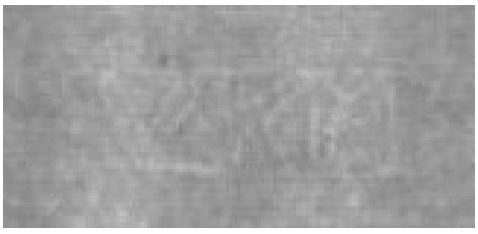

Figure 41: An image in the initial evaluation set with an unclear watermark

system further, a new dataset was created of 200 images. Images in this dataset were chosen on the criteria that the watermarks must be visible to the human eye. It is important to mention that because these images were not randomly selected, it cannot be claimed that they are truly representative of the dataset. Half of the dataset contained traced watermarks, and the other half contained untraced watermarks, with a total of 50 unique watermarks present. This dataset was then divided into a training and an evaluation set, following a 75-25 split.

The database was automatically generated with the training set, and the evaluation was performed by running the automatic pipeline for the evaluation images. The results depicted an overall accuracy of 82%, with 96% accuracy for traced watermarks, and 68% accuracy for untraced watermarks. Another way of representing these numbers can be seen in Figure 42. As can be seen, about half of the correct similar images are found within the top 50 images. This is a significant improvement compared to Figures 39 and 40.

These results show a significant increase compared to the scores in Table 1. Clearly, using a dataset of images that have better defined outlines improves the accuracy of the system greatly.

8.6 Evaluation of the Graphical User Interface

To evaluate the potential user experience with the GUI, several interviews were conducted. In the beginning, the same introduction prompt was read to all participants to provide context (refer to Appendix 6). All gave consent to having their audio recorded and uploaded for the purpose of this project. Their names and personal information are not disclosed. Participants were chosen from varying backgrounds. Some users were inexperienced with both history and technology, some had backgrounds in history or engineering, and some had technological expertise. To access the full audio recordings refer to Appendix 5.

Regarding the style, the pages' simplistic and clean design was appreciated. The instructions were considered clear for everyone, including those with non-technical expertise. Some improvements

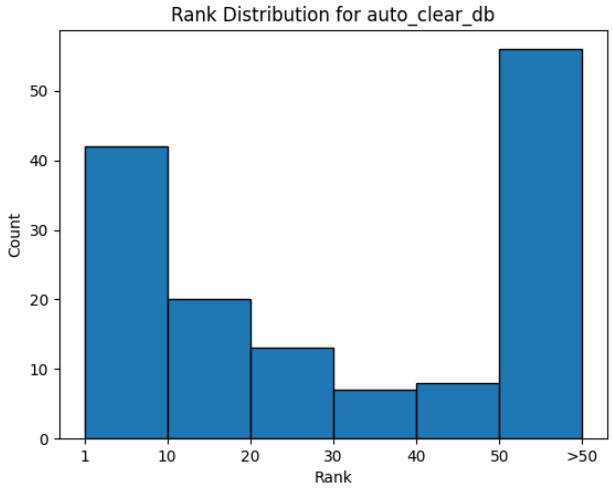

Figure 42: A histogram representing the rank at which similar watermarks are found for evaluation images for the automatically generated database of clearer images.

were suggested like text boxes being too large and too empty, or the spacing between elements being too small. It was also pointed out that the overall color scheme and design of the page didn't seem to match the context of the assignment, as they did not seem historically themed. Lastly, it was mentioned that the text may be too small to comfortably read for users with sight issues.

In terms of interaction, participants believed that the goal of the website was clear and that the results were satisfactory. The participants appreciated the small loading time and the incorporation of user interaction, particularly the drawing tool. Some critiques were made that the slider on the editing screen was unclearly labeled and that a tutorial on how to edit the image could be helpful. Additionally, when choosing an image for denoising or thresholding, it was unclear that the image needed to be clicked. Another critique was made that when uploading a folder for building the database manually, it was unclear which folder needed to be uploaded and whether the selected folder had been successfully uploaded.

A bug was encountered by participants which occurred on the edit image screen. When the user paints outside of the image their drawing does not stop. This bug was known to the developers and was not considered a significant problem, but would still be a point of improvement.

To conclude, participants were overall satisfied with the functionality of the system. Regardless, several improvements can be made, primarily regarding aesthetics and ease of use.

8.7 Testing the System

To ensure that the system behaved as expected, the pipeline as well as the GUI was tested. Testing the pipeline involved inputting values and images to individual methods, and asserting that the output fulfilled certain expectations. Testing the methods provides confidence that the whole pipeline works as expected. Currently, the test coverage, specifically the statement coverage, of the pipeline is at 87%. To see a more detailed coverage report, refer to Appendix 7.

Additionally, the GUI interface for the pipeline has been tested. The input, denoising, thresholding, editing, and output screens were tested to ensure that the functionality of each screen works as expected.

8.8 Limitations of the System

As the evaluation has shown, the system works well but there is room for improvement. Untraced images typically have lower accuracy. This is likely because many untraced images have watermarks that are almost indistinguishable from the paper itself. These images, therefore, rely more on user interaction to find the watermark outline.

Another challenge for the pipeline is recognizing clipped images. Even if the harmonization works perfectly, the feature extraction cannot reliably match clipped images to an original, particularly if a significant amount of detail is missing. SIFT can accommodate clipping, but neither the Hu nor Zernike moments are able to account for this. This results in a lower likelihood for finding clipped images.

Finally, in order to get the best possible results the database should be built manually. This process involves manually harmonizing and potentially editing the images through the GUI, which can take over a minute per image. Doing this for larger datasets of several thousand images would take a

significant amount of work. Fortunately, once the database has been built it does not need to be built again.

9 Conclusion

The purpose of this report was to present a system for watermark identification and similarity matching. This report sought to show the system's effectiveness in tackling the need for a digital watermark analysis system. The implemented watermark similarity tool is intended as a prototype to be further developed in the future.

The goal of the watermark similarity tool would be that, given an input watermark, similar watermarks to it would be output. The approach for calculating watermark similarity was focused on traditional image processing techniques and involved three main steps: harmonization, feature extraction, and similarity matching. In harmonization, a watermark outline is isolated. In feature extraction, the watermark is encoded as a list of numbers. In similarity matching, the similarity between two watermarks is calculated. For running the similarity system, one interface is through the command line, and the other is a more user-interactive graphical user interface. Additionally, a console and a GUI interface have been provided for building the database of comparison watermarks.

The developed system was evaluated and tested additionally to verify its effectiveness. The evaluation showed that the system worked most effectively with the GUI. This is probably due to the user interaction enabled by the GUI, which allows a clearer watermark outline to be found. The system was less effective when automatically analyzing the watermarks, due to the variety of the dataset. It was also found that the system's accuracy significantly increased when using a dataset of images with clearly visible watermarks. A recommendation for future improvement would be to incorporate some type of machine learning algorithm to aid with matching watermarks. Another improvement could be to enable segmenting watermarks into different pieces and allow the user to assign semantics to each segment. This could provide more data for the watermark analysis.

To conclude, the automatic watermark similarity system shows promising results. As a prototype, it can be greatly improved in the future to improve usability and accuracy. Ideally, this approach may serve as the basis for a complete watermark analysis product, accessible to historians and researchers around the world.

10 Recommendations for Future Work

Although this project was successful in completing the most important requirements, there are still ways in which the system can be improved. These include adding segmentation and semantic analysis of the watermarks, incorporating artificial intelligence in the system, and including watermark metadata.

There are two "could have" functional requirements that were not implemented. The first is segmenting the watermark into different parts. The second is enabling the user to label these segments of a watermark. By identifying the components of a watermark, it is possible to compare watermarks on similar elements, not just on whole watermarks. Labeling the watermark could provide more data for categorizing watermarks, which may also aid the similarity-matching process.

Another way of improving results is by utilizing machine-learning techniques for matching similar features. The code already contains a Convolutional Neural Network (CNN) used for matching images. However, it is never used in the pipeline because it needs to be retrained after changing the database, since this would take a significant amount of time. A goal of this project was to avoid retraining machine learning models, so currently the project does not actively use machine learning.

The CNN is a deep learning algorithm, which is trained on labeled data that consists of the Zernike and Hu moments, associated with different images. During training, the model iteratively adjusts its internal weights to minimize the defined loss function. The goal is to improve accuracy in predicting the correct class labels for the given input features.

For evaluation purposes, the CNN was implemented in the pipeline to see if the accuracy improved. Note that the CNN is not integrated into the proper pipeline. Using the manual database with the automatic pipeline and the 500 image dataset, the CNN was evaluated. The results improved overall, increasing the untraced accuracy from 35% to 43%, the traced accuracy from 61% to 64%, and overall accuracy from 41% to 48%. CNN parameters could be refined further for even better accuracy.

Finally, another improvement that could be made would be to include metadata about the watermark in the system. The user would then be able to see similar watermarks, and also see when and where they came from.

Overall, our system works well as a prototype, but there are many future improvements and additions that could be included to make it more accurate and useful for the user.

References

[1] L. Müller, "Understanding Paper: Structures, Watermarks, and a Conservator's Passion," *Harvard Art Museums*, May 07, 2021. Available: https://harvardartmuseums.org/article/understanding-paper-structures-watermarks-and-a-conservator-s-passion

[2] L. Vincent, "Morphological Area Openings and Closings for Grey-scale Images," *Shape in Picture*. NATO ASI Series, 1994, vol. 126, no. 1, pp. 197–208. Accessed: Jun. 14, 2023. [Online]. Available: https://doi.org/10.1007/978-3-662-03039-4_13

[3] S. Khedkar, K. Akant, and M. Khanapurkar, "Image Denoising Using Wavelet Transform," *International Journal of Research in Engineering and Technology*, vol. 5, no. 4, pp. 206-212. Accessed: Jun.14, 2023. [Online]. Available: https://ijret.org/volumes/2016v05/i04/IJRET20160504040.pdf

[4] B. Münch, P. Trtik, F. Marone, and M. Stampanoni, "Stripe and ring artifact removal with combined wavelet — Fourier filtering," *Optics Express*, vol. 17, no. 10, pp. 8567–8591, May 2009, doi https://doi.org/10.1364/OE.17.008567.

[5] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognition*, vol. 33, no. 2, pp. 225–236, Feb. 2000, doi: https://doi.org/10.1016/s0031-3203(99)00055-2.

[6] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, doi: https://doi.org/10.1023/b:visi.0000029664.99615.94.

[7] M. Hu, "Visual pattern recognition by moment invariants," *IEEE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, Feb. 1962, doi: https://doi.org/10.1109/tit.1962.1057692.

[8] Z. Chen and S.-K. Sun, "A Zernike Moment Phase-Based Descriptor for Local Image Representation and Matching," *IEEE Transactions on Image Processing*, vol. 19, no. 1, pp. 205–219, Jan. 2010, doi: 10.1109/tip.2009.2032890.

[9] "Bernstein," *Bernstein – The Memory of Paper*, Mar. 08, 2023. Available: https://www.memoryofpaper.eu/BernsteinPortal/appl_start.disp

[10] X. Shen et al., "Large-Scale Historical Watermark Recognition: dataset and a new consistency-based approach," PDF, *arXiv Cornell University*, 2019. Accessed: Apr. 29, 2023. [Online]. Available: https://arxiv.org/pdf/1908.10254.pdf

[11] H. Hiary, "Paper-based Watermark Extraction with Image Processing," PDF, *The University of Leeds*, 2008. Accessed: Apr. 29, 2023. [Online]. Available: https://etheses.whiterose.ac.uk/1355/

[12] R. Zhao, D. P. K. Lun, and K.-M. Lam, "NTGAN: Learning Blind Image Denoising without Clean Reference," PDF, *The Hong Kong Polytechnic University*, 2018. Accessed: Apr. 29, 2023. [Online]. Available: https://www.bmvc2020-conference.com/assets/papers/0046.pdf

[13] D. Clegg and R. Barker, *Fast-track : A RAD approach*. Wokingham, England: Addison-Wesley Publishing Company, 1994. Available: https://dl.acm.org/citation.cfm?id=561543

[14] Fisher, R, et al. “Point Operations - Contrast Stretching,” *Hypermedia Image Processing Reference*, The University of Edinburgh, Accessed: Jun. 14, 2023. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm

[15] “Measure region properties — skimage 0.21.0 documentation,” *scikit-image.org*. https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_regionprops.html Accessed: Jun. 14, 2023.

[16] A. Polesel, G. Ramponi and V. J. Mathews, ”Image enhancement via adaptive unsharp masking,” *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 505-510, March 2000, doi: 10.1109/83.826787.

[17] J. Torres et al., “Cone Beam Volume CT Image Artifacts Caused by Defective Cells in X-Ray Flat Panel Imagers and the Artifact Removal Using a Wavelet-Analysis-Based Algorithm,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 40, no. 11, pp. 216–228, 2001, doi: 10.1364/OE.17.008567

[18] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, “Image denoising with block-matching and 3D filtering,” *SPIE-IS&T Electronic Imaging*, San Jose, 2006, vol. 6064, no. 606414. Accessed: Jun. 14, 2023. [Online]. Available: https://webpages.tuni.fi/foi/3D-DFT/BM3DDEN_article.pdf

[19] M. Kuwahara, K. Hachimura, S. Eiho, and M. Kinoshita, “Processing of RI-Angiocardiographic Images,” *Digital Processing of Biomedical Images*, pp. 187–202, Jan. 1976, doi: https://doi.org/10.1007/978-1-4684-0769-3_13.

[20] M. Teague, “Image analysis via the general theory of moments*,” *Journal of the Optical Society of America*, vol. 70, no. 8, p. 920, Aug. 1980, doi: 10.1364/josa.70.000920.

[21] H. Shu, L. Luo, and J.-L. Coatrieux, “Moment-Based Approaches in Imaging. 1. Basic Features [A Look At ...],” IEEE Engineering in Medicine and Biology Magazine, vol. 26, no. 5, pp. 70–74, Sep. 2007, doi: 10.1109/emb.2007.906026.

Appendix 1. Usage of ChatGPT

Artificial Intelligence, specifically ChatGPT[11], was occasionally used in the writing of this report. ChatGPT was never used to write things from scratch, it was rather used as a way of rephrasing certain sections. For example, queries such as "Can you rephrase X" or "Can you make this sound better/more formal/less formal/shorter/make sense" were used. Of course, accompanying these queries would be the sentences that we wanted to rephrase. The output of these queries would then be edited further to make them more appropriate for the context of the report. It's important to note that in finalizing the report, several editing stages changed large parts of text for the purpose of brevity, grammar, and clarity. These final edits were all made without any AI assistance. So, it is unlikely that any sentences generated by AI exist in their unedited form in this paper.

ChatGPT was also used to aid certain aspects of the programming. Several techniques were used in the watermark pipeline, and ChatGPT was used to get ideas on which techniques may be effective for the problem, or what certain techniques meant. So examples of queries would be "What is a dyadic decimated wavelet analysis?", "What are examples of edge-preserving denoising techniques?", or "If we have horizontal lines in an image, spaced at every 20 pixels how would I use X technique to remove them?". Additionally, ChatGPT was used to gain insight into the web development process. Queries were used such as "How do I save the images' paths in a global variable for the application?" or "How to iterate over all files in a folder python os?". It is important to note that all queries used for programming were to get an idea of how to approach a problem. Any code that was provided by ChatGPT was heavily altered to fit the context of this project. No code that was generated by ChatGPT was used in an unaltered form.

[11] https://openai.com/blog/chatgpt

Appendix 2. Requirements Completion Overview

Functional			
Must Haves			
1.Upload Image			Complete
2.Input number of watermarks			Incomplete
3. Output top n watermarks			
4.Show similarity percentage			
5.Functionality for mirrored, rotated and scaled images			
6.Database must be extended			
Should Haves			
7. Run the system on a specific image			
8. Get result through GUI			
9. Simple GUI			
10.User can choose harmonized image			
11. Crop the image			
12. Augment harmonization			
13. State whether the watermark is traced			
Could Haves			
14. Segment input			
15.Outputs common segmented elements			
16.Add sematic meaning			
17. Edit harmonized image			
18. Input image cropped automatically			
19. Marking unsatisfactory results as unresolved			
Won't Haves			
20.No contextual data			
21.No metadata			
22.No non-image formats			
23.No image not in grayscale			

Figure 43: Functional requirements

Non-Functional			
Must Haves			Complete
1.Accessible through command line			Incomplete
2.Use Python 3			
3.Use JS for GUI			
4.Use pkl files for dataset			
5.Must be able to run on Mac, Windows, Linux			
6.Only one user at a time			
Should Haves			
7.Accessible through web-based GUI			
8.Web-based GUI run locally			
9.Testable, maintainable and well-documented			
Could Haves			
10. Dutch/German/ Italian verrsion of the system			
Won't Haves			
10.Portable and extendable			
11.No security			
12.No public server			
13.No database			

Figure 44: Nonfunctional requirements

Appendix 3. Work Distribution

	Anna	Diana	Vladi	Sydney	Alex
Database setup	■	■	■		■
Contrast Enhancement	■	■	■		
Feature Extraction				■	■
Perform Matching Query of Image	■			■	■
GUI Processing of the Input Image				■	■
GUI Display Top N best matches	■			■	■
Design an Evaluation Metric		■			
Sharpening	■	■			
Inversion		■			
Binarization	■	■	■		
Cropping Database Images	■		■		
Project skeleton/outline				■	
Dataset Construction		■	■		
Dataset Labeling		■			
Dataset Splitting		■	■		
Harmonize Traced Watermarks	■	■			
Wavelet Denoising		■			
Line Removal		■	■		
Create similarity methods					■
Harmonize Untraced Watermarks			■		
Integration of the system	■	■	■	■	■
Design and implement Image Editing Tool GUI	■			■	
Processing page	■				■
Design GUI Start Screen				■	■
Design and implement Output GUI				■	■
Design and implement Thresholding GUI	■			■	■
Design and implement Denoising GUI	■			■	■
Database processing GUI			■		
Integrate Backend to Flask	■				■
CNN feature comparison					■

■ Worked On

Not Worked On

Figure 45: Work distribution

Appendix 4. Overview of Hyperparameters

Below is an overview of the hyperparameters available for the harmonization. There are no hyperparameters to adjust for feature extraction or similarity matching. The hyperparameters are organized by whether they are traced or untraced, and then further categorized based on the parts of the harmonization: pre-processing, denoising, thresholding and finally post-processing. Note that for the traced images, there are different thresholding techniques for lighter and heavier noise. There are no parameters for denoising traced images with light noise, and also none for pre-processing untraced images.

Traced

Pre-processing

```
wavelet_denoising.
        wavelet_traced(image, levels=8, wavelet='dmey', sigma=2.5,
                       option=1)
```

- `levels`: Number of levels of decomposition.
- `wavelet`: The type of wavelet to use.
- `sigma`: Damping coefficient.
- `option`: Option to use one of four options of line removal. 1: vertical, 2: horizontal, 3: both, 4: neither.

```
wavelet_denoising.
        wavelet_fourier_vertical(image, levels=4, wavelet='sym4',
                                 sigma=3)
```

- `levels`: Number of levels of decomposition.
- `wavelet`: The type of wavelet to use.
- `sigma`: Damping coefficient.

```
wavelet_denoising.
        wavelet_fourier_horizontal(image, levels=4, wavelet='db2',
                                   sigma=3)
```

- `levels`: Number of levels of decomposition.
- `wavelet`: The type of wavelet to use.
- `sigma`: Damping coefficient.

Denoising

```
denoise_traced_heavy_noise(denoise_sigma=0.05, gaussian_sigma=2)
```

- `denoise_sigma`: Sigma value passed on to `skimage.restoration.denoise_wavelet()`
- `gaussian_sigma`: Sigma value for the gaussian used to blur the image at the end

```
skimage.
        restoration.denoise_wavelet(image, sigma=denoise_sigma)
```

- `sigma`: Sigma value to change the extent of the denoising.

Thresholding

```
threshold_traced_light_noise(dilation_shape=(3,3), window_size=25,
                                                     k=0.2)
```

- `dilation_shape`: Shape of the kernel to dilate the thresholded watermark with.
- `window_size`: Window size passed on to `binarize.sauvola_thresholding`.
- `k`: K value passed on to `binarize.sauvola_thresholding`.

```
binarize.
        sauvola_thresholding(image, window_size=25, k=0.2, r=None)
```

- `window_size`: Window size for the sliding window that is used for thresholding.
- `k`: A scaling factor.
- `r`: A normalization factor for the standard deviation.

```
threshold_traced_heavy_noise(threshold_value=190, closing_shape=(6,6),
                             dilation_shape(3,3))
```

- `threshold_value`: Global threshold value, ranging from 0 to 255. Any value lower will be set to 0, and any value above will be set to 255.
- `closing_shape`: Shape of the kernel used for the closing operation.
- `dilation_shape`: Shape of the kernel used for the dilation operation at the end. The shape will be filled in with an ellipse using

  `cv2.getStructuringElement(cv2.MORPH_ELLIPSE, dilation_shape)`.

Post-processing

```
post_process_traced(iteration)
```

- `iteration`: Either 1 or 2. If 1, will remove noisy regions. If 2, will remove noisy regions, then crop the image based on this result and will redo the harmonization with this new cropped image, without ameliorating the borders. This should improve results for images that are affected negatively by ameliorating the borders.

Untraced

Denoising

```
utils_untraced_harmonization.
        denoise_untraced(image, sigma_psd=25)
```

- `sigma_psd`: The expected amount of noise for bm3d.

```
pykuwahara.
        kuwahara(image, method='gaussian', radius=3)
```

- `method`: The method to use for filtering.
- `radius`: The window radius for filtering.

Thresholding

```
utils_untraced_harmonization.
        threshold_untraced(image, window_size=25, k=0.1,
                           morph_kernel=(3,3), iterations=3)
```

- `window_size`: Window size for Sauvola thresholding.
- `k`: Scaling factor for Sauvola thresholding.
- `morph_kernel`: Kernel dimensions for the closing operation.
- `iterations`: The number of times to repeat the closing operation.

Post-processing

```
utils_untraced_harmonization.
        connected_component_analysis(image, min_size=200):
```

- `min_size`: Minimum size, in number of pixels, to keep a component.

Appendix 5. Accessing Raw Data and Datasets

All raw data used, including full evaluation files, recordings for evaluating the GUI, and the dataset, can be found in Group 18B's Software Project GitLab repository[12]. The repository also contains all the `.pkl` files that were used for the databases. If the repository is inaccessible for whatever reason, access to these files can be requested by emailing `m.skrodzki@tudelft.nl`. Additionally, to access the full watermark dataset that this project had access to, a request can be sent to the German Museum of Books and Writing.

[12] https://gitlab.ewi.tudelft.nl/cse2000-software-project/2022-2023-q4/ta-cluster/cluster-18/sp-18b/cs-for-the-humanities-watermarks-18b/

Appendix 6. GUI Evaluation Script

For the GUI evaluation, discussed in section 8.6, a script was read to each participant before their interaction with the website. This script was intended to provide context for the project and the website. Below is the script read to participants.

Today we are going to give you a website that is used for watermark similarity matching. This system deals with watermarks, which has two types: traced [show example of traced watermark] and untraced [show example of untraced watermark]. The purpose of the website is to upload a watermark, find the outline of the watermark itself, and then see the output of similar watermarks.

Today, you will use this webpage. You as a user want to upload a watermark, go through the process of outlining where the watermark is, and then you will see similar watermarks. Throughout the process, please state what you like about the website, what you find confusing, what you dislike, and any other thoughts you may have about it.

Before starting, please note that we will ask you to input a path to a database. Don't worry about filling this out, the default value will work for you.

Appendix 7. Python Test Coverage

Coverage report: 87%

coverage.py v7.2.7, created at 2023-06-25 10:27 +0200

Module	*statements*	*missing*	*excluded*	*coverage*
Pipeline.py	36	12	8	67%
app.py	281	84	21	70%
build_database.py	50	0	12	100%
database/extract_images.py	57	43	8	25%
feature_extraction/__init__.py	0	0	0	100%
feature_extraction/feature_extraction.py	14	0	6	100%
feature_extraction/strategies/DetectionStrategy.py	15	0	6	100%
feature_extraction/strategies/ExtractionStrategy.py	82	0	23	100%
feature_extraction/strategies/__init__.py	0	0	0	100%
harmonization/__init__.py	0	0	0	100%
harmonization/binarize.py	22	0	5	100%
harmonization/contrast_enhancement.py	45	0	4	100%
harmonization/guided_filter/filter.py	70	0	3	100%
harmonization/guided_filter/smooth.py	24	0	4	100%
harmonization/harmonization.py	86	0	14	100%
harmonization/sharpen.py	20	0	4	100%
harmonization/utils_traced_harmonization.py	172	0	16	100%
harmonization/utils_untraced_harmonization.py	26	0	9	100%
harmonization/wavelet_denoising.py	49	0	9	100%
similarity_comparison/__init__.py	0	0	0	100%
similarity_comparison/compare.py	23	0	4	100%
similarity_comparison/similarity.py	38	0	4	100%
Total	**1110**	**139**	**160**	**87%**

coverage.py v7.2.7, created at 2023-06-25 10:27 +0200

Figure 46: Python test coverage report