Applying Machine Learning on Image Matching: Die Studies for Ancient Coins

Master Thesis

Markus Fiedler

6601487

s8343050@stud.uni-frankfurt.de

Supervisor:

Dr. Karsten Tolle

Institut für Informatik

Goethe-Universität Frankfurt am Main

Frankfurt am Main

Summer semester 2024/ Winter semester 2024/2025

Date of submission: 08.01.2025

Abstract

Die studies are complex and time-consuming operations for numismatists. Computer vision tools can assist in operation and even improve quality and speed up the process.

This thesis builds upon existing approaches to digital die studies, in particular on the study by Deligio et al. which used image matching techniques to compare the images of coins. They use a three-step approach: preprocessing, computing distances between the images of coins by image matching and finally clustering of the distances.

In this thesis preprocessing is primarily discussed regarding improvement of computation time, for this process required a couple of hours to complete until now. The main focus lies on distance computation consisting of keypoint detection and description, image matching and converting the matchings into distances. Candidate functions taken from current publications are examined and tested which proves the following functions to be most effective:

- DISK for keypoint detection and description,
- mutual nearest neighbour distance ratio matching for image matching and
- the Pearson correlation correlation coefficient for distance computing.

The results were significantly improved by implementing these functions. Finally, two clustering alternatives for agglomerative hierarchical clustering with complete linkage are presented: firstly, agglomerative hierarchical clustering with average linkage and secondly, HDBSCAN.

With regards to the performance metrics recall and specifity, the pipeline using agglomerative hierarchical clustering with complete linkage achieves close to perfect results for specifity and nearly doubles recall compared to the pipeline by Deligio et al.

Using agglomerative hierarchical clustering with average linkage trades a slightly lower specifity for a much higher recall. HDBSCAN performs a little less effectively, but therefore does so without any preliminary knowledge about the number of clusters.

Zusammenfassung

Stempelanalysen sind komplexe und zeitaufwendige Arbeiten für Numismatiker. Computer Vision Werkzeuge können diese Arbeiten unterstützen und dabei die Qualität verbessern und den Prozess beschleunigen.
Diese Masterarbeit baut auf existierenden Herangehensweisen für digitale Stempelanalysen auf, insbesondere dem Ansatz von Deligio et al., welcher mittels Image Matching Münzbilder vergleicht. Sie verfolgen einen dreistufigen Ansatz: Vorverarbeitung, Distanzberechnung zwischen den Münzbildern mittels Image Matching und schließlich Clusteranalyse.
Die Vorverarbeitung wird in dieser Masterarbeit vor allem in Hinblick auf Zeiteinsparnis betrachtet, da der Prozess bis dato mehrere Stunden in Anspruch nahm. Einen Schwerpunkt bildet die darauf folgende Distanzberechnung mit ihren Prozessen Keypoint Detection and Description, Image Matching und Umwandlung in Distanzen. Für alle drei Schritte werden alternative Funktionen aus aktuellen Publikationen getestet und geprüft, wobei die folgenden Funktionen die besten Ergebnisse erzielten:

- DISK zur Keypoint Detection and Description,
- Mutual Nearest Neighbour Distance Ratio Matching für das Image Matching und
- der Pearson-Korrelationskoeffizient für die Umwandlung in Distanzen.

Durch den Einsatz dieser Alternativen konnten die Ergebnisse deutlich verbessert werden. Abschließend werden zwei Alternativen zur agglomerativen hierarchischen Clusteranalyse mit der Maximum-Fusionierungsmethode präsentiert: zum einen die Durchschnitt-Fusionierungsmethode und zum anderen HDBSCAN.
Bezüglich der statistischen Gütekriterien Sensitivität und Spezifität erreicht die Pipeline mit agglomerativer hierarchischer Clusteranalyse mit der Maximum-Fusionierungsmethode nahezu maximale Werte für die Spezifität und verdoppelt den Wert der Sensivität gegenüber der Pipeline von Deligio et al.
Bei der Anwendung der agglomerativen hierarchische Clusteranalysen mit der Durchschnitt-Fusionierungsmethode tauscht man eine etwas geringere Spezifität für eine deutlich höhere Sensitivität. HDBSCAN bietet zwar geringfügig schlechtere Ergebnisse, benötigt dafür jedoch kein Vorwissen über die Anzahl der Münzstempel.

Acknowledgements

Thank you to my supervisor, Dr. Karsten Tolle, for taking me on as student writing this Master thesis. I am very grateful to have the opportunity to learn and write about die studies, develop digital tools and improve them for the future.
Thank you to Sebastian Gampe. I would like to express my gratitude and appreciation for his patience, guidance and in-depth knowledge which has been invaluable throughout writing this thesis.
Thank you to my family and friends for encouraging and supporting me whenever I needed them and for proofreading my thesis.

Contents

List of Acronyms

i.e. id est

et al. et alii/aliae/alia

cf. confer

e.g. exempli gratia

Chapter 1

Introduction

When humans perceive an image the process goes beyond the mere detection of an arrangement of shapes or colours. We interpret the shapes and colours and finally put them together in the form of a complex pattern which e.g. may become a specific object or person. In order to make a computer aware of these patterns, we need to employ algorithms from simple image alteration algorithms to complex neural networks and quite often a combination thereof. This field of computer science is called computer vision[1]. It makes use of digital images as its primary data and gathers information from these images, such as object detection or classification.
Image matching is a specific task in computer vision which describes the process of finding correspondences between images. This can be used to detect a distinct object across multiple images, in which the object may be visible in different positions, at different scales or from different angles.
In this thesis, something very similar is achieved. Within a given set of ancient Celtic coins, it should be identified which coins were made by using the same coin die. Since the dies of these coins have not been passed down, one has to rely on the visuals of the coins to identify the dies. These coins were minted far before the introduction of machines. Due to the inaccuracy of the minting process, even coins with the same die may visually differ significantly from each other. Corrosion, wear and other effects over hundreds of years contributed to the dissimilarity among the coins. This leads to a less trivial case of image matching where coin images should be found, which on the one hand are similar to each other in their relief and on the other may still differ in many ways.
This results in an approach using computers to assist with die studies: In order to identify coins with the same die image matching techniques are used to find simi-

[1] `https://www.wikidata.org/wiki/Q844240`

larities between images. These similarities will be turned into numbers representing the similarity distances between the images: The more similar the coin images are, the smaller the (similarity) distance. These similarity distances are next submitted to a clustering algorithm. By means of certain parameters, the clustering algorithm will compute clusters of images with small distances (higher similarities) which are supposed to represent clusters of images with the same coin die.

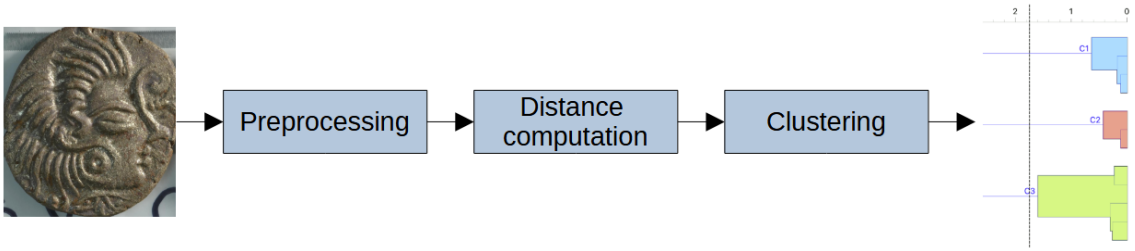


Figure 1.1: The pipeline starts with a set of images as input. They run through three steps: preprocessing, matching and distance computation and finally clustering. The above output clustering image is generated by the tool Orange[13].[2]

The outline of this process is shown in Figure 1.1. Chapter 2 presents the preliminaries: section 2.1 outlines the structure of the algorithm and section 2.3 describes how the results are evaluated. Section 2.2 presents and describes the data that is used in this thesis to evaluate the pipelines. Section 2.4 and section 2.5 introduce the libraries and functions used in this thesis.
The pipeline used in this thesis to compute digital die studies begins with preprocessing the data, using established computer vision algorithms to facilitate analyses of the data in the following image matching process. The tests and improvements regarding preprocessing are described in detail in chapter 3. Chapter 4 is concerned with image matching and its three most important components: keypoint detection, keypoint description as well as the matching itself. The last section of this chapter analyses a couple of distance functions with regard to their application on the outcome of the image matching. Finally, the resulting similarity distances are ready to be clustered. Chapter 5 explains the tested changes regarding the clustering process. Chapter 6 presents a few approaches to further improve our results. Chapter 7 provides a small insight into a small program in the form of a Jupyter Notebook [36] that is designed to easily process the image matching pipeline on one's own data.
In chapter 8 the thesis presents the achievements, discusses them in the context of the literature and provides a brief outlook on further research.

[2] `https://orange3.readthedocs.io/projects/orange-visual-programming/en/latest/widgets/unsupervised/hierarchicalclustering.html`

1.1 Motivation

To this day, money is the primary driver in our economic system. It enables an easy and convenient evaluation and exchange of goods and services. In modern times money can have different forms. These days we even know digital currencies. The first types of money were commodities because their physical properties gave them a value of their own.

While the use of commodities, especially metal for the purpose of trade may date back until 2000 BCE, the usage of standardised and certified coins probably started around 700 BCE[3]. Since then, coins have been used all over the world in different countries and throughout cultures.

An important advantage of coins over many other commodities is that they hardly decayed. For historians, this fact makes ancient coins very interesting objects as they can survive many centuries with only little deterioration. Hence they allow for an insight into history, historical processes and historical cultures.

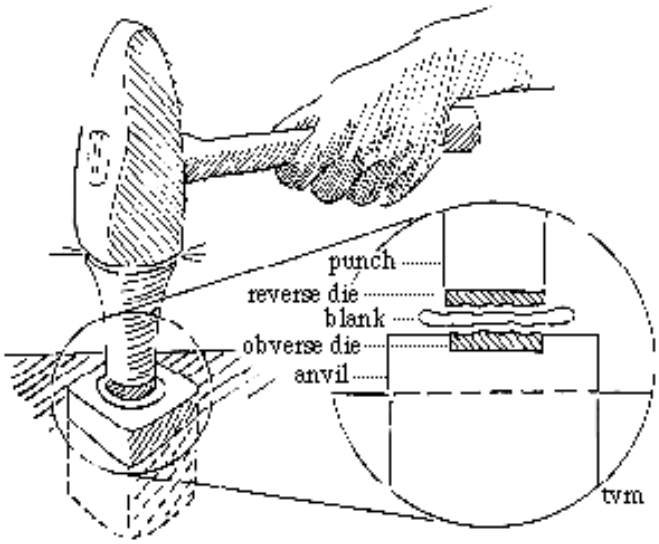


Figure 1.2: An image showing the process of Roman coins being minted.[4]

When making coins, which is called minting or coinage, a blank (a "blank" metal disk[5]) was placed between a lower die (obverse) and an upper die (reverse). With pressure, often through striking the reverse die directly or indirectly, the two types incised in the dies were impressed into the blank and so created the coin[6].

[3] https://www.britannica.com/story/a-brief-and-fascinating-history-of-money
[4] https://www2.lawrence.edu/dept/art/BUERGER/ESSAYS/PRODUCTION7.HTML
[5] https://nnp.wustl.edu/library/dictionarydetail/510404
[6] https://www2.lawrence.edu/dept/art/BUERGER/ESSAYS/PRODUCTION7.HTML

The coin die had a large impact on the contours of the coins created. In particular older coins of the same kind, whose minting process required a lot of manual work, showed differences depending on various dies being used.
Recognizing the distinctive dies which were used to mint coins can help numismatists and historians to make predictions about the history of the coins which themselves can give insights into the trading processes, societies and anthropogeography. However, running the die study manually requires a huge amount of time. The solution is an algorithm that facilitates and speeds up this task.
Automated die studies are not new, but the literature specific to this topic is sparse and only few solutions do exist. The next chapter provides an overview on the various solutions and how they relate to this thesis.

1.2 Related work

Essentially this thesis is a successor of Deligio et al.'s paper [12], testing and presenting new and improved ways for digital die studies. The paper by Deligio et al. presents a pipeline for a digital die study by computing coin die relationships between coins using the images of the coins as input. This thesis makes use of a partial set of coin images used in that paper (further details in section 2.2).
Deligio et al. adopted a preprocessing pipeline inspired by Heinecke et al. [18] combined with new ideas like applying a circle crop to the image of circular coins to further reduce unnecessary information from the image. Keypoints were detected and described on the image by using ORB (Oriented FAST and rotated BRIEF) [48] and then matched. Finally, this data was further processed into distances and clustered within the tool "Orange Data Mining"[13].
Z. Tylor's paper [55] takes a different approach called CADS (The Computed-Aided Die Study). CADS is a tool for numismatists to assist with die studies by providing a fully automated "die study pipeline". The pipeline itself is similar to the approach in Deligio et al.'s paper [12] but adds a classifier to separate coins with different motives before using a distance function to distinguish between the different coin dies.
Cornet et al. also designed a fully automated approach [11], trying to improve CADS and alter its semi-automatic handling to a fully automatic one. They used an all-in-one keypoint detector, descriptor and matching algorithm called XFeat [37]. They also used outsider-detectors like MAGSAC++ [3] rather than the circle crop approach used by Deligio et al. [12] to filter out matches not on the coin

area. However, the most significant innovation is the clustering algorithm AGLP (Adaptive Graph Label propagation). This algorithm does not need a manually generated threshold for the clustering but computes the "best" clustering by itself. It also claims to substantially outperform the results from CADS as they present in [11].

Chapter 2

Preliminaries

2.1 Algorithm structure

A set of coin images constitutes the input of the algorithm used in this thesis. Format aspects of the input images are outlined in section 2.2. The output of the algorithm is a clustering of the images. A cluster represents coins based on the same die. The algorithm itself is the process to compute clusterings from these input images. The algorithm structure contains three main parts:

- preprocessing,
- distance computation by image matching,
- clustering.

2.1.1 Preprocessing

Preprocessing is applied to the input in order to make it fit for further processing. It is typical for most computer vision tasks. After the coin images have been run through, they are suitable to enter the actual processing. The preprocessing step is a pipeline making use of these five subprocesses:

- resizing,
- grayscaling,
- histogram equalization,
- total variation denoising (as proposed by [10]),
- circle cropping.

With regard to optimising the results the order of the subprocesses may require variation or subprocesses may need to be executed multiple times.

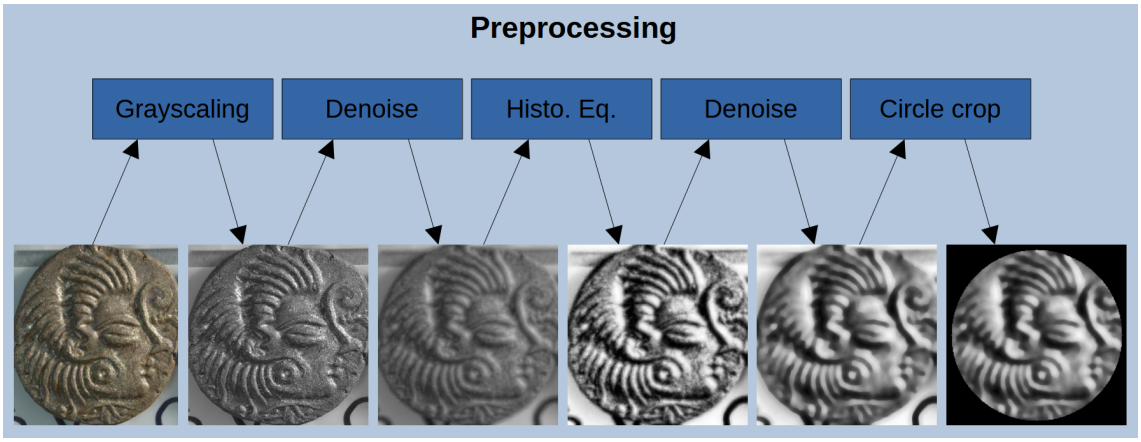


Figure 2.1: Overview of the preprocessing as it was used by Deligio et al. [12].

2.1.2 Distance computation by image matching

Computing distances between coin images is the crucial step that allows for the clustering in the last step of the present die study. In the distance computation step a distance matrix between all images is generated. This distance matrix does not contain physical distances: Instead, the distance here measures the similarity of the coin images: lower distances equal more similar coins.
In order to receive a distance matrix image matching is applied to compute similarities between images, of which the distances will be derived. There are a variety of different forms of image matching. Most of them do not consider the whole image but rather employ so-called keypoints: A keypoint, sometimes also named local feature, contains the position and description of a chosen point on an image. The image matching algorithm makes use of these keypoints to generate a matching for each pair of images. A matching of a pair of images therefore consists solely of those keypoint pairs that the algorithm matched. The matching has to be translated into a "similarity score" which finally is turned into a distance utilizing a fitting function. In conclusion, distance computation by image matching contains three critical success factors:

- detecting and describing keypoints,
- image matching based on keypoints and interpreting the matchings as numerical similarity value,
- applying a function to convert similarity values into one algorithm.

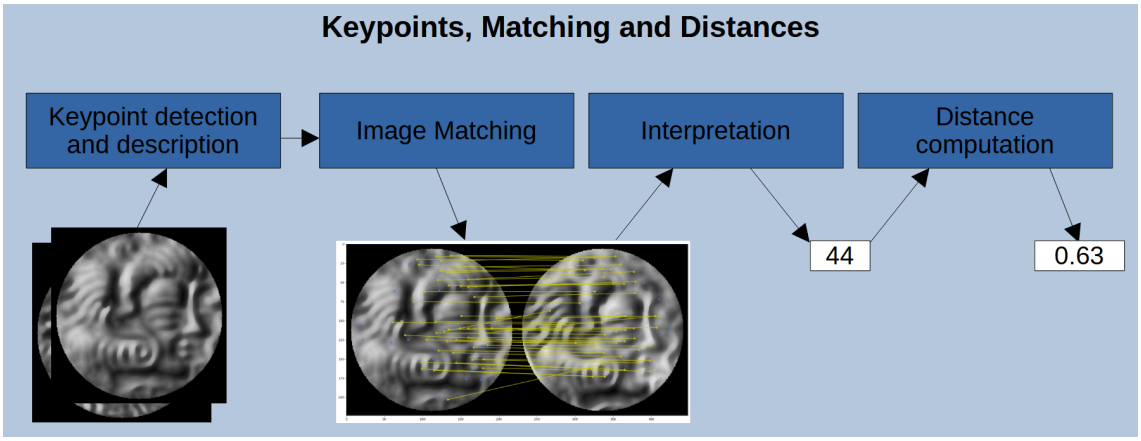


Figure 2.2: Overview of the distance computation process. The input contains the preprocessed images and the output is a distance matrix containing a number for each pair of images.

Keypoint detection and description

Though keypoint detection and keypoint description are two independent steps, they are often combined in a single function. This is done to better coordinate the detection and description processes. While keypoint detection mostly differs in the manner of detection, keypoint description approaches can distinguish clearly from each other.
LoFTR [54] even combines keypoint detection, keypoint description and image matching. But apart from LoFTR, the following three classes of functions are used here:

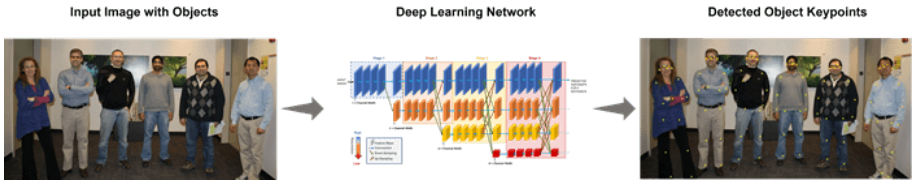


Figure 2.3: The process of keypoint detection visualised. The keypoints in this image could be used for human pose estimation.[7]

- **Keypoint detection** is the process of finding the points in an image, that are significant for its description: Points within a single-coloured area are rarely meaningful. Edges and corners represent the simplest kind of keypoints. And for more complex tasks, for example human pose estimation, handcrafted solutions may be implemented to detect keypoints (as visualised in Figure 2.3).

[7] `https://de.mathworks.com/help/vision/keypoint-detection.html`

When detecting keypoints, position, scale and neighbourhood are the most used indicators for their selection (detection).

- **Keypoint description** is important for the matching of the keypoints in the image matching step. The keypoint descriptor characterises the keypoints of an image by its position as well as its neighbourhood. Keypoints are best chosen when they are well distinctive and invariant to a certain degree of rotation, scaling, change in illumination and other image transformations.[8]

- **Keypoint detector and descriptors** do both of these tasks in one algorithm. While being less flexible than choosing detector and descriptor separately, combining both functions in a single one can harmonise the algorithms and therefore optimise the outcome.

Image matching

After having determined the keypoints of the images, the comparison of the keypoints runs much more efficiently than by comparing entire images. Additionally, the image matching algorithm can focus on the crucial parts of the image.
When the (image) matcher function is fed all computed keypoints, it looks for any matches of keypoints of two images. Figure 2.4 presents an example matching of two images using a method described by Dmytro Mishkin for matching and visualisation[9].
Finally, the matching function returns the matches found and, depending on the matcher, additional information about the matching.
The simplest image matching algorithm is the nearest neighbour matching, in OpenCV called brute-force matcher. For each descriptor in the first image, it looks for the closest descriptor in the second image (not position-wise, but similarity-wise) and returns this as the matching.
Based on the matches found a similarity score is extracted for each pair of images. Depending on the output of the matching function, the number of matches found, the distances of the matches or other data can be interpreted as a similarity score for each image pair.

[8] `https://onlinelibrary.wiley.com/doi/10.1155/2015/310704`
[9] `https://kornia.github.io/tutorials/nbs/image_matching_disk.html`

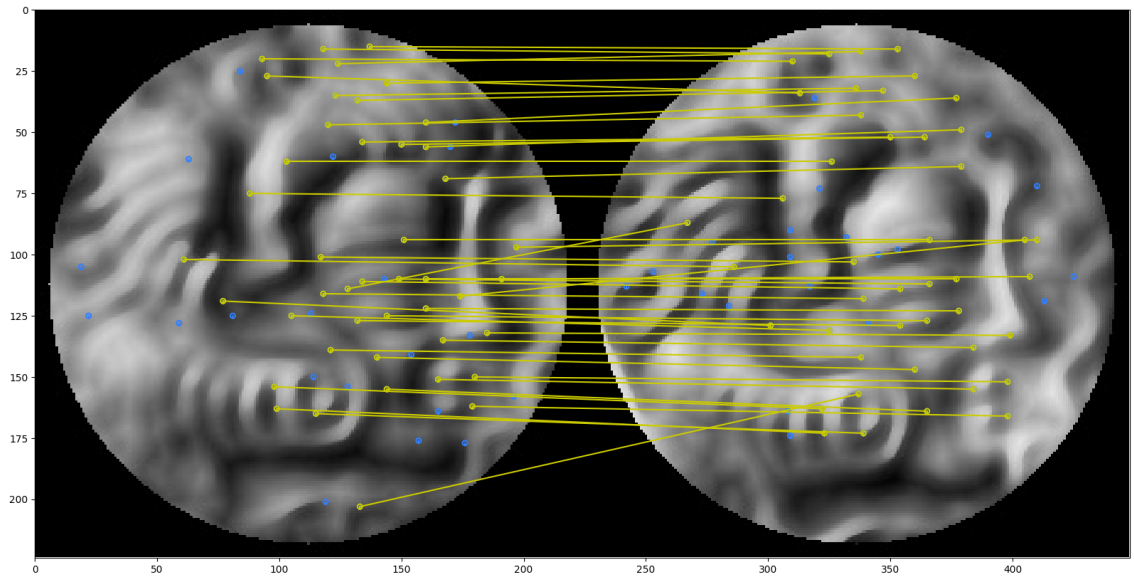


Figure 2.4: Visualisation of the matching of keypoints of two very similar coin images using DISK [58] for keypoint detection and description and Lightglue [23] for matching. The yellow lines between two yellow dots are the (tentative) matches, and the blue dots are keypoints that remained unmatched.[10]

Distances

The similarity matrix as returned by the previous matching might not yet be well suitable as a distance matrix which is needed for the clustering. Therefore a distance function is applied to make appropriate modifications that improve the result during the clustering.

2.1.3 Clustering

Clustering makes use of the distance matrix and existing technologies to establish clusters. In the present case this step still needs some human parameter inputs, though other die study pipelines like in [11] already presented approaches eliminating this problem. The clustering returns a table with a number representing the die of each coin image. Coin images with the same number represent the same die, different numbers mean different dies.

[10] `https://ducha-aiki.github.io/kornia_moons/viz.html#draw_laf_matches`

2.2 Data

The dataset[12] used throughout this thesis contains images of Celtic coins from Brittany, one image of the obverse and one of the reverse of the coin.

Figure 2.5: The observe (on the left) and reverse (on the right) of a Stater coin.[11]

The coins were discovered in the hoard (find) "Le Châtillon II" in Jersey from the year 2012. A big part of these coins were so-called Stater coins: Figure 2.5 pictures such a Stater coin. The obverse displays a stylised head surrounded by multiple beads. The reverse displays a horse in the foreground; it has a human and a charioteer.

These Stater coins were divided into six classes with respect to the shape of the nose and the eye on the obverse of the coin by Colbert de Beaulieu in 1957. Since then some of the classes were further divided into subclasses.

A manually conducted die study for class VI of these coins exists which was already used by Deligio et al. [12] to help develop die study tools. Therefore the coin images from this class are interesting to use as testing data, as this existing die study provides a (near) ground truth for the tests in this thesis.

Figure 2.6: An example image from the dataset used in this thesis (after object detection).

The images specifically used in this thesis were adopted from Deligio et al. [12]. They were already slightly modified because object detection has been applied to the images, to cut off the sections from the image not containing the coin. Figure 2.6 shows an example.

The coins and images respectively differ in quality: The images vary in lumination. Many coins show different degrees of deformations or changes in colour caused

[11] Source: `https://clarenet.hypotheses.org/num-teil3`

[12] German description of the coins in the dataset: `https://clarenet.hypotheses.org/num-teil3`

by physical reactions such as oxidation. This adds to variations typical in hand-crafted production: different coin silhouettes even for coins minted with the same die, different positioning of the motives on the coins, or the use of different dies.
The algorithms used for the present pipeline are sometimes specifically selected for this kind of input data. When using different data as input like for example square coins the pipeline needs to be modified accordingly; otherwise a drastic decrease in the quality of the result has to be taken into account.
The images of the obverses of the coins were chosen as testing dataset in this thesis as it was also chosen by Deligio et al. [12]. This enables easier comparison.

2.3 Testing and runtime preliminaries

2.3.1 Evaluation graphs

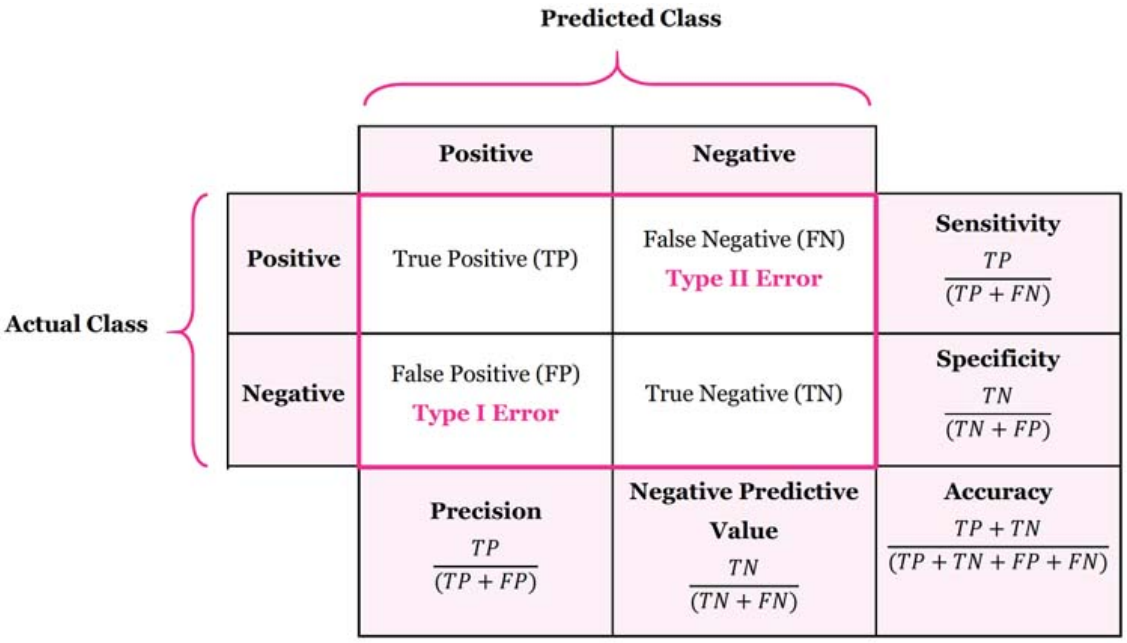


Figure 2.7: A so-called confusion matrix, visualizing true positives, true negatives, recall (called sensitivity in this image) and specifity.[13]

Common performance statistics are used to evaluate the generated clusterings, in particular recall and specifity. A ground truth is required to compute these metrics. This thesis makes use of a clustering established by the numismatist Philip de Jersey while keeping in mind that not even professionals claim their clusterings to be perfect.

[13]Source: `https://encord.com/glossary/confusion-matrix/`

Specifity and recall are statistics that compare the computed results with the ground truth. A pair of images is true positive when they belong to the same cluster in both the clustering and the ground truth. Accordingly, a pair of images is a true negative, when they belong to different clusters in both the clustering and the ground truth. Recall and specifity each display the ratio of true positives or true negatives from all possible true positives or true negatives (see Figure 2.7).
The Rand-index is a single fraction of the sum of true positives and negatives divided by the total number of possible pairs of (different) images. In the present case, it is not suitable due to a total of nearly 1.5 million negative pairs and only 90 thousand positive pairs in the ground truth. Hence, true positives have nearly no impact on the Rand-index which means a substantial imbalance within the Rand-index for the present dataset.
Figure 2.8 shows a graph displaying recall and specifity with regard to two different approaches of the die study. For convenience, the legend usually just refers to the name of the keypoint detector and/or descriptor, the name of the matching function and the name of the distance function. Solely in cases where the important difference between the two methods does not (mainly) involve the matching or distance function, another suffix is added.

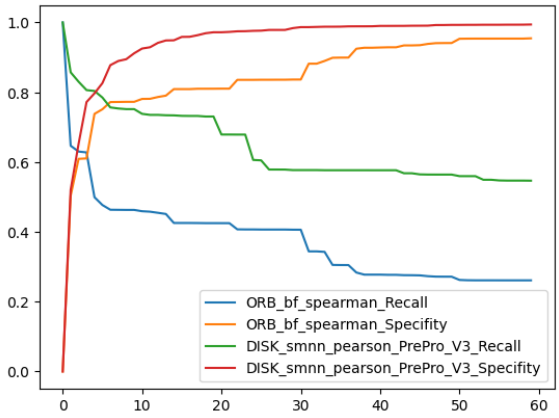


Figure 2.8: Example image, representing a recall/specifity graph used throughout this paper to compare different attempts at the die study.

In Figure 2.8 and in subsequent graphs in this paper the Y-axis describes ratios (recall and specifity respectively); the X-axis indicates the number of clusters given as a variable to the clustering algorithm.

2.3.2 Hardware

All functions and tests were primarily run on one computer with the following configuration:

- CPU: AMD Ryzen 7800X3D with 8 CPU cores and 16 threads
- Graphics card: NVIDIA GeForce RTX 4070 Ti with CUDA support for computations on the GPU

- RAM: 64GB DDR5
- OS: Microsoft Windows 11 Home

2.4 Libraries

OpenCV [5] (Open Source Computer Vision Library) is a python library that offers several hundred solutions and implementations of computer vision algorithms[14]. Particularly noteworthy for this paper are the following functions and implementations:

- keypoint detection and description algorithms,
- image matching algorithms,
- many different image filtering and editing functions used during preprocessing.

Many essential functions like reading images from files or resizing them are provided by OpenCV.
The kornia [43] library also is a python library for computer vision and builds upon Pytorch, which is "an optimized tensor library for deep learning using GPUs and CPUs"[15]. It specifically focuses on having differentiable modules which is a feature needed for deep learning. The kornia library is smaller in scope than the OpenCV library but provides many implementations in selected tasks in computer vision. This includes the `kornia.feature` module, which specializes in local features and image matching. Throughout this paper, many of the implementations from this module will be tested and used.
SCIPY [60] is a library of implementations of functions from mathematical, scientific or engineering fields. In this thesis, this applies to functions in section 4.3.
Some functions used in preprocessing also originate from the image processing library scikit-image library [61] or from the python machine learning library scikit-learn [35]. A few implementations were also used from the Python library belonging to Orange's data mining software [13].

[14] `https://docs.opencv.org/4.10.0/0`
[15] `https://pytorch.org/docs/stable/index.html`

2.5 Functions and algorithms

2.5.1 Preprocessing

In the preprocessing step, the input images for the die study are modified, changed and adapted in ways that benefit the overall results.
The data used in this thesis is, as described in Deligio et al. [12], already preprocessed with regard to object detection: it was used on the images to extract just the part of the image containing the coin which should be centred in the resulting image. Since the further steps in preprocessing expect the images to already have undergone object detection, the overall quality of the object detection is not always sufficient as some coins do not fit the image after object detection as well as the one in Figure 2.6.
The pipeline used in this thesis for the die study most importantly contains these functions:

Grayscaling

This simple function converts a coloured image, with normally three colour channels, into a so-called grayscaled image with only one channel for each pixel which contains the "lightness". This takes away most of the colour information about the images.

Denoising

Functions used for denoising smooth the images and remove small impurities in the images, which are commonly called noise.
An example of a denoising function is the *median filter* which attributes the median value of itself and all surrounding pixels to each pixel. This way pixels with extreme values are replaced by a median value of their surroundings.
Averaging describes a filter similar to the median filter but applies the average of surrounding values instead of the median.
In *Gaussian blurring* a Gaussian kernel is applied to each pixel. This is similar to averaging but instead of giving each pixel the same weight the Gaussian function attributes a weight g to each pixel at position (x, y):

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}, \tag{2.1}$$

where σ is the standard deviation of the Gaussian function.

A different attempt is used in *total variation denoising*: total variation denoising reduces the sum of all gradients of the image, i.e. "the amount of change", but still returns a similar image. *Chambolle's total variation denoising* [10] is such a function where the total variation reduction is defined by the Rudin-Osher-Fatemi (ROF) [49] minimisation problem

$$\min_u \sum_{i=0}^{N-1} \left(|\nabla u_i| + \frac{\lambda}{2}(f_i - u_i)^2 \right). \tag{2.2}$$

These functions are implemented in OpenCV except for Chambolle's total variation denoise function, for which an implementation from GitHub was used[16].

Histogram Equalisation

Histogram equalisation is a technique to enhance the contrast of an image. This leads to an overall better distribution of intensities by allowing the use of the full range of intensities. It also makes differences in intensity better discernible.
CLAHE (Contrast Limited Adaptive Histogram Equalization) is a variation of histogram equalisation that combines adaptivity to avoid local contrast loss with contrast limiting so as not to overamplify noise. The adaptivity is achieved by dividing the image into smaller patches and computing the histogram equalisation for each of those. AHE (Adaptive Histogram Equalization) often suffers from overamplifying noise. Therefore contrast limiting is used in each tile to counteract this. For contrast limiting the histograms are "clipped off" at a certain threshold, redistributing all values over that limit uniformly over the histogram. The tiles are then combined again using bilinear interpolation. CLAHE is also implemented in OpenCV.

Circle cropping

As the name suggests circle cropping crops the image in the form of a circle: the position and size of the circle are chosen by arguments given to the function. Everything beyond this circle is set to black removing information and noise in that area. It is implemented by using functions from OpenCV.

Image segmentation

Image segmentation is a computer vision technique used to separate or segment an image into parts that describe different visual environments of the image.

[16] `https://github.com/danoan/image-processing`

A simple way to implement image segmentation is *thresholding*. After a threshold of intensity is preselected, a binary image is created dividing the pixels of the image into below and above the threshold. Thresholding can be applied "automatically" by letting the algorithm choose the threshold itself or applied "adaptively" if local thresholds are used.
The python package `rembg`[17] provides a background removal tool based on U^2-Net [40]. U^2-Net is a machine learning model for Salient Object Detection (SOD). SOD aims to identify the most visually striking objects or regions within images or videos [26] which in our case are the coins.

Laplace Operator

The Laplace operator is the second-order differential operator. It can be applied to an image to highlight the contours of objects. It is also implemented in OpenCV.

2.5.2 Keypoint detection and description

Keypoint detectors

The *Shi-Tomasi detector* assists in finding corners in an image. A corner can be considered as a sudden brightness change. The Shi-Tomasi detector uses the formula

$$R = \min(\lambda_1, \lambda_2) \tag{2.3}$$

where λ_1 and λ_2 are the eigenvalues at each position. A threshold is selected which is an empirically set value. If R exceeds the threshold, the existence of a corner can be assumed. Values below the threshold may be neglected.
The *DoG* function (Difference of Gaussian) computes the difference between the original image and a Gaussian filtered version of the image. In order to compute the difference this thesis makes use of kornia's standard values for this function: the image with $\sigma_1 = 1$ represents the original image and $\sigma_2 = 1.6$ is the Gaussian blurred image (σ is the standard deviation). The Gaussian function for 2-dimensional images is

$$\Phi(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{||x||^2}{2\sigma^2}}. \tag{2.4}$$

[17] `https://github.com/danielgatis/rembg/tree/main`

Keypoint descriptors

SIFT [25] stands for Scale-Invariant Feature Transform and is an early feature descriptor from 1999. Though it was patented, the patent expired in 2020 and therefore can be used in this thesis.
SIFT is an improvement on the Harris corner detector [17]. The Harris corner detector operates rotation-invariant, but not scale-invariant as it detects corners only in fixed-size image tiles. SIFT uses a scale space to make the features scale invariant and the representation of the directions of the corners are more granular [18].
The *MKD* are Multiple Kernel local-patch Descriptors [30][31] which are simple handcrafted kernel descriptors which themselves are based on the kernel proposed by Bursul et al. [6]. These kernelised local features try to replicate the benefits of SIFT while adding a more continuous representation of SIFT keypoints. Finally, these kernel keypoints are combined with post-processing like Principal Component Analysis and power law normalization as these additions have proven effective [30].
Hardnet [27] is a trained CNN with a focus on descriptor learning. It employs Lowe's ratio test [25] as learning objective. *Hardnet8* improves on Hardnet by tweaking existing structures and exploring new ones.

Keypoint detector and descriptors

ORB (Oriented FAST and Rotated BRIEF) combines FAST keypoint detection [46] and BRIEF keypoint description [7]. Both of these methods work much faster than SIFT. The issue of BRIEF's rotation invariance is bypassed in ORB by adding an orientation component to FAST. ORB is implemented in OpenCV.
SOLD2 [34] (Self-supervised Occlusion-aware Line Description and Detection) follows a different approach. Rather than putting the focus on points, it is a modern line segment detector that detects repeatable lines.
DeDoDe [14] (Detect, Don't Describe — Describe, Don't Detect) is a keypoint detector and descriptor whose name indicates the characteristic of its algorithm. Both the detector and descriptor parts were trained separately to achieve their objective. This algorithm supports two different descriptor models. This thesis uses the model with increased performance according to [14], called DeDoDe-G, which uses DINOv2 [33] features.
DISK (DIScrete Keypoints) is a promising keypoint detection and description model developed by Tyszkiewicz et al. in [58] and originally crafted for 3D-reconstruction.

[18] `https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`

DISK introduces a fully probabilistic model based on a dense matching approach. The feature extraction model itself is based on U-Net [45] with one detection and n description channels, where $n = 128$ is standard. While DISK can be trained manually, the sample of this thesis is not sufficient to successfully train machine learning algorithms.
Descriptor distance: Computing distances between descriptors depends on the descriptor model used, but for most models like the ones used for ORB or DISK it works the following way: The descriptors are stored within a vector. Normal distance functions can be used for vectors, like the Euclidean distance which is used by most matching functions from kornia, to compute the distance.

2.5.3 Matching functions

Nearest neighbour matching compares two sets of keypoint descriptors and generates a list of matchings. It starts with a descriptor of the first set and matches it with the closest unmatched descriptor of the second set. The process iterates for all descriptors of the set. There is an implementation of this algorithm in kornia and in OpenCV as well (called "brute-force matcher" in OpenCV).
FLANN on the other hand is an approximate algorithm implemented in OpenCV, that trades certainty for speed. It is designed to work faster than nearest neighbour matching on large datasets.
Nearest neighbour distance ratio matching as proposed by Lowe [24] and also known as "first-to-second nearest neighbour ratio check" wants to give evidence of a matching that way that there is sufficient difference between the first and second nearest descriptor in each matching. To do so a threshold parameter th is determined. The requirement is that the first nearest neighbour's distance ($d1$) divided by the threshold (th) is smaller than the second nearest neighbour's distance ($d2$) – or in other words: the ration of $d1$ and $d2$ has to be smaller than the threshold th:

$$th > \frac{d1}{d2}.$$

When computing matches all those are discarded that do not satisfy this formula. With regard to this thesis, the sweet spot for this parameter is identified to be $th = 0.85$; any higher or lower values resulted in degrading performance on the present sample.
Mutual nearest neighbour matching works like normal nearest neighbour matching where the mutually best matching descriptors are matched contrary to just selecting

the best matching descriptors for the descriptors from the first image.
Mutual nearest neighbour distance ratio matching combines both the mutuality and the distance ratio threshold for nearest neighbour matching. The optimal (regarding the present sample) parameter for the distance ratio threshold is again $th = 0.85$.
First to second Geometrically Inconsistent Nearest Neighbour from Mishkin et al. [28] is an algorithm that defines the distance threshold between first and second nearest neighbour as a geometrical distance rather than a (for example) Euclidean distance.
AdaLAM (Adaptive Locally-Affine Matching) is a matching algorithm focussing on outlier detection [9]. AdaLAM combines handcrafted methods to outlier filtering with efficiency through parallelism.
This algorithm strongly focuses on solving geometric differences between images. It is less intended for the present study because it contains only very few geometric distances between the images.
LightGlue [23] is a deep neural network used for feature matching based on SuperGlue [50] which itself uses a transformer-based model. LightGlue promises to be more accurate while also being faster and easier to train than SuperGlue.
LightGlue also adapts to the "difficulty" of each matching task which partially accounts for the improvement in computation time compared to SuperGlue.
LightGlue usually allows for better results by training its neural network on a fitting image set. This is not applicable in the current case due to the small amount of data. Earlier tests [12] also proved that using this data for training a machine learning model as ineffective. LightGlue pretrained for DISK keypoints is the suitable variant used in this thesis.
Another matching option is *LoFTR* [54] (Local Feature Transformer). This algorithm does not only compute the feature matching but also operates feature detection and description. LoFTR uses a transformer with self and cross-attention layers. This suggests that its keypoints are more aware of the global context than normal CNNs.

2.5.4 Distance functions

The *Pearson correlation coefficient* is a standard correlation coefficient being the basis for other correlation coefficients like Spearman's rank correlation coefficient.

Definition 2.5.1 (Pearson correlation coefficient)**.** The *Pearson correlation coeffi-*

cient $\rho_{X,Y}$ of two variables X and Y is defined as

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{2.5}$$

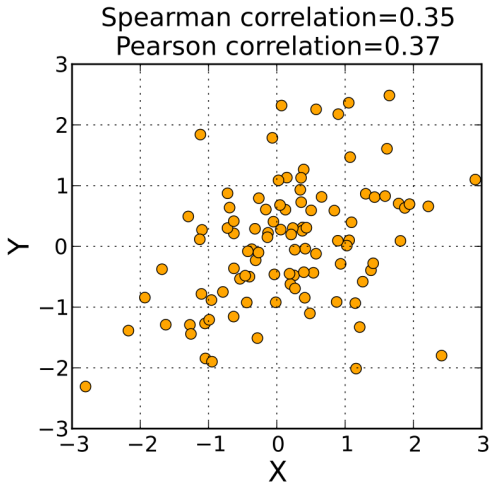
where *cov* is the covariance and σ_X and σ_Y are the standard deviations of X and Y.

Whereas the Pearson correlation coefficient also displays the linearity between the variables, *Spearman's rank correlation coefficient* only assesses the monotonicity. The rank variables $R[X]$ and $R[Y]$ used by Spearman's rank correlation coefficient only represent the rank of each former value of X or Y (for example the second largest value receives (rank-)value 2, the third largest receives value 3 and so on). Apart from that, the formula for Spearman's rank correlation coefficient equals the Pearson correlation coefficient:

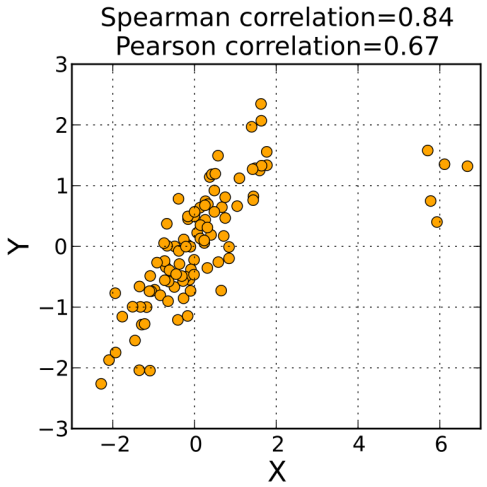
Definition 2.5.2 (Spearman's rank correlation coefficient)**.** The *Spearman's rank correlation coefficient* ρ_s of two variables X and Y is defined as

$$\rho_s = \frac{cov(R[X], R[Y])}{\sigma_{R[X]} \sigma_{R[Y]}} \tag{2.6}$$

The difference between the Pearson correlation coefficient and Spearman's rank correlation coefficient is seen in Figure 2.9.



(a) Plot where points have a smaller correlation coefficient.



(b) Plot where points have a larger correlation coefficient.

Figure 2.9: Visualisation of Spearman's rank and the Pearson correlation coefficient.[19]

[19]Source: `https://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient`

Another rank base correlation coefficient is the *Kendall rank correlation coefficient* τ. The Kendall rank correlation coefficient counts how many indices $1 \leq i, j \leq |X| = |Y|$ either meet $x_i > y_i$ and $x_j > y_j$ or $x_i < y_i$ and $x_j < y_j$ holds. If both variables were equal ($x_i = y_i$ and $x_j = y_j$), then the standard variant of `SciPy`'s tie-solving would be used. One of those tie-solving variants is called variant b or Tau-b. The following definitions are needed in order to describe Kendall's tau-b:

Definition 2.5.3 (Concordant)**.** We call two value pairs (x_i, y_i) and (x_j, y_j) *concordant* if

$$\begin{aligned} \text{either: } & x_i < x_j \text{ and } y_i < y_j \\ \text{or if: } & x_i > x_j \text{ and } y_i > y_j \end{aligned}$$

Definition 2.5.4 (Discordant)**.** We call two value pairs (x_i, y_i) and (x_j, y_j) *discordant* if

$$\begin{aligned} \text{either: } & x_i < x_j \text{ and } y_i > y_j \\ \text{or if: } & x_i > x_j \text{ and } y_i < y_j \end{aligned}$$

Definition 2.5.5 (Tied)**.** We call two value pairs (x_i, y_i) and (x_j, y_j) *tied* if $x_i = x_j$ and/or $y_i = y_j$.

Now that we have defined terms for all possible relations between two number pairs (x_i, y_i) and (x_j, y_j), the Kendall rank correlation coefficient can be defined.

Definition 2.5.6 (Kendall Tau-b correlation coefficient [19], [20][20])**.** The *Kendall Tau-b correlation coefficient* is a variant of the Kendall rank correlation coefficient that also allows handling ties between the value pairs. It is defined as

$$\tau_B = \frac{P - Q}{\sqrt{(P + Q + X_0)(P + Q + Y_0)}} \tag{2.7}$$

where

$$\begin{aligned} P = &\text{ Number of concordant pairs} \\ Q = &\text{ Number of discordant pairs} \\ X_0 = &\text{ Number of pairs tied only in “x”} \\ Y_0 = &\text{ Number of pairs tied only in “y”} \end{aligned}$$

[20]Source: `https://online.stat.psu.edu/stat509/lesson/18/18.3`

Definition 2.5.7 (Cosine similarity)**.** When given two n-dimensional vectors of values X and Y the *cosine similarity* S_C is defined as

$$S_C(A, B) := \frac{X \cdot Y}{||X||||Y||} = \cos(\theta) \tag{2.8}$$

where θ is the angle between X and Y.

Definition 2.5.8 (Cosine distance)**.** The *cosine distance* is simply

$$dist_{cos}(A, B) = 1 - S_C(A, B) \tag{2.9}$$

The *correlation distance* is defined as follows:

Definition 2.5.9 (Correlation distance)**.** The *correlation distance* used in the SciPy library between two 1-D vectors of values is defined as

$$1 - \frac{(x - \overline{x}) \cdot (y - \overline{y})}{\|(x - \overline{x})\|_2 \, \|(y - \overline{y})\|_2}$$

where $\overline{x}$ is the mean on x and $a \cdot b$ is the dot product of a and b[60].

The *linregress* function from `scipy.stats` identifies the linear function that minimizes the squared error between the linear function and the values of two vectors which are interpreted as x and y values.
The *jensenshannon* from `scipy.stats` is a distance function usually used for computing distances between vectors of probabilities (or probability distributions):

Definition 2.5.10 (Jensen-Shannon distance)**.** The *Jensen-Shannon distance* between two (probability) vectors x and y is defined as

$$JSD(x, y) = \sqrt{\frac{D(x\|m) + D(y\|m)}{2}}$$

where $D(x||y)$ is the Kullback-Leibler divergence of x and y and m is the pointwise mean of x and y.

The *Yule* dissimilarity, also known as the coefficient of colligation, is a dissimilarity function originally for 1-D boolean vectors:

Definition 2.5.11 (Yule dissimilarity)**.** The *Yule dissimilarity* between two vectors x and y is defined as

$$Y(x, y) = \frac{R}{c_{TT} * c_{FF} + \frac{R}{2}}$$

where c_{ij} is the number of occurrences of $x[k] = i$ and $y[k] = j$ for $k < n$ and $R = 2.0 * c_{TF} * c_{FT}$.

2.5.5 Clustering

Agglomerative hierarchical clustering is bottom-up hierarchical clustering, meaning that each element starts in its own cluster and the two closest clusters are combined until a threshold is met. There are different methods for combining clusters, called linkage:
The *complete* linkage implies that the maximum distance of two elements amongst the two clusters X and Y is used as distance:

$$D(X, Y) = \max_{x \in X, y \in Y} d(x, y).$$

As Orange does not offer its own Python implementation of hierarchical clustering, `AgglomerativeClustering` from `sklearn.cluster` [35] is used for evaluating the die studies within Python.
Further linkage options for hierarchical clustering are:

- *single*:
$$D(X, Y) = \min_{x \in X, y \in Y} d(x, y)$$
- *ward*:
$$D(X, Y) = \frac{|X| \cdot |Y|}{|X \cup Y|} \|\mu_X - \mu_Y\|^2$$
- *average*:
$$D(X, Y) = \frac{1}{|X| \cdot |Y|} \sum_{x \in X} \sum_{y \in Y} d(x, y)$$

HDBSCAN (Hierarchical Density-based Spatial Clustering of Applications with Noise) [8] is a single linkage hierarchical clustering based variant of DBSCAN [15]. As the name already suggests this algorithm clusters high-density areas. It is able to handle noise because low-density areas are cut off via thresholding[21].

2.5.6 Other

RANSAC (RANdom SAmple Consensus) is an iterative method that estimates parameters to divide the input data into inliers and outliers. As the name suggests RANSAC operates on random samples and is therefore not deterministic.

[21] `https://pberba.github.io/stats/2020/07/08/intro-hdbscan/`

`USAC_ACCURATE`[22] is an improvement upon RANSAC, applying Graph-Cut RANSAC [2] with degeneracy tests. It is implemented in OpenCV and was chosen in this thesis as it provides a solid improvement on normal RANSAC[23].

[22] `https://docs.opencv.org/4.x/de/d3e/tutorial_usac.html`

[23] `https://ducha-aiki.github.io/wide-baseline-stereo-blog/2021/05/17/OpenCV-New-RANSACs.html`

Chapter 3

Preprocessing

The preprocessing step is not a prerequisite to computing a clustering. However, preprocessing its input data can significantly enhance the digital die study. It further allows for employing some techniques that presuppose specific image properties.
In order to meliorate the input data the same procedures should be applied on each image during preprocessing. The goal of running these preprocessing steps is to remove factors that might interfere with the favoured result or that sidetrack the algorithm. The following preprocessing steps that can be considered standard for computer vision tasks turned out to be very useful according to [12]:

- **Grayscaling** Whereas coin dies have a large effect on the contours of the coin during minting, the colour of the images is of little interest. Different colours may just be the result of external effects such as environmental conditions, handling of the coins or varying illumination during the image-capturing process. The image matching algorithm shall not be influenced by this but instead focus on tangible characteristics of the coin's contours that are inherent in the coin die. Grayscaling the images fades out any useless or misleading colour information.

- **Denoising** Denoising is a classic technique in the field of computer vision used to reduce any noise that does not belong to the "true image". This task can be achieved through simple filters like a median filter, statistical methods or even machine learning based approaches. Denoising can be very useful to help the keypoint detection algorithms put the focus on the rougher contours of the coins rather than on small impurities or other noise.

  Partially, denoising is already realised in the pipeline by resizing the images to a much smaller size.

- **Histogram Equalization** While grayscaling and denoising focus on eliminating unwanted noise and information from the images, histogram equalization highlights and enhances the remaining features. By stretching the images to the maximum possible contrast, the remaining features like the contours of the coins are displayed much more concisely.

3.1 The Baseline

Deligio et al.'s paper presents the baseline of this master thesis. Their paper describes the original image preprocessing pipeline in which images are run through a standard computer vision pipeline with some additions at the end. The pipeline consists of four steps:

- It starts by grayscaling the images.
- Figure 3.1 shows the many small impurities in the image. These can be removed by using Chambolle's total variation denoising.
- Contrast and lighting differences are tackled by applying CLAHE to the image. CLAHE achieves this without overamplifying the noise like other histogram equalization functions. Running Chambolle's total variation denoising once again reduces and removes any of the remaining noise.

Figure 3.1: Closer shot of a coin, visualising the impurities.

- Finally circle cropping is applied to get rid of any parts of the image beyond the coin. This is run last so it does not interfere with denoising or histogram equalization.

3.2 Searching for improvements

Some efforts were taken to find out about whether any alteration in preprocessing could lead to any improvement during the later clustering. This section describes the findings with special regard to coin segmentation and denoise functions.

The functions for image matching in this chapter are already different from the ones used by Deligio et al., being DISK for keypoint detection and description, Mutual Nearest Neighbour distance ratio matching and the Pearson correlation coefficient for distance computation. The reasons for this become apparent in the subsequent chapters.

3.2.1 Coin segmentation

A first candidate to improve preprocessing that comes into mind is the final step in the preprocessing pipeline: circle cropping. When coins are not perfectly round circle cropping cuts off either too much or too little of an image. Further, it does not take into account any damage to the coins or mistakes in the object detection step beforehand.

A technique that promises to be a lot more accurate and flexible is image segmentation. Rohm et al. achieved good results for coin segmentation in their paper [44]. However, their images were mostly very clean, had a pure white background and no visible shadow. They tested three coin segmentations in their paper, namely global thresholding [32][59], Hough transformation [42] and a more complex method that is based on local entropy and grey value range that also promised to work with different shapes of images and also different image qualities [62].

In the present case, global thresholding revealed major problems with regard to image quality due to the existence of shadows and objects of varying luminance in the background of the images. Hough transformation was meant to detect one class of shapes - circles in the present case - which conflicted with the irregularly formed coins. The algorithm by Rohm et al. only showed its potential under ideal circumstances.

In particular shadows were still a challenge for image segmentation. And as their test images showed the algorithm only performed on hardly deformed round coins.

Image segmentation by machine learning appeared to be more promising. A common tool for this task was *rembg*:

A pipeline with rembg image segmentation instead of circle cropping allowed comparing the results. For testing pur-

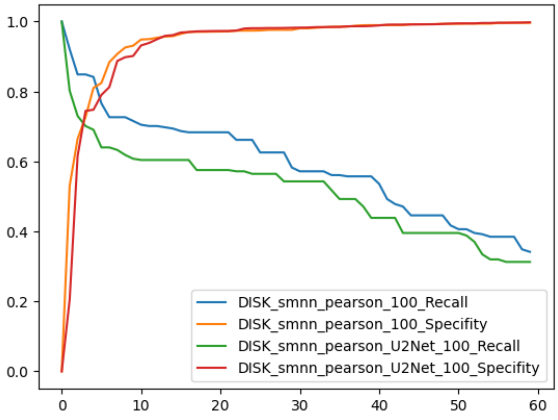


Figure 3.2: Original preprocessing against preprocessing with U2Net image segmentation.

poses, only 100 images were used in this comparison. Ten percent of these images had visible errors from the segmentation process as shown in Figure 3.3b. The pipeline for this preprocessing was slightly optimized to produce as few images with errors as possible. However, if it was possible to further reduce the number of images with errors by using different parameters or different image segmentation functions, this method might also turn out to be more effective than the current pipeline.

(a) An image that looked correct after SOD with U^2-Net.

(b) An image that looked incorrect after SOD with U^2-Net.

Figure 3.3: Results from SOD (Salient Object Detection) with U^2-Net.

3.3 Denoise functions

Throughout preprocessing one function took a major part in the execution time: the denoise function.

While most functions just took a couple of seconds to compute, the denoise function required over two hours (see Table 3.1). Such a huge difference raised the question of whether it was possible to utilize a much swifter function.

The implementation of this function was adopted from the paper of Deligio et al. [12]. It even promised to run in parallel. However, this was clearly not sufficient to get an attractive runtime in preprocessing like the other functions. Therefore, a much simpler solution was found

Function	**time**
Grayscale	$15.8s$
Denoise 1	$13m$ $10.8s$
Hist. Equal.	$7.9s$
Denoise 2	$156m$ $2.4s$
Circle Crop	$6.5s$
Total	$169m$ $43.4s$

Table 3.1: Preprocessing functions and their computation times.

and shall be presented:

Employing an implementation of the Chambolle total variation function in `skimage.restoration` [61] turned out to speed up computation time significantly as shown in Table 3.2. The complete preprocessing was reduced to less than a minute. The computation times here were not calculated under strict conditions but merely taken from the times of the Jupyter Notebook cells. However, the enormous execution time decrease of more than two hours presented sufficient evidence.

Function	**time**
Grayscale	16.5s
Hist. Equal. 1	5.3s
Denoise	16.6s
Hist. Equal. 2	5.1s
Circle Crop	5.6s
Total	49.1s

Table 3.2: Preprocessing functions and their computation times with the new denoise function.

This new function also leads to a slightly improved pipeline, swapping the places of denoising and histogram equalisation. This way computation time was even reduced further because the new Chambolle total variation denoise function was only performed once rather than twice. Histogram equalisation was computed twice at the former positions of denoising, once before and once after Chambolle's denoising function.

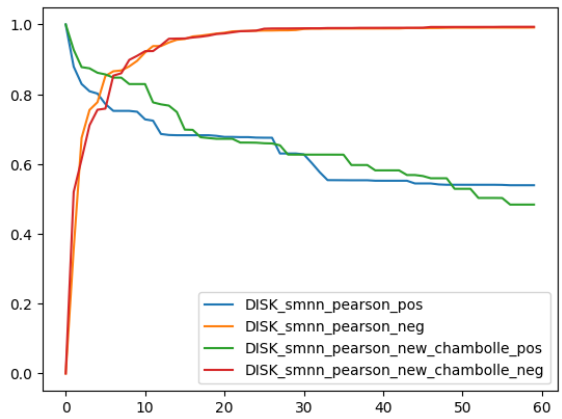


Figure 3.4: Original preprocessing against preprocessing with a different Chambolle total variation function and a different preprocessing order.

The overall performance of specifity was nearly identical to the old pipeline. With regard to recall the old pipeline appeared to generate better results for very high numbers of clusters, whereas the new pipeline was preferable for lower and medium numbers of clusters. The new pipeline might even be strictly better than the old one, as the high numbers of clusters visible in Figure 3.4 only reduced recall, without gaining significantly higher specifity. Lower numbers of clusters should therefore be preferred here.

3.4 Other tested improvements

Figure 3.5: Coin image with applied Laplace Operator.

Another option that appeared promising is the (Discrete) Laplace Operator. The application of this operator to an image highlights corners and edges as shown in Figure 3.5. In theory, this looked like a useful step. However, after testing a few different implementations of the Laplace operator in the present pipeline, the empirical results did not support this.

Prior to evaluating a different implementation of the Chambolle total variation denoise function further denoise functions were tested in an attempt to improve computation time and, if possible, the quality of the result. Such functions tested were the fast Non-local Means denoising[24], median blur, Gaussian blur and bilateral filtering (implemented in OpenCV).

Many of these functions made use of more parameters than the Chambolle total variation denoise function and therefore various combinations of these functions with different parameters were tested, for example multiple Gaussian filters with increasing kernel sizes and decreasing filter strengths. On one hand, a gain in computation time could be achieved by all of these functions; preprocessing took a few minutes instead of hours. However, the performance was usually at least slightly worse than the original results. Since this algorithm was not time-critical, better results were of greater importance than diminished computation times.

3.5 Preprocessing result

Throughout the attempt to develop a more effective pipeline for preprocessing, a lot of different functions were tested, primarily for denoising or add-ons to the existing pipeline. While no denoise function was able to strongly improve the results of the die study compared to the pipeline of Deligio et al. [12], many of them at least were

[24] `https://www.ipol.im/pub/art/2011/bcm_nlm/`

close in recall and specifity when being implemented in an optimised way.
The conclusion was that the preprocessing pipeline was already optimised with regard to the quality of the results. However, the restrained execution time was quite obstructive. Replacing the function with another implementation and slightly modifying the order of functions used, finally resulted in an equally good (possibly even slightly better) and much faster preprocessing pipeline as can be seen by comparing Table 3.1 to Table 3.2.
In summary: the results of the preprocessing were already very good; just minor improvements were achieved. In contrast, its execution time could be drastically reduced by an alternative implementation of the Chambolle total variation denoise function.
Hence, the present pipeline for preprocessing made use of grayscaling, histogram equalization, denoising through the Chambolle total variation function followed by another histogram equalization and finally circle cropping the image (see Figure 3.6). The improvements of the pipeline were a slightly higher specifity and most strikingly significantly reduced preprocessing times.

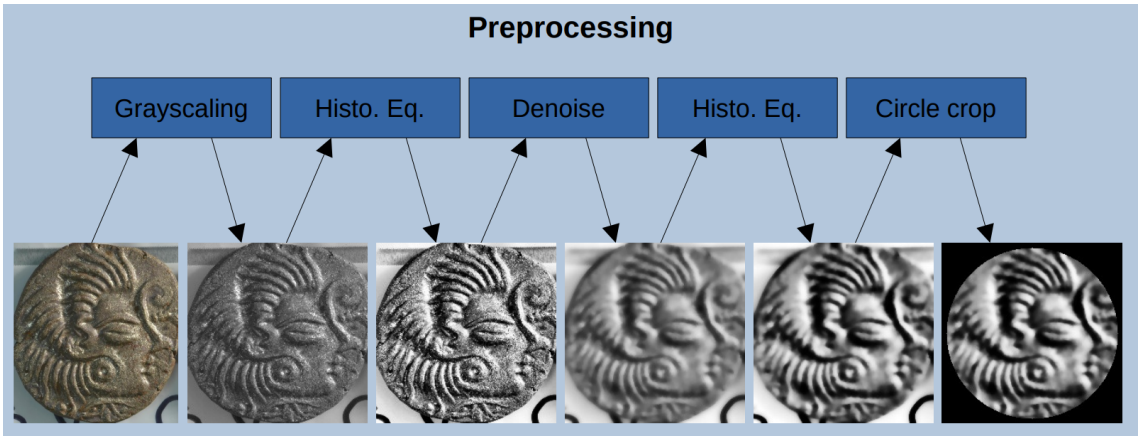


Figure 3.6: Overview of the new preprocessing pipeline.

Chapter 4

Keypoints, Matching and Distances

After the images had gone through preprocessing, they were ready for the actual image matching followed by a distance function being applied to the matching results. This step was necessary before clustering (cf. next chapter) could be done because clustering required numbers or rather distances as input. The step of distance computation was also crucial for the clustering later as these distances represented the images in the clustering algorithm and directly influenced the outcome of the clustering.
This chapter is split into three sections:

- keypoint detection and description,
- matching,
- and distance functions.

Each section gets started with the implementation by Deligio et al. It is presented and evaluated to serve as a basis. Upon this, new or advanced approaches are introduced, evaluated and briefly explained whenever deemed necessary.
Before starting with these, the results of Deligio et al.'s pipeline are presented to set the baseline for comparison.

Setting the baseline

Preprocessed images were the input, as described in section 3.1. They also formed the input for all subsequent tests in this chapter. The results from the preprocessing were given to ORB [48] for keypoint detection and descriptors. To match the

keypoints, OpenCV's `BFMatcher` using Hammond distances was applied. Amongst the resulting matches only those with a distance less than 45 were counted to produce the similarity value. These similarity values were turned into distances using Spearman's rank correlation coefficient. The clustering that was used is described in chapter 5.

The results are displayed in Figure 4.8. The orange graph displays the specifity, which is above 0.8 for most of the graph. Recall is considerably lower at around 0.3 for higher numbers of clusters and between 0.4 and 0.5 for lower numbers. They serve as a baseline against which other techniques were set in the further course.

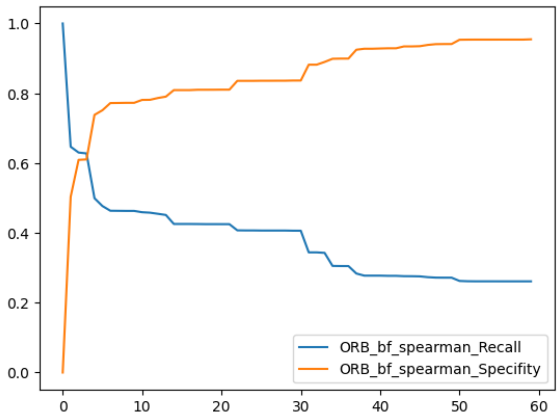


Figure 4.1: Results of the ORB detection and description, brute-force matching with Hammond distances and Spearman's rank correlation coefficient as distances.

The following sections deliver insight into a set of different methods for keypoint detection and description, different matching techniques mainly implemented in the kornia library and also different distance functions as interpretations of the matching results.

OpenCV vs kornia comparison info: Many implementations in kornia [43] allow using ORB for keypoint detection and description. The brute-force matcher by OpenCV (according to its description) does just nearest neighbour matching. An implementation of kornia's nearest neighbour counterpart together with ORB could not achieve the results from OpenCV.

A trial of this approach (cf. results in Figure 4.2) including the computation of the hamming distance before matching was not able to deliver results similar to the OpenCV implementation, too. Therefore, in order to facilitate the comparison, the "nearest neighbour distance ratio matching" function (smnn) was used for matching the keypoint results from kornia.

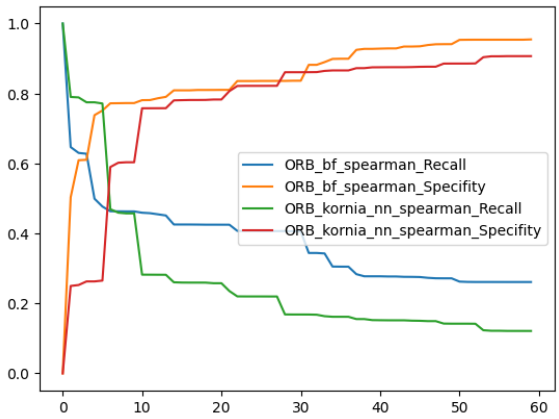


Figure 4.2: ORB implemented in OpenCV versus ORB implemented by me with kornia's nearest neighbour matching.

The kornia image matching overview[25] recommends DISK. Therefore, DISK keypoint detection and description was used as comparison standard in this chapter for algorithms implemented in kornia.

4.1 Keypoint description and detection

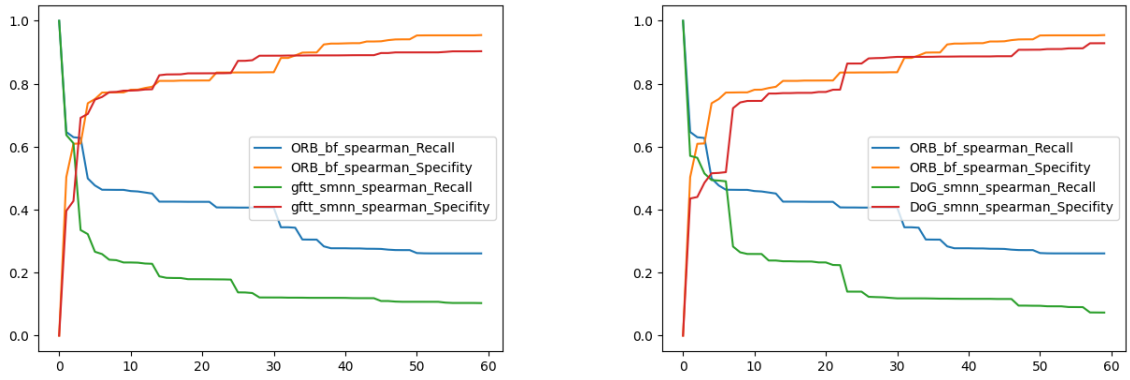
Function name	**Application**	**Implementation**
`Difference-of-Gaussian`	Keypoint detection	kornia [43]
`Shi-Tomasi corner detection`	Keypoint detection	kornia
`Dense SIFT [25]`	Keypoint description	kornia
`SIFT [25]`	Keypoint description	kornia
`MKDDescriptor [30]`	Keypoint description	kornia
`HardNet [27]`	Keypoint description	kornia
`Hardnet8 [38]`	Keypoint description	kornia
`HyNet [56]`	Keypoint description	kornia
`TFeat [1]`	Keypoint description	kornia
`SOSNet [57]`	Keypoint description	kornia
`ORB [48]`	Keypoint detection and description	OpenCV [5]
`SOLD2 [34]`	Keypoint detection and description	kornia
`DeDoDe [14]`	Keypoint detection and description	kornia
`DISK [58]`	Keypoint detection and description	kornia
`SIFTFeature [25],[43]`	Keypoint detection and description	kornia
`GFTTAffNetHardNet [29], [27], [43]`	Keypoint detection and description	kornia
`KeyNetAffNetHardNet [29], [4], [43]`	Keypoint detection and description	kornia

Table 4.1: Overview of the functions for keypoint detection and description used in this chapter.

[25] `https://kornia.readthedocs.io/en/latest/feature.html#matching`

4.1.1 Detectors

Kornia offers the `LocalFeature` module to combine feature detector and descriptor. Throughout testing it became clear that this did not work as intended and combining detection and description modules was not easily possible. To circumvent this incompatibility issue, the detector function could be applied to the image before using the detector. As this was not the way keypoint descriptors and detectors should be used, this probably explained the bad results.
The two detectors tested in this chapter were the *Shi-Tomasi corner detection* and the *Difference-of-Gaussian* (DoG) function. After applying the detection function, the DISK keypoint detection and description function was run on it to extract the keypoints because manually combining detectors with descriptors was hardly possible as instructions were sparse and testing often resulted in incompatible detectors and descriptors.



(a) Shi-Tomasi corner detection

(b) Difference-of-Gaussian

Figure 4.3: Comparing matching without using a detector beforehand against with the usage of a detector.

The results in Figure 4.3 present very obviously that the usage of detectors as implemented here is counterproductive.

4.1.2 Descriptors

Keypoint detectors and descriptors often work together to coordinate both modules and achieve the best performance. As the manual combination of detectors and descriptors did not work with the `LocalFeature` module, they were implemented in simple ways according to their functions.

The kornia library implemented nine descriptors. Seven of those descriptors worked with a so-called `patch_size`, whereas only two did not: The `DenseSiftDescriptor` computed its keypoints densely on the image like a kernel in a machine learning task. The other one was `SOLD2` which was a line detection algorithm.
Only two of the remaining seven detectors allowed setting the `patch_size` per argument, the other five detectors provided hardly any flexibility. The two algorithms providing `patch_size` changeable per argument were `SIFTDescriptor` and `MKDDescriptor`. The standard `patch_size` for these two was 41 and 32 and the `patch_size` for all other descriptors without changeable `patch_size` was also 32. For the present images had a size of 224×224 pixels, 32 was the comparison `patch_size` of this study.

4.1.2.1 Overview

These seven keypoint description functions took a considerable time to compute the images. Due to the time consumption the number of images being tested was reduced to a hundred. Table 4.3 provides an overview of the test results. These results were generated using the nearest neighbour function from the kornia library and summing up the distances for a first impression. These sums were transformed into distances by Spearman's rank correlation coefficient and clustered with *sklearn*'s implementation of agglomerative hierarchical clustering for 12 clusters with linkage set to complete.
The descriptors themselves were mostly left to standard configuration except for the `SIFTDescriptor` descriptor whose `patch_size` is adapted to 32 conforming to the others. The images (224×224) pixels were cut in 49 (7×7) patches 32×32 in size each. The descriptors were generated of those with the respective functions and fed into the matching function. If available the descriptor function was pretrained. Due to `SOLD2` not returning results compatible with DISK, its scores could not be computed (in this section) and only the remaining eight descriptors were compared. The following table shows the names of the descriptors and the results of each descriptor in the form of three values: recall, specifity and rand index compared to the "ground truth".

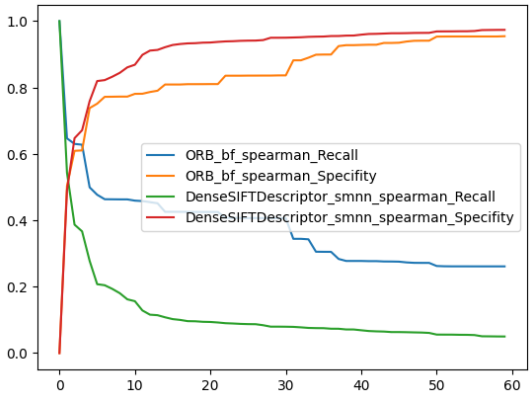
Descriptor name	Recall	Specifity	Rand index
`Dense SIFT [25]`	0.122	0.907	0.859
`SIFT [25]`	0.194	0.813	0.775
`MKDDescriptor [30]`	0.158	0.873	0.829
`HardNet [27]`	0.306	0.747	0.720
`Hardnet8 [38]`	0.144	0.876	0.831
`HyNet [56]`	0.234	0.797	0.763
`TFeat [1]`	0.144	0.840	0.798
`SOSNet [57]`	0.147	0.850	0.807

Table 4.3: Comparing the results of eight descriptor functions from the kornia [43] library. These tests were only made on a subset of 100 images.

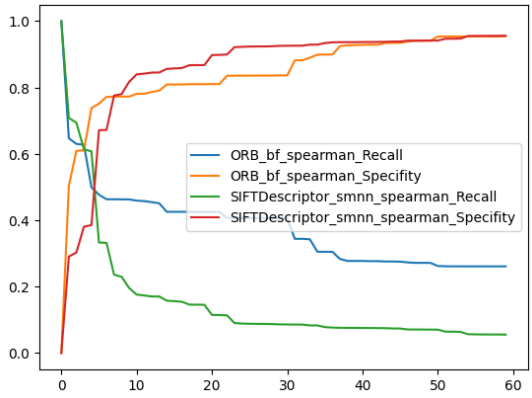
The results themselves did not show a clear ranking here. For most results recall and specifity summed up to around 1.0. The differences in rand index results were mainly caused by the number of true negatives being much higher than the number of true positives (556 to 8546). The descriptor with the highest recall, namely `Hardnet`, also had the lowest specifity and the function with the highest specifity had the lowest recall.
In order to provide a more in-depth comparison a subset of descriptors was selected: These descriptors were:

- `Dense SIFT` because it had the highest specifity,
- `Hardnet` because it had the highest recall, and
- the `MKDDescriptor` which was somewhere in between.

SIFT: For completeness, the results of both SIFT variants are shown (Figure 4.4). The dense SIFT descriptor on the left computed the keypoints densely over the image with a stride and padding of 1. The normal SIFT descriptors took patches of variable size, 32×32 in the present case.
ORB was already shown in [48] to produce similar or even better results than SIFT. The bad results probably resulted from the same issue as in the previous subsection: Manual combination of keypoint detectors and descriptors did not work. In the next subsection, a complete keypoint detector and descriptor using SIFT (`SIFTFeature`) was tested and gave evidence that SIFT could perform much better than seen here.



(a) Nearest neighbour matching with Spearman's rank correlation coefficient and DISK against Dense SIFT for keypoint detection and description.



(b) Nearest neighbour matching with Spearman's rank correlation coefficient and DISK against SIFT for keypoint detection and description.

Figure 4.4: Testing of kornia's built-in SIFT descriptor implementations against DISK.

The results of MKDDescriptors with a patch size of 32 were even slightly below the results from SIFT. This might also have been due to the matching method, as [48] recommended the use of "match kernels". These were not implemented in kornia though, therefore they were not tested.

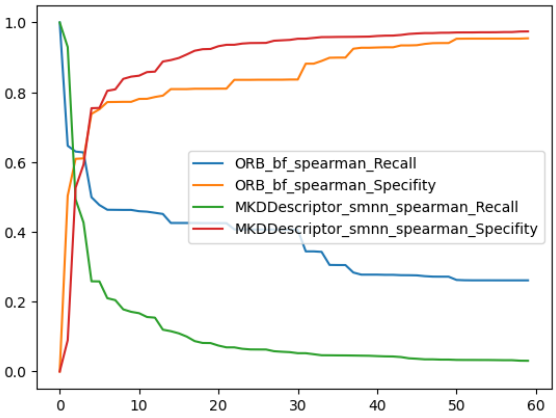
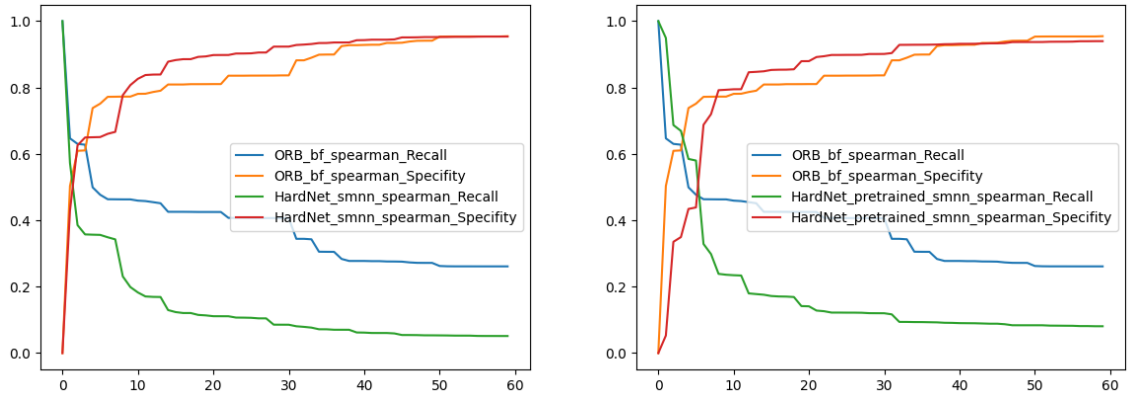


Figure 4.5: Comparison of the DISK keypoint detector and descriptor and the MKDDescriptor.

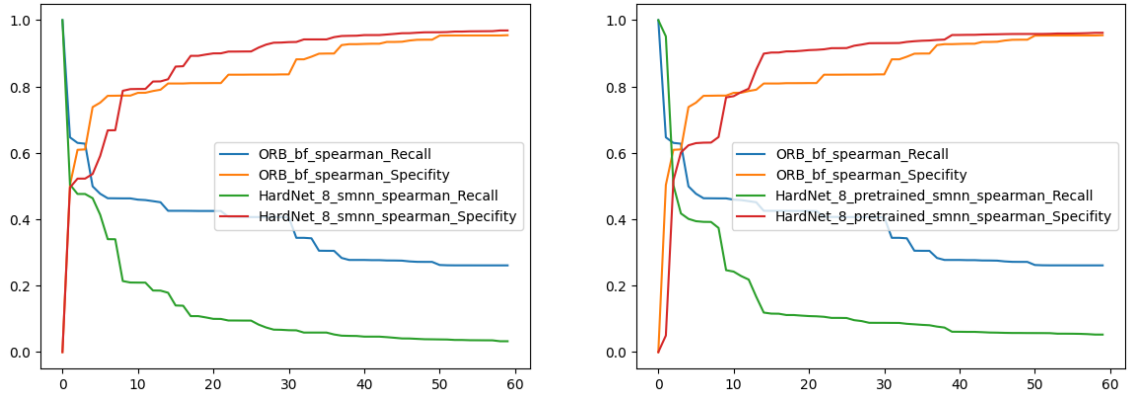
Another descriptor architecture was *Hardnet*[27]. Hardnet was a learned CNN descriptor and had its focus on Lowe's ratio test [24]. Lowe's ratio test checks that the distances between the two best matches for each keypoint were sufficiently different, otherwise the match was disregarded. The reason behind this was to eliminate matchings with high uncertainty or small significance due to the option's similarity.

The kornia library offered both a vanilla or a pretrained version of Hardnet and additionally an improved version of Hardnet based on [38] called *Hardnet8*. Hardnet8 promised to outperform Hardnet consistently.



(a) Hardnet

(b) Pretrained Hardnet



(c) Hardnet8

(d) Pretrained Hardnet8

Figure 4.6: Testing of Hardnet and Hardnet8 descriptor implementations against nearest neighbour matching with DISK keypoints. Both implementations used Spearman's rank correlation coefficient.

But as visible in the graphs in Figure 4.6, neither Hardnet nor Hardnet8 performed well (in comparison to DISK) on our data and using the pretrained version did not help much either. While the variants of Hardnet with and without pretraining had different results, no single function stood out as best.

Since a patch size of 32 performed very badly, it could be assumed that a different patch size might be more suitable for the size of the present images (224×224). Larger patch sizes did not

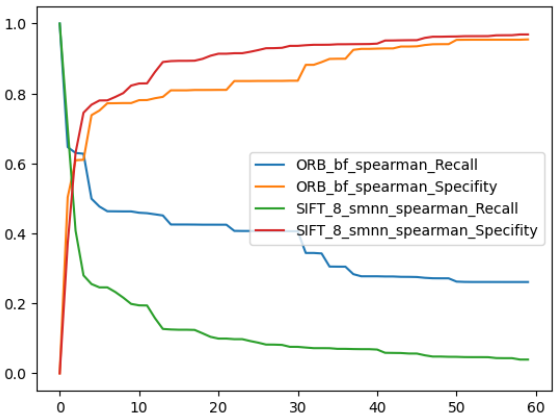


Figure 4.7: Comparison between the ORB keypoint detector and descriptor and SIFT descriptor with a patch size of 8.

make much sense for the small image size, but another benchmark of SIFT with a patch size of only 8 could provide a different result and possible improvement:
But while this test with the smaller patch size took much longer to compute, the results did not improve at all. As demonstrated in the two subsections about keypoint descriptors and about keypoint detectors, the result of these functions separated from each other did not result in good die studies; anyway, it had already become obvious that functions doing both the detection and description together could achieve better results compared to the results of ORB.

4.1.3 Detector and descriptor

This chapter presents functions, that both detect and describe the keypoints. This firstly leads to the detector and descriptor function used by Deligio et al. [12] that this thesis aims to improve on, namely ORB [48].
Figure 4.8 shows recall and specifity of the ORB algorithm with Brute-Force matching of the OpenCV library, as it was also used in [12]. It achieved quite high values for specifity ($\sim$ 0.3 for 53 clusters) and a high specifity of more than 0.8 for most of the graph. This was already a lot better than most other attempts though it lacked a little in terms of specifity for lower values of clusters.

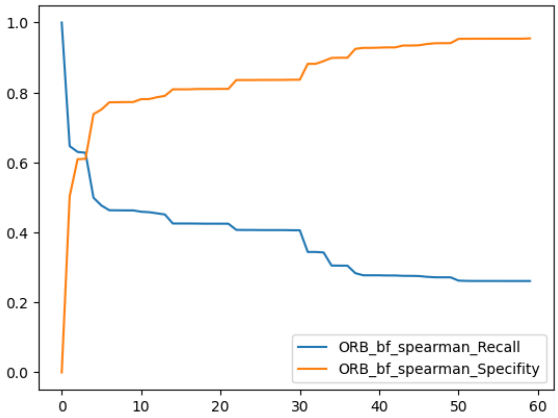


Figure 4.8: Results from ORB keypoint detection and detection with brute-force matching in OpenCV.

SOLD2 is a line segment detector. For the coins from the dataset did not contain many straight lines, the results were not expected to be very good. The normal configuration of SOLD2 in kornia was configured for images with sizes 400 $\sim$ 800px. Reducing parameters in the configuration allowed for adjusting this and also helped with the computation time from approximately more than 30 hours down to 15 hours.
The matching finally showed that no single match could be found which suggested that SOLD2 (in the used configuration) is not suitable for the present task. This also meant that no result could be computed as the distance matrix could not be computed on an empty matching result.

The DeDoDe-G algorithm took close to fifteen hours to detect and describe keypoints and match them which is around four times as much as ORB took. The results also fell short of the results by ORB in regards to recall and specifity, as shown in Figure 4.9.
The DISK model used was pretrained to a checkpoint called `depth`[26] and used the standard parameter of 128 features. As displayed in Figure 4.10a, when using DISK with the `smnn` matching function, much stronger results than with ORB were achieved.

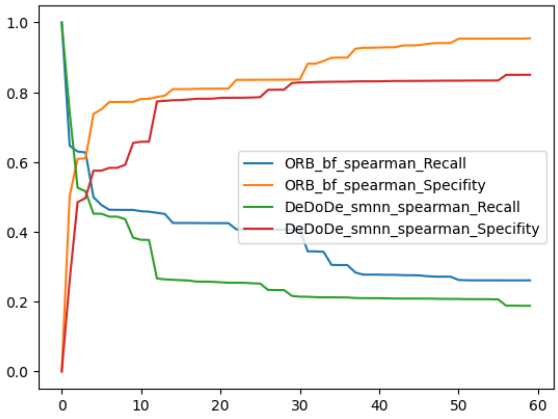
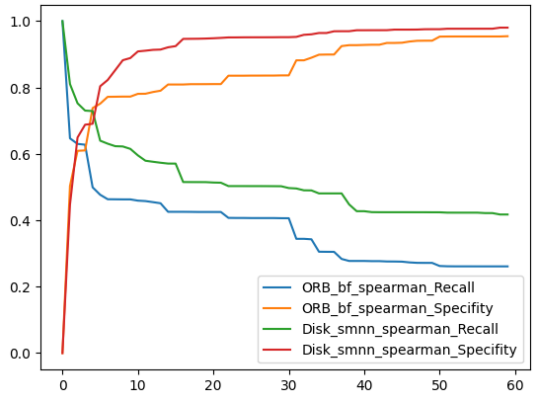
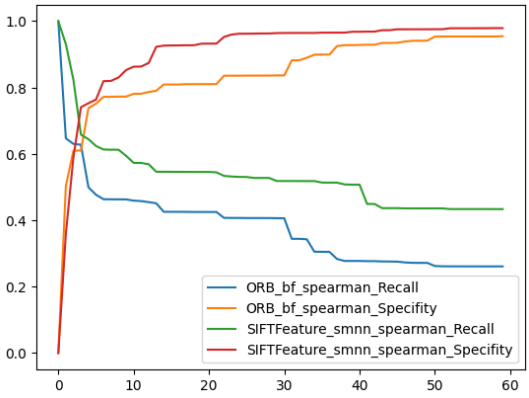


Figure 4.9: ORB keypoint detection and description next to DeDoDe keypoint detection and description.



(a) ORB keypoint detection and description next to DISK keypoint detection and description.



(b) ORB keypoint detection and description next to DoG keypoint detection and SIFT descriptors in the module `SIFTFeature`.

Figure 4.10: Testing of kornia's built-in SIFT descriptor implementations against DISK.

Using higher numbers of features did not improve the results and a small insight in the matching results led to the assumption that images of size 224×224 did not contain more than 128 keypoints with DISK. For images with higher resolution, this might have been different.
We already looked at SIFT in the subsection 4.1.2. The model `SIFTFeature` however

[26] `https://kornia.readthedocs.io/en/latest/feature.html#kornia.feature.DISK.from_pretrained`

combined DoG (Difference of Gaussian) detection with SIFT descriptors, providing for ease of use.

The `SIFTFeature` model uses 8000 features by default and also took over three times the computation time of DISK (which used only 128 features): about 15 hours. Regarding the performance, both SIFTFeature and DISK were both similar improvements to ORB, with SIFTFeature having achieved slightly higher recall and about the same specifity (see Figure 4.10), whereas DISK was much faster.

`GFTTAffNetHardNet` was another joint module, combining GFTT detector, AffNet for detection [29] and HardNet [27] descriptors. AffNet was a method for detecting local affine-covariant regions, which were more accurate than scale-covariant features that ORB uses. This algorithm, like SIFTFeature, used 8000 features by default and took a significant time to compute with nearly 17 hours of computation time for 1325 images. But unlike SIFTFeature recall of this function laid far below ORB's recall. And while having achieved higher specifity than ORB, both DISK and SIFTFeature also achieved this with much higher recall.

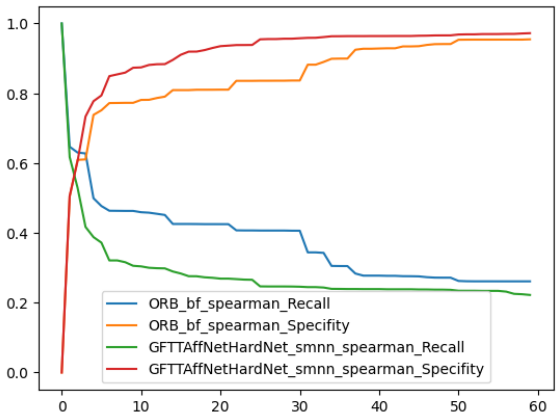


Figure 4.11: ORB keypoint detection and description next to GFTTAffNetHardNet keypoint detection and description.

`KeyNetAffNetHardNet` combined the KeyNet [4] detector with AffNet for detection and HardNet descriptors. The algorithm was very similar to GFTTAffNetHardNet except for the keypoint detector, but the performance was a lot better.

Figure 4.12 shows that KeyNetAffNetHardNet also performed better than ORB. Specifity is higher and with regards to recall, KeyNetAffNetHardNet was very close to the results of DISK and SIFTFeature. The computation time though was more than 17 hours and thus took even longer than SIFTFeature.

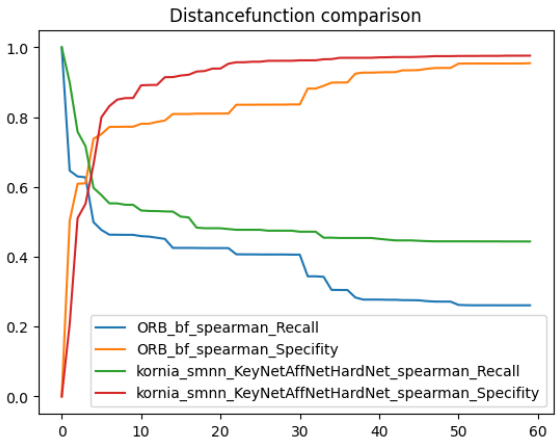


Figure 4.12: ORB keypoint detection and description next to KeyNetHardNet for keypoint detection and description.

4.2 Matching

In the literature, many image matching tasks focussed on the problem of detecting the exact same object in two different images. The object in these two images often had a different position, scale or viewing angle. These algorithms tried to tackle obstacles like high viewing angle differences of other geometry based differences. All these may also apply to a die study as the coins in the images were often turned slightly due to human inaccuracy. However, a specific characteristic of die studies for ancient coins is that there are no two coins exactly the same. Even coins made with the same die look different. To address this specific problem a function was needed that provided a continuous similarity output for each matching of two coins. Images of similar coins should achieve higher matching results than more different coins. Based on such similarities the diverse coin dies had to be derived.

Function name	**Implementation**
`BFMatcher`	OpenCV [5]
`FlannBasedMatcher`	OpenCV
`Nearest Neighbour Matching`	kornia [43]
`Mutual Nearest Neighbour Matching`	kornia
`Nearest Neighbour distance ratio matching`	kornia
`Mutual Nearest Neighbour distance ratio matching`	kornia
`First to first Geometrically Inconsistent Nearest Neighbour [28]`	kornia
`AdaKAM [9]`	kornia
`LightGlue [23]`	kornia
`LoFTR [54]`	kornia

Table 4.4: Overview of the functions for image matching used in this chapter.

4.2.1 Baseline

The method described by Deligio et al. [12] employed the ORB-method from the OpenCV library which itself used so-called `BRIEF`-descriptors. These were combined with the `cv.BFMatcher()` that performed "nearest neighbour matching". This matcher then returned the matched keypoint-pairs together with the distances for

each pair.
Deligio et al. also filtered these matching pairs by distance, assuming that a higher distance between a pair of keypoints could hint towards a wrong matching. Therefore all matchings with a distance higher than the threshold, set at 45, were removed from the matching. The counted number of remaining matches for a pair of images represented the similarity of the matching.

4.2.2 Improvements & changes

At the outset other matching options within OpenCV are presented, staying close to the solution by Deligio et al.
Next, the kornia library got the attention which implemented many more matching functions which were either standards or newer additions to the literature. Once again Deligio et al.'s pipeline with ORB in OpenCV was taken for reference. For comparability reasons, DISK was applied here for keypoint detection and description. It was not only proven very effective in Figure 2.1.2 but also computed results much faster than SIFTFeature.

4.2.3 More matching functions

There were two so-called `DescriptorMatcher` in the OpenCV library, `BFMatcher` (brute force matcher) and `FlannBasedMatcher`. The brute force matcher just used a given distance metric and returned the closest match for each keypoint. Testings within this paper indicated that the approximate matching algorithm FLANN produced worse results than the BFMatcl
Both of these matchers supported the `match()` and the `knnMatch()` functions. The `match()` function returned only the best match for each keypoint. The `knnMatch()` function returned the best k matches.
Using `knnMatch()` with parameter $k = 2$ allowed to apply Lowe's ratio test [24]. As shown in Figure 4.13 this led to a significant improvement in specifity. On the other hand, recall also improved for larger numbers of clusters.

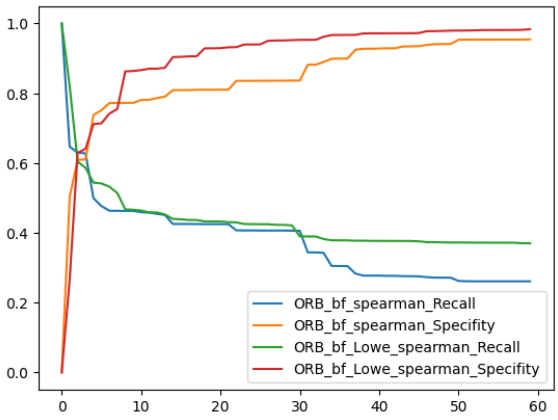


Figure 4.13: Comparison between the original matching and a variant using k-nearest-neighbour matching with parameter $k = 2$.

The trade-off as shown in Figure 4.14 was not gainful. Higher recalls for a lower number of clusters were traded off against a loss of specifity, especially when compared to the matching with Lowe's ratio test. Unfortunately, Lowe's ratio test and "crosscheck" could not be combined in OpenCV[27].

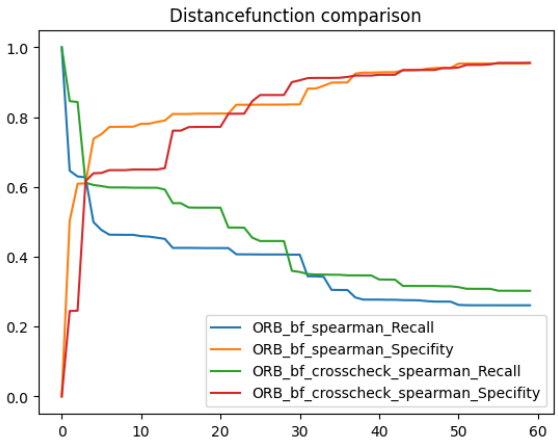


Figure 4.14: Comparison between OpenCV matching with Lowe's ratio test versus matching with crosscheck (mutual matching).

OpenCV library's options for matching functions were exhausted at this point. But different options to interpret the matches were still worth testing, like counting the number of all found matches, the average distance of matches and other obvious options. But they all ended up in more or less small aggravations of the result. At the end counting the number of matches below the distance threshold 45 still proved to be the best interpretation of the matching results.

There were many more matching options available. The kornia library [43] provided lots of implementations in Python. It therefore was a great tool for testing better matching options and developing more potent matching pipelines.

As already shown in the "Benchmarks and recommendations" section of `kornia.feature`[28] many of the algorithms presented and implemented here promised improvements over OpenCVs implementations. These were implementations of many image matching functions from the literature. Some of these functions required different approaches or differently preprocessed data. The following paragraphs provide information on implementations used in this thesis.

DISK [58] was recommended as a strong feature detection and description tool for outdoor scenes. LoFTR, the recommendation for indoor scenes, was not compatible with other matching functions. Therefore DISK was used for the tests of the matchers from the kornia library (where possible).

Nearest Neighbour: The nearest neighbour matching function in the kornia library provided three added upon nearest neighbour matching functions, similar to the

[27]The occurring error was also described here: `https://github.com/MasteringOpenCV/code/issues/46`

[28]`https://kornia.readthedocs.io/en/latest/feature.html#benchmarks-and-recommendations`

modifications to nearest neighbour matching in OpenCV in the passage before. In order to better compare these nearest neighbour matching variants, firstly nearest neighbour matching is presented: for each descriptor in the first image, the closest descriptor in the second image was matched.

For testing purposes, DISK was configured the same way as in Figure 2.1.2. The graphs in Figure 4.15 demonstrated that kornia's standard nearest neighbour matching with DISK keypoint detection and description produced overall slightly worse results than the brute-force matching with SIFT from OpenCV: OpenCVs brute-force matching had a higher specifity and the nearest neighbour matching resulted in a higher recall.

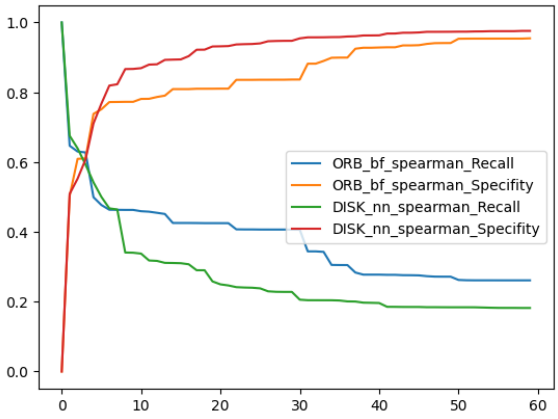


Figure 4.15: Comparison between ORB keypoints with nearest neighbour matching in OpenCV and DISK keypoints with nearest neighbour matching in kornia.

A slightly different interpretation of the computed matches was applied for this method: instead of using the number of matches below a certain distance threshold, just the sum of all distances proved most advantageous.

For the *Mutual nearest neighbour* algorithm the interpretation of the ORB-matches was used, i.e. counting the matches that drop below a certain distance threshold. Due to different computation methods, the threshold suitable for this matching algorithm was only 1, as distances returned by this algorithm were far smaller.

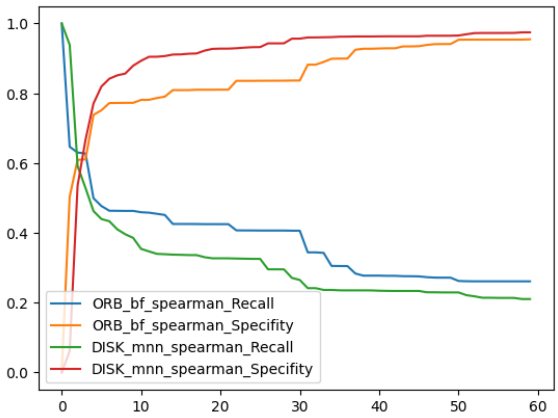


Figure 4.16: Comparison between nearest neighbour matching and mutual nearest neighbour matching.

Figure 4.16 displays that mutual nearest neighbour matching provided a small but clear advantage compared to normal nearest neighbour matching. This improvement only applied to recall. Specifity was around the same for normal nearest neighbour matching. The mutual nearest neighbour matching was meant to be the same as the `match()` function in OpenCV with `crosscheck=True`.

Nearest neighbour distance ratio matching applied Lowe's ratio test [24] with a selected threshold of 0.85. The interpretation of the matches used again the sum of distances of all matches.

Figure 4.17 indicates that the distance ratio threshold resulted in an improvement in recall compared to the normal nearest neighbour matching. Regarding recall, these improvements were on par with the OpenCV implementation with ORB but kept better specifity already achieved in nearestneighbour matching. For higher numbers of clusters recalls even outperformed recalls by the OpenCV method, although the improvement in specifity got smaller.

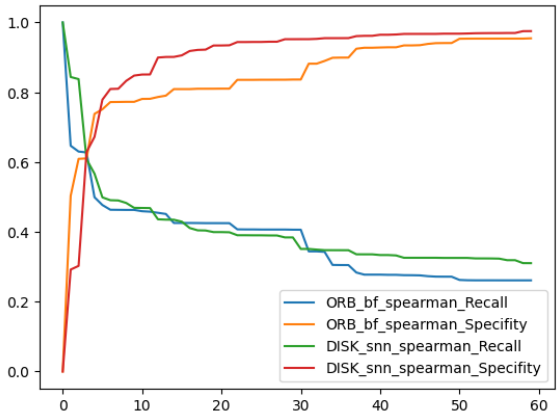


Figure 4.17: Comparison between nearest neighbour distance ratio matching and nearest neighbour matching.

As now both the *mutual nearest neighbour* and the *nearest neighbour distance ratio* improved upon the vanilla *nearest neighbour matching*, the combination of both in *Mutual nearest neighbour distance ratio matching* was expected to yield the best results yet:

Figure 4.18 presents the result of Mutual nearest neighbour distance ratio matching. It turned out to be a major improvement on all previously tested nearest neighbour variants. This improvement occurred mainly within specifity whereas recall had left little opportunity for improvement anyway. It was a profound improvement against OpenCV matching with ORB in terms of both specifity and recall. In conclusion, Mutual nearest neighbour distance ratio matching made its competitors irrelevant.

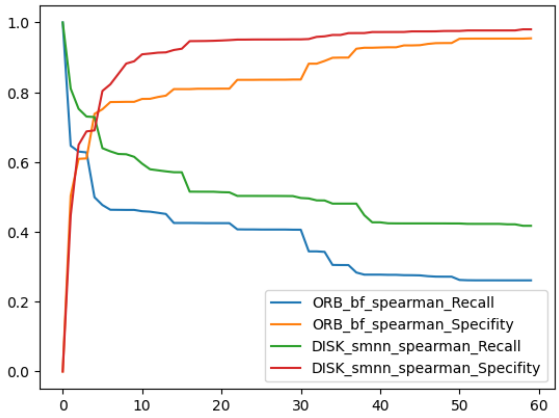
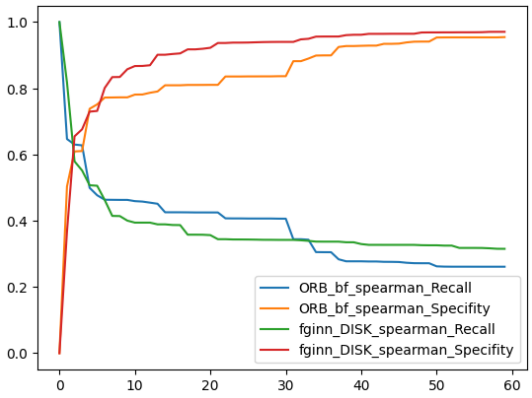


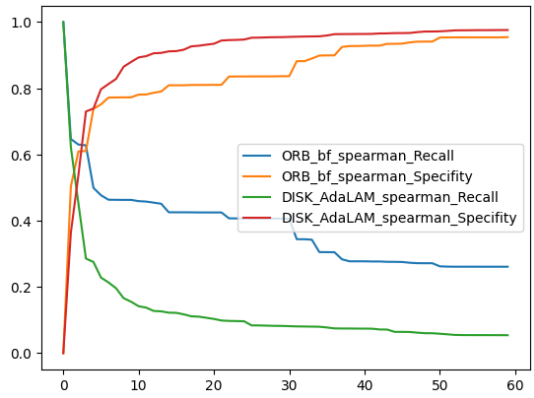
Figure 4.18: Comparison between mutual nearest neighbour distance ratio matching and nearest neighbour matching.

First to second Geometrically Inconsistent Nearest Neighbour was another variant of nearest neighbour matching. But as demonstrated in Figure 4.19a FGINN performed better than standard nearest neighbour matching on DISK descriptors, but was far behind the results of mutual nearest neighbour distance ratio matching. Mishkin et al. recommended in [28] the usage of vector-based descriptors like SIFT. However,

the results with SIFT were even worse for our implementation than standard nearest neighbour matching.



(a) Comparison between first to first Geometrically Inconsistent nearest neighbour matching and nearest neighbour matching.



(b) Comparison between AdaLAM matching and nearest neighbour matching.

Figure 4.19: Testing of kornia's built-in SIFT descriptor implementations against DISK.

According to Figure 4.19b, it was obvious that the results of *AdaLAM* using DISK keypoints were far behind the results of nearest neighbour matching. The parameters of the algorithm were set as in a tutorial for DISK and AdaLAM[29] i.e. using a number of features of 2048.

For this matcher's implementation in kornia required the local affine frames of each image, the `get_laf_orientation()` function from kornia was used to compute those and passed them on to the LightGlue matcher. Again, computing the sum of all distances from the matchings proved most effective as interpretation of the matchings.

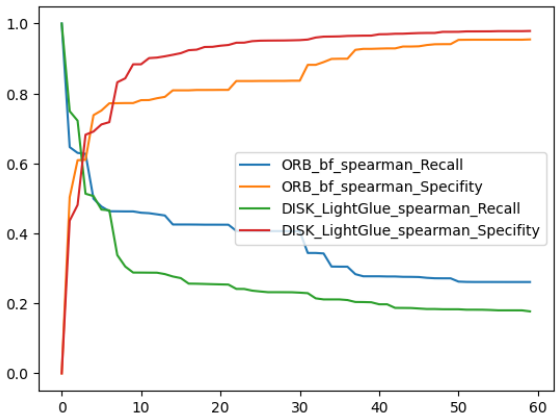


Figure 4.20: Comparison between LightGlue and nearest neighbour matching.

While being able to present top-level results in standard image matching tasks, like matching the same object amongst multiple images, LightGlue did not achieve good results in the present case. With regard to high expectations such a result was rather astonishing, but there were (mainly) two reasons that might

[29] `https://kornia.github.io/tutorials/nbs/image_matching_disk.html`

have led to this result: Firstly, the image selection was very different to the images found in the respective literature. This could refer to neural networks having been badly pretrained for the present application. Secondly, the algorithm might have been too exact. Since the algorithm in the present case was never presented with images of the exact same object (coin), the strengths of the algorithms might not have been able to come into effect. There was a chance that this could change for a more manually adapted version of LightGlue though.
Again, we used the sum of distances as interpretation.
The *LoFTR* algorithm pretrained on the `indoor_new` dataset was used. As shown in Figure Figure 4.21 the results of LoFTR on the present issue were not competitive with the nearest neighbour variants. LoFTR even struggled to keep up with the standard nearest neighbour matching.
The best interpretation for the matchings found was summing up the confidence scores of all matches.
From all the matching functions, the Mutual nearest neighbour distance ratio matching function alone returned much better results with DISK keypoints compared to the other ones.

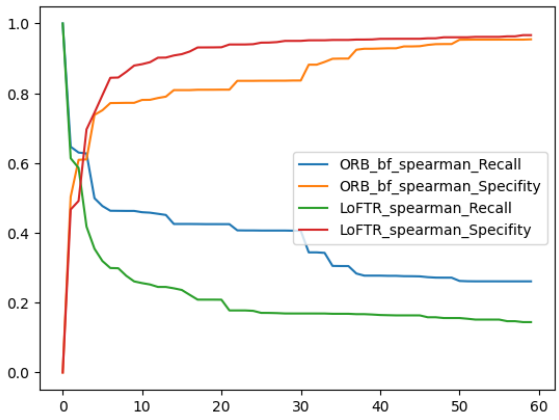


Figure 4.21: Comparison between LoFTR and nearest neighbour matching.

4.3 Distance function

The matcher returns matches and other information which can be understood as similarities (numbers) being returned. While this similarity matrix can be used as distance matrix, applying certain functions on the similarity values might improve their behaviour as distances and improve the overall results from the clustering.
Deligio et al. [12] applied Spearman's rank correlation coefficient on the similarity values which operates similarly to the Pearson correlation coefficient.
Spearman's rank correlation coefficient was a comprehensible first choice, but many different options existed for this task: the `scipy.spatial.distance` module contained twelve functions that were labelled "Distance functions between two numeric

vectors[...]"[30]. Additionally within `scipy.stats`, neighbouring Spearman's rank correlation coefficient, ten other functions that were labelled under "Association/ Correlation Tests"[31] were promising options, where both Spearman's rank correlation coefficient and the Pearson correlation coefficient, as well as other standards like the cosine distance function, could be found implemented.
Since there were so many options, a preselection was chosen with the help of a table of precomputed values for each function. Prior to this, some functions related to Spearman's rank correlation coefficient were evaluated.
With regard to testing the functions DISK for keypoint detections and description, the mutual nearest neighbour distance ratio matching function as described in section 4.2 and agglomerate clustering with complete linkage were used.

Function name	**Implementation**
`Spearman's rank correlation coefficient [53]`	Orange [13]
`Pearson correlation coefficient`	Orange
`Kendall Tau-b [19], [20]`	`scipy.stats` [60]
`Cosine similarity function`	`scipy.spatial.distance` [60]
`Linear least-squares regression`	`scipy.stats`
`Point biserial correlation coefficient [22]`	`scipy.stats`
`Somers' D [52]`	`scipy.stats`
`Bray-Curtis distance`	`scipy.spatial.distance`
`Canberra distance[21]`	`scipy.spatial.distance`
`City Block (Manhattan) distance`	`scipy.spatial.distance`
`Correlation distance`	`scipy.spatial.distance`
`Euclidean distance`	`scipy.spatial.distance`
`Jensen-Shannon distance (metric)`	`scipy.spatial.distance`
`Minkowski distance`	`scipy.spatial.distance`
`Standardized Euclidean distance`	`scipy.spatial.distance`
`Squared Euclidean distance`	`scipy.spatial.distance`
`Yule dissimilarity`	`scipy.spatial.distance`

Table 4.5: Overview of the functions for distance computation used in this chapter.

[30]Source: `https://docs.scipy.org/doc/scipy/reference/spatial.distance.html`
[31]Source: `https://docs.scipy.org/doc/scipy/reference/stats.html#association-correlation-tests`

4.3.1 Correlation coefficients

Spearman's rank correlation coefficient [53] was the standard distance function in [12] and achieved good results. Therefore it was employed here to serve as a comparison with other distance functions.

An obvious competitor was the Pearson correlation coefficient, the continuous variant of Spearman's rank correlation coefficient. Due to it not using ranks, it could depict the distance relations more accurately.

This expected improvement manifests in the results in Figure 4.22, improving greatly in terms of recall and even improving specifity.

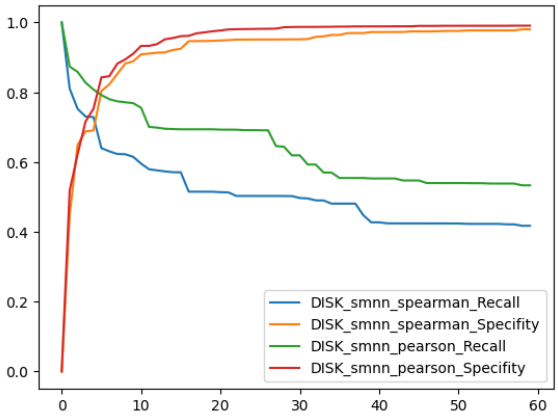


Figure 4.22: Comparison between Spearman's rank correlation coefficient and the Pearson correlation coefficient.

The *Kendall rank correlation coefficient* was another rank correlation coefficient, which had different variants for different options of handling ties amongst values. The Kendall Tau-b was such a variant. Figure 4.23 displays that it was worse than Spearman's rank correlation coefficient, not just regarding recall and specifity, but also computation time: This method took nearly 70 times longer to compute.

The Scipy [60] library provided eight other functions next to the Pearson, Spearman's rank correlation coefficient and Kendall's Tau, which were listed in the `scipy.stats` section as "Association/Correlation Tests". However, not all of those were feasible for the present data and objective. Therefore three out of eight functions were chosen that fit the purpose `linregress`, `pointbiserialr` and `somersd`.

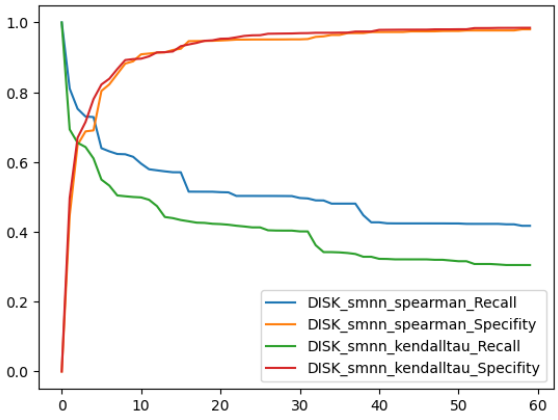


Figure 4.23: Comparison between Spearman's rank correlation coefficient and Kendall Tau-b.

`linregress`, `pointbiserialr` and `somersd` were part of a bigger evaluation of functions (cf. Table 4.6) in order to sort out less worthwhile contenders of distance functions without having to compute the whole clustering for all of them.

4.3.2 Further distance functions

Apart from the determination of similarity through correlation coefficients, there were functions not based on correlation coefficients which could be useful, too. One candidate was the cosine distance function and the related cosine similarity.

While the cosine distance function performed quite well regarding recall, it weakened regarding specificity. Pertaining to Spearman's rank correlation coefficient the differences are getting smaller for higher numbers of clusters, but a clustering with Spearman's rank correlation coefficient achieves such results with fewer clusters, too.

The Cosine distance was not a bad competitor to Spearman's rank correlation coefficient, but it performed not as well as the Pearson correlation coefficient did.

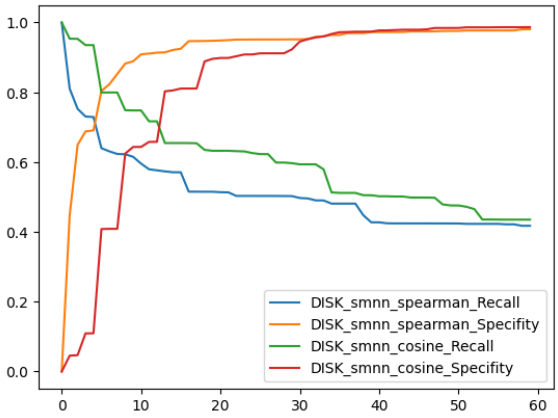


Figure 4.24: Comparison between Spearman's rank correlation coefficient and cosine distance.

Ten further functions from `scipy.spatial.distance` were listed as distance functions for a pair of numeric vectors, but among them the `mahalanobis` function did not work on square matrices which was needed. Hence, nine functions were left for testing. These functions were implemented using the `scipy.spatial.distance.pdist()` function to easily run them on the whole matrix.

The `yule` function from `scipy.spatial.distance` was a distance function for boolean vectors which was tested accidentally. When having computed values for it they were surprisingly good, therefore it was added to the test.

All of these functions were tested with the original ORB matching and with the nearest neighbour matching from kornia with DISK keypoints and agglomerative hierarchical clustering.

These data values differ from previous values in the following ways: The values visible in the Table 4.6 were computed on only a subset of the data containing the first 100 images. Further, the ground truth file contained 53 classes for the obverse of the coins. As just a subset of the data was used, the clusterer was given a smaller number as argument: the chosen argument for the number of clusters was 12.

Function name	ORB R.	ORB S.	ORB R.i.	NN R.	NN S.	NN R.i.
`None`	0.097	0.879	0.831	0.518	0.899	0.876
`Spearman`	0.219	0.804	0.768	0.432	0.879	0.851
`Pearson`	0.309	0.687	0.664	0.640	0.923	0.906
`Cosine`	0.396	0.620	0.606	0.525	0.933	0.908
`linregress`	0.104	0.922	0.872	0.626	0.950	0.930
`pointbiserialr`	0.104	0.922	0.872	0.626	0.950	0.930
`kendalltau`	0.151	0.913	0.866	0.291	0.914	0.876
`somersd`	0.151	0.913	0.866	0.291	0.914	0.876
`braycurtis`	0.252	0.768	0.737	0.399	0.915	0.884
`canberra`	0.122	0.856	0.812	0.579	0.928	0.907
`cityblock`	0.349	0.713	0.690	0.399	0.915	0.884
`correlation`	0.209	0.823	0.785	0.640	0.938	0.920
`euclidean`	0.302	0.723	0.697	0.658	0.948	0.930
`jensenshannon`	0.212	0.786	0.751	0.626	0.938	0.919
`minkowski`	0.302	0.723	0.697	0.658	0.948	0.930
`seuclidean`	0.270	0.765	0.734	0.626	0.950	0.930
`sqeuclidean`	0.302	0.723	0.697	0.658	0.948	0.930
`yule`	0.119	0.921	0.872	0.640	0.950	0.931

Table 4.6: R. = Recall, S. = Specifity, R.i. = Rand index. Comparison of distance function options. These tests were made on just 100 images once with ORB in OpenCV (ORB) and once with DISK and nearest neighbour matching (NN). Values were rounded to the third decimal place.

Before having analysed the performance of these functions some conspicuousness evolved: firstly, the number of possible true positives was a lot smaller than the number of true negatives (556 against 8564). Therefore the rand index was mostly influenced by the specifity. Secondly, except for `somersd`, the diverse distance functions had very similar computation times.
Some functions produced identical results. Just one of those had to be tested. The groups of distance functions with the same results were:

- `linregress` and `pointbiserialr`
- `kendalltau` and `somersd`
- `euclidean`, `minkowski` and `sqeuclidean`

Looking at the results computed with the nearest neighbour matcher with DISK keypoints which overall were better than than results with ORB, all except two functions had a specifity over 0.9. Recall had a broad spectrum ranging from 0.29 up to 0.658. The following functions (from those not already being examined) appeared less suited:

`None`: Using no distance function with the result from nearest neighbour matching was not particularly bad with a recall of 0.518, but had the second worst specifity. With the ORB matching, this function achieved the worst result in recall. While it performed surprisingly well for using no distance function, this function was not considered any further.

Both `Braycurtis` and `cityblock` only returned average results in both matching scenarios.

`Canberra` had similar results. While it had a recall close to 0.6 with the nearest neighbour matching setting, many other functions were simply better. As `canberra` also performed very badly on the ORB matching data, the focus was directed on the remaining functions.

For duplicate functions were omitted, the following functions remained for further testing: `linregress`, `correlation`, `euclidean`, `jensenshannon` and `yule`.

When testing these functions were compared to the Pearson correlation coefficient instead of Spearman's rank correlation coefficient, because this function was the strongest competitor so far.

Linear regression returned multiple arguments. The decision of whether to choose the argument "slope" or "rvalue" as distance turned out irrelevant since both produced the same result.

The performance of `linregress` on the present data was similar to the results of Spearman's rank correlation coefficient and therefore quite a bit worse than the results from the Pearson correlation coefficient. The difference to the Pearson correlation coefficient and likewise to Spearman's rank correlation coefficient was mainly in the recall; specifity appeared pretty stable throughout the testing of most functions.

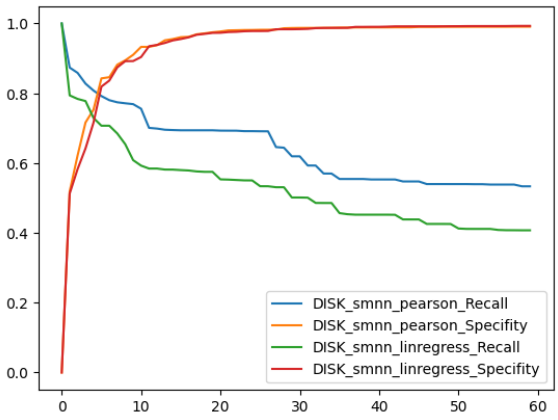


Figure 4.25: Comparison Pearson rank correlation coefficient against the slope of a linear regression.

`correlation` stands for the *correlation distance*. The correlation distance overtook the performance of the Pearson correlation coefficient at one point regarding the recall. But on average it still had a recall of around 0.05 below the Pearson correlation coefficient. On different datasets or under other circumstances this function might be a viable alternative or even outperform the Pearson correlation coefficient. Overall this was the strongest alternative to the Pearson correlation coefficient yet.

The *euclidean distance* is the classical distance method and expectedly performed very well on the present data. On the other hand, the Euclidean distance had a slightly worse specifity than most other distance functions.

Having performed even slightly better than the correlation distance regarding recall and at some points even better than the Pearson correlation coefficient, on average recall of the Euclidean distance remained below recall of the Pearson correlation coefficient.

This difference got smaller for higher amounts of clusters though.

Looking at the results, the low specifity of the *Jensen-Shannon distance* was striking. Next to this, the Jensen-Shannon distance offered a really good recall that might even have exceeded recall of the Pearson correlation coefficient on average, though not recall of the Pearson rank coefficient.

The Jensen-Shannon distance's good re-

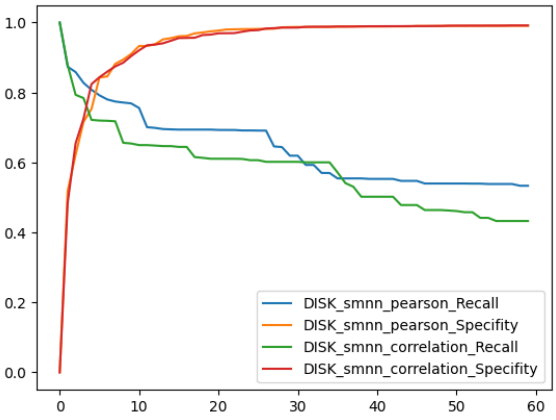


Figure 4.26: Comparison Pearson rank correlation coefficient against the correlation distance.

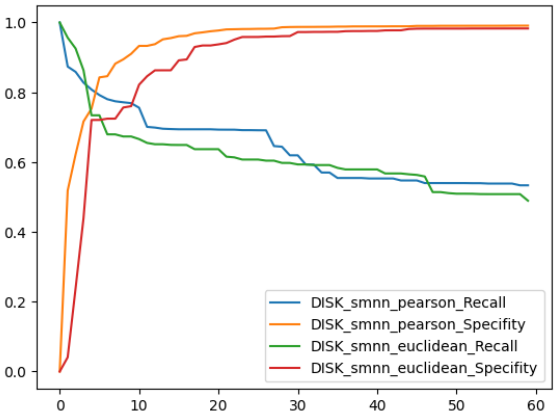


Figure 4.27: Comparison Pearson rank correlation coefficient against the correlation distance.

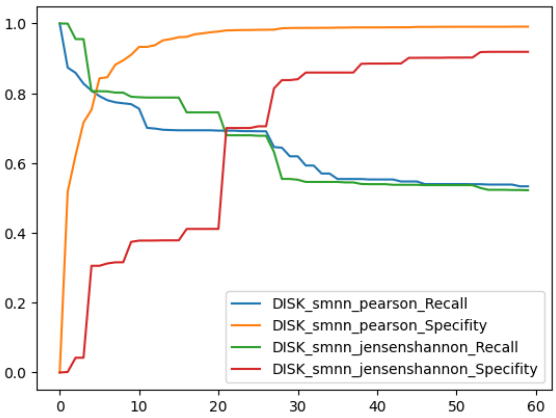


Figure 4.28: Comparison Pearson rank correlation coefficient against the Jensen-Shannon distance.

call with bad specifity still left the Pearson correlation coefficient as the overall best function. However, the Jensen-Shannon distance computed decent results for higher cluster amounts.

The *Yule dissimilarity* performed a little better than Spearman's rank correlation coefficient. But regarding the recall, it still performed an average of 0.1 worse compared to the Pearson correlation coefficient, while performing very similarly regarding specifity. This is especially impressive when learning that this function was written for a different purpose.

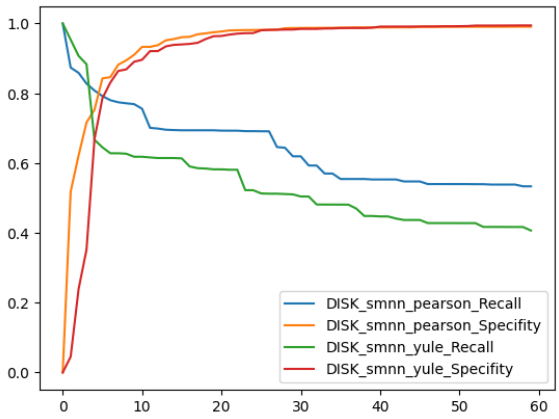


Figure 4.29: Comparison Pearson rank correlation coefficient against the Yule dissimilarity.

With the *smnn* matcher and DISK keypoints, the values returned by the input in the distance function rarely exactly equalled 1, but many values equalled 0. Per definition of the Yule dissimilarity, this was the only real input given to the function. Other matchers or interpretations of its result might therefore be able to improve results with this distance function.

4.4 Results

Beginning with keypoint detection and description, mainly two functions stood out of the rest: DISK and SIFTFeature which both are keypoint detection and description algorithms (although SIFTFeature is just a convenience module combining DoG and SIFT). KeyNetAffNetHardNet also produced good results but took slightly longer than SIFTFeature together with slightly worse results.

Regarding matching functions mutual nearest neighbour distance ratio matching stood far above the rest. This might only apply to the data and implementations used for this thesis, but with those, it performed clearly best.

Regarding the distance function, the Pearson correlation coefficient performed best. The cosine distance function and the Euclidean both had good results too but traded a higher recall for lower specifity while still not achieving the same amount of recall as the Pearson correlation coefficient.

Chapter 5

Clustering

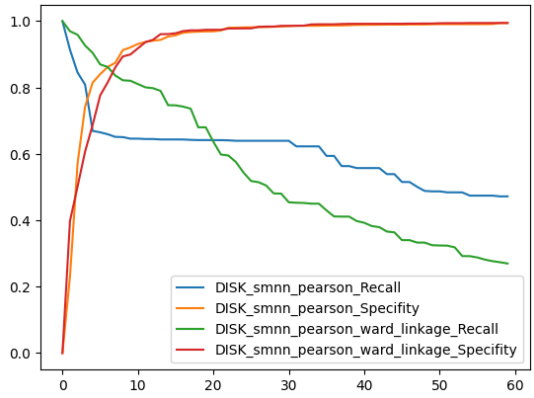
After having computed the distances between the images, the last step was to cluster the data. The clustering subdivided the distances into clusters by applying a distance function. The generated clusters were meant to just consist of images being close to each other, while images of different clusters are more distanced.

5.1 Baseline

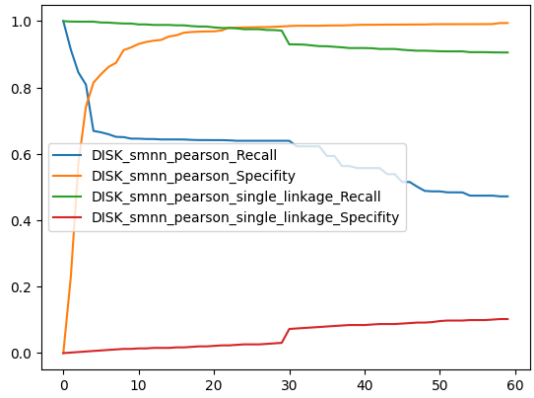
Deligio et al.' [12] used Orange's [13] implementation of agglomerative hierarchical clustering with complete linkage. The clustering algorithm stopped at the value threshold of 0.3 at which point the clustering was returned (when the smallest distance between clusters is larger than 0.3).

5.2 Improvements and Changes

Rather than setting up value thresholds, the clustering algorithm was provided with numbers of clusters to get a more controlled environment for testing and to create graphs like in subsection 2.3.1.
Three other linkage options were implemented in the SciPy library `SciPy` library: "ward", "average" and "single".
The *ward* linkage only accepted the "Euclidean" metric option which differed from the "precomputed" metric option that was chosen for the other linkage options in the tests in this thesis. While single linkage produced very bad results, the results from *average* linkage were better than complete linkage regarding recall for small numbers of clusters but struggled heavily with higher numbers of clusters.



(a) Clustering with complete linkage against clustering with ward linkage.



(b) Clustering with complete linkage against clustering with single linkage.

Figure 5.1: Hierarchical clustering ward linkage and single linkage variants.

Single linkage was a better option. While not achieving that high values in specifity, the recall was much higher compared to all other clustering options. Additionally, the point of intersection of recall and specifity was higher than in any other clustering variant.

Spectral clustering [51] was also tested. It turned out to be slightly worse than complete linkage hierarchical clustering regarding both recall and specifity.

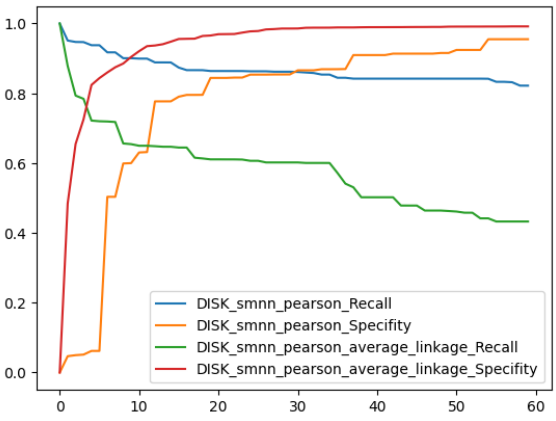


Figure 5.2: Clustering with complete linkage against clustering with average linkage.

HDBSCAN [8] did not need a number of clusters as input which made it a good alternative for datasets with an unknown number of clusters. Figure 6.1 displays the results of HDBSCAN which remained unaffected by the changing number of clusters among the x-axis. In the present example, the number of clusters computed by HDBSCAN was 22. It could not match the results of hierarchical clustering with average linkage though, at least with standard parameters and "precomputed" distances.

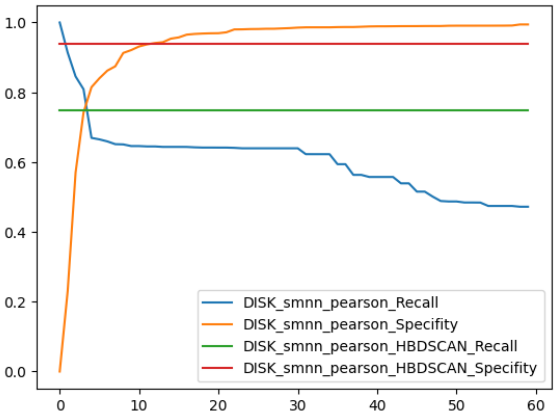


Figure 5.3: Hierarchical clustering with complete linkage against HDBSCAN.

5.2.1 AGLP clustering

Another way of clustering was presented by Cornet et al. in [11]: "Adaptive Graph Label Propagation" applied a Label Propagation Algorithm [41] with a threshold τ^*. The optimal parameter τ^* was determined by computing the Silhouette Coefficient [47] for all possible τ and selecting the τ with the best quality of clusters according to those Silhouette Coefficients.
Trying to implement this as described by Cornet et al.[32] spawned two things: firstly, running their whole die study on all 1325 images would have taken 32 hours. Therefore the amount of images for testing was reduced to only 100 images. Clustering then was successful, but running the evaluation function provided by Cornet et al. was not feasible. Secondly, when monitoring the computed clustering, it clustered all images into one cluster, except for one image. This led to the conclusion, that either the present data did not synergise well with their algorithm or the application of their tool was erroneously run.

5.3 Result

Hierarchical clustering with average linkage provided the best results from all clusterers in this chapter. However, HDBSCAN appeared to be a good alternative for datasets with a yet unknown number of clusters (coin dies used). As the focus of this thesis was put more on the matching process, many good options for clustering had to be left untested and HDBSCAN remained only tested with mostly vanilla arguments.

[32] `https://github.com/ClementCornet/Auto-Die-Studies`

Chapter 6

Further improvements

6.1 Estimator

Many of the tutorials on image matching by kornia[33] suggested the use of estimators to filter out matches that did not fit the schema. RANSAC [16] is one option that estimated the geometric transformation between two images and filters out matches not conforming to this.

An improved version of this was `USAC_ACCURATE`. Due to errors in the process, the way the estimator was implemented slightly differed from the implementation in the tutorials.

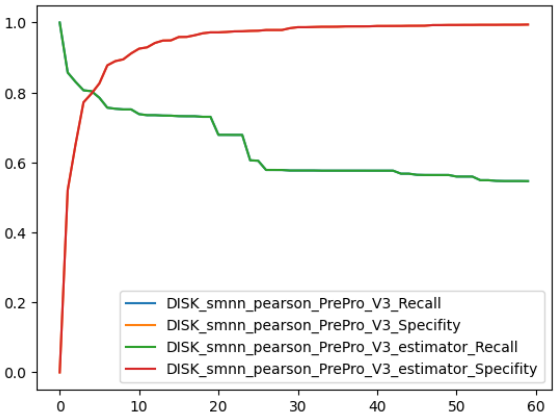


Figure 6.1: Using the estimator USAC˙ACCURATE to filter our bad matches.

The usage of the `USAC_ACCURATE` estimator after the matching step did not change the results in any way. This might be due to the small amount of rotation or other geometric transformation in our data so that its effect could not be determined in this thesis. But depending on the input data or the parameters used, it might have an influence on the results.

[33] `https://kornia.github.io/tutorials/`

6.2 Variable image sizes

The images of all coins were scaled to 224 × 224px. Because not every coin was perfectly round, the resizing process compressed or stretched some images. While most descriptors should be invariant to this form of transformation, keeping the aspect ratio could have still helped with the die study.

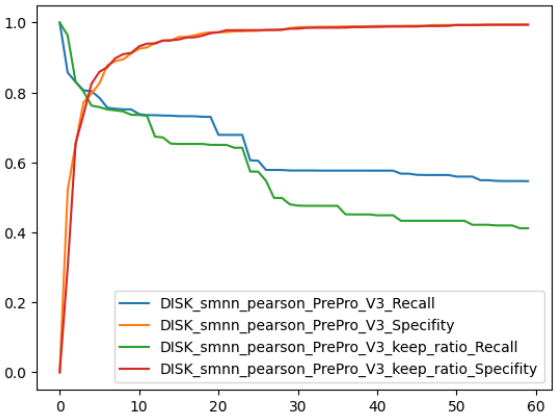


Figure 6.2: Keeping the size ratios of the images while preprocessing.

Figure 6.2 indicates that trying to preserve the size ratio of the image did not actually work as expected, as recall gets much lower when preserving the original ratio of the images. While the reason for this decrease in performance remained unknown, this technique was not applied further.

Chapter 7

DieStudyTool

A tool to use the techniques described in this paper is provided at [34]. The installation and user guide can be found at this URL too.
This tool contains a Jupyter notebook `DieStudyTool.ipynb` at its core that can be used to run die studies similar to the ones described throughout this paper on one's own data.
The standard parameters configure the die study pipeline such that it performs according to the first pipeline described in section 8.1. The functions can be changed to nearly any function described in this paper. It is also written in a way that should allow adding further functions to any step in the pipeline.
The die study tool returns one important resulting file; its name is formatted by the argument `clustering_file_name`.
When using the data mining tool Orange [13] and opening the `.ows` file provided in the die study tool within Orange, the "CSV File Import" node can be selected to import this clustering file.
After a short computation time the file is read and can be used: By parallelly opening the "Hierarchical clustering" and the "Image viewer" nodes, the user can select the different clusters within the "Hierarchical clustering" and view the clusters at the same time in the "Image viewer". Selecting such a cluster within Orange might be a little difficult at the present state (see Figure 7.1).

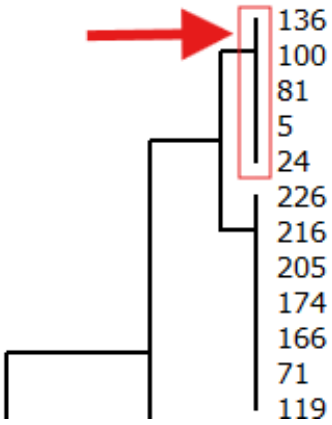


Figure 7.1: One cluster, as it can be seen and selected in Orange.

[34] `https://github.com/Urjarm/DieStudyTool`

Chapter 8

Results

The objective of this thesis was to improve upon existing digital die studies and specifically to improve on the work presented by Deligio et al. [12]. To achieve this, this thesis investigated different alternatives for all steps of the die study pipeline: preprocessing, image matching and clustering. However, the main focus was put on the second step.

The preprocessing chapter covered new methods and attempts to improve both the quality of the results and the computation time. In the way they were implemented in this thesis, the Laplace operator proved to be counterproductive and the image segmentation to be inconsistent. In contrast, changes to the order of the preprocessing pipeline and the replacement of the implementation of the denoise function turned out to be great achievements: The computation time could be significantly reduced and even a slight improvement in the result quality was achieved.

This thesis examined various options to compute similarity distances. Two capable candidate keypoint detectors and descriptors were presented, namely DISK and SIFTFeature of the kornia library. They performed particularly well when paired with the mutual nearest neighbour distance ratio matching function of kornia. With regard to the result quality, both keypoint detector and descriptors were similar improvements on the ORB keypoint detection and description model. However, DISK required less than a quarter of the time of SIFTFeature to compute the matching. The mutual nearest neighbour distance ratio matching function outperformed all other matching functions. It improved upon the brute-force matching of OpenCV when applied to keypoints from DISK or SIFTFeature. Regarding the above mentioned methods, the sum of distances of the matching returned the best results. Though depending on the keypoint detection and description method and the matching function, different interpretations of the matchings proved to be best.

Deligio et al. applied Spearman's rank correlation coefficient to the results to further improve them. This thesis demonstrated that the Pearson correlation coefficient was able to excel over Spearman's rank correlation coefficient.
With regard to agglomerative hierarchical clustering, alternative linkage methods were tested. Compared to the complete linkage, average linkage improved recall at the cost of specifity. Considering HDBSCAN as a clustering option returned good results, in particular since this option did not require any knowledge about the data. Additionally, the inclusion of an estimator in the image matching process and preserving the aspect ratios of the images during preprocessing was additionally tested. However, no improvements were recognisable.

8.1 Overall improvement

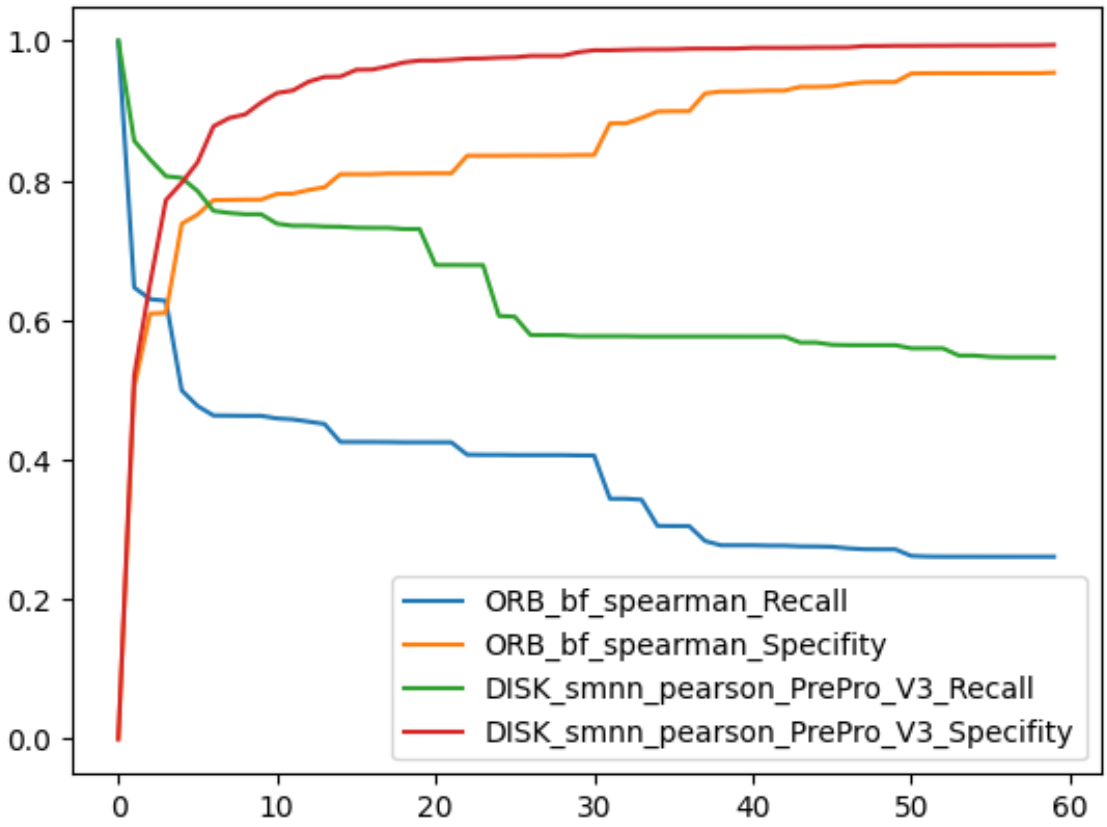


Figure 8.1: A figure showing the performance of the ORB matching against the matching with DISK keypoint detection and description, mutual nearest neighbour distance ratio matching and the Pearson correlation coefficient after using the new preprocessing pipeline with hierarchical clustering using complete linkage.

Figure 8.1 lines out the improvement of the new die study pipeline realised in this thesis over the die study pipeline implemented in Deligio et al. [12]. This new pipeline used *DISK* [58] for keypoint detection and description and computes the sum of all matches returned by the *mutual nearest neighbour distance ratio matching* function. On the resulting similarity matrix, the Pearson correlation coefficient was applied, turning it into a distance matrix. On this distance matrix, the final die study was computed with *hierarchical clustering with complete linkage*.

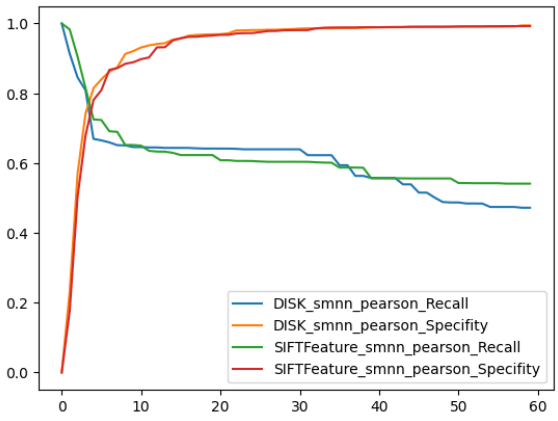


Figure 8.2: A figure comparing DISK and SIFTFeature.

SIFTFeature would be an alternative for DISK as both functions demonstrated similarly good results in Figure 2.1.2. But as shown in Figure 8.2: when using the Pearson correlation coefficient, SIFTFeature did return results similar to DISK, but took nearly four times as much computing time.

While *hierarchical clustering with average linkage* did not improve on the pipeline by Deligio et al. in terms of specifity, it offered a much higher recall than using complete linkage and also a higher point of intersection between specifity and recall. Depending on the application, this variant could yield better results.

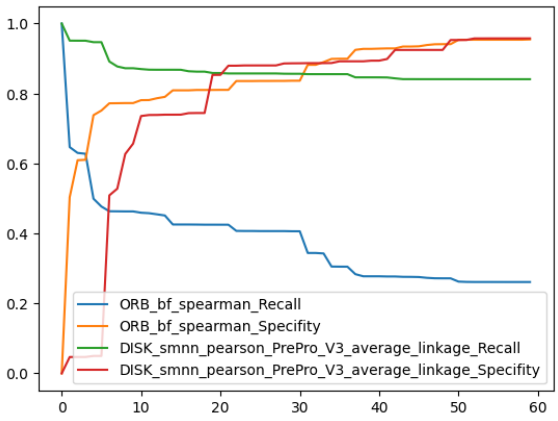


Figure 8.3: Like Figure 8.1, but with hierarchical clustering using average linkage.

In case the number of clusters, i.e. the number of dies used in the dataset, is unknown, HDBSCAN might be the preferred clustering algorithm.

8.1.1 Comparison to Deligio et al.

The paper by Deligio et al. [12] used a statistic to evaluate the results that was different than the one used throughout this thesis. For evaluation purposes, they used a clustering threshold of 0.3 which produced 256 clusters with the ORB method and counted the number of clusters which had at least 70% and 100% coins respectively

from the same die.
In order to compare results with the metric used by Deligio et al., the number of clusters was set to 256 manually for the method presented in this thesis, as the value threshold did not translate well into the present scenario.
The results from ORB and the three possible pipelines presented in the previous section are presented in the following table:

Method	**70%**	**#70%**	**100%**	**#100%**
ORB [12]	208		194	489
ORB[35]	189	667	173	373
New pipeline with complete linkage	226	1175	219	1078
New pipeline with average linkage	243	1191	237	1003
New pipeline with HDBSCAN[36]	18	906	13	664

Table 8.1: Numbers of clusters that contained at least 70% and 100% of coins respectively with the same die. The subsequent columns "#70%" and "#100%" express the number of images in these clusters (from the maximum number of 1325).

The results from the first row were taken directly from Deligio et al.'s paper [12], while the results from all other rows were computed with the same algorithm.
Table 8.1 shows that based on this metric the new pipeline strongly improved upon the pipeline by Deligio et al. It especially showed much higher numbers of coin images within the correctly clustered clusters.

8.2 Outlook

The digital die study by pipeline Deligio et al. could be modified in such a way, as presented in this thesis, that the results were drastically improved. But while many different approaches were already covered, there is definitely more potential within this topic.
Regarding image segmentation in the preprocessing step, IS-net developed by Qin et al. [39] or other image segmentation techniques might be able to improve upon the image segmentation implementations tested in this thesis. Future work on (ancient) coin segmentation specifically might achieve this, as indicated by Zaharieva et al.

[35]These results for Deligio et al.'s pipeline were computed with the same method as for the "new pipeline".
[36]This pipeline only produced 22 clusters.

[44] or in scikit-image's web-tutorial[37].
This thesis put its emphasis on testing a lot of different algorithms and techniques rather than elaborating and optimising these implementations. This is even more valid for techniques that produced weak results at the outset, for these were immediately subordinated rather than attempted to be improved. This might also have led to some functions performing weaker than expected. Especially the AGLP clustering was very promising, but its initial implementation did not meet these expectations and led to the assumption that its implementation was not yet compatible with the present algorithms.
The images throughout this thesis were downscaled to a size of 224×224px. A few rough tests did not present an immediate improvement when using images scaled down to a resolution larger than 224×224px. But using higher resolution images can be supposed to allow the image matching to detect more filigree contours and thereby improve the results.

[37] `https://scikit-image.org/docs/dev/user_guide/tutorial_segmentation.html`

References

[1] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. "Learning local feature descriptors with triplets and shallow convolutional neural networks." In: *Bmvc*. Vol. 1. 2. 2016, p. 3.

[2] Daniel Barath and Jiří Matas. "Graph-cut RANSAC". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6733–6741.

[3] Daniel Barath, Jana Noskova, Maksym Ivashechkin, and Jiri Matas. "MAGSAC++, a fast, reliable and accurate robust estimator". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1304–1312.

[4] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. "Key. net: Keypoint detection by handcrafted and learned cnn filters". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 5836–5844.

[5] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[6] Andrei Bursuc, Giorgos Tolias, and Hervé Jégou. "Kernel local descriptors with implicit rotation matching". In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. 2015, pp. 595–598.

[7] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. "Brief: Binary robust independent elementary features". In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer. 2010, pp. 778–792.

[8] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. "Density-based clustering based on hierarchical density estimates". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2013, pp. 160–172.

[9] Luca Cavalli, Viktor Larsson, Martin Ralf Oswald, Torsten Sattler, and Marc Pollefeys. "Adalam: Revisiting handcrafted outlier detection". In: *arXiv preprint arXiv:2006.04250* (2020).

[10] Antonin Chambolle. "An algorithm for total variation minimization and applications". In: *Journal of Mathematical imaging and vision* 20 (2004), pp. 89–97.

[11] Clément Cornet, Héloïse Aumaître, Romaric Besançon, Julien Olivier, Thomas Faucher, and Hervé Le Borgne. "Automatic Die Studies for Ancient Numismatics". In: *arXiv preprint arXiv:2407.20876* (2024).

[12] Chrisowalandis Deligio, Karsten Tolle, and David Wigg-Wolf. "Supporting the analysis of a large coin hoard with AI-based methods". In: (2023). URL: `https://doi.org/10.5281/zenodo.8301561`.

[13] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. "Orange: Data Mining Toolbox in Python". In: *Journal of Machine Learning Research* 14 (2013), pp. 2349–2353. URL: `http://jmlr.org/papers/v14/demsar13a.html`.

[14] Johan Edstedt, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. "DeDoDe: Detect, Don't Describe—Describe, Don't Detect for Local Feature Matching". In: *2024 International Conference on 3D Vision (3DV)*. IEEE. 2024, pp. 148–157.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[16] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[17] Chris Harris, Mike Stephens, et al. "A combined corner and edge detector". In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[18] Andreas Heinecke, Emanuel Mayer, Abhinav Natarajan, and Yoonju Jung. "Unsupervised statistical learning for die analysis in ancient numismatics". In: *arXiv preprint arXiv:2112.00290* (2021).

[19] Maurice G Kendall. "A new measure of rank correlation". In: *Biometrika* 30.1-2 (1938), pp. 81–93.

[20] Maurice G Kendall. "The treatment of ties in ranking problems". In: *Biometrika* 33.3 (1945), pp. 239–251.

[21] Godfrey N Lance and William T Williams. "Computer programs for hierarchical polythetic classification ("similarity analyses")". In: *The Computer Journal* 9.1 (1966), pp. 60–64.

[22] Joseph Lev. "The point biserial coefficient of correlation". In: *The Annals of Mathematical Statistics* 20.1 (1949), pp. 125–126.

[23] P Lindenberger, PE Sarlin, and M Pollefeys. "Lightglue: Local feature matching at light speed. arxiv 2023". In: *arXiv preprint arXiv:2306.13643* (2023).

[24] David G Low. "Distinctive image features from scale-invariant keypoints". In: *Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[25] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[26] "MEANet: An effective and lightweight solution for salient object detection in optical remote sensing images". In: *Expert Systems with Applications* 238 (2024), p. 121778. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.121778. URL: https://www.sciencedirect.com/science/article/pii/S0957417423022807.

[27] Anastasiia Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. "Working hard to know your neighbor's margins: Local descriptor learning loss". In: *Advances in neural information processing systems* 30 (2017).

[28] Dmytro Mishkin, Jiri Matas, and Michal Perdoch. "MODS: Fast and robust method for two-view matching". In: *Computer vision and image understanding* 141 (2015), pp. 81–93.

[29] Dmytro Mishkin, Filip Radenovic, and Jiri Matas. "Repeatability is not enough: Learning affine regions via discriminability". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 284–300.

[30] Arun Mukundan, Giorgos Tolias, Andrei Bursuc, Hervé Jégou, and Ondřej Chum. "Understanding and improving kernel local descriptors". In: *International Journal of Computer Vision* 127.11 (2019), pp. 1723–1737.

[31] Arun Mukundan, Giorgos Tolias, and Ondrej Chum. "Multiple-kernel local-patch descriptor". In: *arXiv preprint arXiv:1707.07825* (2017).

[32] Michael Nölle, Harald Penz, Michael Rubik, Konrad Mayer, Igor Holländer, and Reinhard Granec. "Dagobert-a new coin recognition and sorting system". In: *Proceedings of the 7th Internation Conference on Digital Image Computing-Techniques and Applications (DICTA'03), Syndney, Australia*. 2003.

[33] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. "Dinov2: Learning robust visual features without supervision". In: *arXiv preprint arXiv:2304.07193* (2023).

[34] Rémi Pautrat, Juan-Ting Lin, Viktor Larsson, Martin R Oswald, and Marc Pollefeys. "SOLD2: Self-supervised occlusion-aware line description and detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11368–11378.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] Fernando Pérez and Brian E. Granger. "IPython: a System for Interactive Scientific Computing". In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[37] Guilherme Potje, Felipe Cadar, André Araujo, Renato Martins, and Erickson R Nascimento. "XFeat: Accelerated Features for Lightweight Image Matching". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 2682–2691.

[38] Milan Pultar. "Improving the hardnet descriptor". In: *arXiv preprint arXiv:2007.09699* (2020).

[39] Xuebin Qin, Hang Dai, Xiaobin Hu, Deng-Ping Fan, Ling Shao, and Luc Gool. "Highly Accurate Dichotomous Image Segmentation". In: Nov. 2022, pp. 38–56. ISBN: 978-3-031-19796-3. DOI: 10.1007/978-3-031-19797-0_3.

[40] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin Jagersand. "U2-Net: Going deeper with nested U-structure for salient object detection". In: *Pattern recognition* 106 (2020), p. 107404.

[41] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks". In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 76.3 (2007), p. 036106.

[42] Marco Reisert, Olaf Ronneberger, and Hans Burkhardt. “A fast and reliable coin recognition system”. In: *Joint Pattern Recognition Symposium*. Springer. 2007, pp. 415–424.

[43] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. “Kornia: an open source differentiable computer vision library for pytorch”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 3674–3683.

[44] Maia Rohm, Martin Kampel, and Sebastian Zambanini. “Image Based Recognition of Ancient Coins”. In: vol. 4673. Aug. 2007, pp. 547–554. ISBN: 978-3-540-74271-5. DOI: 10.1007/978-3-540-74272-2_68.

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[46] Edward Rosten and Tom Drummond. “Machine learning for high-speed corner detection”. In: *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer. 2006, pp. 430–443.

[47] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[48] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International conference on computer vision*. Ieee. 2011, pp. 2564–2571.

[49] Leonid I Rudin, Stanley Osher, and Emad Fatemi. “Nonlinear total variation based noise removal algorithms”. In: *Physica D: nonlinear phenomena* 60.1-4 (1992), pp. 259–268.

[50] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. “Superglue: Learning feature matching with graph neural networks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 4938–4947.

[51] Jianbo Shi and Jitendra Malik. “Normalized cuts and image segmentation”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[52] Robert H Somers. “A new asymmetric measure of association for ordinal variables”. In: *American sociological review* (1962), pp. 799–811.

[53] Charles Spearman. “The proof and measurement of association between two things.” In: (1961).

[54] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. “LoFTR: Detector-free local feature matching with transformers”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 8922–8931.

[55] Zachary McCord Taylor. “The Computer-Aided Die Study (CADS): A tool for conducting numismatic die studies with computer vision and hierarchical clustering”. In: (2020).

[56] Yurun Tian, Axel Barroso Laguna, Tony Ng, Vassileios Balntas, and Krystian Mikolajczyk. “HyNet: Learning local descriptor with hybrid similarity measure and triplet loss”. In: *Advances in neural information processing systems* 33 (2020), pp. 7401–7412.

[57] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. “Sosnet: Second order similarity regularization for local descriptor learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 11016–11025.

[58] Michał Tyszkiewicz, Pascal Fua, and Eduard Trulls. “DISK: Learning local features with policy gradient”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 14254–14265.

[59] Laurens J Van Der Maaten and P Poon. “Coin-o-matic: A fast system for reliable coin classification”. In: *Proc. of the Muscle CIS Coin Competition Workshop, Berlin, Germany*. 2006, pp. 7–18.

[60] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[61] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. "scikit-image: image processing in Python". In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[62] Sebastian Zambanini and Martin Kampel. "Robust automatic segmentation of ancient coins". In: *International Conference on Computer Vision Theory and Applications*. Vol. 1. SCITEPRESS. 2009, pp. 273–276.

List of Figures

List of Tables