Faculty of Engineering, LTH

Centre for Mathematical Sciences

Master's Thesis

1:N fingerprint classification

Linda Xiang
elt13lxi@student.lu.se

Ellen Petersson
nat12ep1@student.lu.se

Supervisor

Magnus Oskarsson
magnuso@maths.lth.se

Examiner

Niels Chr. Overgaard
nco@maths.lth.se

July, 2018

Abstract

Biometric recognition systems are widely used to recognize an individual. Fingerprints is a biometric identifier and are today widely used in smartphones for biometric recognition. The fingerprint software used in smartphones are great and fast, and usually implemented for one person usage. A fingerprint software used for smartphones often conducts a one-to-one comparison between the sample and the enrolled templates. Software systems that can be used to recognize a person out of many are desirable. Such a system would conduct one-to-many comparisons. However, using the existing software in smartphones in a product used by many people would be too slow since it has to conduct many one-to-one comparisons between the sample and the enrolled templates.

This thesis examines whether it is possible to classify the enrolled templates by K-centroid, such that the number of comparisons in the one-to-many authentication problem is reduced. It was shown that the bag-of-words representation of an image is the best feature to use, together with the cosine similarity, during classification. To further improve the clustering, SVM and balancing were also applied. This combination showed good results and is the most promising method out of all methods examined in this thesis.

Keywords. K-centroid, similarity measure, SVM, templates, Bag-of-words.

Acknowledgements

We would like to thank Precise Biometrics for giving us the opportunity to make this possible. Special thanks to Björn Völcker for excellent guidance and encouragement along this thesis. We further thank the R&D team at Precise Biometrics for insightful comments and helping us with technical struggles. We would also like to thank Magnus Oskarsson at LTH for helping us and giving us feedback and academic guidance during the work on this thesis.

Contents

1 Introduction **3**
- 1.1 Fingerprint systems . . . 4
- 1.2 Fingerprint characteristics . . . 7
- 1.3 Templates . . . 8
- 1.4 Machine learning and computer vision . . . 10
- 1.5 Problem formulation . . . 11

2 Theory **12**
- 2.1 Features . . . 12
  - 2.1.1 Orientation image . . . 13
  - 2.1.2 Frequency image . . . 14
  - 2.1.3 Bag-of-Words . . . 14
  - 2.1.4 Scale Invariant Feature Transform (SIFT) . . . 17
  - 2.1.5 Good Features To Track (GFTT) . . . 21
- 2.2 Clustering . . . 23
  - 2.2.1 Partitioning methods . . . 23
  - 2.2.2 Hierarchical clustering . . . 25
- 2.3 Similarity measures . . . 27
  - 2.3.1 Earth mover's distance . . . 27
  - 2.3.2 Euclidean Distance . . . 28
  - 2.3.3 Cosine similarity . . . 28
- 2.4 Support Vector Machine (SVM) . . . 29
  - 2.4.1 Linearly separable data . . . 29
- 2.5 Prediction of a verification template . . . 33
  - 2.5.1 Centroid based prediction . . . 33
  - 2.5.2 Prediction using SVM . . . 34

3 Dataset and Tools **35**
- 3.1 Data . . . 35
- 3.2 Tools . . . 37

3.2.1 Python . . . 37
3.2.2 OpenCV . . . 37
3.2.3 Scikit-learn . . . 37
3.2.4 SciPy . . . 38

4 Experimental setup **39**
4.1 Introduction . . . 39
4.2 Features . . . 40
4.2.1 Frequency and Orientation histograms . . . 40
4.2.2 Bag-of-Words . . . 40
4.3 K-centroid . . . 42
4.3.1 Hierarchical clustering . . . 42
4.4 Support vector machine . . . 43
4.5 Performance measure . . . 44

5 Result **47**
5.1 Visualization of the data . . . 47
5.2 Results from frequency and orientation features . . . 48
5.2.1 Direct clustering . . . 49
5.2.2 Divisive clustering . . . 50
5.2.3 Without clustering . . . 52
5.3 Results from bag-of-words features . . . 52
5.3.1 Detector and similarity measure evaluation . . . 53
5.3.2 Divisive clustering . . . 53
5.3.3 Without clustering . . . 56
5.3.4 Example of BoW histograms . . . 56
5.3.5 Finger scores . . . 61

6 Discussion and future work **63**
6.1 Discussion . . . 63
6.1.1 Experimental setup . . . 63
6.1.2 Results . . . 64
6.2 Future work . . . 69
6.2.1 Keypoint detectors and descriptors . . . 69
6.2.2 Different method for bag-of-words vocabulary . . . 69
6.2.3 Bigger and smaller images . . . 70
6.2.4 Similarity measures . . . 70
6.3 Conclusion . . . 71

Chapter 1

Introduction

Biometric recognition refers to the use of biometric identifiers for automatically recognizing an individual. Biometric identifiers are anatomical and behavioral characteristics, e.g, fingerprints, face, iris, and voice. Biometric identifiers are considered more reliable for person recognition than traditional systems such as keys, ID-cards, password, and PIN. Biometric identifiers cannot be shared or misplaced while passwords or PIN can be guessed or shared and ID-cards can be lost or stolen. No two individuals have the same fingerprint, which is one reason why fingerprints are the most widely deployed biometric characteristics. Fingerprint-based recognition methods are being increasingly deployed in civilian and government applications [1, p. 1 - 3].

Precise Biometrics AB develops fingerprint software for convenient and secure authentication of identity in mobile devices, smart cards, and other products with fingerprint sensors. The software is well suited for products with small sensors and limited processing power and memory. The software for authentication of identity in the mentioned products is developed for one person usage, i.e., it conducts an one-to-one comparison. A software which can recognize and authenticate a person in a product designed for authentication of many people, i.e., conducting one-to-many comparisons, is desirable. However, using the existing software in such a product is too slow. This project evaluates the possibilities of reducing the number of comparisons in the one-to-many authentication problem.

This report is divided into six chapters. The first chapter contains an overview of concepts relevant for fingerprint systems and fingerprint recognition. The second chapter contains a more in depth description of the theory of the algorithms used in this project. The datasets and tools used in this project are introduced in the third chapter. Proposed setups of how the theory, datasets, and tools can be connected

as a solution to the problem are described in chapter 4. How to evaluate the setups is also described in the fourth chapter. Results obtained from the proposed setups are presented in the fifth chapter. The results and the proposed setups are discussed and a conclusion is presented in chapter 6.

1.1 Fingerprint systems

When designing a system for recognition using biometric identifiers one must determine how the person is going to be recognized. There are two types of biometric systems called *verification system* and *identification system* [1, p. 3]. A verification system does an one-to-one comparison to verify if the claim of identity by a person is true. For a submitted claim of identity by a person, the verification system either rejects or accepts it. An identification system, on the other hand, does an one-to-many comparison to search if the person is present in the database. If the person is present then the system will return the identifier of the enrollment reference that matched. In comparison with the verification system, the identification system answers the question on "Who are you?" and in verification systems we answer the question "Are you...You?".

The verification system and the identification system can be divided into three main processes, *enrollment*, *verification*, and *identification*. A verification system uses the enrollment and verification processes while an identification system uses the enrollment and the identification processes [1, p. 5].

The enrollment process registers the fingerprint information from individuals in the system storage. The identification process outputs a list containing templates of possible candidates for a match. The verification process is responsible for confirming the claim of identity by the subject. As seen in Figure 1.1 the enrollment, the identification and the verification processes can be broken down into the following modules:

- *Capture*: a raw digital fingerprint image is captured by a fingerprint sensor. The digital representation is often called a *sample*.
- *Feature extraction*: the sample is further processed by a feature extractor. This generates a compact but expressive representation of the sample, known as a *feature set*.
- *Template creation*: this module organizes one or more feature sets into an *enrollment template*. The enrollment template will be saved in the system storage.

- *Pre-selection*: takes a *verification template* (obtained from a new sample) and the enrolled templates of all subjects in the system storage as input. The module reduces the size of the enrolled template database in order to minimize the number of enrollment templates that have to be compared with the verification template. The module is mainly used in an identification system when the number of enrolled templates is large.
- *Matching*: the matching module, also known as the *matcher* takes a verification template as input. The verification template is compared against the enrollment templates of that subject and the similarity between these is computed, known as a *similarity score*. The similarity score is then compared with a *system threshold* to make the final decision. The person is recognized if the similarity score is higher than the system threshold. In this case, the matcher produces a match decision. If the similarity score is lower than the system threshold, the person is not recognized and the matcher produces a non-match decision.
- *Data storage*: is responsible for storing the templates.

The focus in this project is on the identification process, in particular on the feature extraction, template creation and the pre-selection modules. The pre-selected templates will then be sent to Precise Biometrics' verification system for matching.

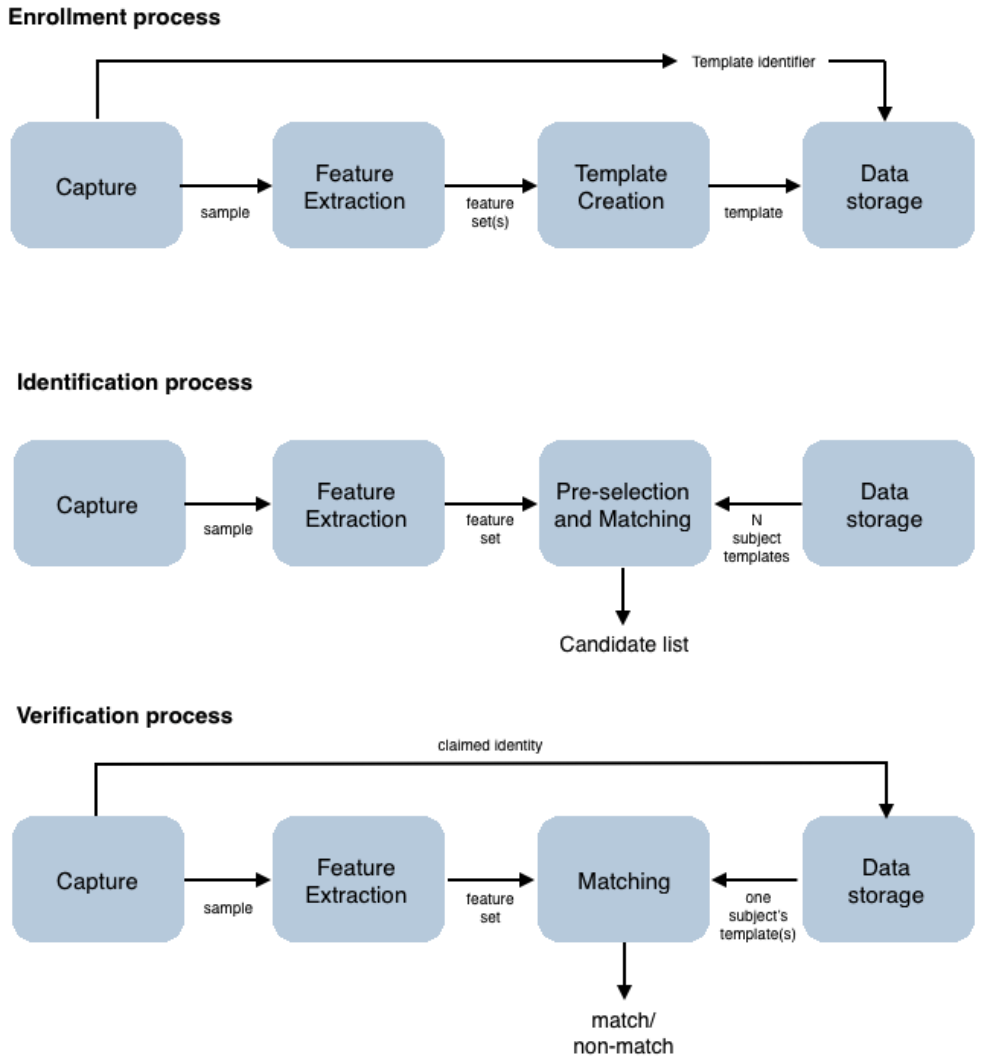


Figure 1.1: Enrollment, identification and verification processes. The enrollment process is responsible for creating enrollment templates from the captured samples and storing them. The identification process recognizes a subject by comparing a verification template created from a new sample with all enrolled templates in the data storage, the pre-selection and matching module are often combined. The verification process authenticates a subject's identity by comparing a verification template with the subject's enrollment templates.

1.2 Fingerprint characteristics

A sample of a fingerprint is captured by a fingerprint scanner during the enrollment process. The extraction of biometric characteristics is an essential component when identifying a person.

In an identification system, it is needed to determinate features of a fingerprint sample that can discriminate between different fingers as well as remain invariant for a given finger. A feature space in which fingerprint images belonging to different fingers have high inter-class variations and fingerprint images belonging to the same finger have low intra-class variations needs to be determined [1, p. 39].

A fingerprint is the reproduction of the exterior appearance of the fingertip epidermis (outer skin) [1, p. 97]. *Ridges* and *valleys* form the most noticeable structural characteristic of a fingerprint. A fingerprint consists of a pattern of interleaved ridges and valleys. Ridges, also called ridge lines, are dark lines and valleys are bright lines, see Figure 1.2. The pattern of ridges and valleys has different characteristics for different fingerprints, the ridge structure of every fingerprint is permanent and unchanging [1, p. 32]. The ridge details are usually analyzed in a hierarchical order at three different levels which exhibit different types of features. Level 1 describes the overall global ridge flow pattern, Level 2 is the local level which describes minutiae, and Level 3 is the very-fine level which can detect intra-ridge details.

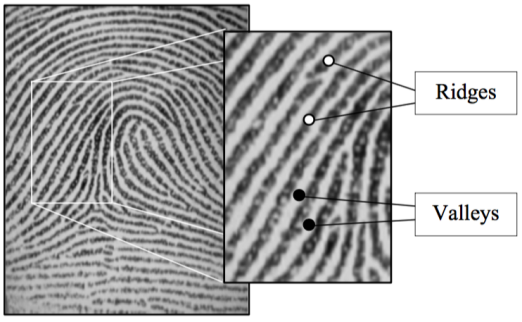


Figure 1.2: Ridges and valleys in a fingerprint image. Figure from [1, p. 97].

The global ridge flow pattern (Level 1) is mainly a pattern of ridges that run smoothly in parallel, but there are regions that assume distinctive shapes characterized by frequent ridge endings and high curvature [1, p. 98]. These regions are called *singular regions* and can be classified into three categories, *loops*, *deltas*, and *whorls*. An example of the singular regions can be seen in Figure 1.3. The fingerprint *orientation image* and *frequency image* are also features that can be detected at the global level.

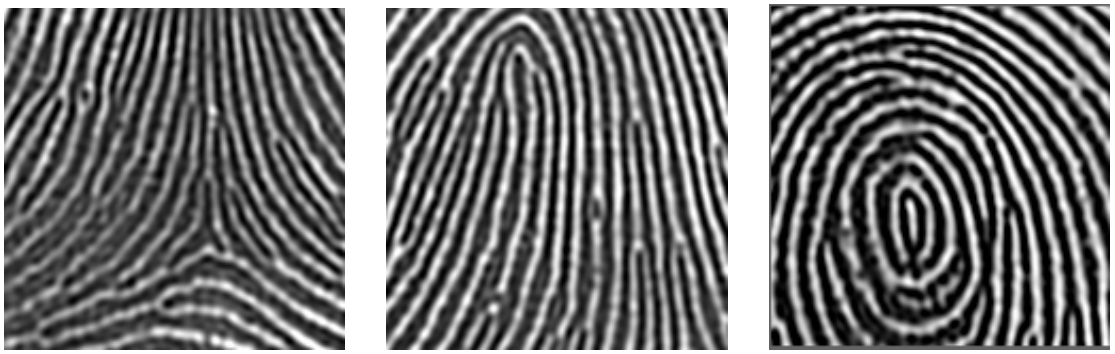


Figure 1.3: From left to right: delta ($\triangle$), loop ($\cap$) and whorl ($\bigcirc$).

Local ridge characteristics called *minutiae* can be found at the local level (Level 2). Minutia means small details, *ridge endings* and *bifurcations* are the two most prominent minutiae [1, p. 39]. A ridge ending is where a ridge suddenly comes to an end and a bifurcation is where a ridge divides into two ridges, see Figure 1.4 for an illustration.

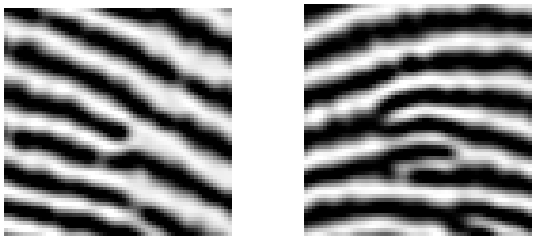


Figure 1.4: A ridge ending (left) and a bifurcation (right).

The intra-ridge details detected at the very-fine level (Level 3) include width, shape, curvature, and edge contours of ridges but also incipient ridges, pores, creases, breaks, and scars [1, p. 101]. Sweat pores located at the ridges are considered the most important detail at the very-fine level. However, high-resolution fingerprint images of high quality are necessary for extraction of details at this level.

1.3 Templates

In both identification and verification processes, one verification template created from a new sample has to be compared with an enrollment template in order to decide if the verification template is from the same finger as the enrollment template or not. This comparison is important for the decision match/non-match in the verification

process and for the decision if the enrollment template should be in the output candidate list generated by the identification process.

Sensor manufacturers tend to reduce the size of the fingerprint sensors in order to lower the cost of production. A smaller fingerprint sensor means a smaller sensing area which will capture a partial fingerprint. It is difficult to compare templates created from samples of partial fingerprints because of the possibility that the templates only have a little overlap, even though the templates are from the same finger. Figure 1.5 illustrates this problem. The error that occurs when the verification template and the enrollment template are from different parts of the same finger and have a little overlap is called *information limitation* [1, p. 12]. The pre-selection module and the matching module cannot make a correct decision in situations like this.

Figure 1.5: Overlap between a verification template (green) and an enrollment template (red) captured by a small sensor. Original fingerprint image from [1, p. 131].

To increase the possibility of overlap between the verification template and an enrollment template, some systems collect multiple samples of a user to produce a set of enrollment templates for each finger. A set of enrollment templates for one finger is known as a *multi-template*, an illustration of a multi-template can be seen in Figure 1.6. The verification template is compared with all templates in the multi-template, if the similarity score for one of these comparisons is high enough a correct decision can be made.

Important core features such as loops and whorls or deltas are in most cases not included in a sample captured by a small sensor. Users have difficulties placing the center of the fingerprint in the center of the sensing area if the sensor is too small [2]. In order to increase the possibility of large overlap between a verification template and an enrollment template, it is desirable that the enrollment templates in the multi-template are evenly distributed across the fingerprint. The verification template will in this case not overlap most of the enrollment templates in the multi-

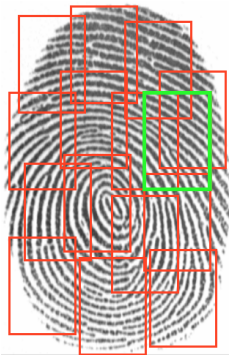


Figure 1.6: Overlap between a multi-template made from a set of enrollment templates (red) and a verification template (green). Original fingerprint image from [1, p. 131].

template, but the overlap is expected to be large in cases where an overlap exists. The enrollment templates will possibly have little overlap with each other.

1.4 Machine learning and computer vision

Machine learning is a type of artificial intelligence which allows computer systems to learn with data without being explicitly programmed[1]. The computer makes a prediction based on learned patterns from the training data. The goal is to allow the computer to learn by-itself without the interference or assistance by humans. The algorithms or tasks in machine learning can be divided into two subgroups, supervised and unsupervised learning. In supervised learning, the computer is presented with example inputs and with the desired outputs[1]. The goal for the computer is to map inputs with outputs, to find a pattern or a general rule for this map. In unsupervised learning, we don't have any desired outputs. In other words, the data is not labeled. With unlabeled data we allow the computer to find structures and hidden patterns in the data.

Computer vision can be seen as a way to automate tasks that the human visual system can do[2]. The tasks in computer vision include methods for acquiring, processing, and analyzing digital images such that the system can make some form of decision. Computer vision can be used for recognition tasks. A task to determine whether a specific object is present in an image. It can also be used for identification, for example, to identify a persons face, fingerprint, handwritten digits or

[1] https://en.wikipedia.org/wiki/Machine_learning
[2] https://en.wikipedia.org/wiki/Computer_vision

a specific vehicle. Other tasks include motion analysis, scene reconstruction, and image restoration[2].

1.5 Problem formulation

Biometric recognition systems using fingerprints as the biometric identifier can be seen in many smartphones. To unlock your phone, you need to have your fingerprint stored on the device. When a person is trying to unlock the phone, the captured fingerprint image needs to be matched with the stored fingerprint image, which is the definition of a verification system. An identification system, for example in a door lock, can be implemented by using a verification system that does many one-to-one comparisons. If the algorithm for unlocking your phone were applied to a door lock, the matching time would be very slow, because of the many one-to-one comparisons.

This thesis investigates the possibilities of reducing the number of comparisons in a one-to-many authentication problem. This can be done by reducing the number of possible matching templates, such that a search through the whole database can be avoided. If the database contains 1000 persons, and each person has enrolled 10 templates, the database will consist of $10,000$ templates. This means that $10,000$ comparisons need to be conducted to authenticate one person. The goal of this thesis is to find a method to reduce the number of comparisons by reducing the number of possible matching templates. If there's $10,000$ templates in the database in total, the goal is to reduce it to about $100-200$ possible templates during authentication.

Chapter 2

Theory

In this chapter, all relevant theory for this thesis is presented. This chapter is organized as follows; in section 2.1, different feature extracting methods are introduced. In section 2.2, the reader are introduced to different clustering methods. In section 2.3, different similarity measures for histograms are presented. Then in section 2.4, the Support Vector machine is introduced. Finally, in the last section, 2.5, methods for labeling new data are discussed.

2.1 Features

In this section, the different features found in fingerprints and methods for feature extraction are presented. First, the difference between feature detectors and descriptors will be explained. In the following sections orientation, frequency and bag-of-words will be introduced as features that are found in fingerprints. In sections further down SIFT and GFTT feature extractors are presented.

A detector detects interesting points or image features in an image. These features can, for example, be corners or blobs. A feature detector can be seen as a searching window. The searching window looks for regions in the image where it has maximum variation when the searching window is moved by a small amount[1].

A feature descriptor is a description of the feature that was found by the feature detector. A feature descriptor describes the region around the detected feature. This description is of great importance because it makes it possible to find the same

[1] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning

feature in another image, an image taken on the same object but maybe from a different angle[1].

2.1.1 Orientation image

The fingerprint orientation image is one of the features that can be found in the global level (Level 1). The orientation image represents the directionality of ridges [3]. The orientation image is a matrix whose elements encode the local ridge orientation [1, p. 103]. The local ridge orientation at a pixel $[x, y]$ is the angle $\theta_{x,y}$ that the fingerprint ridges, crossing through an arbitrary small neighborhood centered at $[x, y]$, form with the horizontal axis. Instead of computing the local ridge orientation at each pixel, it is estimated at discrete positions. Each element θ_{ij} denotes the average orientation of the ridges in a neighborhood of the pixel $[x_i, y_j]$. To denote the reliability of the orientation, a value, r_{ij}, is often associated with each element θ_{ij}. The length of each element is proportional to its reliability [1, p. 103]. An illustration of an orientation image is shown in Figure 2.1.

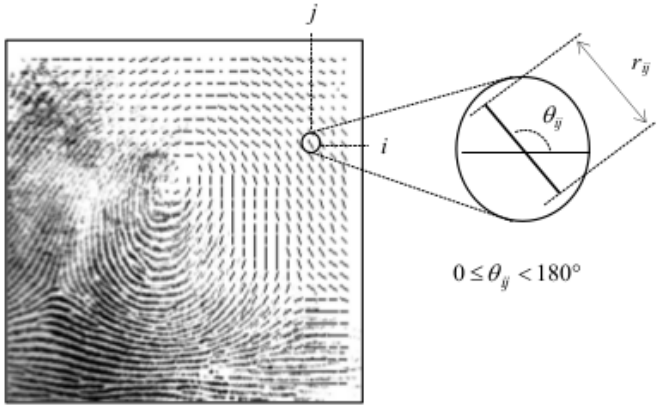


Figure 2.1: An orientation image faded into the corresponding fingerprint image. Each element denotes the local ridge orientation. Figure from [1, p. 103]

The most popular approach for extracting local ridge orientations is based on computation of gradients in the fingerprint image. The orientation image feature is represented with a 12 bin histogram. The orientation image histograms are not rotation invariant.

2.1.2 Frequency image

The frequency image for a fingerprint will be introduced shortly in this section. The method of how to find this image and how to generate the frequency feature will not be described. As the frequency histograms are provided by Precise Biometrics.

The most noticeable structural characteristics of a fingerprint is its ridges and valleys, as has been described in section 1.2. The ridges are dark lines and the valleys are bright lines. The ridges and valleys vary across different regions of the same finger. At some region, there are more ridges than other regions. The number of ridges across different fingers also varies. Generally, the frequency feature calculates the local ridge frequency at different locations in the fingerprint image. At a point $[x, y]$ in the image, the local ridge frequency f_{xy} is measured as the number of ridges along a hypothetical segment centered at $[x, y]$ and orthogonal to the local ridge orientation [1, p. 112]. The frequency image is created by estimating the frequency at discrete positions.

The fingerprint frequency image is also found in the global level (Level 1). The frequency image provided, is represented by a 12 bin histogram, as the orientation image.

2.1.3 Bag-of-Words

The bag-of-words (BoW) model was originally used in natural language processing and information retrieval [4, p. 149] for document representation. The representation of a document using this model is a bag of its words[2]. In this bag, there are words that occur many times in the document. The count of how many times a word is present in the document is the definition of the *frequency of occurrence*. These words and their frequency of occurrence can be conceptualized as a feature that can be used to train a classifier [4, p. 150]. BoW is a common model used in methods for document classification.

The same idea or model can be applied to computer vision to create a bag-of-words, or bag-of-visual-words (BoVW), for image classification [4, p. 150]. The difference is that objects in the image are counted instead of words. The objects are the visual words, and they are important points in the image. Images and documents can then be represented in the same way, by a histogram of (visual) words and their frequency of occurrence.

[2] `https://en.wikipedia.org/wiki/Bag-of-words_model`

To be able to create this model for images, a method for extracting interest points is needed. These methods are called feature extractors. A feature extractor detects interest points in an image called *keypoints*. For every keypoint, the region around it is described by the descriptor. There are different feature extractors like Scale-Invariant Feature Transform (SIFT), Good Features To Track (GFTT) and Speeded-Up Robust Features (SURF). SIFT and GFTT will be explained later.

A BoW trainer is also needed to be able to represent the image by its visual words. Before each image can be represented by its visual words, a vocabulary of all possible visual words in the dataset is needed. The images in this dataset will be represented by a subset of the words in the vocabulary. The vocabulary is created by using a BoW trainer. The trainer is commonly a k-means clustering method. The visual words are created by clustering the descriptors from the feature extractor into K clusters. The center for each cluster will describe a visual word. When the clustering has converged the vocabulary is made up of the centers from the clusters [4, p. 151]. A larger dataset will make the vocabulary richer in words.

Let's summarize the algorithm for the BoVW model [4, p. 151]:

1. For every image in a dataset, use a feature extractor to extract descriptors
2. Add the extracted descriptors to a BoW trainer
3. The descriptors are then clustered into k clusters. The center of these clusters are the visual words.

In Figure 2.2 there is a picture of the bag-of-words process

Term frequency-inverse document frequency (tf-idf)

Term frequency-inverse document frequency, tf-idf, is often used as a weighting term for text documents. The tf-idf calculation determines how relevant a given word is in a particular document. In a large (English) text document there will be many words like "a", "the", and "and". Classifying documents with occurrences of these words will not tell much of what kind of text it is. These common words in a text can be weighted by the tf-idf term. The term frequency (tf) in tf-idf is the number of times a term appears in a document [5]. The term frequency is denoted

$$\mathrm{tf}(t, d) = f(t, d),$$

where t is the term and d is the document.

The inverse document frequency (idf) varies inversely with the number of documents to which a term is assigned in a collection of documents [5]. Common words such

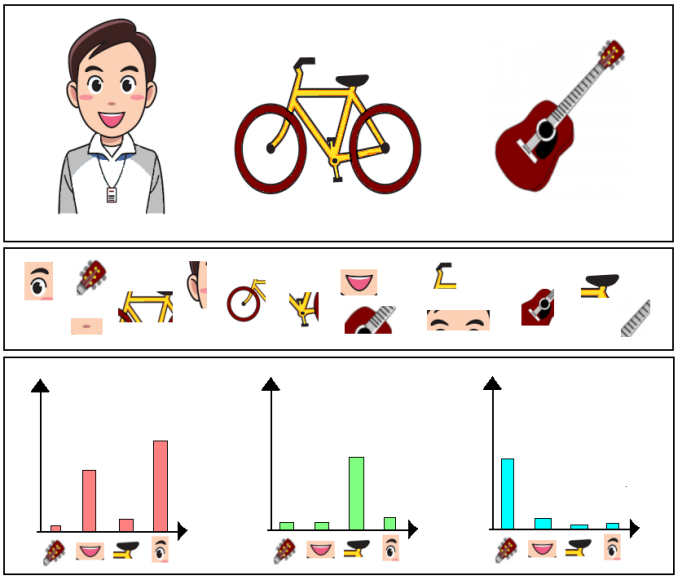


Figure 2.2: Bag-of-words process. At the top we see a set of images that represents our dataset. In the middle we see our vocabulary of words. At the bottom we see a histogram of words and its frequency for each image in the dataset.

as articles and prepositions tend to have smaller tf-idf numbers than words that are common in a single or a small group of documents. The inverse document frequency is calculated as

$$\text{idf}(t) = \log\left(\frac{N}{1+n_t}\right),$$

where N is the total number of documents and n_t is the number of documents containing the term.

Terms that are able to distinguish certain individual documents from the remainder of the documents are the best terms for document content identification. Such terms should have high term frequencies but low document frequencies [5]. A measure of term importance can be calculated as

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t).$$

The tf-idf weighting is usually used after the BoW-vocabulary has been created. Then for every histogram, the tf-idf term is applied to each bin.

In the same way as the tf-idf term is applied to the histograms representing a document, it can be applied to the histograms representing an image. The tf-idf weights down the most common features in an image instead of common words in a docu-

ment. A characteristic specific representation for an image is yielded by applying tf-idf.

2.1.4 Scale Invariant Feature Transform (SIFT)

David G. Lowe proposed the SIFT-algorithm in 2004 [6], which extracts features that are invariant to image scale and rotation.

Four major stages of computation are used to generate the set of image features. The stages are listed and described below.

Detection of scale-space extrema

Potential interest points that are scale- and rotation-invariant, which are called *keypoints*, are detected at the first stage of computation.

Using scale-space extrema in the difference-of-Gaussian (DoG) function convolved with the image, $D(x, y, \sigma)$, to detect stable keypoint locations in scale-space was proposed by Lowe [6]. It can be computed as the difference of two nearby scales separated by a constant factor k, mathematically expressed as

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y), \tag{2.1}$$

where $*$ is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{2.2}$$

The first reason for choosing this function is that $D(x, y, \sigma)$ is efficient to compute since it can be computed by simple image subtraction. The second reason is that it is a good approximation to the scale-normalized Laplacian of Gaussian. It has been shown that the maxima and minima of the scale-normalized Laplacian of Gaussian produce very stable image features.

One approach to the construction of D is shown in Figure 2.3. For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images separated by a constant k in scale space, illustrated with the left stack in Figure 2.3.

A purple stack in the left stacks is called an octave, each octave is divided into an integer number, s, of intervals, so $k = 2^{1/s}$. Each octave must contain $s + 3$ images

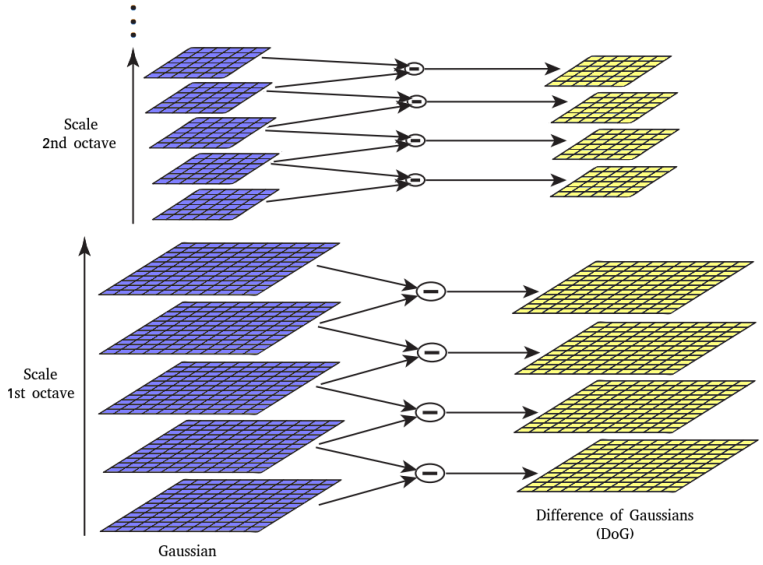


Figure 2.3: Gaussian pyramids (purple) with two octaves. For each octave, the image is repeatedly convolved with Gaussians to produce the set of scale space images (purple). To produce the difference-of-Gaussian images (yellow), adjacent Gaussian images are subtracted. The next octave is produced when the current octave is completed and the Gaussian image has been down-sampled.

so that the extrema detection covers a complete octave. The difference-of-Gaussian images shown on the right are produced by subtracting adjacent image scales. When an octave is completed, the second Gaussian image from the top is re-sampled by taking every second pixel in each row and column (half the image resolution) and next octave can be produced.

Local maxima and minima of $D(x, y, \sigma)$ are detected by comparing each sample point to its eight neighbors in the current image and nine neighbors in the scale above and below, as shown in Figure 2.4. The sample point is selected as a keypoint candidate if it is either larger than all of its neighbors or smaller than all of them.

Keypoint localization

Once potential keypoints have been found, the next step is to eliminate bad keypoints which are selected based on measures of their stability. For each candidate keypoint, this step performs a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows candidate keypoints, which are sensitive to noise or are poorly localized along an edge, to be rejected.

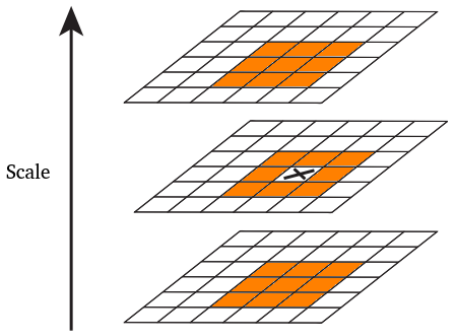


Figure 2.4: Detection of extrema of the difference-of-Gaussian images is done by comparing a pixel (marked with x) to its neighbors (orange) at the current and adjacent scales.

A method for fitting a 3D quadratic function to the keypoint to determine the interpolated location of the maximum is used to give the keypoint a more accurate location. This method uses Taylor series expansion of the scale-space function, $D(x, y, \sigma)$, shifted so that the origin is at the sample point. The intensity of the extrema at the interpolated location given by the Taylor expansion function is useful for rejecting unstable extrema with low contrast (points that are sensitive to noise).

The DoG function has a strong response along edges, even if the location along the edge is poorly determined. A poorly defined peak in the DoG function will have large principal curvature across the edge but small principal curvature along the edge. The principal curvatures can be computed from a 2×2 Hessian matrix, $\mathbf{H}$. The eigenvalues of $\mathbf{H}$ are proportional to the principal curvatures. A concept similar to Harris corner detection is used to avoid computing the eigenvalues since the ratio of the two eigenvalues is sufficient. Candidate keypoints that have a ratio of principal components greater than some threshold is eliminated.

Orientation assignment

The next step is to assign each keypoint with one or more orientations. The orientations are based on local image gradient directions. The keypoint will after this step achieve invariance to image rotation since the keypoint can be represented relative to the assigned orientations.

The Gaussian smoothed image, L, with the closest scale to the scale of the keypoint is selected. The gradient magnitude $m(x, y)$, and the orientation, $\Theta(x, y)$ are precomputed for each image sample, $L(x, y)$, at this scale using pixel differences

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (2.3)$$

$$\Theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))) \quad (2.4)$$

This information is extracted in a neighborhood around the keypoint in the Gaussian smoothed image. An orientation histogram with 36 bins is formed, where each bin covers 10 degrees. Each image sample is weighted by the magnitude. Dominant directions correspond to peaks in the histogram, the highest peak is detected and local peaks within 80% of the highest peak are used to create a keypoint with that orientation. There will be multiple keypoints but with different directions located at locations with multiple peaks of similar magnitude. Creating these extra keypoints contributes to the stability of matching.

Keypoint descriptor

A location, scale, and orientation have been assigned to each keypoint in the previous steps. This step will compute a descriptor for each keypoint. The computations in this step are performed on the image closest in scale to the keypoint's scale.

The descriptor extraction procedure proposed by Lowe [6] first samples the image magnitudes and orientations around the keypoint. Orientation invariance is achieved by rotating the coordinates of the descriptor and the gradient orientations relative to the keypoint orientation. For efficiency, the gradients are precomputed for all levels of Gaussian blur for the image as described in the section about orientation assignment. The left side of Figure 2.5 illustrates the precomputed gradients with small arrows at each sample location. The magnitude of each sample point is weighted with a Gaussian weighting function with σ equal to one half of the descriptor window. The Gaussian weighting function is illustrated with a circular window in Figure 2.5. The purpose of the weighting is to avoid sudden changes in the descriptor with small changes in the position of the Gaussian window and to give less emphasis to gradients far from the center. For each 4×4 sample region, an orientation histogram is created. Each orientation histogram has eight directions, illustrated with arrows on the right side of Figure 2.5. The orientation histogram will, therefore, be an 8 bin histogram where the magnitude of each bin corresponds to the lengths of the arrows. The histograms of all sample regions are then concatenated, the resulting vector is the descriptor. Figure 2.5 is an illustration of a 2×2 array of orientation histograms, the SIFT paper [6] uses a 4×4 array of orientation histograms (16 sample regions) with 8 bins in each, resulting in a $4 \times 4 \times 8 = 128$ length keypoint

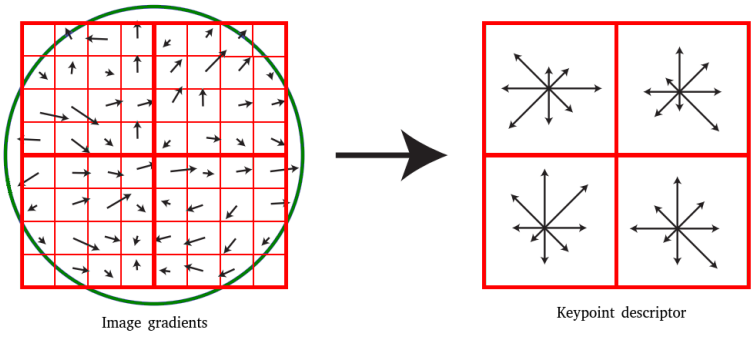


Figure 2.5: The descriptor is created by computing gradient magnitudes and orientations in a region around the keypoint location, these are weighted by a Gaussian window illustrated as a circle in the figure to the right. An eight bin orientation histogram is created for each 4×4 subregion, the length of each arrow (left figure) corresponds to a bin in that histogram. These histograms are concatenated, resulting in the keypoint descriptor.

descriptor. Finally, the descriptor is modified to make it invariant to affine changes in illumination, and to reduce the effects of non-linear illumination changes.

The theory of the SIFT algorithm is presented in detail in [6].

2.1.5 Good Features To Track (GFTT)

Good Features To Track (GFTT) also known as Shi-Tomasi Corner Detector was first introduced by Shi-Tomasi in 1994 [7]. It is an improved version of the Harris corner detector which was introduced in 1988 by Chris Harris and Mike Stephens [8]. The implementation of the scoring function is the main difference between the two methods[3]. This section is organized as follows; the Harris corner detector will be explained first, and then the difference between the two detectors is explained.

Harris corner detection

A corner in an image is defined as a region of large intensity variation in all directions. Harris corner detection looks for such changes in the intensity I of an image, by the

[3] `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi`

following expression[8]:

$$E(x,y) = \sum_u \sum_v w(u,v)(I(u+x, v+y) - I(u,v))^2, \tag{2.5}$$

for a displacement (x, y). Here E denotes the change of intensity between the original and the moved window, $w(u, v)$ is a window function at position (u, v), $I(u+x, v+y)$ is the intensity of the moved window, and $I(u, v)$ is the intensity of the original window. The window function can either be rectangular or gaussian.

If the function $E(x, y)$ is maximized, it will look for corners in the image. The function is maximized when the second term $(I(u+x, v+y)-I(u, v))^2$ is maximized. By applying Taylor series expansion and after some computation on the second term, the function $E(x, y)$ can be written as, (for full computations refer to [8]):

$$E(x,y) = \begin{bmatrix} x & y \end{bmatrix} M \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.6}$$

where M is

$$M = \sum_x \sum_y w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{2.7}$$

where I_x and I_y are the first order derivatives of I in x and y directions.

Let the matrix M have the eigenvalues λ_1 and λ_2. These eigenvalues represent the changes in intensity in the orthogonal direction of the image[9]. Instead of computing the eigenvalues of matrix M, we consider computing the cornerness measure or the response function R [8][9]. The response function is computed as

$$R = det(M) - \kappa \cdot tr(M)^2, \tag{2.8}$$

for a small constant $\kappa > 0$ [9]. The response function can also be written as

$$R = \lambda_1 \lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2, \tag{2.9}$$

where

$$det(M) = \lambda_1 \lambda_2, \tag{2.10}$$

$$tr(M) = \lambda_1 + \lambda_2. \tag{2.11}$$

If both λ_1 and λ_2 are small, then R is small, and the region will be *flat*. If one eigenvalue is much bigger than the other, then $R < 0$, and the region is an *edge*. Finally we have a *corner* if $R > 0$, and if both eigenvalues are large and $\lambda_1 \sim \lambda_2$.

Shi-Tomasi Corner Detector

As described above for the Harris corner detector, a region is a corner if the response function is positive, $R > 0$. Later in 1994, J.Shi and C.Tomasi [7] made a small change to the Harris response function, which resulted in better results compared to the original. The Harris response function is computed as

$$R = det(M) - \kappa \cdot tr(M)^2.$$

In Shi-Tomasi corner detector, the response function is computed as

$$R = min(\lambda_1, \lambda_2). \tag{2.12}$$

If the response function is bigger than a given threshold value λ, then the region is a corner[4].

2.2 Clustering

Clustering is an unsupervised learning problem. The task is to find a structure in a collection of unlabeled data and grouping the data according to the found structure. The data in a group, called a *cluster* [10, p. 1], is a collection of objects that are more similar to each other than to objects in other clusters. The similarity measure between two objects can, for instance, be distance. The objects will belong to the same cluster if they are close according to the chosen distance metric.

There are various algorithms for solving the clustering problem. An algorithm for hard clustering and an algorithm for hierarchical clustering are described in the following sections.

2.2.1 Partitioning methods

Clustering can be divided into two subgroups, which are hard clustering and soft clustering. In hard clustering, each data point can only be assigned to one of the K clusters [11, p. 428]. Whereas in soft clustering, every data point is assigned with a possibility or likelihood of belonging to the different clusters. Soft clustering is desirable for clustering problems where many data points are between clusters. In this thesis, only hard clustering will be considered. A figure illustrating the difference between hard and soft clustering can be seen in Figure 2.6.

[4] `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html`

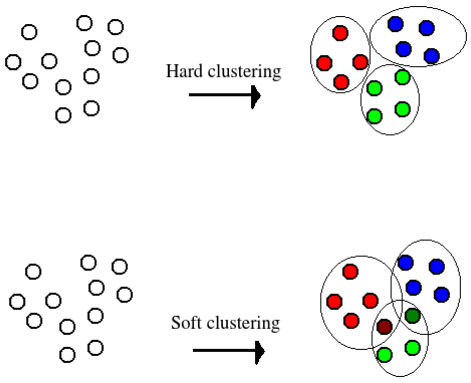


Figure 2.6: The difference between hard and soft clustering.

K-centroid

K-centroid is similar to K-means, the only difference is how the center is updated. K-centroid is a machine learning method under the category of unsupervised learning. It can be used to cluster unlabeled data into K different clusters. The algorithm iteratively assigns each data point to one cluster, by looking at the similarity of features between each cluster center and the data point. When the data points have been assigned to one cluster, each cluster center is updated to find a new center within the cluster. After the center update, the data points are again assigned to a cluster. If the centers don't update in the next iteration, the algorithm has converged. Alternatively, the algorithm has converged if it has exceeded a given maximum number of iteration.

The K-centroid algorithm can be summarized in a few steps:

1. Choose the number of clusters K.
2. Randomly initialize the centroids μ_k for each cluster.
3. For each data point, assign it to one cluster according to a similarity measure. There are different similarity measurements that can be used. In section 2.3, different definitions will be described.
4. Update the cluster center. The center is updated by assigning one of the data points in the cluster as the center. This is done by iteratively calculating the similarity from one data point to all other data points in the cluster. The data point with the smallest total similarity is assigned as the center.

Repeat step 3 and 4 until convergence. In this thesis, convergence is reached when

the center doesn't update in the next iteration step. When convergence has occurred we have reached a local minimum of the objective function J

$$J = \sum_{i=1}^{n} \sum_{k=1}^{K} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2, \tag{2.13}$$

where $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ is the data set and $\boldsymbol{\mu}$ is the set of cluster centers.

2.2.2 Hierarchical clustering

In hierarchical clustering, a hierarchy of clusters is built, the clusters merge together or split at different stages in the hierarchy. A hierarchy means that objects belonging to a child cluster also belong to the parent cluster.

Hierarchical clustering can be divided into two types, agglomerative and divisive clustering. Agglomerative clustering is a bottom-up approach, starting with each object as an individual cluster which is merged when moving up in the hierarchy [10, p. 44]. Divisive clustering is a top-down approach, all objects start in one cluster which is split recursively when moving down in the hierarchy. Only divisive clustering will be considered in this thesis and it is presented in the following section.

Divisive clustering

At each step in the hierarchy, the divisive method splits up a cluster into two smaller ones until all clusters contain only a single element [10, p. 253]. The decision of which cluster to split at next step is based on the size of the clusters, the bigger clusters are divided first. The hierarchy is built in $n-1$ steps when the data contains n objects. The method can, however, be stopped before reaching the last step, if larger clusters are desired. An illustration of divisive clustering is shown in Figure 2.7.

A divisive clustering method starts from one large cluster containing all objects, the first step of the method is to split the first cluster into two new clusters. In order to get the best split of the large cluster, all possible divisions of the data may be considered. Considering all possible divisions amounts to $2^{n-1} - 1$ possibilities [10, p. 254]. The number of possibilities grows exponentially fast and divisive clustering could, therefore, be very computationally heavy. It is, however, possible to construct divisive methods that do not consider all possible divisions. The split in each step can, for instance, be done using the K-centroid algorithm explained in section 2.2.1, where $K = 2$.

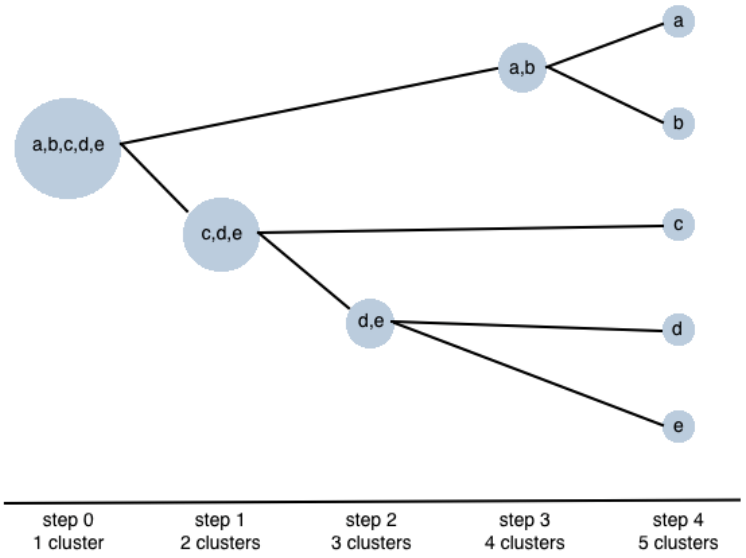


Figure 2.7: Divisive clustering. The method starts with one cluster containing all single objects (a, b, c, d, and e) and proceeds with successive splits until all clusters contain only one object. The hierarchy is built in four steps since the data contains five objects.

Balancing clusters

K-centroid clustering which is used in the divisive clustering algorithm clusters the data without respect to class labels. An unsupervised method is used to find underlying structures that do not depend on class labels. During clustering, the assignment of an object that is close to both cluster centers is not as reliable as an assignment of an object which has a large difference in distance to both cluster centers. If the objects have ground truth class labels, these labels can be used in order to find objects that possibly are wrongly assigned to a cluster. An object is possibly wrongly assigned if the majority of the objects in the same class, according to the class labels, are assigned to the other cluster.

To reduce the number of objects that are wrongly assigned to a cluster, some objects can be re-assigned to the other cluster. First, the most impure of the two clusters is found, this cluster is called the positive super cluster. The other cluster is called the negative super cluster. The most impure cluster is defined as the cluster that contains the most classes, i.e., the larger cluster. The classes of the positive super cluster are then sorted in descending order of class size, a percentage of the objects of the minority classes are moved to the negative super cluster [12]. This reduces the

noise in the positive cluster and it balances the two clusters in terms of the number of classes and size.

2.3 Similarity measures

Similarity measures play an important role in how to measure the similarity of data-features. The similarity measures that are going to be presented here are Earth mover's distance, Euclidean distance, and Cosine similarity. For these similarity measures, two data-features are similar if the similarity measure is small. Bigger values indicate that the data-feature is dissimilar.

2.3.1 Earth mover's distance

Earth mover's distance, EMD, is a similarity measure that computes the similarity between two distributions. EMD measures the similarity by computing the minimal amount of work that must be done to transform one distribution into another distribution by moving "distribution mass" around [13].

As described in *A Metric for Distributions with Applications to Image Databases* [13] for two given distributions, one of the distribution can be seen as a mass of earth properly spread in space. The other distribution can be seen as a collection of holes in the same space. Let's further assume that there are at least as much earth needed to fill the holes. EMD will be the least amount of work that is needed to fill the holes with earth. One unit of work is the same as multiplying one unit of earth by a unit of (ground) distance [13]. The formula for computing the EMD-distance is [13]:

$$EMD(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i \in I} \sum_{j \in J} c_{ij} f_{ij}}{\sum_{i \in I} \sum_{j \in J} f_{ij}}, \tag{2.14}$$

where f is the flow, and c is the ground distance. Equation 2.14 is based on the solution to the old *translation problem* [13]. In the translation problem, I is the set of suppliers and J is the set of consumers, and the cost to take a unit of supply from $i \in I$ to $j \in J$ is denoted as c_{ij}. For this problem, the set of flows f_{ij} that minimizes the overall cost is wanted

$$\sum_{i \in I} \sum_{j \in J} c_{ij} f_{ij}, \tag{2.15}$$

with the following constraints:

$$f_{ij} \geq 0 \qquad i \in I, j \in J, \tag{2.16}$$

$$\sum_{i \in I} f_{ij} = y_j \qquad j \in J. \tag{2.17}$$

$$\sum_{j \in J} \leq x_i \qquad i \in I. \tag{2.18}$$

Here x_i denotes the total supply of supplier i, and y_i is the total capacity of consumer j. Constraint 2.16, only allows the supplier to ship supplies to the consumer and not the other way around. Constraint 2.17 indicates that the consumer must fill up all of their capacities. Finally, constraint 2.18 puts a limit on the supply that the supplier can ship to its total amount.

The transportation problem can easily be used for matching between signatures or distributions. Where one is defined as the supplier and the other is defined as the consumer. The cost c_{ij} is then translated as the ground distance between element i and element j of the two distributions. When the transportation problem is solved, and when the optimal flow F is found, the earth mover distance will be defined as in equation 2.14.

2.3.2 Euclidean Distance

The Euclidean distance is measured as the root of the square between two feature vectors $\boldsymbol{A}$ and $\boldsymbol{B}$, the formula can be seen below

$$\|\boldsymbol{A} - \boldsymbol{B}\|_2 = \sqrt{\sum_i (a_i - b_i)^2}, \tag{2.19}$$

where a_i and b_i are elements of $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively. For two histograms, the Euclidean distance measures the total difference or similarity between the bins.

2.3.3 Cosine similarity

The cosine similarity is another similarity measure that measures the similarity between two vectors. Instead of looking at the magnitude, the cosine similarity measures the cosine angle between two vectors to decide if they are similar or not[5]. It is computed as

$$cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}}, \tag{2.20}$$

[5] `https://en.wikipedia.org/wiki/Cosine_similarity`

where a_i and b_i are elements of the vector $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively.

2.4 Support Vector Machine (SVM)

Support vector machine, SVM, is a supervised machine learning algorithm. SVM can be used for both classification and regression problems[6]. The method is used on both linear and nonlinear data. In this section, we will only focus on SVM for classification tasks on linearly separable data.

SVM is a classification method that finds a separating hyperplane for the labeled data. If the data is linearly separable in the input space, then SVM can find a hyperplane that separates the classes from each other [14, p. 4]. If the data is not linearly separable in the input space, then a parameter can be passed to the SVM that tells it to apply a nonlinear mapping to transform the data into a higher dimension. In this new dimension, it tries to find a linear separating hyperplane. A hyperplane can therefore always separate the data into two classes [15, p. 408]. The separating hyperplane is found using *support vectors* and *margins*.

2.4.1 Linearly separable data

For the simplest case, let's look at a two-class problem that is linearly separable. Let D denote the data set which is given by $(\boldsymbol{X}_1, y_1), (\boldsymbol{X}_2, y_2), ..., (\boldsymbol{X}_{|D|}, y_{|D|})$. Where $\boldsymbol{X}_i$ is the training tuples and y_i is the associated class label. For a two class problem, y_i can take two values, let's say that y_i is either $+1$ or -1. To better understand the problem, consider an example with two input attributes A_1 and A_2, see Figure 2.8.

As illustrated in Figure 2.8, the data can be separated by drawing a straight line between the two classes. For this problem, there are multiple ways of drawing a line that separates the classes, as can be seen in the same figure. Intuitively, one wants to find the line that separates the classes the "best". The "best" one is the one that will have a minimum classification error on the test data. For 3-D data problems, the best separating plane is wanted. If generalized to n dimensions, the best hyperplane is desired. The term "hyperplane" will be used to refer to the decision boundary regardless of the dimension.

SVM is a method that finds the best separating hyperplane by searching for the maximum *marginal* hyperplane. In Figure 2.9 two potential hyperplanes with their respective margins are illustrated. The two hyperplanes classify the two classes

[6] `https://en.wikipedia.org/wiki/Support_vector_machine`

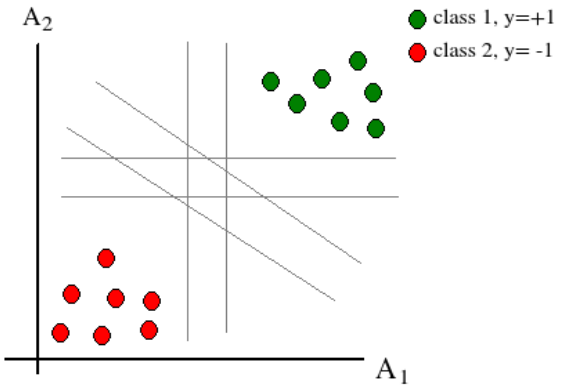


Figure 2.8: The two class problem in 2-D is linearly separable. As seen in this figure, there are many possibilities to separate the classes from each other. Some of these possibilities are drawn as lines.

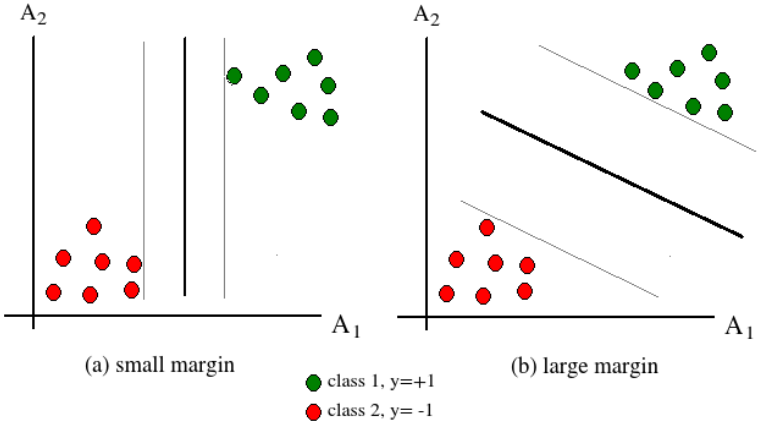


Figure 2.9: Two possible separating hyperplanes with their respective margins. In a) a hyperplane with a small margin, and in b) a hyperplane with a large margin.

correctly. However, the hyperplane that classifies the data the best need to be considered. The hyperplane in Figure 2.9 b) is expected to be more accurate in classifying new data, than the one with a small margin. Since the one with a larger margin is more robust as it has the largest separation between the two classes. During training, SVM will search for the hyperplane with the largest margin, that is, the *maximum marginal hyperplane*, MMH [15, p. 409]. The length of the margin is defined as the shortest distance from the MMH to the closest training tuple of either class. Note that the length of the margin on both sides of the hyperplane is equal [15, p. 410].

Mathematically the separating hyperplane can be written as

$$\boldsymbol{W} \cdot \boldsymbol{X} + b = 0, \tag{2.21}$$

where $\boldsymbol{W}$ is a weight vector $\boldsymbol{W} = \{w_1, w_2, ..., w_n\}$, n is the number of attributes (dimension), and b is a bias. Let's again consider two-class problem in 2-D. The training tuples in 2-D are $\boldsymbol{X} = (x_1, x_2)$, where x_1 and x_2 are values of the attributes A_1 and A_2. Further, let's think of b as an additional weight w_0 [15, p. 410], equation 2.21 for the hyperplane can be rewritten as

$$w_0 + w_1 x_1 + w_2 x_2 = 0. \tag{2.22}$$

Points that lie above this plane will satisfy the following condition

$$w_0 + w_1 x_1 + w_2 x_2 > 0. \tag{2.23}$$

Points that lie below will satisfy

$$w_0 + w_1 x_1 + w_2 x_2 < 0. \tag{2.24}$$

Instead of defining the hyperplane, the "sides" of the margin can be defined, after some adjustments, as

$$H_1 : w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \tag{2.25}$$

$$H_2 : w_0 + w_1 x_1 + w_2 x_2 \leq -1 \quad \text{for } y_i = -1. \tag{2.26}$$

Points that lie on or above H_1 belongs to class $+1$, and points that lies on or below H_2 belongs to class -1. The two equations above can be combined to get the following [15, p. 411]

$$y_i(w_0 + w_1 x_1 + w_2 x_2) \geq 1, \forall i. \tag{2.27}$$

The training tuples that lie on the hyperplanes H_1 or H_2 satisfy eq. 2.27 and are called *support vectors*. In Figure 2.10 the support vectors are shown. The support vectors are very hard to classify. However, they give much information regarding the classification [15, p. 411] as they define the maximum marginal hyperplane.

The complexity of the support vector machine classifier is not described by the dimension of the data, but by the number of support vectors [15, p. 412]. Thus, SVM is less likely to suffer from overfitting than other methods. The support vectors of this method is essential. If all other training tuples were removed and the model trained again, the same separating hyperplane would be found. If a small number of support vectors are found for high dimensional data, the SVM can still have a good generalization performance.

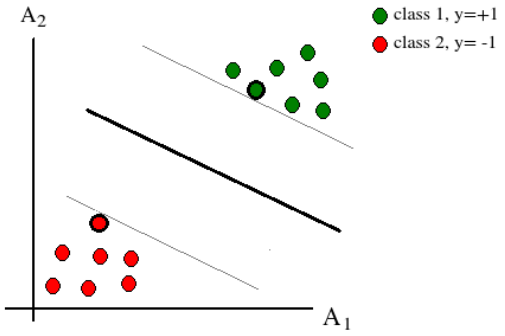


Figure 2.10: The support vectors are shown in a thicker border. The support vectors are the data tuples that lies on the margin.

Margin computation

As has been described above, SVM finds the hyperplane with the largest margin between the different classes as the decision boundary. Let's take the two-class problem in 2-D again and say that $\boldsymbol{X}_+$ and $\boldsymbol{X}_-$ lie on either side of the separating plane, and on the margin. By definition, $\boldsymbol{X}_+$ and $\boldsymbol{X}_-$ are the support vectors, and they fulfill $\boldsymbol{W} \cdot \boldsymbol{X}_+ + b = 1$ and $\boldsymbol{W} \cdot \boldsymbol{X}_- + b = -1$. The distance between a point on the margin and the hyperplane can be computes as

$$d = \frac{g(\boldsymbol{X})}{\|\boldsymbol{W}\|}, \tag{2.28}$$

where $g(\boldsymbol{X}) = \boldsymbol{W} \cdot \boldsymbol{X} + b$, and $\|\boldsymbol{W}\|$ is the Euclidean norm of $\boldsymbol{W}$. For computational details on d see [16, p. 44]. The distance from any point on H_1 to the separating hyperplane is $\frac{1}{\|\boldsymbol{W}\|}$. By definition, this distance is equal to the distance from any point on H_2 to the separating hyperplane. The total length of the margin can then be computed as the distance between the two support vectors as:

$$\begin{aligned} M = d_+ - d_- &= \frac{g(x_+) - g(x_-)}{\|\boldsymbol{W}\|} = \frac{\boldsymbol{W} \cdot \boldsymbol{x}_+ + b - (\boldsymbol{W} \cdot \boldsymbol{X}_- + b)}{\|\boldsymbol{W}\|} \\ &= \frac{1 + b - (-1 + b)}{|\boldsymbol{W}\|} = \frac{2}{\|\boldsymbol{W}\|}, \end{aligned} \tag{2.29}$$

or by summing the distances from the hyperplane to the margin on either side. The margin is inversely proportional to $\|\boldsymbol{W}\|$. SVM searches for the hyperplane with maximum margin, thus by minimizing $\|\boldsymbol{W}\|$ the margin is maximized.

Multi-class classification

The support vector machine is designed for binary classification, classification with only two classes [15, p. 430]. There are different approaches one can take to extend these algorithms to allow for multi-class classification, classification with more than two classes. One of these approaches is *all-versus-all* (AVA), sometimes also called one-versus-one (OVO). In this approach, a classifier is learned for each pair of classes. Given m classes, there will be $\frac{m(m-1)}{2}$ binary classifiers learned. During training of the classifiers, data tuple from two classes is used. In other words, a pair of classes are considered at a time. If an unknown tuple needs to be classified, the learned classifiers votes [15, p. 431]. The unknown tuple is assigned to the class with most votes.

2.5 Prediction of a verification template

In the previous sections, different methods for training and classifying data have been discussed. But how to classify the test data has yet been mentioned. In the following section, two different methods to label the test data based on which method is used during training will be discussed.

2.5.1 Centroid based prediction

The centroid based prediction is used to classify the test data if the K-centroid method has been used for classifying train data. The output of the K-centroid method is labels of our training data and the K cluster centers. For a given test data, the similarity between the K centers and the test data is computed, using any of the similarity measures mentioned in section 2.3. The similarity measure that is used during training is also used for prediction. When the similarity between test data and the different centers have been computed, the test data is assigned to the cluster with the smallest similarity measure. To put it in mathematical terms

$$\min_k [D(\boldsymbol{x}^{\boldsymbol{T}}, \boldsymbol{\mu}_k)]_1^K, \tag{2.30}$$

where $\boldsymbol{x}^T$ is the test data, $\boldsymbol{\mu}$ is the set of centers, and D is the chosen similarity measure.

2.5.2 Prediction using SVM

Prediction using the trained SVM model is used if SVM has been used during training. The output from a trained SVM model is the support vectors, and they define the MMH. Using some mathematical techniques, the MMH can be rewritten to define the decision boundary (for computational details see [15, p. 412])

$$d(\boldsymbol{X}^T) = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{X}_i \boldsymbol{X}^T + b_0, \tag{2.31}$$

where y_i is the class label of the support vector $\boldsymbol{X}_i$, $\boldsymbol{X}^T$ is a test tuple, α_i and b_0 are parameters that can be determined during the training of SVM, or by an optimization method [15, p. 412] [16, p. 43].

For a given test data $\boldsymbol{X}^T$ in equation 2.31, the sign of the result is checked. The sign decides which side of the hyperplane the test data falls on, and which class the test data belongs to. If it is positive, then the test data is on or above the MMH. If the sign is negative, then it falls on or below the MMH.

Chapter 3

Dataset and Tools

In this chapter, the datasets and tools used in this project are introduced. Section 3.1 describes the general structure of a database at Precise Biometrics and the details of two datasets that are used as a basis for the evaluations in chapter 5. A presentation of ground truth for the mentioned datasets is also given. The tools, e.g., the programming language and libraries used in the implementation of the proposed setups, are introduced in section 3.2.

3.1 Data

A fingerprint image database at Precise Biometrics is constructed using volunteers who provide samples from multiple fingers. The volunteers typically provide multiple samples both for enrollment and verification for each finger. Multiple samples for enrollment are captured in case of using a small sensor in order to cover the whole fingerprint. Each sample has a label with information about which person and which finger the sample comes from, the samples are also labeled according to the order in which the samples are captured.

The experiments in this thesis are performed on one dataset which contains large images. The dataset is described in detail in the following section.

Dataset

The dataset consists of fingerprint images from 111 persons. From each person, images from six fingers are captured. The captured fingers are the thumb, index

finger and middle finger from both left and right hands. There are 72 captured images from each finger. In the experiments, 16 images from each finger are used, resulting in $111 \cdot 6 \cdot 16 = 10656$ images in total. Each person also provided 10 verification images for each captured finger, resulting in $111 \cdot 6 \cdot 10 = 6660$ verification templates in total. The size of each image is 192×192 pixels, which is considered as rather large images.

Ground truth

For each database, a text file containing information about how the enrolled samples overlap each other, and how the verification samples overlap the enrolled samples can be generated. The information about overlap is used as ground truth when evaluating an experiment in this thesis.

An example of a generated text file containing ground truth is shown in Figure 3.1. The ground truth in this example is for one finger which has 7 enrolled samples. Each row in the text file gives the ground truth for one sample, the samples are counted from 0. The first column (from left) is the image file path. Columns 3-5 are information about how each sample is aligned in a multi-template, the multi-template consists of all enrolled samples from one finger. Each sample in the multi-template is assigned a translation, (dx, dy), and a rotation, θ. The last columns are the overlapping area between all samples, the overlap is represented as a percentage of the sample area. For example, the first row and the seven last columns says that; the overlap between sample 0 and itself is 99%, the overlap between sample 0 and sample 1 is 4%, there is no overlap between sample 0 and sample 2 or sample 3, the overlap between sample 0 and sample 4 is 83%, the overlap between sample 0 and sample 5 is 27%, and the overlap between sample 0 and sample 6 is 70%. Following rows can be read in the same way.

	Alignment (dx, dy, rotation) in HR			Area overlap in %						
01-000.png :	0	3495	4060	64256	99	4	0	0	83	27	70
01-001.png :	0	5219	-35116	64256	5	100	49	55	9	0	0
01-002.png :	0	22272	-44285	64768	0	49	100	15	0	0	0
01-003.png :	0	-12191	-34370	64000	1	56	15	100	6	0	0
01-004.png :	0	-1136	2051	64256	84	8	0	6	99	19	69
01-005.png :	0	14650	31255	63488	26	0	0	0	18	100	40
01-006.png :	0	634	15254	63744	69	0	0	0	68	41	99

Figure 3.1: Example of generated ground truth. The first column is the image path, columns 3-5 are alignment information and the last 7 columns are overlap information.

In this thesis, the overlap data is used to calculate a score in order to evaluate the experiments. How to calculate scores is described in section 4.5.

3.2 Tools

The programming language and libraries that are used for our proposed setup and analysis is going to be introduced briefly in this section.

3.2.1 Python

Python is an interpreted high-level programming language for general-purpose programming. It features dynamic semantics system and has automatic memory management. Python also supports multiple programming paradigms which includes object-oriented, functional and procedural programming. It also supports modules and packages and has a large standard library[1].

3.2.2 OpenCV

OpenCV (Open Source Computer Vision Library) is, as the name implies, an open source computer vision and machine learning software library. There are more than 2500 optimized algorithms which is a set of both classic and state-of-art algorithms used for computer vision and machine learning. The algorithms that the library provides can be used for tasks like face detection and recognition, object identification and finding similar images from a database. OpenCV has C++, Python and Java interfaces and supports Windows, Linux, Android and Mac OS[2].

3.2.3 Scikit-learn

Scikit-learn is a free software machine learning library for Python. It has various machine learning applications such as classification, regression and clustering algorithms. Such algorithms are support vector machines, random forest and k-means. It is designed to operate with Python's numerical and scientific libraries, NumPy and SciPy[3].

[1] https://www.python.org/doc/essays/blurb/
[2] https://opencv.org/about.html
[3] https://en.wikipedia.org/wiki/Scikit-learn

3.2.4 SciPy

SciPy is a open source library for Python. It is used for scientific computing and technical computing[4]. The modules in this library is for optimization, linear algebra, signal- and image processing and many more.

[4] https://en.wikipedia.org/wiki/SciPy

Chapter 4

Experimental setup

4.1 Introduction

The main objective of this thesis is to investigate whether it is possible to apply clustering on the database provided by Precise Biometrics, to reduce the searching time. The goal is to cluster the database into a smaller subset by looking at the property of each finger. Hence, if a person is trying to verify her-/himself, the system will search through only one of these subsets and not the whole database to find a possible match.

Properties that distinguishes one finger from another is desirable. It can be global ridge flow patterns, local ridge characteristics, intra-ridge details, and scars on the finger. These properties are called *features* which the clustering makes use of. If there's a variety of features and they are distinct, the clustering will hopefully be better, as it can cluster fingers with similar features together. In such a case, the variance between two clusters will be high, since the clusters contain fingers of different characteristics. Apart from distinct features, an appropriate *similarity measure* is also needed. Since it measures the similarity between different features to decide whether they are similar or not. The features that are used for our proposed setup is frequency histograms, orientation histograms, and bag-of-words histograms. The corresponding similarity measures that are going to be tested are Earth mover's distance, Euclidean distance, and Cosine similarity. The purpose is to find the best combination of feature and similarity measure.

The main clustering method used in this thesis, for clustering the features, is *K-centroid*. There are different ways of applying clustering to a set of features. The ones that are investigated in this project is; to cluster the features in K clusters "directly";

and through hierarchical clustering which uses a tree-like clustering process. Divisive clustering is an approach in hierarchical clustering which is going to be tested.

In this chapter, the overall strategy and how the experiment is setup for analysis is described. It is organized as follows; in section 4.2, how the features are generated is described. In section 4.3, the general steps in K-centroid is introduced, and where the different similarity measures are taken from is also presented. In section 4.4, how to use OpenCv's support vector machine algorithm is explained. In the last section, how the performance of our proposed setup is evaluated is described.

4.2 Features

In this section, the main steps of generating the different feature histograms needed for our research is provided.

4.2.1 Frequency and Orientation histograms

The frequency and orientation histograms used for our research are provided by Precise Biometrics. They are provided as feature vectors, where each element are bins in the histogram.

4.2.2 Bag-of-Words

In our work, OpenCV's implementation of Bag-of-words has been used. The steps to generate a bag-of-word representation for the images in the database can be summarized as follows:

1. Set up the BoW, by initializing the number of words.
   `cv2.BOWKMeansTrainer(nbr_words)`
   For this experiment: `nbr_words` = 1000, 5000, and 10000
2. Initialize the detector. Use either the GFTT or the SIFT detector. Decide on the number of keypoints that are going to be detected.
   `detector = cv2.GFTTDetector_create(maxCorners = nbr_keypoints)`
   `detector = cv2.xfeatures2d.SIFT_create(nfeatures = nbr_keypoints))`
   For this experiment: `nbr_keypoints` = 50, 100, 250, 200 and 250

3. Initialize the SIFT descriptor. The SIFT descriptor is always used in combination with different detectors.
   ```
   descriptor = cv2.xfeatures2d.SIFT_create()
   ```
4. (optional) Add image processing to the dataset if needed.
5. Detect keypoints in the image and compute the descriptor for the keypoints. Add the descriptor to the BoW trainer.
   ```
   kpts = detector.detect(img,None)
   _, feature = descriptor.compute(img, kpts)
   bow_trainer.add(np.float32(feature))
   ```
6. Train the BoW trainer to get the vocabulary.
   ```
   vocab = bow_trainer.cluster().astype(feature.dtype)
   ```
7. Set up the BoW descriptor that is used to find the subset of words for each image in the dataset.
   ```
   bow_descr = cv2.BOWImgDescriptorExtractor(descr,cv2.BFMatcher(cv2.NORM_L2))
   bow_descr.setVocabulary(vocab)
   ```
8. Compute the BoW-features for each image in the dataset. The returned features are a set of histograms. Each histogram is a subset of words in the vocabulary.
   ```
   kps = detector.detect(img,None)
   bow_decr.compute(img, kps)
   ```
9. Weight the bins in the BoW-histograms using the tf-idf term.

BoW vocabulary

In the OpenCV implementation for BoW, the BoW-trainer uses K-means clustering to obtain the vocabulary. If a vocabulary of 500 words is wanted, then a K-means clustering with 500 clusters is trained. The center of each of these clusters is a word in the vocabulary. The K-means implementation is using a mean-centroid update. That is, for each iteration the centers are updated by taking the mean of the data points in each cluster. The distance metric used in this implementation is the Euclidean distance.

4.3 K-centroid

When the features have been generated, it is time to perform clustering on the data using the K-centroid method. The implementation of the K-centroid is done by the authors, and the main steps can be summarized as follows:

1. Initialize the K centers by randomly select K of the data points as centers.
2. Cluster all data points to one of the K clusters by assigning the data point to the nearest cluster center. Use a suitable similarity measure.
3. Update the center of each cluster.
4. Check if the old centers (from the last iteration) is equal to the new centers. If they are equal stop the clustering otherwise continue to step 2.

The main steps for the K-centroid algorithm can be seen above. In step two, the data points are assigned to one of the K clusters. The similarity measure that is used is Earth mover's distance, Euclidean distance, or Cosine similarity. These three similarity measures are tested on the frequency feature. For bag-of word feature, the last two similarity measures are used. The similarity measure used for the orientation feature is EMD.

The functions for computing the Euclidean distance and Cosine similarity distance are taken from the SciPy library. With the following code the two similarity measures are accessed:

```
scipy.spatial.distance.euclidean(hist1,hist2)
scipy.spatial.distance.cosine(hist1,hist2)
```

The method for computing EMD is taken from openCVs c++ implementation:

```
EMD(hist1, hist2, distType = cv2.CV_DIST_L2)
```

All the code in this thesis is written in Python, but OpenCV's function for EMD is in c++. That's why a Python binding had to be done to be able to use this function.

4.3.1 Hierarchical clustering

Divisive

The hierarchical divisive clustering method is implemented by dividing the cluster in the next step of the hierarchy using the K-centroid clustering method. The K-centroid clustering method clusters a cluster in the hierarchy into two new clusters. The method is interrupted before it reaches its final state where each cluster contains

only one histogram. When to stop is decided using a pre-defined threshold related to the cluster sizes. The method is interrupted when the size of the clusters, which have not been divided, is smaller than the pre-defined threshold.

The clustering of the two new clusters after each step in the hierarchy can be done in three alternative ways. The first alternative is to label the data only based on the clustering from the K-centroid algorithm. The second alternative is to balance the clusters (section 2.2.2) after the clustering, label the data according to the modified clusters and then train an SVM classifier using the modified clusters. The third alternative is to to train an SVM classifier using the clusters obtained from the clustering. The resulting hierarchy using the divisive clustering method is obtained by the following steps:

1. Cluster all data points into two clusters.
2. Find the largest cluster which has not yet been divided.
3. Cluster the largest cluster into two new clusters.
4. (Optional) Balance the clusters.
5. (Optional) Train a SVM classifier using the labels obtained from the clustering or the labels obtained from the balancing.

The chosen steps are iterated until the size of all clusters to be divided are smaller than the pre-defined threshold. The first alternative iterates step 2 and 3. The second alternative iterates step 2-5. The last alternative iterates step 2,3, and 5.

4.4 Support vector machine

The support vector machine classifier implementation is taken from the Scikit-learn library. The main functions that are used can be seen below:

Initializing the SVM by passing the argument 'ovo' for multi-class classification and specifying that the data is linearly separable
`clf = svm.SVC(decision_function_shape='ovo', kernel='linear')`

Train the classifier
`clf.fit(data, labels)`

Prediction or classification of data is done by
`clf.predict(test_data)`

4.5 Performance measure

Performance measures are used in order to evaluate the performance of each experiment. In this thesis, two different scores will be calculated and used as performance measures. Before describing how to calculate the scores, some notation regarding enrollment samples and verification samples are introduced.

Notation

The set of enrollment samples for a finger, $n = 1 \dots N$ where N is the number of captured fingers, is denoted

$$\mathcal{T}_n = \{\mathbf{T}_{n,1}, \dots, \mathbf{T}_{n,M}\},$$

where M is the number of enrollment samples for each finger. The set which contains the sets of enrollment samples for all captured fingers is denoted

$$\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, \dots \mathcal{T}_N\}.$$

This is the dataset that is clustered using some of the clustering methods described in section 2.2.

The set of verification samples for a finger, n, is denoted

$$\mathcal{I}_n = \{\mathbf{I}_{n,1}, \dots, \mathbf{I}_{n,P}\},$$

where P is the number of verification samples for each finger. The set which contains the sets of verification samples for all captured fingers is denoted

$$\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, \dots \mathcal{I}_N\}.$$

A verification sample, p, captured from one finger, n, creates a pair with the set of enrollment samples captured from the same finger,

$$(\mathcal{T}_n, \mathbf{I}_{n,p}),$$

where $p = 1 \dots P$.

The clustering methods presented in section 2.2 clusters the data into K clusters, these clusters form a set of clusters

$$\mathcal{C} = \{\mathbf{C}_1, \dots \mathbf{C}_K\}.$$

Each verification sample is assigned, one by one, to one of the clusters in $\mathcal{C}$. The enrollment samples in the chosen cluster which are captured from the same finger as the assigned verification sample is found. Two scores based on the information from the found enrollment samples are calculated. The first score is based on the overlap between the found enrollment samples and the verification sample, this score is denoted S_o. The second score is based on if any enrollment samples were found or not, this score is denoted S_f.

Score S_o

This score is an average score based on how much each verification sample overlaps all enrollment samples from the same finger which are assigned to the same cluster as the verification sample. The overlap is expressed in percent of the verification sample area. The overlap between two samples can be seen as the possibility to match. If the overlap is large, the possibility to make a correct match decision is large while it is harder to make a correct decision if the overlap is small.

For each sample pair, $(\mathcal{T}_n, \mathbf{I}_{n,p})$, the score based on the overlap between the verification sample and all enrollment samples will be calculated as,

$$S(\mathcal{T}_n, \mathbf{I}_{n,p}) = 1 - \prod_{m=1}^{M} 1 - \frac{\mathbf{T}_{n,m} \cap \mathbf{I}_{n,p}}{100}, \tag{4.1}$$

where the intersection is the overlap between the verification sample, $\mathbf{I}_{n,p}$, and one of the enrollment samples, $\mathbf{T}_{n,m}$. If the enrollment sample, $\mathbf{T}_{n,m}$, is assigned to a different cluster than the verification sample, the intersection is 0. The term inside the product will, therefore, be equal to 1 and it will not affect the score.

The scores of all sample pairs, $(\mathcal{T}_n, \mathbf{I}_{n,p})$, are added together and then divided by the total number of captured verification samples,

$$S_o = \frac{1}{N \cdot P} \sum_{n=1}^{N} \sum_{p=1}^{P} S(\mathcal{T}_n, \mathbf{I}_{n,p}). \tag{4.2}$$

The score is bounded by $0 \leq S_o < 1$. A higher score corresponds to a better performance in the experiment.

Score S_f

This score is a performance measure describing how many of the verification samples that are correctly assigned to a cluster. Here, correctly assigned means that the chosen cluster should contain at least one of the enrollment samples captured from the same finger as the verification sample.

If none of the enrollment samples, which are in a pair with the verification sample, are found in the cluster to which the verification sample is assigned, the score of that sample pair is 0. If at least one enrollment sample, which is in a pair with the verification sample, is found in the cluster that $\mathbf{I}_{n,p}$ is assigned to, the score of that sample pair equals 1. This is, for each pair of samples $(\mathcal{T}_n, \mathbf{I}_{n,p})$, mathematically expressed as

$$\begin{aligned} (\forall \mathbf{T} \in \mathcal{T}_n : \mathbf{T} \notin \mathbf{C}_k) \wedge \mathbf{I}_{n,p} \in \mathbf{C}_k &\implies S(\mathcal{T}_n, \mathbf{I}_{n,p}) = 0, \\ (\exists \mathbf{T} \in \mathcal{T}_n : \mathbf{T} \in \mathbf{C}_k) \wedge \mathbf{I}_{n,p} \in \mathbf{C}_k &\implies S(\mathcal{T}_n, \mathbf{I}_{n,p}) = 1, \end{aligned} \tag{4.3}$$

where $\mathbf{C}_k$ is the cluster which the verification template is assigned to and $k = 1 \dots K$.

The scores of all sample pairs are added together and then divided with the total number of captured verification samples,

$$S_f = \frac{1}{N \cdot P} \sum_{n=1}^{N} \sum_{p=1}^{P} S(\mathcal{T}_n, \mathbf{I}_{n,p}). \tag{4.4}$$

The score is bounded by $0 \leq S_f \leq 1$. A higher score corresponds to a better performance in the experiment.

Chapter 5

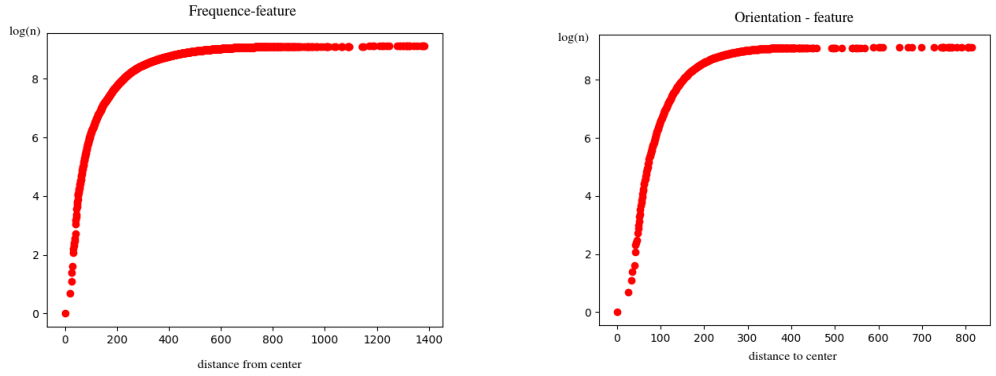
Result

The experimental results are presented in this chapter. First, a visualization of the data represented by the different features is presented in section 5.1. Performance of the experiments made when using frequency and orientation features are presented in section 5.2. In section 5.3, performance of the experiments performed when using the bag-of-words feature is presented. An example of three BoW histograms and the similarity between them are also presented in section 5.3. Finally, some images from fingers that often seem to have low performance are presented.

5.1 Visualization of the data

Since it is hard to picture the data in a very high dimensional space, a simple visualization is done. The visualization illustrates how the data is distributed in the high dimensional space. For this visualization, all data points are considered to be in one large cluster. First, the centroid of the cluster is found. The similarities between the centroid and all other data points are then computed. The data is visualized using the frequency feature, the orientation feature, and the BoW feature generated with the GFTT detector.

Figure 5.1a shows the visualization of the data using the frequency feature. The visualization of the data using the orientation feature is shown in Figure 5.1b. Figure 5.2 shows the visualization of the data using the BoW feature generated with the GFTT detector. The x-axis measures the similarity from the centroid, which can be viewed as a radius from the center. The y-axis is the logarithm of the number of data points which lie inside or on the sphere which is spanned by the similarity radius.



(a) Frequency feature (b) Orientation feature

Figure 5.1: Visualization of the data using the frequency feature (a) and the orientation feature (b). Both of these features have been clustered using the EMD similarity measure

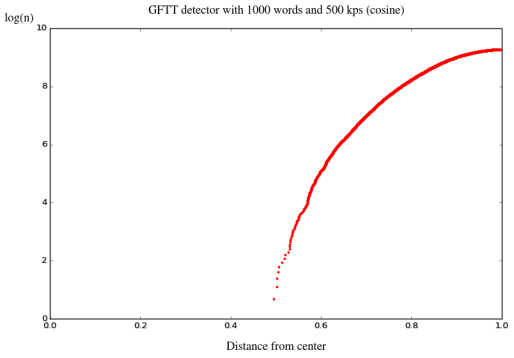


Figure 5.2: Visualization of the data using the bag-of-words feature generated with GFTT detector. The BoW histograms were produced by a combination of 500 keypoints and a vocabulary containing 1000 words.

5.2 Results from frequency and orientation features

Performances of the experiments made when using the frequency features and the orientation features are presented in this section. Section 5.2.1 presents the performance obtained using direct clustering. The performance obtained using divisive clustering is presented in section 5.2.2. For comparison, section 5.2.3 presents the performance of a method for predicting the label of a verification template without using clustering.

5.2.1 Direct clustering

The K-centroid clustering method is applied using different number of clusters on both orientation and frequency features, called direct clustering. The number of clusters varies from $2 - 69$ clusters. The result of comparing the frequency features clustered with different similarity measures is shown in Figure 5.3. A comparison between clustering the frequency feature combined with EMD and the orientation feature combined with EMD is presented in Figure 5.4. In table 5.1, the best scores obtained from direct clustering are presented. The results shows that the frequency feature in combination with Euclidean distance performed the best.

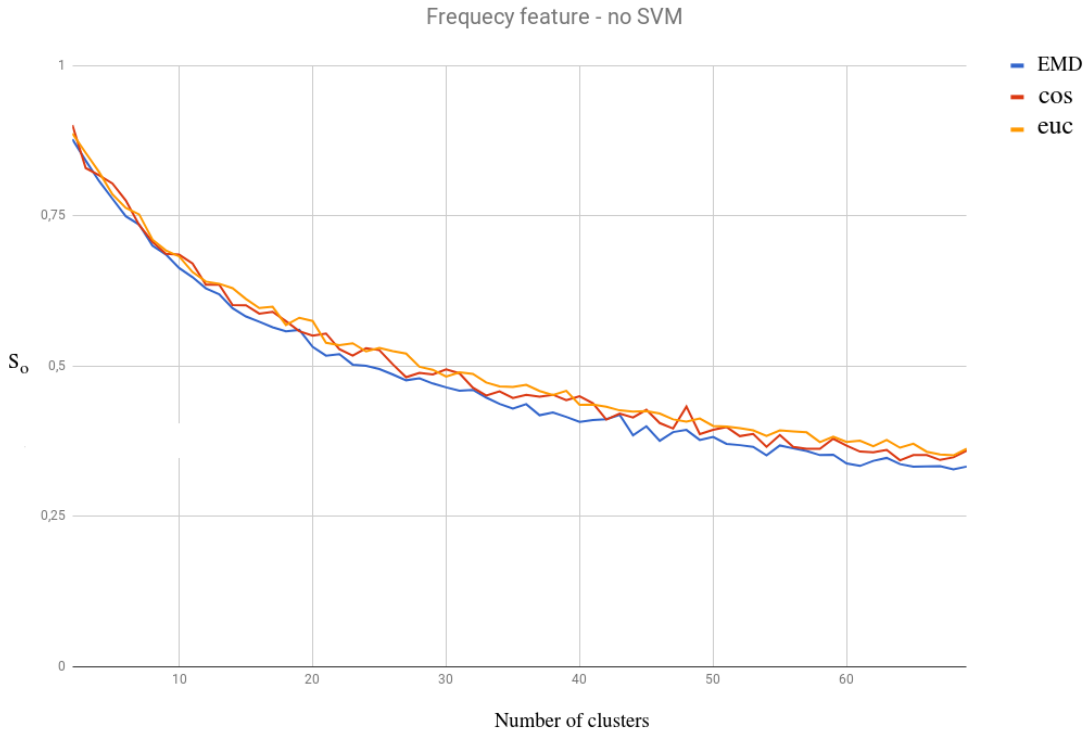


Figure 5.3: Clustering of frequency features into 2-69 clusters using different similarity measures. The blue line shows clustering with EMD distance, the red line is clustering with Cosine similarity, finally, in yellow the Euclidean distance clustering is plotted. In this figure, it can be seen that for frequency feature, the Euclidean distance measure has a performance higher than the two other measures.

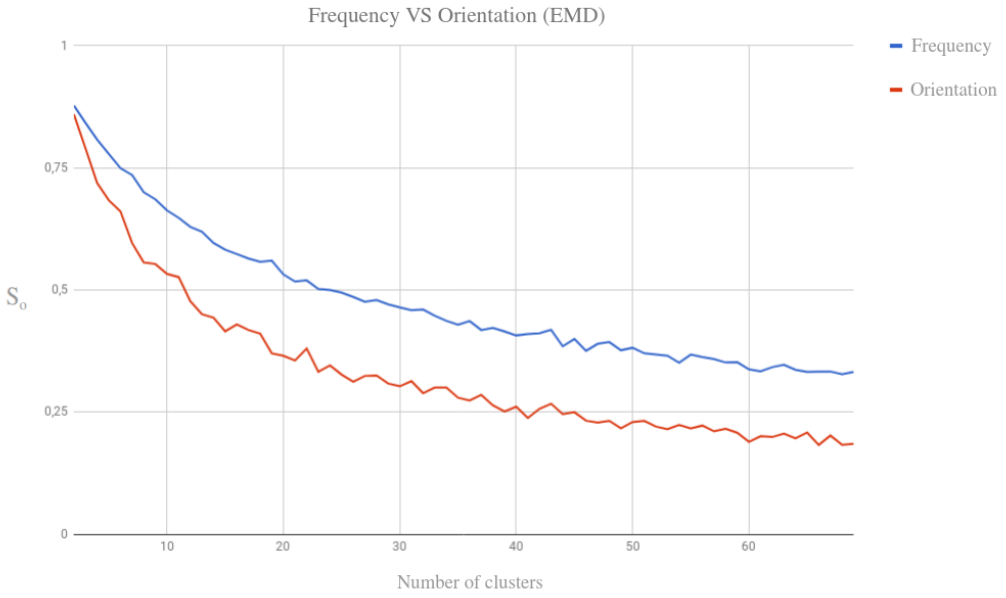


Figure 5.4: Comparing clustering of frequency and orientation features, using EMD similarity measure. Using the frequency feature as a representation of the images has better performance than representing the images using the orientation feature. The blue line in the figure shows clustering with frequency features. In red is clustering using orientation features.

Table 5.1: The best scores obtained by direct clustering of the frequency features is shown in the following table.

	Euclidean distance	
Feature/Score	S_o	S_f
Frequency	0.3624	0.5275

5.2.2 Divisive clustering

In direct clustering, the cluster sizes of the different clusters may vary a lot. Thus, divisive clustering is applied on both orientation and frequency features. The result of comparing frequency features clustered with different similarity measures and orientation features clustered with EMD can be seen in Figure 5.5. In Figure 5.6, a comparison between clustering using SVM, balancing in combination with SVM, and without SVM can be seen. In table 5.2, the best scores obtained from divisive clustering are presented. The frequency feature in combination with Euclidean distance without balancing and SVM performed the best.

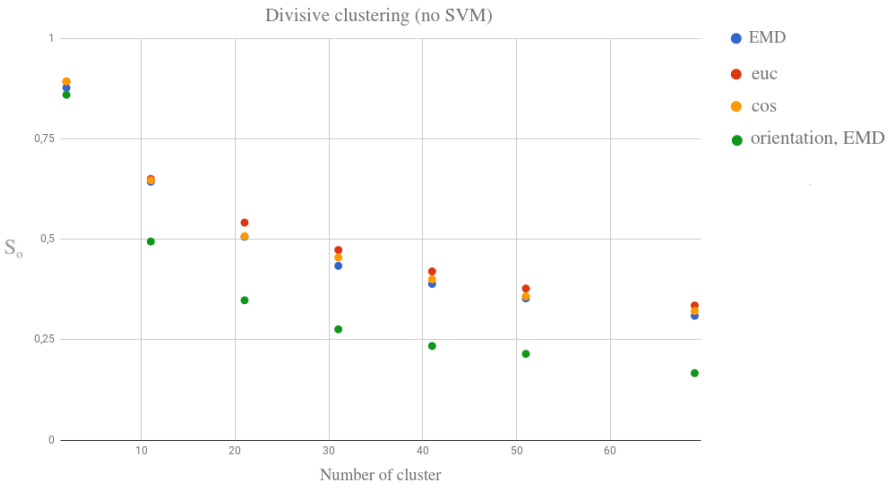


Figure 5.5: Clustering using the divisive approach to compare the different similarity measures for the frequency feature. The performances are plotted as dots where the blue dots represent EMD, red is for Euclidean distance, yellow dots are for Cosine similarity, and finally, in green are the performance for clustering orientation features using EMD.

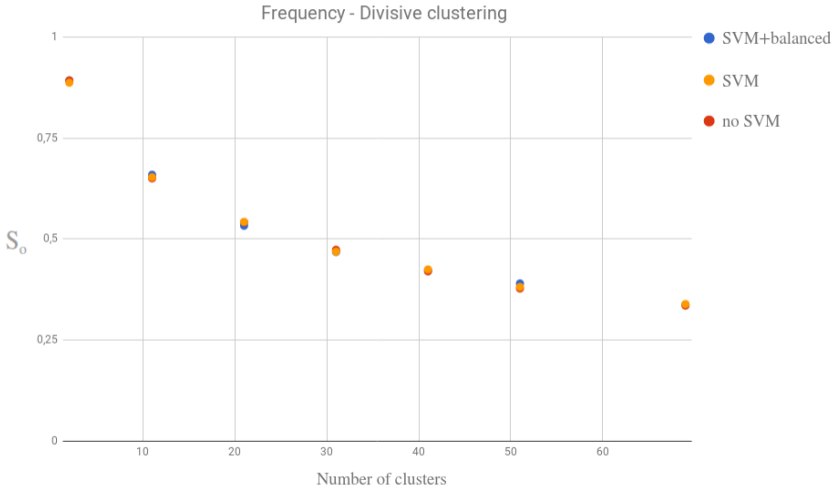


Figure 5.6: Clustering using the divisive approach with Euclidean distance on the frequency feature, with SVM and SVM combined with balancing, and without SVM. In blue dots are frequency features clustered with SVM and balancing, in yellow only SVM was used, and in red no SVM or balancing were applied.

Table 5.2: The best scores obtained from divisive clustering. Using the frequency feature with Euclidean distance without balancing and SVM showed the best performance.

	Euclidean distance	
Feature/Score	S_o	S_f
Frequency	0.3406	0.5085

5.2.3 Without clustering

An experiment which predicts the labels of the verification templates without clustering was performed. Instead of clustering the data and label it according to the clusters, the data is manually labeled according to the person from which the templates are captured. The templates are captured from 111 persons in total, it will, therefore, be 111 different labels. Each person has one label and is considered as one cluster. The verification templates are predicted using two methods. First, an SVM classifier is used. The second method computes the centroids of the clusters. The verification template is assigned to the most similar centroid. The scores obtained from these two methods are shown in table 5.3.

Table 5.3: Scores obtained from two methods for prediction of verification templates without using clustering.

	SVM		**Centroids**	
Feature/Score	S_o	S_f	S_o	S_f
Frequency	0.06892	0.07072	0.04124	0.04264

5.3 Results from bag-of-words features

Performances of the experiments made when using the BoW feature combined with SIFT and GFTT are presented in this section. The performances, that the decision of which detector and similarity measure to use is based on, is presented in section 5.3.1. The performance obtained using divisive clustering is presented in section 5.3.2. For comparison, section 5.3.3 presents the performance of a method for predicting the label of a verification template without using clustering. Some examples of BoW histograms and the similarities between them are presented in section 5.3.4. Some images from fingers that often get low performances are presented in section 5.3.5.

5.3.1 Detector and similarity measure evaluation

An evaluation of the detectors, the similarity measures and the use of weighted and non-weighted features were made. The purpose of the evaluation is to chose a detector combined with a similarity measure that would achieve the best performance in the subsequent experiments. A vocabulary containing 5000 words and 200 keypoints from the images are used in this evaluation. A comparison between the scores obtained, when clustering the features using the Euclidean distance into two clusters, using the SIFT and GFTT detectors both when weighting the features and when not weighting them is shown in table 5.4. Table 5.5 shows a comparison of clustering weighted features, generated with SIFT and GFTT detectors, into two clusters with the cosine similarity.

Table 5.4: Comparing SIFT and GFTT detectors by clustering the features, weighted and non-weighted, into two clusters using Euclidean distance.

	Weighting		**No weighting**	
Detector/Score	S_o	S_f	S_o	S_f
SIFT	0.8426	0.9725	0.8484	0.9667
GFTT	0.8515	0.9659	0.9287	0.9898

Table 5.5: Comparing features generated with the SIFT and the GFTT detectors, by clustering the features, weighted and non-weighted, into two clusters using cosine similarity.

	Weighting	
Detector/Score	S_o	S_f
SIFT	0.8700	0.9669
GFTT	0.8715	0.9737

5.3.2 Divisive clustering

In this section, bag-of-words features are generated with the GFTT detector and the SIFT descriptor. All BoW features are weighted. The divisive clustering approach using cosine similarity is applied to the BoW features. The divisive clustering is done without balancing and without the use of an SVM. The number of divisions is 68. In Figure 5.7, the scores obtained when clustering features generated from a vocabulary of 5000 words and different numbers of keypoints can be seen. In Figure 5.8, the scores obtained when clustering features generated with different numbers of

words and 100 keypoints are presented. Table 5.6 presents the scores obtained when clustering features generated by a vocabulary containing 1000 words in combination with 250 and 500 keypoints. Figure 5.9 presents the overlap score when clustering, using cosine similarity on BoW features generated with the GFTT detector, 500 keypoints, and 1000 words in the vocabulary, into a varied number of clusters. Both clustering using SVM and SVM in combination with balancing are shown.

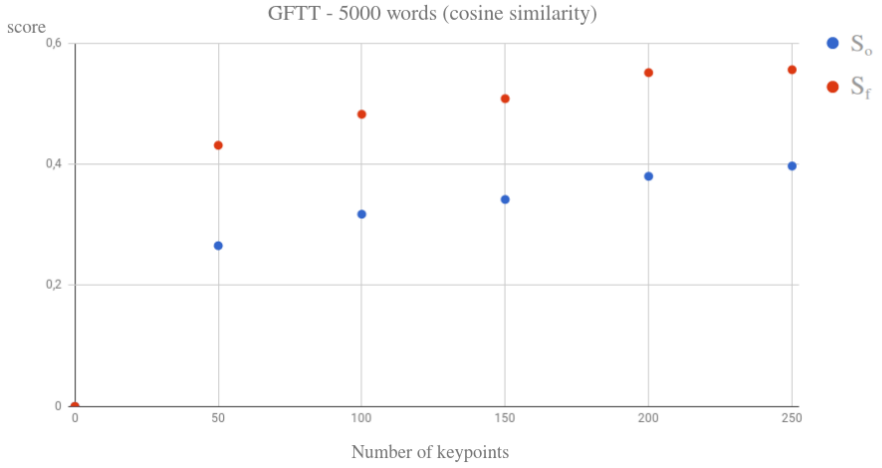


Figure 5.7: Clustering BoW features generated with the GFTT detector using the divisive approach. The scores are obtained by using one vocabulary in combination with different numbers of keypoints.

Table 5.6: Comparing the scores obtained by using a vocabulary of 1000 words combined with 250 and 500 keypoints.

Keypoints	S_o	S_f
250	0.3975	0.5615
500	0.4303	0.5856

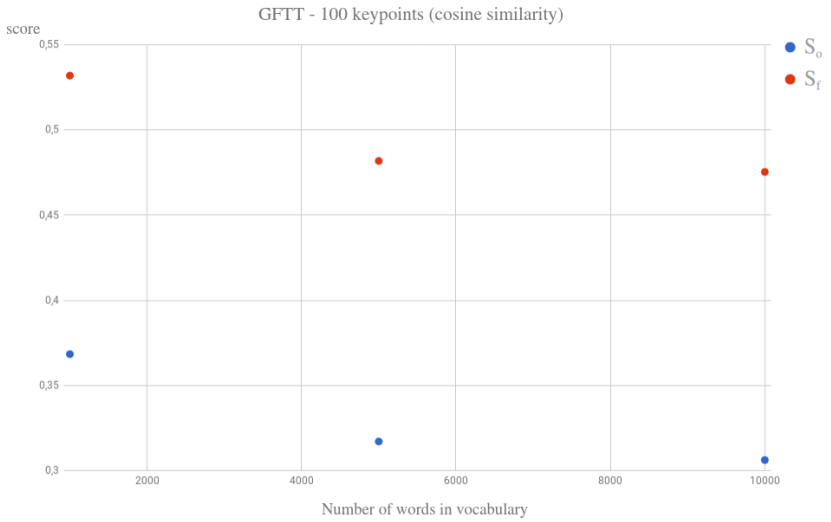


Figure 5.8: Clustering BoW features generated with the GFTT detector using the divisive approach. The scores are obtained by using a set number of keypoints in combination with different sizes of the vocabulary.

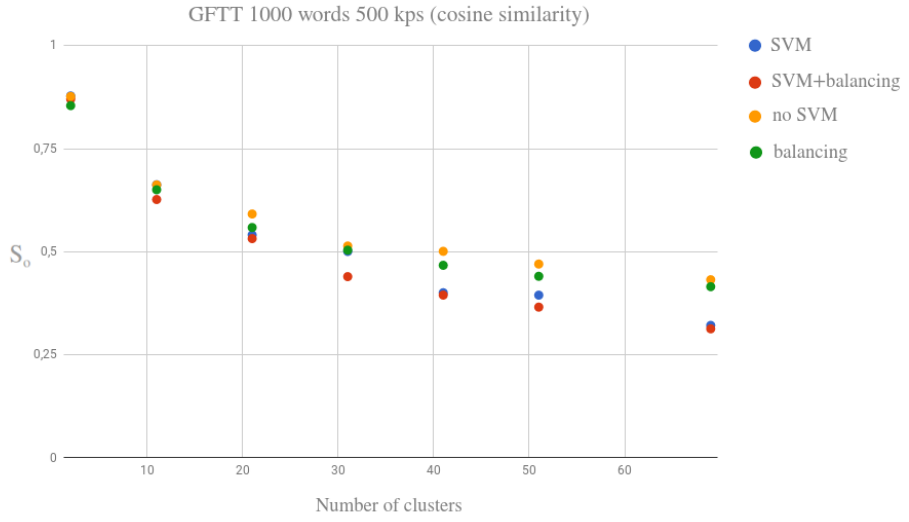


Figure 5.9: Clustering using the divisive approach with cosine similarity on the BoW feature. In this figure the performance for clustering with SVM, and clustering with SVM combined with balancing can be seen. In blue dots are the performance with SVM, in red performance for SVM combined with balancing, in yellow are no SVM, and in green is only balancing.

5.3.3 Without clustering

An experiment which predicts the labels of the verification templates without clustering was performed, details are described in section 5.2.3. The data is manually labeled according to the persons from which the templates are captured, resulting in 111 different labels. Each person has one label and is considered as one cluster. The verification templates are predicted using an SVM classifier and using the centroids of the clusters. The scores obtained from these two methods are shown in table 5.7.

Table 5.7: Scores obtained from two methods for prediction of verification templates without using clustering.

	SVM		**Centroids**	
Feature/Score	S_o	S_f	S_o	S_f
BoW	0.1607	0.1644	0.1180	0.1209

5.3.4 Example of BoW histograms

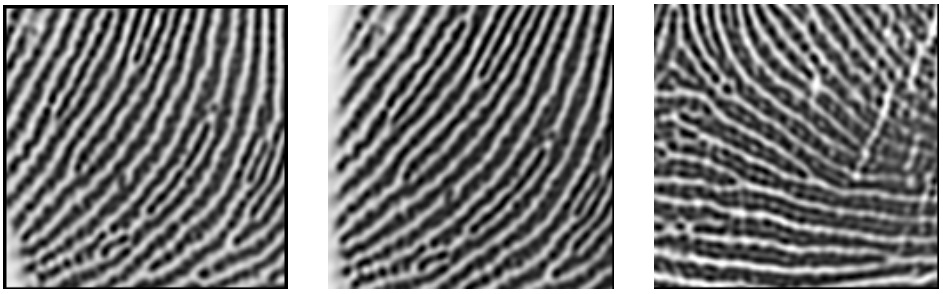
In order to better understand the results obtained when clustering BoW features, some examples of BoW histograms and distances between them is presented. Three images were chosen for this visualization which can be seen in Figure 5.10. Image (a) and (b) is captured from the same finger and person. The overlap between (a) and (b) is 91%. Image (a) and (c) is captured from different persons and are, therefore, not overlapping each other.

Features obtained from two vocabularies of different sizes represent each image. The first feature is obtained from 100 keypoints and a vocabulary containing 10000 words, the resulting BoW histogram is very sparse. The second feature is obtained from 500 keypoints and a vocabulary containing 1000 words, the resulting BoW histogram is denser.

Sparse BoW histograms

Bag-of-words histograms created by finding a small number of keypoints in an image, and using a large vocabulary are sparse. In this example, 100 keypoints and a vocabulary containing 10000 words were used, i.e., at most 100 different words are found. Since each bin represents a word, at most 100 bins are filled with a value.

Figure 5.11 shows the BoW histogram corresponding to image (a). The BoW histogram corresponding to image (b) can be seen in image 5.12. The histogram in



(a) (b) (c)

Figure 5.10: Image (a) and image (b) are from the same finger and their overlap is 91% according to the ground truth. Image (a) and (c) are from different fingers and do not overlap.

Figure 5.13 corresponds to image (c). The mentioned histograms are weighted BoW histograms. Images that overlap each other are expected to have more similar BoW histograms than images that do not overlap.

The similarity between the BoW histograms for image (a) and image (b) and the similarity between the BoW histograms for image (a) and image (b) are calculated using cosine similarity. The similarities are shown in table 5.8.

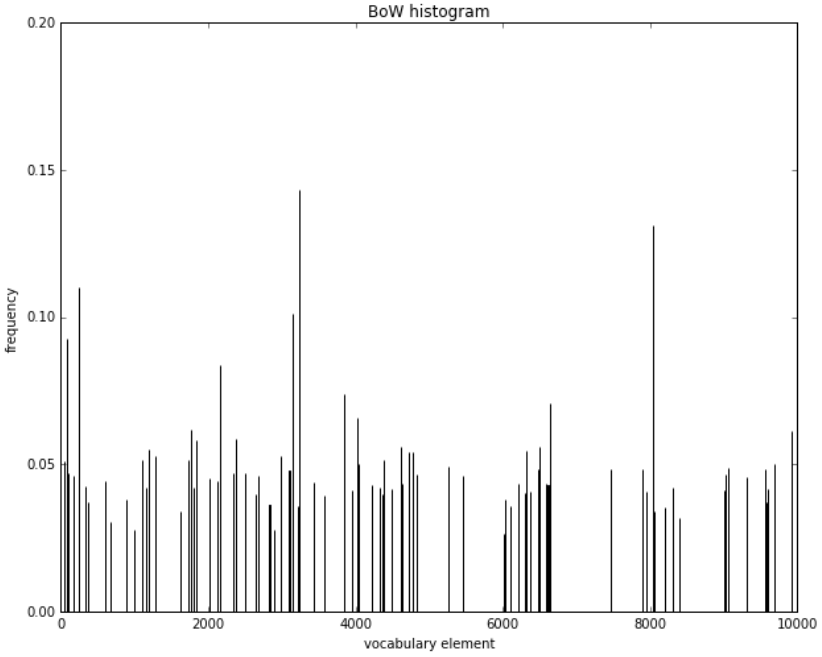


Figure 5.11: BoW histogram for image (a). 100 keypoints and 10000 words.

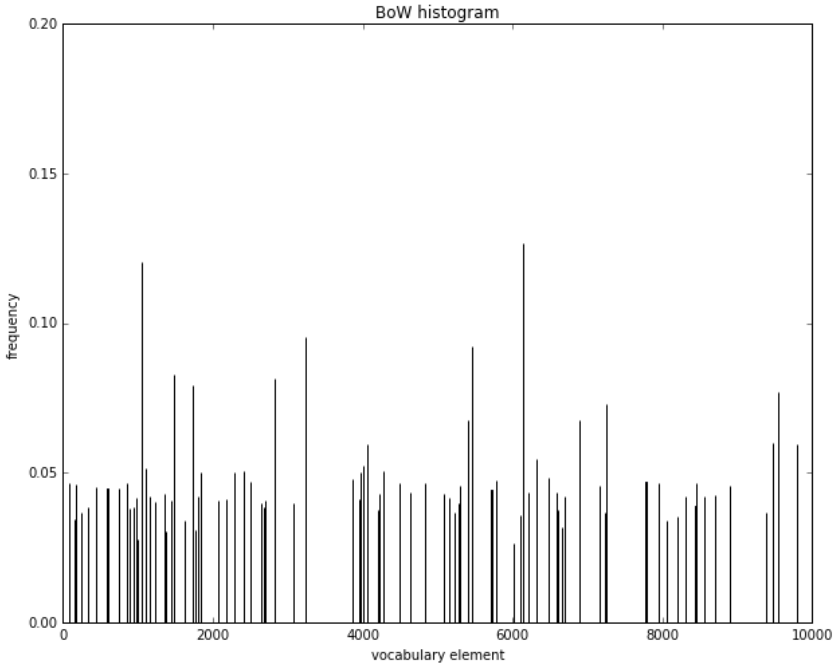


Figure 5.12: BoW histogram for image (b). 100 keypoints and 10000 words.

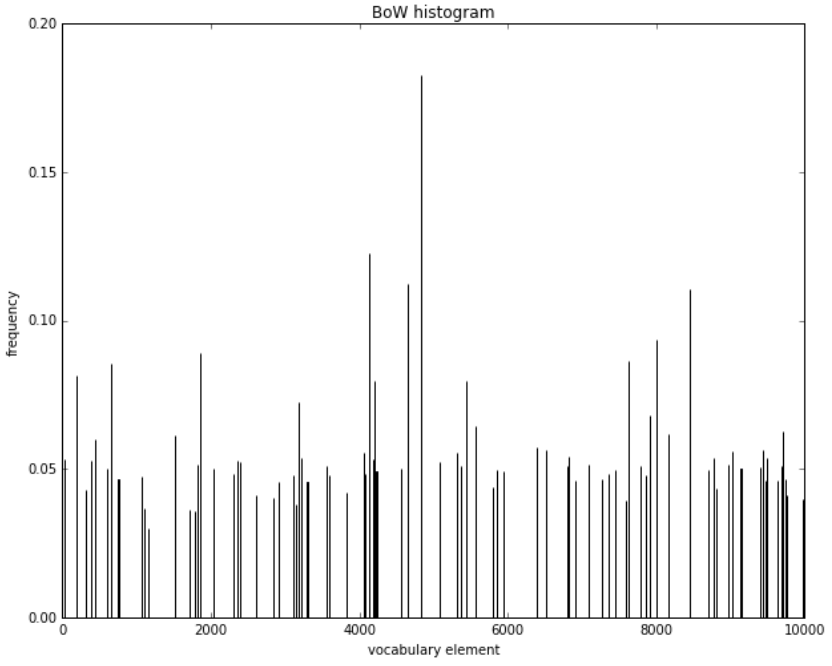


Figure 5.13: BoW histogram for image (c). 100 keypoints and 10000 words.

Table 5.8: Similarities obtained by using cosine similarity, 100 keypoints and a vocabulary containing 1000 words.

Histograms	**Cosine similarity**
(a),(b)	0.7163
(a),(c)	1.0

Dense BoW histograms

Bag-of-words histograms created by finding a large number of keypoints in an image and using a small vocabulary are denser. In this example, 500 keypoints and a vocabulary containing 1000 words were used, i.e., at most 500 different words are found. Since each bin represents a word, at most 500 bins are filled with a value.

Figure 5.14 shows the BoW histogram corresponding to image (a). The BoW histogram corresponding to image (b) can be seen in image 5.15. The mentioned histograms are weighted BoW histograms. The histogram in Figure 5.16 corresponds to image (c).

The similarity between the BoW histograms for image (a) and image (b) and the similarity between the BoW histograms for image (a) and image (b) are calculated using cosine similarity. The similarities are shown in table 5.9.

Table 5.9: Similarities obtained by using cosine similarity, 500 keypoints and a vocabulary containing 1000 words.

Histograms	**Cosine similarity**
(a),(b)	0.2111
(a),(c)	0.9928

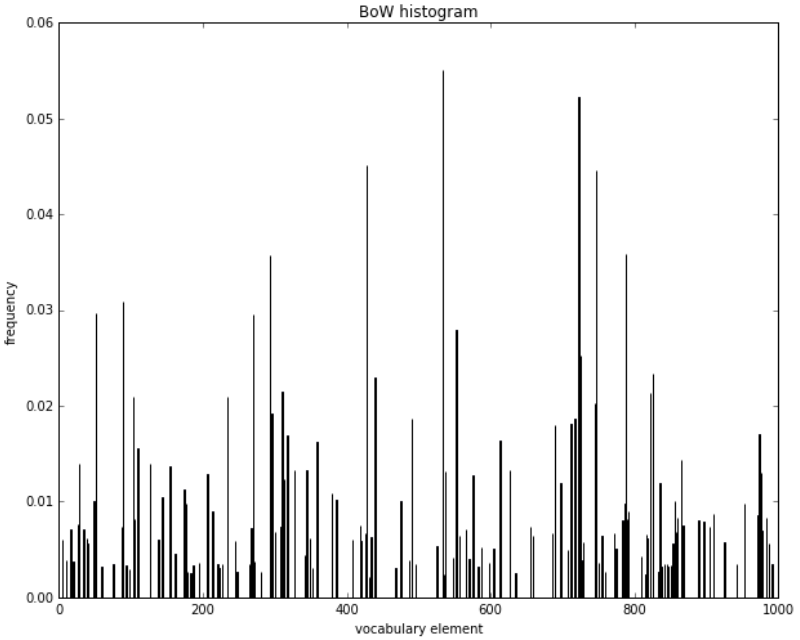


Figure 5.14: BoW histogram for image (a). 500 keypoints and 1000 words.

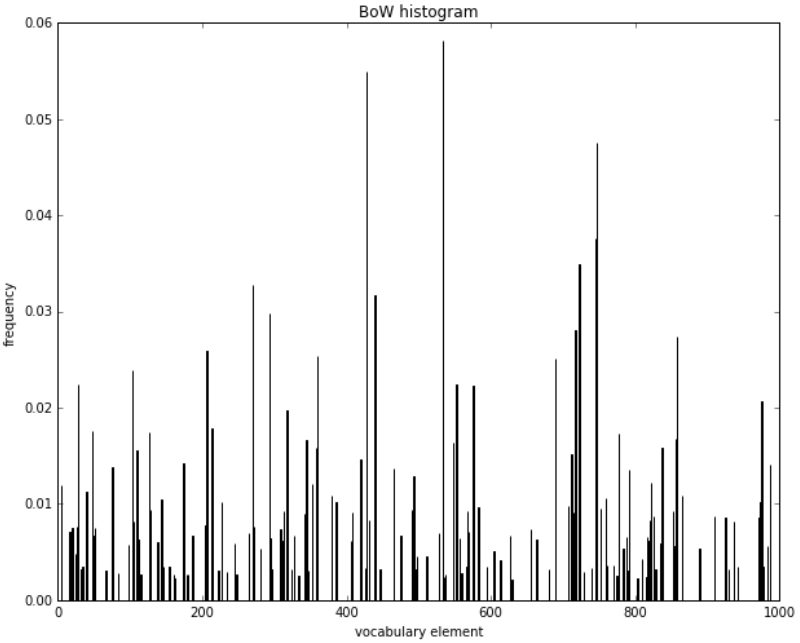


Figure 5.15: BoW histogram for image (b). 500 keypoints and 1000 words.

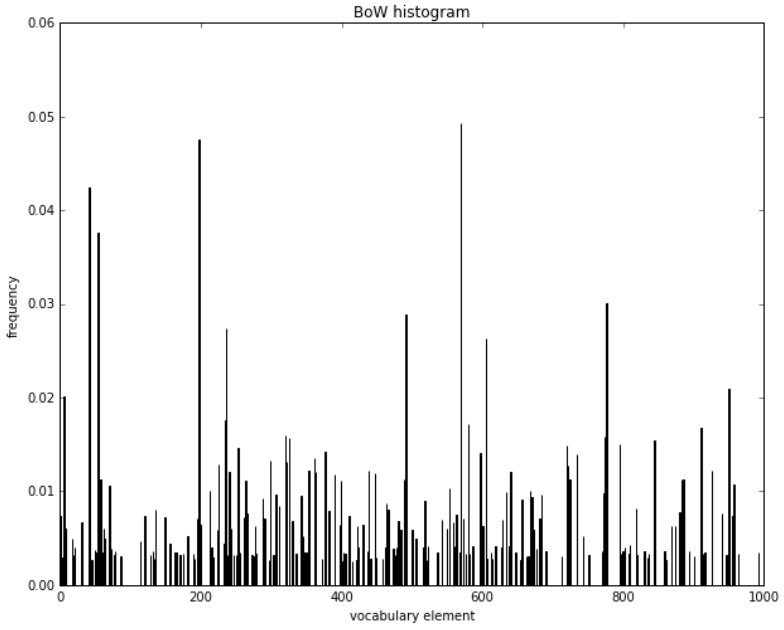


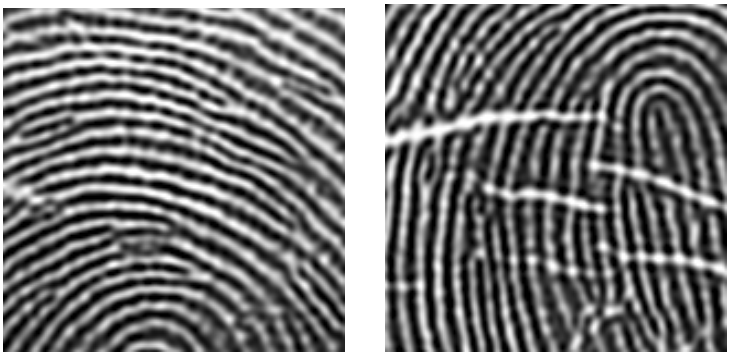
Figure 5.16: BoW histogram for image (c). 500 keypoints and 1000 words.

5.3.5 Finger scores

An example of templates captured from two fingers that often seem to have few correctly predicted verification templates is presented in this section. Both presented fingers have 0 correctly predicted verification templates when using divisive clustering, without SVM and balancing, in combination with cosine similarity.

Figure 5.17 shows an example of an enrollment template and a verification template which are captured from the same finger. Image (a) is the enrollment template which represents the overall appearance of the 16 captured enrollment templates. Image (b) is the verification template which represents the overall appearance of the 10 captured verification templates which are used for prediction. The verification templates show a lot of creases while the enrollment templates do not.

Figure 5.18 shows the overlap ground truth for another finger. The 16 captured enrollment templates overlap the captured verification templates a little. The overlap is in many cases 0%.



(a) (b)

Figure 5.17: Image (a) is an enrollment template and image (b) is a verification template. Both are captured from the same finger.

08-017.png	0	0	0	3	1	0	0	0	0	0	0	0	0	0	86
08-018.png	0	0	0	0	0	0	0	0	0	0	0	0	0	0	74
08-019.png	8	0	4	11	2	0	0	0	0	0	0	0	0	0	14
08-020.png	26	1	13	28	9	0	1	0	0	0	0	2	0	0	16	18
08-021.png	10	12	47	2	0	0	0	2	6	6	17	41	20	34	8	0
08-022.png	17	18	57	9	0	0	2	5	11	12	23	41	20	31	14	0
08-023.png	53	19	16	62	31	1	26	11	9	1	4	3	0	0	43	18
08-024.png	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	33
08-025.png	8	0	3	12	1	0	0	0	0	0	0	0	0	0	1	26
08-026.png	2	0	9	2	0	0	0	0	0	0	0	0	0	2	0	0
08-027.png	0	4	30	0	0	0	0	0	2	4	11	31	31	53	0	0

Figure 5.18: Overlap ground truth for 16 enrollment templates and 10 verification templates captured from the same finger.

Chapter 6

Discussion and future work

In this chapter, the experimental setup and the results are discussed. Future work that may improve the performance is also discussed and a conclusion is presented.

6.1 Discussion

6.1.1 Experimental setup

It should be mentioned that training a model was very time-consuming. Training a model for frequency and orientation features using the EMD distance took days to be done. One can, therefore, imagine how many days or even weeks it took to get the optimal model. A better implementation of the EMD distance would, therefore, be preferred. Models trained with the two other similarity measures were faster. However, it still took half a day to do a full run. To train a vocabulary for the bag-of-words model was also very time-consuming. The time for training a vocabulary increases as the size of the vocabulary and number of detected keypoints increases. Instead of making the vocabulary using the whole database, one could choose fingers that generally represents the whole database. The challenge with a smaller dataset is that fewer keypoints are available when generating the vocabulary, which could affect the possibility of generating a large vocabulary. If a larger number of keypoints were found in each image to avoid this problem, the keypoints get less descriptive. Less descriptive keypoints may generate a vocabulary containing words describing noise.

6.1.2 Results

Visualization of data

In Figure 5.1, the frequency and orientation features have been clustered into one cluster. Thereafter, the distances from that center to all other data points was calculated. The distance can be seen in Figure 5.1. This figure visualizes how the feature points are distributed in space. It shows whether the points are all enclosed inside a globe, or if they are spread out in space. As can be seen in Figure 5.1, the feature points for both frequency and orientation seem to be confined inside a globe. It can be understood that the points are close to each other, which in turns means that clustering could be difficult. Considering the fact that they already seem to belong to one big cluster.

Comparing the range in the x-axis (distance from center) for the two plots, a) and b), one can see that the radius varies from 0 to 800 for orientation, and from 0 to 1400 for frequency. The variation for frequency features is bigger than for orientation, which implies that the orientation features are more centered together inside the globe. Consequently, it means that the orientation feature is harder to separate into different clusters.

The distance from the center to all other points using BoW-features can be seen in Figure 5.2. The cosine similarity measures the cosine angle between two feature vectors instead of the magnitude. That's the reason why the range in the x-axis ranges from 0 to 1. The distance from the center, for BoW-features, are more distributed across the whole range than for orientation and frequency features. This could imply that BoW features potentially are easier to cluster than the two other features.

Frequency and orientation feature

A plot of clustering the frequency feature into 2-69 clusters with different similarity measures can be seen in Figure 5.3. Clustering using the Euclidean distance showed the best performance, even though EMD might be a better method to use considering that the frequency histogram can be seen as a distribution. The EMD compares the similarity between two histograms, by how much work must be done to transform one of the distribution to the other one. The Euclidean distance, on the other hand, compares the bins of two histograms and computes the Euclidean norm between the histograms. With that said, the EMD method is a better method for these types of histograms. However, the Euclidean distance was chosen in the proceeding analysis instead of the EMD, as it had better performance and was computationally faster.

The comparison between the orientation and the frequency feature is seen in Figure 5.4. The features have been clustered using EMD similarity measure. The performance for the frequency feature is better than the orientation feature. Out of the three similarity measures compared in Figure 5.3, EMD was the one with the lowest performance measure. Orientation features clustered using EMD was even worse. Hence, in the proceeding analysis, the orientation feature won't be considered.

Even in the divisive clustering, where cluster sizes are approximately equal, the Euclidean distance performed the best for the frequency feature, see Figure 5.5. Divisive clustering is applied as the cluster sizes may vary a lot when direct clustering is performed. The performance scores are therefore not trustworthy and can be misleading since clusters with a bigger size give better scores as many fingerprints can be clustered together. To get a better foundation for decision making, divisive clustering must be done to get even clusters.

The same as in the case of direct clustering, divisive clustering using EMD has a performance score below the performance of the two other similarity measures. In the same figure, the performance for the orientation feature is still bad compared to the frequency feature. Thus, this proves, again, that Euclidean distance combined with frequency features is a good combination for proceeding experiments.

When a combination of feature and similarity measure has been chosen, further improvements are desired. In this project, SVM and SVM combined with balancing are two improvements that are tested. The result can be seen in Figure 5.6. As can be seen, using the two methods mentioned above, no improvements are made. The performance is the same for all three methods. SVM is applied to improve the verification stage when a verification template needs to be labeled. The SVM finds the separating hyperplane between the different classes, when new data need to be labeled, all the learned classifiers votes. The new data is assigned to the class with most votes. Compared to predicting a new label using the centroid-based method, where new data is assigned to the cluster with the smallest similarity. Balancing, on the other hand, is applied to get approximately equal cluster sizes for all clusters. To conclude the above, in order to further improve the performance; SVM and balancing are applied. However, for the frequency feature, there are no improvements.

The frequency feature has been the best feature choice so far. This is consistent with the result from the visualization in section 5.1, where the orientation feature was less probable of being sufficiently good clustered than the frequency feature. The suitable combination of a method to use during clustering is; clustering with the divisive approach using Euclidean distance, without SVM, and without balancing. This combination showed the best performance, and the scores for this combination is $S_o = 0.3406$ and $S_f = 0.5085$, Table 5.2. Comparing these scores with the scores

obtained from direct clustering, $S_o = 0.3624$ and $S_f = 0.5275$, Table 5.1, one can see that the scores from direct clustering are slightly better than those for divisive. As mentioned, this can be due to the fact that the cluster sizes are uneven in direct clustering. The scores are thus not trustworthy.

The scores from divisive clustering can be compared with the scores without any clustering, Table 5.3. The scores with clustering are much better than without clustering, which is good since it shows that improvements are made during clustering. Considering that fingers from the same person vary as much as fingers from different persons, these results are satisfying. Thus, labeling the fingerprints according to a person is not the best approach, and clustering can be considered as a better method for labeling.

BoW feature

To decide whether to use the SIFT or the GFTT detector to detect keypoints in an image, comparisons were made by clustering the BoW-features into two clusters and comparing the scores, see Table 5.4. In this table, the features were clustered using the Euclidean distance. The GFTT without weighting applied has the best scores. However, the sizes of the two clusters varied incredibly and should therefore not be considered. In this case, it is hard to make a decision as the two scores S_o and S_f vary slightly between the three other options. The S_o score for GFTT is better than for SIFT's S_o score. However, the S_f score for SIFT with weighting is slightly better than the S_f score for GFTT with weighting.

Weighting the BoW-features seem to be a better approach as it gives even cluster sizes. Thus, only weighted features will be considered in the sequential analysis. In Table 5.5, the scores from clustering the features using cosine similarity can be seen for both GFTT and SIFT. The scores for GFTT are higher than for SIFT. These scores are also better than all the scores in Table 5.4. Since weighted feature using GFTT detector clustered with cosine similarity seem to have the best scores, this combination will be considered in the proceeding analysis.

The main challenge of generating BoW-features is to find a suitable combination of the number of keypoints and the size of the vocabulary since these two variables seem to be correlated. For this project, analysis using vocabularies of sizes 1000, 5000 and 10000 combined with a number of keypoints equal to 50, 100, 150, 200 and 250 is performed. To find the appropriate combination of the number of keypoints and size of the vocabulary, one of these variables must be fixed and the other changed during testing. To examine how many keypoints that are going to be used, clustering using a vocabulary of 5000 words and different numbers of keypoints was evaluated, see

Figure 5.7. A vocabulary of size 5000 was chosen as 5000 is between 1000 and 10000. The results in this figure show that the score increases as the number of keypoints increases.

To evaluate the relation between the number of keypoints and the size of the vocabulary, analysis on different sizes of the vocabulary combined with 100 keypoints was also performed. The result is found in Figure 5.8. In this case, the score is decreasing as the size of the vocabulary is increasing. From the two figures, 5.7 and 5.8, it can be concluded that a large number of keypoints combined with a small size of the vocabulary is desired.

A divisive clustering, with a vocabulary size of 1000 words and the number of detected keypoints equal to 250, was performed. This combination was based on the analysis mentioned above. A clustering with 500 detected key points in each image and a vocabulary of 1000 words was evaluated, to investigate whether an even higher number of key points would yield higher scores. The result can be seen in Table 5.6. Detecting a higher number of keypoints shows better performance. Hence, a vocabulary of size 1000 combined with 500 keypoints will be used in the proceeding experiments.

When a chosen model has been decided, further improvements on the performance score for this model is desired. As mention in section 6.1.2, the improvement methods used in this project are SVM and SVM combined with balancing. In figure 5.9, the performance scores for the two methods can be seen. The performance of these methods are generally equal, in some cases, the SVM has a higher score. However, the best scores obtained when the data is divided into 68 clusters, are still the scores where no SVM and no balancing is applied, Table 5.6.

The explanation of why a small size of the vocabulary combined with a big number of keypoints is better, is shown in section 5.3.4. In this section, histogram representations of images that have big overlap and no overlap are considered. In Figure 5.10, the images (a) and (b) are from the same finger, i.e., large overlap. Image (c) is taken from a different finger than (a) and (b), i.e., no overlap. The BoW histogram representation of the images of the same finger, using a vocabulary of size 10000 combined with 100 keypoints, differ from each other even though they are from the same finger. If it was not known that these histograms, Figure 5.11 and Figure 5.12, are from the same finger, one would think that they are from different fingers since the histograms looks different. The difference in appearance between histogram 5.11 and 5.12, and between histogram 5.11 and 5.13, seem to be equal. In other words, they are equally different from each other. The similarity measures between these three images (a) and (b) (same finger), and (a) and (c) shows that (a) and (b) are almost equally different from each other as (a) and (c). The similarity

measure indicates that (a) and (b) are not from the same finger, and has a small overlap which is not true.

Representing the three images with a smaller vocabulary combined with a large number of keypoints seem to be suitable. The histogram representation of the three images can be seen in the figures 5.14, 5.15 and 5.16. Comparing histogram 5.14 with 5.15, this pair is more similar than the histogram pair 5.11 and 5.12. The bin with maximum frequency is equal for both histograms, (5.14 ,5.15). One could tell that these two histograms have a big overlap, without knowing in advance that they are from the same finger. This could not be guessed in the case with a bigger vocabulary and a smaller number of keypoints. The histogram representation of (a) and (c) are still very different from each other, which is good as they are from different fingers with no overlap. The similarity measure, Table 5.9, between the histograms from the same finger is small. It indicates that the images have a big overlap, as they are more similar to each other. The similarity measure between (a) and (c) is big, it indicates that they have small or no overlap.

It can be concluded that a histogram representation of the images using a small vocabulary and a large number of detected keypoints is desired. As it represents the images better, i.e., images with big overlap are more identical to each other. This, in turn, means better clustering, since similar histograms can be clustered together. This is consistent with the result in Table 5.6, which shows that a small sized vocabulary combined with a large number of keypoints, yields better performance results.

For the purpose of understanding why the scores are bad, the finger scores was evaluated, see section 5.3.5. Some fingerprints seem to have few or none correctly predicted verification templates. The finger scores were therefore evaluated to find these hard cases, and to examine why they are hard to cluster. In Figure 5.17, there are an example of an enrollment template and a verification template. The fingerprints from this person was hard to cluster, and in turn gave bad scores, as the enrollment templates and the verification templates had different appearances. The enrollment templates from this persons finger consisted of templates without scars. However, most of the verification templates had scares. The enrollment templates will therefore be clustered in a cluster without scares, whereas the verification templates will most likely be clustered together with fingers that have scares.

The second hard case is when the verification templates have small or no overlap with the enrollment templates, see Figure 5.18. The ground truth shows that, overall, the verification templates have zero overlap with the enrolled templates. If there is no overlap, the same keypoints/features can not be found. The clustering will, again, be bad since the enrollment and verification templates are going to be clustered into

different clusters.

If the database consists of many of these hard cases, the overall score will be affected negatively. If the verification templates are incorrectly predicted, it will result in a low score. For many incorrectly clustered verification templates the overall score will be low.

K-centroid

The drawback of the K-centroid clustering method is that every time it converges, it converges to a local minimum rather than a global minimum. The best clustering can, therefore, not be ensured. This fact needs to be taken into consideration when making decisions on which feature representation that is going to be used, and when comparing the scores from different methods. It could be solved by doing a number of clusterings and choosing the model with the highest performance. This approach was desired, but considering how much time it takes to train one model it could not be done practically.

6.2 Future work

6.2.1 Keypoint detectors and descriptors

Just two different interest point detectors and one descriptor were tested in this thesis. The choice of detectors and descriptor was made based on a master's thesis performed at Precise Biometrics AB in 2016 [17]. It was shown that the combination of the chosen detectors and descriptor made good performance when matching two fingerprint images. Precise Biometrics AB has since then moved on to other detectors. There exist numerous algorithms for interest point detection and for region description. Extensive testing of detectors and descriptors in order to find the combination that fits our problem the best would be interesting. Finding such a combination would improve the performance.

6.2.2 Different method for bag-of-words vocabulary

The bag-of-words method used in this thesis is from OpenCV, which uses a k-means clustering with Euclidean distance to generate the vocabulary. As mentioned, there are different similarity measures that could be used. It would, therefore, be

interesting to try different similarity measures when creating the vocabulary using the k-means method.

Generating the vocabulary with K-means clustering is the most common method. However, there are other methods one could use to generate the vocabulary. Neural networks can, for example, be used to create the vocabulary. If time was not a problem, this approach could be tested to see if creating a vocabulary with a different method affects the clustering performance.

6.2.3 Bigger and smaller images

The images used in this thesis are of the size 192-by-192. It would be worth the effort to try the proposed method on bigger and smaller images, to examine whether the images have too much information or too little. With too little information in the image, the vocabulary won't be rich in words, which in turns leads to a poor representation of each image. Smaller images also mean less overlap. The probability of two smaller images, from the same finger, overlapping each other would be small. However, with smaller images, one feature could be captured instead of many.

If the images were bigger, for example, if the image captured the whole fingerprint, they would contain a lot of information. The vocabulary would be richer in words, and the overlap between two images from the same finger would be big. The probability of finding the same keypoints from two images of the same finger is then high, and the histogram representation would be more describing. Computing the similarity measure between such histograms results in a small value.

6.2.4 Similarity measures

The similarity measures used in this project are EMD, Euclidean distance, and cosine similarity. EMD is a good similarity measure for distributions such as the frequency and orientation histograms. The Euclidean and Cosine similarity measure is better for comparing the bins in the BoW histograms. Cosine similarity has been used for text classification, and it performed well for image classification too. To further improve the clustering, it would be worth to try different similarity measures than the mentioned measures in this thesis. The Jaccard similarity, for example, has also been widely used for document classification. It would be interesting to know if this measure gives better clustering in terms of performance than the other similarity measures.

6.3 Conclusion

The main objective of this thesis was to investigate whether it was possible to cluster the database provided by Precise Biometrics. The goal was to cluster the database into smaller subsets, such that a search through the entire database could be avoided during verification. The investigation in this project has resulted in a divisive K-centroid model. The feature with the highest score for this model is a weighted BoW histogram representation of the fingerprints. The histograms represent the fingerprints in the best way when detection of 500 keypoints in each image is used, and when the vocabulary consists of 1000 words. Out of the three mentioned similarity measures, Cosine similarity was the best measure to use for BoW features.

The performance of the final chosen model is not good enough to be implemented in a product. The performance of the best model was $S_o = 0.4303$ and $S_f = 0.5856$. The clustering method is, however, better than labeling the fingerprints according to the persons from which the fingerprints were captured. This gave a performance score (SVM) $S_o = 0.1607$ and $S_f = 0.1644$. The BoW-feature representation of the images showed the most promising results in the clustering, and could for sure be improved to reach the desired performance result.

Bibliography

[1] Davide Maltoni, Dario Maio, Anil K. Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Buisness Media, London, 2009.

[2] Belen Fernandez-Saavedra, Raul Sanchez-Reillo, Rodrigo Ros-Gomez, and Judith Liu-Jimenez. Small fingerprint scanners used in mobile devices: the impact on biometric performance. *IET Biometrics*, 5(1):28–36, 2016.

[3] Jayant V. Kulkarni, Bhushan D. Patil, and Raghunath S. Holambe. Orientation feature for fingerprint matching. *Pattern Recognition*, 39(8):1551 – 1554, 2006.

[4] Joe Minichino and Joseph Howse. *Learning OpenCV 3 and Computer Vision with Python*. Packt Publishing Ltd, Birmingham, 2015.

[5] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513 – 523, 1988.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[7] Jianbo Shi and Carlo Tomasi. Good features to track. *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, 1994.

[8] Chris Harris and Mike Stephens. A combined corner and edge detector. *In Alvey vision conference*, 15:50, 1988.

[9] Reinhard Klette. *Concise Computer Vision - A Introduction into Theory and Algorithms*. Springer, London, 2014.

[10] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data - An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2009.

[11] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer Science & Buisness Media, Singapore, 2006.

[12] Thanh-Nghi Do, Philippe Lenca, and Stéphane Lallich. Classifying many-class high-dimensional fingerprint datasets using random forest of oblique decision trees. *Vietnam J Comput Sci*, 2(1):3–12, 2015.

[13] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. *IEEE International Conference on Computer Vision*, pages 59–66, 1998.

[14] V. Kecman. Support vector machines - an introduction. *Support Vector Machines: Theory and Applications*, pages 1–47, 2005.

[15] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: Concepts and techniques*. Morgan Kaufmann, Waltham USA, third edition, 2012.

[16] M.N. Murty and Rashmi Raghava. *Support Vector Machines and Preceptrons: Learning, Optimization, Classification, and Application to Social Networks*. Springer, Switzerland, 2016.

[17] Johan Hagel and Alexander Karlsson. Fingerprint matching - hard cases. Master's thesis, Lunds Tekniska Högskola, Sweden, 2016.