Block-wise Finger Image Quality Assessment Based on Machine Learning

Ivan Danov

Kongens Lyngby 2013
IMM-M.Sc.-2013-111

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2013-111

Summary

The development of fingerprint quality assessment algorithms has seen increasing popularity throughout the recent years, since the accuracy of the fingerprint biometric systems is heavily dependent on the quality of the acquired samples. There exist a variety of image analysis methods for quality estimation, however they are usually characterized with high computational complexity. Block-wise finger image quality assessment based on Self-Organizing Maps (SOM) is a novel approach that has shown promising results in terms of speed and performance.

The goal of the thesis is to conduct experiments by training SOM networks with a large dataset of raw fingerprint image blocks in order to extract quality features. These features are to be interpreted by another machine learning model trained to predict the quality score of the fingerprint image. In this thesis two datasets with two block sizes each are used for training linearly initialized SOM networks.

The results from the trainings are analyzed and the extracted quality features are used as an input for the training of four different machine learning models. These four machine learning models are used to predict the quality scores of the fingerprint images and are based on three machine learning techniques: Self-Organizing Maps, Generative Topographic Mapping and Random Forests. The performance of the models is comparatively evaluated with two state of the art approaches using ERC curves and Spearman's Correlation matrices.

Some of the proposed methods show improvement of the results obtained by previous work in the field and can possibly take part of the ISO/IEC or NIST Finger Image Quality 2.0 standards.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark during an exchange visit at the Center for Advanced Security Research Darmstadt (CASED) in fulfilment of the requirements for acquiring an M.Sc. in Computer Science and Engineering.

The thesis deals with block-wise fingerprint image quality assesment techniques based on applying a set of machine learning methods.

The thesis consists of introduction, theoretical background chapters, state of the art, previous work, proposed methods, experimental setup and results, conclusions, direction for future works and a couple of appendices.

Lyngby, 29-September-2013

Ivan Danov

Acknowledgements

I would like to thank Prof. Dr. Christoph Busch for inviting me to do my thesis at the Center for Advanced Security Research Darmstadt (CASED). I am very grateful for inspiring and motivating me all the way through my stay there. I would like to thank Prof. Rasmus Larsen for supervising my project at the Technical University of Denmark.

I want to express my deepest gratitude to MSc. Martin Aastrup Olsen for all the meetings, discussions and support throughout his supervision on my thesis progress. His precious suggestions and ideas have encouraged me to keep on working energized and overcome the emerging problems far earlier before they become apparent.

Special thanks to my colleagues and friends Jinghua Wang, Elakkiya Ellavarason, Marek Dusio and Vikas Gupta for being cheerful office mates and good buddies. I cannot image my stay in Darmstadt without any one of them.

Last but not least, I want to thank my girlfriend and my family for encouraging and supporting me throughout my MSc studies. During these two years they have always been there for me, no matter what, so they deserve to be acknowledged.

Contents

Chapter 1

Introduction

As more and more people get involved in the expanding digital life and new technologies arise easing the work of both citizens and governments, *biometrics* gain increasing popularity. For a single person accessing many security contexts it becomes harder to manage all the tokens (keys, ID cards, RFID tokens, etc.) she owns and to remember all the passwords and PIN codes required for each context by itself. Moreover different security contexts require different levels of security, so ease of use is not the only concern. Since biometrics offer both flexible security settings and ease of use, they seem to be a logical solution to most of the faced *identification* and *verification* problems.

There are many different *biometric characteristics* and *fingerprints* are probably the most popular of them all. Fingerprints have been used extensively throughout the history of humankind, especially for the last couple of hundred years. Due to their uniqueness and ease of collecting, they have been used for authentication, identification and verification purposes as well as in *forensics*. Their long history of usage has proved them as a preferred biometric characteristic used in medium to high security contexts.

Fingerprints are used by private, corporate and governmental institutions. Some of the largest-scale examples of *fingerprint recognition biometric systems* are the *Customs and Border Protection Management System of USA* [oHS] and the *Schengen Area's Visa Information System* [Com]. Another example is the

Unique Identification Authority of India, which is collecting the fingerprints for more than 1.2 billion citizens for issuing their *Unique Identification Numbers* [oI].

1.1 Motivation

During the years fingerprint biometric systems have evolved and have reached maturity. *Comparison subsystems* offering sufficient accuracy have been developed by researchers and corporations. Nowadays the error levels of these subsystems are mainly stemming from the quality of the *probe* and *reference* samples. Therefore recently researchers have raised their interest in suppressing the error levels by developing *quality measurement algorithms* and implement them as a subsystem of the *Automated Fingerprint Identification Systems (AFIS)*. As a result many different fingerprint quality measurement algorithms were proposed. In August 2004 a standardized tool has been proposed by *National Institute of Standards and Technology of the U.S. Department of Commerce (NIST)* and it was named *NIST Fingerprint Image Quality (NFIQ)* [TWW04]. A new version of the NFIQ (NFIQ2.0) is currently under development [NIS12a] by NIST and the *International Organization of Standardization (ISO)*, based on ISO/IEC 29794-1:2009 [ISO12b].

Most of the already proposed methods are image analysis based. However, some recent researches [Mak12] [OTMB13] have shown that applying machine learning methods, in particular *Self-Organizing Maps*, could be used to extract quality features from the raw image data and have shown promising results. Therefore a further research in the field is welcome and different approaches must be tried out in order to improve the performance of the considered methods.

1.2 Goals of the research

The research goals of the thesis are:

- Repeat the research of block-wise fingerprint image quality assessment using machine learning methods [OTMB13]
- Investigate whether the block-wise fingerprint image modeling could be improved

- Investigate different feature interpretation methods of obtaining fingerprint image quality
- Compare all considered approaches in terms of performance and speed

1.3 Thesis overview

This thesis consists of eleven chapters, that could be split into four parts. Firstly, chapters from 1 to 4 could be considered as an introduction to the topic and the theoretical background needed for the methods used in the thesis. Secondly, chapters 5 and 6 show state of the art and current developments in the field. The third part consists of chapters 7 to 9, which present the approaches proposed in this thesis and the performed experiments throughout the course of the research. The last chapters, forming the fourth and last part, are dedicated to conclusions and some directions for future work.

Chapter 2

Biometrics

This chapter contains an overview of the field of biometrics for the unfamiliar reader. Basic concepts of biometrics like *biometric recognition*, *biometric systems*, *biometric performance* and *biometric errors* are discussed. The definitions and the terms used are synchronized with the ISO/IEC 2382-37:2012 [ISO12a] vocabulary, developed by *International Organization for Standardization/International Electrotechnical Commission (ISO/IEC)*. The provided terminology in this chapter is to be used by the following chapters as well.

2.1 Biometric recognition

Biometric recognition refers to the use of distinctive anatomical and behavioral characteristics for automatically recognizing individuals. The word *biometrics* comes from the Greek words *bios* (life) and *metron* (measurement) depicting the process of measuring of living human body. The *biometric characteristics* are also called *traits* or *biometric identifiers* and should satisfy specific requirements, if to be used as biometric identifiers:

- ***Universality***: each person should possess the trait

- ***Distinctiveness***: any pair of persons should possess sufficiently different traits
- ***Permanence***: the biometric characteristic should be invariant over time
- ***Collectability***: the trait should be available for measuring quantitatively
- ***Performance***: recognition accuracy, speed, resource requirements, robustness to operational and environmental factors should be acceptable
- ***Acceptability***: the measurement and collection of the trait should be positively perceived amongst the subjects
- ***Circumvention***: it should be hard for fraudulent activities to achieve circumvention of the biometric system

Biometric identifier	Universality	Distinctiveness	Permanence	Collectability	Performance	Acceptability	Circumvention
Face	H	L	M	H	L	H	H
Fingerprint	M	H	H	M	H	M	M
Hand geometry	M	M	M	H	M	M	M
Hand/finger vein	M	M	M	M	M	M	L
Iris	H	H	H	M	H	L	L
Signature	L	L	L	H	L	H	H
Voice	M	L	L	M	L	H	H

Table 2.1: Comparison of most popular biometric traits. Level of satisfying the requirements are denoted (H)igh, (M)edium and (L)ow. Taken from [MMJP09]

2.2 Biometric systems

Biometric system is a system for the purpose of automated recognition of individuals based on their behavioral and biological traits [ISO11]. The recognition could be either *verification* or *identification*. The term *authentication* from the information technology language has the meaning to let the system know the identity of the user, regardless of *verification* or *identification* mode has been used. However, authentication is used sometimes in the field of biometrics as a

synonym for verification. Throughout this thesis the more generic term *recognition* will be used.

- A ***verification system*** authenticates the identity of a person by capturing a *biometric characteristic* and comparing it with a previously captured and stored *biometric reference template*. After *one-to-one* comparison, a verification system rejects or accepts the submitted identity claim.
- An ***identification system*** recognizes the identity of a person by *one-to-many* comparisons with the entire *enrollment template database*. The result of an identification system is either an established identity of the subject or a response that the subject is not enrolled in the system database.

2.2.1 Biometric system processes

A *biometric system* is involved in three main functional processes, namely *enrollment*, *verification*, and *identification*. A *verification system* uses only the first two processes, where *identification system* uses enrollment and identification processes. The three processes are more precisely defined as follows:

- ***Enrollment process***: this process is responsible for registering an individual who is a subject of the *enrollment*. The biometric characteristics of the subject are captured by a *biometric scanner* producing a *sample* that is usually itself a subject of a quality check. Later on, features of the sample are extracted and enrolled as a *reference* in the *enrollment database* with the *identity reference*.
- ***Identification process***: the biometric characteristics of the subject are captured and the extracted biometric features are searched against the *enrollment database* to produce a *candidate list*. That list consists of identities of individuals, whose references match the features extracted from the captured sample, and could be an *empty list* if none match.
- ***Verification process***: the subject presents an *identity reference* for a claim of identity along with her *biometric characteristics* to the capturing device. The acquired *biometric sample* is used for extracting the features and comparing them with the *biometric reference* linked to the claimed identity.

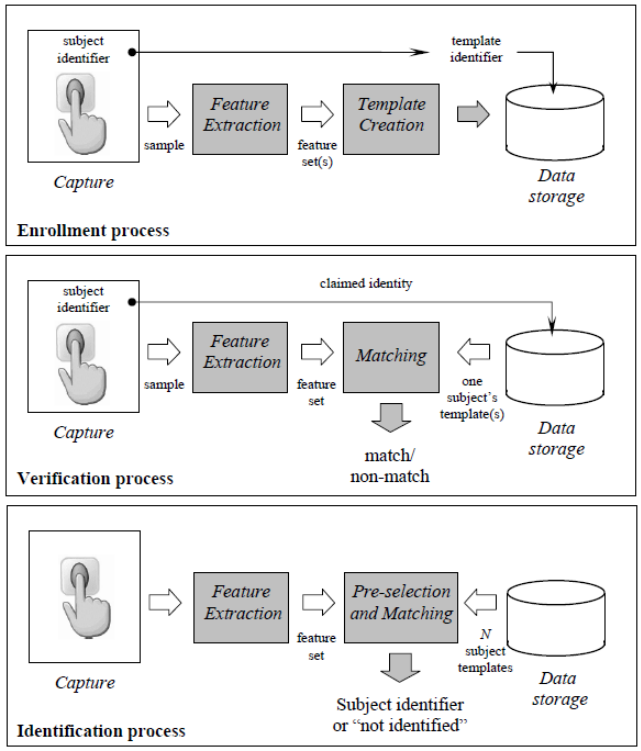


Figure 2.1: Enrollment, identification and verification processes. Taken from [MMJP09].

2.2.2 Biometric system components

Biometric systems consists of *subsystems*, that have certain roles in the *biometric system processes*. Figure 2.2 shows the interactions between the following technical and functional blocks:

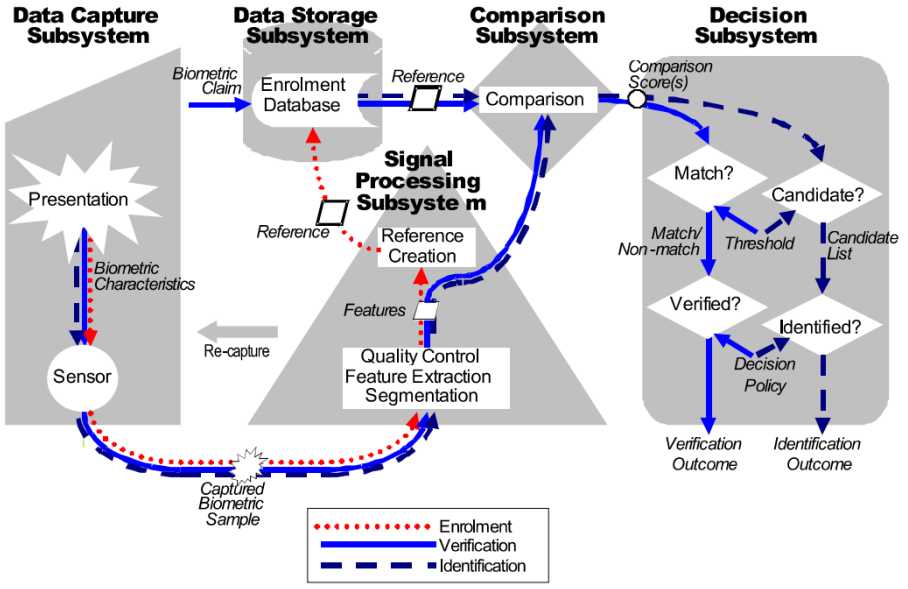


Figure 2.2: Components of a general biometric system. Taken from [ISO11].

- ***Data capture subsystem***: this subsystem consists of *biometric capture devices* and *sensors* to collect *biometric characteristics*. The result of this subsystem is a *biometric sample* such as a *fingerprint image*, *iris image*, *facial image*, *voice recording*, *gait sequential data*, etc.

- ***Signal processing subsystem***: this subsystem extracts *features* from the *biometric sample* as numbers or labels, so that the resulting *biometric features* can be compared with those extracted from other biometric samples. Features extracted during the *enrollment process* are stored in the *data storage subsystem* as a *biometric reference*.

- ***Data storage subsystem***: this component of the biometric system stores the references in the *enrollment database* usually along with *identities* linked to the *references*. The reference can be stored as a sample, or features, or even as both of them. This storage of references makes possible subsequent *identification* or *verification* of an identity.

- ***Comparison subsystem***: this subsystem performs the comparison between the *captured biometric samples* (or the extracted features) and the *stored biometric references*. In the case of *verification*, a *one-to-one* comparison is performed for a *claimed identity*. In the case of *identification*, a *one-to-many* comparison is performed of the *captured biometric sample* (or its extracted features) with the *stored references* in the database. The output of that subsystem is a *comparison score* for each comparison that has been done.
- ***Decision subsystem***: this subsystem makes a decision, based on the *produced comparison scores* by the comparison subsystem, whether the captured biometric sample and the reference(s) used for comparison have the same *biometric source*. For the *verification process*, an identity claim is accepted or rejected according to the *comparison score*. For the *identification process*, a list of candidate identities (possibly an empty one) is returned. A decision subsystem has a *decision policy*, that is to be followed for making the decisions.

2.3 System errors

Every system is prone to errors, so are biometric systems. There are several types of errors that one should be aware of and their short overview is presented in this section. More comprehensive definitions could be found in ISO/IEC 19795 and more specifically in ISO/IEC 19795-2 (2007) [ISO07].

2.3.1 Failure To Acquire (FTA)

FTA error is caused by failures in the data capture subsystem and the signal processing subsystem. A high rate of FTA affects the throughput of the biometric system and cause user frustration due to the need of reacquiring the biometric trait. This error is resulted by the combination of the following errors:

- ***Failure To Detect (FTD)***: this error occurs when the *biometric scanner* used by the data capture subsystem fails to detect the presence of the *biometric source*.
- ***Failure To Capture (FTC)***: this error occurs when the *biometric scanner* detects the presence of the *biometric source*, but fails to capture a sample. This failure occurs when the *biometric sample* is of poor quality or due to misuse of the data capture subsystem.

- ***Failure To Process (FTP)***: this error occurs when the *captured biometric sample* is sent to the signal processing subsystem, which fails to extract usable *feature set*. Captured biometric samples of low quality usually cause this error.

2.3.2 Failure To Enroll (FTE)

Failure To Enroll (FTE) error occurs when the extracted features from the biometric sample are either not enough discriminatory or too noisy, so creating a template and storing it to the data storage system fails. If this error is not considered and ignored, this will result in increase of the recognition error rate.

2.3.3 Recognition errors

Given two samples the comparison subsystem outputs a *comparison score*, which is later on used by the decision subsystem. If the two given samples are from the same biometric source, they are called *mated* samples H_1 and their *comparison score* is known as *genuine comparison score*. If they are from different biometric sources, they are *non-mated* samples H_0 and they introduce *impostor comparison score*. All impostor scores from a given dataset give a rise to an *impostor distribution* $p(s|H_0)$ and genuine scores give rise to a *genuine distribution* $p(s|H_1)$. Considering that comparison scores are usually *similarity scores*, impostor comparison scores are expected to be lower than genuine scores. Therefore a *threshold* t is defined to enable the decision subsystem to conclude on match D_1 or non-match D_0 decision for each queried pair of samples.

Ideally, all impostor scores should be lower than the genuine scores, so defining a threshold for the decision subsystem would be a trivial task. However, in reality there is an overlap between genuine and impostor distributions, as shown in figure 2.3. This overlap introduces two types of errors regardless of the choice of threshold and a trade-off between their rate is to be faced:

- ***False Non-Match Rate (FNMR)***: the rate of error caused by making non-match decision, when a match decision should be made.

$$FNMR = P(D_0|H_1) = \int_0^t p(s|H_1)\,\mathrm{d}s \tag{2.1}$$

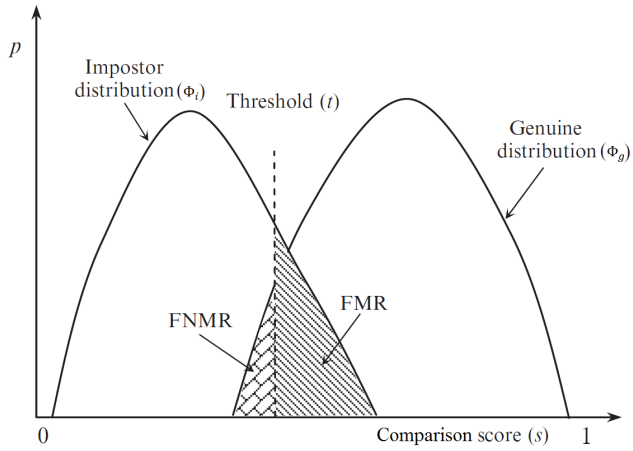


Figure 2.3: FMR and FNMR for a given threshold t are shown over the genuine and impostor comparison score distribution. Taken from [MMJP09]

- ***False Match Rate (FMR)***: the rate of error caused by making a match decision, when a non-match decision should be made.

$$FMR = P(D_1|H_0) = \int_t^1 p(s|H_0)\,\mathrm{d}s \tag{2.2}$$

The trade-off of the two error rates is chosen in respect to the required security level. Higher security contexts would require lower or even zero FMR (ZeroFMR), conversely lower security context would require a threshold closer to the zero FNMR (ZeroFNMR) level. The point of *Equal Error Rate (EER)* is also used to get the best of both rates.

2.4 Fingerprint recognition systems

Fingerprint recognition systems are a specific type of biometric systems focused on fingerprints. A *fingerprint* is an impression or mark made on a surface by a person's fingertip, which has a distinctive and unique skin texture that does not change over time. The *skin texture pattern* is made up of small ridges and valleys. The first usages of fingerprints could be traced back to ancient times, however their study has attracted scientists just a couple of centuries ago. They have started being used for the purpose of forensics in the mid of 19th century

and began being widely used for personal identification in the beginning of 20th century [MMJP09].

2.4.1 Fingerprint acquisition

Fingerprint sample acquisition could be classified as *off-line* or *live-scan*. Images acquired in *off-line* mode are typically obtained by smearing ink on the fingertip and taking an inked impression of the fingerprint on paper. Digitalizing the impression is done by scanning it using an optical scanner or a still camera. Off-line acquisition is of high interest in forensics due to the so called *latent fingerprints* that could be found at crime scenes. However, modern *Automated Fingerprint Identification Systems (AFIS)* rely on *live-scan* images acquired by scanning the fingertip on contact using a capable *live-scan* sensor.

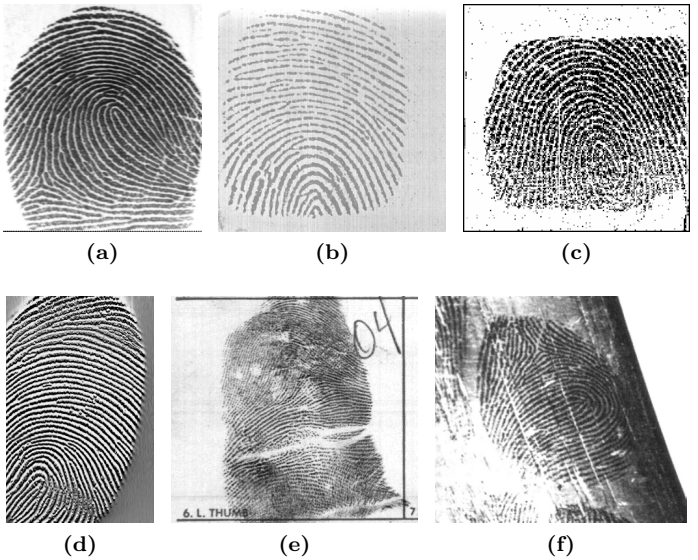


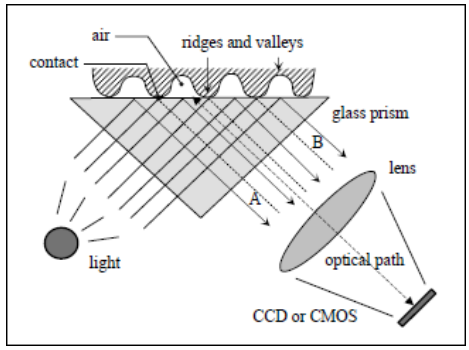
Figure 2.4: Fingerprint images from: (a) a live-scan FTIR-based optical scanner; (b) a live-scan capacitive scanner; (c) a live-scan piezoelectic scanner; (d) a live-scan thermal scanner; (e) an off-line inked impression; (f) a latent fingerprint. Taken from [MMJP09]

There are different live-scan sensing technologies, but the most popular ones could be divided into optical and solid-state sensors:

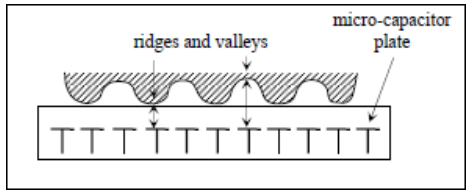
- ***Optical sensors***: they are usually based on the technology of *Frustrated*

Total Internal Reflection (FTIR), where the fingertip is placed on a side of a prism and is beamed with light from another side of the prism. Then the image of the fingerprint is projected to the third side of the prism, where a CCD or a CMOS camera is placed. The ridges touching the prism absorb the light, unlike the valleys that reflect it to the camera. Therefore the resulting image is with dark ridges and white valleys on white background. There are also other types of optical sensors, most robust of them use improved FTIR technology.

- ***Solid-state sensors***: there are different types of solid-state sensors (also known as silicon sensors), e.g. capacitive, thermal, electric field, piezo-electric, etc. However, *capacitive solid-state sensors* are the most common used ones. They are basically an array of micro-capacitor plates embedded in a single chip. When the fingertip skin touches the chip, small electrical charges are induced between the two surfaces. The charges are different at ridge and valley regions, so each micro-capacitor registers different charge level representing different pixel value in the resulting image.



(a)



(b)

Figure 2.5: Basic schemes for: (a) FTIR-based optical scanner; (b) capacitive solid-state sensor. Taken from [MMJP09]

2.4.2 Fingerprint features

The characteristics of a fingerprint should be described at three different levels: global level, local level and very local level. Each of these levels are characterized by different features that could be extracted:

- ***Global level (level 1)***: at this level ridges run usually in parallel, except some distinctive regions, called *singularities*, that could be classified into three subtypes: *loop*, *delta* and *whorl*. For the purpose of fingerprint image alignment, a *core point* is defined as the center of the northernmost loop type singularity, or if such does not exist - the point of maximum ridge line curvature. Singular regions are typically used for fingerprint classification among a set of distinct classes for easing further search and retrieval in the database.

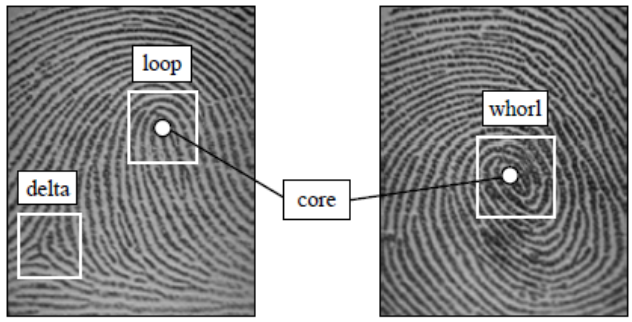


Figure 2.6: Examples of different singular points: loop, delta, whorl and core. Taken from [MMJP09]

- ***Local level (level 2)***: at this level other interesting patterns could be found in the fingerprint image, called *minutiae* (literally meaning small detail). There are several types of minutiae, namely - *ridge line endings*, *bifurcations*, *lakes*, *islands*, *crossovers*, etc. Their type, spatial coordinates and orientation (the angle between the tangent to the ridge line at the minutia position and the horizontal axis) are of particular interest to the comparison algorithms. Minutiae are important features in fingerprint sample comparison and usually 12-15 coinciding minutiae are sufficient evidence for high confidence *match* between two samples. However, a full rolled fingerprint can have more than 100 minutiae points altogether. *Local ridge orientation* is another local level feature, which could be used in low-quality fingerprint images, if minutiae extraction is not possible. These features distinctiveness is generally lower than minutiae.

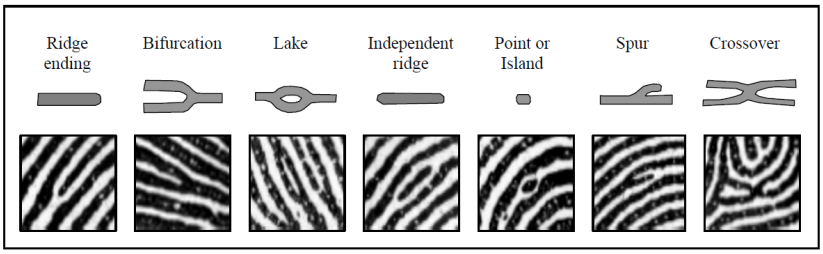


Figure 2.7: Seven most common minutia types. Taken from [MMJP09]

- ***Very local level (level 3)***: at this level very fine details could be extracted such as width, shape, contours, scars, etc. However, the most common used features are the *sweat pores* on the ridges. Detection of these features requires very high resolution (e.g., 1000 dpi) and very good quality of the image sample, therefore not many AFIS use the features at this level.

Figure 2.8: Fingerprint image taken with 1000dpi, where sweat pores are visible. Taken from [MMJP09]

2.4.3 Fingerprint comparison

Comparing fingerprint images is part of both identification and verification processes and is a difficult process, since the same fingertip can produce different impressions. These variations amongst fingerprints can be caused by displacement, partial overlap, rotation, non-linear distortion, different pressure, different skin conditions, image noise, unreliable feature extraction, etc. In order to claim that a probe and a reference originate from the same biometric source three types of approaches could be considered:

- ***Correlation-based matching***: two fingerprint images are compared pixel-wise and the *correlation* between them is computed for different displacements and rotations. Almost no feature extraction is needed, except some global level features to help the alignment of the fingerprint images.
- ***Minutiae-based matching***: for both the probe and the reference minutiae are extracted. Then an alignment between the samples is found such that the number of *paired minutiae* is the highest. This approach is the most widely used one.
- ***Non-minutiae feature-based matching***: for very low-quality images, where minutiae extraction is extremely difficult, other types of features are used for comparison such as local ridge orientation, local frequency information, ridge shape, texture information, etc. However, distinctiveness of these features is generally lower than minutiae.

Chapter 3

Biometric sample quality

This chapter is focused on *biometric sample quality* and its definitions, applications, ways to measure it and assess the performance of *quality measurement algorithms (QMAs)*. More attention is regarded to *fingerprint sample quality*. Vocabulary from ISO/IEC 29794-1 [ISO12b] is used for the definitions and the terms related to the subject.

3.1 Definitions

Biometric sample quality is the degree to which a biometric sample fulfills specified requirements for a targeted application [ISO12b]. It is dependent on the *biometric source* and the degree of its representation by the *captured biometric sample*. It has several components that contribute to the performance of the biometric system:

- ***Character***: an expression of quality based on *inherent features* of the biometric source. For example, scars on the fingertip would cause poor character of the taken biometric sample.
- ***Fidelity***: an expression of quality reflecting the *similarity* of the biometric sample to the biometric source of that sample. It is comprised of

different components, contributed by different aspects like *user behavior*, *environment conditions*, *biometric system modules performance*, etc.

- ***Utility***: an expression of quality reflecting the contribution of an *individual sample* to the *overall performance* of a biometric system. It depends on both the *fidelity* and the *character* of the biometric sample and is intended to be more predictive of system performance than fidelity or character alone.

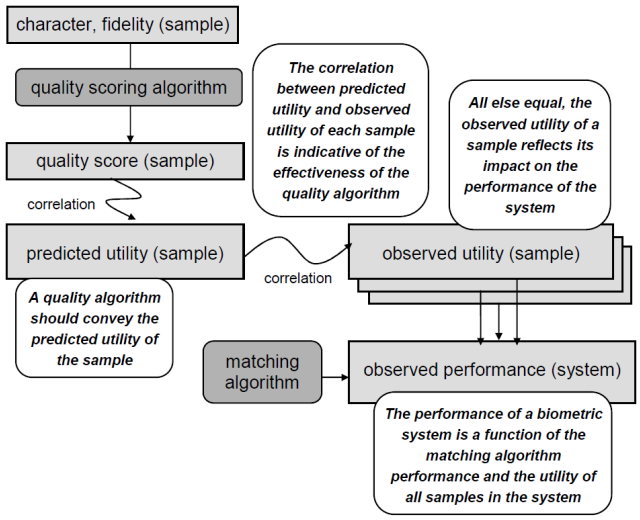


Figure 3.1: Relationship between quality and system performance. Taken from [ISO12b]

However, one measure of quality is biometric system dependent, as shown in [GT07], since different comparison systems are sensitive to different sample artifacts like blurriness, contrast, disposition, rotation, etc. Due to that fact establishing a universal quality standard defining a singular metric for the utility of the biometric sample for all applications is a challenge [ISO12b].

3.2 Applications

Biometric quality is of high importance to the modern biometric systems, therefore *quality estimation* and *quality measurement* have wide range of applications. There are at least several uses of biometric quality values, each aiming to improve the overall performance of biometric systems.

		Fidelity	
		Low	*High*
Character	*Low*	Low fidelity and low character results in low utility. Recapture might improve utility. However, if possible use of other biometric characteristics is recommended.	High fidelity and low character results in low utility. Recapture will not improve utility. Use of other biometric characteristics is recommended.
	High	Samples with high character and low fidelity typically will not demonstrate high utility. Utility can be improved upon recapture or image enhancement techniques.	Samples with high character and high fidelity indicate capture of useful sample. High utility is expected.

Table 3.1: Relationship between fidelity, character and utility [ISO12b].

3.2.1 Enrollment process

In the enrollment process a number of biometric samples are captured and stored in the database subsystem as *reference samples*. Ensuring high quality of the finally stored sample increases the overall performance of the biometric system. Even though the enrollment process is usually supervised and the quality of the samples could be checked by the operator, human judgement is subjective, slow, requires training and not scalable. Therefore both supervised and unsupervised systems would benefit from a good biometric sample *quality measurement algorithm* that can give *quantitative definition* of the quality of captured biometric samples. Then based on this quantitative measure a decision for reacquiring the biometric sample might be taken by either the supervisor or the automated system.

3.2.2 Quality assurance

Measuring quality levels and comparing them with *aggregated data* from different historical or geographical points, thus monitoring the performance throughout the usage of a biometric system, might signal some possible *unexpected performance issues*. This way some actions could be taken to avoid *biometric system performance* drops in the future.

3.2.3 Verification process

The policy "*up to three attempts*" is commonly used to ensure the good quality of the captured sample for verification purposes. If a positive match occurs within the three attempts, the sample is considered to be of a *good quality*

even if the source happens to be an *impostor*. However, this policy might turn non-applicable due to either computational expenses for the comparison or reacquisition expenses (i.e. time-waste, user frustration, etc.). Quality measurement algorithms usually provide faster execution times due to the fact that they require access to only one image, since quality is computed *per image*. However, they can be applied to both the probe and the reference, so they could be used to predict the performance of the comparison itself.

3.2.4 Identification process

Biometric sample quality measurement could be useful for the identification process for couple of reasons:

- ***Missing reference***: , a *one-to-many comparison* is very *inefficient* in terms of execution time and processing power. If a user has no associated reference record in the database system, it can be also *inconclusive*, since the returned set of candidate identities might include only impostor ones. Therefore using a policy similar to the "*up to three attempts*" one would be both *inefficient* and *inconclusive* method for assessing the probe sample's quality.
- ***Evasion detection***: for *negative identification systems* (e.g. border security checks), the subjects providing the biometric sources might be motivated to *evade* detection, thus they would likely *try* to submit a poorly performing biometric sample.
- ***FMR dominance compensation***: since identification involves one-to-many comparisons, FMR becomes *increasingly dominant* as the number of comparisons grows. Using *high quality* samples that produce *higher* genuine scores would influence the genuine scores distribution by increasing its *mean* and decreasing its *variance*, thus reducing its overlap with the impostor scores distribution. That would lower both FMR and FNMR, so the *matching threshold* could be increased to compensate for the FMR domination over FNMR.

3.2.5 Differential processing

The quality information for a biometric sample could be used for *conditional processing* of the captured sample. For example, low-quality samples might suggest the usage of better and slower: preprocessing steps, image restoration

algorithms, feature extraction algorithms, comparison algorithms, etc.; on the other hand, newly captured high quality samples could be used to *update* poorer reference templates.

3.3 Quality measurement

The previous sections have shown *what quality is* and *how it is useful* for biometric systems. However, the question for measuring the quality and developing quality measurement algorithms remains open for research. For the sake of assessing the performance of quality measurement algorithms, couple of metrics should be introduced.

3.3.1 Utility-based quality

Utility was aforementioned as quality component, depending on both *fidelity* and *character* of the sample, thus being better performance predictor in terms of FNMR and FMR. If the performance of a dataset over a set of comparison algorithms is known, a *performance-based* quality score directly related to the results of the execution of the given biometric systems could be extracted for that dataset. This performance-based quality score is known as *observed utility* and a procedure for calculating it is defined in ISO/IEC 29794-1 [ISO12b]. Finally, the produced results give a rise to a *Quality Reference Dataset (QRD)*, enabling a consistent and interoperable interpretation of the *quality score*.

3.3.1.1 Observed utility calculation

For calculating the observed utility score, a biometric dataset for a given biometric characteristic (e.g. fingerprint) containing $N_i \geq 2$ samples, $d_i^{(1)}$, $d_i^{(2)}, \ldots, d_i^{(N_i)}$, for each of M subjects, $i = 1, \ldots, M$, should be considered. Then for all images in the reference dataset, utility values $utility_i^1$, $utility_i^2, \ldots, utility_i^{N_i}$ are assigned by the following procedure [ISO12b]:

For each instance record $d_i^{(u)}$ (i.e. the u^{th} sample of subject i) for comparator V_k, $k = 1, \ldots, K$:

1. Generate the set of all possible genuine comparison scores given by the

k^{th} comparator

$$\begin{aligned} S_{ii} =& \{s_{i,i}^{u,v} \mid s_{i,i}^{u,v} = V_k(d_i^{(u)}, d_i^{(v)})\} \\ u =& 1, \ldots, N_i \text{ and } v = u+1, \ldots, N_i \\ i =& 1, \ldots, M \end{aligned} \tag{3.1}$$

2. Generate the set of all possible impostor comparison scores given by the k^{th} comparator

$$\begin{aligned} S_{ij} =& \{s_{i,j}^{u,v} \mid s_{i,j}^{u,v} = V_k(d_i^{(u)}, d_j^{(v)})\} \\ u =& 1, \ldots, N_i \text{ and } v = 1, \ldots, N_j \\ i =& 1, \ldots, M \text{ and } j = 1, \ldots, M \text{ and } i \neq j \end{aligned} \tag{3.2}$$

3. Compute the utility for sample $d_i^{(u)}$

$$utility_i^u = \frac{m_{i,u}^{genuine} - m_{i,u}^{impostor}}{\sigma_{i,u}^{genuine} + \sigma_{i,u}^{impostor}} \tag{3.3}$$

where $m_{i,u}^{genuine}$ is the mean of sample $d_i^{(u)}$'s genuine comparison scores:

$$m_{i,u}^{genuine} = \frac{\sum_{\substack{v=1 \\ v \neq u}}^{N_i} s_{i,i}^{u,v}}{N_i - 1} \tag{3.4}$$

and $m_{i,u}^{impostor}$ is the mean of sample $d_i^{(u)}$'s impostor comparison scores:

$$m_{i,u}^{impostor} = \frac{\sum_{\substack{j=1 \\ j \neq i}}^{M} \sum_{v=1}^{N_j} s_{i,j}^{u,v}}{\sum_{\substack{j=1 \\ j \neq i}}^{M} N_j} \tag{3.5}$$

where $\sigma_{i,u}^{genuine}$ is the standard deviation of sample $d_i^{(u)}$'s genuine comparison scores:

$$\sigma_{i,u}^{genuine} = \sqrt{\frac{\sum_{\substack{v=1 \\ v \neq u}}^{N_i} (s_{i,i}^{u,v} - m_{i,u}^{genuine})^2}{N_i - 1}} \tag{3.6}$$

and $\sigma_{i,u}^{impostor}$ is the standard deviation of sample $d_i^{(u)}$'s impostor comparison scores:

$$\sigma_{i,u}^{impostor} = \sqrt{\frac{\sum_{\substack{j=1 \\ j\neq i}}^{M} \sum_{v=1}^{N_j} (s_{i,j}^{u,v} - m_{i,u}^{impostor})^2}{\sum_{\substack{j=1 \\ j\neq i}}^{M} N_j}} \tag{3.7}$$

3.3.1.2 Utility binning

Working with directly computed utility scores is not convenient due to the fact that they are not normalized and can have quite different range across different comparators. Therefore a binning procedure [ISO12b] is defined to separate the scores into several quality levels for each comparator k:

1. A set T is defined by collecting all samples (i, u) having genuine comparison scores greater than all theirs impostor comparison scores, i.e. $s_{i,i}^{u,v} > s_{i,j}^{u,w}\ \forall j \neq i, v \neq u, w$, known as rank 1 condition.
2. Define two empirical cumulative distribution functions:

$$C(z) = \frac{|\{utility_i^u \mid (i,u) \in T, utility_i^u \leq z\}|}{|\{utility_i^u \mid (i,u) \in T\}|} \tag{3.8}$$

   and another for those not in that set,

$$W(z) = \frac{|\{utility_i^u \mid (i,u) \notin T, utility_i^u \leq z\}|}{|\{utility_i^u \mid (i,u) \notin T\}|} \tag{3.9}$$

3. Choose number of quality levels $L(2 \leq L \leq 100)$, so quality scores will be $q_i^{(u)} = 1, \ldots, L$ where 1 is the lowest and L is the highest.
4. Bin utility scores into L bins according to the quantiles of the target utility distributions $C(.)$ and $W(.)$. An example binning for $L = 5$ is shown in table 3.2, where $W^{-1}(.)$ and $C^{-1}(.)$ are the quantile functions, such that $C^{-1}(0)$ and $C^{-1}(1)$ ($W^{-1}(0)$ and $W^{-1}(1)$) denote the empirical minima and maxima. Respectively, x and y are appropriate percentile points selected based on the shape of $C(.)$. An example plot of a cumulative distribution function is shown in figure 3.2

Bin	Range of target utilities
1	$\{z_i \mid -\infty < z_i < C^{-1}(0.01)\}$
2	$\{z_i \mid C^{-1}(0.01) \leq z_i < W^{-1}(1)\}$
3	$\{z_i \mid W^{-1}(1) \leq z_i < C^{-1}(x)\}$
4	$\{z_i \mid C^{-1}(x) \leq z_i < C^{-1}(y)\}$
5	$\{z_i \mid C^{-1}y \leq z_i\}$

Table 3.2: Binning utility scores [ISO12b].

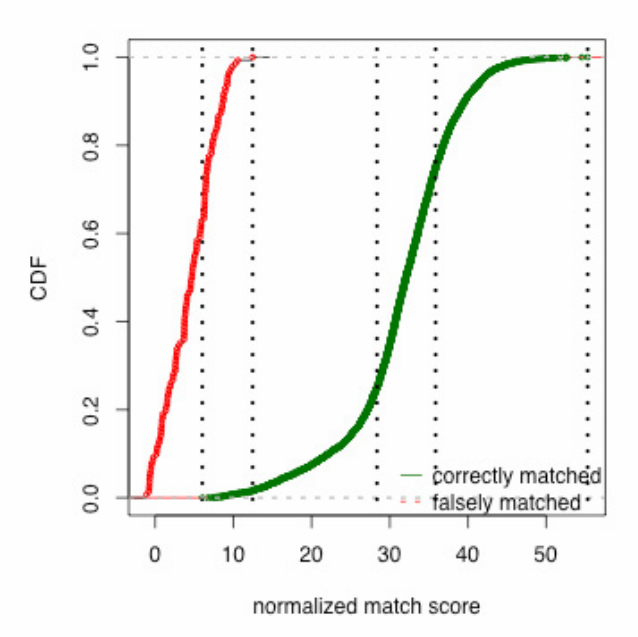


Figure 3.2: Example empirical cumulative distribution of the impostor scores (red) and genuine scores (green). Taken from [ISO12b]

3.3.1.3 Quality score fusion

Since quality score computation is comparator specific, a generalized quality score might be produced by aggregating the results of all comparators. When considering different aggregate functions, one should be aware of the possible options:

- ***Unanimity***: Samples with identical quality scores from all K comparators become members of the QRD, where all the rest are discarded.
- ***Median (or other percentile point)***: Samples with identical quality scores from more than X percent of K comparators become part of QRD and the rest are discarded. Note that $X = 100$ is the unanimity and $X = 50$ is the majority vote rule.
- ***Arithmetic mean***: The final quality score of each sample will be the mean of its quality scores from all K comparators.

3.3.2 Spearman correlation

After a quality measurement algorithm is developed, its performance needs to be assessed. Direct comparison of the estimated quality scores with the binned *observed utility scores* for a given dataset is one way of assessing it. However, quality score resolution and score assigning might differ across different QMAs, thus making direct comparison of the produced scores is unfit for the purpose. Therefore different methods need to be applied. One of them is *Spearman correlation*:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 (y_i - \bar{y})^2}} \tag{3.10}$$

Spearman's rank correlation coefficient was first introduced by Charles Spearman [Spe87] for statistical measurement of the degree of *correlation* between two variables. It is denoted ρ and it shows the degree of *monotonicity* of the relation of the two variables. A perfect correlation will give value of $\rho = \pm 1$, where positive value indicates that the variables share increasing monotonic trend and negative value indicates that the monotonic trend is decreasing. A

value of $\rho = 0$ indicates *statistical independence* of the two variables, i.e. they are completely unrelated.

3.3.3 Error versus Reject Curves

Error versus Reject Curves (ERC) was proposed by Grother and Tabassi [TG09] aiming to model *real-world* operational cases where quality is maintained by reacquisition. The method visualizes the relation between FNMR and the *fraction* of *rejected* poor quality samples. If well-performing QMA is causing rejection of poor quality samples and requiring their reacquisition until the quality is good enough, the performance of the biometric system should improve in terms of lowering FNMR, since poor quality samples tend be falsely rejected by receiving genuine comparison scores below the biometric threshold. Rejecting more than one third of the samples might lead to user frustration, therefore only part of the ERC plot is of interest, namely the part showing the rejection of the *poorest* 35% of the samples.

For a given dataset of samples $d_i^{(u)}$ (i.e. the u^{th} sample of subject i), its all genuine comparison scores $s_{i,i}^{u,v}$ produced by comparator k, and all estimated by the specified QMA quality scores $q_i^{(u)}$, the procedure of generating the ERC curve is as follows:

1. For each genuine comparison score $s_{i,i}^{(u,v)}$, there is a pair of biometric samples $p_i^{(u,v)} = (d_i^{(u)}, d_i^{(v)})$. Combine their quality scores into $q_i^{(u,v)}$

$$q_i^{(u,v)} = H(q_i^{(u)}, q_i^{(v)}), \text{ e.g. } H(.) = \sqrt{q_i^{(u)} q_i^{(v)}} \tag{3.11}$$

   If the number of quality levels is less than the suggested by ISO/IEC value of 100, in order to arrive with non-stepwise and smoother approximation of the lower convex hull of the geometric mean curve, a Gaussian noise should be added to $\sqrt{q_i^{(u)} q_i^{(v)}}$, e.g. $N(0, 0.01 \times L_{step})$, where L_{step} is the step between two quality levels. This adding of Gaussian noise is called *jittering*.

2. Define a set of low quality entries $R(m)$ by introducing level of acceptable quality threshold m

$$R(m) = \left\{ p_i^{(u,v)} \mid q_i^{(u,v)} < m \right\} \tag{3.12}$$

3. Determine a threshold $t = C^{-1}(f)$ for the genuine comparison scores, where $C^{-1}(.)$ is the quantile function (the inverse of the empirical cumulative distribution) for the genuine comparison scores and f is a reasonable FNMR, e.g. 10% for $f = 0.1$

4. Define the value of FNMR for all quality thresholds caused by the gradual change of m from 0 to the fraction of rejected samples of interest, e.g. 0.35 for 35%

$$FNMR(t, m) = \frac{\left|\left\{s_{i,i}^{(u,v)} \mid s_{i,i}^{(u,v)} \leq t,\, p_i^{(u,v)} \notin R(m)\right\}\right|}{\left|\left\{s_{i,i}^{(u,v)} \mid p_i^{(u,v)} \notin R(m)\right\}\right|} \tag{3.13}$$

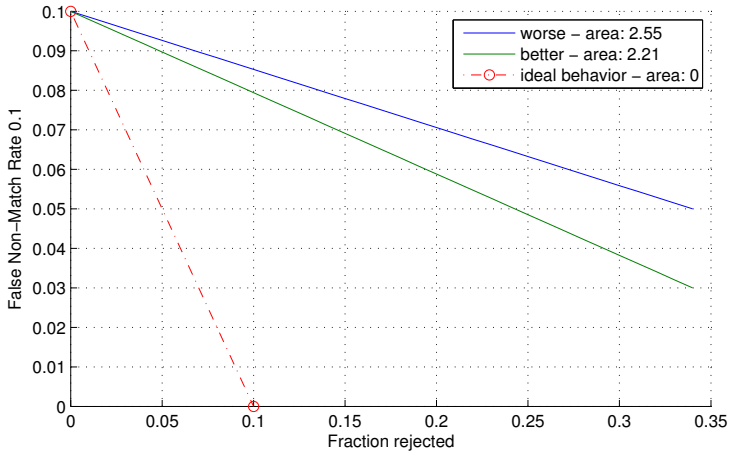


Figure 3.3: An example ERC plot. FNMR is set to 0.1 and at most 35% are rejected.

By following the procedure and gradually changing the fraction of rejected samples m, an ERC plot could be drawn, as shown on figure 3.3. There are three curves on the example plot, all of them starting from the same point of $f = 0.1$, i.e. the threshold is such that the FNMR is 10% with no samples rejected. The dashed red one is denoting the ideal case, where rejecting the samples with the poorest 10% of the quality scores would cause decrease in the FNMR by the same fraction. However, such behavior in practice is unrealistic, so QMAs usually have curves more similar to the other two curves. For better QMAs the curve goes down faster (e.g. the green curve), thus approaching the ideal case curve.

3.4 Fingerprint sample quality

Fingerprint image samples are compared by the comparison algorithms for their *ridge* impressions, either by extracting minutiae or by extracting some other features. As mentioned in section 3.1, biometric sample quality has two components: *character* and *fidelity*; both affecting the third component *utility*. On the other hand QMAs try to assign quality score to each sample, which reflects more or less their utility. Therefore a closer look at both character and fidelity for fingerprint samples should be done.

(a) (b) (c) (d) (e)

Figure 3.4: Examples of fingerprint samples acquired with an optical scanner from the same finger in different conditions:
(a) too much pressure;
(b) dry skin;
(c) normal conditions;
(d) dirty skin;
(e) wet skin.
Taken from [Dus13]

There are number of factors that influence the character of the fingerprints, such as scars, creases, blisters, etc. All of them are inherent to the fingerprint source and cannot be avoided. However, for improving the *fidelity* of the fingerprints some actions before reacquiring could be taken, since it is highly dependent on several controllable factors:

- ***Skin condition***: moisture, dryness, cleanness, etc. Too wet, too dry fingers or dirty fingertips cause capturing of poor impressions due to presence of artifacts in the captured image. Cleaning the fingertips and drying/moisturizing them before reacquisition improves the quality of the captured samples.
- ***User behavior***: finger positioning, pressure, deformations, etc. Improper positioning of the fingertip causes rotation and translation image artifacts,

for which many comparison algorithms are highly sensitive. Pressure and ridge deformations may cause ridges to change their shape or even make some valleys disappear completely. Reacquisition attempts under the supervision of an assistant are very likely to improve the quality of the captured samples.

- ***Environmental conditions***: sensor platen cleanness, temperature, etc. Sensor platen dirtiness causes artifacts or latent fingerprints to be captured additionally to the impression of the ridges. Cleaning of the sensor solves that issue for the subsequent attempts.

Chapter 4

Considered machine learning methods

The previous chapters have introduced the reader to the field of *biometrics* and *biometric sample quality*. Throughout biometric sample quality analysis, different features are considered and used to determine sample quality. The problem of *classifying* or *clustering samples*, given a list of features each, is a good invitation for applying machine learning methods for biometric features analysis. Since machine learning is a broad topic itself, this chapter is aimed to introduce the reader to the machine learning methods used in the experiments. Some basic prior knowledge of *machine learning*, *linear algebra* and *probability theory* is assumed.

4.1 Self-organizing maps

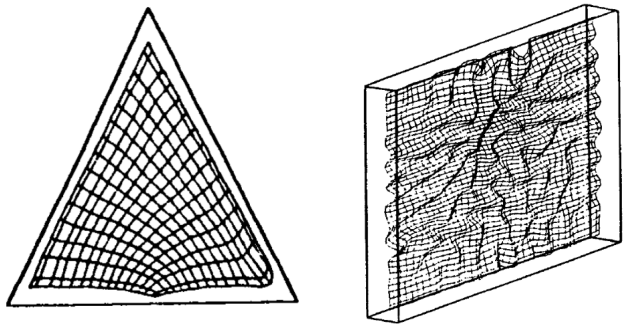
4.1.1 Essentials

Self-organizing maps (SOM) have been introduced by [Koh82] and are a biologically inspired unsupervised learning neural network model [KSH01]. It is a dimensionality reduction technique, similar in principle to *vector quantization*, where a number of *codebook vectors* $\mathbf{m}$ are fitted to represent some larger list

of *data vectors* $\mathbf{x}$. The dimensionality of the codebook vectors is the same as the dimensionality of the data and a distance metric is defined, e.g. *Euclidean distance*:

$$d(\mathbf{x}, \mathbf{m}) = \|\mathbf{x} - \mathbf{m}\| = \sqrt{\sum_{i=1}^{|\mathbf{x}|} (x_i - m_i)^2} \tag{4.1}$$

Unlike vector quantization, SOM maintains *globally* and *spatially* ordered codebook vectors, usually forming a two-dimensional grid for visualization purposes. After the model has reached *convergence*, a *non-linear* mapping of the high-dimensional data space to the lower-dimensional grid space is achieved. The training data topology is preserved as much as possible by the resulting *network (map)*. Later on this network could be analyzed to find interesting patterns of the data and cluster it.



(a) triangular 2D data **(b)** cuboidal 3D data

Figure 4.1: Example SOM networks trained with (a) triangular 2D data and (b) cuboidal 3D data. Images taken from [KSH01]

The network could be calibrated if the labels of the training data are known. *Calibration* is done in the following manner: each *node* gets a label, estimated from the training samples mapped to that node, usually by *majority voting*. If not calibrated, the coordinates of the *best-matching unit* (the node that is closest to the sample) could be used as features themselves. This way the dimensionality of the data is reduced to the dimensionality of the network *lattice*.

SOM lacks a well-defined *objective function* due to its *heuristic* nature, which makes convergence hard to prove and measure mathematically. Therefore *mean quantization error* (4.2) has been adopted to assess the quality of the trained

maps, where N is the number of all samples $\mathbf{x}(t)$ and $\mathbf{m}_c(t)$ is their best-matching unit.

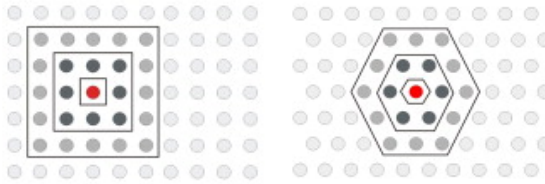
$$E = \frac{1}{N} \sum_{t=1}^{N} \|\mathbf{x}(t) - \mathbf{m}_c(t)\| \tag{4.2}$$

4.1.2 SOM network architecture

The network grid arrays could be constructed in a couple of different ways. The most common of them is the *regular grid array*, where the two-dimensional array represents a sheet in the data space. All nodes have a number of neighboring nodes, except the bordering nodes at the edge of the sheet. This type of grid arrays introduce *bordering effects*: the spacings of the neighboring nodes are not as regular near the borders as in the middle of the SOM [Koh13].

However, if the data has a cyclic structure and to avoid some of the bordering effects, a *cyclic arrays* could be used, either *toroidal* or *spherical*.

Depending on the number of node neighbors, a map could be *rectangular* (four neighbors) or *hexagonal* (six neighbors) as shown in figure 4.2. Hexagonal maps are more accurate and better suited for visualizations, therefore they are recommended [Koh13]. Some other number of neighbors are rarely used.



(a) (b)

Figure 4.2: Depending on the number of neighbors, a map could be: (a) rectangular; (b) hexagonal. Images taken from [SMN+13]

4.1.3 SOM network initialization

The size of the network (the number of the grid nodes) is usually determined heuristically. Too small map would not reflect the data distribution well enough,

while too big map would introduce some empty nodes, i.e. some nodes will be responsible for no samples. Therefore the map size is taken as a fraction of the number of training samples. However when dealing with large datasets, the computational load should be taken into account, so a trade-off between the processing power and the precision of the network is to be faced. Once the number of nodes is determined, they should be initialized.

4.1.3.1 Random initialization

A fast and easy way to initialize the nodes is to randomize their codebook vectors or to assign them to randomly selected samples from the training data. Since no data structure information is known, the number of nodes in each dimension of the grid lattice is taken to be the same. If the data is not available prior to the training process, this way is the only way to initialize the SOM.

However, random initialization is not robust, so every new training on the same data could result in different convergent state, if the random generator seed is different. Also data structure is not taken into account and that may cause poorer fitting of the map. Codebook vectors initially are not ordered globally, so a *rough* training phase has to be introduced before the actual training begins.

4.1.3.2 Linear initialization

Another way of initializing the codebook vectors is to *linearly initialize* them across the first *principal components* of the training data. The number of grid nodes along each dimension of the lattice is determined respective to the *eigenvalues* of the principal components.

However, calculation of principal components is computationally expensive for large datasets of high-dimensional vectors and requires availability of the training data prior to the training process. Since usually a two-dimensional SOM maps are used, only the first two principal components are required, so they can be estimated iteratively efficiently by [Row98].

4.1.4 Stepwise SOM algorithm

The original SOM algorithm is the *stepwise SOM* algorithm. The data is treated as *sequential data*, where each data sample is a step t in a *time sequence* $\mathbf{x}(t)$.

The codebook vectors are also treated as vector sequences $\mathbf{m}_i(t)$, where i is the spatial index of the grid node. The algorithm is as follows:

- Determine the best-matching unit

$$c = \underset{i}{\operatorname{argmin}} \{\|\mathbf{x}(t) - \mathbf{m}_i(t)\|\} \tag{4.3}$$

- Determine neighbors of each model $\mathbf{m}_c(t)$ using a *neighborhood function*

$$h_{ci}(t) = \alpha(t) \exp\left[-\frac{\|\mathbf{m}_c(t) - \mathbf{m}_i(t)\|^2}{2\sigma^2(t)}\right] \tag{4.4}$$

- Update grid node c and its spatial neighbors

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)\,[\mathbf{x}(t) - \mathbf{m}_i(t)] \tag{4.5}$$

One has to notice that the *neighborhood* of the best-matching node is determined by all the non-zero values of the neighborhood function, so only they contribute to the update of the *winner node* and get updated themselves.

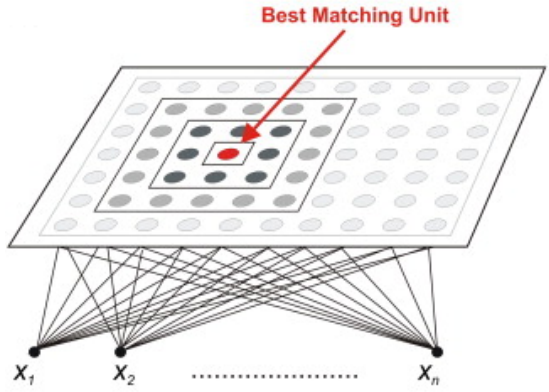
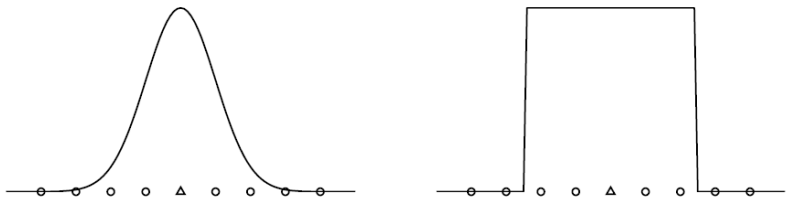


Figure 4.3: Best-matching unit is determined using equation (4.3), then the neighborhood is determined by (4.4) and the winner node and its spatial neighbors get updated. Images taken from [SMN+13]

The neighborhood function (4.4) changes over time, controlling the *smoothness* of the training process. A simple values could be $h_{ci} = 1$ within certain neighborhood from the best-matching unit and $h_{ci} = 0$ otherwise. This type of neighborhood is called *bubble* neighborhood. However, equation (4.4) shows a more

general *Gaussian*-like neighborhood function, where $\alpha(t)$ is a monotonically decreasing scalar *learning rate* function and $\sigma(t)$ is a monotonically decreasing scalar range representing function. While $\alpha(t)$ determines the influence to the models of every new sample, i.e. the *elasticity* of the "*memory*" of the model, $\sigma(t)$ affects the neighborhood directly and usually starts with quite large values (half the diameter of the grid in the data space) gradually decreasing to only fraction of it, avoiding to reach zero.



(a) **(b)**

Figure 4.4: The neighborhood function could be either (a) a Gaussian or (b) a bubble one. Images taken from [Sve98]

4.1.5 Batch SOM algorithm

The *batch SOM* algorithm relies on availability of all the training data, therefore no learning rate function $\alpha(t)$ is needed in the neighborhood function $h_{ci}(t)$. A list of best-matched samples $\mathbf{N}_c$ is maintained for each model $\mathbf{m}_c(t)$, where references to all samples, whose winner node is c, are kept. The algorithm is as follows:

- Determine the winner node $\boldsymbol{m_c}(t)$ for each $\boldsymbol{x}(t)$ using (4.3) and put each $\boldsymbol{x}(t)$ to the list $\boldsymbol{N}_c$ of their $\boldsymbol{m_c}(t)$
- Determine the neighbors of each model $\boldsymbol{m_c}(t)$ using $h_{ci}(t)$ from (4.4) without $\alpha(t)$
- Update each model $\mathbf{m}_c(t)$ with the weighted mean of $\boldsymbol{N_c}$ and all $\boldsymbol{N_i}$, where each sample $\mathbf{x}(t)$ of node list $\boldsymbol{N_i}$ is weighted with $h_{ci}(t)$
- Clear the list of vectors $\boldsymbol{N_c}$ for each $\mathbf{m}_c(t)$

The update of the models is performed in a *batch update* iteration for all the grid models at once, unlike the *sequential update* iterations used in the stepwise

SOM. This makes the algorithm multithread- and multiprocess-friendly, so good advantage of modern CPU architectures could be taken. However, it introduces problems with storing all the data in the operating memory, so it should be carefully implemented, if meant to deal with large datasets.

4.2 Generative topographic mapping

4.2.1 Essentials

Generative Topographic Mapping (GTM) was introduced by [Sve98] as a principled alternative to the SOM. It is a probabilistic counterpart of SOM, that models the distribution $p(\mathbf{t})$ of the D-dimensional data by using a number of L-dimensional *latent* variables $\mathbf{x}$, where usually $L = 2$ for purposes of visualization. The mapping from the *latent space* to the *data space* is done by using the transformation $\mathbf{y}(\mathbf{x}; \mathbf{W})$, where $\mathbf{W}$ is a parameter matrix of some *non-linear* model, e.g. *neural network* [Bis07]. Usually $\mathbf{y}(\mathbf{x}; \mathbf{W})$ is chosen to be a *generalized linear regression model* [Bis07] of the form $\mathbf{y}(\mathbf{x}; \mathbf{W}) = \mathbf{W}\phi(\mathbf{x})$, where $\phi(\mathbf{x})$ consists of M fixed *radial basis functions* [Bis07].

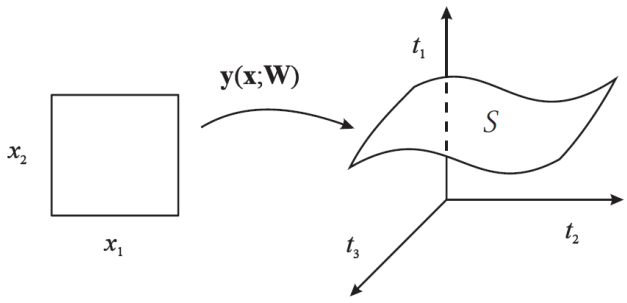


Figure 4.5: The mapping from latent space (left) to data space (right). Taken from [Sve98]

Defining a *probability distribution* $p(\mathbf{x})$ in latent space gives a rise to a corresponding distribution $p(\mathbf{t} \mid \mathbf{W})$ in data space, which is confined to the image of the L-dimensional manifold in the data space under the mapping $\mathbf{x} \to \mathbf{y}$ of the non-linear function $\mathbf{y}(\mathbf{x}; \mathbf{W})$. Since in reality the data can only approximately lie on a lower-dimensional manifold, a *noise model* for the $\mathbf{t}$ vector should be included. The distribution for $\mathbf{t}$, given $\mathbf{x}$ and $\mathbf{W}$, becomes *radially-symmetric*

Gaussian centered on $\mathbf{y}(\mathbf{x};\mathbf{W})$ having variance β^{-1} so that

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{W}, \beta) = \left(\frac{\beta}{2\pi}\right)^{D/2} \exp\left\{-\frac{\beta}{2} \|\mathbf{y}(\mathbf{x};\mathbf{W}) - \mathbf{t}\|^2\right\} \tag{4.6}$$

In order to obtain the distribution in the data space, for a given weight matrix $\mathbf{W}$, an integration over the latent distribution should be done

$$p(\mathbf{t} \mid \mathbf{W}, \beta) = \int p(\mathbf{t} \mid \mathbf{x}, \mathbf{W}, \beta) p(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{4.7}$$

However, computing the integration would be *infeasible*, therefore a discrete form of $p(\mathbf{x})$ is taken, such that

$$p(\mathbf{t} \mid \mathbf{W}, \beta) = \frac{1}{K} \sum_{i=1}^{K} p(\mathbf{t} \mid \mathbf{x}_i, \mathbf{W}, \beta) \tag{4.8}$$

where all $\mathbf{x}_i$ are K nodes, centered on a regular grid in the latent space.

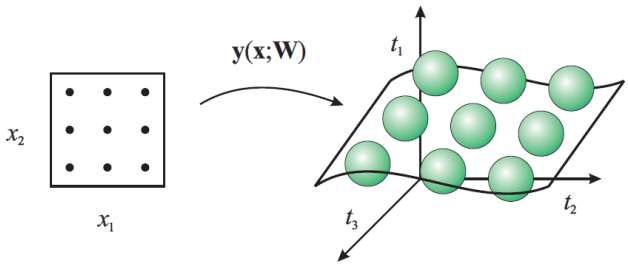


Figure 4.6: The discrete regular grid of nodes in latent space (left), mapped to the data space (right), where they correspond to centers of Gaussians with variance β^{-1}. Taken from [Sve98]

For a given data set, the *log likelihood* is defined as

$$\mathcal{L}(\mathbf{W}, \beta) = \ln \prod_{n=1}^{N} p(\mathbf{t}_n \mid \mathbf{W}, \beta) \tag{4.9}$$

In order to determine the parameters $\mathbf{W}$ and β, one has to maximize the log likelihood. That is done by an *EM-algorithm* [DLR77].

4.2.2 EM-algorithm

The *Expectation-Maximization algorithm (EM-algorithm)* is done in two steps, called respectively *Expectation (E-step)* and *Maximization (M-step)*.

In the **E-step**, given the current weight matrix $\mathbf{W}_{old}$ and the current inverse noise variance β_{old}, the responsibilities of each Gaussian component i for every training data vector $\mathbf{t}_n$ are evaluated using *Bayes' theorem* by:

$$R_{in}\left(\mathbf{W}_{old}, \beta_{old}\right) = p(\mathbf{x}_i \mid \mathbf{t}_n, \mathbf{W}_{old}, \beta_{old}) = \frac{p(\mathbf{t}_n \mid \mathbf{x}_i, \mathbf{W}_{old}, \beta_{old})}{\sum_{i'=1}^{K} p(\mathbf{t}_n \mid \mathbf{x}_{i'}, \mathbf{W}_{old}, \beta_{old})} \tag{4.10}$$

Then the expectation of the *complete log likelihood* is calculated by:

$$\langle \mathcal{L}_{comp}(\mathbf{W}, \beta) \rangle = \sum_{n=1}^{N} \sum_{i=1}^{K} R_{in}(\mathbf{W}_{old}, \beta_{old}) \ln p(\mathbf{t}_n \mid \mathbf{x}_i, \mathbf{W}, \beta) \tag{4.11}$$

In the **M-step**, equation (4.11) is maximized with respect to $\mathbf{W}$ and β. The maximized equation with respect to $\mathbf{W}$ can be written as:

$$\mathbf{\Phi}^T \mathbf{G}_{old} \mathbf{\Phi} \mathbf{W}_{new}^T = \mathbf{\Phi}^T \mathbf{R}_{old} \mathbf{T} \tag{4.12}$$

where $\mathbf{\Phi}$ is a $K \times M$ matrix for all $\boldsymbol{\phi}_m(\mathbf{x}_k)$, $\mathbf{T}$ is the $N \times D$ data matrix, $\mathbf{R}$ is the $K \times N$ matrix with elements R_{in}, $\mathbf{G}$ is $K \times K$ diagonal matrix, given by $G_{ii} = \sum_{n=1}^{N} R_{in}(\mathbf{W}, \beta)$.

If needed, a *regularization* term λ could be added to control the mapping $\mathbf{y}(\mathbf{x}; \mathbf{W})$, which could be interpreted as a *radially-symmetric Gaussian prior* over the weights $\mathbf{W}$:

$$p(\mathbf{W} \mid \lambda) = \left(\frac{\lambda}{2\pi}\right)^{MD/2} \exp\left\{-\frac{\lambda}{2} \sum_{j=1}^{M} \sum_{k=1}^{D} w_{jk}^2\right\} \tag{4.13}$$

That will lead to the modification of equation (4.12) as follows:

$$\left(\mathbf{\Phi}^T \mathbf{G}_{old} \mathbf{\Phi} + \frac{\lambda}{\beta} \mathbf{I}\right) \mathbf{W}_{new}^T = \mathbf{\Phi}^T \mathbf{R}_{old} \mathbf{T} \tag{4.14}$$

Similarly, maximizing equation (4.11) with respect to β, gives:

$$\frac{1}{\beta_{new}} = \frac{1}{ND} \sum_{n=1}^{N} \sum_{i=1}^{K} R_{in}(\mathbf{W}_{old}, \beta_{old}) \left\| \mathbf{W}_{new} \phi(\mathbf{x}_i) - \mathbf{t}_n \right\|^2 \tag{4.15}$$

By solving the equations for $\mathbf{W}_{new}$ and β_{new}, the **M-step** is finalized.

4.2.3 Parameter selection

As with all parametric models, the GTM model depends on its choice of parameters. Having chosen the form of the non-linear mapping to be a generalized linear regression model of the form $\mathbf{y}(\mathbf{x}; \mathbf{W}) = \mathbf{W}\phi(\mathbf{x})$, where $\phi(\mathbf{x})$ consists of M fixed radial basis functions, a couple of more parameters of the non-linear model are introduced as well, i.e. the *common width* of the radial basis functions σ and their number M.

Usually machine learning methods distinguish parameters that influence directly the model ($\mathbf{W}$) and ones that have an implicit effect (λ, β, σ) by calling the latter *hyper-parameters*, however here all of them will be called simply *parameters* and $\mathbf{W}$ will be called *weights*.

4.2.3.1 Parameters

A list and description of all the model parameters will be provided here:

- M - the number of radial basis functions to be used; each radial basis function is situated on a regular grid in the latent space.
- σ - the common width of the radial basis functions, controlling the global smoothness of the spanned manifold. Increasing σ would cause stiffer manifolds, where decreasing it will make the radial basis functions increasingly uncorrelated, thus non-linearity of the manifold will gradually diminish.
- K - the number of sample points $\mathbf{x}_i$ controls the precision of the *approximation* of the integration from equation (4.7). It influences the flexibility of the model and choosing $K \geq N$ can easily cause model *overfitting*. Usually K is chosen to be $N > K \geq M$.

- **W** - these weights directly control the mapping from latent to data space and are estimated by the EM-algorithm.
- β - this parameter is the inverse noise variance and is estimated by the aforementioned EM-algorithm. It affects the smoothness of the manifold at local level. Too high values of β lead to training an inflexible model, where too small values force the model to fit better the training data, causing overfitting, if enough sample points K are given.
- λ - this parameter is the inverse variance of the prior over the weights **W** and is used as regularization term, controlling the magnitude of the weights and the scale of the manifold. Therefore it restricts further capturing the noise of the data, since the underlying distribution is of interest.

4.2.3.2 Cross-validation

Parameters of the model could be determined by *cross-validation*. For cross-validation of given parameter values, the *training set* is divided into several equally sized sets S. Each of the smaller sets is used as a *validation set* once and the union of the other sets is used as training set. Therefore, a number S of models are trained and the log likelihood of the trained model over the validation set could be calculated. A mean of all log likelihoods is taken, in order to improve the *generalizability* of the results.

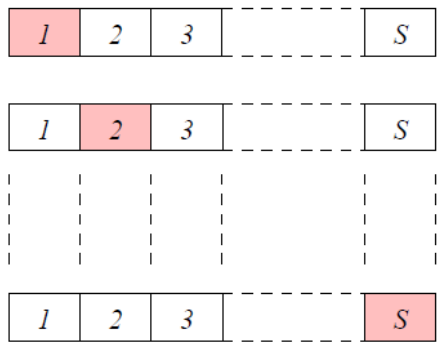


Figure 4.7: A pictorial illustration of cross-validation. Each row correspond to a different division of the data into validation set (shaded) and training set (others). Taken from [Sve98]

The aforementioned procedure is performed for a range of parameter values and the combination of parameters yielding the best average log likelihood is chosen

for the final model training. However, since the number of trainings grows *exponentially* in the number of parameters to be chosen, some of them should be estimated by the *rule of the thumb* in order to substantially reduce the computational expenses. Therefore the parameters M, K are determined heuristically by human supervision, while for the parameters σ and λ are tried out range of values. $\mathbf{W}$ and β are estimated by using the EM-algorithm, explained earlier in this chapter. This way only a *quadratic* number of trials are performed to approximate the best pair of σ and λ. However, since small changes in the parameter values may cause large differences, *hierarchical parameter space exploration* could be considered for better results, thus slowing down further the process of model selection.

4.2.3.3 Baysian approach

Since the GTM is a probabilistic model, estimating the parameters could be done in *Baysian framework*. As shown in details in [Sve98], a distribution over possible weight matrices depending on the training data could be defined

$$p(\mathbf{w} \mid \mathbf{T}) = \frac{p(\mathbf{T} \mid \mathbf{w})p(\mathbf{w})}{p(\mathbf{T})} \tag{4.16}$$

where $\mathbf{w}$ is a vector of all the weights in $\mathbf{W}$, $p(\mathbf{T} \mid \mathbf{w})$ is the likelihood for $\mathbf{w}$, also called *evidence*, $p(\mathbf{w})$ is the prior distribution over the weights, before seeing any data and $p(\mathbf{T})$ is a *normalization* constant ensuring that the posterior distribution over the weights integrates to one.

New equations for λ, β as well as for the *log-evidence* for a given σ_i are derived in section 5.3 of [Sve98]. These equations and derivations will not be listed here, since they are rather complex and the Baysian approach for estimating the parameters was not used in this thesis. However, the algorithm is to be shown due to its ability to approximate the parameters to their close to optimal values at relatively low computational cost. The algorithm is as follows:

This is done for a range of σ_i values and the one with best recorded log-evidence is chosen for the model training. The other parameters are estimated as shown in the algorithm.

for *i=1 to I* **do**
  initialize GTM using σ_i
  repeat
    repeat
      optimize $\mathbf{W}$ by EM-algorithm with λ and β kept fixed
    until *stop criterion for* $\mathbf{W}$ *is met*;
    re-estimate λ and β, using the provided equations
  until *stop criterion for* λ *and* β *is met*;
  record the log-evidence for σ_i
end

4.3 Random forests

Random forests was introduced by [SB01] as an *ensemble* machine learning method for supervised classification and regression. A brief overview of *Random Forests (RF)* will be presented here in the context of classification by the means of *decision trees* [Qui86].

4.3.1 Decision trees

Decision trees are trees with decision nodes, where splitting a training dataset with class labels is done gradually by discriminating the samples by the values of some of their features. If the dataset has N samples and each of these samples has M features, at each node of the decision tree one or more features are to be tested to make a *split* of the dataset. If all the training samples in the node are from the same class or no more splitting can be done, the procedure stops and the reached node is called a *leaf* and labeled with the most *populous* class. When all training samples end up being part of leaves, the *growing* of a decision tree is done.

Determining the splits and the features used for the split could be done in different manners, however most of them try to minimize the impurity of the resulting subsets adopting some *impurity measure*, such as *Gini index*, and making splits according to gains in that measure. Tree nodes can have any number of children, though binary decision trees are most common. Growing full decision trees may cause overfitting, therefore there are different pruning methods [Min89] defined to deal with that problem. Decision trees are not limited to classification tasks and could also be adapted for regression tasks.

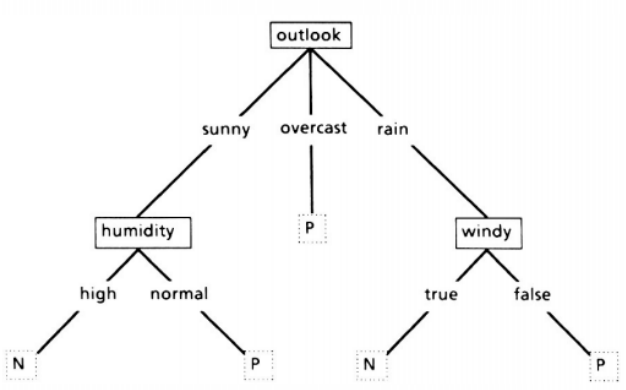


Figure 4.8: A simple decision tree for data vectors with three categorical variables: outlook, humidity and windy. Taken from [Qui86]

4.3.2 Bagging

Bagging is an ensemble method [Bre96] improving the *stability* and the *accuracy* of *unstable* machine learning algorithms. It also decreases overfitting of the trained models, since it is similar to *model averaging*. The principle of bagging is to *bootstrap* a number of datasets D out of the existing N samples. A size $n \leq N$ of the bootstrapped datasets is defined and by drawing randomly n samples from the original dataset with replacement, i.e. possibly duplicating samples, all D datasets are created.

After having D datasets, D models, e.g. decision trees, are trained using these datasets. Then *unseen* data is classified using all trained models, where the class label is determined by choosing the most populous choice among the models.

4.3.3 Random feature selection

Random feature selection is based on choosing randomly m *distinct* features out of M features to be used by the decision tree for the split at given node, where $m \ll M$. For every node, a new set of m features are to be chosen, which could be *repeated* among different splits.

4.3.4 Random forest construction

Constructing a *random forest* is done by combining both *bagging* and *random feature selection* methods, described in the previous subsections. The algorithm proceeds as follows. Firstly, a number of decision trees D has to be chosen. Then a dataset for each of these trees is bootstrapped by the bagging procedure. Then the tree is *fully grown* by random feature selection method and *not pruned*. Later on, classifying an unseen sample involves classifying it using all the trees grown and labeling it with the most populous choice among the classifiers.

4.4 Discussion

The machine learning methods used in the thesis have been introduced in the current chapter. Two unsupervised methods for modeling the data distribution have been presented, one *heuristically* derived (SOM) and one derived using *Baysian* setup (GTM).

The first one is very easy to understand and implement; it is also very popular among image processing algorithms for different purposes, as it has been used extensively for more than two decades. On the other hand GTM is harder for implementation due to its probabilistic nature, requiring computations of very small *posterior probabilities*, that may cause computational *floating point underflow*. This problem becomes increasingly prominent as the dimensionality of the feature vectors grows. However, modeling the data distribution with GTM avoids some of the drawbacks of using SOM, described in details in [Sve98].

The third machine learning method (RF) is a supervised one and requires class labels to be given prior the training. Therefore the trained model is dependent on the class labeling and if different labeling is given, a new model should be trained. All of the three machine learning methods are used in the current work for specific purposes. Their usage is shown in the successive chapter.

Chapter 5

State of the art in fingerprint quality metrics

This chapter is focused mainly on two quality measurement algorithms, that are to be used as a reference for the new proposals presented in section 7. There are a couple more well-performing algorithms that will be also mentioned briefly as part of the *state of the art*.

5.1 Introduction

Recently biometric quality measurement algorithms have drawn the attention of the international biometric scientists community. Fingerprint *quality assessment* algorithms have been developed and moves towards standardization have been made. After the successful development of NFIQ, development of a second version NFIQ 2.0 [NIS12a] have been started by NIST collaboratively with several German institutions, namely BSI, BKA, Fraunhofer IGD, CASED (Hochschule Darmstadt), Secunet Security Networks AG.

Definitions of state of the art *quality features* have been presented in ISO/IEC 29794-4 [ISO12c] as well as in *NFIQ 2.0 Feature Definitions Document* (currently v0.5) [NIS12b]. Their evaluated performance has been shown in *NFIQ 2.0*

Feature Evaluation Document (currently v0.5) [NIS12a]. According to ISO/IEC 29794-4 feature analysis methods could be split into two categories: *global features analysis* methods, that analyze the sample as a whole in order to produce the quality score; *local features analysis* methods, that analyze blocks of the sample and then producing the score as a result of the aggregated output of *block-wise analysis*.

5.2 Orientation Certainty Level

Orientation Certainty Level (OCL) is a local feature analysis method for quality estimation introduced by [LJY02]. It is part of both ISO/IEC 29794-4 [ISO12c] and *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b]. This method has grounds in the observation that good quality samples have well defined ridges and valleys. If taken block-wise, ridge and valley lines have the same *orientation* and determining that orientation for good quality samples could be done with *high certainty*, unlike for poor quality samples. Therefore the measured *orientation certainty* is used as *quality feature* for each block, which are later aggregated into a single *quality score*.

The procedure is defined in both ISO/IEC 29794-4 [ISO12c] and *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b], where block size of 32×32 pixels is suggested. However, NFIQ 2.0 has one difference from ISO/IEC 29794-4 and that is the computation of OCL, where ISO/IEC 29794-4 indicates with 0 the strongest OCL and with 1 the weakest, which is inverse for NFIQ 2.0. For each block b_j the NFIQ 2.0 procedure is as follows:

1. Compute the *block-wise intensity gradient* by applying a 3×3 *Sobel* operator on the block image. The result for each pixel $p_i = (x, y)$ is the gradient (dx, dy), representing the strength and the direction of the orientation at that pixel.
2. Compute the *covariance matrix* C by iterating over all N pixels in the block
$$C = \frac{1}{N} \sum_{p_i} \left\{ \begin{bmatrix} dx \\ dy \end{bmatrix} \begin{bmatrix} dx & dy \end{bmatrix} \right\} = \begin{bmatrix} a & c \\ c & b \end{bmatrix} \tag{5.1}$$
3. Compute the *eigenvalues* α_{min} and α_{max}
$$\begin{aligned} \alpha_{min} &= \frac{a + b - \sqrt{(a-b)^2 + 4c^2}}{2} \\ \alpha_{max} &= \frac{a + b + \sqrt{(a-b)^2 + 4c^2}}{2} \end{aligned} \tag{5.2}$$

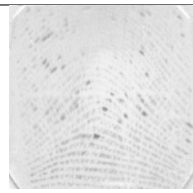
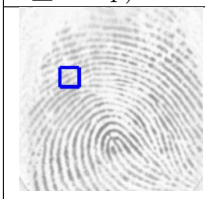
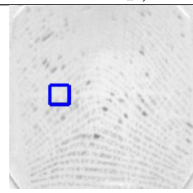
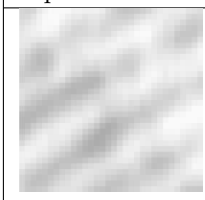
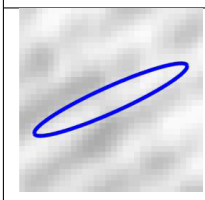
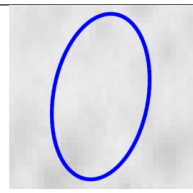
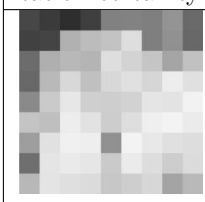
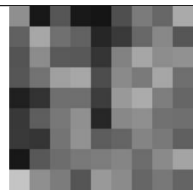
Processing steps for Orientation Certainty Level	
	
Input (FVC2000Db1 1_1.bmp)	Input (FVC200Db1 100_1.bmp)
	
Input with current block marked	
	
Zoomed view of block	
	
Ridge orientation indicated as an ellipse. The eccentricity of the ellipse corresponds to the orientation certainty.	
	
Orientation certainty level in each block. High intensity corresponds to high level of certainty.	

Table 5.1: Computing the orientation certainty level. Taken from [NIS12b].

4. Compute the *orientation certainty level* OCL

$$b_{OCL} = 1 - \frac{\alpha_{min}}{\alpha_{max}} \tag{5.3}$$

Once the computing of the OCL for all N_b blocks is done, the quality score is computed as:

$$Q_{OCL} = \frac{1}{N_b} \sum_{p_i} b_{OCL} \tag{5.4}$$

5.3 NIST Finger Image Quality

NIST Finger Image Quality (NFIQ) was proposed in [TW05] and is currently the best performing QMA according to the *NFIQ 2.0 Feature Evaluation Document (v0.5)*. The output of the QMA is within the range $1-5$, where 1 is the highest quality level and 5 is the lowest quality level. NFIQ is very robust and works good for wide range of comparison algorithms. It is also performing well in terms of execution time for estimation, thus it could be used in live-fingerprint scanners [TW05] [NIS12b].

Its working principle is based on extracting a number of different features by analyzing the fingerprint image with *NIST Fingerprint Image Software (NFIS)* [NIS] to produce a *feature vector*. Later on, this feature vector is given as an input to an artificial neural network to classify the image to one of the five quality levels. The neural network has been trained on subset of images from five datasets with different capture conditions. The procedure of producing a quality score from a fingerprint sample is as follows:

1. Generate *image quality map* by generating several image maps and then combine the information from all the maps into one. The map is divided into regions, each of them with assigned *quality level* from $0-4$, where 4 indicates highest quality. Blocks with quality 0 are regarded as *background blocks* and the total fraction of *foreground blocks* out of all blocks is calculated. Then the percentage of the foreground quality blocks with quality 1, 2, 3 and 4 are computed, the resulting percentages are called *quality zones* $1-4$. Here is a list of image maps used to generate the quality map:

- ***Direction map***: this map is produced by using *Discrete Fourier Transform (DFT)* in several orientations of the block, rotated incrementally. Rotation of the block is done within a larger *window* overlapping with the neighboring blocks for minimizing *discontinuity*.
- ***Low contrast map***: to produce this map the *pixel intensity distribution* within the surrounding window of the block is computed. Then blocks with *narrow dynamic range* (defined by an empirically determined threshold) are *flagged* as low contrast, therefore unreliable for minutiae detection.
- ***Low flow map***: this map shows the areas that could not be assigned with a *dominant ridge flow*, where detected minutiae are not reliable.
- ***High curve map***: this map shows the blocks, where *high curvature* exists. Areas with *high curvatures* are also not reliable for minutiae detection.

2. Extract minutiae by using NFIS and estimate their quality by calculating the mean and the standard deviation of the area they are detected. A good minutiae should have high standard deviation and medium mean pixel intensity. NFIS assigns a quality within the range of $0.01 - 0.99$ of each minutiae based on that assumption.

3. Produce 11-dimensional feature vector as described in table 5.2

	Name	Description
1	*foreground*	number of block that are quality 1 or better
2	*total number of minutiae*	total number of minutiae found in the fingerprint
3	*min05*	number of minutiae having quality 0.5 or better
4	*min06*	number of minutiae having quality 0.6 or better
5	*min075*	number of minutiae having quality 0.75 or better
6	*min08*	number of minutiae having quality 0.8 or better
7	*min09*	number of minutiae having quality 0.9 or better
8	*quality zone 1*	percentage of the foreground blocks with quality = 1
9	*quality zone 2*	percentage of the foreground blocks with quality = 2
10	*quality zone 3*	percentage of the foreground blocks with quality = 3
11	*quality zone 4*	percentage of the foreground blocks with quality = 4

Table 5.2: Description of the NFIQ 11-dimensional feature vector. Taken from [TW05].

5.4 Other QMAs

The above-described methods have been chosen to be used as *reference methods* for evaluating the performance of the proposed approaches in this thesis. However, there are several more QMA methods presented in *NFIQ 2.0 Feature Definitions Document* and some of them will be briefly discussed here.

5.4.1 Gabor Shen

Gabor Shen [SKK01] is one of the best performing methods and it is based on applying a *Gabor filter* [FS89]. The image is separated into background and foreground blocks and then the foreground is separated on *good* and *bad* blocks. The quality score is the ratio of poor blocks in the foreground. *Gabor Shen* is part of *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b].

5.4.2 Local Clarity Score

Local Clarity Score (LCS) is a local analysis method that computes the block-wise *clarity* of ridge and valleys by using *linear regression* to determine a pixel intensity threshold. Then pixels are classified as either ridge or valley and compared with the normalized ridge and valley width of the block, looking for misclassifications. Then ratio of misclassified pixels in the block is calculated for each block. The method is described in *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b] as LCS and in ISO/IEC 29794-4 [ISO12c] as *Ridge-valley Structure Analysis*.

5.4.3 Ridge Valley Uniformity

The *Ridge Valley Uniformity* is defined in both ISO/IEC 29794-4 [ISO12c] and *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b] and is a measure of *consistency* of the ridge and valley widths. It is a global analysis method calculating the *standard deviation* of the ratio of *ridge width* to *valley width* from the mean ratio, where lower deviation indicates a better quality image.

5.4.4 Orientation Flow

Orientation Flow is a global analysis method and it has found its place in ISO/IEC 29794-4 [ISO12c] and *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b]. It is a measure of the *ridge flow continuity* by calculating the absolute *orientation difference* between a block and its neighbors. If there is small difference (usually there is a small angular tolerance), the block is given high *orientation quality score*. By increasing the difference, the *orientation quality score* drops. The final score is obtained as the mean block-wise orientation quality scores of all blocks.

5.5 Discussion

This chapter has provided basic overview of the state of the art in fingerprint quality metrics. There are several well-performing QMAs, that are standardized in the specifications of ISO/IEC 29794-4 [ISO12c] and *NFIQ 2.0 Feature Definitions Document (v0.5)* [NIS12b]. However, NFIQ 2.0 is under development and the list of feature definitions is open for new proposals. The research conducted on the topic and the resulting proposed methods are meant to contribute to the research for finding new candidates that are to be incorporated in the ISO/IEC or NFIQ 2.0 standards.

Chapter 6

Related work

Previous works have used *Self-Organizing Maps* to model the distribution of the image data of fingerprint samples and then use the trained *unsupervised model* to estimate the quality of *unseen* samples. Two approaches have been applied for that purpose: the first one is by considering the whole image as the input vector for the training of the model and the second one is to divide the image into blocks. Both of them will be briefly introduced in this chapter.

6.1 Holistic approach

Self-Organizing Maps is a very popular unsupervised learning method, that has proved to be very useful for *clustering* and *data visualization*. In the field of image analysis, one of its best uses is for *texture analysis* [KSH01]. Therefore using it in the field of fingerprint image sample analysis is a possible good application, since fingerprint samples have very specific textures caused by the ridge impressions. Anton Makarov et al [Mak12] have experimented with modeling the *whole-image* data of a sample by applying SOM and then classifying the new samples according to their *best-matching unit*. Since SOM is a mapping of similar samples to the same node, then it is assumed that the unseen data, being mapped to a given node, would have similar features as the previously

mapped training data to that node. Thus the biometric performance of these samples is expected to be similar as well.

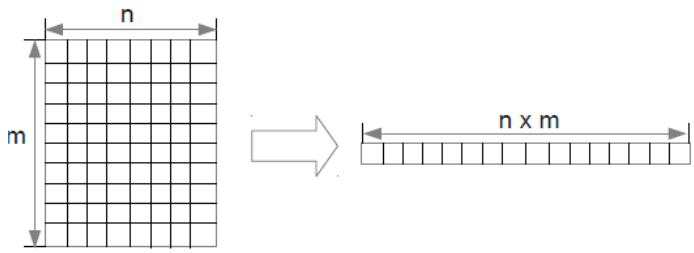


Figure 6.1: Feature vector construction. Taken from [Mak12]

The *holistic* approach is based on the following principle: the *whole-image* data is taken *row-wise* to construct a single feature vector consisting of the pixel values. Then a set of samples are used for the training of a randomly initialized hexagonal SOM map with a predefined network sizes, e.g. 16×16, 24×24, etc. The quality scores of the training samples are calculated by *binning* the observed utility score and after the training has finished for each node a quality class is assigned by the *mean quality score* of all training samples mapped to that node. In the performed experiments, also "*winner takes it all*" assignment was tried, but did not show better results than the mean assignment. Also image segmentation for *background* and *foreground* separation was applied is some of the experiments, so the results could be compared with no image preprocessing at all.

The main goal of the research was to prove that SOM is capable to provide a mapping, such that it would be useful for predicting quality. Therefore a validation set was used to produce the results. As seen from figure 6.4, the samples predicted by the model to be of low quality consist of more samples with low observed utility (left plot); respectively, the samples that are predicted to be of high quality consist of more samples with high observed utility (right plot). That gave promising results on the CASIAFPV5 dataset [oSIoA] and fueled further research of the usage of SOM for modeling fingerprint image data.

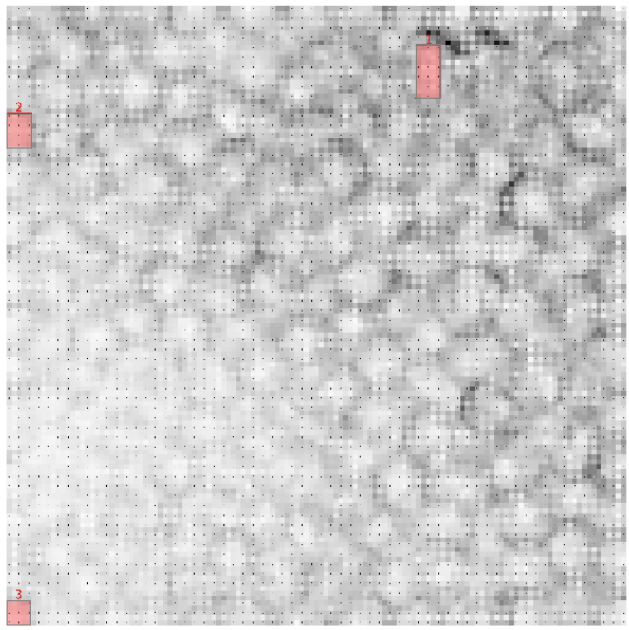


Figure 6.2: U-matrix of the trained SOM map of size 64×64 with the holistic approach. Taken from [Mak12]

(a) Group 1 **(b)** Group 2 **(c)** Group 3

Figure 6.3: Fingerprint images mapped to nodes in the trained SOM map, whose U-matrix is shown in figure 6.2. Taken from [Mak12]

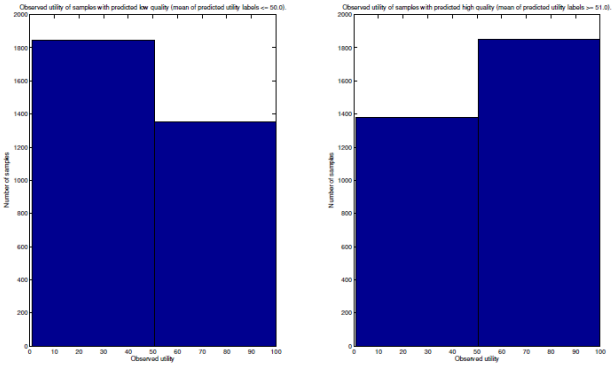


Figure 6.4: Histogram of the results binned into low and high predicted quality scores from Makarov's holistic approach research: left plot shows histogram of samples with low observed quality; right plot shows histogram of samples with high observed quality. Taken from [Mak12]

6.2 Block-wise approach

Even though the holistic approach have shown promising results, the trained models were not able to capture some specific differences in the images. The *global analysis* of the images leads to separating them into different groups, but often these groups have large inter-group variations as it could be observed in figure 6.3b. Therefore, the next step of the research was done by Martin Olsen [OTMB13] and it was based on *block-wise fingerprint image division* and training a *randomly initialized SOM model* that has to capture the block images data. Before dividing images into blocks, they were normalized using pyramid decomposition [FKB08].

Then each of the nodes of the trained SOM of size, say 24×24, has been given an *index* number. Therefore, each block of the fingerprint image sample is assigned with its *best-matching unit's index* number.

Having each block of the image labeled with a number, a vector of numbers can be constructed. Then, for this vector the *normalized histogram* could be computed, i.e. another feature vector with the length of the number of SOM nodes, containing the *normalized histogram* is constructed. One might expect that the histograms of low quality and high quality samples will be different and this expectation has proven to be true, as seen in figure 6.8.

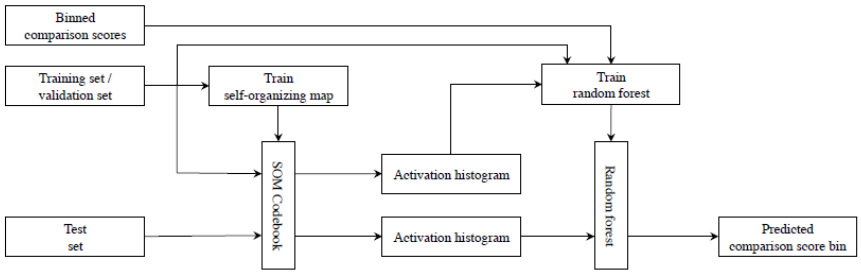


Figure 6.5: Overview of the block-wise system architecture. Taken from [OTMB13]

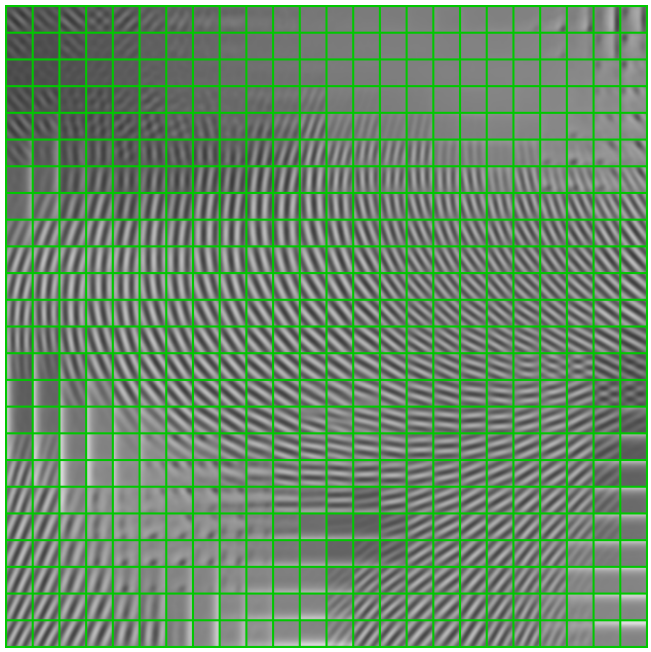
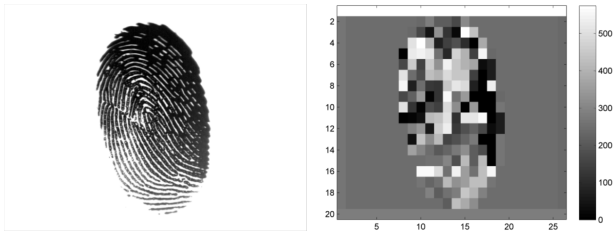


Figure 6.6: Visualization of the nodes of the trained SOM map of size 24×24 for blocks of size 24×24. Taken from [OTMB13]



(a) raw image **(b)** block-wise best-matching unit indices

Figure 6.7: Illustration of relationship between finger image and SOM codebook shown in figure 6.6. Taken from [OTMB13]

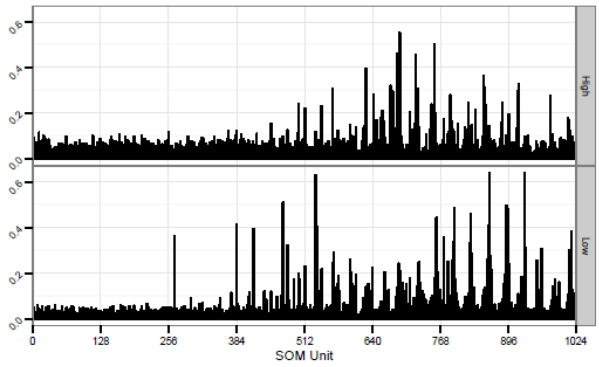


Figure 6.8: Aggregated histograms of the *Low* and *High* comparison scores sets of 64 images each. Taken from [OTMB13]

Therefore these *histogram feature vectors* are used for training a supervised *Random Forest (RF)* model to classify them into five quality classes, determined by the produced comparison scores. Once the two models (the SOM and the RF) are trained, the same feature extraction procedure is done using the SOM model for the validation set and the constructed feature vectors are used to classify the samples into one of the five quality classes by using the RF model. The procedure was applied on a NIST operational dataset and the method has shown very good results, surpassing the performance of the *Orientation Flow* method defined in [NIS12b].

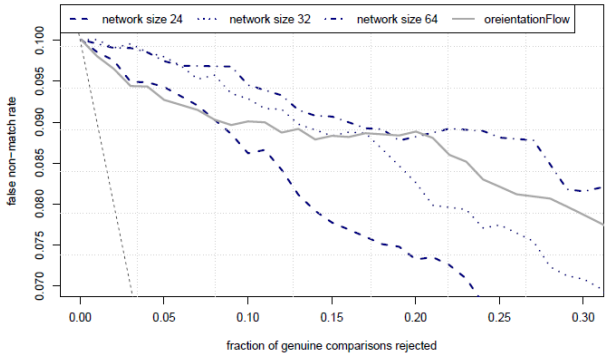


Figure 6.9: ERC plot showing the performance of the method with different SOM map sizes compared to the performance of *Orientation Flow*. Taken from [OTMB13]

6.3 Discussion

The novel approach of modeling fingerprint image data with *Self-Organizing Maps* has shown very good results. The *holistic* approach has shown the capability of SOM to model fingerprint image data, where the *block-wise* approach has applied these findings and has produced results, comparable with some of the best state of the art methods.

However, the RF model is a supervised training model and *precomputed* target classes are used for the training. Since the target classes are computed out of the comparison scores produced for each sample by a *specific* comparison algorithm, the trained RF model should be *retrained* for every other comparison algorithm that is to be used. Also the training of the underlying SOM model is done by random initialization of the codebook nodes, so a linear initialization could

be introduced in the hopes of improving the obtained results. Moreover the experiments should be repeated and the results analyzed, so a better view for the direction of further studies could be gained. Therefore, the goal of this thesis is to deal with all these aspects and try to contribute to the improvement of the developed method.

Chapter 7

Proposed two-tier approach

This chapter describes the proposed *two-tier* approach for quality estimation, inspired by the explained related works in the previous chapter. According to the obtained results, the block-wise approach shows good results, therefore repeat of the experiment and further development of the method is presented here.

7.1 Processing pipeline

The overview of the proposed approach is presented here. The input of the algorithm is a list of gray-scale fingerprint images and their corresponding averaged genuine comparison scores as a column vector $\mathbf{y}$. For the purpose of modeling and testing the algorithm, the data is divided into *training* and *validation* set. If desired or needed a prior image preprocessing may be performed on the fingerprint images, e.g. *pyramid decomposition*, *normalization*, etc.

The algorithm is divided into two phases. For the *low-level phase*, the images are cut into blocks and a model is trained over the image blocks of the training set of images. Then by using the trained model, feature vectors are extracted for each fingerprint image in the dataset. These features are further processed by

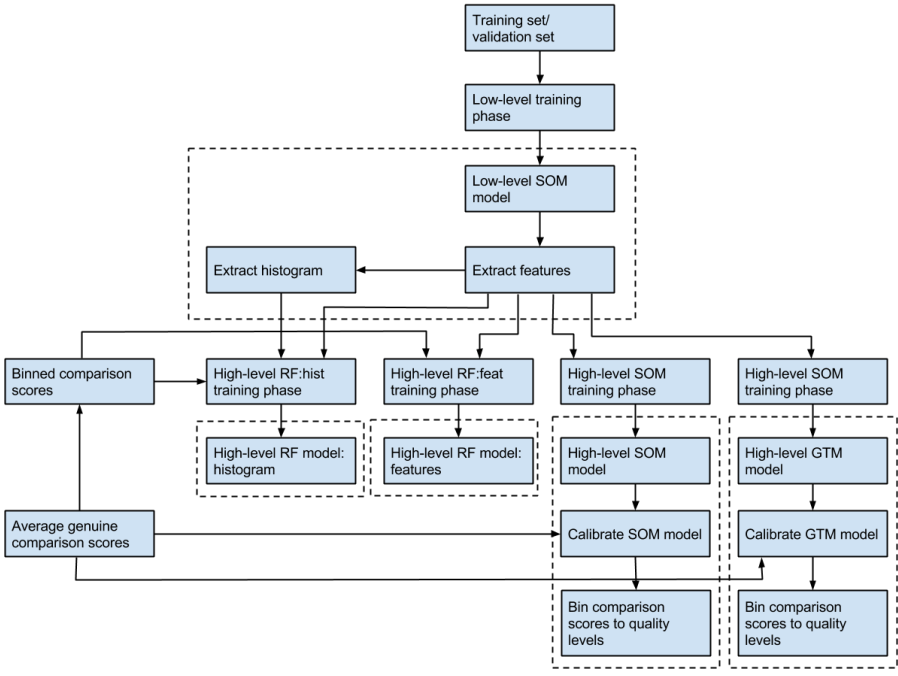


Figure 7.1: Overview of the training process pipeline. Each process or data source is separated in a blue block and the communication between processes is denoted as arrows. Some of the processes are grouped to form a big dashed blocks which are used for unseen data in the prediction mode.

the *high-level*, where another model is trained using the features of the training set to *interpret* them into *quality levels*. The *high-level* model is used to predict quality levels over the whole dataset.

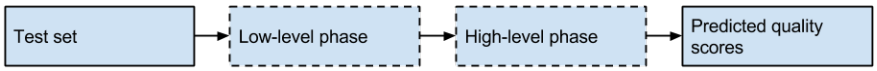


Figure 7.2: The process of quality score prediction. Each of the dashed blocks corresponds to the processes grouped together in figure 7.1.

7.2 Low-level phase

As already mentioned, the *low-level* processing of the training data results into a trained model. This model is used for extracting the feature vectors of both training and validation data. The chosen model is a SOM one and the algorithm for training it and extracting the features is as follows:

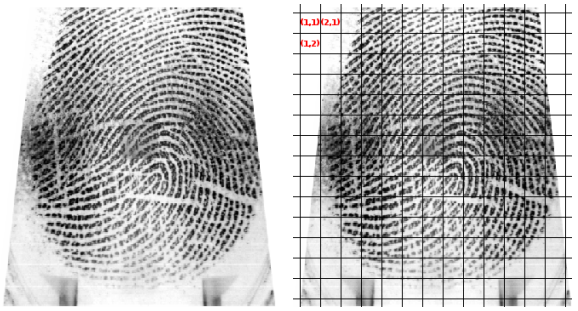
1. Given the list of fingerprint images meant for training and the block size b_{size}, all the training images are divided into square blocks of that size. Depending on the image size, different number of blocks are produced along each of the two dimensions $N_b^{(x)}$ and $N_b^{(y)}$.

2. For each block $b_I^{(x,y)}$, its fingerprint image source I and spatial index (x, y) are kept, so later this information could be used in feature construction for the high-level modeling.

3. All image blocks are converted into row-vectors $\mathbf{v}_I^{(x,y)}$ of length $b_{size} \times b_{size}$ by taking the pixel values of the blocks row-wise as previously shown in figure 6.5.

4. All N_{train} training vectors are combined in a matrix T. No special ordering of the vectors is required, though random ordering is preferred.

$$T = \begin{vmatrix} \mathbf{v}_I^{(x,y)} \\ \mathbf{v}_I^{(x+1,y)} \\ \cdots \\ \mathbf{v}_{N_{train}}^{(N_b^{(x)},N_b^{(y)})} \end{vmatrix} \tag{7.1}$$



(a) raw image **(b)** divided image

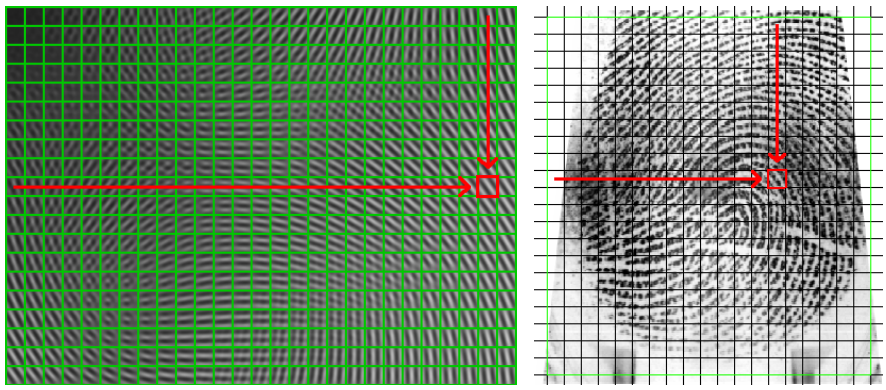
Figure 7.3: Example of dividing a fingerprint image into blocks of size 24×24. The spatial index (x, y) of the block is taken as screen coordinates, i.e. starting from the left top-most block and x stands for horizontal and y for vertical coordinates. Fingerprint sample taken from [oSIoA]

5. Linearly initialize SOM as shown in section 4.1.3.2 by doing a PCA analysis of the data matrix T. For too big matrices to fit in the main memory, an iterative PCA algorithm [Row98] should be used. The number of map nodes M_{low} is determined heuristically by $M_{low} = \left\lceil \sqrt{N_{train}^{(blocks)}} \right\rceil$ in dependence of the total number of training image blocks $N_{train}^{(blocks)}$ and is upper bounded by a manually chosen number (e.g. 4000 in the current thesis), such that a trade-off between computational speed and SOM map smoothness for big datasets is achieved.

6. Stepwise SOM training process is performed with previously determined parameters: initial and final $\alpha(t)$ and $\sigma(t)$, which define the neighborhood function and the training length. These parameters could be either heuristically determined or by cross-validation over the training data.

7. After the model has been trained, each block vector $\mathbf{v}_I^{(x,y)}$ is propagated through the network, so the best-matching unit is determined and its coordinates on the SOM map produce a two-dimensional coordinate row-vector $\mathbf{c}_I^{(x,y)}$ for each block of the image. See figure 7.4.

8. For each image I all blocks' coordinate vectors $\mathbf{c}_I^{(x,y)}$ are concatenated, where block indices in the image are taken row-wise. Thus, for each image I a feature vector $\mathbf{f}_I$ of length $2 \times N_b^{(x)} \times N_b^{(y)}$ is constructed.



(a) the BMU on a part of the SOM map

(b) propagated block from the fingerprint image

Figure 7.4: The best-matching unit on the SOM map (left) and the block of the fingerprint image propagated through the map (right). The coordinates of the best-matching unit from figure (a) are saved in a coordinate row vector $\mathbf{c}_I^{(x,y)}$, indexed by (x, y), which are the coordinates of the block itself on the fingerprint image from figure (b).

7.3 High-level phase

The *low-level* model is used to extract feature vectors $\mathbf{f}_I$ for each image. These feature vectors are used to assign a *quality level* for each image. However, in order to construct the model labeling each feature vector to a quality level, one has to train that model. There are several approaches used in this thesis for the *high-level* modeling and they will be described here.

In order to quantize the results of the algorithms, a binning procedure for converting from *genuine comparison scores* into *quality scores* must be performed. That is done by defining a cumulative distribution function $C(.)$ of the *observed genuine comparison scores* of the training set. Then a number of levels L is chosen and $L - 1$ thresholds in the range $0 - 1$ are defined. Using the inverse cumulative distribution function $C^{-1}(.)$, these fraction thresholds are converted into genuine comparison score thresholds. Having these thresholds and a given genuine comparison score, a quality level could be obtained.

7.3.1 Self-organizing maps

Similarly to the *low-level* training procedure, for the *high-level* a SOM model is to be used for modeling the feature vectors $\mathbf{f}_I$ input data. Since the datasets are usually consisting of tens of thousands samples and each feature vector for block sizes higher or equal to 16×16 are of length up to couple of thousand features, the batch SOM algorithm could be used without requiring too much RAM for the training. Thus, the learning rate $\alpha(t)$ becomes obsolete and only the training length and the neighborhood range $\sigma(t)$ should be determined either by cross-validation or heuristically.

After having chosen the right parameters, a final SOM model is trained with the whole training data. The SOM model is linearly initialized prior to the training and the number of nodes M is heuristically determined by $M = \lceil 5 \times \sqrt{N_{train}} \rceil$. The nodes of the constructed SOM map are then each labeled with the *mean average genuine comparison scores* of the image samples, that are mapped to that node. This labeling process is called SOM map *calibration*. Thus, by propagating a feature vector through the SOM map and assigning an estimated *average genuine comparison score*, a quality score could be determined using the already defined binning procedure.

7.3.2 Generative topographic mapping

Modeling the feature vectors $\mathbf{f}_I$ data by GTM is done as shown in section 4.2. Parameters like the number of grid nodes K and the number of radial basis functions M are determined heuristically. σ and λ could be determined either heuristically or by cross-validation, while $\mathbf{W}$ and β are estimated by the EM-algorithm. All the parameters might be estimated using the Baysian approach, however in the current thesis all parameters are determined heuristically.

The training process is the process of estimating $\mathbf{W}$ and β, given all other parameters and the training data feature vectors. The final GTM model is constructed once the EM-algorithm converges and after that a calibration of the GTM map is performed. Since GTM is a mixture of Gaussians, each node $\mathbf{x}_i$ of the GTM map is labeled by computing its *expected average genuine comparison*

score by calculating the *Gaussian components' means* z_i using

$$z_i = \frac{\sum_n s_n p(\mathbf{x}_i \mid \mathbf{t}_n, \mathbf{W}, \beta)}{\sum_n p(\mathbf{x}_i \mid \mathbf{t}_n, \mathbf{W}, \beta)} \tag{7.2}$$

where s_n is the *average genuine comparison score* for training sample $\mathbf{t}_n$ and n is iterating over all N_{train} training samples. After having all z_i values, one could predict the *expected conditional mean* $y(\mathbf{t}_n)$ of an arbitrary sample $\mathbf{t}_n$ by computing the convex combination of the components means z_i in the output space

$$y(\mathbf{t}_n) = \sum_i^M z_i P(\mathbf{x}_i \mid \mathbf{t}_n, \mathbf{W}, \beta) \tag{7.3}$$

where $P(\mathbf{x}_i \mid \mathbf{t}_n, \mathbf{W}, \beta)$ is the calculated responsibility R_{in}. This way an *estimated average genuine comparison score* $y(\mathbf{t}_n)$ is calculated and could be converted into a quality score using the defined binning procedure.

7.3.3 Random forests: features

Another approach for the *high-level phase* is to use RF to model the data. RF is a supervised learning model and it requires not only the feature vectors, but also the output quality levels of the data prior the training. Since different comparison algorithms result in different *average genuine comparison scores* for the training samples, for each comparison algorithm a new model should be trained. A classification RF is used, so the binning of the genuine comparison scores is done prior the training as well. Then a RF model is trained, where the parameters m and n are defined by either cross-validation or heuristically, like in the current thesis.

7.3.4 Random forests: histogram

The feature vector $\mathbf{f}_I$ is only one way of representing the information retrieved by the *low-level phase*. Another way is by constructing a histogram of the SOM hits for each image. Each node of the SOM map is assigned with an index in

the range $1 \dots M_{low}$, where nodes are counted column-wise. Then each pair of coordinates in $\mathbf{f}_I$ is converted into index, resulting into a vector of indices ind_I of length $N_b^{(x)} \times N_b^{(y)}$. The values in that vector could be summarized in a histogram $\mathbf{h}_I$ with bins $1 \dots M_{low}$.

Using the feature vector $\mathbf{h}_I$ for training the RF classification model could be considered as another *high-level* modeling approach. This approach is the same as the one described in section 7.3.3, except that the input data for this one is the vector $\mathbf{h}_I$ and for the other one is $\mathbf{f}_I$. By using the histogram features the information of the spatial coordinates of the blocks is lost and only their best-matching units are considered. However, it is not trivial to state whether this information is important or not for constructing better models, therefore the two approaches are considered in the current thesis.

Chapter 8

Experimental setup

This chapter deals with the details for the experimental setup used for applying the proposed approaches. It provides information about the hardware and software used, datasets experimented on and specific configuration parameters for the trained models. Some implementation issues will be discussed as well.

8.1 Hardware configuration

Hardware used for the experiments is a Windows machine with the following properties:

- 2x quad-core processor CPU AMD Opteron 4130 2.60GHz
- 16GB RAM
- Windows Server 2008 R2 Datacenter SP1 64-bit
- HDD with more than 300GB free space

8.2 Software used

8.2.1 Matlab

MATLAB [MAT12] is a computer software and a programming language for implementing algorithms. It is especially useful for mathematical computations with matrices. The proposed methods from chapter 7 are implemented in Matlab scripts and run in MATLAB R2012a. Most of the work in this thesis is done in Matlab, including data preparation, low-level SOM map linear initialization and results interpretation and visualization.

8.2.2 SOM_PAK

For constructing the low-level SOM maps, the software package SOM_PAK Version 3.1 [KHKL96] is used. This package is providing implementation of the original stepwise SOM algorithm. It is using its own data format for the input and the output of the trainings, so the data should be converted prior to the training. It is written in C and compiled as a separate executable.

8.2.3 SOM Toolbox

The SOM Toolbox 2.0 [VHAP00] is a Matlab implementation of many helpful functions and algorithms, needed for constructing SOM maps. It has an implementation of the batch SOM algorithm and interface functions for interacting with SOM_PAK. This package has been used for initializing and constructing the SOM maps in the high-level phase, where the data matrix is much smaller and able to fit in the memory, so batch SOM is used.

8.2.4 Netlab Toolbox

The Netlab Toolbox 3.3 [Nab02] is a neural network software package for Matlab, developed by a team at *Aston University*. It is a comprehensive Matlab toolbox for machine learning, however in the current thesis only the GTM related functions are used. It has not been optimized for working with small probabilities, therefore the code had to be patched with a logarithmic numerical trick, so the

equation (4.10) is transformed as follows:

$$R_{in}\left(\mathbf{W}_{old}, \beta_{old}\right) = \frac{\exp\left\{\log p(\mathbf{t}_n \mid \mathbf{x}_i, \mathbf{W}_{old}, \beta_{old}) - A_{max}^{(n)}\right\}}{\sum_{i'=1}^{K} \exp\left\{p(\mathbf{t}_n \mid \mathbf{x}_{i'}, \mathbf{W}_{old}, \beta_{old}) - A_{max}^{(n)}\right\}} \tag{8.1}$$

where $A_{max}^{(n)}$ is computed as

$$A_{max}^{(n)} = \max_i \left\{\log p(\mathbf{t}_n \mid \mathbf{x}_i, \mathbf{W}_{old}, \beta_{old}\right\} \tag{8.2}$$

This way the mathematical underflow is avoided and the responsibilities computed without the need of increasing the precision of the data types in Matlab.

8.2.5 R

R is a language and environment for statistical computing [R C13] and in this thesis R scripts are used for training the RF models due to existence of legacy code from previous experiments with RF models. R provides very large number of packages for dealing with linear and non-linear modeling, classification, clustering, etc. Therefore it is often used by scientists for performing experiments with any kind of input data.

8.3 Datasets

8.3.1 CASIAFPV5

CASIA Fingerprint Image Database Version 5.0 (CASIAFPV5) [oSIoA] is collected by the *Chinese Academy of Sciences' Institute of Automation* and all fingerprint images are 8-bit gray-scale BMP files with resolution 328×356. The dataset consists of 20,000 fingerprint samples, taken from 500 volunteers contributing 40 fingerprint images each. 5 fingerprint samples were captured from all the fingers, but the pinkie, on both of their hands using URU4000 optical fingerprint sensor in one session. The fingerprint samples are with significant intra-class variations, caused by rotation, dislocation and high or low pressure.

The training set is composed by choosing randomly 12,000 fingerprint samples of the dataset and the rest 8,000 are used as a validation set.

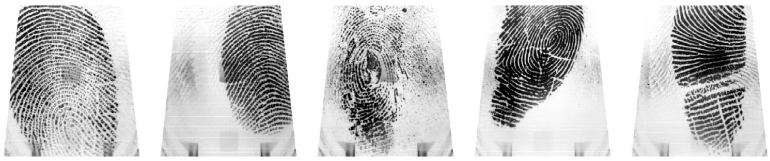


Figure 8.1: Example fingerprint image samples taken from [oSIoA]

8.3.2 MCYT330-DP

MCYT330-DP is a subset of the dataset of *Ministerio de Ciencia y Tecnologia (MCYT)* 330 bimodal Fingerprint subcorpus [OGFAS+03]. For collecting the dataset 330 individuals have provided 10 fingers and 12 acquisitions with an optical sensor UareU from *Digital Persona*. The dataset consists of 39,600 fingerprint samples with resolution 296×400 gray-scale BMP files. For this dataset much smaller intra-class variations are observed, unlike CASIAFPV5. The training set is composed of randomly drawn 23,760 samples from the full dataset. The remaining 15,840 samples are used as a validation set.

Figure 8.2: Example fingerprint image samples taken from [OGFAS+03]

8.4 Low-level

Training for the low-level is done for the training set of each dataset separately. Two models per dataset are constructed: one for block size 16×16 and one for block size 24×24. After the images are cut into blocks, the number of 16×16 training block samples for CASIAFPV5 is 5,280,000 samples and for MCYT330-DP is 9,504,000 samples. The number of 24×24 training samples is 2,184,000 and 3,801,600 for CASIAFPV5 and MCYT330-DP respectively.

The training is done by using SOM_PAK, where the starting parameter $\alpha(0) = 0.05$, the neighborhood function is a Gaussian one with initial $\sigma(0) = 2 \times unit_{dist}$. After the linear initialization of the SOM map $unit_{dist}$ is calculated by taking the diagonal distance of the map in data space and dividing it by the number of nodes along the longest side of the map.

The total number of nodes and the SOM map dimensions are determined after its linear initialization as shown in section 7.2. The number of training iterations for block size 16×16 is 5,000,000 for both databases and for block size 24×24 is 2,100,000 and 3,800,000 for CASIAFPV5 and MCYT330-DP respectively.

Dataset	Block size	Training set	Training length
CASIAFPV5	16×16	5,280,000	5,000,000
CASIAFPV5	24×24	2,184,000	2,100,000
MCYT330-DP	16×16	9,504,000	5,000,000
MCYT330-DP	24×24	3,801,000	3,800,000

Table 8.1: Summarized numbers of samples and training length for the four different models.

8.5 High-level

After the low-level model is constructed, both the training and the validation sets are propagated through the trained SOM maps in order to compose the feature vectors that are to be used by the high-level phase of the process. For CASIAFPV5 these feature vectors are of length 880 and 364 for 16×16 and 24×24 respectively. For MCYT330-DP the feature vectors are of length 900 and 384 for 16×16 and 24×24 respectively.

For obtaining results and converting them to quality scores, *average genuine comparison score* is determined for each fingerprint sample by using a single comparison algorithm as a black box for computing all possible *genuine comparison scores* and taking their average value. Since there are three comparison algorithms available at CASED, codenamed as {28}, {63} and {83}, each sample will receive three *average genuine comparison scores*. Therefore the unsupervised models (SOM and GTM) have to be calibrated separately for each comparison algorithm and the supervised models (RF-feat and RF-hist) have to be trained for each comparison algorithm separately. The performance evaluation of all high-level approaches is done by using ERC curves and Correlation tables for each comparison algorithm alone.

8.5.1 SOM

Since for the high-level SOM training the batch SOM algorithm is used over a linearly initialized map, only the number of batch iterations and the initial $\sigma(0)$ are to be set. The number of batch iterations is set to 30 and $\sigma(0) = 2 \times unit_{dist}$, where $unit_{dist}$ is calculated as noted in section 8.4. After the model is trained, three calibrations are done for all three comparison algorithms and evaluated using the validation set of the data.

8.5.2 GTM

For constructing the GTM model the EM-algorithm is used to estimate the parameters $\mathbf{W}$ and β. The size of the grid of sample points is heuristically set to 20×20, resulting in $K = 400$ sample points $\mathbf{x}_i$. The form of the RBF grid is set to 9×9 RBFs, so the number of *radial basis functions* used is $M = 81$. The other two parameters, the *common width* of the radial basis functions σ and the *regularization term* λ, are determined heuristically as well and set to $\lambda = 1$ and $\sigma = 2s$, where s is the spacing of the radial basis functions centers. Similarly to the previous section 8.5.1, three different calibrations are done for the different comparison algorithms.

8.5.3 Random Forests: features and histogram

For constructing the RF model, two parameters are to be set. The first one is the number of trees D to be grown, where each tree uses $n = N$ bootstrapped samples. The second parameter is the number of features to be chosen at each split m. For both RF-feat and RF-hist, these parameters are the same and are set to $D = 500$ and $m = 100$. The only difference between RF-feat and RF-hist is that the first one uses the feature vectors extracted from the low-level as input data, where the second one uses a histogram of these features as defined in section 7.3.4.

Chapter 9

Experimental results

This chapter presents overview of the results from the experiments. Results are presented and discussed separately for the low-level phase, the high-level phase and the final ERC curves and Correlation tables. More figures from the results can be found in the appendix.

9.1 Low-level

The low-level training is performed after dividing the fingerprint images and linearly initializing the SOM map. The results are discussed where evaluation is done in terms of speed and SOM maps visualization.

9.1.1 General statistics

In table 9.1 one could observe the most important numbers taken from the experiments. In all time calculations I/O time is included, therefore it is not easy to state whether the method can be considered as fast or not. However, the best-matching unit calculation time (BMU time) is the one of the highest

importance, since it is the one determining the operational time for calculating the quality score of an unseen sample. As it can be seen from the table, for all of the tested configurations, the time per image is more than $2s$. Combined with the image dividing time, which is always more than $1s$, at least $3s$ are needed for the process of constructing the feature vector of a fingerprint image. Obviously this is not an acceptable speed, since total processing times more than $1s$ may cause user frustration.

	CASIAFPV5		MCYT330-DP	
	16×16	24×24	16×16	24×24
Blocks (per image, $w \times h$)	20×22	13×14	16×25	10×16
Dividing time (per image)	2.62s	1.38s	3.34s	1.45s
Training set (blocks)	5,280,000	2,184,000	9,504,000	3,801,600
Validation set (blocks)	3,520,000	1,456,000	6,335,200	2,534,400
Map size ($w \times h$)	27×148	28×143	26×154	28×143
Lininit time (per block)	32.43ms	15.59ms	34.54ms	16.33ms
Lininit time (per image)	14.27s	2.84s	13.82s	2.61s
Init qerr	577.822	954.996	465.007	791.548
Final qerr	446.403	818.578	391.196	695.737
Training steps	5,000,000	2,100,000	5,000,000	3,800,000
Training time	1742min	1607min	1760min	2904min
BMU time (per block)	6.28ms	14.92ms	6.54ms	20.46ms
BMU time (per image)	2.76s	2.72s	2.62s	3.28s

Table 9.1: Summarized times and data for the trained models with different datasets.

The poor performance in terms of time is not a big issue, since the code should be evaluated without the I/O operations for real-world uses. That will eventually halve the operational time. Also better multi-threaded algorithm for finding the BMUs could be applied, that can further bring down the operational time needed. And finally, the current SOM maps are of very big size (around 4000 nodes each). Therefore finding the BMUs take long time and cutting the number of nodes will definitely improve the running speed. Smaller maps may cause worse data distribution mapping, but experiments with maps of size 24×24 were done in [OTMB13] and they have shown quite sufficient results in constructing the maps (see figure 6.6). Therefore any number of nodes between 500 and 4000 will suffice and it should be chosen to optimally use the available processing power.

Table 9.2 shows the PCA analysis of the data. The variance explained by the first three principal components are shown and as evident from the table, the first principal component is very dominant. In the next section 9.1.2 it could

Variance explained by PCA components in %				
	CASIAFPV5		MCYT330-DP	
	16×16	24×24	16×16	24×24
Number of PC	256	576	256	576
1st PC	64.03	59.86	57.58	52.93
2nd PC	2.66	3.28	2.12	2.69
3rd PC	1.99	1.71	1.82	1.45
All other PCs	31.38	35.15	38.48	42.93

Table 9.2: Variance of the training data explained by the first 3 PC and all other PCs in %. Numbers do not sum to 100% due to roundings.

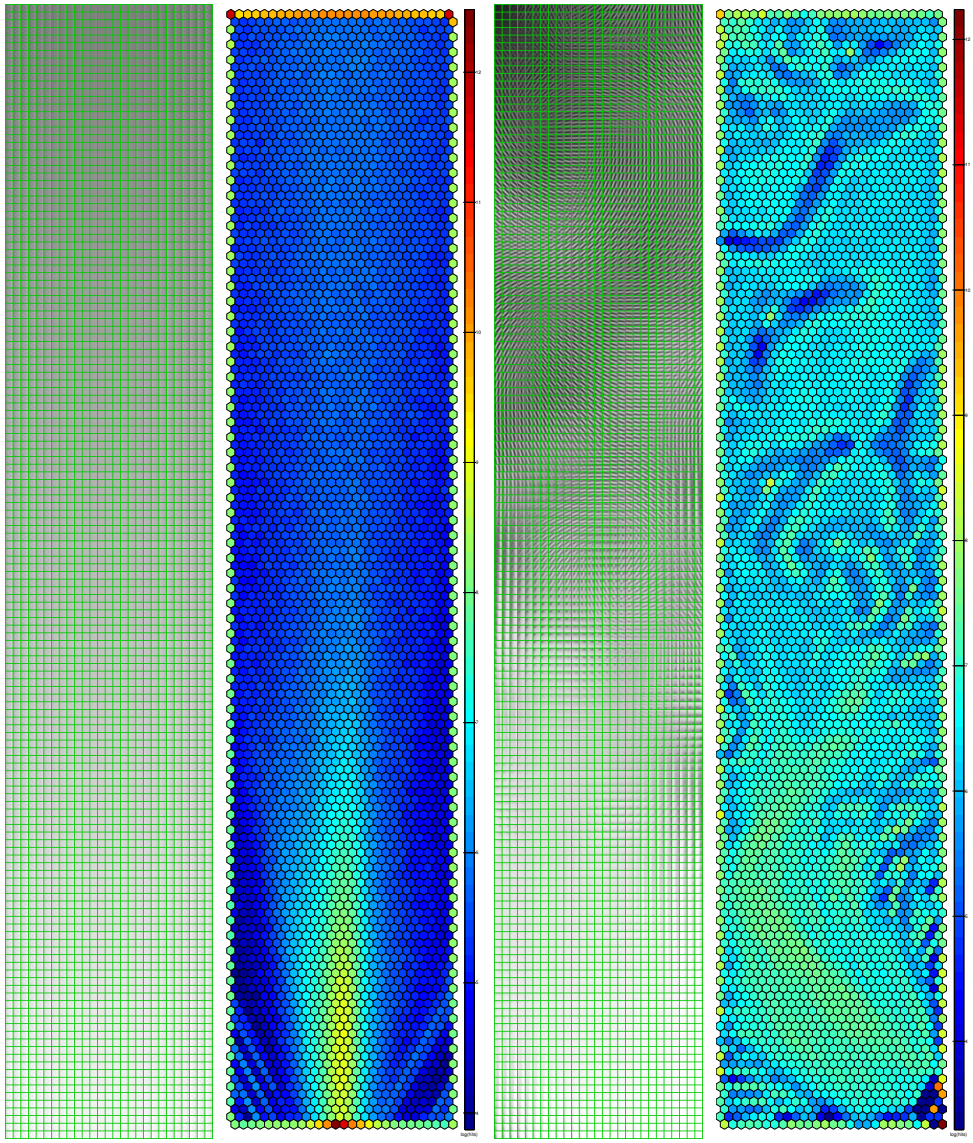
be seen that the first PC is the brightness of the image.

9.1.2 Resulting maps

Analyzing the results from the training is done by observing the constructed SOM maps. Each of the nodes of these maps is a data vector, that could be transformed back into image block. Each of the SOM map nodes has different number of blocks mapped to it and analyzing the number of mapped blocks to each node is done using hitmaps. However, the data from these experiments is not smoothly distributed over the SOM map, therefore a log-hitmap is considered. These blocks show the data distribution and give better feeling of the data cloud being mapped by the SOM map.

Observing the results from figure 9.1, one might notice that a lot of the data is concentrated in the parts of the SOM map, representing image blocks from the background. That is because there are too many background images and since they look quite similar, they appear in close regions, where the nodes are not that far in the data space from each other, as it could be seen in figure 9.2d. The result of that is capturing a lot of variance in the background image data during the training.

Capturing the variance in the background is probably useless for evaluating the quality of the fingerprint images, since the comparison algorithms usually segment out the background before comparing the fingerprint images. A lot of training time is wasted for propagating background blocks through the SOM map, which slows down the training process. The background data is white, where the fingerprint patterns are darker, causing the first principal component to be the brightness of the blocks. Since the SOM maps are linearly initialized, the presence of so many blocks of background data distorts the eigenvalues of

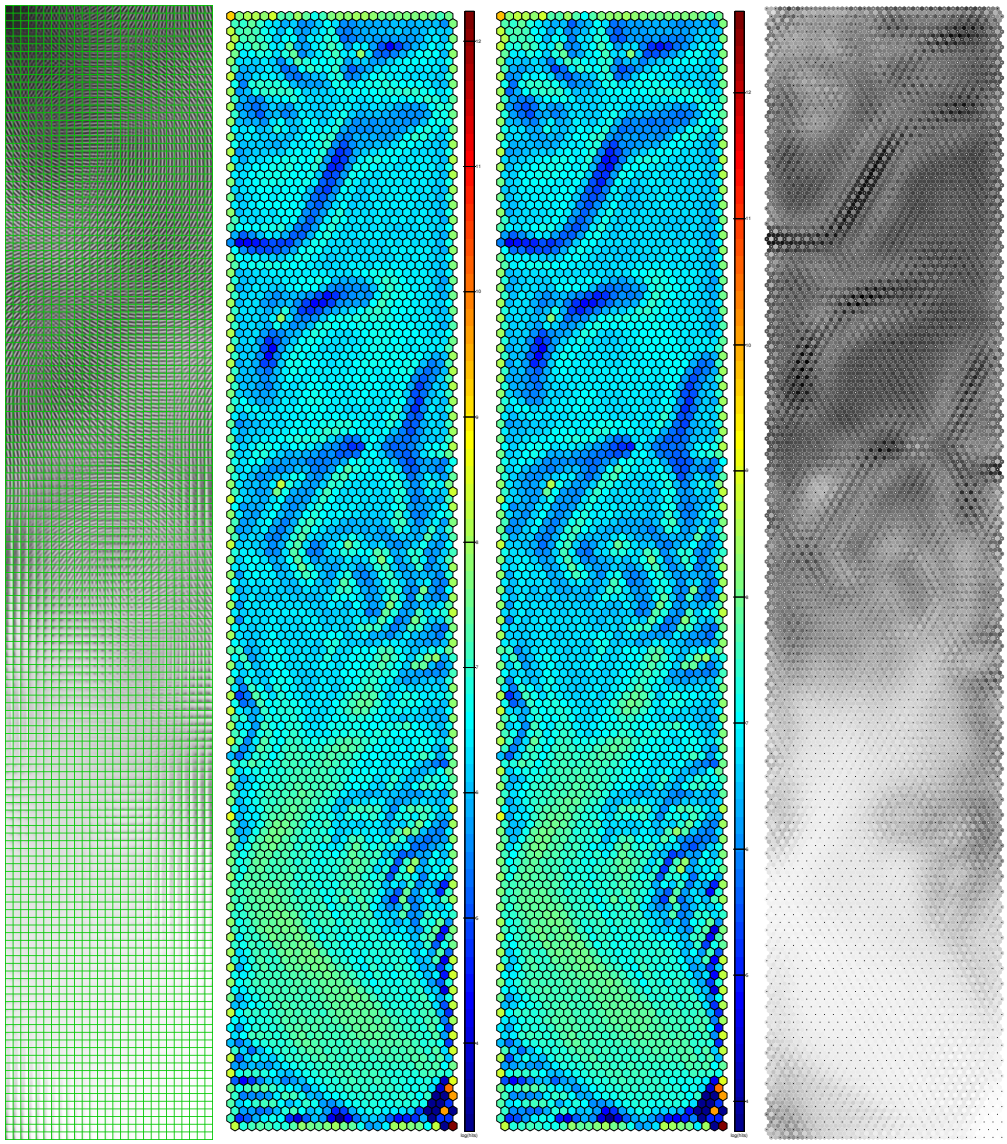


(a) initial SOM map **(b)** initial SOM map's log-hitmap **(c)** trained SOM map **(d)** trained SOM map's log-hitmap

Figure 9.1: Results for dataset CASIAFPV5 and block size 16×16
(a) the SOM map after being linearly initialized;
(b) the log-hitmap before the training (using validation data);
(c) the SOM map after the training;
(d) the log-hitmap after the training (using validation data).

the principal components, thus the SOM map's elongated shape is not reflecting the real foreground data distribution.

Figure 9.3 shows more detailed what kind of samples are mapped to the nodes. As one might observe, the SOM map nodes represent more or less a mean of all training blocks, mapped to them. So the training process could be interpreted as unsupervised block image pattern learning and propagating through the SOM map is block image pattern classification. This classification is local feature extraction, so the resulting local features are used as input for the high-level phase.



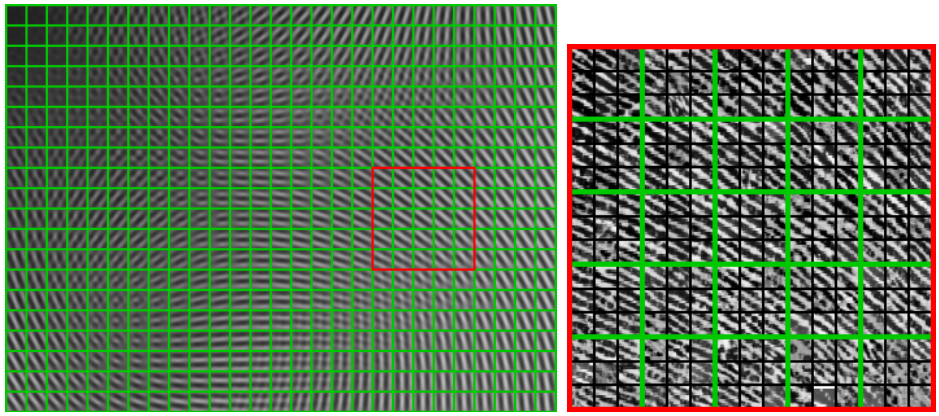
(a) trained SOM map

(b) trained SOM map's log-hitmap (validation set)

(c) trained SOM map's log-hitmap (training set)

(d) trained SOM map's distance matrix (U-mat)

Figure 9.2: The trained SOM map, the log-hitmaps for the validation and the training sets and the distance matrix U-mat. The dataset used for training is CASIAFPV5 and the block size is 16×16.



(a) top 20 rows of the trained map **(b)** example block samples

Figure 9.3: Part of the trained map from figure 9.1c with selected region of nodes and example validation block samples from each of the selected nodes.
(a) top 20 rows of the trained map from figure 9.1c;
(b) randomly chosen validation block samples, whose best-matching units are the nodes in the selected region.

9.2 High-level

After having observed the details in the results of the low-level training phase, the high-level phase results should be discussed. There are couple of approaches used for the high-level phase and the results of all of them will be presented here.

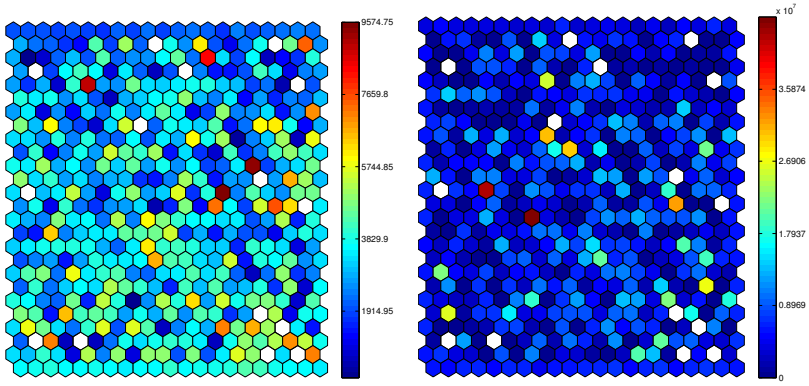
9.2.1 SOM

Using the low-level results, a high-level SOM map is trained. Then the calibration is done in accordance with the scores obtained by the given comparison algorithm provider. Each node is labeled with the mean average comparison score of the mapped to it samples. The variance for that node is calculated in the same manner after having the mean values.

The results from this calibration are shown in figure 9.4. As it could be seen from the variance maps, the majority of the nodes possess quite low variance, therefore they are expected to have quite stable predictions. Unfortunately most of the high-variance nodes are labeled with low prediction scores, thus low predicted values are prone to errors, which results in degrading performance.

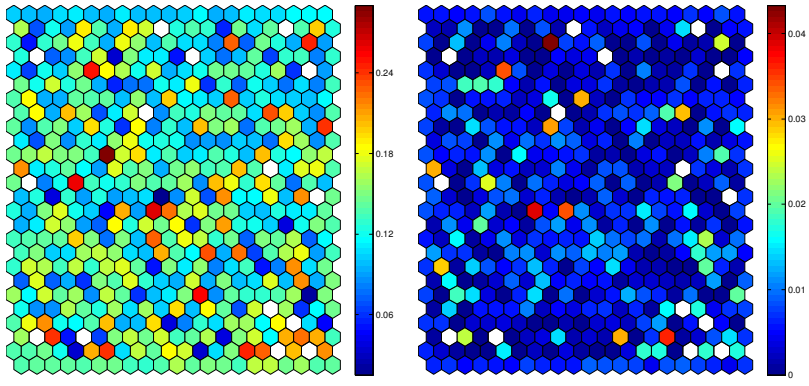
The SOM map training is very fast and it takes less than a minute altogether with the initialization of the map. The calibration takes just a couple of seconds, therefore a lot of experiments could be done to improve the training and once trained, the map could be calibrated in accordance with any comparison algorithm provider without any hassle.

The SOM map's hitmap shows strong bordering effect, meaning that the data is concentrated in the edges of the map. As it could be noticed from the prediction maps as well, there are some empty nodes, that do not have any samples assigned. That might suggest the need for better training parameters or even different map topology, e.g. toroidal SOM, in order to be able to capture the data distribution better. However, only experiments with the data could prove better effectiveness of the proposed improvements. More images showing the results could be found in the appendix of this thesis.



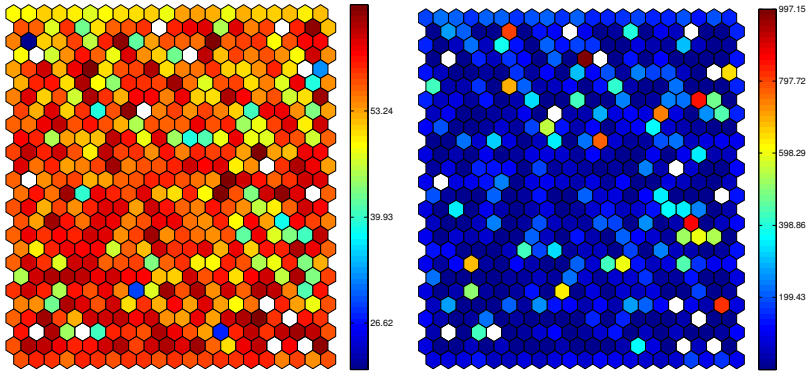
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

(d) prediction variances map; provider {63}



(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

Figure 9.4: The trained high-level SOM map for CASIAFPV5 and block size 16×16 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.

9.2.2 GTM

The GTM training finished in less than a minute altogether with the initialization. The calibration is done for a couple of seconds, therefore calibrating the map for different providers is very fast and easy process. Figure 9.5 shows the prediction and the variance maps, similarly to the ones shown for the SOM map from the previous section 9.2.1.

Observations similar to the ones from the previous section could be formed for the GTM as well. However, the prediction maps here contain more homogeneous regions and the low-valued nodes are grouped together, unlike in the SOM map. That might suggest that GTM is able to capture the data distribution better. Another figure showing how the data is captured by the model is the log-hitmap seen in figure 9.6, where the data distribution could be analyzed by observing the concentration of the data posterior modes.

9.2.3 RF: features and histogram

	1	2	3	4	5
1	852	519	338	192	75
2	248	303	284	204	136
3	231	297	305	293	200
4	179	261	395	501	508
5	105	142	226	389	817

(a) provider {28}

	1	2	3	4	5
1	710	449	395	323	212
2	372	422	293	200	131
3	229	229	266	271	183
4	219	290	392	512	382
5	165	163	194	263	735

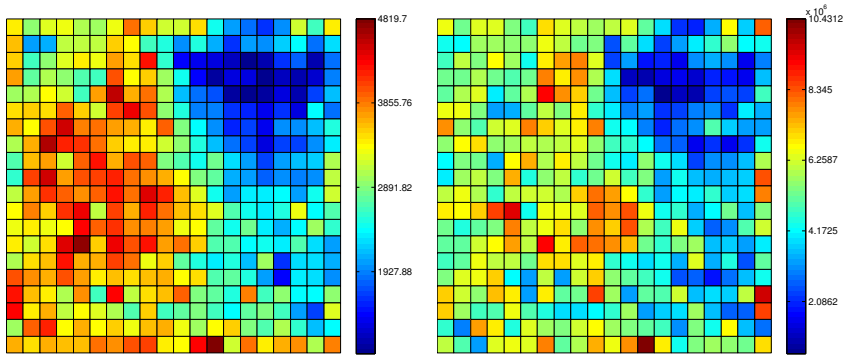
(b) provider {63}

	1	2	3	4	5
1	850	470	328	227	101
2	299	286	251	253	182
3	191	232	246	251	234
4	195	298	312	353	361
5	167	239	389	471	814

(c) provider {83}

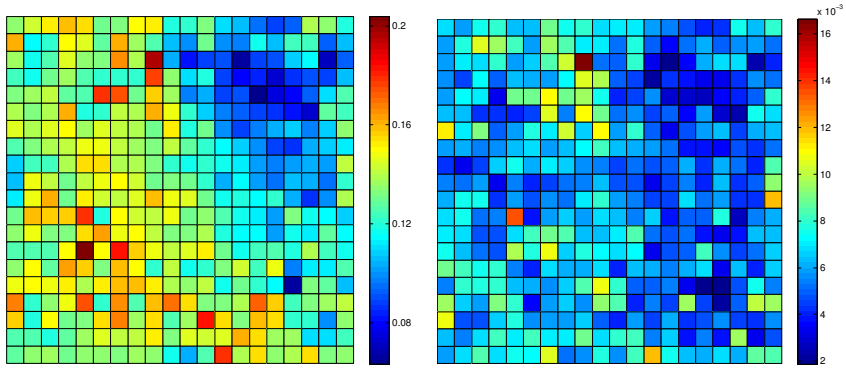
Table 9.3: Confusion matrices for RF:feat for different providers and dataset CASIAFPV5 with block size 16×16.

There are couple of RF models constructed during the training process for the RF high-level phase. Each comparison score provider requires a new model to be trained. The training of the RF-feat model needs between an hour and two hours to complete and is trained to classify the samples into five quality score



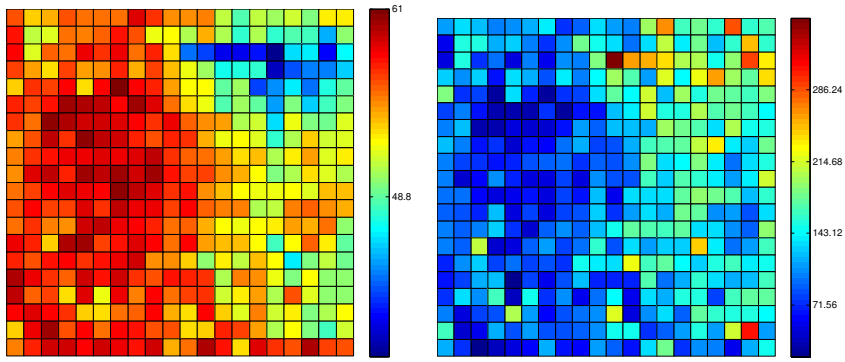
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

(d) prediction variances map; provider {63}



(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

Figure 9.5: The trained high-level GTM map for CASIAFPV5 and block size 16×16 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.

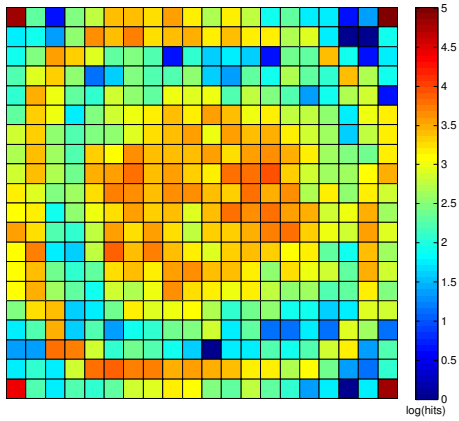


Figure 9.6: The GTM log-hitmap for CASIAFPV5 and block size 16 × 16, where the *posteriors' modes* are counted as *hits*.

classes. The training of the RF-hist model needs more than ten hours, due to the larger feature vectors used. The confusion matrices for each of the models are shown in table 9.3 for RF-feat models and in table 9.4. Results for the other dataset and different block sizes could be found in the appendix.

	1	2	3	4	5
1	1041	658	450	291	140
2	192	218	212	174	119
3	186	296	309	293	181
4	135	235	343	455	515
5	61	115	234	366	781

(a) provider {28}

	1	2	3	4	5
1	801	516	467	395	264
2	271	282	210	191	142
3	239	259	259	246	172
4	251	321	403	455	463
5	133	175	201	282	602

(b) provider {63}

	1	2	3	4	5
1	1032	613	429	321	156
2	205	214	193	214	139
3	168	232	277	269	245
4	193	266	330	366	413
5	104	200	297	385	739

(c) provider {83}

Table 9.4: Confusion matrices for RF:hist for different providers and dataset CASIAFPV5 with block size 16 × 16.

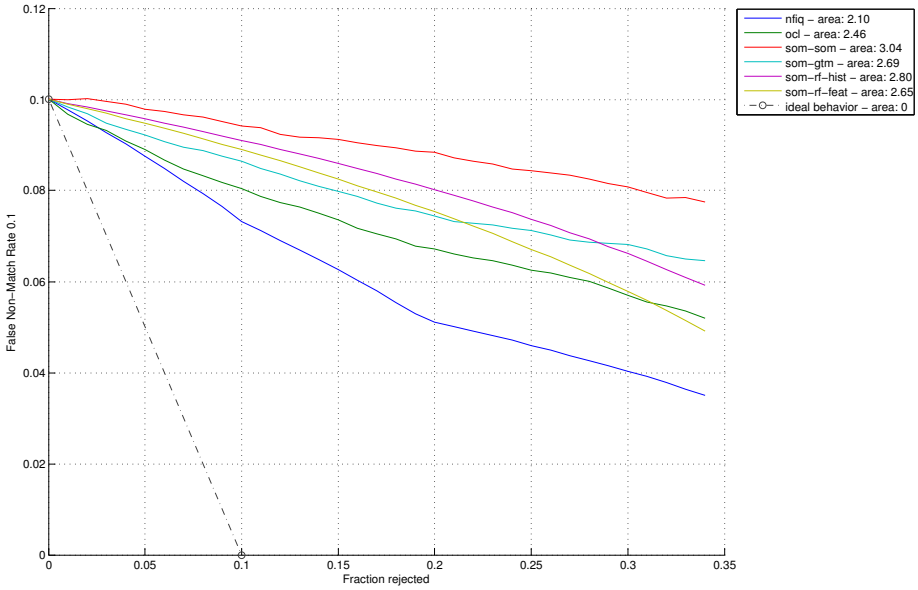
9.3 Final results

In order to evaluate the performance of the proposed approaches, the ERC curves and the Spearman correlation matrices should be analyzed. Since the quality levels used by the RF approaches and NFIQ are quantized into five levels, a *jittering* is added for drawing them as shown in section 3.3.3. In order to eliminate the fluctuations caused by the randomness, 20 ERC curves are calculated for each approach and the mean curve is drawn in the figure. Area under the curve is computed and the area of the perfect curve is subtracted, so the approaches could be evaluated quantitatively how close they are to the ideal curve.

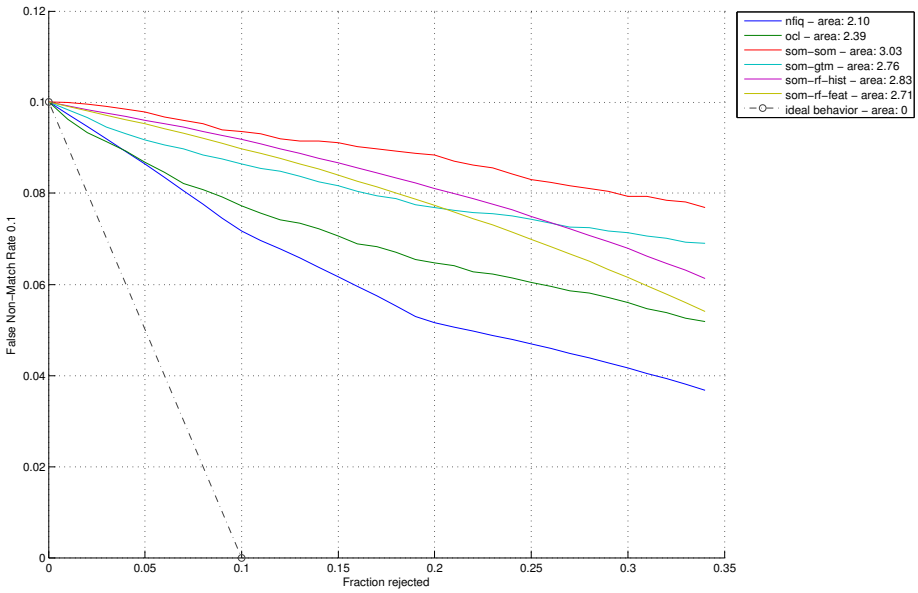
The resulting ERC curves for the providers {28} and {83} are shown in figure 9.7. These results show that the worst performing approach is using SOM for the high-level training. Second worst is the high-level approach used in [OTMB13], which is the RF model trained with the histogram as an input, rather than using directly the extracted features. The other two methods are performing quite similar quantitatively, however the GTM approach is better than the RF-feat approach if only the worst 20% of the samples are rejected.

The ERC curve for provider {63} is shown separately in figure 9.8, since it is not very illustrative of the performance of the methods. That is because the comparison algorithm {63} is performing poor with the samples from the dataset CASIAFPV5, due to existence of rotated samples and this algorithm gives very low, even zero, genuine comparison scores for samples that have larger difference in the angle of rotation. However, the results shown in figure 9.7 are coherent and conclusions for the performance of the methods could be drawn easily. These conclusions are confirmed by the results obtained for the other tested dataset MCYT330-DP, which could be seen in the appendix.

Figure 9.9 shows the Spearman correlation tables for the dataset CASIAFPV5 with block size 16×16. From these tables is visible that NFIQ shows highest correlation with the average genuine comparison scores of the samples. OCL and the two RF methods show also quite high correlation with the comparison scores and the NFIQ as well. On the other hand, the GTM approach is very highly correlated with the SOM approach and poorly correlated with the comparison scores. Looking at these tables, it is easily visible that the SOM and GTM approaches are somehow related, the same is true for the two RF approaches.



(a) provider {28}



(b) provider {83}

Figure 9.7: ERC curves for CASIAFPV5 with block size 16×16 for two providers.

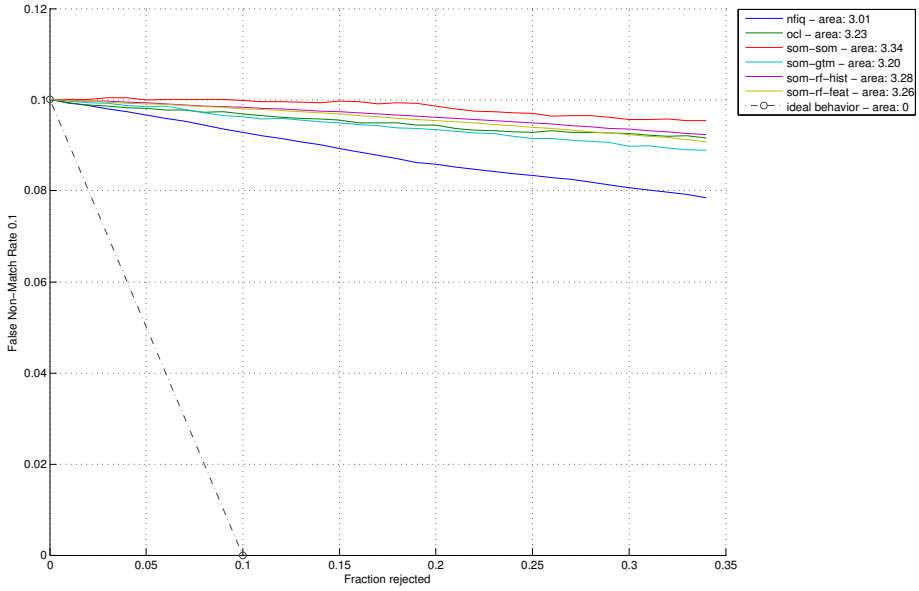
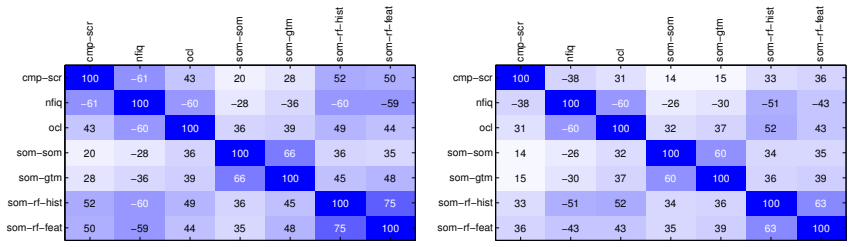
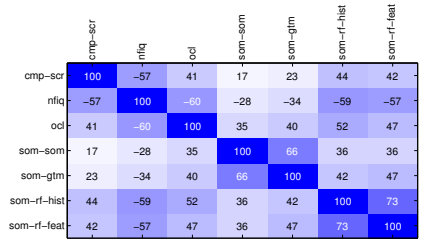


Figure 9.8: ERC curves for CASIAFPV5 with block size 16×16 for provider {63}.



(a) provider {28} **(b)** provider {63}



(c) provider {83}

Figure 9.9: Correlation tables for CASIAFPV5 with block size 16×16.

Chapter 10

Conclusions

The goals of the thesis, listed in section 1.2, are met and some more useful information about the data distribution is obtained. As proposed in the beginning of the thesis, the experiments performed in [OTMB13] were repeated with slightly changed low-level training process, result of the investigation whether this low-level phase could be improved. A couple more high-level phase methods (SOM, GTM and RF-feat) were proposed and evaluated for their performance.

Since the low-level training phase used linear initialization, computation of the principal components (PCs) of the block image data was performed. As the eigenvalues of the covariance matrix is showing, the first principal component explains more than the half of the variance of the data. However, by looking at the constructed initialized map, one might easily notice that the first principal component is the brightness of the image data, which is not surprising. However, by looking at the log-hitmap of the initial SOM map, the U-matrix and the log-hitmap of the final SOM map, as well as the SOM map itself, one might notice that more than a third of the map is explaining the variance in the background blocks. The variance data in the background blocks is useless, due to the fact that the comparison algorithms usually segment the background out. Therefore only the fact that the block is part of the background is important, but not the information what type of background block it is actually. By removing the background blocks from the training, smoother SOM map could be constructed for the foreground image data, thus better final results are expected.

Initializing linearly the low-level SOM map gives better overview of the data cloud that is inspected, however performance comparison with random initialization was not done due to the limited time for training and computational resources available. Despite that the deeper analysis of the low-level SOM map have shown better the form of the data cloud and how this low-level phase could be possibly improved by applying the suggestions proposed in the next chapter 11.

The evaluation of different high-level approaches has shown good results for the methods GTM and RF-feat, which have outperformed the proposed in previous works RF-hist for dataset CASIAFPV5. For dataset MCYT330-DP the method RF-hist outperforms slightly the new methods. The histogram features used in [OTMB13] for the RF training increase the length of the training vectors for big maps and loose important information, namely the position of the blocks in the fingerprint image sample. However, this seems to be crucial only for dataset CASIAFPV5 and stating that these observations are valid for all datasets, requires more experiments with better model selection for the high-level models.

Another conclusion that could be drawn is that the RF training requires much longer time, compared to the other two approaches - SOM and GTM. Also for each comparison algorithm, a different model should be trained, where the constructed SOM and GTM maps need only to be calibrated, which is fast and easy process. The training of the SOM and GTM maps does not require quality levels to be defined prior to the training and the number of quality levels does not influence the training process, unlike the two RF approaches. Therefore, this observation is arguably an important one, if more quality levels are needed. Comparative analysis of the operational speed for the high-level phase methods unfortunately was not done, however the propagation time for all the high-level methods is neglectable compared to the time needed by the low-level phase. Thus the gain of optimizing the low-level phase is much bigger.

As final according to the ERC plots GTM and RF-feat have outperformed the old high-level method RF-hist for CASIAFPV5, but for MCYT330-DP the results are controversial and more experiments should be conducted. The resulting low-level maps for both datasets for block size 24×24 have shown to be undertrained, thus more training cycles are needed for this block size. Additionally much better model parameter selection could be performed for both low- and high-level models and this will yield more robust and stable results, that could be used as confirmation to the drawn conclusions. For the low-level phase, training without the background blocks is expected to construct a much finer low-level model, thus improving the results of all high-level methods.

Chapter 11

Future works

The conclusions from the previous chapter 10 lead to the rise of new questions to be asked. These new questions are mainly focused on the improvement of the performance of the proposed approaches. They should be used to determine the directions for the future works, based on the current progress. The future works could be focused on improving either the low-level phase or the high-level phase.

The low-level training is performed using linearly initialized SOM. However, SOM maps could be initialized randomly as well and no evaluation of the results was done using both methods. Therefore research focused on comparing linear versus random initialization could prove empirically the superiority of the either of methods. However, linear initialization is suggested by [Koh13] as better alternative, so the expectations are that the linear initialization would outperform random initialization. Also linear initialization guarantees robust starting point of the training process, thus determining the training parameters could be done easily by cross-validation. The parameters in the current thesis are determined heuristically, therefore applying cross-validation may yield much better training results, which could be tried out in future works.

The observations showing that about a half of the data variance is explained by the first principal component, which turned out to be the brightness of the block image, suggests that the SOM map nodes are spent to explain that variance.

However, by looking at the final SOM map codebook vectors, one might notice that at least a third of them explain the variance within the different background blocks. Therefore, a process of segmenting out the background blocks could contribute for decreasing the influence of the first principal component and focus the training of the SOM on the foreground data. That would lead to the need of smaller SOM maps and shorter and faster trainings of better maps. However, that would require additional image preprocessing and the background blocks features to be extracted by segmentation, where the foreground features by propagating through the trained low-level SOM map.

For the feature extraction low-level phase, the SOM method could be replaced by GTM. However, interpreting the results and extracting feature vectors of quite low dimensionality, e.g. less than 1000 features, might be a challenge, if GTM is used. A totally different approach by using deep belief networks [LGRN09] might show good results, if one decides to follow the trend of deep learning from the more recent couple of years.

For the high-level phase the first thing to be done is better parameter selection, since for the current thesis a heuristic approach was taken and it was not backed by many experimental runs. The high-level training phase is very fast to compute, thus a cross-validation approach is also possible. Comparative analysis of the speed of all tested methods should be done, even though all of them are expected to be very fast. Another approach to improving the high-level phase is to suggest other supervised or unsupervised learning methods that might prove useful.

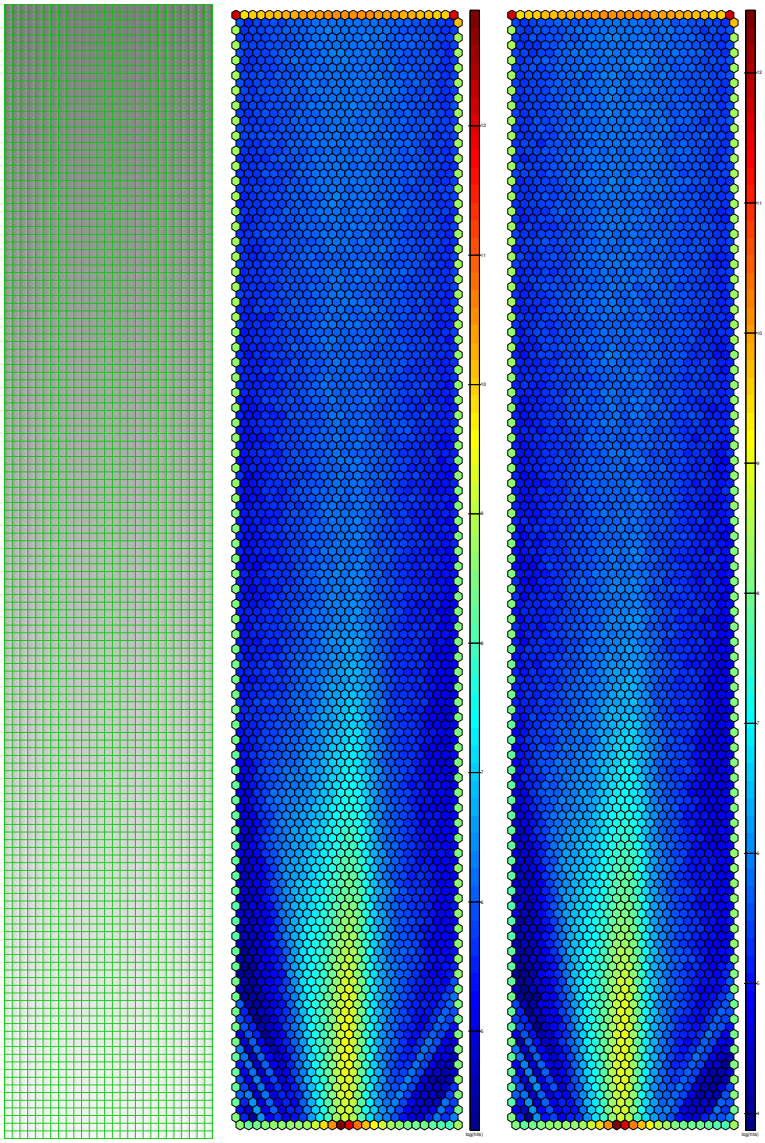
The problem of fingerprint image quality estimation is a hard one and machine learning techniques are a very natural choice for dealing with it. It is an active research topic and more experiments in the field would definitely contribute to the improvement of the fingerprint quality assessment in general.

Appendix A

Results for CASIAFPV5

A.1 Block size 16×16

A.1.1 Low-level

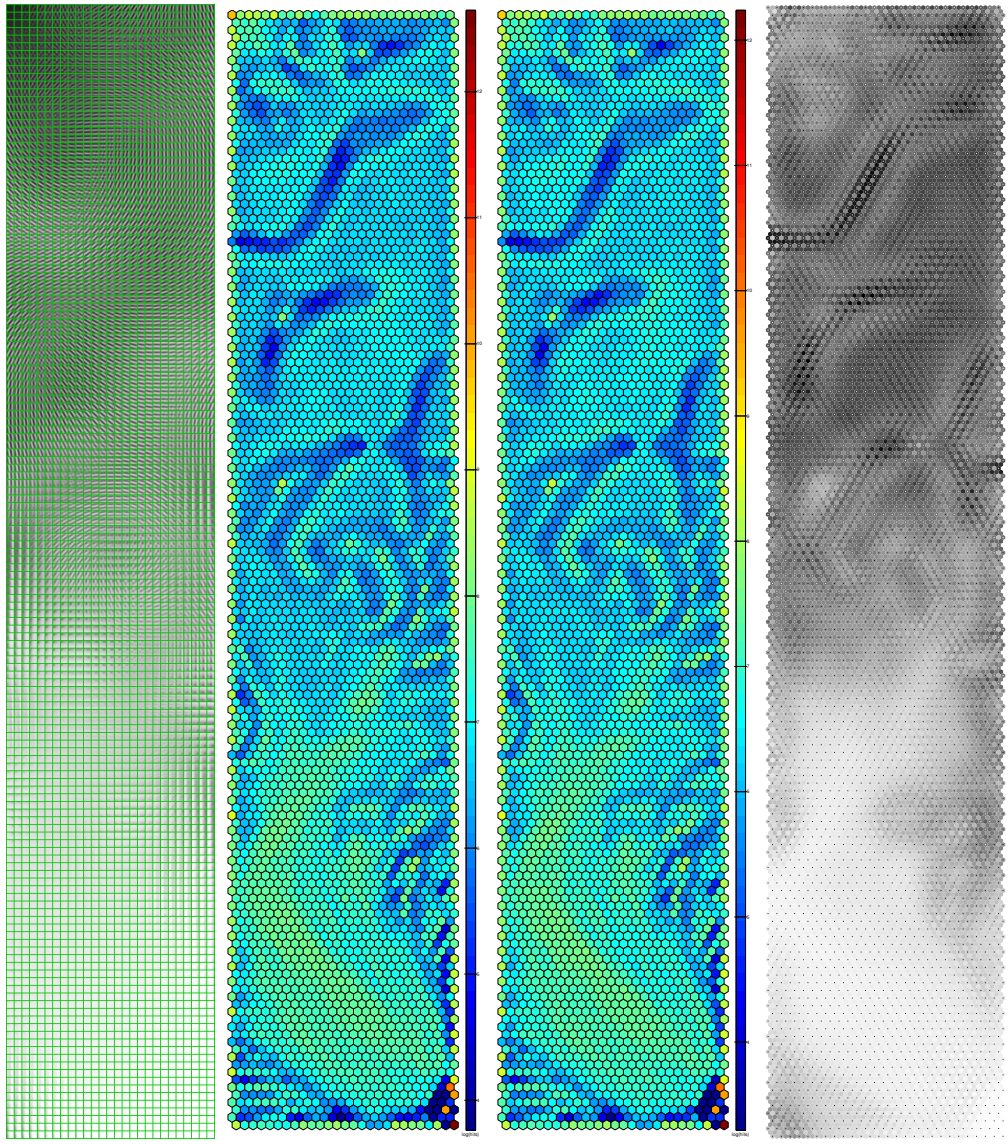


(a) initial SOM map

(b) initial SOM map's log-hitmap (training set)

(c) initial SOM map's log-hitmap (validation set)

Figure A.1: Linearly initialized SOM map for dataset CASIAFPV5 and block size 16×16



(a) final SOM map **(b)** final SOM map's log-hitmap (training set) **(c)** final SOM map's log-hitmap (validation set) **(d)** final SOM map's distance matrix (U-matrix)

Figure A.2: Results of the training for dataset CASIAFPV5 and block size 16×16

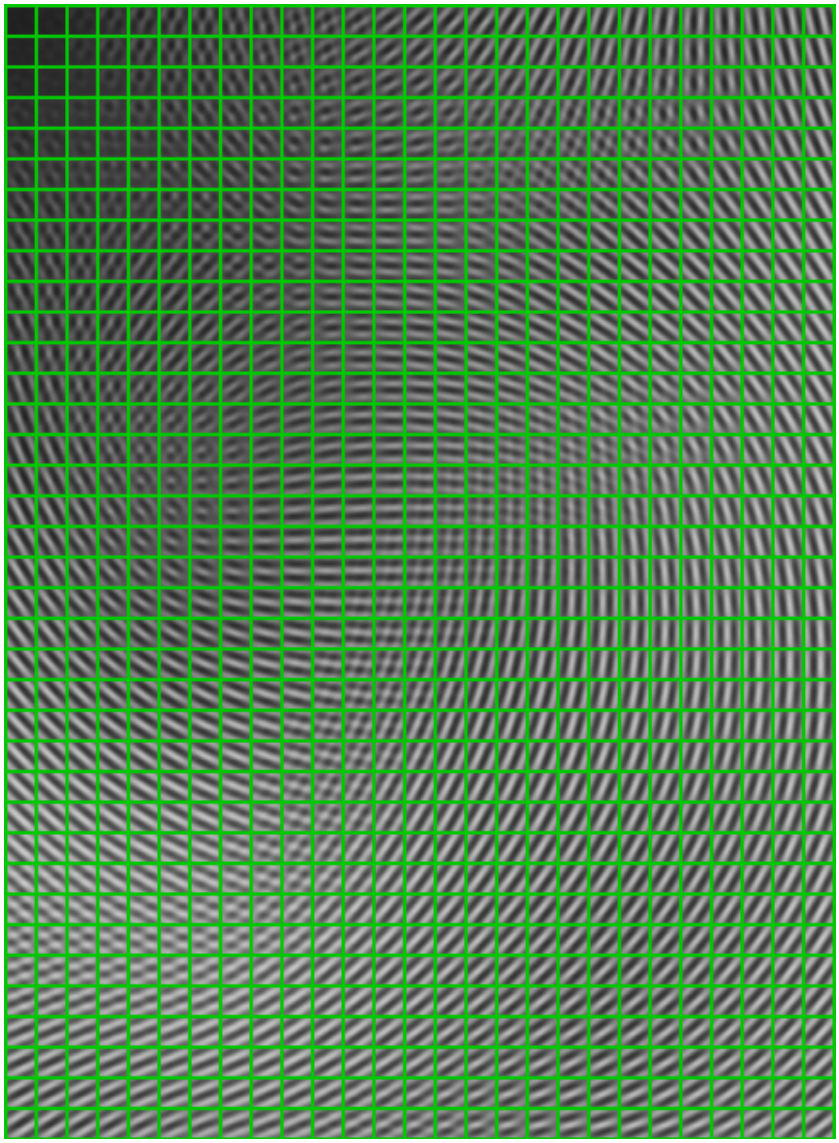


Figure A.3: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 16×16, first quarter of the map.

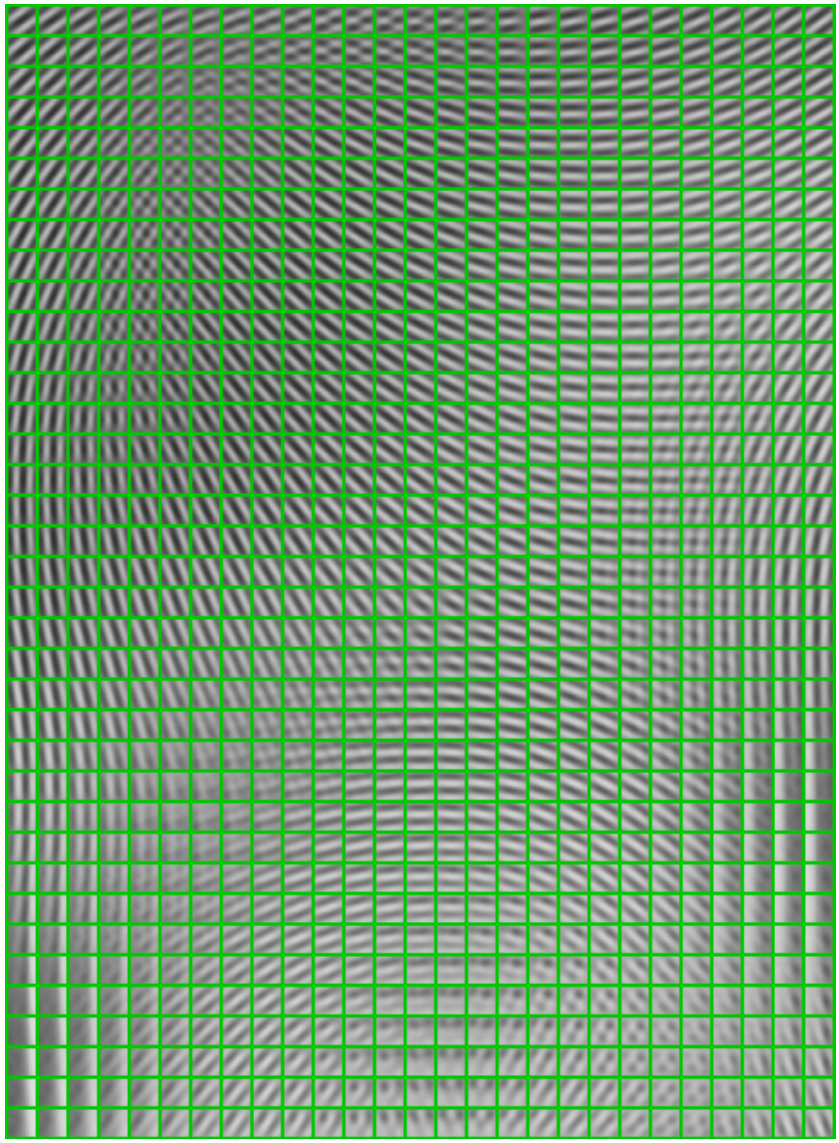


Figure A.4: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 16×16, second quarter of the map.

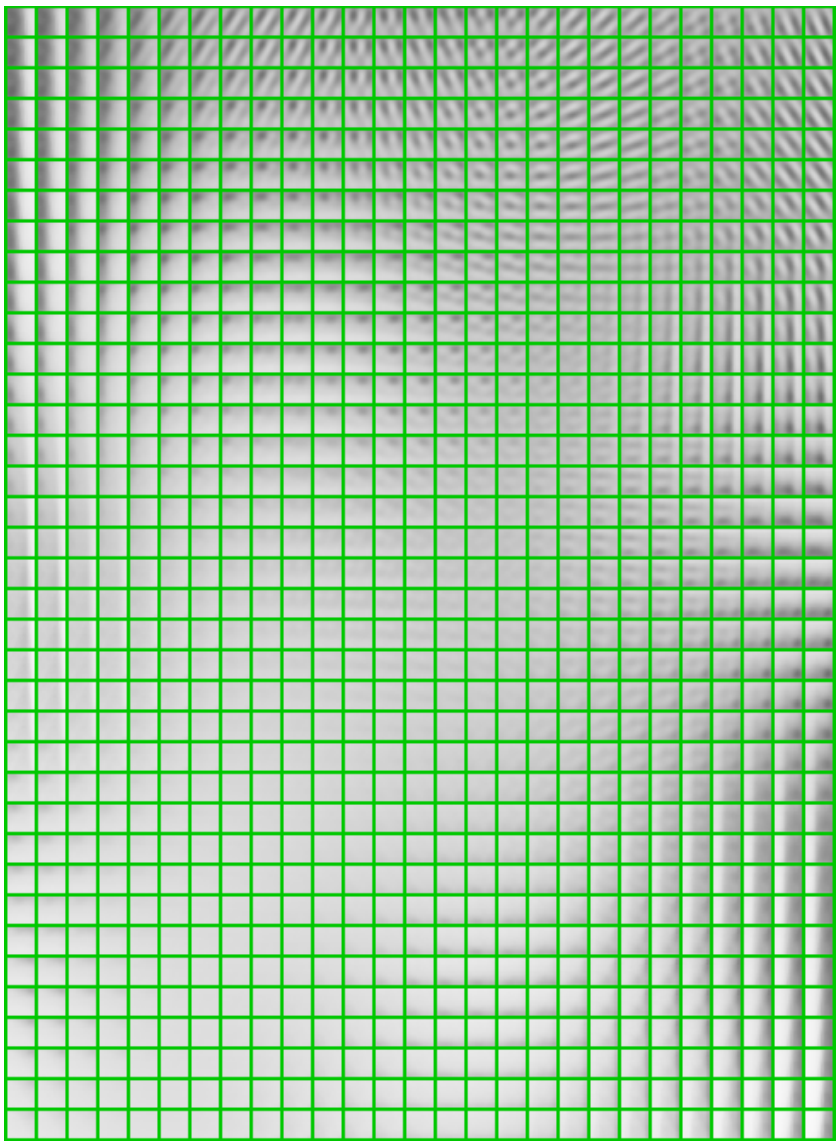


Figure A.5: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 16×16, third quarter of the map.

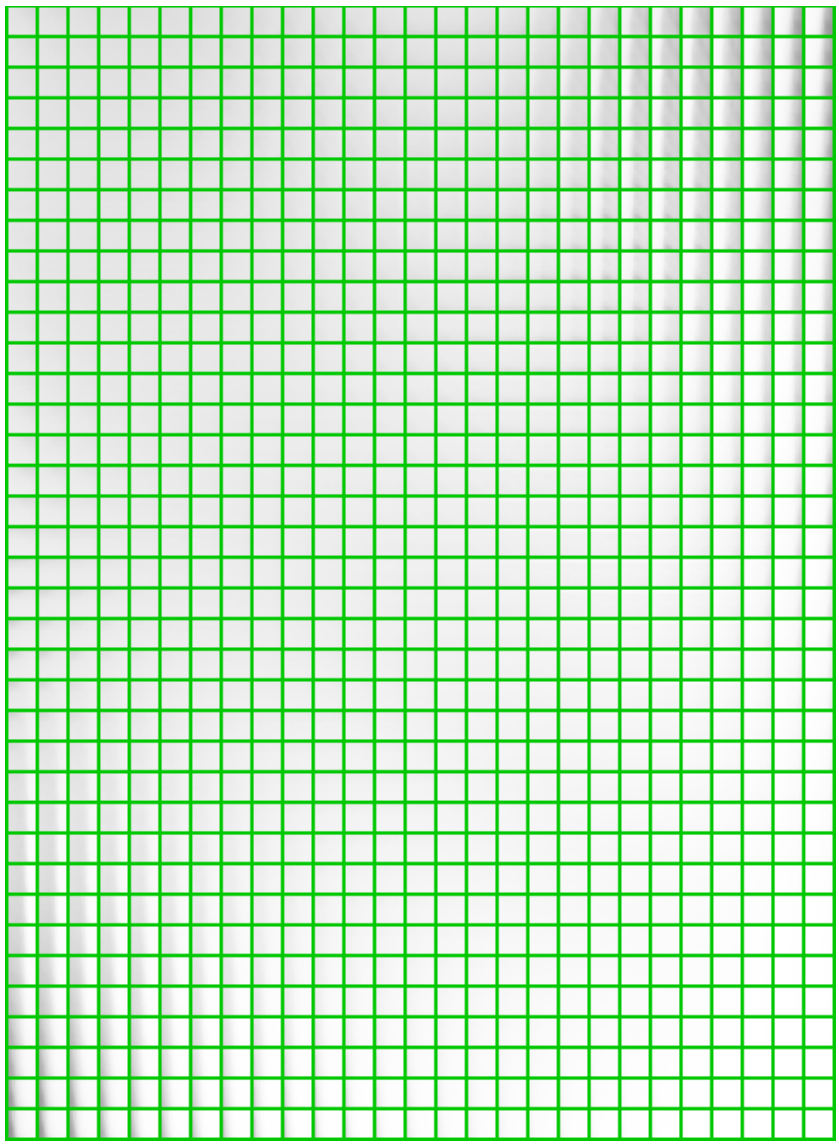


Figure A.6: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 16×16, last quarter of the map.

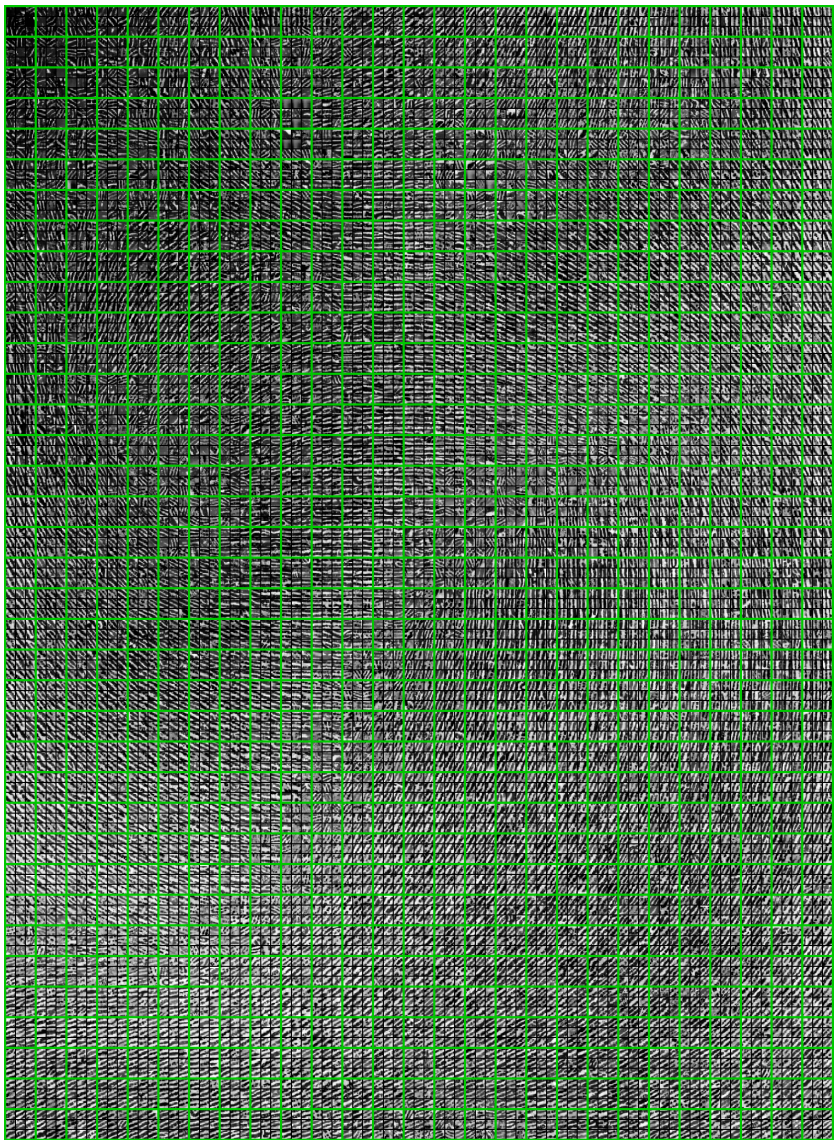


Figure A.7: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 16×16, first quarter of the map.

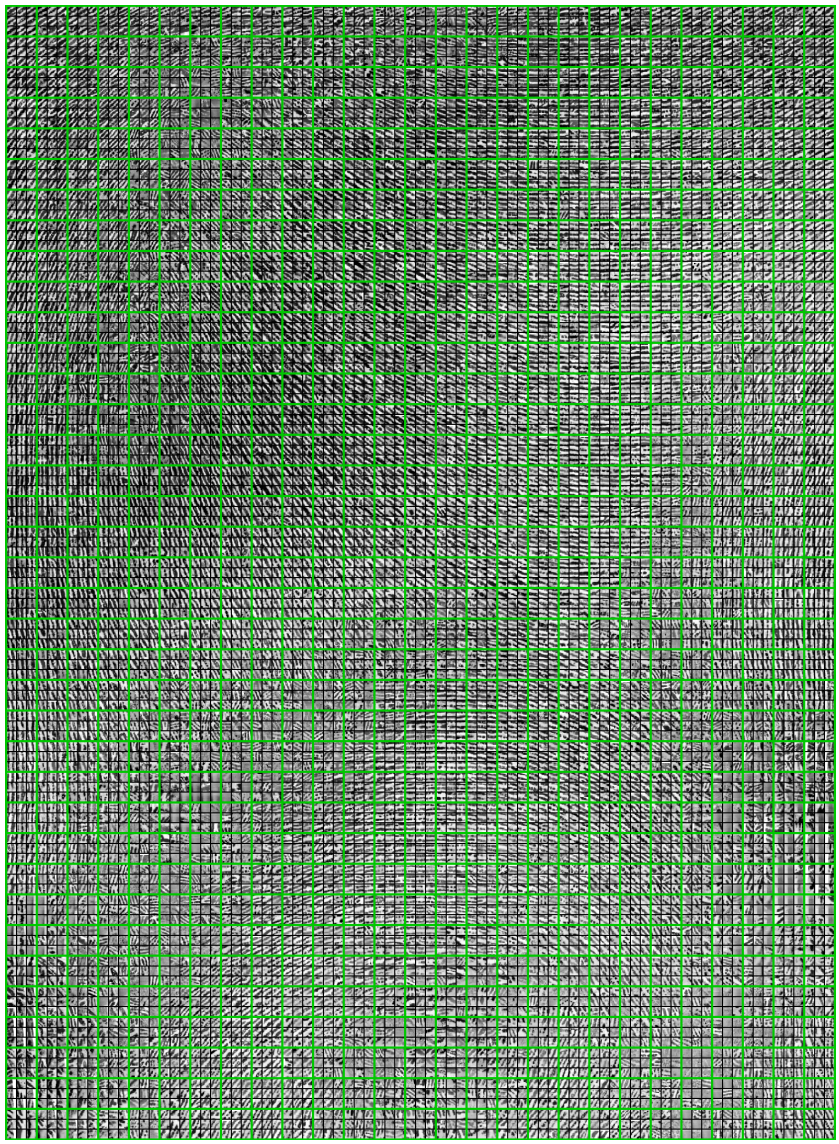


Figure A.8: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 16×16, second quarter of the map.

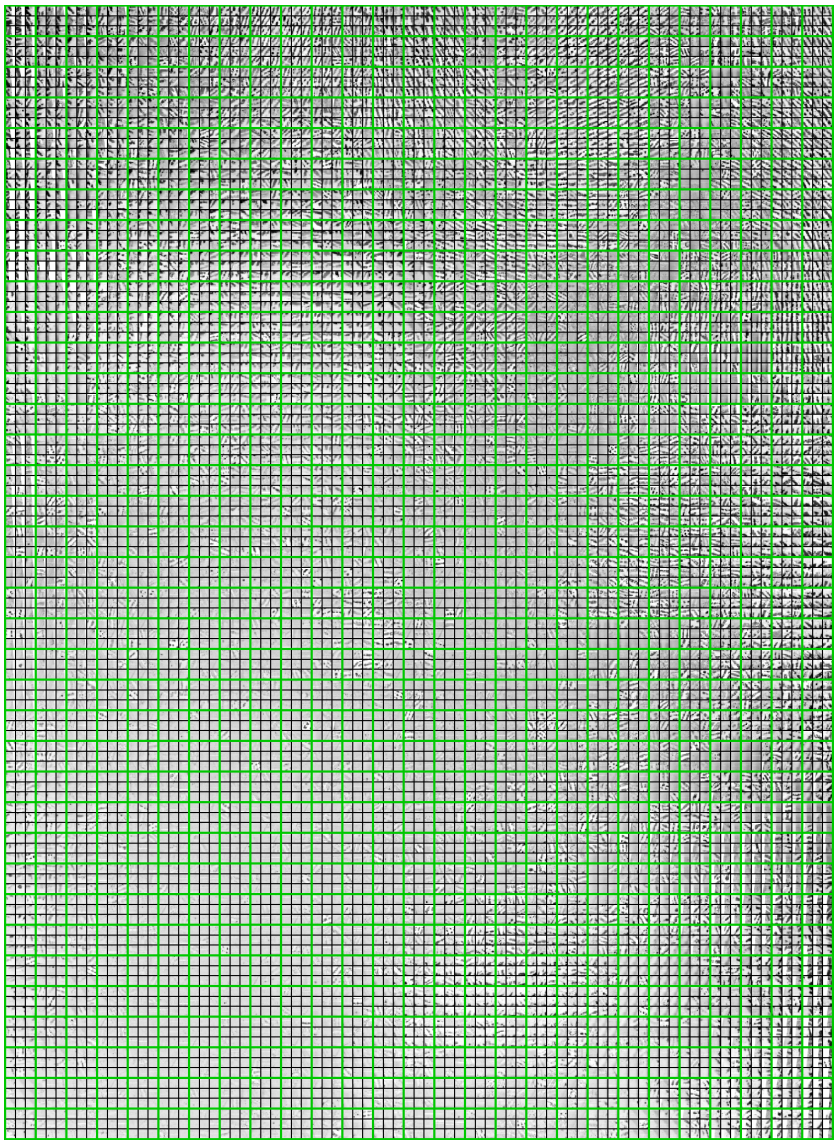


Figure A.9: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 16×16, third quarter of the map.

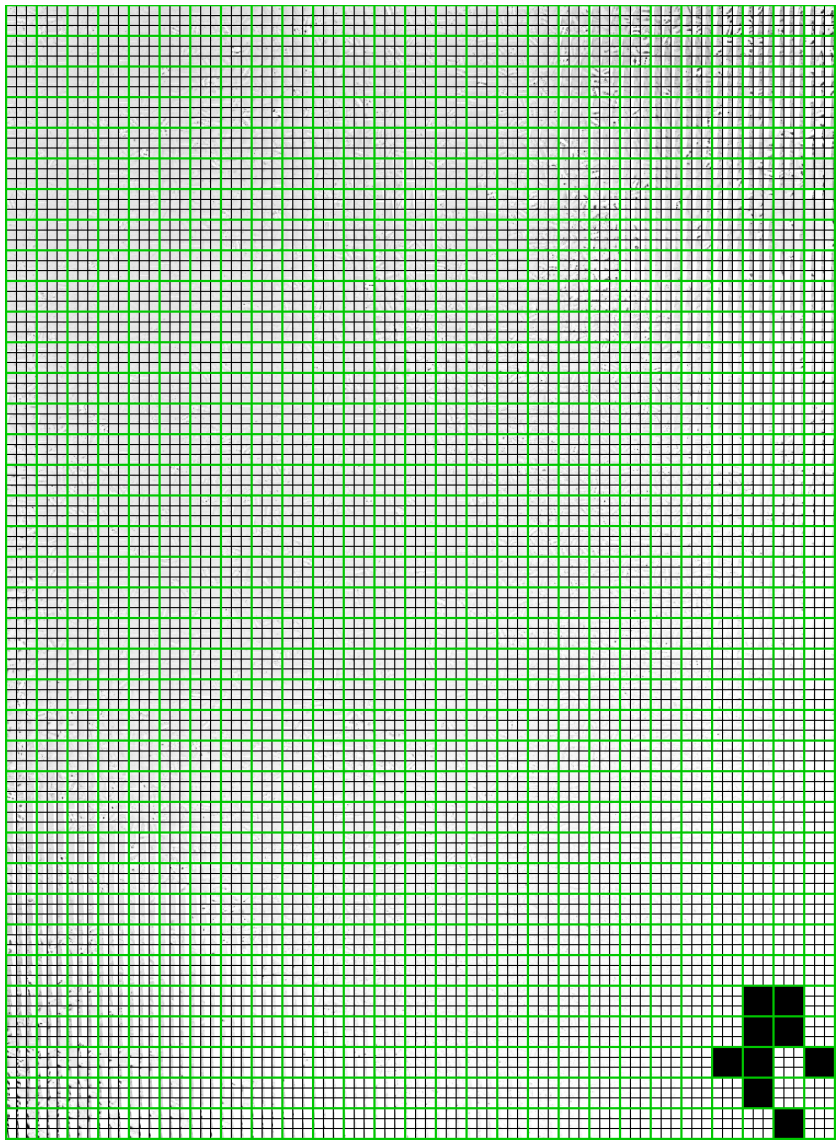


Figure A.10: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 16×16, last quarter of the map.

A.1.2 High-level

	1	2	3	4	5
1	852	519	338	192	75
2	248	303	284	204	136
3	231	297	305	293	200
4	179	261	395	501	508
5	105	142	226	389	817

(a) RF-feat; provider {28}

	1	2	3	4	5
1	1041	658	450	291	140
2	192	218	212	174	119
3	186	296	309	293	181
4	135	235	343	455	515
5	61	115	234	366	781

(b) RF-hist; provider {28}

	1	2	3	4	5
1	710	449	395	323	212
2	372	422	293	200	131
3	229	229	266	271	183
4	219	290	392	512	382
5	165	163	194	263	735

(c) RF-feat; provider {63}

	1	2	3	4	5
1	801	516	467	395	264
2	271	282	210	191	142
3	239	259	259	246	172
4	251	321	403	455	463
5	133	175	201	282	602

(d) RF-hist; provider {63}

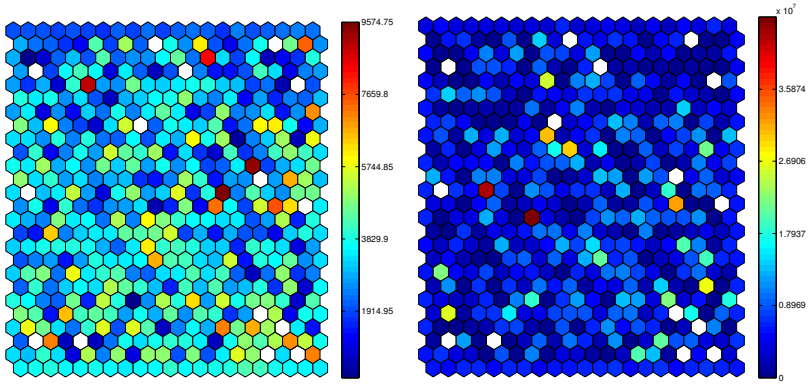
	1	2	3	4	5
1	850	470	328	227	101
2	299	286	251	253	182
3	191	232	246	251	234
4	195	298	312	353	361
5	167	239	389	471	814

(e) RF-feat; provider {83}

	1	2	3	4	5
1	1032	613	429	321	156
2	205	214	193	214	139
3	168	232	277	269	245
4	193	266	330	366	413
5	104	200	297	385	739

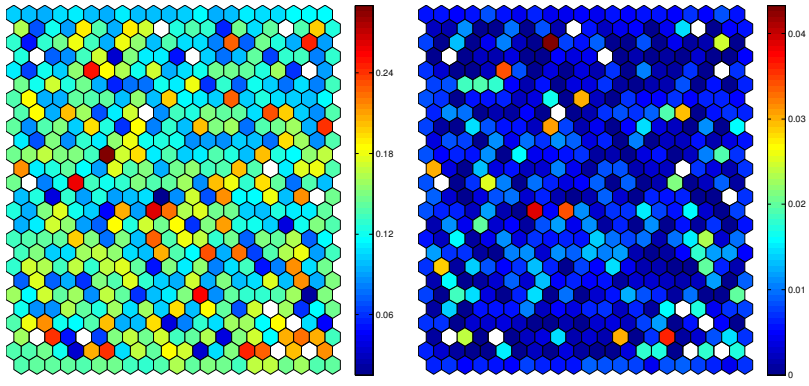
(f) RF-hist; provider {83}

Table A.1: Confusion matrices for RF-feat (left) and RF-hist (right) for different providers and dataset CASIAFPV5 with block size 16×16.



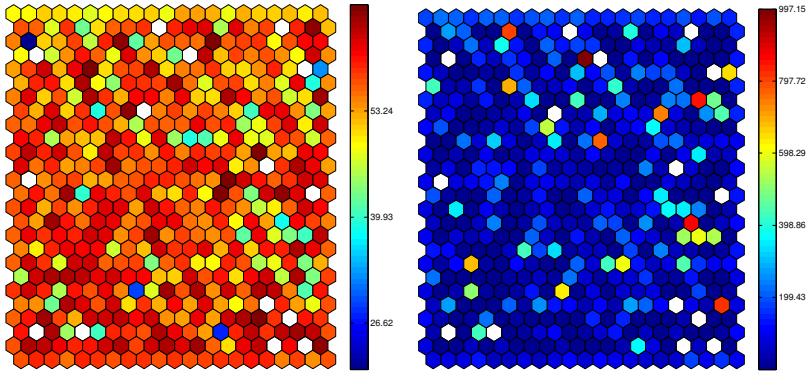
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

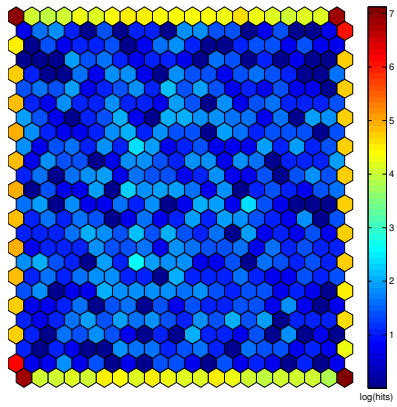
(d) prediction variances map; provider {63}



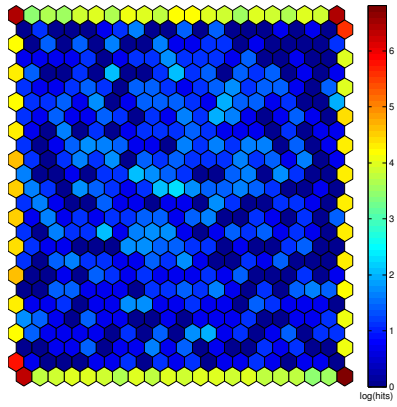
(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

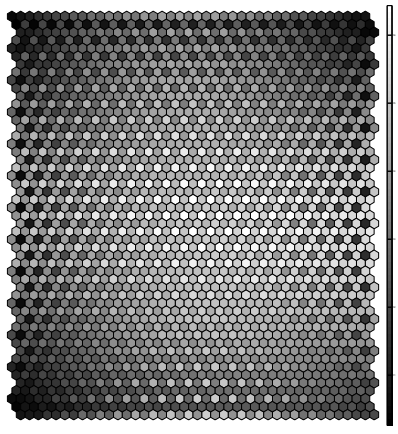
Figure A.11: The trained high-level SOM map for CASIAFPV5 and block size 16×16 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

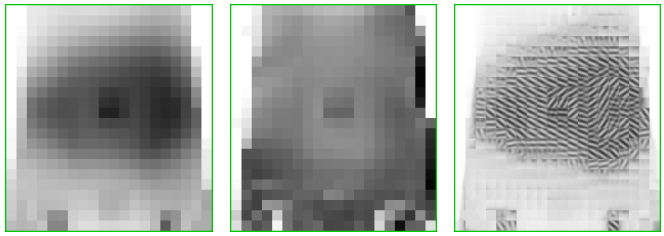


(b) log-hitmap (validation set)

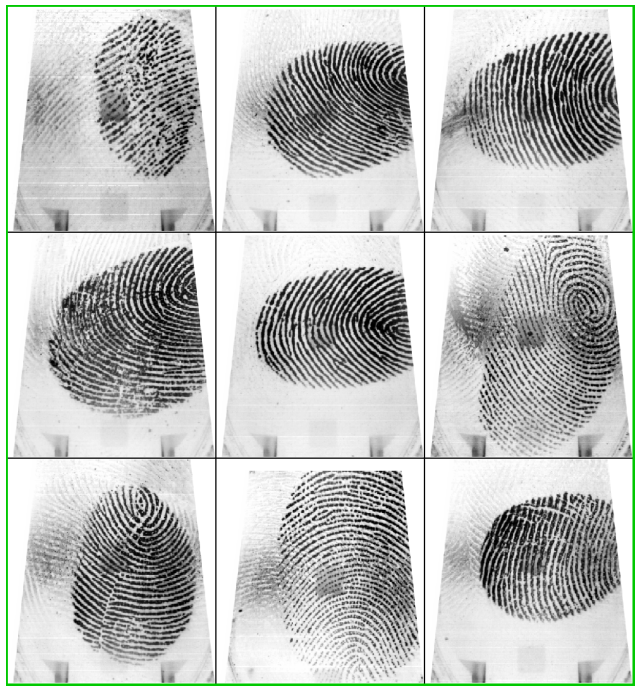


(c) distance matrix (U-matrix)

Figure A.12: Log-hitmaps and distance matrix for the trained high-level SOM map for CASIAFPV5 with block size 16×16.

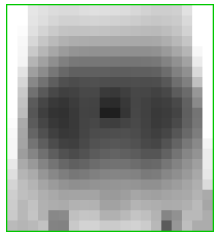


(a) vertical features **(b)** horizontal features **(c)** codebook blocks

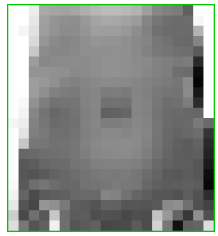


(d) example samples

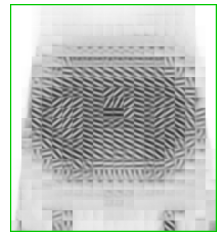
Figure A.13: Top leftmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 16×16.



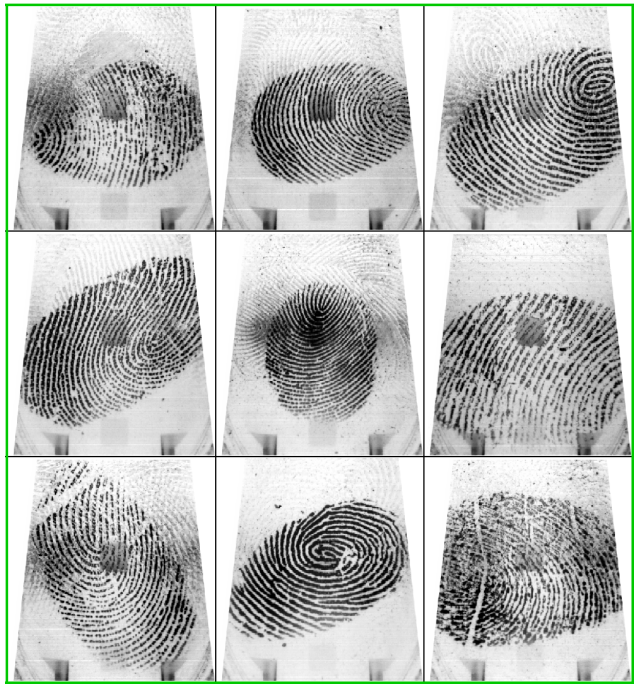
(a) vertical features



(b) horizontal features

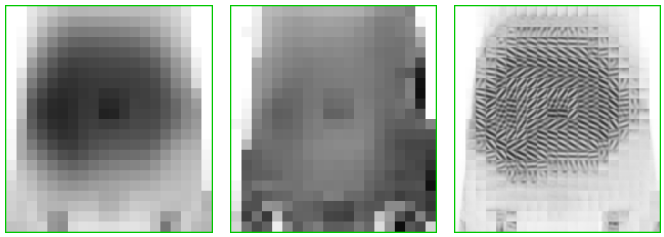


(c) codebook blocks



(d) example samples

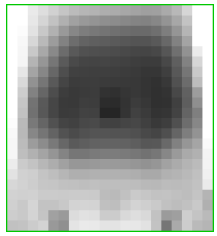
Figure A.14: Top rightmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 16×16.



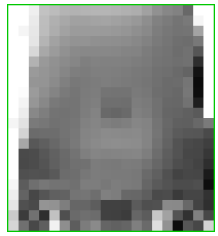
(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

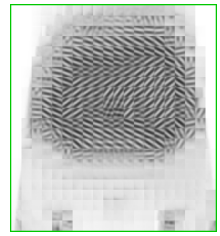
Figure A.15: Bottom rightmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 16×16.



(a) vertical features



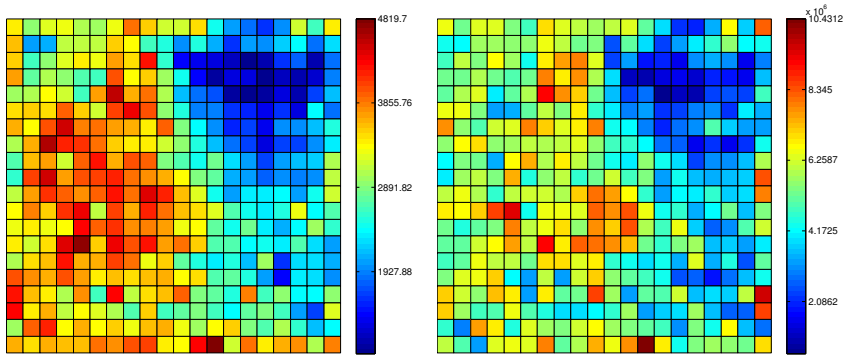
(b) horizontal features



(c) codebook blocks

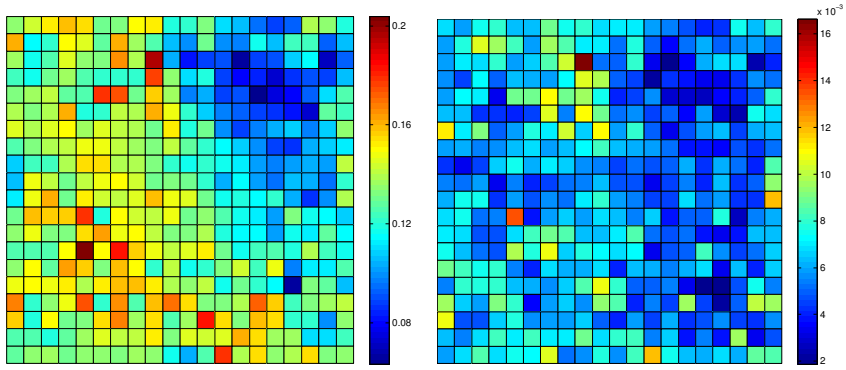
(d) example samples

Figure A.16: Bottom leftmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 16×16.



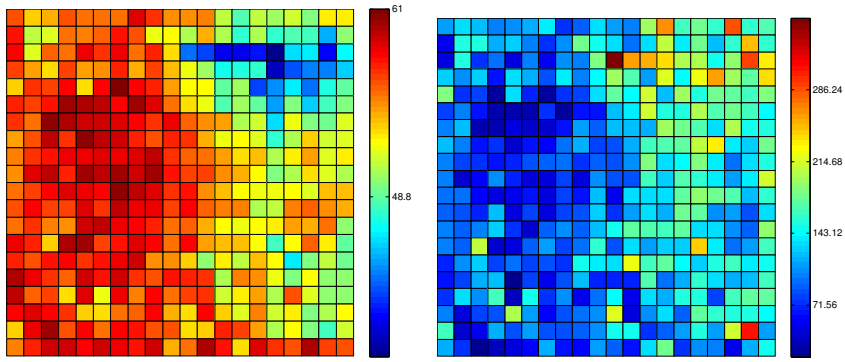
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

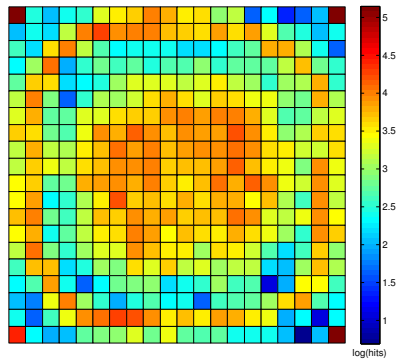
(d) prediction variances map; provider {63}



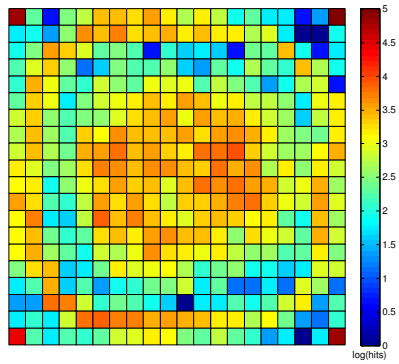
(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

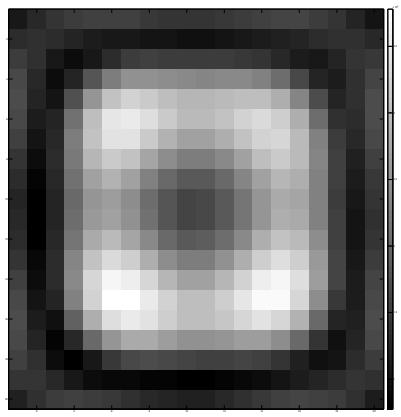
Figure A.17: The trained high-level GTM map for CASIAFPV5 and block size 16×16 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

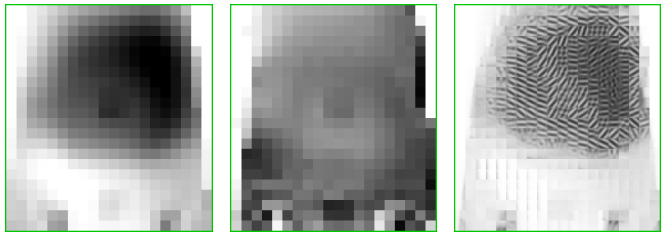


(b) log-hitmap (validation set)

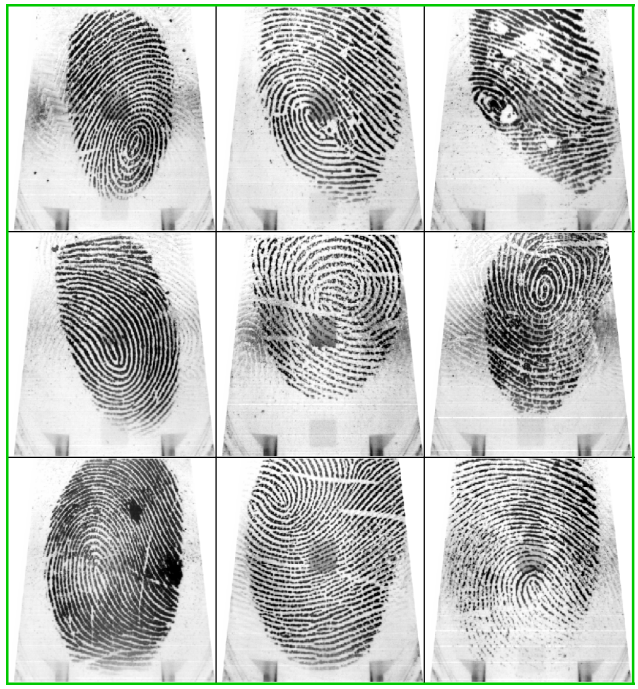


(c) distance matrix (magnification factors)

Figure A.18: Log-hitmaps and distance matrix for the trained high-level GTM map for CASIAFPV5 with block size 16×16, where the *posteriors' modes* are counted as *hits*.

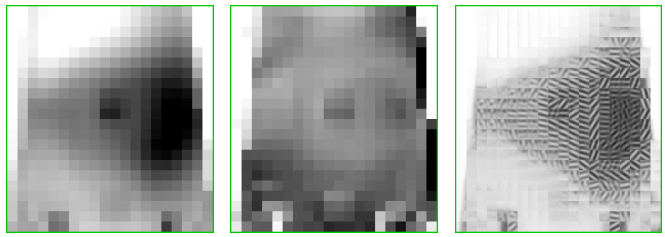


(a) vertical features **(b)** horizontal features **(c)** codebook blocks



(d) example samples

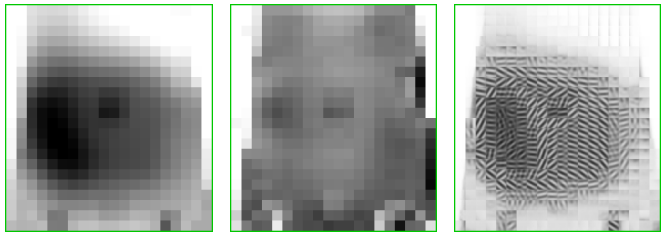
Figure A.19: Top leftmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 16×16.



(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

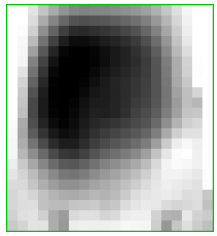
Figure A.20: Top rightmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 16×16.



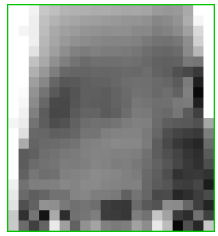
(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

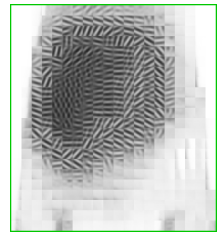
Figure A.21: Bottom rightmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 16×16.



(a) vertical features



(b) horizontal features

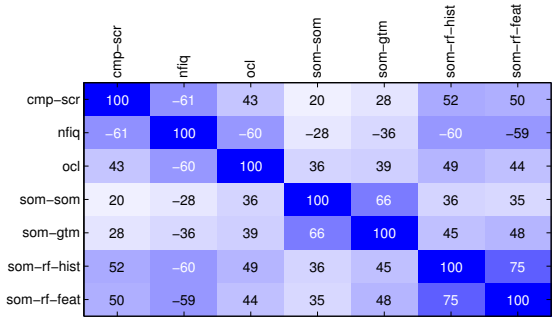


(c) codebook blocks

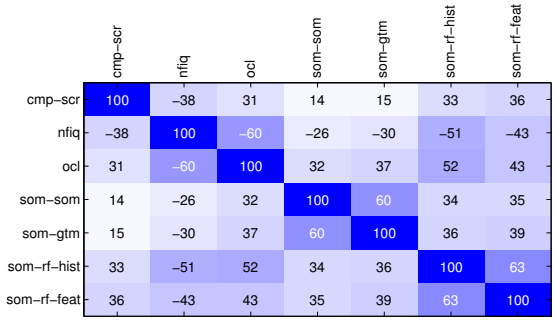
(d) example samples

Figure A.22: Bottom leftmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 16×16.

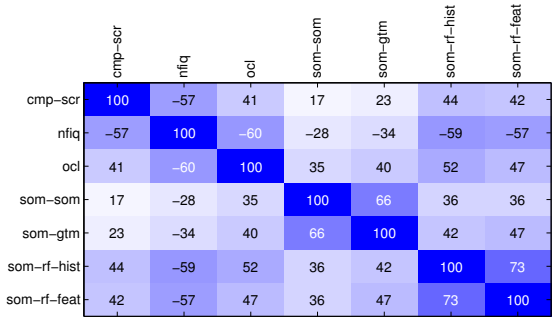
A.1.3 Final results



(a) provider {28}



(b) provider {63}



(c) provider {83}

Figure A.23: Correlation tables for CASIAFPV5 with block size 16×16.

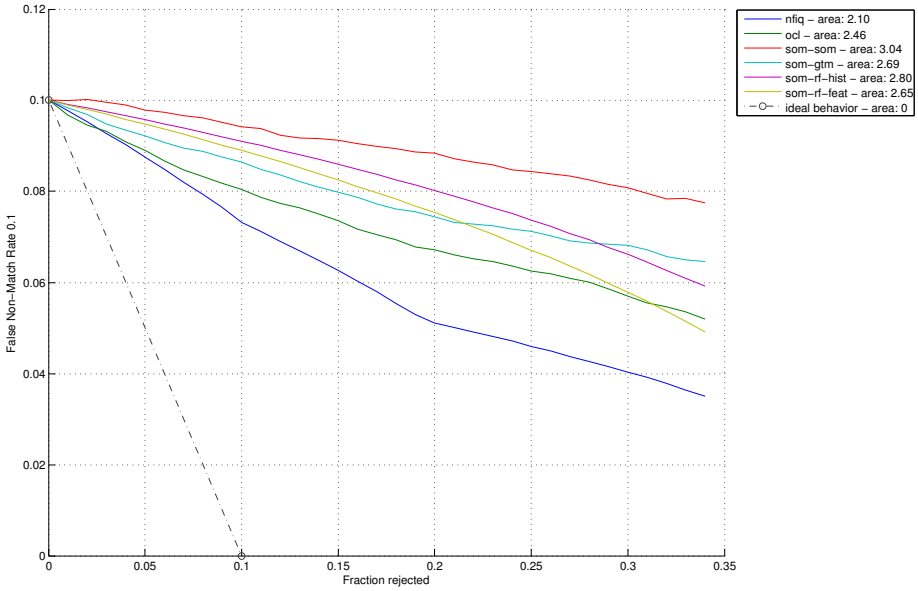


Figure A.24: ERC curves for CASIAFPV5 with block size 16×16 for provider {28}.

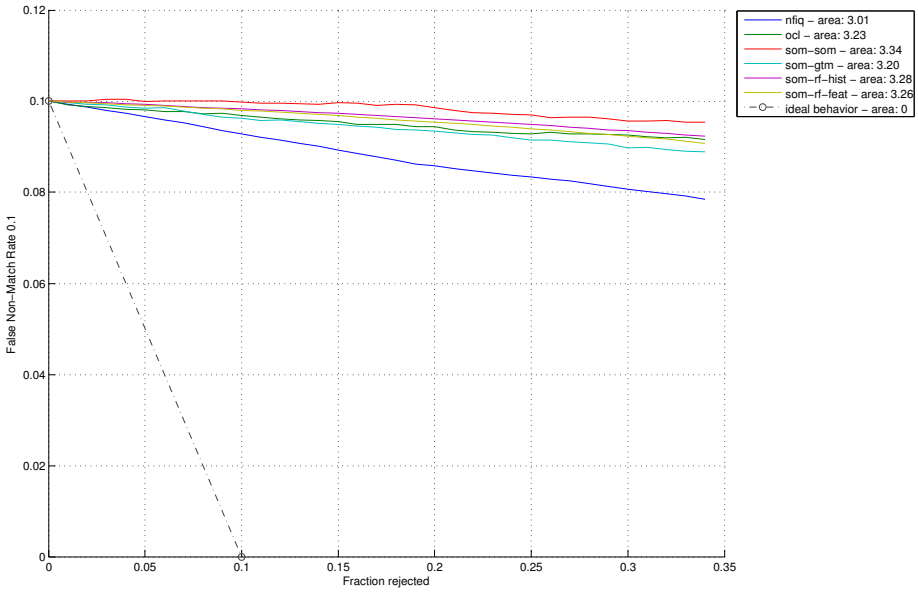


Figure A.25: ERC curves for CASIAFPV5 with block size 16×16 for provider {63}.

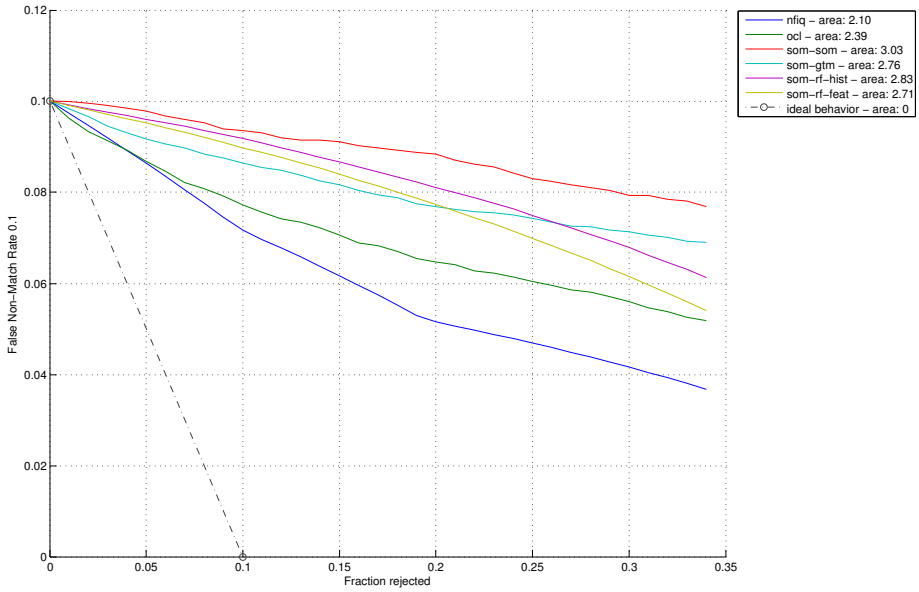
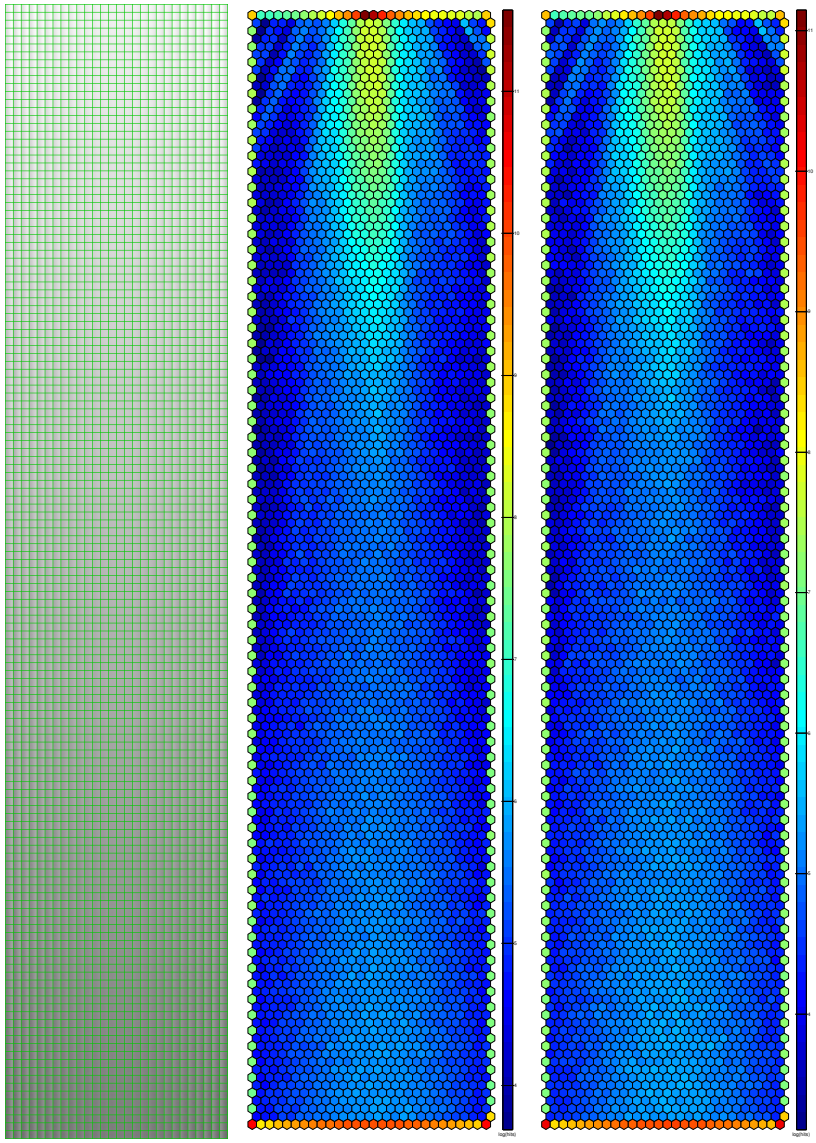


Figure A.26: ERC curves for CASIAFPV5 with block size 16×16 for provider {83}.

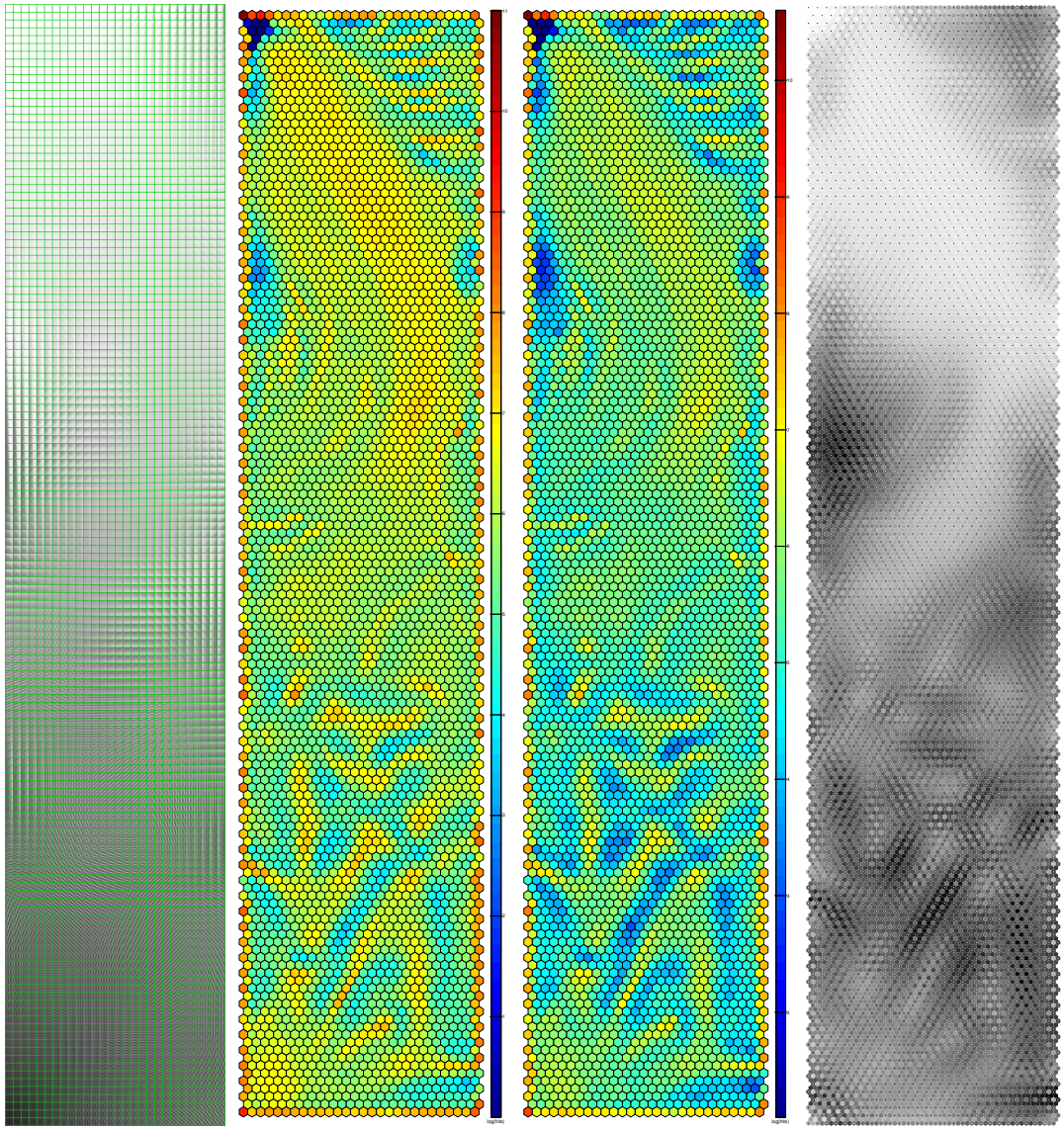
A.2 Block size 24×24

A.2.1 Low-level



(a) initial SOM map **(b)** initial SOM map's log-hitmap (training set) **(c)** initial SOM map's log-hitmap (validation set)

Figure A.27: Linearly initialized SOM map for dataset CASIAFPV5 and block size 24×24



(a) final SOM map

(b) final SOM map's log-hitmap (training set)

(c) final SOM map's log-hitmap (validation set)

(d) final SOM map's distance matrix (U-matrix)

Figure A.28: Results of the training for dataset CASIAFPV5 and block size 24×24

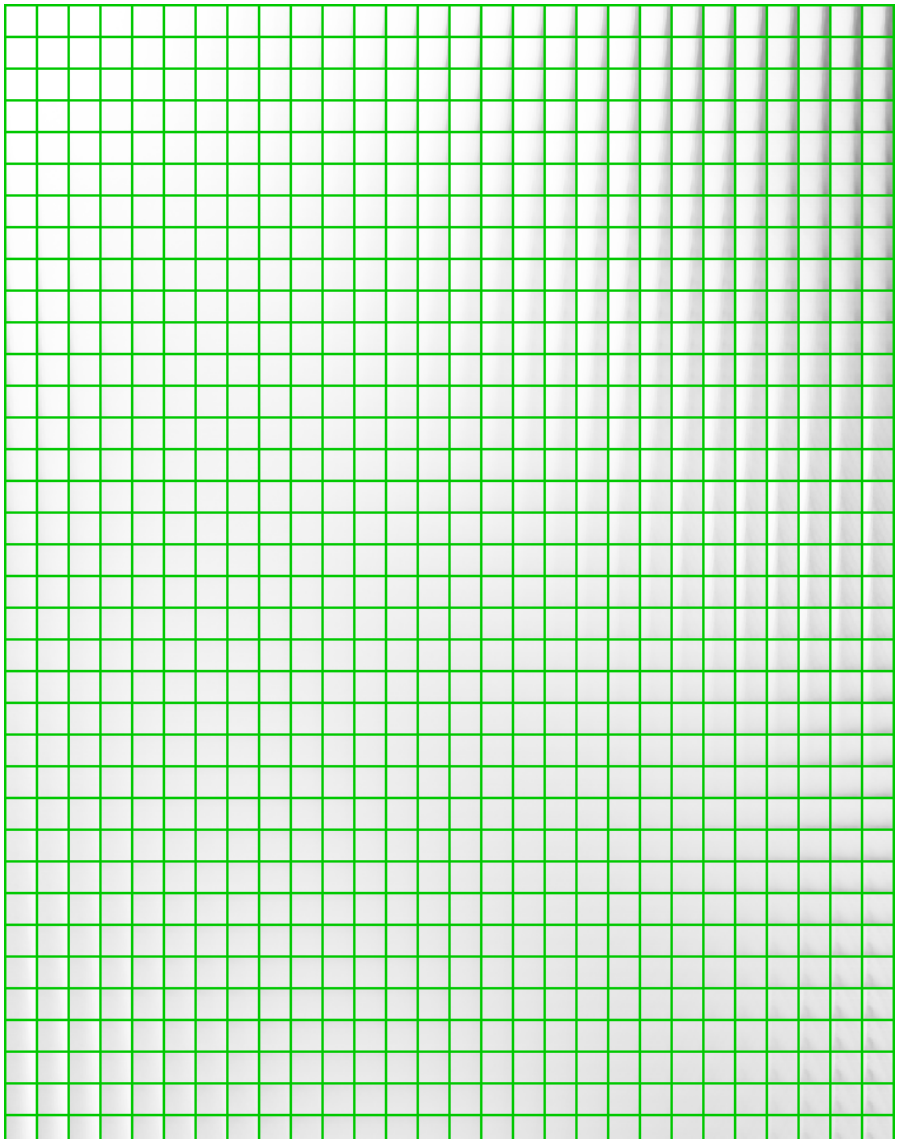


Figure A.29: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 24×24, first quarter of the map.

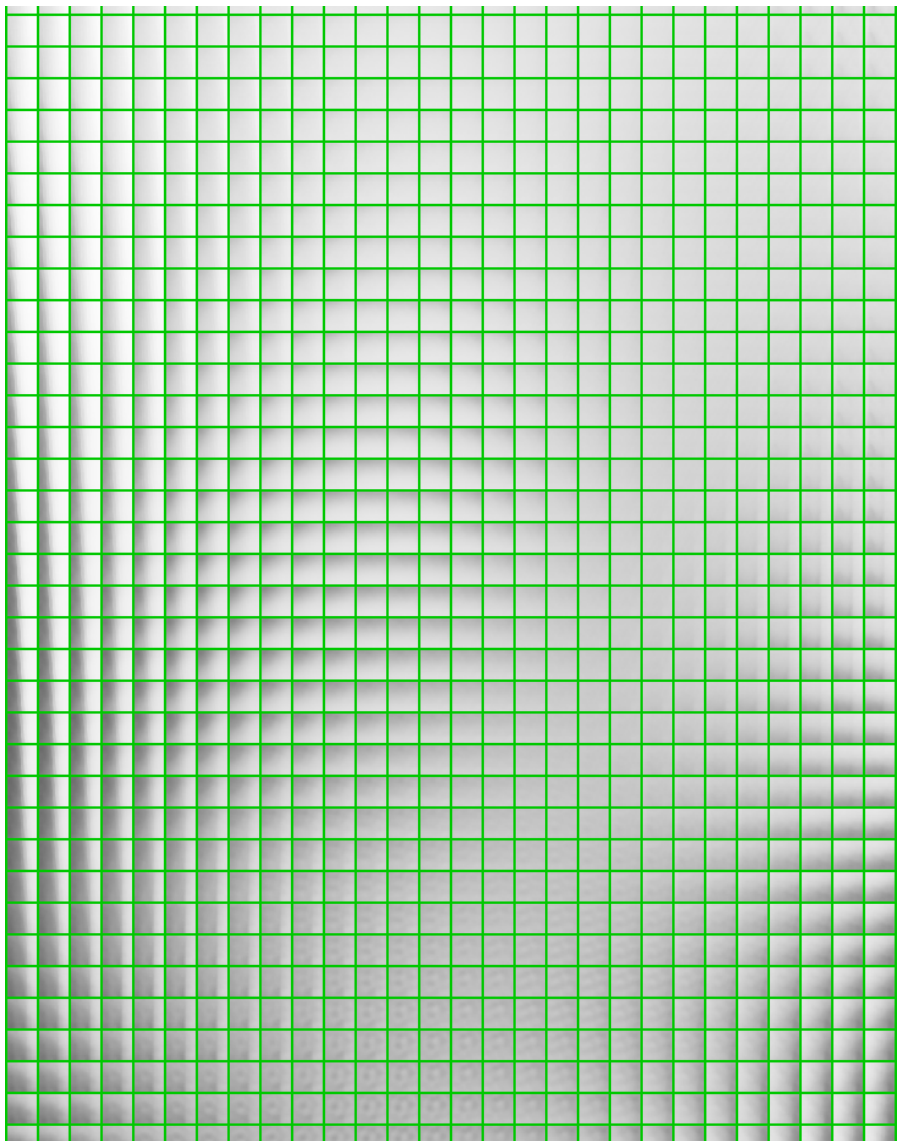


Figure A.30: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 24×24, second quarter of the map.

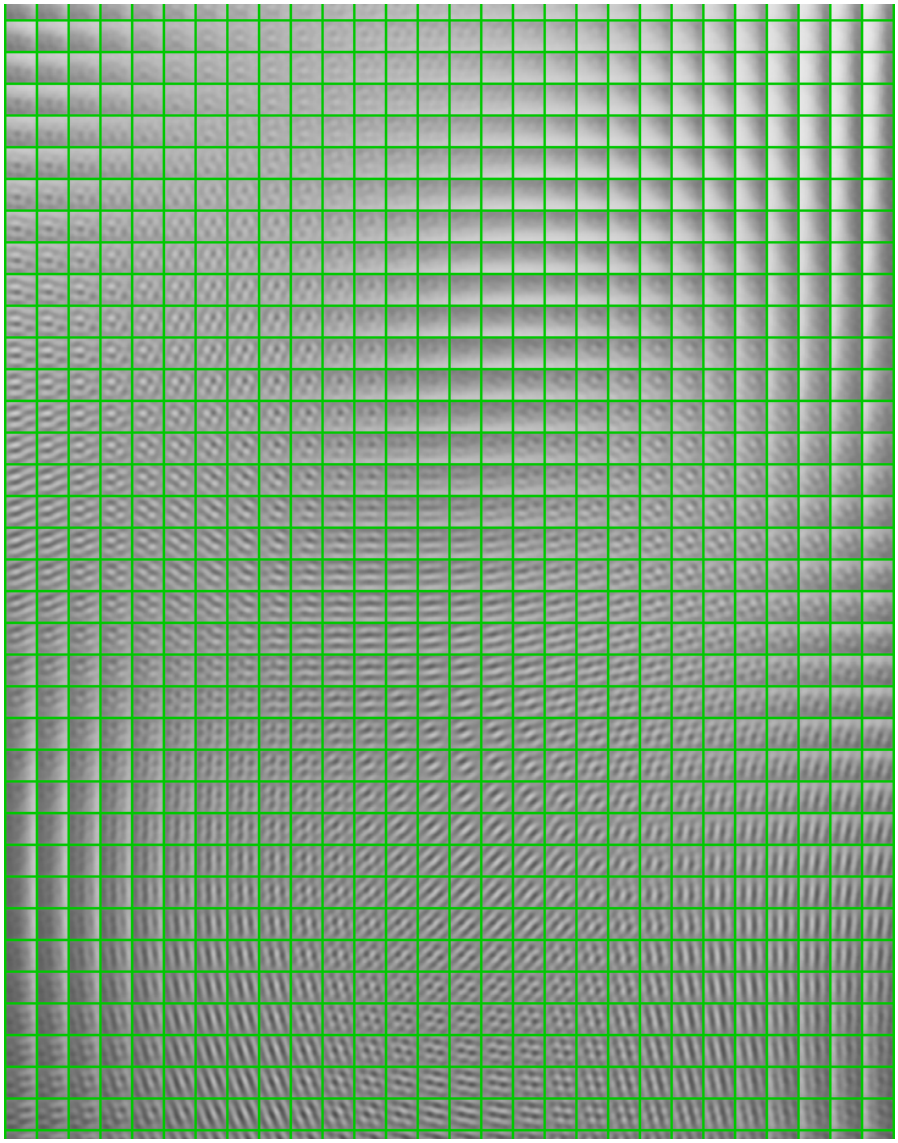


Figure A.31: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 24×24, third quarter of the map.

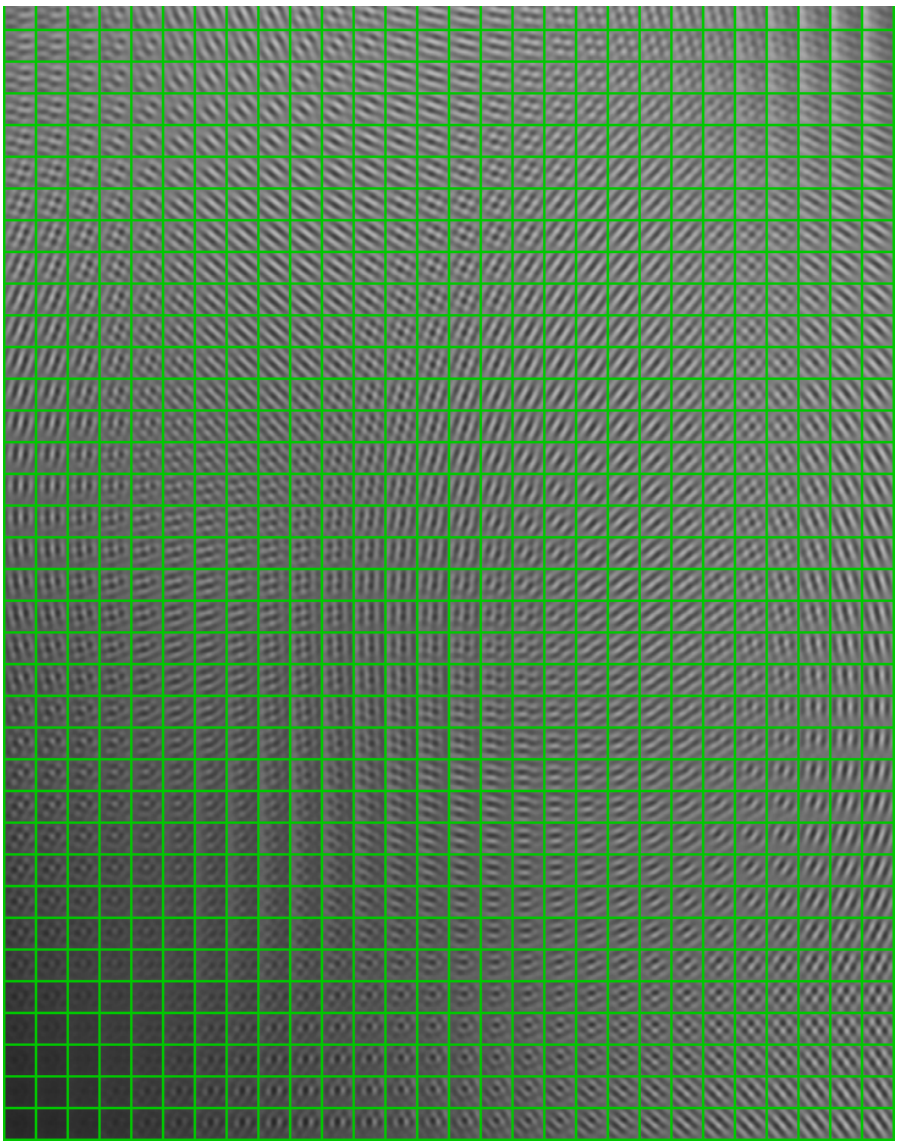


Figure A.32: Final SOM map codebook vectors for dataset CASIAFPV5 and block size 24×24, last quarter of the map.

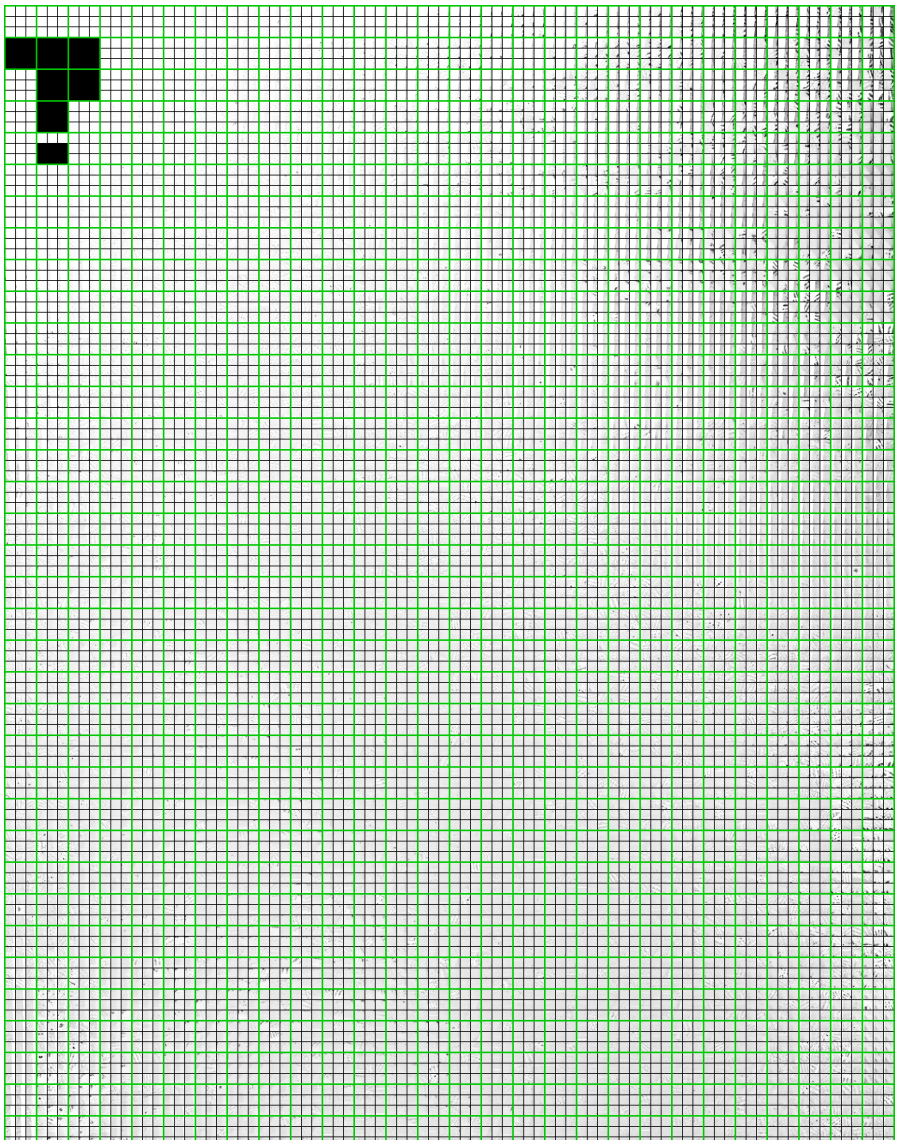


Figure A.33: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 24×24, first quarter of the map.

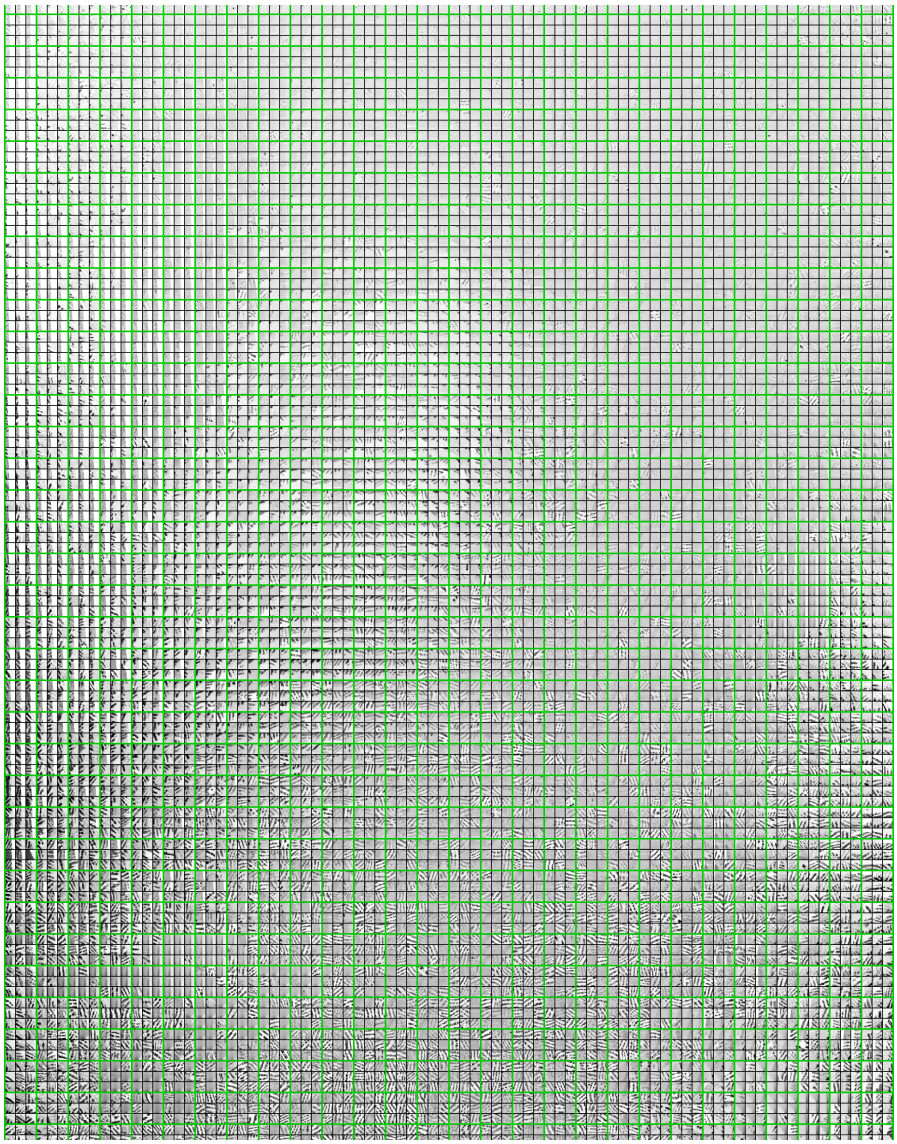


Figure A.34: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 24×24, second quarter of the map.

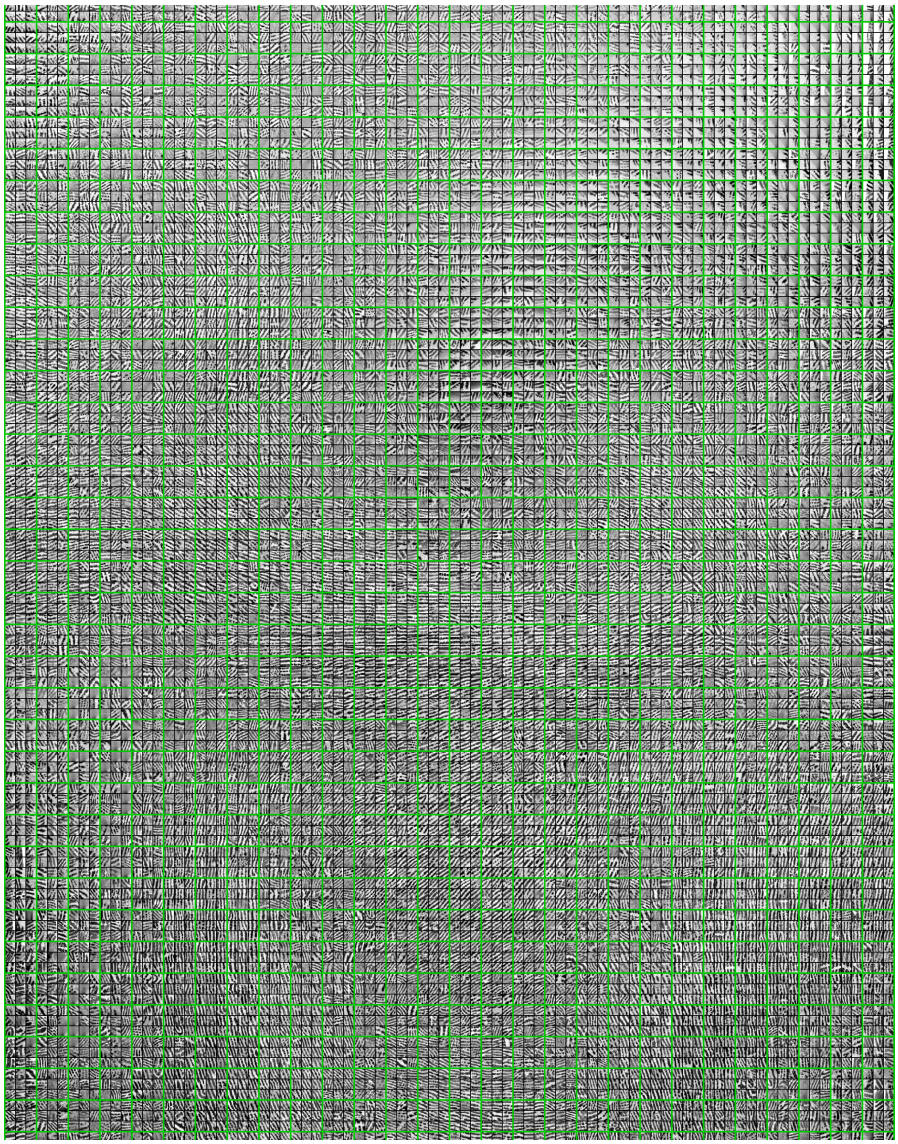


Figure A.35: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 24×24, third quarter of the map.

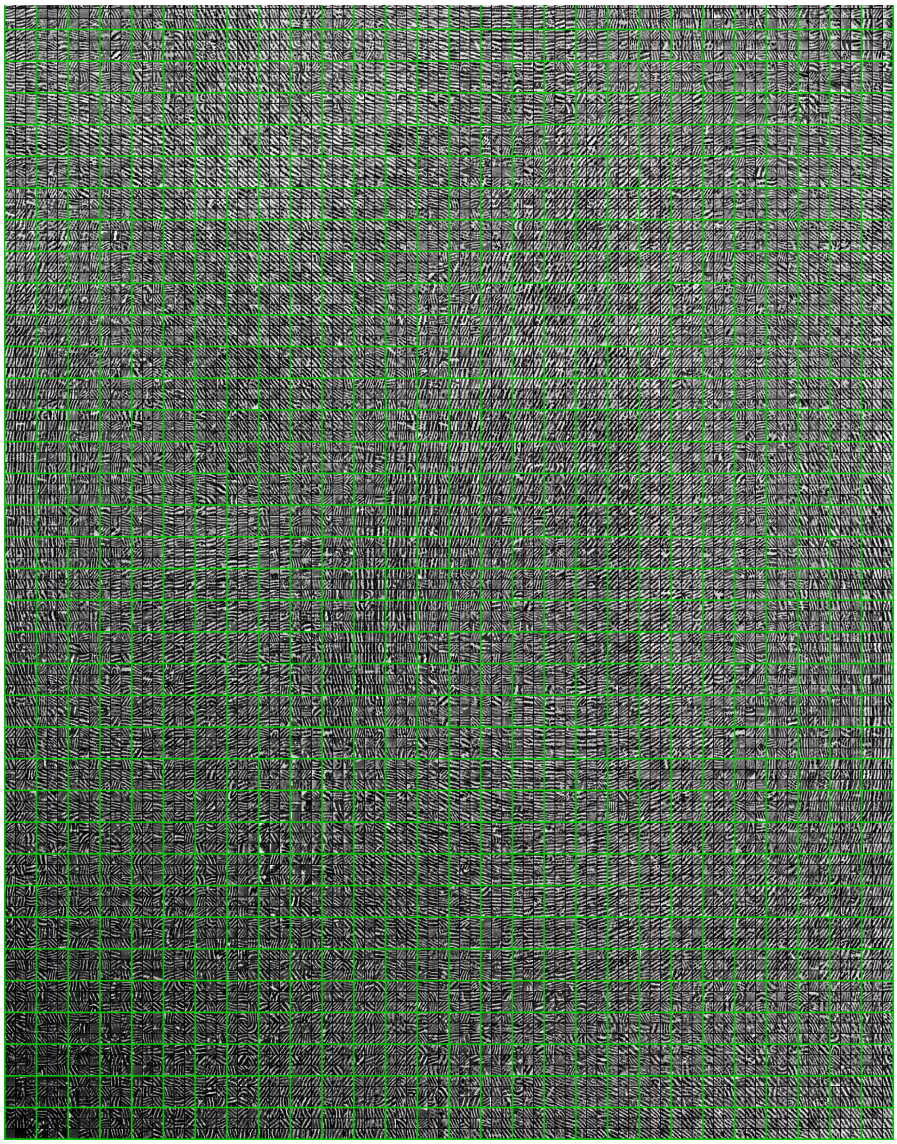


Figure A.36: Subsets of the samples mapped to each node of the final SOM map for dataset CASIAFPV5 and block size 24×24, last quarter of the map.

A.2.2 High-level

	1	2	3	4	5
1	804	505	337	174	90
2	279	303	287	229	134
3	233	303	315	316	197
4	185	268	378	454	480
5	114	143	231	406	835

(a) RF-feat; provider {28}

	1	2	3	4	5
1	1041	644	459	312	183
2	179	229	219	162	88
3	182	281	253	247	162
4	142	252	370	487	507
5	71	116	247	371	796

(b) RF-hist; provider {28}

	1	2	3	4	5
1	659	432	378	303	192
2	341	374	282	250	144
3	236	243	244	257	163
4	252	275	398	459	344
5	207	229	238	300	800

(c) RF-feat; provider {63}

	1	2	3	4	5
1	810	550	468	402	323
2	267	248	204	165	112
3	186	216	239	239	142
4	261	320	360	449	373
5	171	219	269	314	693

(d) RF-hist; provider {63}

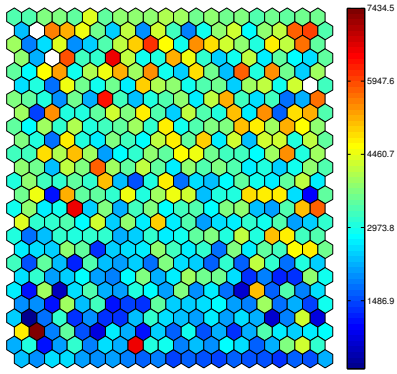
	1	2	3	4	5
1	828	467	330	240	99
2	316	292	240	249	182
3	176	240	281	230	231
4	205	283	301	348	361
5	177	243	374	488	819

(e) RF-feat; provider {83}

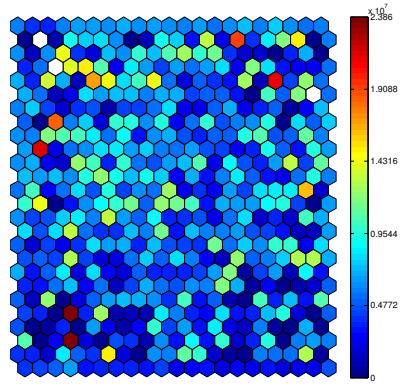
	1	2	3	4	5
1	1010	619	454	351	194
2	212	209	197	199	152
3	168	197	215	254	207
4	183	269	333	377	405
5	129	231	327	374	734

(f) RF-hist; provider {83}

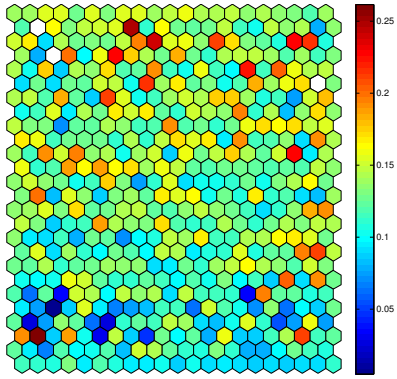
Table A.2: Confusion matrices for RF-feat (left) and RF-hist (right) for different providers and dataset CASIAFPV5 with block size 24×24.



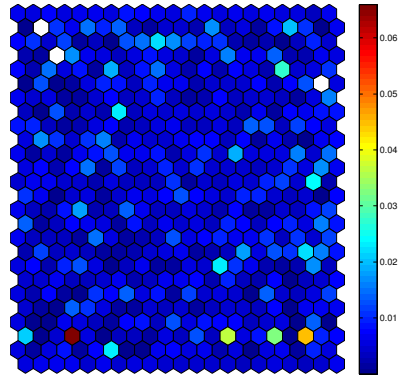
(a) predictions map; provider {28}



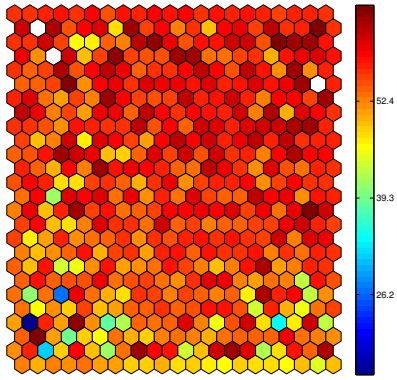
(b) prediction variances map; provider {28}



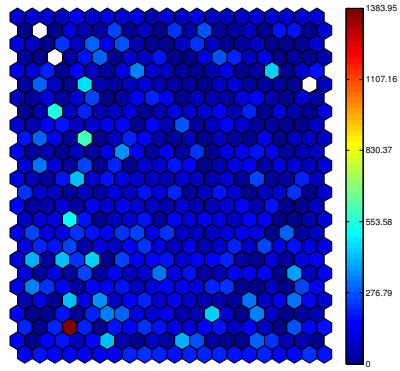
(c) predictions map; provider {63}



(d) prediction variances map; provider {63}

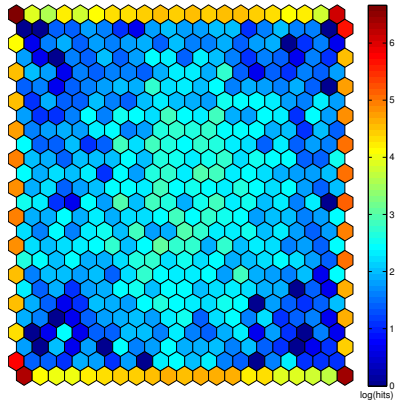


(e) predictions map; provider {83}

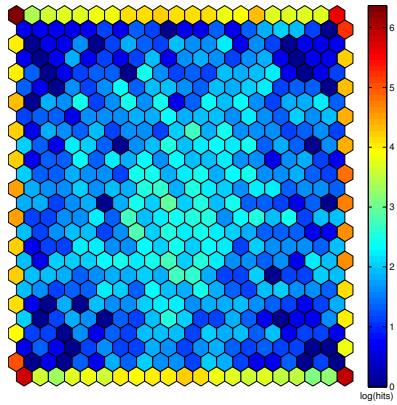


(f) prediction variances map; provider {83}

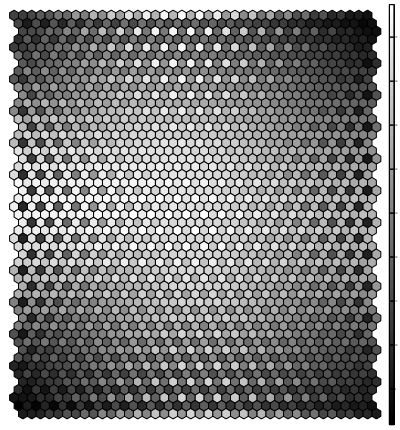
Figure A.37: The trained high-level SOM map for CASIAFPV5 and block size 24 × 24 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

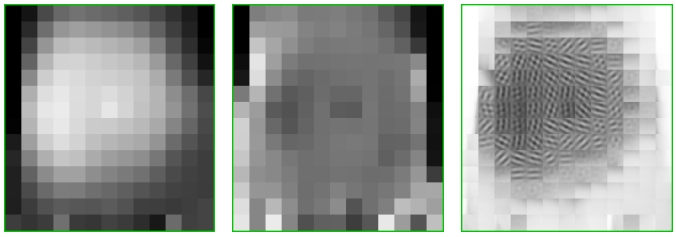


(b) log-hitmap (validation set)



(c) distance matrix (U-matrix)

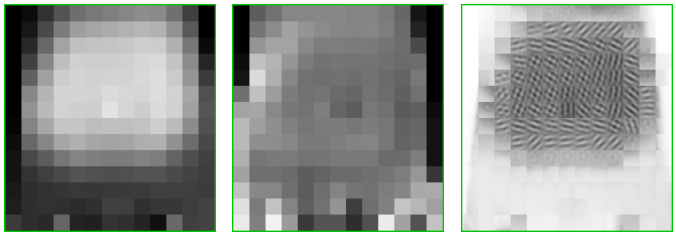
Figure A.38: Log-hitmaps and distance matrix for the trained high-level SOM map for CASIAFPV5 with block size 24×24.



(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

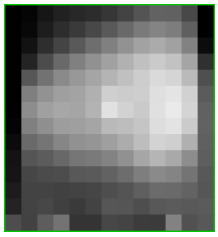
Figure A.39: Top leftmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 24×24.



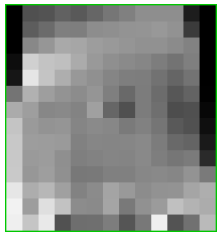
(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

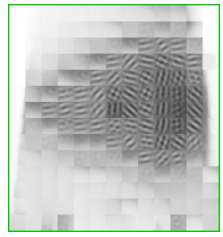
Figure A.40: Top rightmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 24×24.



(a) vertical features



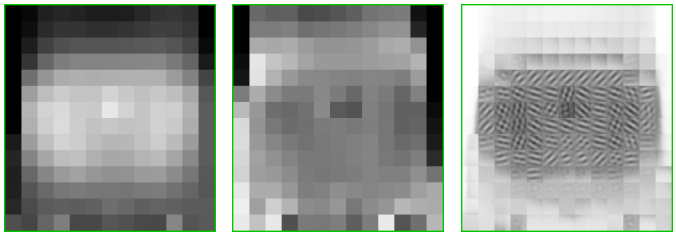
(b) horizontal features



(c) codebook vectors

(d) example samples

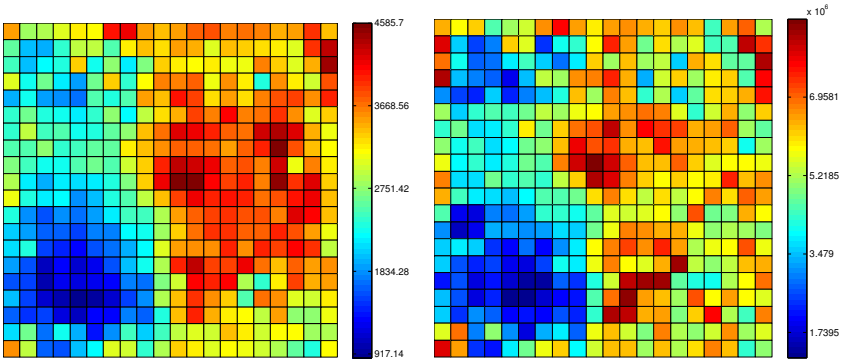
Figure A.41: Bottom rightmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 24×24.



(a) vertical features **(b)** horizontal features **(c)** codebook vectors

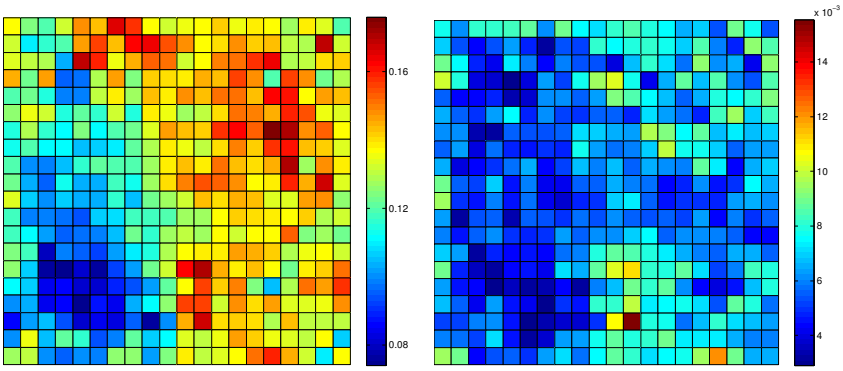
(d) example samples

Figure A.42: Bottom leftmost node of the trained high-level SOM map for dataset CASIAFPV5 with block size 24×24.



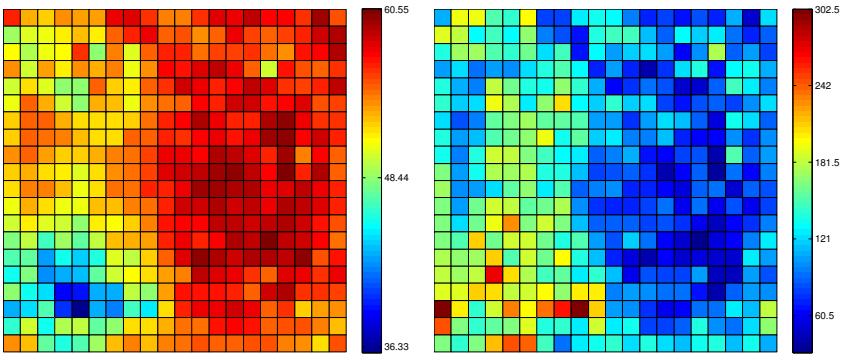
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

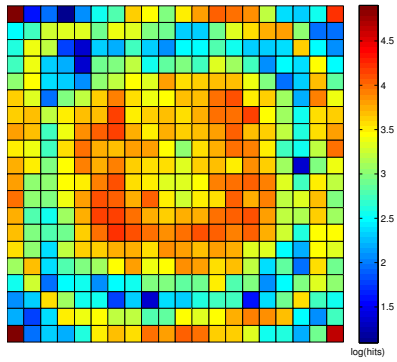
(d) prediction variances map; provider {63}



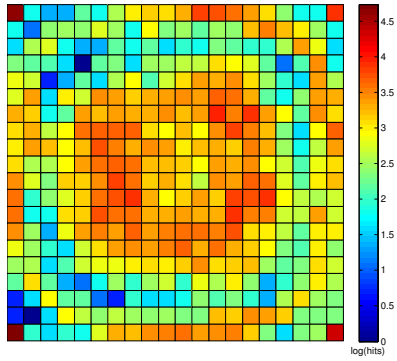
(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

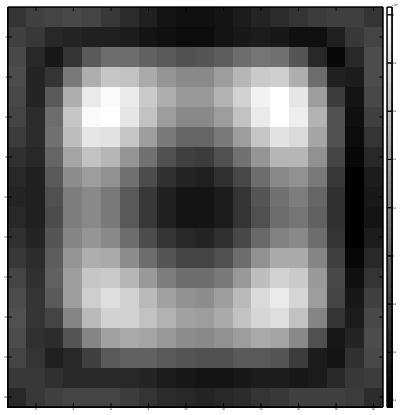
Figure A.43: The trained high-level GTM map for CASIAFPV5 and block size 24×24 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

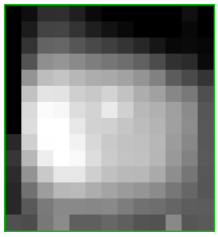


(b) log-hitmap (validation set)

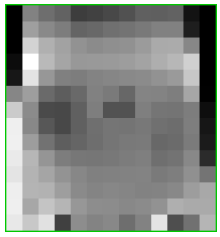


(c) distance matrix (magnification factors)

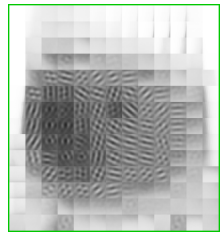
Figure A.44: Log-hitmaps and distance matrix for the trained high-level GTM map for CASIAFPV5 with block size 24×24, where the *posteriors' modes* are counted as *hits*.



(a) vertical features



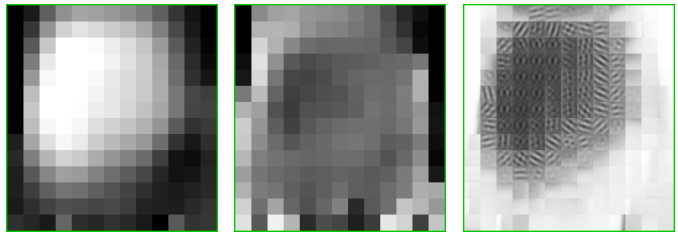
(b) horizontal features



(c) codebook vectors

(d) example samples

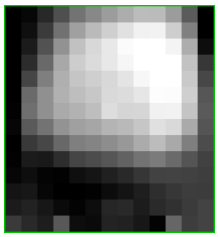
Figure A.45: Top leftmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 24×24.



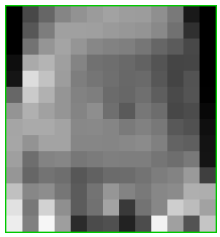
(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

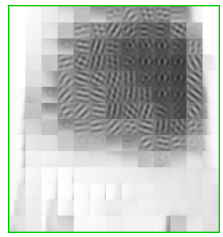
Figure A.46: Top rightmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 24×24.



(a) vertical features



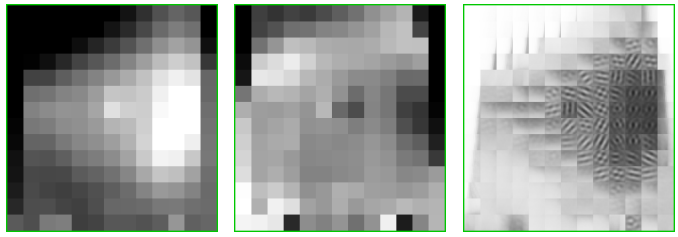
(b) horizontal features



(c) codebook vectors

(d) example samples

Figure A.47: Bottom rightmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 24×24.

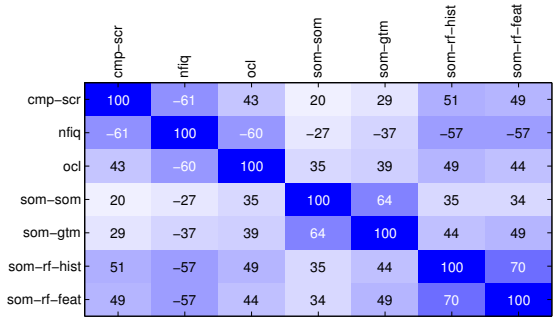


(a) vertical features **(b)** horizontal features **(c)** codebook vectors

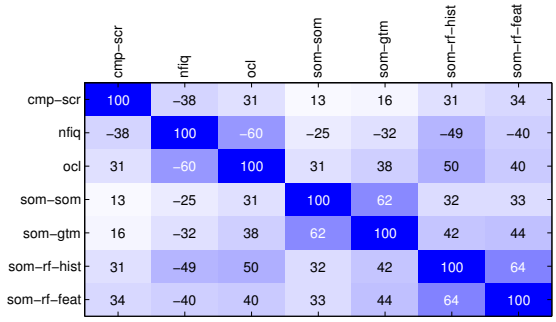
(d) example samples

Figure A.48: Bottom leftmost node of the trained high-level GTM map for dataset CASIAFPV5 with block size 24×24.

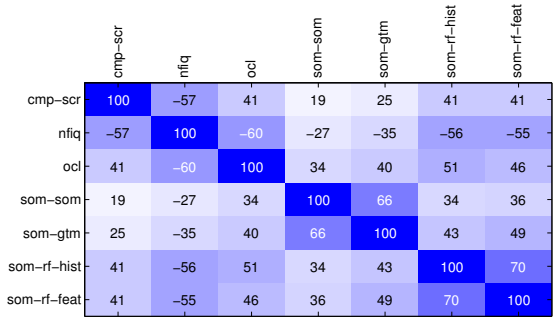
A.2.3 Final results



(a) provider {28}



(b) provider {63}



(c) provider {83}

Figure A.49: Correlation tables for CASIAFPV5 with block size 24×24.

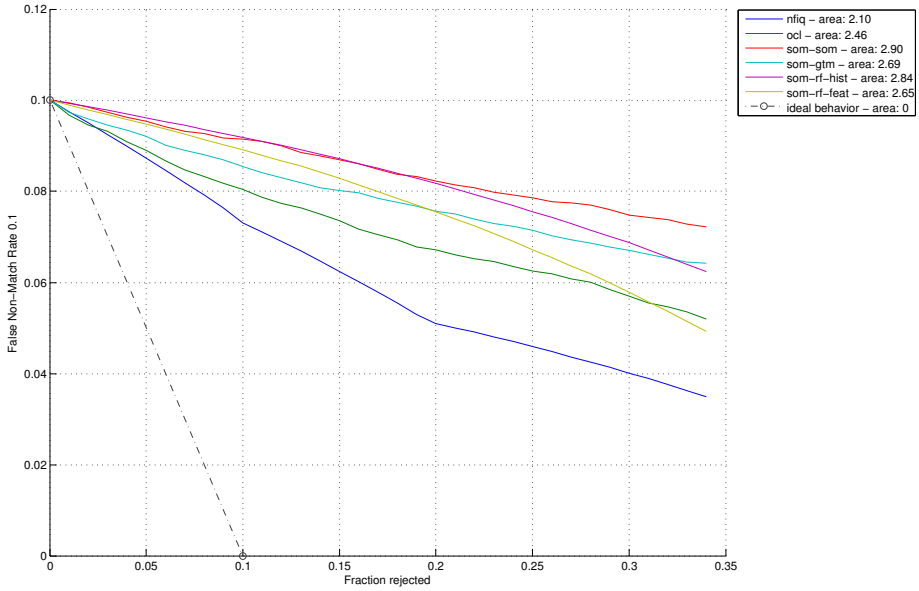


Figure A.50: ERC curves for CASIAFPV5 with block size 24×24 for provider {28}.

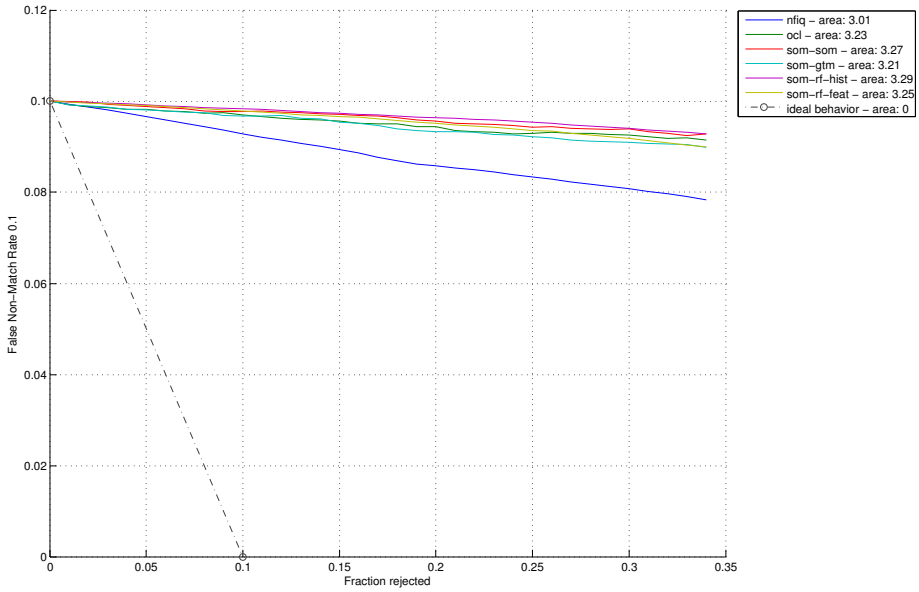


Figure A.51: ERC curves for CASIAFPV5 with block size 24×24 for provider {63}.

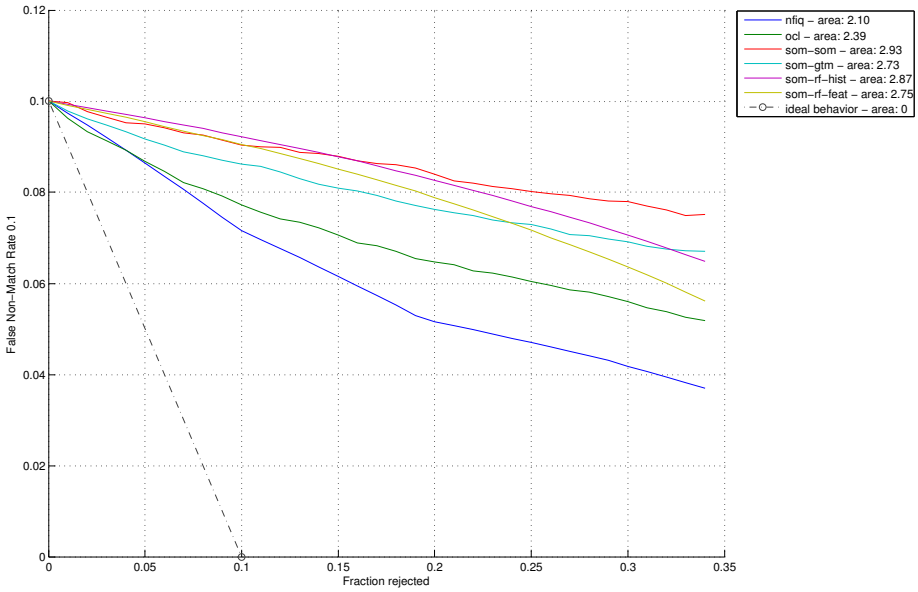


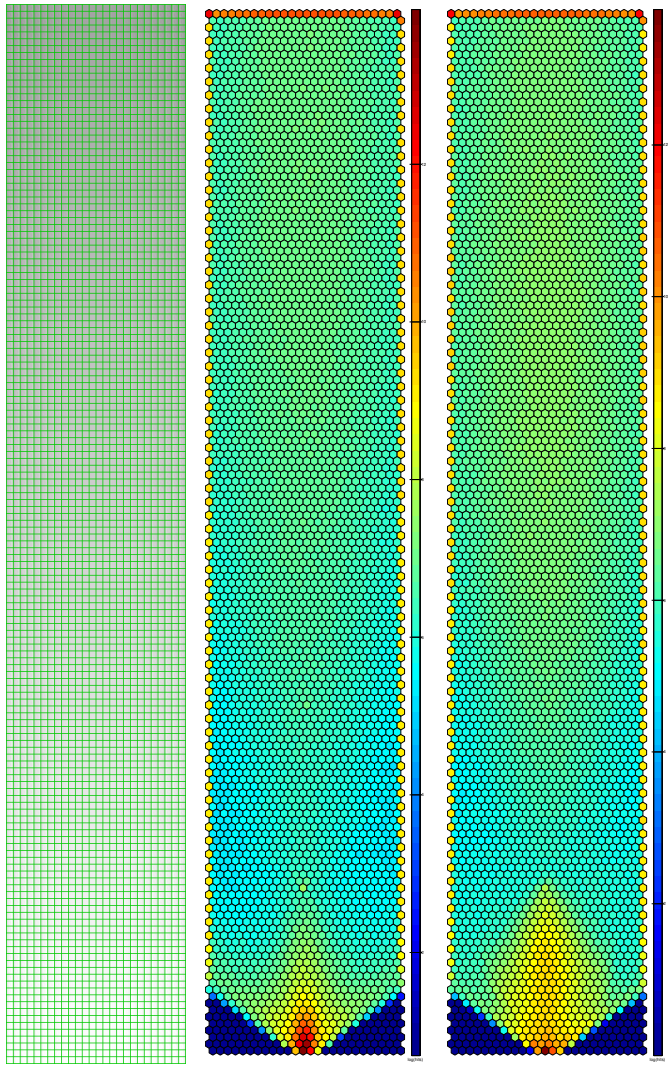
Figure A.52: ERC curves for CASIAFPV5 with block size 24×24 for provider {83}.

Appendix B

Results for MCYT330-DP

B.1 Block size 16×16

B.1.1 Low-level

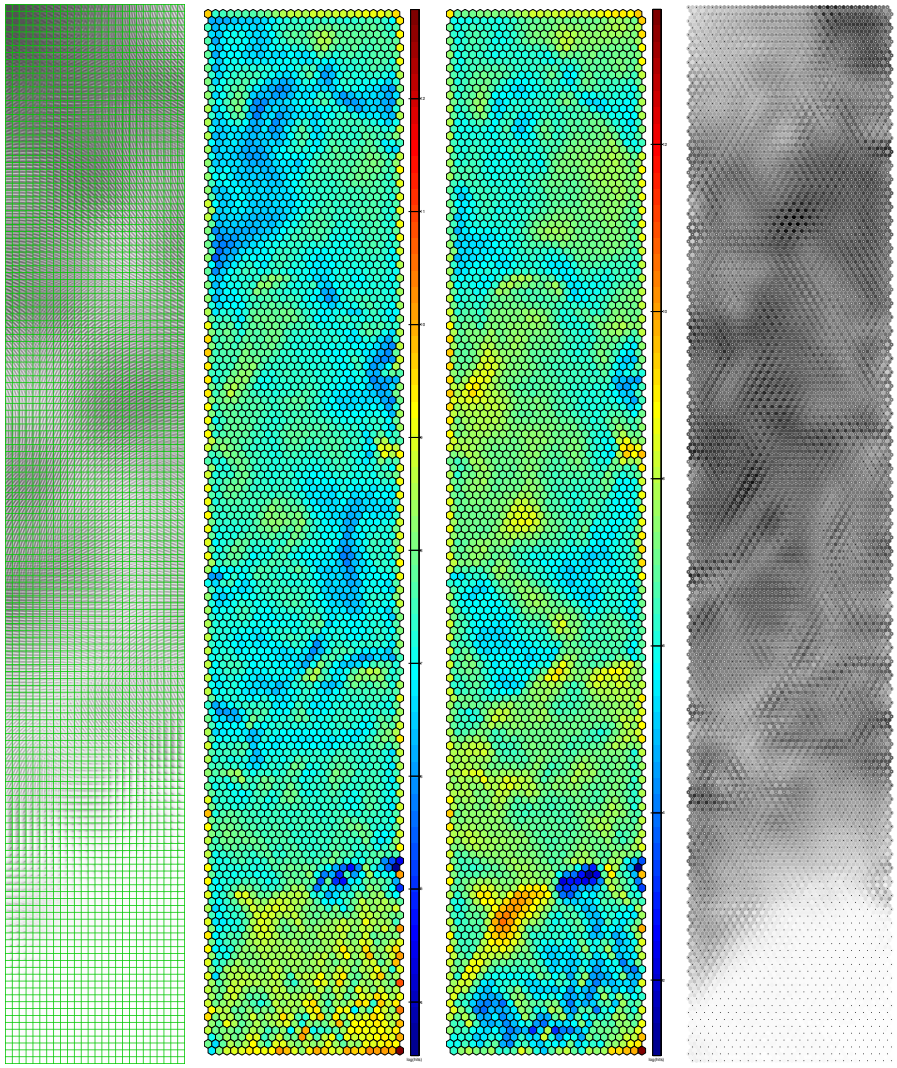


(a) initial SOM map

(b) initial SOM map's log-hitmap (training set)

(c) initial SOM map's log-hitmap (validation set)

Figure B.1: Linearly initialized SOM map for dataset MCYT330-DP and block size 16×16



(a) final SOM map

(b) final SOM map's log-hitmap (training set)

(c) final SOM map's log-hitmap (validation set)

(d) final SOM map's distance matrix (U-matrix)

Figure B.2: Results of the training for dataset MCYT330-DP and block size 16×16

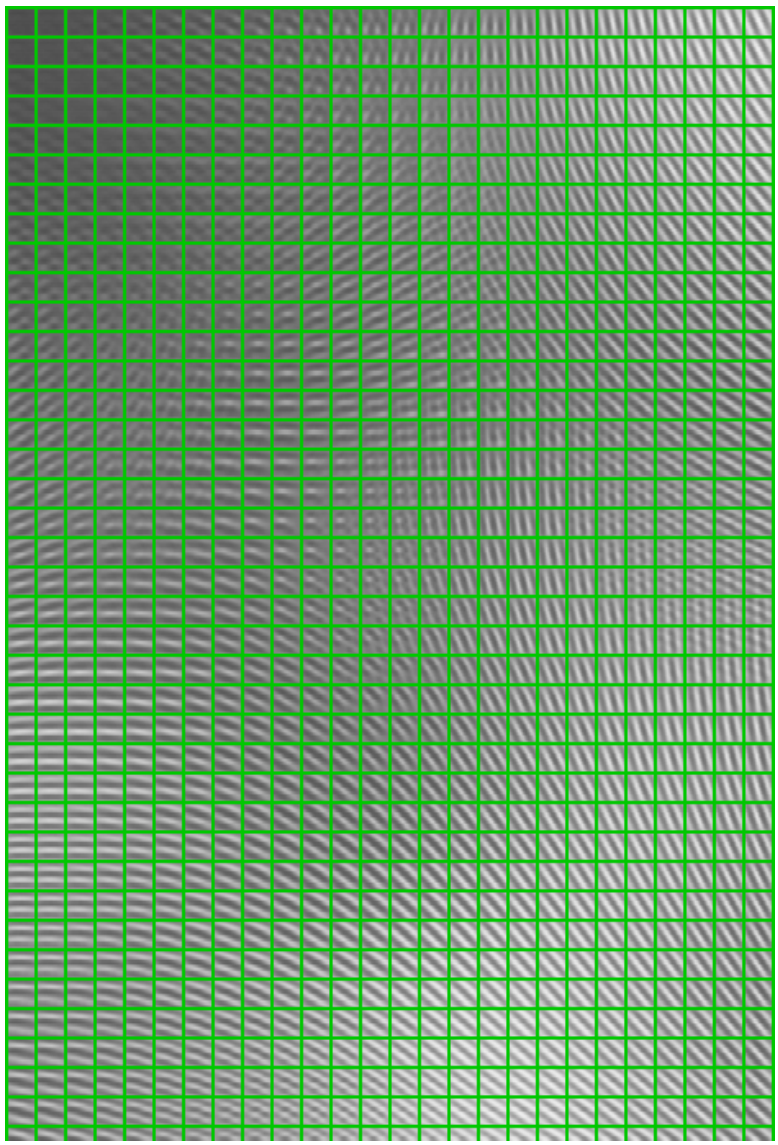


Figure B.3: Final SOM map codebook vectors for dataset MCYT330-DP and block size 16×16, first quarter of the map.

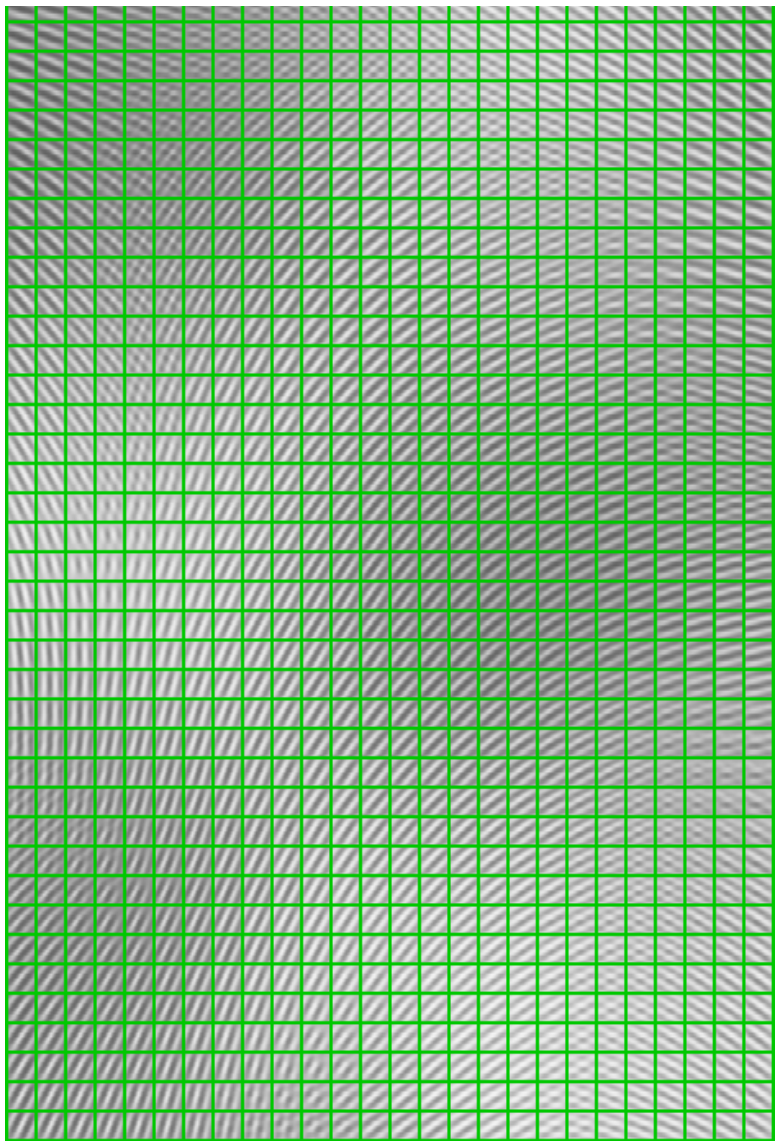


Figure B.4: Final SOM map codebook vectors for dataset MCYT330-DP and block size 16×16, second quarter of the map.

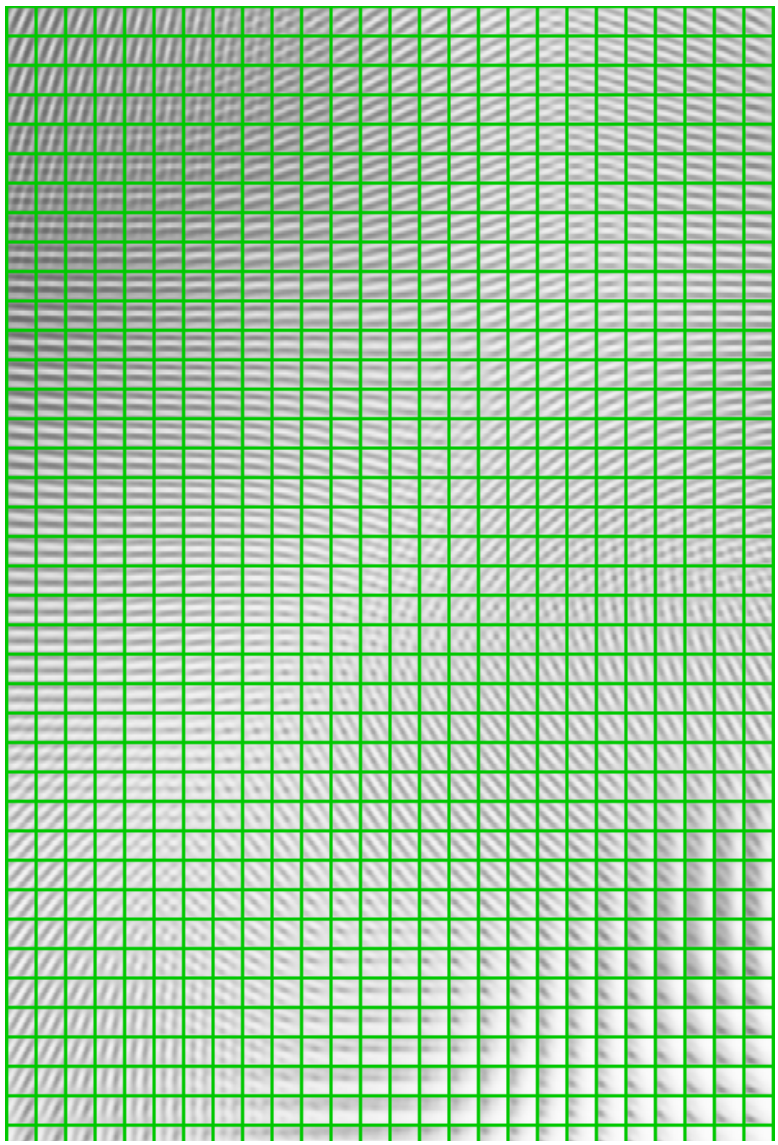


Figure B.5: Final SOM map codebook vectors for dataset MCYT330-DP and block size 16×16, third quarter of the map.

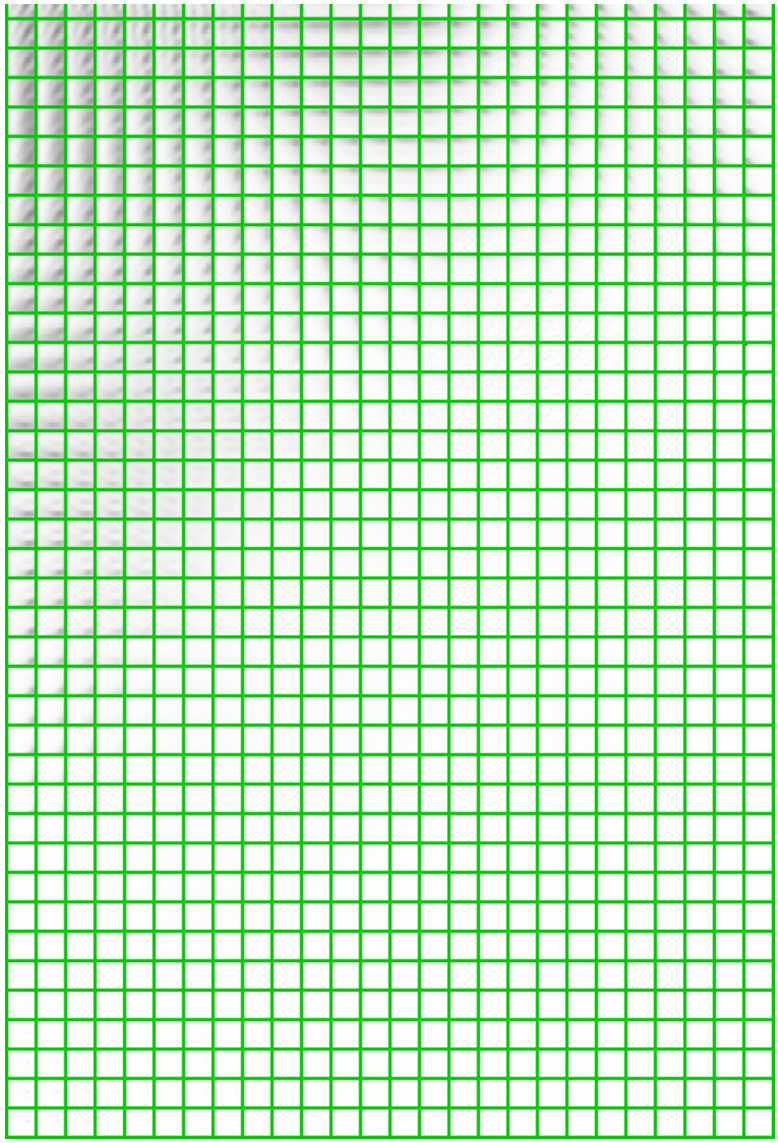


Figure B.6: Final SOM map codebook vectors for dataset MCYT330-DP and block size 16×16, last quarter of the map.

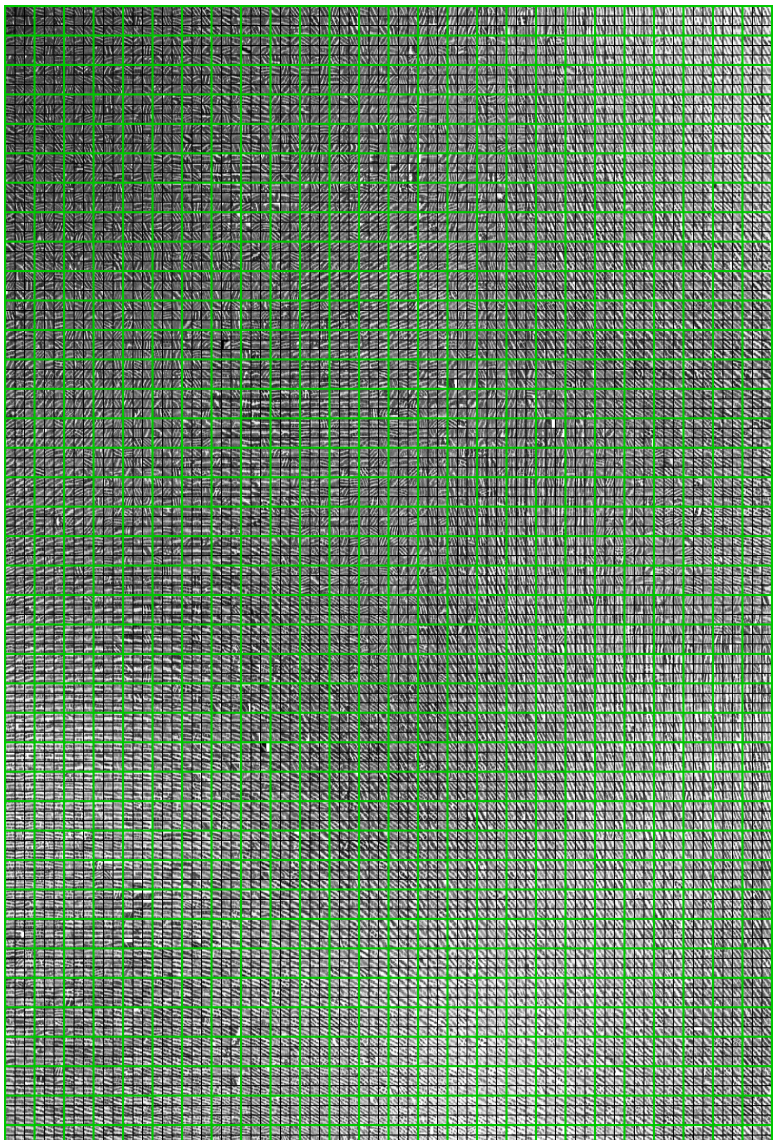


Figure B.7: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 16×16, first quarter of the map.

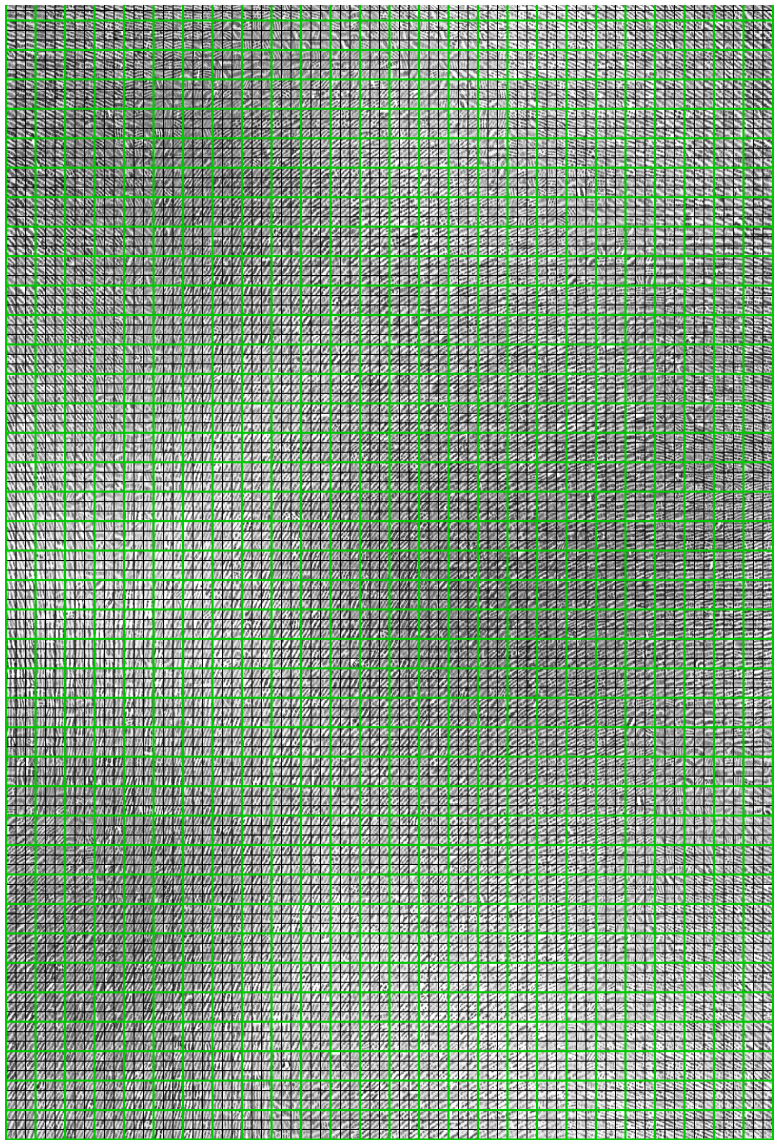


Figure B.8: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 16×16, second quarter of the map.

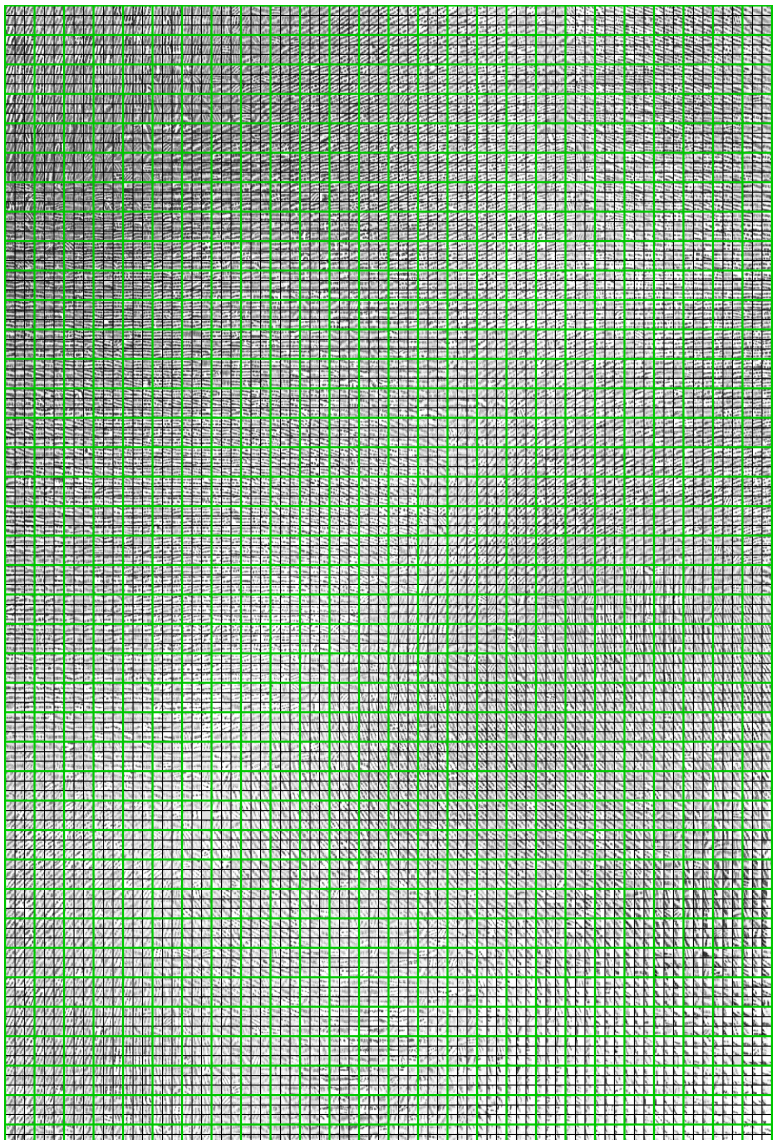


Figure B.9: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 16×16, third quarter of the map.

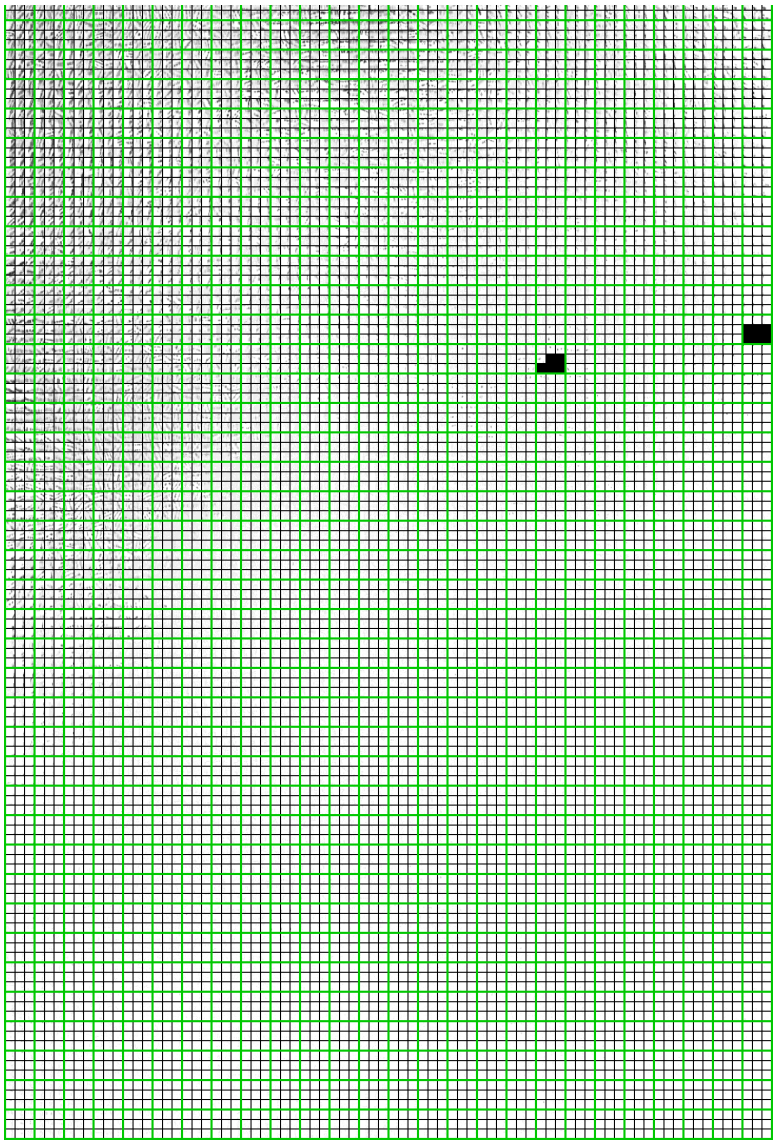


Figure B.10: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 16×16, last quarter of the map.

B.1.2 High-level

	1	2	3	4	5
1	2409	1438	863	381	237
2	751	1021	910	736	664
3	197	435	497	492	473
4	28	133	181	266	272
5	109	393	762	822	1368

(a) RF-feat; provider {28}

	1	2	3	4	5
1	2621	1361	713	355	159
2	428	647	552	401	238
3	196	428	523	393	306
4	58	193	183	187	192
5	191	791	1242	1361	2119

(b) RF-hist; provider {28}

	1	2	3	4	5
1	2750	1790	1145	766	320
2	959	1341	1166	710	269
3	387	586	634	597	406
4	71	166	204	367	384
5	33	66	148	213	360

(c) RF-feat; provider {63}

	1	2	3	4	5
1	2612	1547	914	566	207
2	1287	1828	1626	1125	411
3	194	355	408	367	272
4	80	172	253	422	558
5	27	47	96	173	291

(d) RF-hist; provider {63}

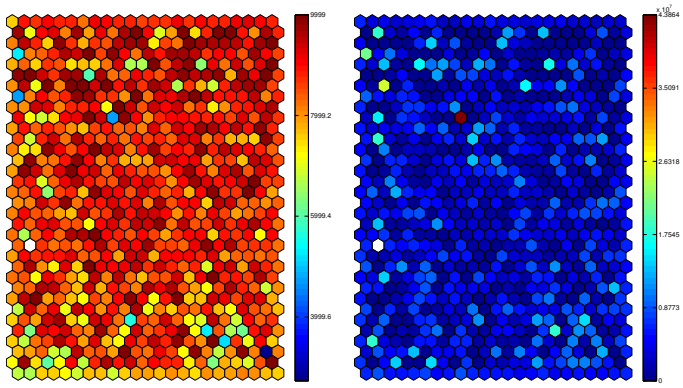
	1	2	3	4	5
1	2337	1591	1125	547	208
2	563	906	758	423	176
3	406	677	864	823	490
4	154	289	498	554	441
5	149	197	314	535	813

(e) RF-feat; provider {83}

	1	2	3	4	5
1	2565	1769	1194	542	182
2	449	830	836	523	192
3	272	484	669	635	341
4	148	296	466	478	370
5	175	281	394	704	1043

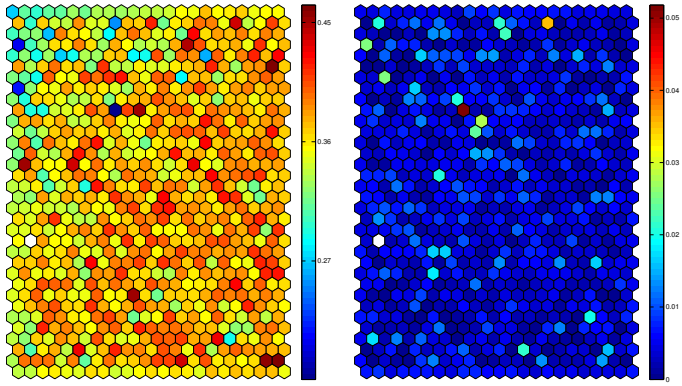
(f) RF-hist; provider {83}

Table B.1: Confusion matrices for RF-feat (left) and RF-hist (right) for different providers and dataset MCYT330-DP with block size 16×16.



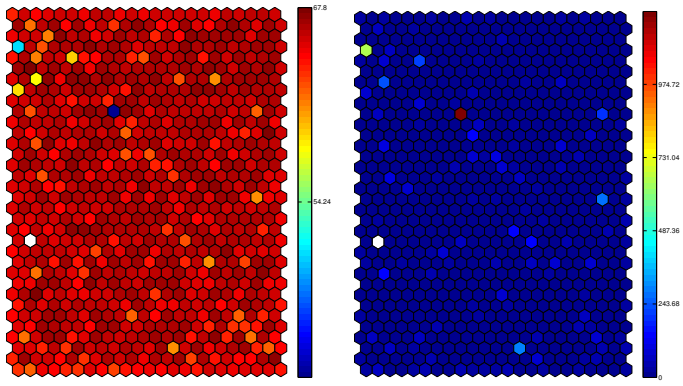
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

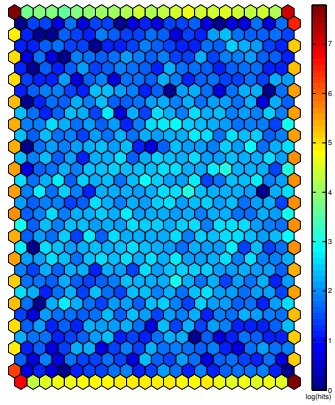
(d) prediction variances map; provider {63}



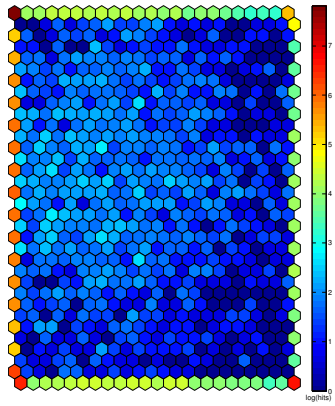
(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

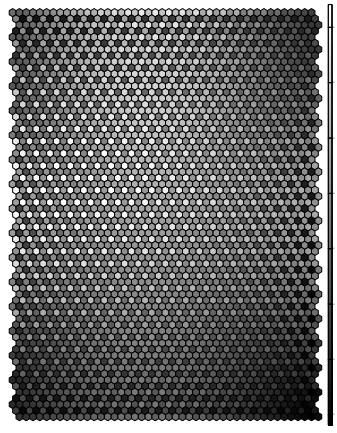
Figure B.11: The trained high-level SOM map for MCYT330-DP and block size 16×16 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

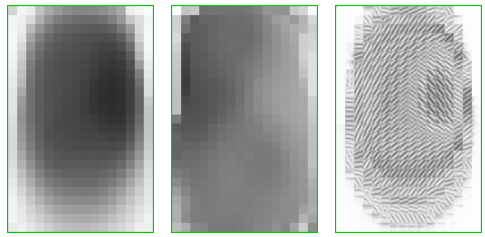


(b) log-hitmap (validation set)



(c) distance matrix (U-matrix)

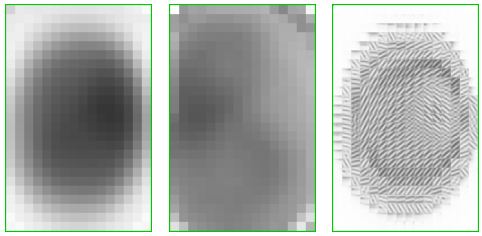
Figure B.12: Log-hitmaps and distance matrix for the trained high-level SOM map for MCYT330-DP with block size 16×16.



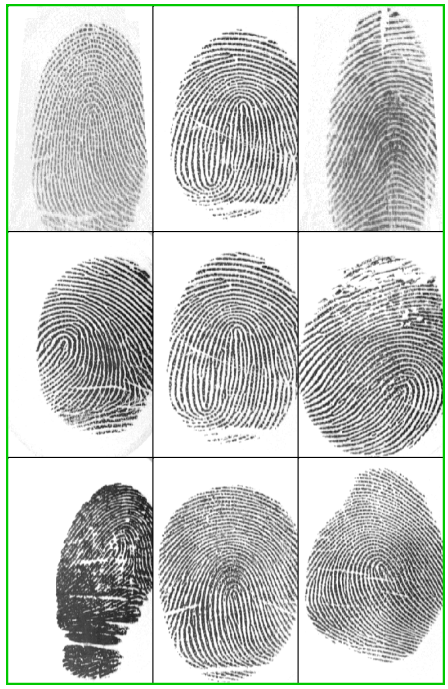
(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

Figure B.13: Top leftmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 16×16.

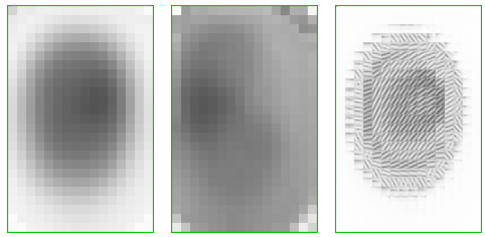


(a) vertical features **(b)** horizontal features **(c)** codebook blocks

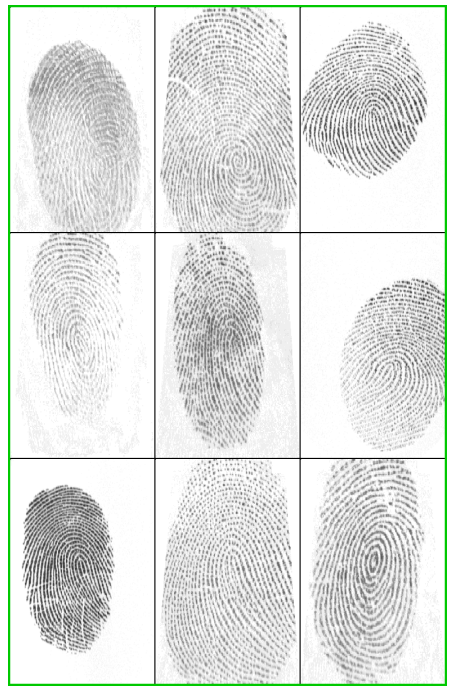


(d) example samples

Figure B.14: Top rightmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 16×16.

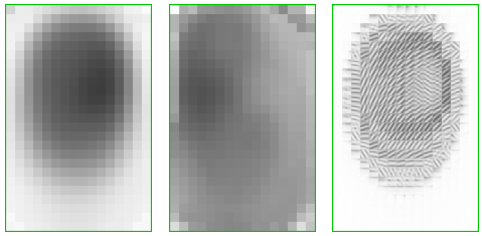


(a) vertical features **(b)** horizontal features **(c)** codebook blocks



(d) example samples

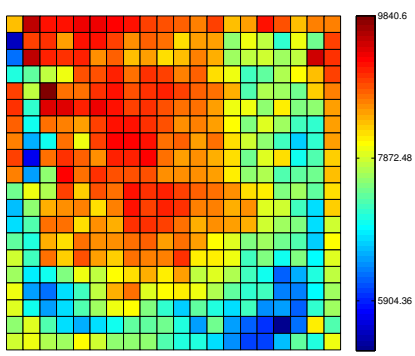
Figure B.15: Bottom rightmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 16×16.



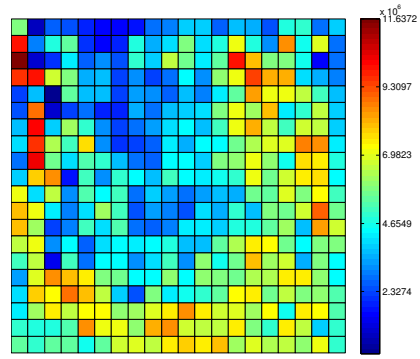
(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

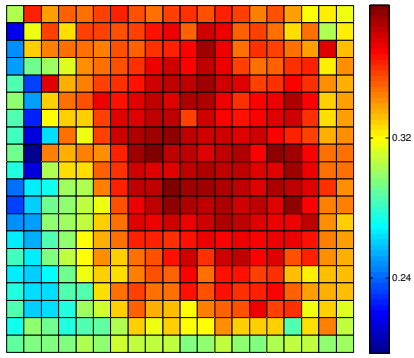
Figure B.16: Bottom leftmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 16×16.



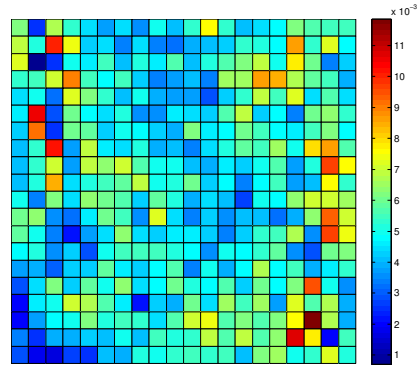
(a) predictions map; provider {28}



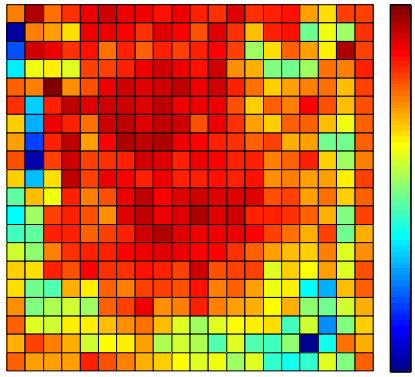
(b) prediction variances map; provider {28}



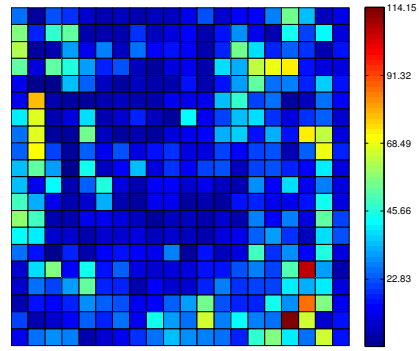
(c) predictions map; provider {63}



(d) prediction variances map; provider {63}

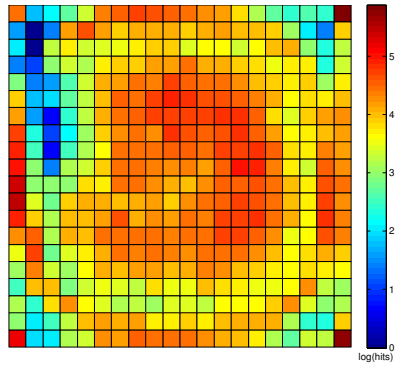


(e) predictions map; provider {83}

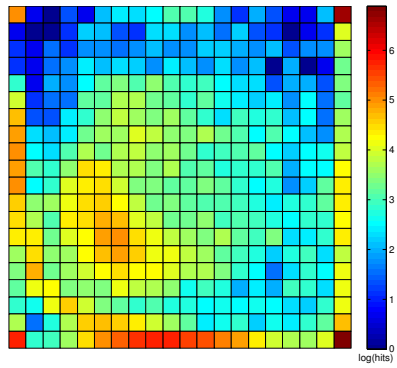


(f) prediction variances map; provider {83}

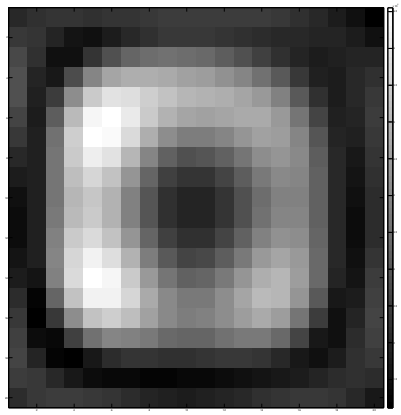
Figure B.17: The trained high-level GTM map for MCYT330-DP and block size 16×16 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

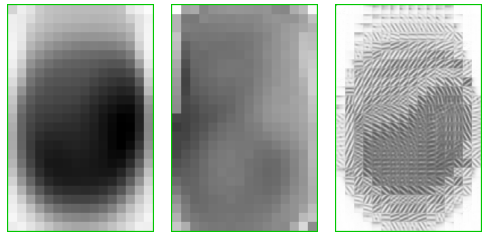


(b) log-hitmap (validation set)



(c) distance matrix (magnification factors)

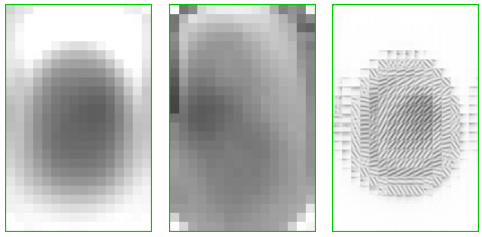
Figure B.18: Log-hitmaps and distance matrix for the trained high-level GTM map for MCYT330-DP with block size 16×16, where the *posteriors' modes* are counted as *hits*.



(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

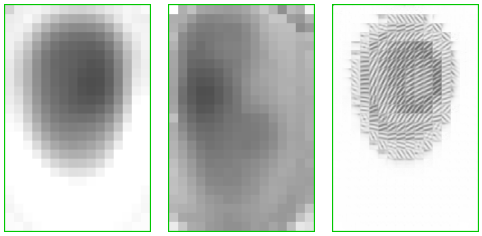
Figure B.19: Top leftmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 16×16.



(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

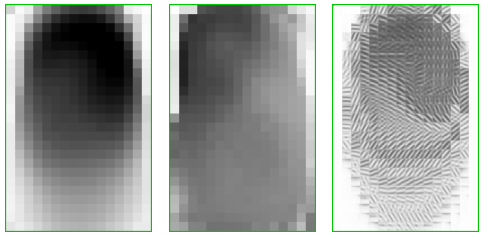
Figure B.20: Top rightmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 16×16.



(a) vertical features **(b)** horizontal features **(c)** codebook blocks

(d) example samples

Figure B.21: Bottom rightmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 16×16.

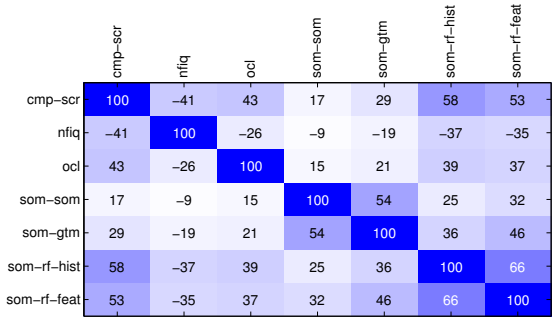


(a) vertical features **(b)** horizontal features **(c)** codebook blocks

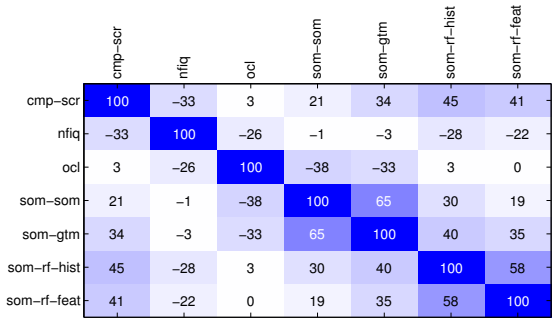
(d) example samples

Figure B.22: Bottom leftmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 16×16.

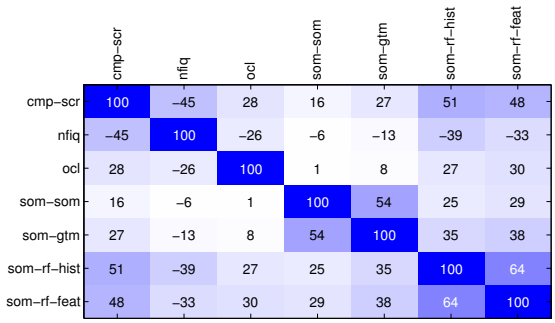
B.1.3 Final results



(a) provider {28}



(b) provider {63}



(c) provider {83}

Figure B.23: Correlation tables for MCYT330-DP with block size 16×16.

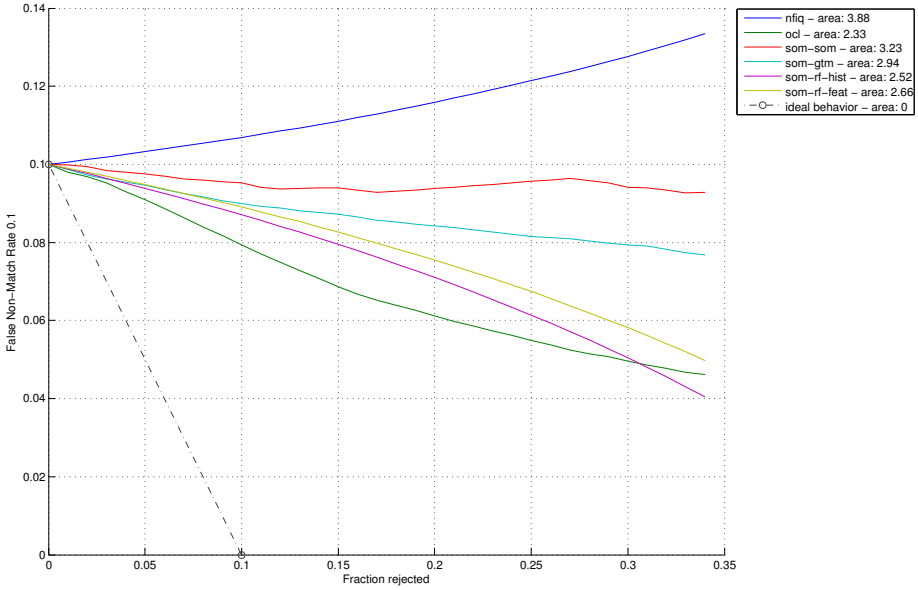


Figure B.24: ERC curves for MCYT330-DP with block size 16×16 for provider {28}.

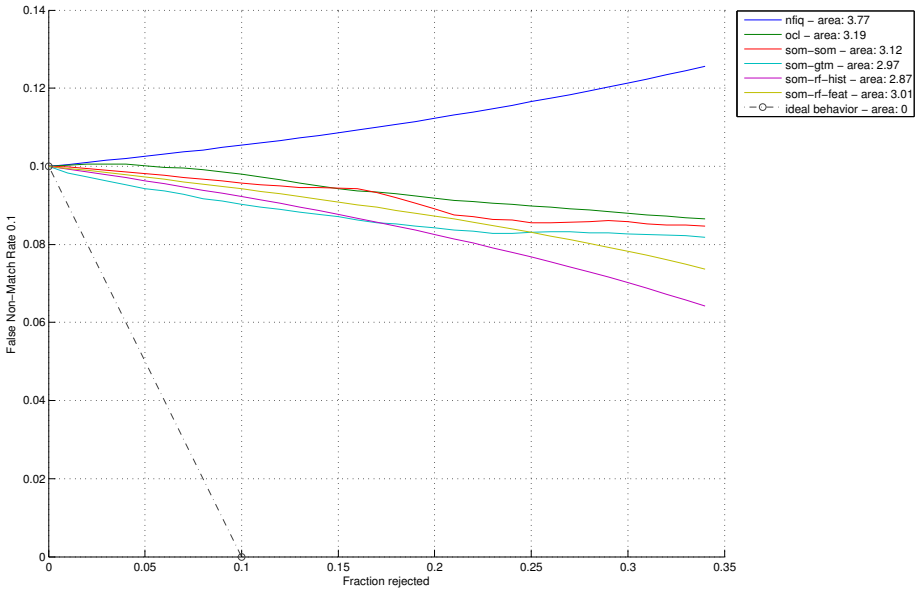


Figure B.25: ERC curves for MCYT330-DP with block size 16×16 for provider {63}.

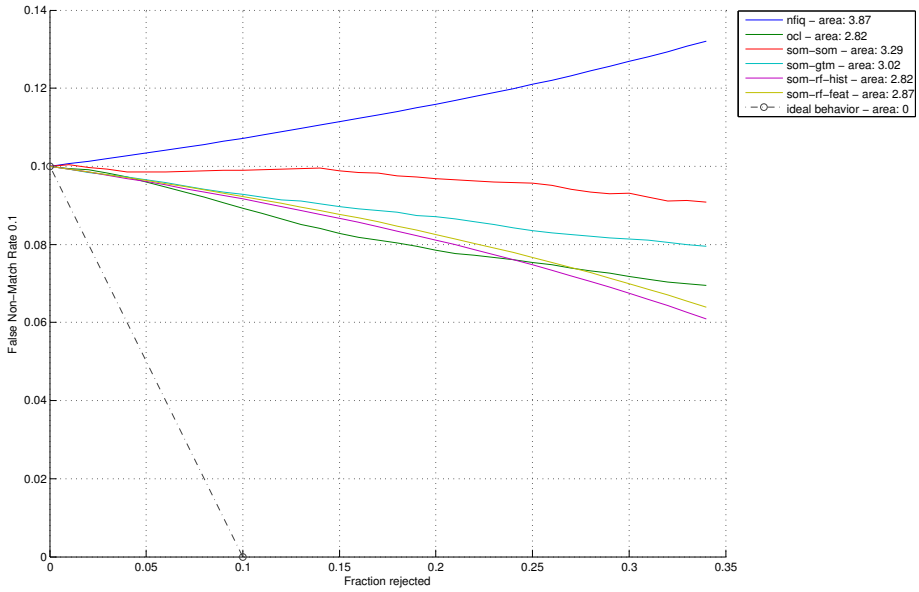
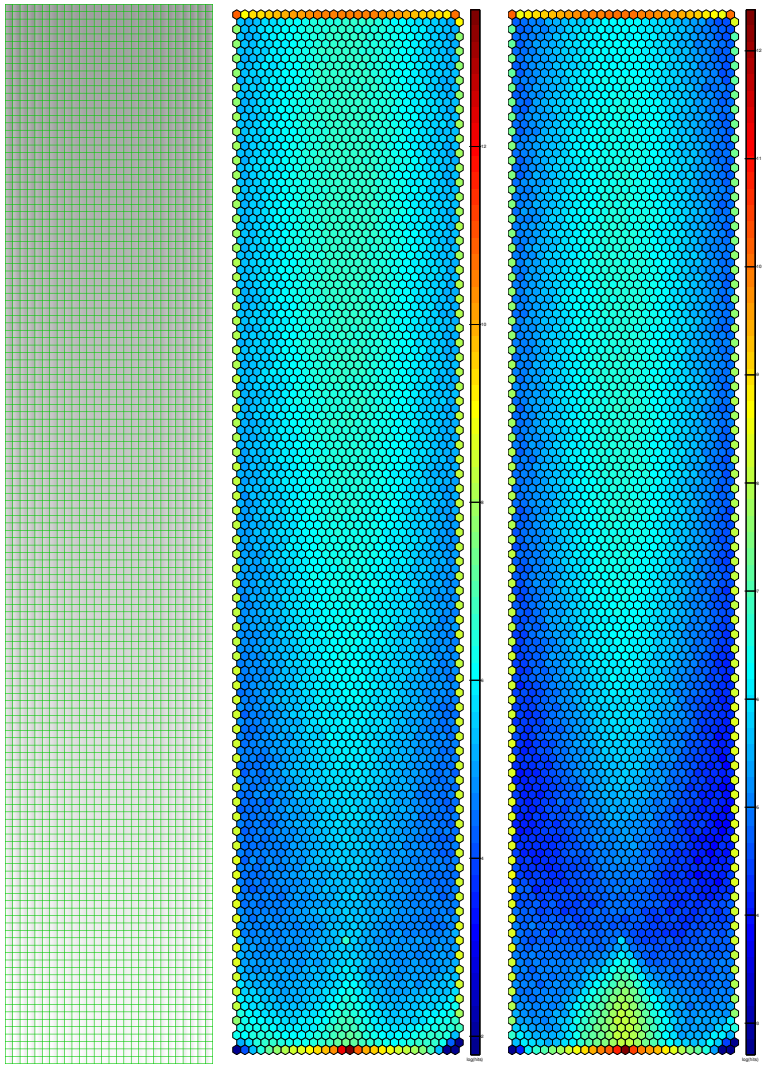


Figure B.26: ERC curves for MCYT330-DP with block size 16×16 for provider {83}.

B.2 Block size 24×24

B.2.1 Low-level

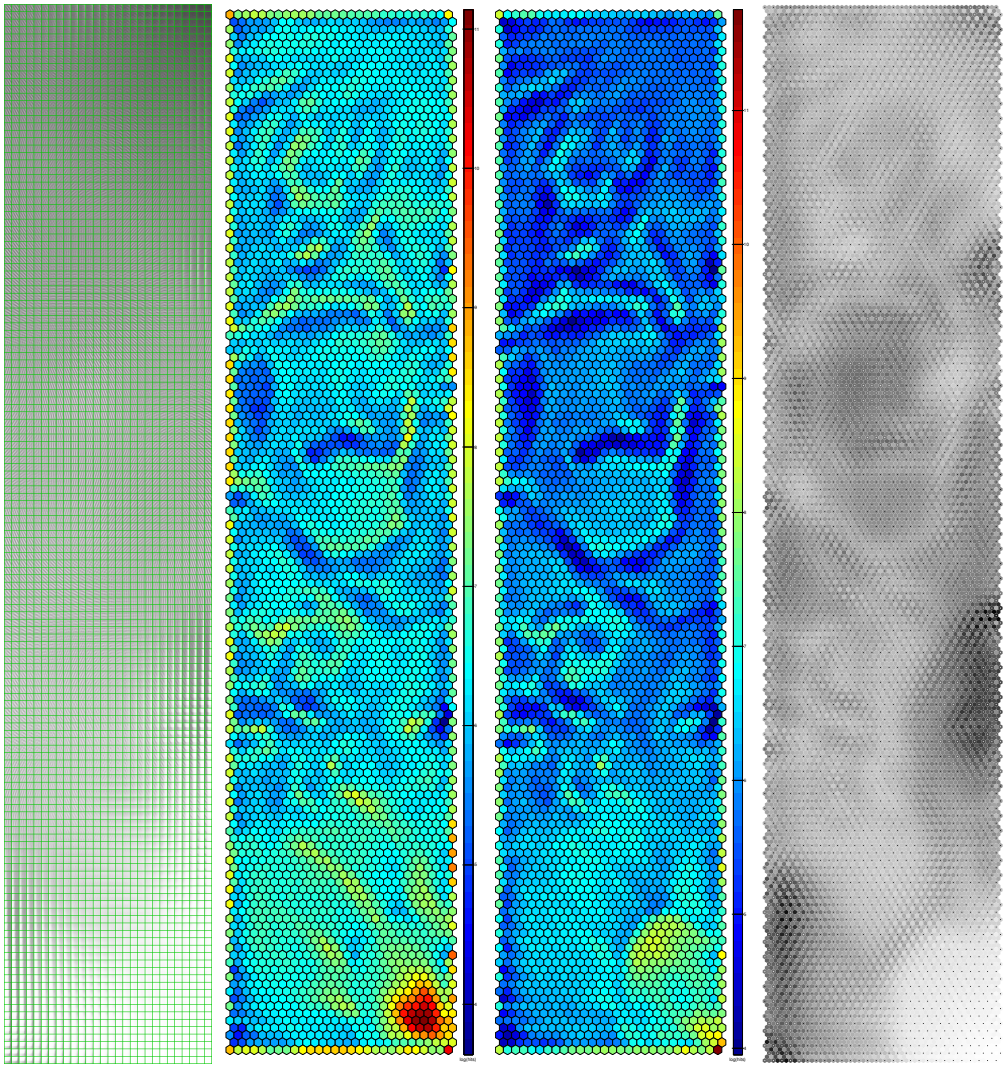


(a) initial SOM map

(b) initial SOM map's log-hitmap (training set)

(c) initial SOM map's log-hitmap (validation set)

Figure B.27: Linearly initialized SOM map for dataset MCYT330-DP and block size 24×24



(a) final SOM map **(b)** final SOM map's log-hitmap (training set) **(c)** final SOM map's log-hitmap (validation set) **(d)** final SOM map's distance matrix (U-matrix)

Figure B.28: Results of the training for dataset MCYT330-DP and block size 24×24

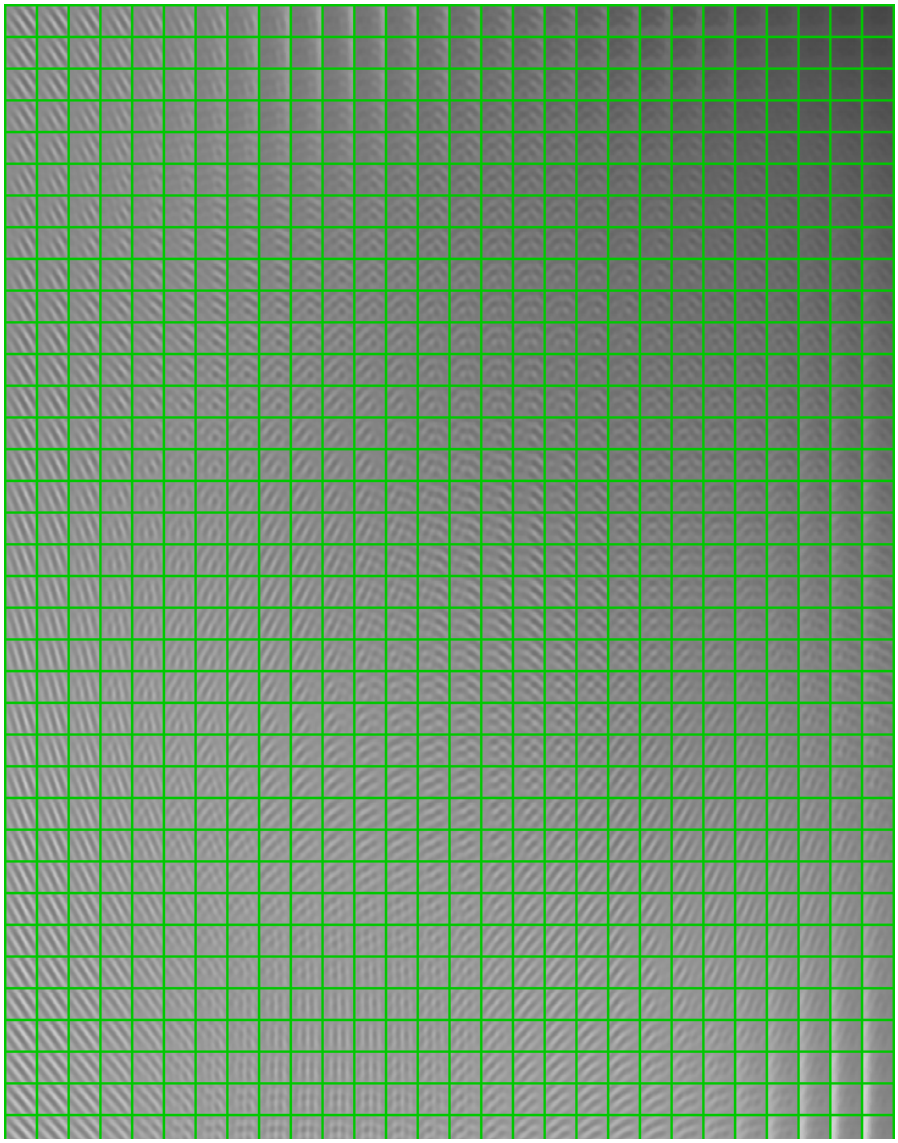


Figure B.29: Final SOM map codebook vectors for dataset MCYT330-DP and block size 24×24, first quarter of the map.

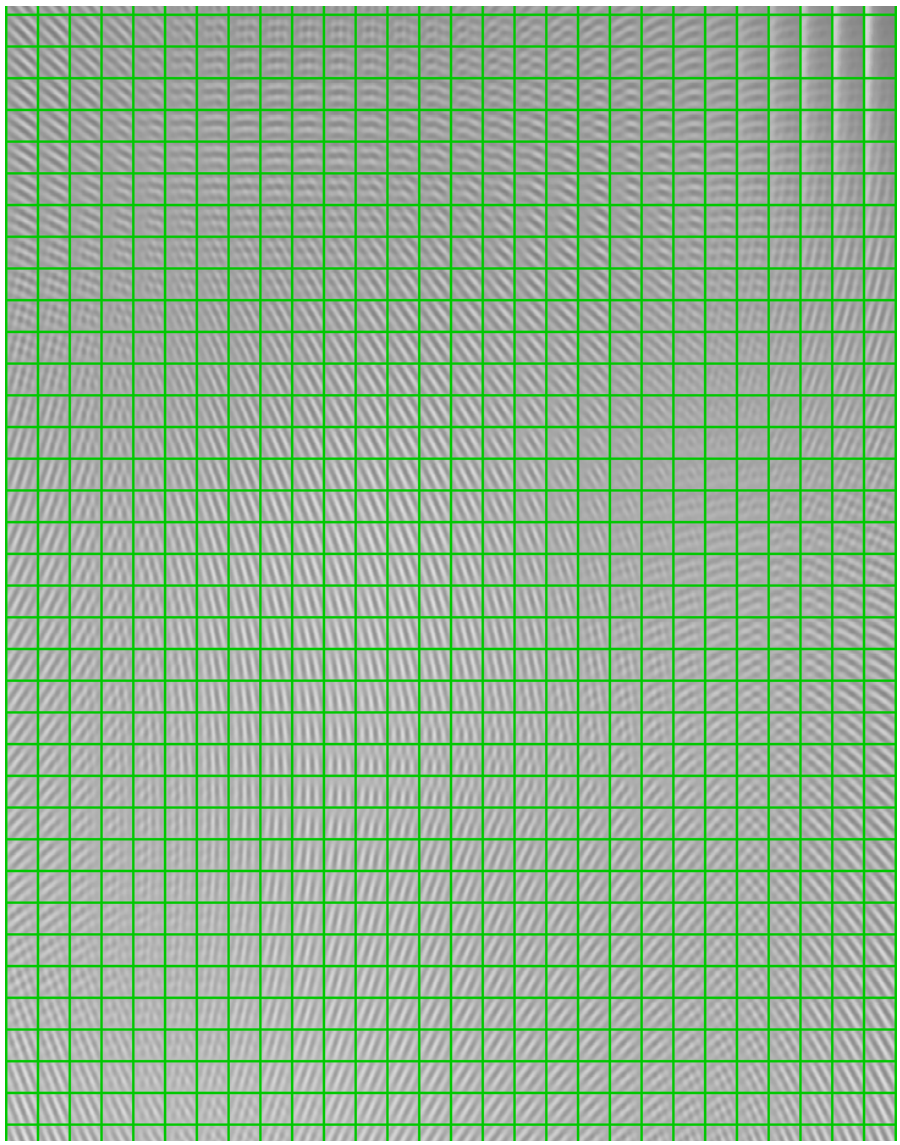


Figure B.30: Final SOM map codebook vectors for dataset MCYT330-DP and block size 24×24, second quarter of the map.

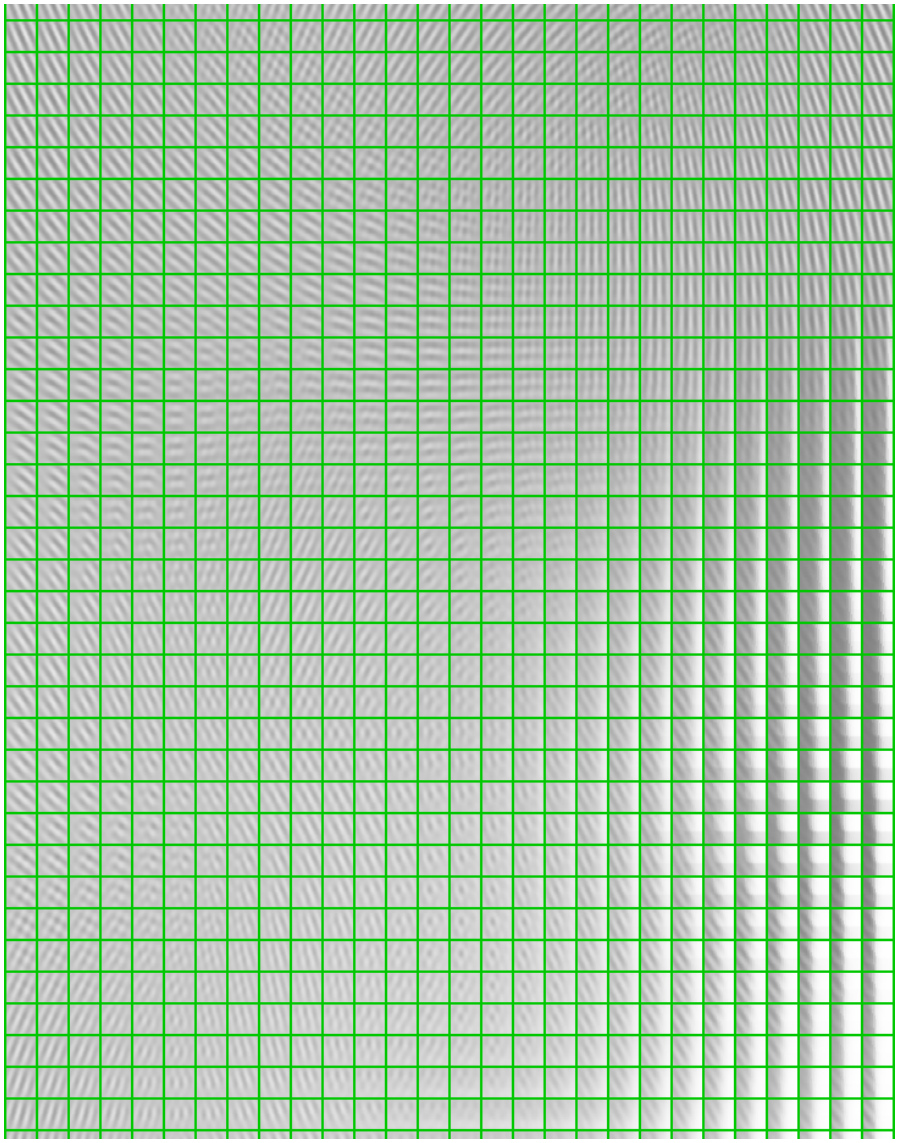


Figure B.31: Final SOM map codebook vectors for dataset MCYT330-DP and block size 24×24, third quarter of the map.

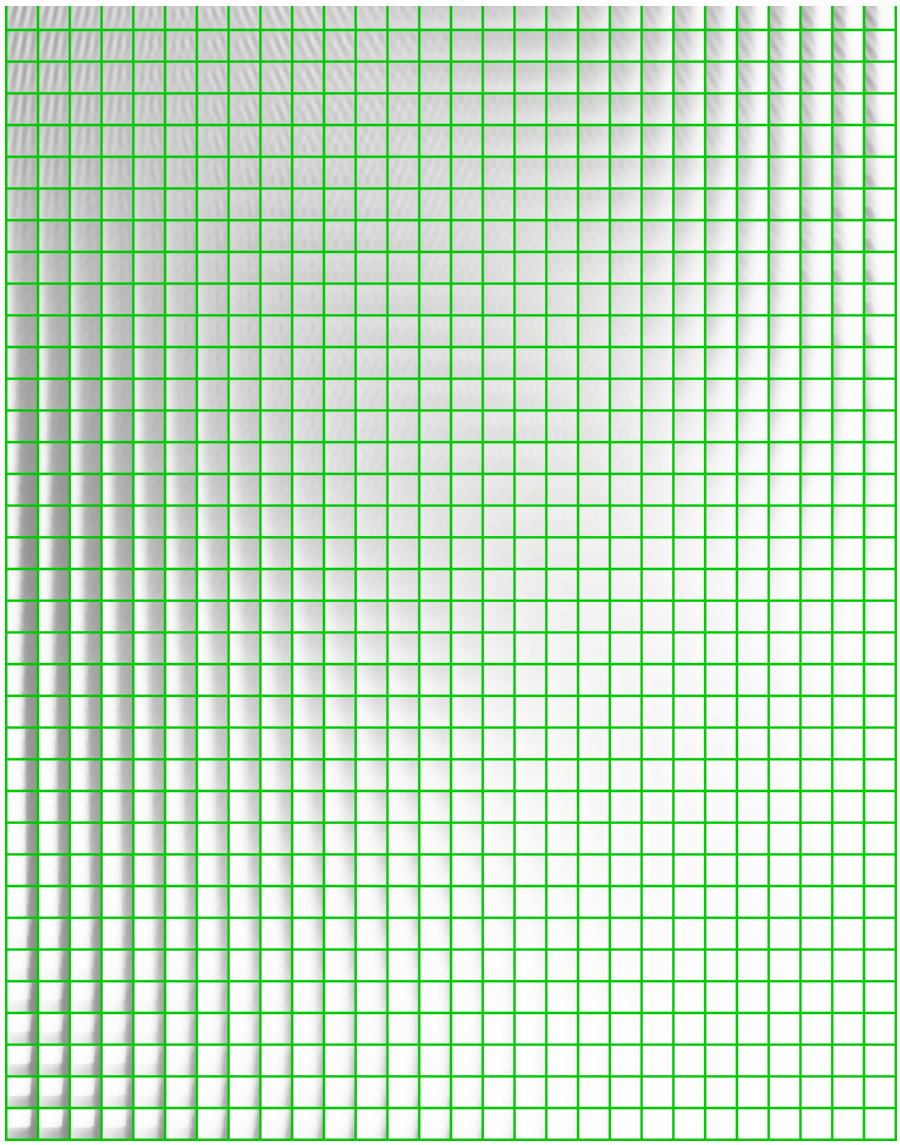


Figure B.32: Final SOM map codebook vectors for dataset MCYT330-DP and block size 24×24, last quarter of the map.

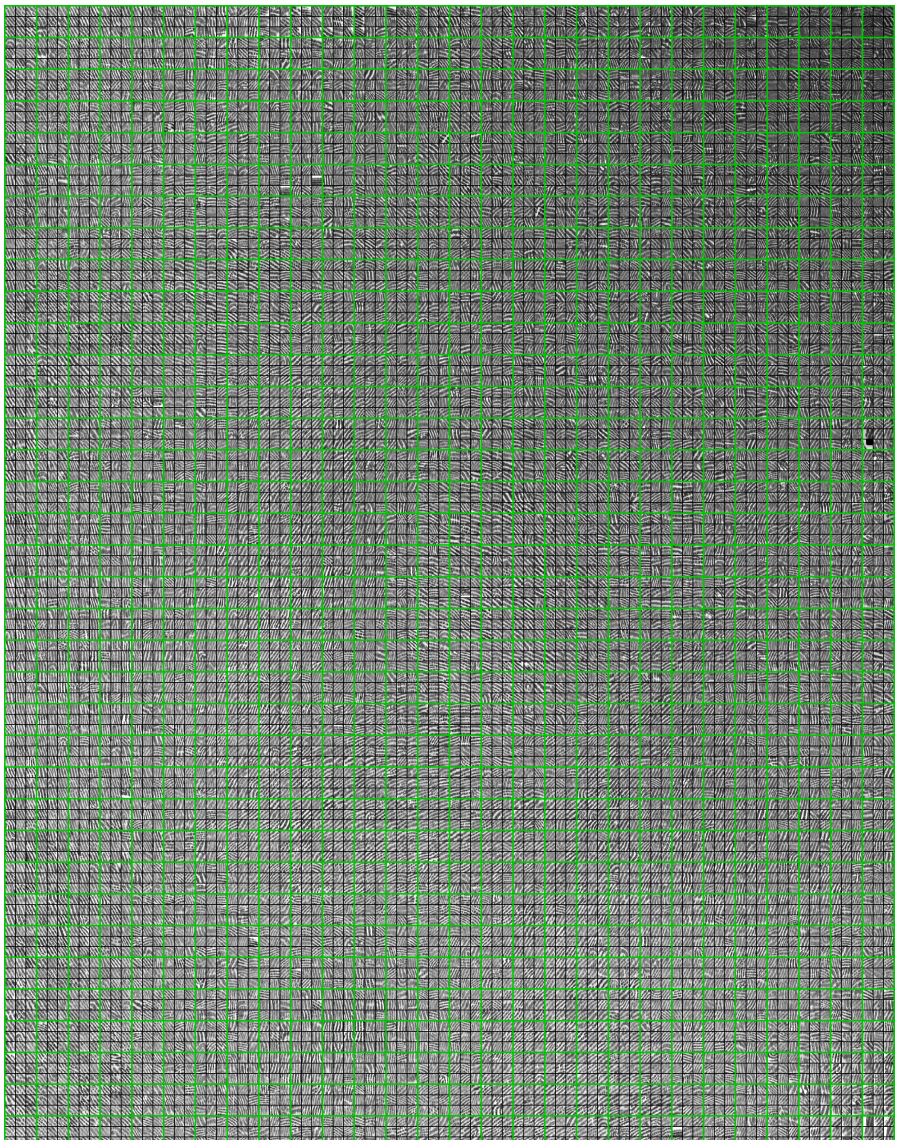


Figure B.33: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 24×24, first quarter of the map.

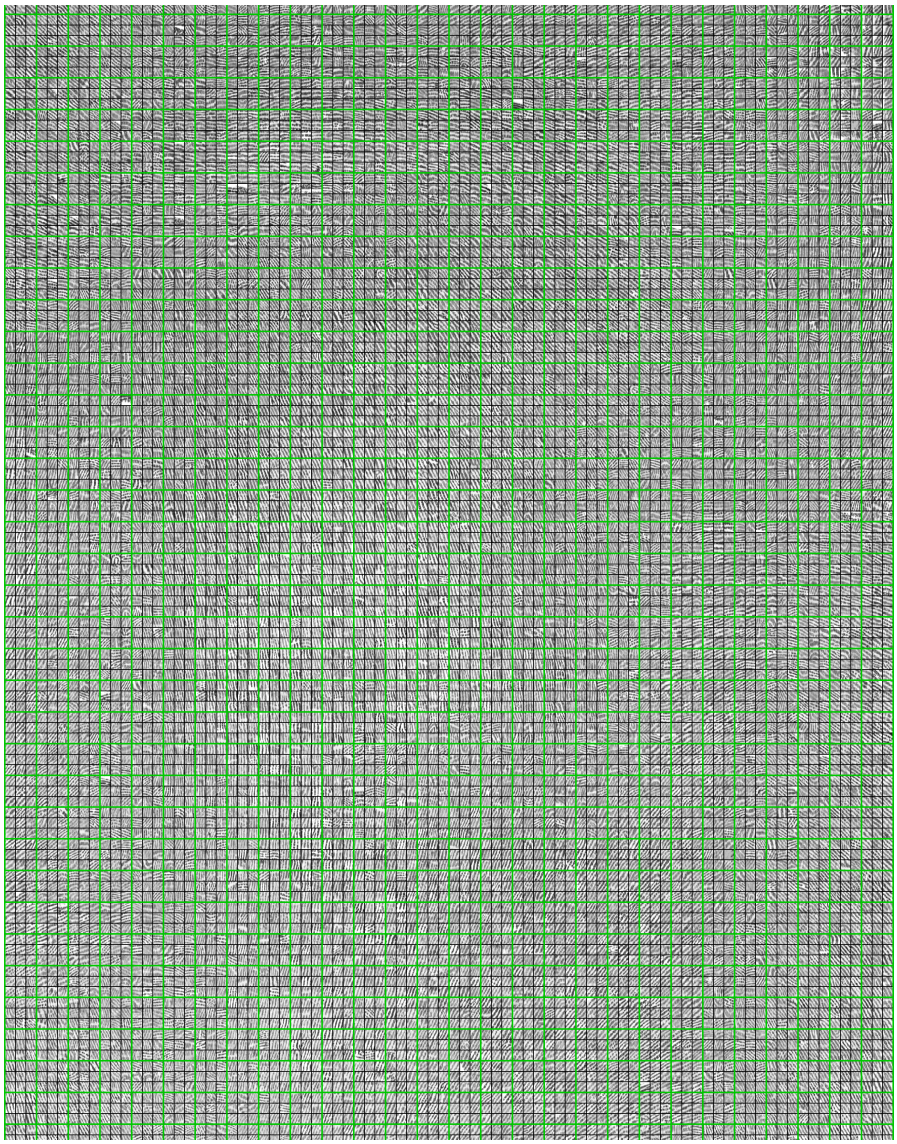


Figure B.34: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 24×24, second quarter of the map.

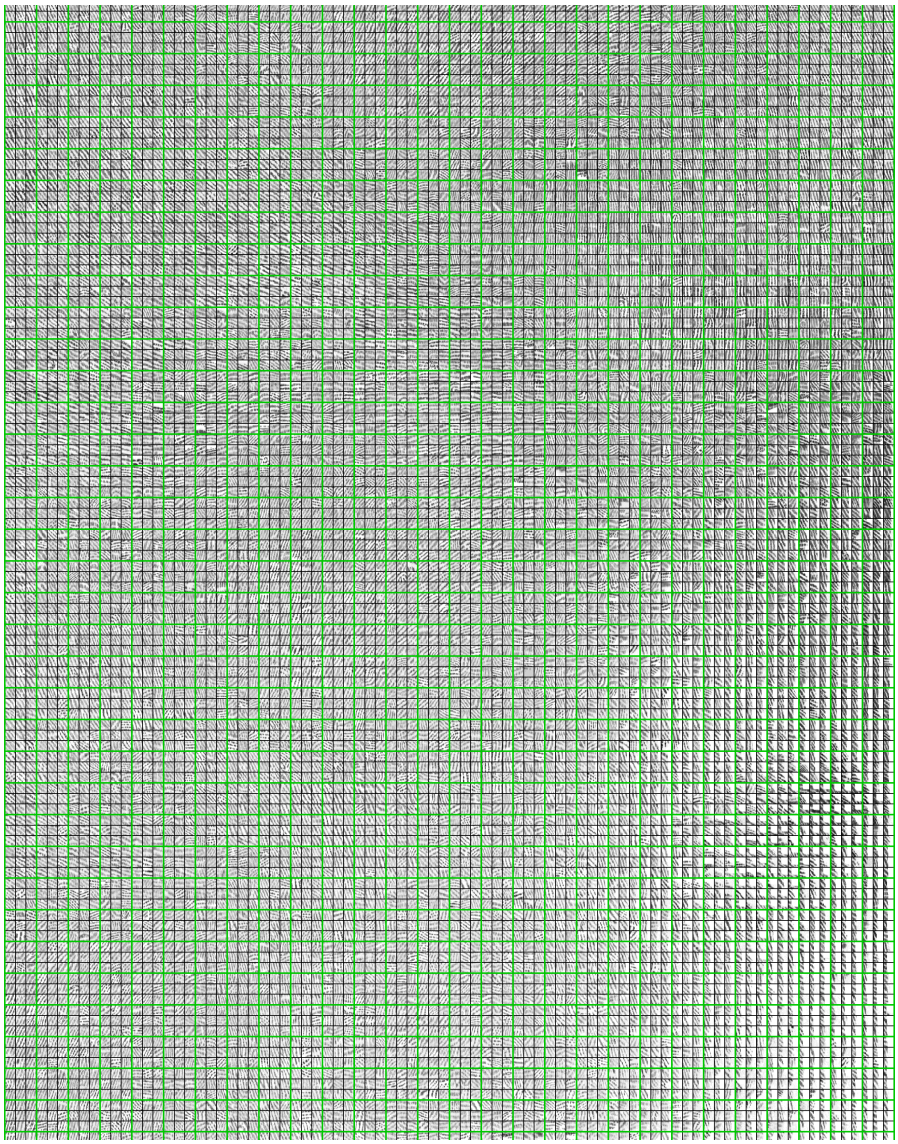


Figure B.35: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 24×24, third quarter of the map.

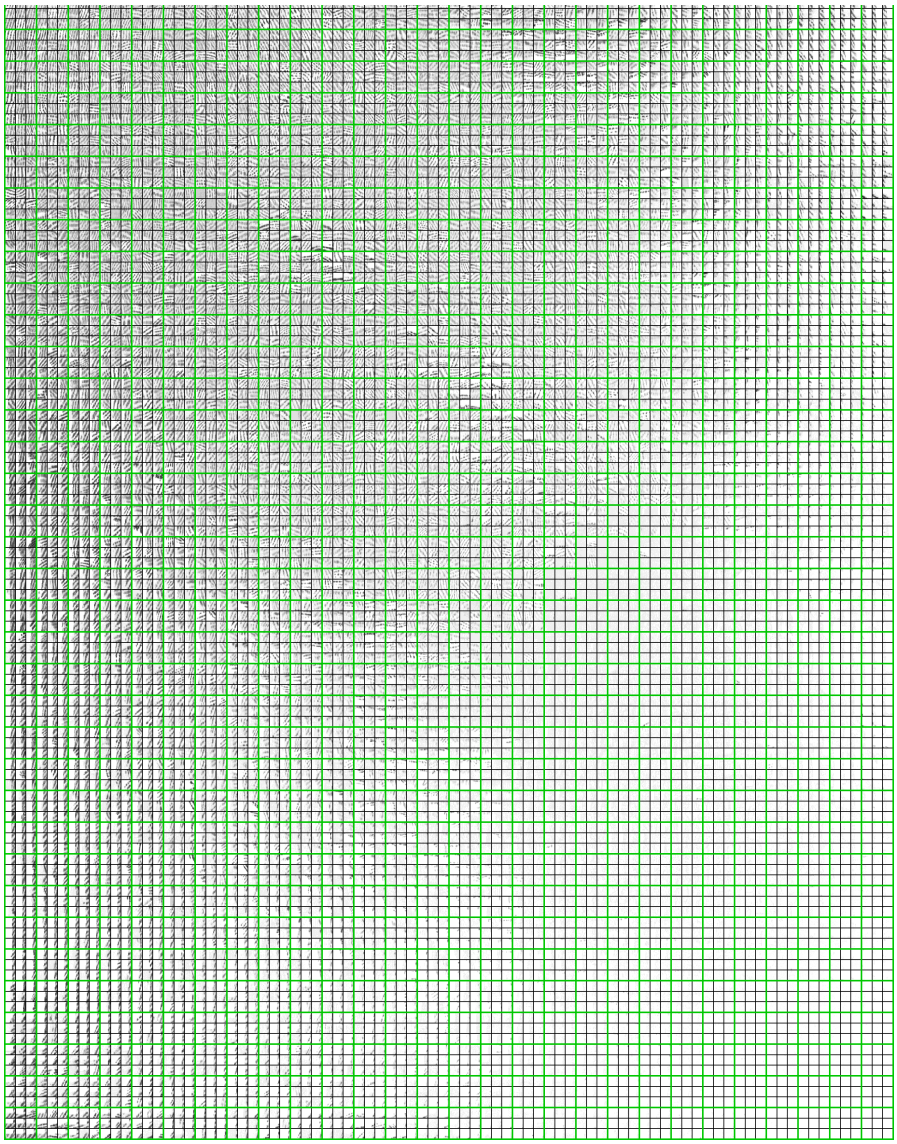


Figure B.36: Subsets of the samples mapped to each node of the final SOM map for dataset MCYT330-DP and block size 24×24, last quarter of the map.

B.2.2 High-level

	1	2	3	4	5
1	2320	1316	811	400	247
2	795	1074	900	677	581
3	213	419	474	482	460
4	59	215	260	281	299
5	107	397	768	857	1428

(a) RF-feat; provider {28}

	1	2	3	4	5
1	2332	1194	713	343	263
2	332	386	233	164	73
3	174	259	223	153	107
4	47	74	88	78	57
5	609	1508	1956	1959	2515

(b) RF-hist; provider {28}

	1	2	3	4	5
1	2767	1946	1337	1049	570
2	989	1326	1186	761	360
3	402	582	635	584	430
4	40	93	134	247	369
5	2	2	6	12	11

(c) RF-feat; provider {63}

	1	2	3	4	5
1	2516	1668	1147	917	542
2	1231	1706	1534	1109	674
3	375	493	489	421	250
4	36	41	46	92	108
5	42	41	82	114	166

(d) RF-hist; provider {63}

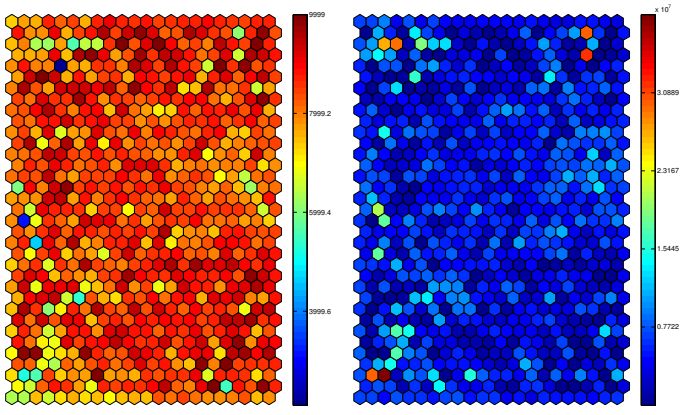
	1	2	3	4	5
1	2233	1516	1116	593	282
2	638	1002	878	551	239
3	413	684	850	749	465
4	193	296	467	529	462
5	133	162	248	460	681

(e) RF-feat; provider {83}

	1	2	3	4	5
1	2532	1997	1544	789	375
2	303	452	451	289	136
3	276	416	501	422	267
4	239	390	568	618	445
5	260	405	495	764	906

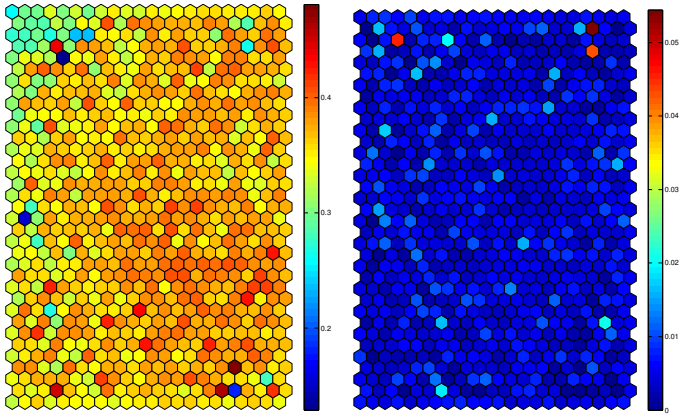
(f) RF-hist; provider {83}

Table B.2: Confusion matrices for RF-feat (left) and RF-hist (right) for different providers and dataset MCYT330-DP with block size 24×24.



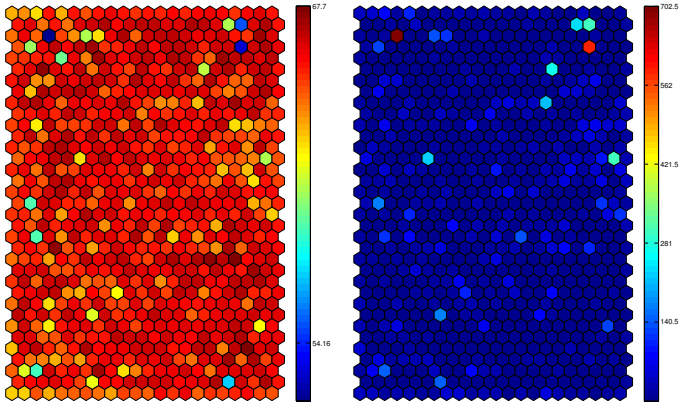
(a) predictions map; provider {28}

(b) prediction variances map; provider {28}



(c) predictions map; provider {63}

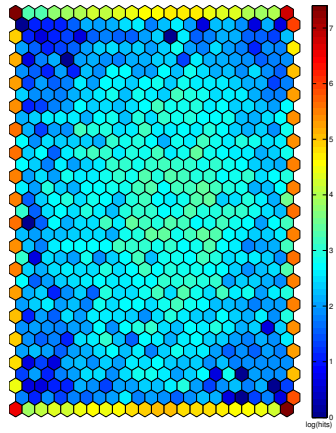
(d) prediction variances map; provider {63}



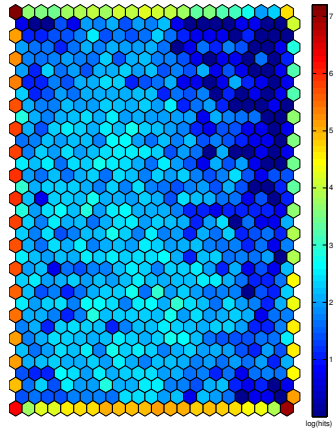
(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

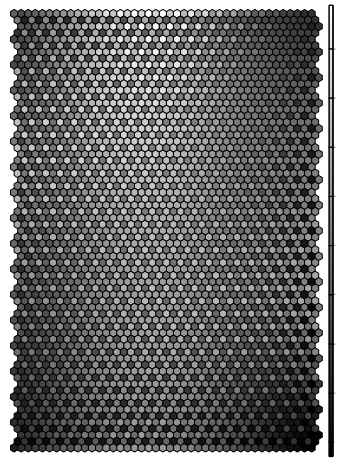
Figure B.37: The trained high-level SOM map for MCYT330-DP and block size 24 × 24 with the assigned prediction and its variance of the average genuine score values for each node. This map is



(a) log-hitmap (training set)

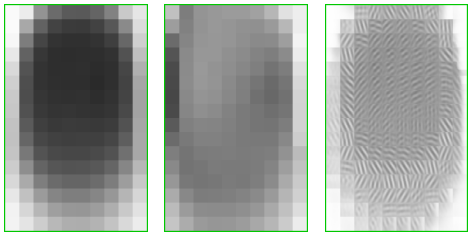


(b) log-hitmap (validation set)



(c) distance matrix (U-matrix)

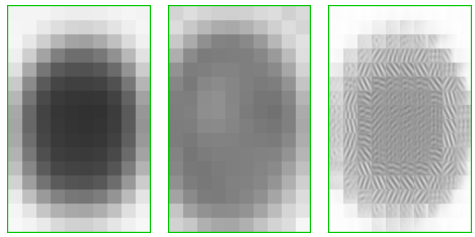
Figure B.38: Log-hitmaps and distance matrix for the trained high-level SOM map for MCYT330-DP with block size 24×24.



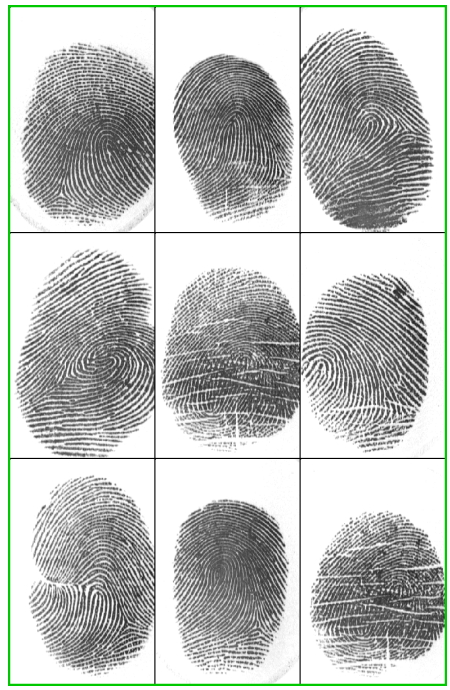
(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

Figure B.39: Top leftmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 24×24.

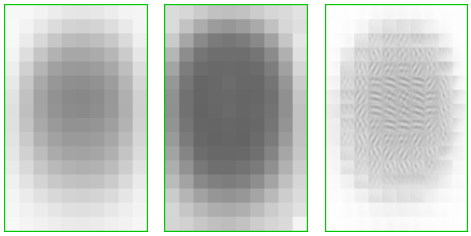


(a) vertical features **(b)** horizontal features **(c)** codebook vectors



(d) example samples

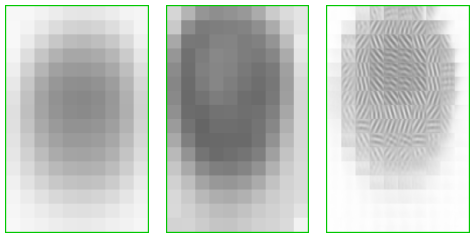
Figure B.40: Top rightmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 24×24.



(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

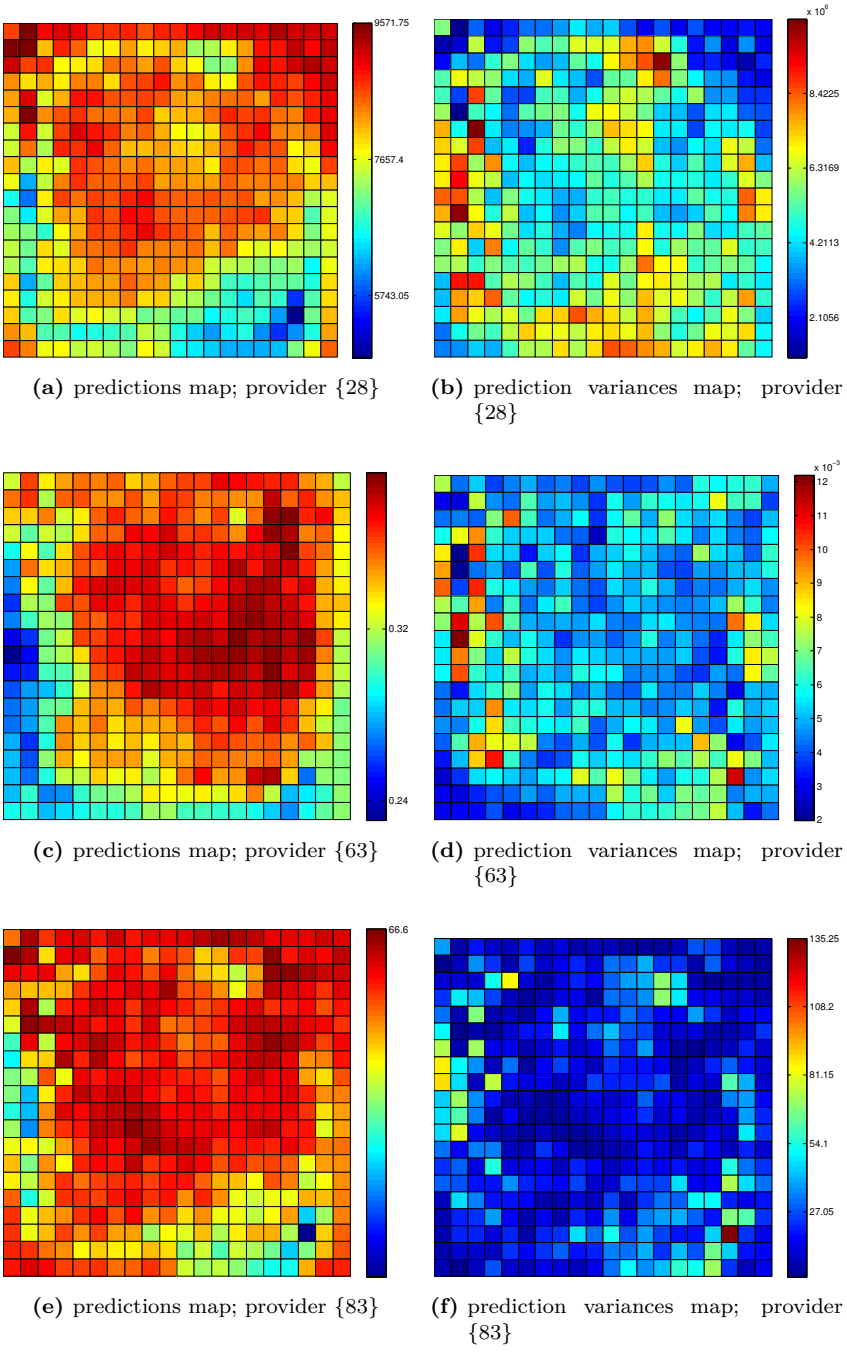
Figure B.41: Bottom rightmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 24×24.



(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

Figure B.42: Bottom leftmost node of the trained high-level SOM map for dataset MCYT330-DP with block size 24×24.



(a) predictions map; provider {28}

(b) prediction variances map; provider {28}

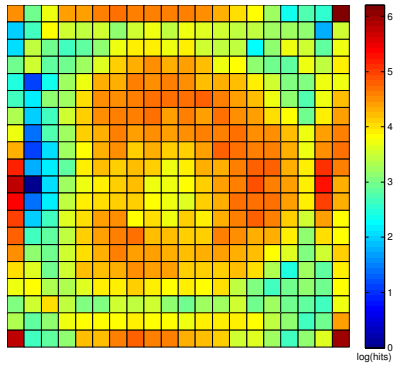
(c) predictions map; provider {63}

(d) prediction variances map; provider {63}

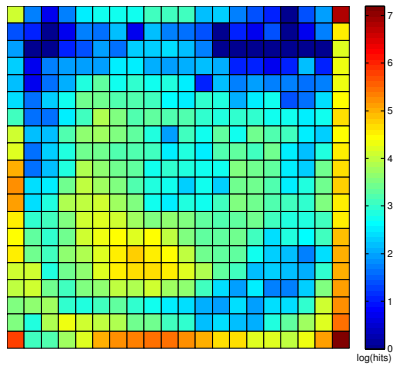
(e) predictions map; provider {83}

(f) prediction variances map; provider {83}

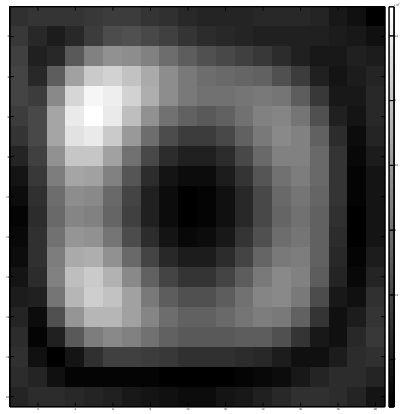
Figure B.43: The trained high-level GTM map for MCYT330-DP and block size 24×24 with the assigned prediction and its variance of the average genuine score values for each node. This map is calibrated for each of the three providers: {28}, {63} and {83}.



(a) log-hitmap (training set)

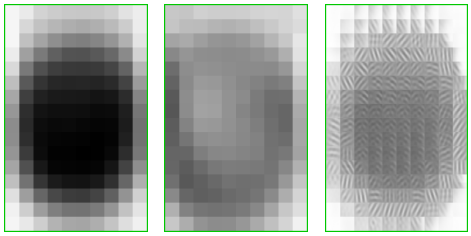


(b) log-hitmap (validation set)

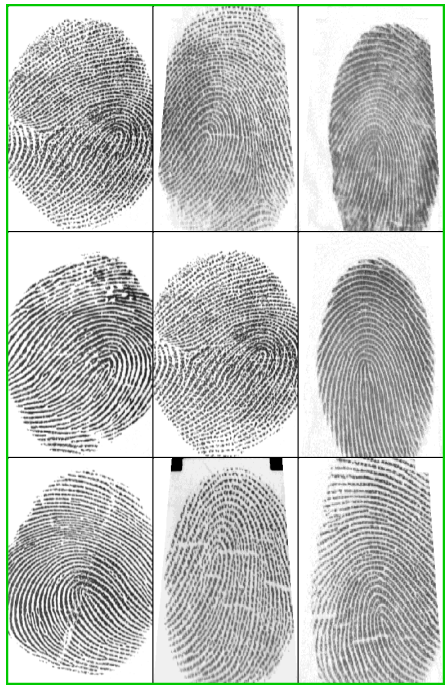


(c) distance matrix (magnification factors)

Figure B.44: Log-hitmaps and distance matrix for the trained high-level GTM map for MCYT330-DP with block size 24×24, where the *posteriors' modes* are counted as *hits*.

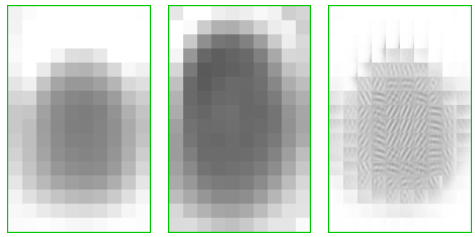


(a) vertical features **(b)** horizontal features **(c)** codebook vectors



(d) example samples

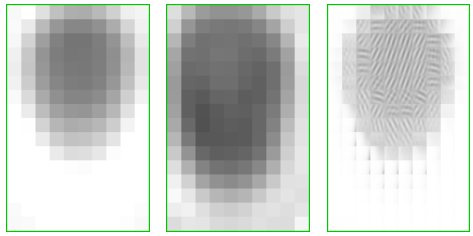
Figure B.45: Top leftmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 24×24.



(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

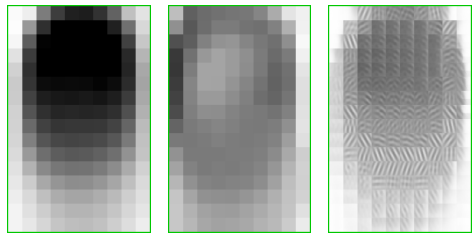
Figure B.46: Top rightmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 24×24.



(a) vertical features **(b)** horizontal features **(c)** codebook vectors

(d) example samples

Figure B.47: Bottom rightmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 24×24.

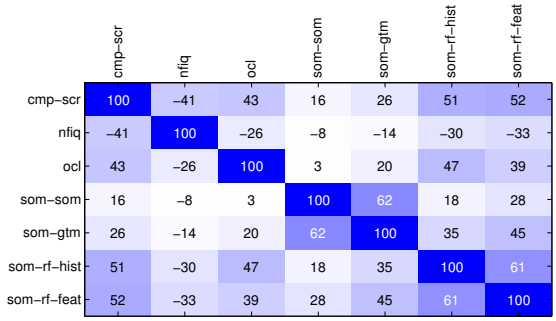


(a) vertical features **(b)** horizontal features **(c)** codebook vectors

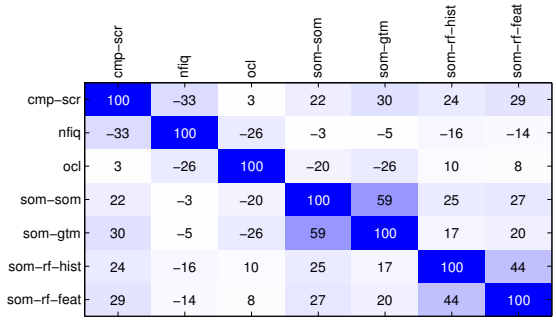
(d) example samples

Figure B.48: Bottom leftmost node of the trained high-level GTM map for dataset MCYT330-DP with block size 24×24.

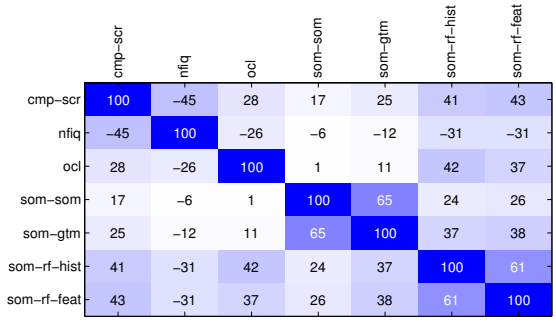
B.2.3 Final results



(a) provider {28}



(b) provider {63}



(c) provider {83}

Figure B.49: Correlation tables for MCYT330-DP with block size 24×24.

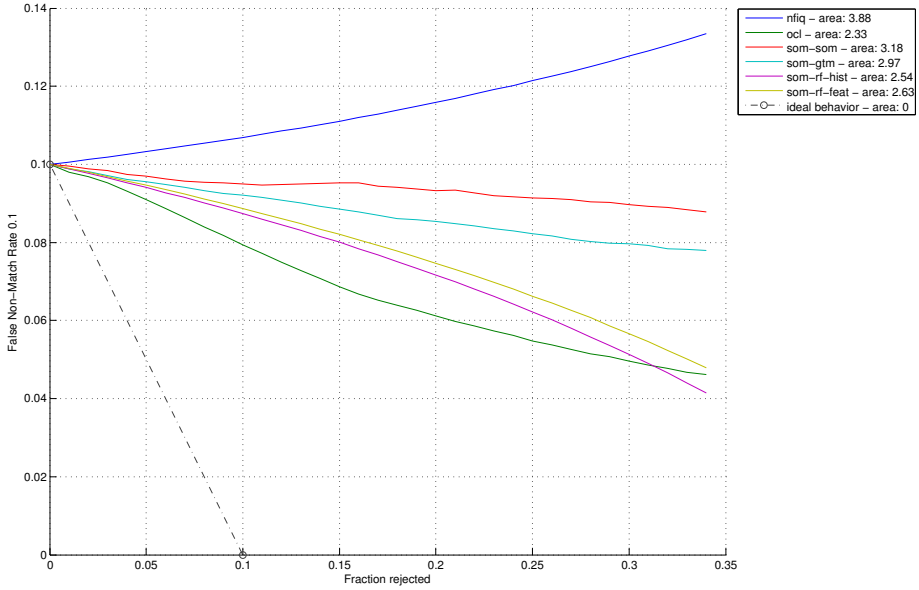


Figure B.50: ERC curves for MCYT330-DP with block size 24×24 for provider {28}.

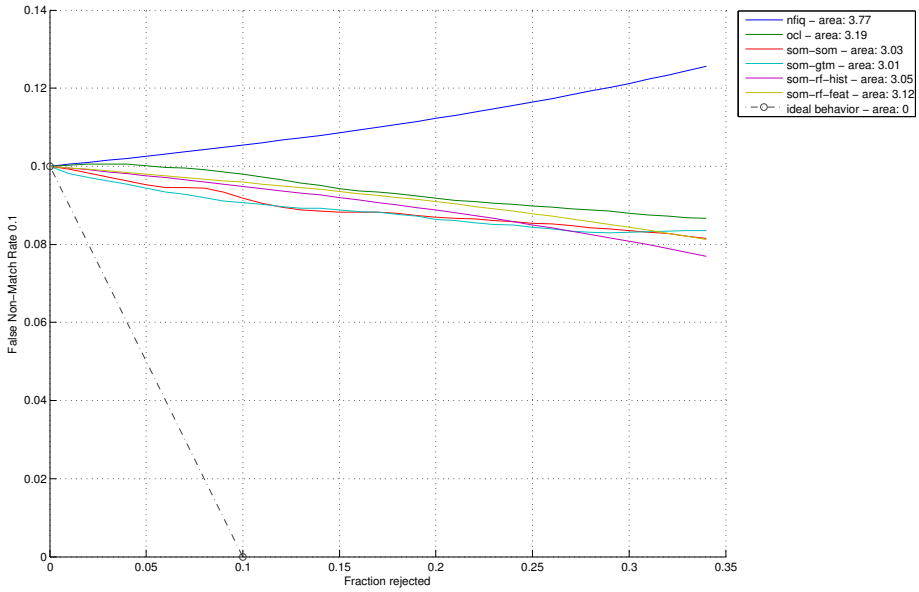


Figure B.51: ERC curves for MCYT330-DP with block size 24×24 for provider {63}.

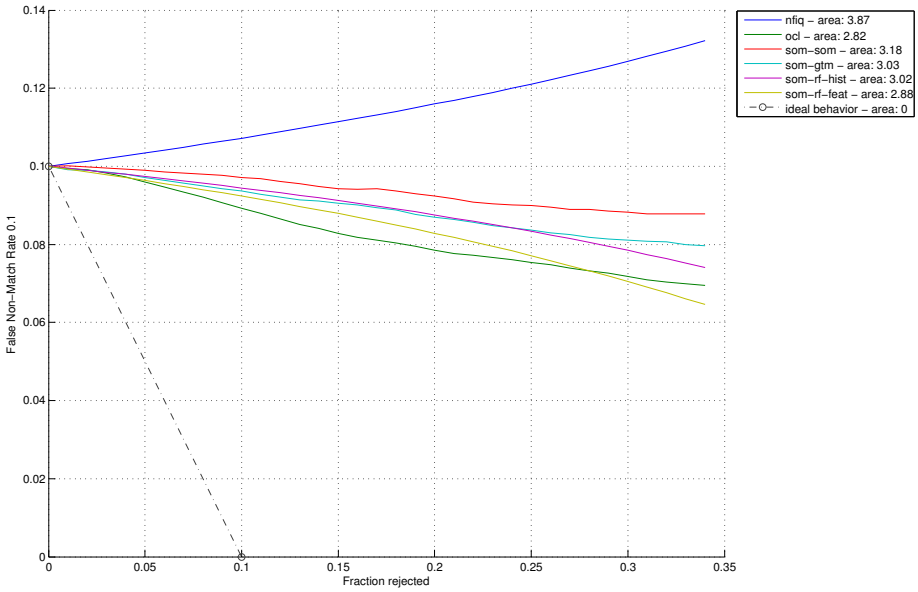


Figure B.52: ERC curves for MCYT330-DP with block size 24×24 for provider {83}.

Bibliography

[Bis07] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing 2011 edition, October 2007.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Com] European Commission. Visa information system. http://ec.europa.eu/dgs/home-affairs/what-we-do/policies/borders-and-visas/visa-information-system/index_en.htm. [Online; accessed 25-August-2013].

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[Dus13] M. Dusio. Fingerprint quality assurance using image processing, 2013. DTU supervisor: Rasmus Larsen, rlar@dtu.dk, DTU Compute.

[FKB08] H. Fronthaler, K. Kollreider, and J. Bigun. Local features for enhancement and minutiae extraction in fingerprints. *Image Processing, IEEE Transactions on*, 17(3):354–363, 2008.

[FS89] I. Fogel and D. Sagi. Gabor filters as texture discriminator. *Biological Cybernetics*, 61(2):103–113, jun 1989.

[GT07] P. Grother and E. Tabassi. Performance of biometric quality measures. *IEEE Trans. Pattern Anal. Mach. Intell*, pages 531–543, 2007.

[ISO07] ISO. ISO/IEC 19795-2:2007 information technology – biometric performance testing and reporting – part 2: Testing methodologies for technology and scenario evaluation. 2007.

[ISO11] ISO. ISO/IEC 19794-1:2011 information technology – biometric data interchange formats – part 1: Framework. July 2011.

[ISO12a] ISO. ISO/IEC 2382-37:2012 information technology – vocabulary – part 37: Biometrics. December 2012.

[ISO12b] ISO. ISO/IEC 29794-1:2009 information technology – biometric sample quality – part 1: Framework. September 2012.

[ISO12c] ISO. ISO/IEC TR 29794-4:2010 information technology – biometric sample quality – part 4: Finger image data. September 2012.

[KHKL96] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. SOM PAK: The Self-Organizing map program package, 1996.

[Koh82] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, jan 1982.

[Koh13] Teuvo Kohonen. Essentials of the self-organizing map. *Neural Networks*, 37:52–65, jan 2013.

[KSH01] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[LGRN09] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[LJY02] E. Lim, X. Jiang, and W. Yau. Fingerprint quality and validity analysis. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–469. IEEE, 2002.

[Mak12] A. Makarov. Fingerprint quality estimation using self-organizing maps. Master's thesis, Technical University of Denmark, DTU Informatics, E-mail: reception@imm.dtu.dk, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark, 2012. Supervised by Professor Rasmus Larsen, rl@imm.dtu.dk, DTU Informatics.

[MAT12] MATLAB. *version 7.14.0 (R2012a)*. The MathWorks Inc., Natick, Massachusetts, 2012.

[Min89] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243, 1989.

[MMJP09] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer Professional Computing. Springer, 2009.

[Nab02] I. T. Nabney. *NETLAB: Algorithms for Pattern Recognition*. Advances in Pattern Recognition. Springer, 1st edition, 2002.

[NIS] NIST. Nist biometric image software. http://www.nist.gov/itl/iad/ig/nbis.cfm. [Online; accessed 15-September-2013].

[NIS12a] NIST. Development of nfiq 2.0 - evaluation of potential image features for quality assessment. June 2012.

[NIS12b] NIST. Development of nfiq 2.0 - quality feature definitions. June 2012.

[OGFAS+03] J. Ortega-Garcia, J. Fierrez-Aguilar, D. Simon, J. Gonzalez, M.s Faundez-Zanuy, V. Espinosa, A. Satue, I. Hernaez, J-J. Igarza, C. Vivaracho, D. Escudero, and Q.-I. Moro. Mcyt baseline corpus: a bimodal biometric database. *IEEE Proceedings-Vision, Image and Signal Processing*, 150(6):395–401, 2003.

[oHS] U.S.A. Department of Homeland Security. Office of biometric identity management. http://www.dhs.gov/obim. [Online; accessed 25-August-2013].

[oI] Unique Identification Authority of India. Aadhaar. http://uidai.gov.in/. [Online; accessed 25-August-2013].

[oSIoA] Chinese Academy of Sciences' Institute of Automation. Casia fingerprint image database version 5.0. http://biometrics.idealtest.org/. [Online; accessed 25-August-2013].

[OTMB13] M. Olsen, E. Tabassi, A. Makarov, and C. Busch. Self-organizing maps for fingerprint image quality assessment. In *Proceedings of the 26th Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2013)*, Portland, OR, USA, June 2013. IEEE Computer Society.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[R C13] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[Row98] Sam Roweis. Em algorithms for pca and spca. In *in Advances in Neural Information Processing Systems*, pages 626–632. MIT Press, 1998.

[SB01] Leo Breiman Statistics and Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[SKK01] L. Shen, A. Kot, and W. Koo. Quality measures of fingerprint images. In *Audio-and Video-based Biometric Person Authentication*, pages 266–271. Springer, 2001.

[SMN+13] Roberto Di Salvo, Placido Montalto, Giuseppe Nunnari, Marco Neri, and Giuseppe Puglisi. Multivariate time series clustering on geophysical data recorded at mt. etna from 1996 to 2003. *Journal of Volcanology and Geothermal Research*, 251(0):65 – 74, 2013. <ce:title>Flank instability at Mt. Etna</ce:title>.

[Spe87] C. Spearman. The proof and measurement of association between two things. by c. spearman, 1904. *The American journal of psychology*, 100(3-4):441–471, 1987.

[Sve98] Johan F.M Svensen. *GTM: the generative topographic mapping*. PhD thesis, Aston University, Aston Triangle, Birmingham B4 7ET, United Kingdom, 1998.

[TG09] E. Tabassi and P. Grother. Fingerprint image quality. In *Encyclopedia of Biometrics*, pages 482–490. 2009.

[TW05] E. Tabassi and C. L. Wilson. A novel approach to fingerprint image quality. In *ICIP (2)*, pages 37–40, 2005.

[TWW04] Elham Tabassi, Charles L. Wilson, and Craig I. Watson. Fingerprint image quality: Nistir 7151," tech. rep, 2004.

[VHAP00] Juha Vesanto, Johan Himberg, Esa Alhoniemi, and Juha Parhankangas. Self-organizing map in matlab: the som toolbox. In *In Proceedings of the Matlab DSP Conference*, pages 35–40, 2000.