



RODRIGO FRASSETTO NOGUEIRA

“Software Based Fingerprint Liveness Detection”

*“Detecção de Vivacidade de Impressões Digitais baseada em  
Software”*

CAMPINAS  
2014





UNIVERSIDADE ESTADUAL DE CAMPINAS  
Faculdade de Engenharia Elétrica e Computação

RODRIGO FRASSETTO NOGUEIRA

“Software Based Fingerprint Liveness Detection”

*“Detecção de Vivacidade de Impressões Digitais baseada em software”*

Dissertation presented to the School of Electrical and Computer Eng. of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Electrical Engineering in the field of Computer Engineering

*Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica na área de Engenharia da Computação*

Supervisor/*Orientador*: Roberto de Alencar Lotufo

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL  
DA DISSERTAÇÃO DEFENDIDA PELO ALUNO  
RODRIGO FRASSETTO NOGUEIRA, E ORIENTADA  
PELO PROF. DR. ROBERTO DE ALENCAR LOTUFO

---

CAMPINAS  
2011

Ficha catalográfica  
Universidade Estadual de Campinas  
Biblioteca da Área de Engenharia e Arquitetura  
Rose Meire da Silva - CRB 8/5974

N689s Nogueira, Rodrigo Frassetto, 1986-  
Software based fingerprint liveness detection / Rodrigo Frassetto Nogueira. –  
Campinas, SP : [s.n.], 2014.

Orientador: Roberto de Alencar Lotufo.  
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de  
Engenharia Elétrica e de Computação.

1. Datiloscopia. 2. Aprendizado de máquina. I. Lotufo, Roberto de  
Alencar, 1955-. II. Universidade Estadual de Campinas. Faculdade de Engenharia  
Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

**Título em outro idioma:** Detecção de vivacidade de impressões digitais baseada em software

**Palavras-chave em inglês:**

Dactylography

Machine learning

**Área de concentração:** Energia Elétrica

**Titulação:** Mestre em Engenharia Elétrica

**Banca examinadora:**

Roberto de Alencar Lotufo [Orientador]

Anderson de Rezende Rocha

Fernando Von Zuben

**Data de defesa:** 31-07-2014

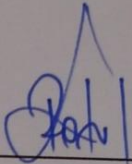
**Programa de Pós-Graduação:** Engenharia Elétrica

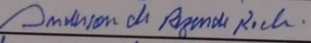
## COMISSÃO JULGADORA - TESE DE MESTRADO

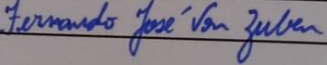
**Candidato:** Rodrigo Frassetto Nogueira

**Data da Defesa:** 31 de julho de 2014

**Título da Tese:** "Detecção de Vivacidade de Impressões Digitais Baseada em Software"

Prof. Dr. Roberto de Alencar Lotufo (Presidente): 

Prof. Dr. Anderson de Rezende Rocha: 

Prof. Dr. Fernando José Von Zuben: 



## ABSTRACT

*With the growing use of biometric authentication systems in the past years, spoof fingerprint detection has become increasingly important. In this work, we implemented and compared various techniques for software-based fingerprint liveness detection. We use as feature extractors Convolutional Networks with random weights, which are applied for the first time for this task, and Local Binary Patterns. The techniques were used in conjunction with dimensionality reduction through Principal Component Analysis (PCA) and a Support Vector Machine (SVM) classifier. Dataset Augmentation was successfully used to increase classifier's performance. We tested a variety of preprocessing operations such as frequency filtering, contrast equalization, and region of interest filtering. An automatic and extensive search for the best combination of preprocessing operations, architectures and hyper-parameters was made, thanks to the fast computers available as cloud services. The experiments were made on the datasets used in The Liveness Detection Competition of years 2009, 2011 and 2013 that comprise almost 50,000 real and fake fingerprints' images. Our best method achieves an overall rate of 95.2% of correctly classified samples - an improvement of 59% in test error when compared with the best previously published results.*

## RESUMO

Com o uso crescente de sistemas de autenticação por biometria nos últimos anos, a detecção de impressões digitais falsas tem se tornado cada vez mais importante. Neste trabalho, nós implementamos e comparamos várias técnicas baseadas em software para detecção de vivacidade de impressões digitais. Utilizamos como extratores de características as redes convolucionais, que foram usadas pela primeira vez nesta área, e *Local Binary Patterns* (LBP). As técnicas foram usadas em conjunto com redução de dimensionalidade através da Análise de Componentes Principais (PCA) e um classificador *Support Vector Machine* (SVM). O aumento artificial de dados foi usado de forma bem sucedida para melhorar o desempenho do classificador. Testamos uma variedade de operações de pré-processamento, tais como filtragem em frequência, equalização de contraste e filtragem da região de interesse. Graças aos computadores de alto desempenho disponíveis como serviços em nuvem, foi possível realizar uma busca extensa e automática para encontrar a melhor combinação de operações de pré-processamento, arquiteturas e hiper-parâmetros. Os experimentos foram realizados nos conjuntos de dados usados nas competições *Liveness Detection* nos anos de 2009, 2011 e 2013, que juntos somam quase 50.000 imagens de impressões digitais falsas e verdadeiras. Nosso melhor método atinge uma taxa média de amostras classificadas corretamente de 95,2%, o que representa uma melhora de 59% na taxa de erro quando comparado com os melhores resultados publicados anteriormente.





# Contents

<b>1. Introduction</b> .....	1
1.1. Related Work.....	1
1.2. Proposed Method.....	4
<b>2. Methodology</b> .....	5
2.1. Basic Concepts .....	5
2.2. Datasets .....	9
2.3. Processing Flow .....	13
2.4. Preprocessing.....	13
2.5. Feature Extraction .....	17
2.5.1. Convolutional Networks .....	18
2.5.2. Local Binary Pattern .....	22
2.6. Feature Normalization, Dimensionality Reduction and Whitening .....	25
2.7. Classifiers .....	26
2.8. Increasing Generalization through Dataset Augmentation .....	28
2.9. Performance Metrics .....	28
2.10. Implementation Details.....	29
2.10.1. Cross-Validation/Grid-Search Algorithm .....	29
2.10.2. Using Fast Computers in the Cloud .....	32
<b>3. Experiments</b> .....	33
<b>4. Results</b> .....	36
4.1. Cross-dataset and Cross-device Experiments.....	38
4.2. Crossmatch 2013 dataset and Dataset Visualization.....	39
4.3. Average Processing Times and Memory Usage.....	42
<b>5. Conclusion</b> .....	44
5.1. Future Work .....	44
<b>References</b> .....	46
<b>Appendix A</b> .....	51



# ***1. Introduction***

Biometric systems have become increasingly important in the past years. The basic aim of biometrics is to discriminate automatically between subjects in a reliable way and according to some target application based on one or more signals derived from physical or behavioral traits, such as fingerprint, face, iris, voice, hand, or written signature. Biometric technology presents several advantages over classical security methods based on either some information (PIN, Password, etc.) or physical device (key, card, etc.). Of all biometric systems, fingerprint recognition systems are the most popular and are extensively being used. However, providing to the sensor a fake physical biometric can be an easy way to overtake the system's security. Fingerprints, in particular, can be spoofed from common materials, such as gelatin, silicone, and wood glue [1]. Its creation can be divided into two categories:

- Without cooperation: The casts are created from latent fingerprints only.
- With cooperation: the user presses his finger onto a cast for creating his fingerprint impression, which normally produces better quality spoof fingerprints.

A safe fingerprint system must distinguish correctly a fake from an authentic finger. Additionally, it is desirable that it should be able to differentiate real from fake fingerprints when new and, therefore, unseen spoof techniques are presented to the system. A particularly interesting fact is that it is difficult for a non-specialist human to distinguish false from real fingerprints. Since humans can recognize patterns better than machines (in many cases), it can be concluded that the automatic fingerprint liveness detection is not a trivial problem.

In practical applications, the classification is mostly made online, that is, a decision whether the fingerprint is false or real is made right after it is inputted to the system. Therefore, the samples must be classified in a short amount of time (typically, less than 5 seconds) in order to provide a pleasant user experience, especially in environments where the number of incoming and outgoing users is high, like banks and public buildings.

## ***1.1. Related Work***

Different fingerprint liveness detection algorithms have been proposed [2] [3] [4]. They can be roughly divided into hardware-based and software-based techniques.

In the hardware-based approach, some specific device is added to the sensor in order to detect particular properties of a living trait such as the blood pressure [5], temperature [6], odor [7], or perspiration [8] [9] [10]. A method proposed in [11] tries to solve the problem using skin distortion, which involves pressing and moving a finger on the scanner surface to create a skin distortion. The distortion produced due to the movement of an elastic real skin is large compared to that produced by the movement of a rigid spoof finger. A method for detecting fake fingers by measuring electrical characteristics of different layers of the skin was proposed in [12]. They have used different characteristics of the skin like stratum corneum impedance, viable skin impedance, dispersive behavior of skin layers in the measured frequency range and anisotropy in stratum corneum for liveness detection. Some of these methods are slow as they need finger to be placed on the scanner surface for a couple of seconds so that information such as perspiration or temperature is available.

In the software-based approach, fake traits are detected once the sample has been acquired with a standard sensor. The features used to distinguish between real and fake fingers are extracted from the fingerprint image, and not from the finger itself.

The advantages of the software-based techniques is that the sensor do not have to be replaced as spoof techniques evolve and they also reduce the cost of a biometric system, as no additional hardware is needed, except for the fact that more computational power could be required to process the images in real-time. On the opposite side, the use of additional information not present in still images acquired from a standard sensor is one of the advantages of the hardware techniques.

There are software-based techniques in which the features used in the classifier are based on specific fingerprint measurements, such as ridge-based [13] [14] and Fourier transform-based [15] features, and there are implementations that the features are extracted using general extractors, such as wavelets or Local Binary Patterns (LBP).

In [13], a variety of quality measurements, such as ridge strength, continuity and clarity, are extracted from the fingerprint image using statistical measurements of the local angles, power spectrum and pixels intensities. A feature selection is then performed in the validation phase and a Linear Discriminant Analysis (LDA) classifier is used to make the final prediction. The results show 90% overall accuracy in two standardized benchmarks.

In [16], wavelets are used as feature vectors. Real and spoof fingerprints have significant differences in inter-ridge distances and ridge frequencies also. Wavelet analysis provides multi-resolution and orientation representation of a fingerprint image via subbands. Due to multi-resolution property of wavelets, minute textural differences in real and spoof fingerprints are analyzed in the wavelet domain. Additionally, the wavelet detail subbands carry high frequency information, which is very significant for texture characterization. However, the method has some drawbacks: the images used were the first image immediately upon placement on the fingerprint scanner. It is not known if this would be applicable to “any” fingerprint image since some devices wait until full development of the image before matching (~1 second). Also, because the method is based on detection of perspiration along the fingerprints, wiping the clothes before scanning may be necessary.

A combination of techniques, such as different classifiers (k-NN, SVM and Adaboost) fused using the “Majority Vote Rule” and trained with different feature extractors (LBP and wavelets) is used in [17]. The authors observe that the performance of both LBP histogram features and wavelet energy features is approximately the same and that the performance of a hybrid classifier is slightly better than the performance of individual classifiers.

A multi-scale variant for LBPs reported in [18] achieves good results in fingerprint liveness detection benchmarks. Since the original LBP operator is able to capture only small spatial support areas, the texture of fingerprint images could be too complex to be completely reflected by it. Besides, the LBP feature is sensitive to noise [19]. Thus, the multi-scale LBP operator (MSLBP) is introduced by applying multiple LBP filters with different radius and combining the histograms of each resulting LBP image into a single feature vector. With the increase in the LBP scale, the large distances between samples make the LBP codes unreliable. Hence, the MSLBP operator is combined with a set of Gaussian low-pass filters. A SVM classifier is then trained to make the final prediction.

Since there is no way to know in advance the materials and techniques of the fake fingerprints used by the attackers, it is necessary to study the inter-operability of the training classifiers across

different materials of fake fingerprints. That is to say, the authors in [18] tried to detect the spoof fingerprint images made of a specific type of material without training the spoof images made of the same material. They reported an averaged error rate of 22%, being much higher than the 7.5% error rate achieved when using the standard training and testing sets. Therefore, the proposed algorithm might have a poor performance in practical applications, as new techniques for spoof fingerprints emerge continuously.

The paper also presents the inter-operability performance of the trained classifiers across different devices, also called “Cross-device”. The error rate is about 40-50%, mainly caused by the huge difference among the images acquired by distinct sensors, according to the authors. Therefore, it was difficult for the classifier trained by the images collected by one sensor to distinguish the live images from the spoof images acquired by another sensor.

In some applications, like fingerprint liveness, image degradations may limit the applicability of the texture information. One class of degradation is blur due to motion or lack of focus. Because image deblurring is very difficult and introduces new artifacts, it is desirable to be able to analyze texture in a way that is insensitive to blur. Reference [20] tries to solve the problem through Local Phase Quantization (LPQ) as features from the fingerprint images. The descriptor utilizes quantized phase of the discrete Fourier transform (DFT) computed locally in a window for every image position. The phases of the four low-frequency coefficients are decorrelated and uniformly quantized in an eight-dimensional space. A histogram of the resulting code words is created and used as a feature in texture classification. Ideally, the low-frequency phase components are shown to be invariant to centrally symmetric blur. Although this ideal invariance is not completely achieved due to the finite window size, the method is still highly insensitive to blur. Because only phase information is used, the method is also invariant to uniform illumination changes.

Thus, the authors argued that the effectiveness of LPQ lies in its ability to represent all spectrum characteristics of images in a very compact feature representation, thus avoiding redundant or blurred information. Since different fingerprint orientations may arise on a sensor surface, they adopted the rotation invariant extension of LPQ. The results show that LBP and LPQ have similar performance and preliminary experiments show that there exist complementarity among them, but it needs further studies.

Reference [21] tries to combine the qualities of both LBP and LPQ through a local image descriptor called Binarized Statistical Image Features (BSIF). The idea behind BSIF is to automatically learn a fixed set of filters from a small set of natural images, instead of using hand-crafted filters such as LBP or LPQ. To characterize the texture properties within each fingerprint sub-region, the histograms of pixels employing BSIF code values are then used. The value of each element (i.e. bit) in the BSIF binary code string is computed by binarizing the response of a linear filter with a threshold at zero. Each bit is associated with a different filter and the desired length of the bit string determines the number of filters used. The set of filters is learnt by independent component analysis (ICA), which maximizes the statistical independence of the filter responses.

The results are promising, but, since the experiments were made using predefined filters learned from only 13 natural images, the performance could be improved if the filters were learned from a larger set of images acquired from particular sensors.

## **1.2. Proposed Method**

We approach the problem by experimenting two general feature extractors: Convolutional Networks, which are, to our knowledge, used for the first time for this task, and Local Binary Patterns, which had a good performance in previous works. Convolutional Networks are a promising technique as they provide the state-of-the-art results in many computer vision tasks. In contrast to all techniques described so far, it uses multiple layers of local descriptors (obtained from a convolution of filter banks and down-sampling operations), which outputs a feature vector whose dimensions can represent large patches of the input image. Therefore, these feature vectors are able to represent more complex structures of the image than the single layer techniques.

Moreover, a variety of preprocessing techniques such as contrast normalization, image reduction, frequency filtering and Region Of Interest (ROI) extraction are tested, as apparently the efficacy of these methods were not explored in past publications. We also used a technique known as Dataset Augmentation to prevent overfitting and to increase the classifier's robustness to small translations. On the top of the pipelines, two classifiers were compared in order to verify their contribution to the overall system's performance: Support Vector Machines (SVM) with linear and Gaussian kernels, and k-Nearest Neighbors (k-NN).

An extensive search for the best combination of preprocessing operations, architectures and hyper-parameters was made during the validation phase, thanks to the fast computers available as cloud services like Amazon's Elastic Compute Cloud (EC2).

There are few works in the field that use standardized benchmarks, such as Liveness Detection Competition (LivDet) or Biometric Recognition Group (ATVs) database [22], to report results. This is mainly because these benchmarks were created just recently. In this work, the experiments were executed in the eleven datasets of the Liveness Detection Competition of the years 2009, 2011 and, 2013 and the results were compared with the state-of-the-art techniques.

The dissertation is organized as follows: in chapter 2 we present the basic concepts, methodology, datasets used in the experiments, an overview of the elements that compose the classification pipelines, followed by a more detailed explanation of each element and implementation particularities, such as code optimizations; in chapter 3 we describe the experiments conducted; in chapters 4 and 5 the results and conclusions are presented, respectively.

## ***2. Methodology***

### ***2.1. Basic Concepts***

Some basic concepts necessary to understand this dissertation are reviewed in this section, starting from Figure 1 that shows common definitions for a fingerprint image:

**Region of Interest:** region of the image where the fingerprint lies.

**Background:** region of the image where there is no fingerprint, that is, region where there is no relevant information for classification.

**Dirtiness on the sensor:** dirtiness deposited on the sensor each time a finger is pressed against the glass cap. It accumulates mainly in the edges of the glass cap.

**Ridge:** The skin on the palmar surface of the hands and feet forms ridges, so-called papillary ridges, in patterns that are unique to each individual and which do not change over time.

**Ridge Distance:** distance between two adjacent ridges.

**Valleys:** region between ridges.

**Micropores:** Also called pores, they are sweat glands irregularly spaced on the ridges. There are no pores between the ridges, though sweat tends to spill into them. The thick epidermis of the palms and soles causes the sweat glands to become spirally coiled.

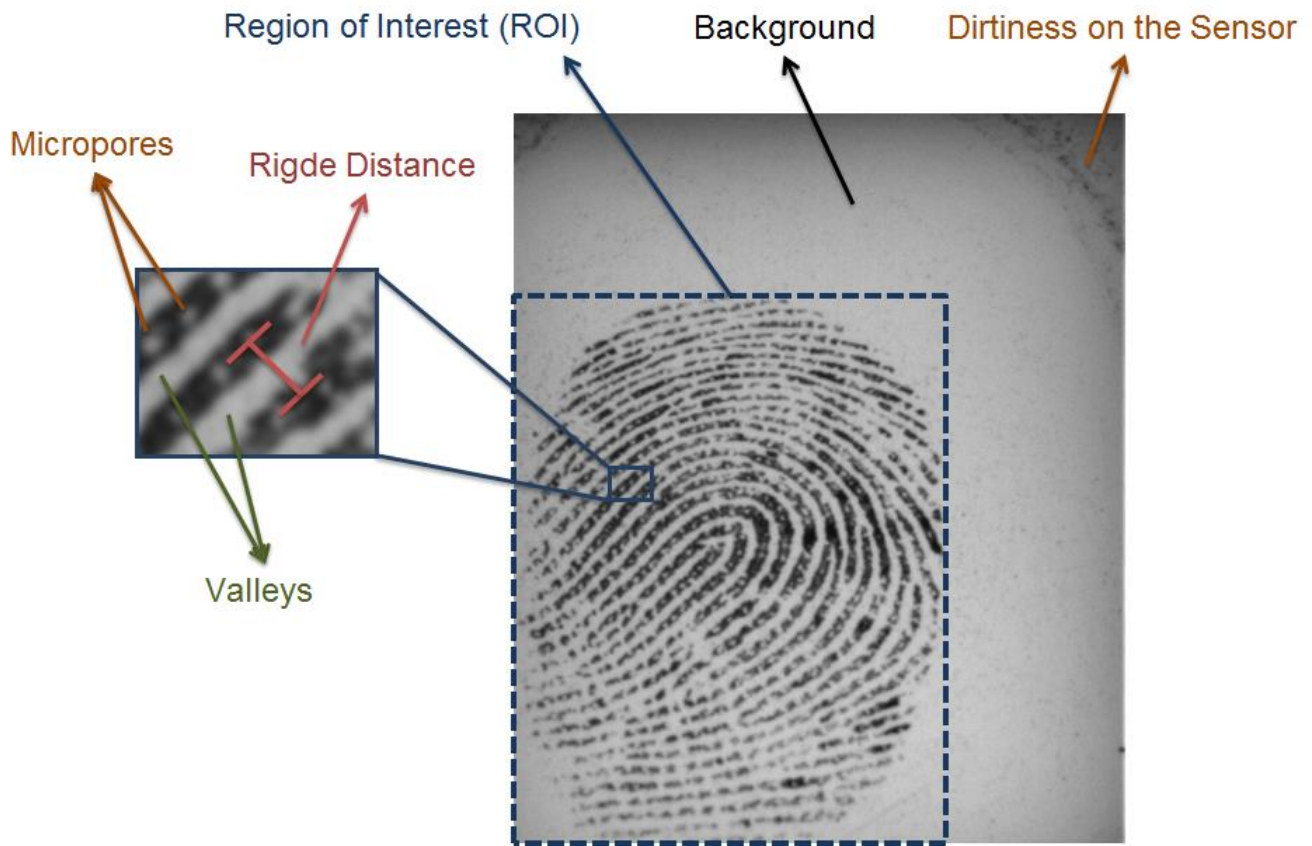


Figure 1 – Some fingerprint image definitions.

Next, some concepts from computer vision and machine learning are described:

**Histogram:** An image histogram is a representation of the tonal (intensities values) distribution in a digital image. It computes the number of pixels for each tonal value, which are referred as “bins”. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance. Histograms are commonly used to represent the features of the images in machine learning algorithms.

**Convolution:** It is a linear filter characterized by its point spread function  $g$ . The equation of a convolution of image  $f$  with a filter  $g$  is:

$$f[m, n] * g[m, n] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u, v]g[u - m, v - n] \quad (1)$$

**Center of Mass** (also called image mean): For a bi-dimensional image of size M-by-N, the center of mass  $(x_{cm}, y_{cm})$  can be calculated as:

$$x_{cm} = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} i v_{ij}}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} v_{ij}} \quad (2)$$



$$y_{cm} = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} j v_{ij}}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} v_{ij}} \quad (3)$$

where  $v_{ij}$  is the pixel value at  $(i, j)$  coordinates.

**Standard deviation:** The standard deviation  $(x_{std}, y_{std})$  of a bi-dimensional image can be calculated as:

$$x_{std} = \sqrt{\frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (i - x_{cm})^2 v_{ij}}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} v_{ij}}} \quad (4)$$

$$y_{std} = \sqrt{\frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (j - y_{cm})^2 v_{ij}}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} v_{ij}}} \quad (5)$$

**Morphological Opening:** In mathematical morphology, opening is the dilation of the erosion of a set  $A$  by a structuring element  $B$ :

$$A \circ B = (A \ominus B) \oplus B, \quad (6)$$

where  $\ominus$  and  $\oplus$  denote erosion and dilation, respectively. Opening removes small objects from the foreground (usually taken as the dark pixels) of an image, placing them in the background. Opening can be used to find things into which a specific structuring element can fit (edges, corners, etc.). One can think of  $B$  sweeping around the inside of the boundary of  $A$ , so that it does not extend beyond the boundary, and shaping the  $A$  boundary around the boundary of the element. [23]

**Supervised Learning:** it is the machine learning task of inferring a function from labeled training data. The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input vector (typically attributes of an object) and a desired output value (also called the *supervisory signal*). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. [24]

**Unsupervised Learning** [25]: Unsupervised learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns.

By contrast with supervised learning, there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output. In other terms, the problem of unsupervised learning is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to directly evaluate a potential solution.

**Feature Extraction/Dimensionality Reduction** [26]: dimensionality reduction or dimension reduction is the process of reducing the number of random variables under consideration and it is

normally used in conjunction with the terms feature selection and feature extraction. Feature extraction creates new features from functions of the original features, whereas feature selection returns a subset of the features. When the input data to an algorithm is too large to be processed and it is suspected to be notoriously redundant then the input data will be transformed into a reduced representation set of features (also named features vector). Transforming the input data into the set of features is called *feature extraction*. If the features extracted are carefully chosen it is expected that the features set will extract the relevant information from the input data in order to perform the desired task using this reduced representation instead of the full size input.

**Overfitting** [27]: It occurs when a statistical model describes random error or noise together with the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model which has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data. The possibility of overfitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. In particular, a model is typically trained by maximizing its performance on some set of training data. However, its efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data. Overfitting occurs when a model begins to interpolate training data rather than learning to generalize from trend. As an extreme example, if the number of parameters is the same as or greater than the number of observations, a simple model or learning process can perfectly predict the training data simply by interpolating the training data in its entirety, but such a model will typically fail drastically when making predictions about new or unseen data, since the simple model has not learned to generalize at all.

**Curse of Dimensionality** [28]: It refers to various phenomena that arise when analyzing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. When the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient. In the specific case of a binary (two-classes) classifier, it refers to when there are much more dimensions than training samples that the classifier is not able to generalize due to large amount of possible separation hypersurfaces.

**Cross-Validation** [29]: it is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction or classification, and one wants to estimate how accurately a predictive model will perform in practice. It is worth highlighting that in a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems

like overfitting, give an insight on how the model will generalize to an independent data set (i.e., an unknown dataset, for instance from a real problem), etc.

## 2.2. Datasets

We used datasets provided by the Livness Detection Competition of the years 2009 [30], 2011 [31], and 2013 [32]. The competition is organized by the Department of Electrical and Electronic Engineering of the University of Cagliari, in cooperation with the Department of Electrical and Computer Engineering of the Clarkson University, and it is open to all academic and industrial institutions. It features two distinct parts; Part 1: Algorithms and Part 2: Systems, with separate protocols for each part. Since the main focus of this work is software techniques, only Part 1 will be described.

Table 1 shows the image size and number of samples for training and testing of each dataset. In all datasets, the real/fake fingerprint ratio is 1/1. The sizes of the images vary from sensor to sensor, ranging from 240x320 to 700x800 pixels. Figure 2 and Figure 3 show some image samples used in the competitions of years 2013 and 2009, respectively.

Livdet 2009 dataset comprises almost 18,000 images from real and fake fingerprints acquired from 3 different sensors: Biometrika FX2000, Crossmatch Verifier 300 LC, and Identix DFR 2100. Fake fingerprints were obtained from three different materials: Gelatin, Play Doh, and Silicone. One third of the dataset is used for training and the remaining for testing.

LivDet 2011 dataset comprises 16,000 images acquired from 4 different sensors: Biometrika FX2000, Digital 4000B, Italdata ET10, and Sagem MSO300, each having 2000 images from fake and real fingerprints. Half of the dataset is used for training and the other half for testing. Fake fingerprints were obtained from four different materials: Gelatin, Wood Glue, Eco Flex, and Silgum.

LivDet 2013 dataset comprises 16,000 images acquired from 4 different sensors: Biometrika FX2000, Crossmatch L SCAN GUARDIAN, Italdata ET10, and Swipe, each having approximately 2,000 images from fake and real fingerprints. Almost half of the dataset is used for training and the other half for testing. Fake fingerprints were obtained from five different materials: Gelatin, Latex, Eco Flex, Wood Glue, and Modasil.

Additionally, we performed the experiments in a private dataset kindly provided by *Griaule Biometrics* (<http://www.griaulebiometrics.com>) that comprises approximately 1000 training images and 1000 testing images acquired from Futronic FS-88 Spoofs sensor. The spoof/real fingerprint ratio is 1 in both training and testing sets. Fake fingerprints were obtained from four different materials: Gelatin, Silicone, Latex, and Wood Glue. We will refer to this dataset throughout this work as “Griaule” dataset.

Table 1 - Datasets details: Image sizes and number of training and testing samples for each sensor.

Competition/ Year	Sensor	Model	Image Size	Samples for Training	Samples for Testing
LivDet 2009	Biometrika	FX2000	372x312	1040	2960
	Crossmatch	Verifier 300 LC	640x480	2000	6000
	Identix	DFR2100	720x720	1500	4500
LivDet 2011	Biometrika	FX2000	372x312	2000	2000
	Digital	4000B	355x391	2000	2000
	Italdata	ET10	640x480	2000	2000
	Sagem	MSO300	352x384	2000	2000
LivDet 2013	Crossmatch	L SCAN GUARDIAN	800x750	2250	2250
	Swipe	N/A	208x1500	2000	2153
	Italdata	ET10	640x480	2000	2000
	Biometrika	FX2000	372x312	2200	2000

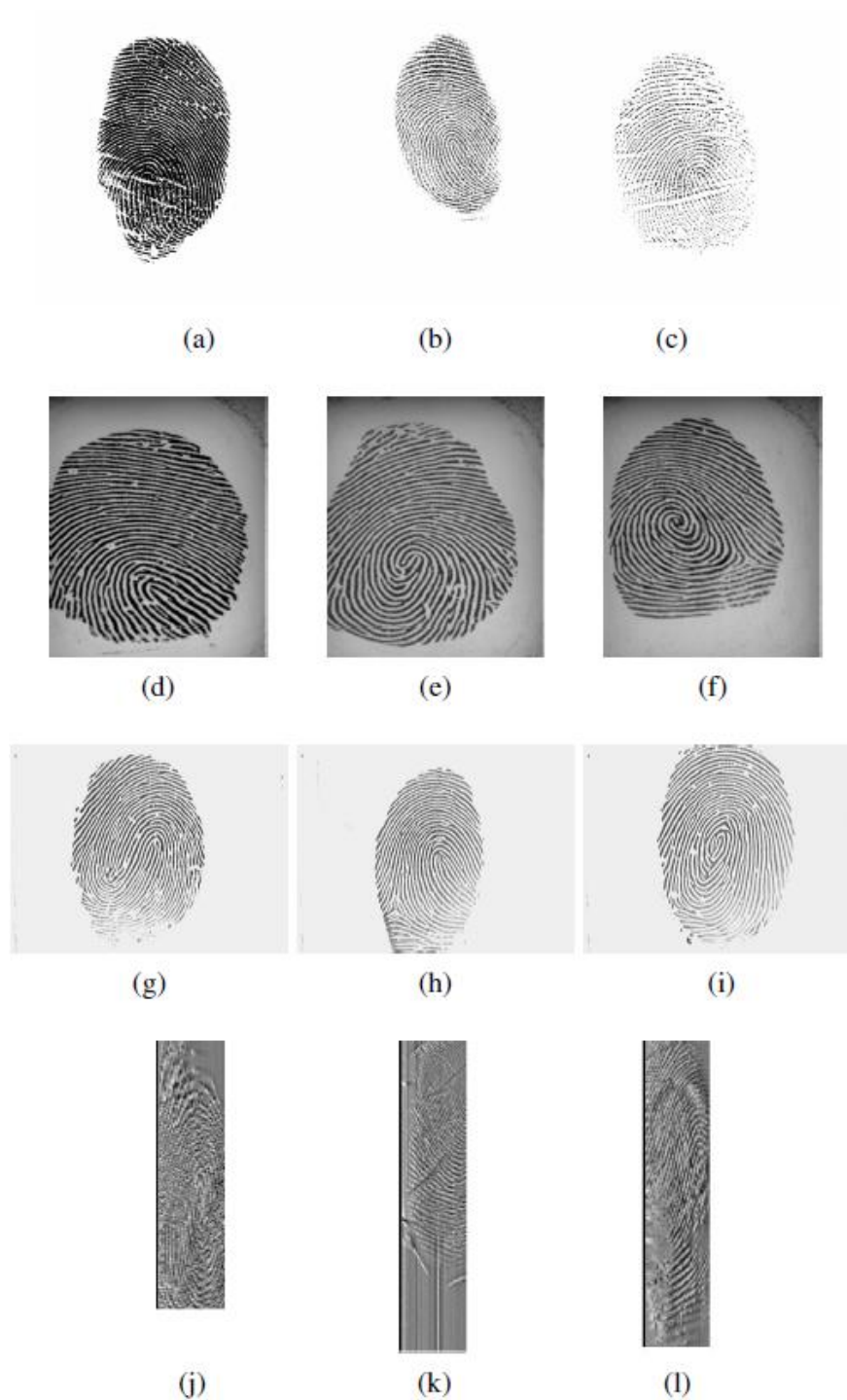


Figure 2 – Examples of fake fingerprints acquired with 4 sensors from LivDet 2013. From Crossmatch (a) body double, (b) latex, (c) wood glue, from Biometrika (d) gelatine, (e) latex, (f) wood glue, from Italdata (g) gelatine, (h) latex, (i) wood glue, from Swipe (j) body double, (k) latex, (l) wood glue.

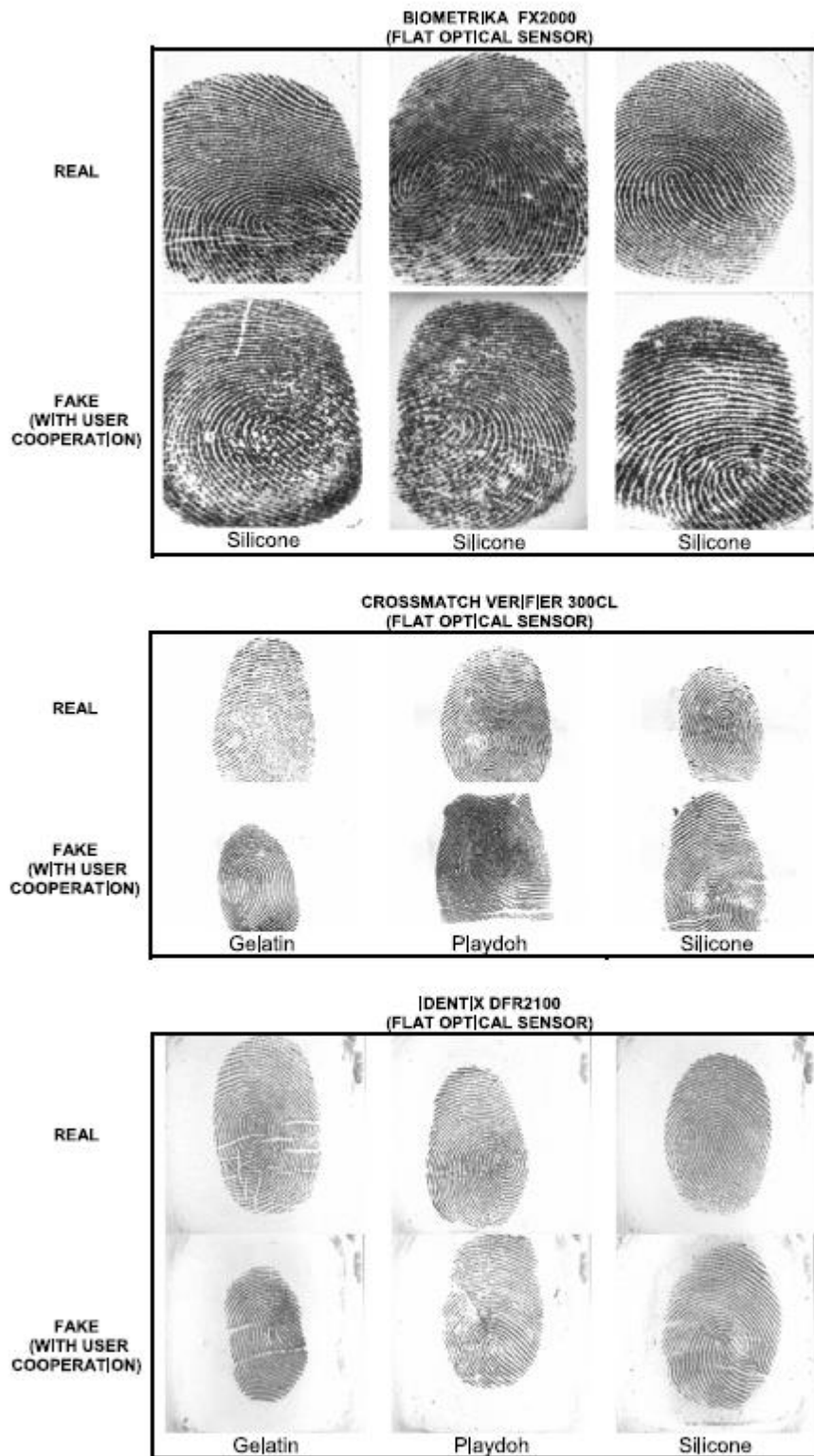


Figure 3 - Typical examples of real and fake fingerprint images that can be found in the LivDet2009 database used in the experiments.

### 2.3. Processing Flow

Figure 4 shows an overview of the pipeline used to train the classifiers, which can be broadly divided into four phases:

- 1- Preprocessing;
- 2- Feature Extraction where two techniques were tried: Local Binary Patterns and Convolutional Networks;
- 3- Dimensionality Reduction and Data Normalization;
- 4- Classification where two classifiers were tried: k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM) with Linear and Gaussian kernels.

Since testing all possible combinations of operations has a prohibitory computational cost, we selected a sub-set of these for our experiments, which will be listed in chapter 3.

For training and testing, we followed the same protocol used in LivDet competitions, that is, a fixed set of images is used for training and validation (for hyper-parameter selection) and the remaining for testing.

The implementation details of each phase will be explained in the following sub-sections.

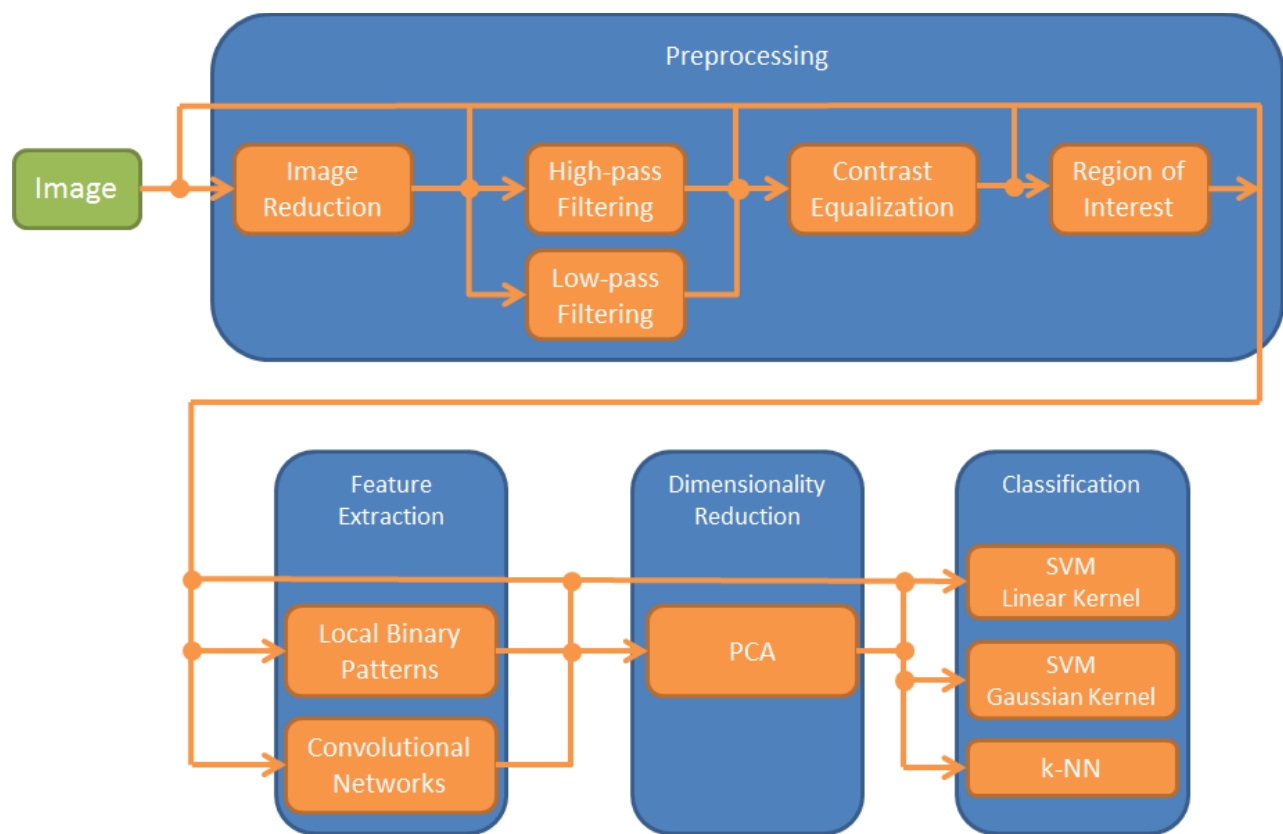


Figure 4: Overview of the processing flow

### 2.4. Preprocessing

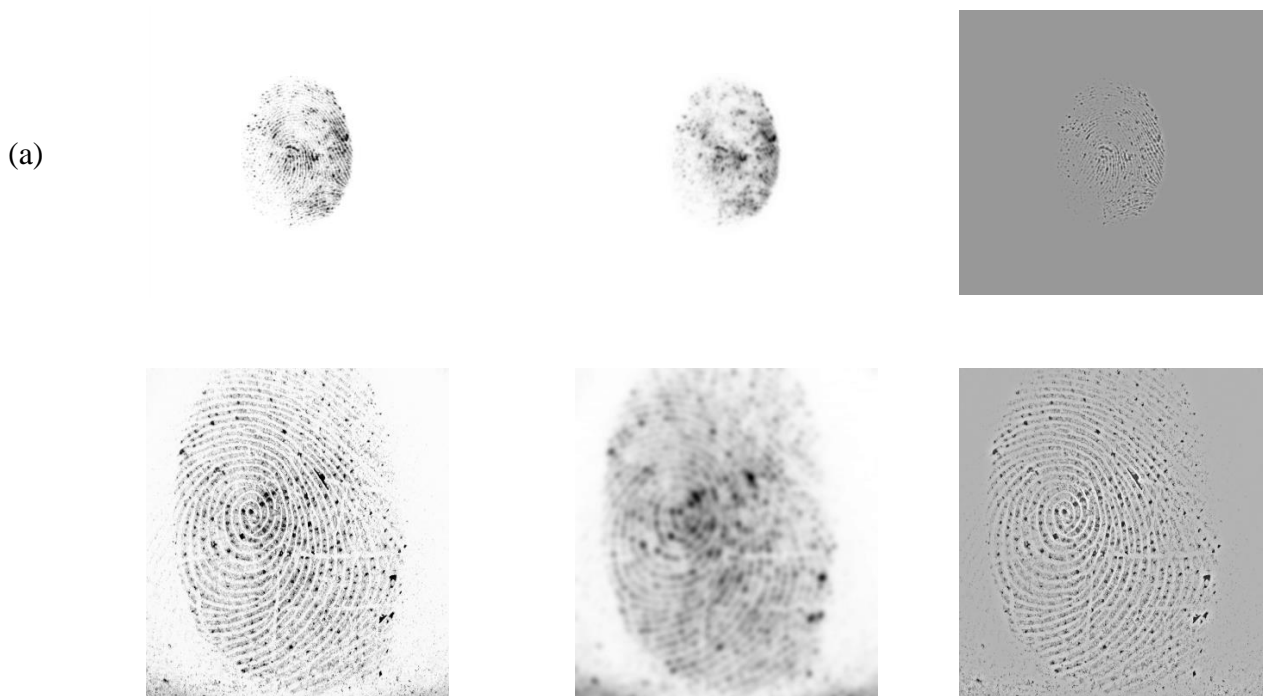
Five preprocessing operations were carried out: Image Reduction using different ratios, Region of Interest (ROI) extraction, Contrast Equalization, High-pass filter, and Low-pass filter. The execution or non-execution of each operation in the final model is decided at validation time, that is, the combination of preprocessing operations that had the lowest validation error were included in the final model.

### Image Reduction

Due to the large size of some images (800 x 750 pixels for Crossmatch sensor, for example) and the small amount of samples for training (approximately 2000 for each dataset), the classification system may not be able to extract the important information. This problem is known as the Curse of Dimensionality, already explained in section 2. Thus, the experiments were also performed in images resized (using bilinear interpolation) to 50% and 25% of its original size, and the best reduction ratio was chosen using cross-validation.

### Frequency Filtering

We inspected how noise removal through a Gaussian low-pass filtering could improve results. We also tested the hypothesis that the relevant information to distinguish between false and real fingerprints is mostly in the high frequency components of the image by applying a Gaussian high-pass filter before extracting the features. The low-pass filter is implemented as the convolution of the input image by a Gaussian kernel and the high-pass filter is implemented as the subtraction of the original image by the low-pass filtered image. In our experiments, either high pass or low-pass filter was applied (never both) and the Gaussian kernels have a standard deviation of 3 pixels and size of 13x13 pixels. Figure 5 shows the effect of both filters when applied to samples acquired from four types of sensors.





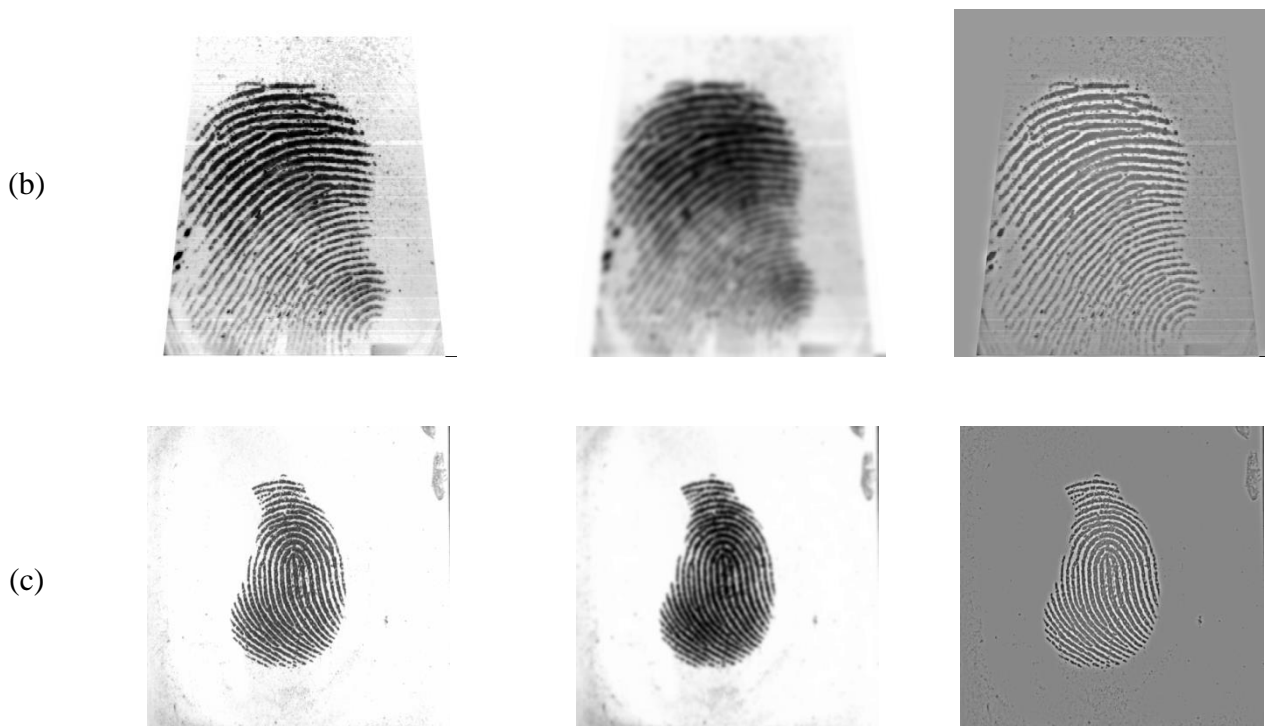


Figure 5 – Original (left), low-pass filtered (middle) and high-pass filtered (right) images. Each row represents an image from a specific dataset: (a) – Crossmatch 2013, (b) – Sagem 2011, (c) – Digital 2011 and (d) – Identix 2009.

### Region of Interest (ROI)

Many fingerprints from some datasets, like Crossmatch sensor from LivDet 2013 competition, are not centered and the background represents a large part of the image. In order to try to input to our classification system the largest area that comprises foreground/fingerprints, we created a simple ROI using the following steps:

1. Apply morphological closing operation to highlight the region where the fingerprint lies. We used a box of size  $21 \times 21$  as the structuring element, which is greater than the maximum ridges distances even in the largest images (that normally have greater ridge distances). This ensures that the fingerprint will become a continuous object after the operation.
2. Negate image, so the foreground/fingerprint will have greater values than the background.
3. Find the center of mass and the standard deviation of the negated image from step 2.
4. Get the Region of Interest: a rectangle centered in the center of mass, whose width and height are three times the standard deviations calculated in the previous step.

Figure 6 illustrates the sequence of operations described above for a sample fingerprint image.

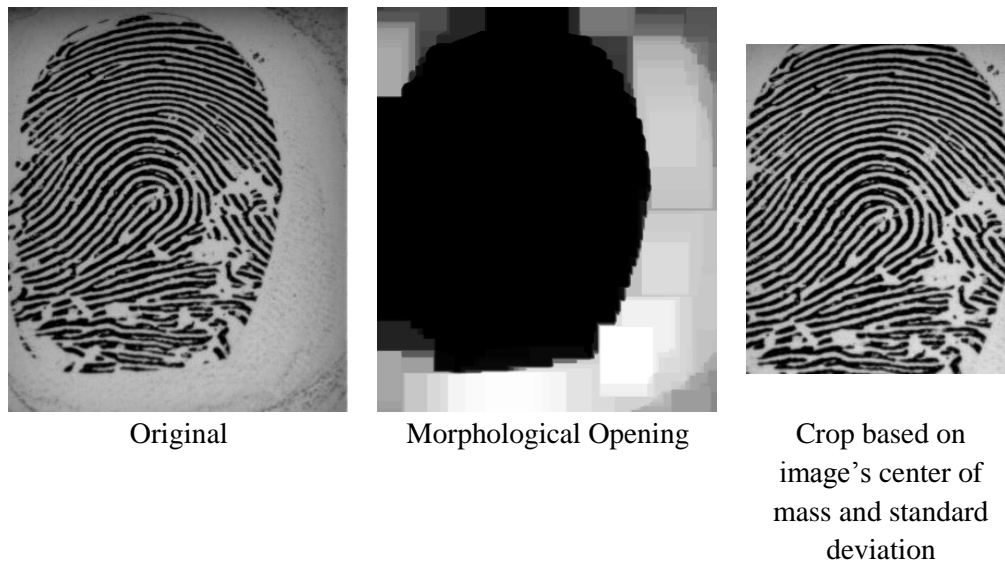
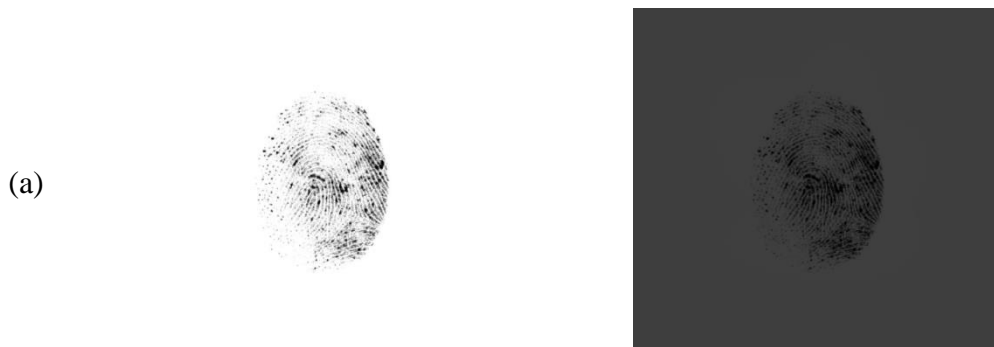


Figure 6 - Sequence of operations to extract the Region of Interest of a fingerprint image.

### Contrast Equalization

We verified if histogram equalization could improve our classifier performance by using a technique called Contrast Limited Adaptive Histogram Equalization (CLAHE) [33], which is a variant of Adaptive Histogram Equalization (AHE) [34]. AHE computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is, therefore, suitable for improving the local contrast of an image and bringing out more details. In our implementation, each pixel is transformed based on the histogram of region of a disk with a diameter of 30 pixels surrounding the center pixel. AHE has a tendency to overamplify noise in relatively homogeneous regions of an image. CLAHE prevents this by limiting the amplification by clipping the histogram at a predefined value before computing the neighborhood cumulative distribution function (CDF). Figure 7 shows the original and CLAHE filtered images for comparison.



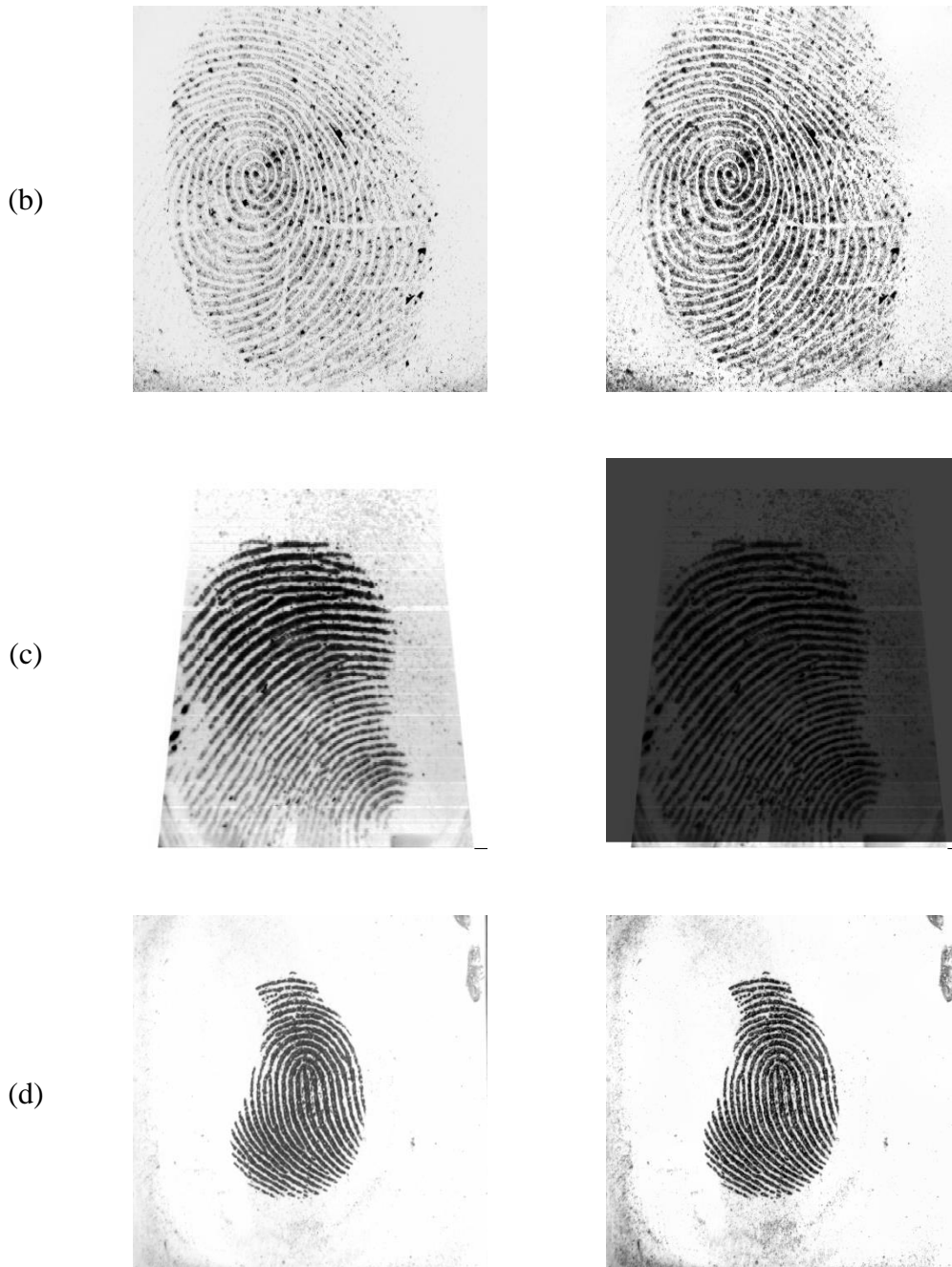


Figure 7 - Original images (left) and CLAHE filtered images (right). Each row represents an image from a different dataset: (a) – Crossmatch 2013, (b) – Sagem 2011, (c) – Digital 2011 and (d) – Identix 2009.

## 2.5. Feature Extraction

Two different feature extractors were tested: Convolutional Networks (CN) with random weights and Local Binary Patterns (LBP).

### 2.5.1. Convolutional Networks

Convolutional Networks [35] are the state-of-the-art technique in a variety of image recognition benchmarks, such as MNIST [36], CIFAR-10 [36], CIFAR-100 [37], SVHN [36] and ILSVRC2011 [38], and to the best of our knowledge, this is the first time it is employed in fingerprint liveness detection.

A classical convolutional network is composed of alternating layers of convolution and local pooling (i.e. subsampling) [39]. The aim of the first convolutional layer is to extract patterns found within local regions of the input images that are common throughout the dataset by convolving a template or filter over the input image pixels and outputting this as a feature map  $c$ , for each filter in the layer.

By stacking multiple layers, it is intended to create a system that would be able to capture more complex structures in the data. However, since the convolution is a linear operation and the combination of two or more linear operations is equivalent to a single linear operation, the effort to build a multiple layer network would be nullified. To avoid this, a non-linear function  $f(c)$  is applied element-wise to each feature map  $c$ :  $a = f(c)$ , resulting in a network composed of multiple non-linear layers. A range of functions can be used for  $f(c)$ , with  $\tanh(c)$  and logistic functions being popular choices. In this work, we use a linear rectification  $f(c) = \max(0; c)$  as the non-linearity function. In general, this has been shown [40] to have significant benefits over  $\tanh()$  or logistic functions.

The resulting activations  $f(c)$  are then passed to the pooling layer. This aggregates the information within a set of small local regions,  $R$ , producing a pooled feature map  $s$  (normally of smaller size) as output. Denoting the aggregation function as  $\text{pool}()$ , for all feature map  $c$  we have:

$$s_j = \text{pool}(f(c_i)) \forall i \in R_j \quad (7)$$

where  $R_j$  is the pooling region  $j$  in feature map  $c$  and  $i$  is the index of each element within it. Among the various types of pooling, two are commonly used: average and max. Average pooling outputs the average (or the summation) of the activations units in a neighbor region  $R_j$ :

$$s_j = \frac{1}{|R_j|} \sum_{i \in R_j} a_i \quad (8)$$

Max pooling selects the maximum value of the region  $R_j$ :

$$s_j = \max_{i \in R_j} a_i \quad (9)$$

The motivation behind pooling is that the activations in the pooled map  $s$  are less sensitive to the precise locations of structures within the image than the original feature map  $c$ . In a multi-layer model, the convolutional layers, which take the pooled maps as input, can thus extract features that are increasingly invariant to local transformations of the input image [41] [42]. This is important for classification tasks, since these transformations obfuscate the object identity. Achieving invariance to changes in position or lighting conditions, robustness to clutter, and compactness of representation, are all common goals of pooling.

Another important characteristic of convolutional networks is its ability to capture larger regions, and possibly more complex structures, of the input images than the single layer methods like LBP. This is achieved by stacking local descriptors (obtained from a convolution of filter banks and down-

sampling operations) in multiple layers, which increases the area of input image represented by single dimension of the final feature vector.

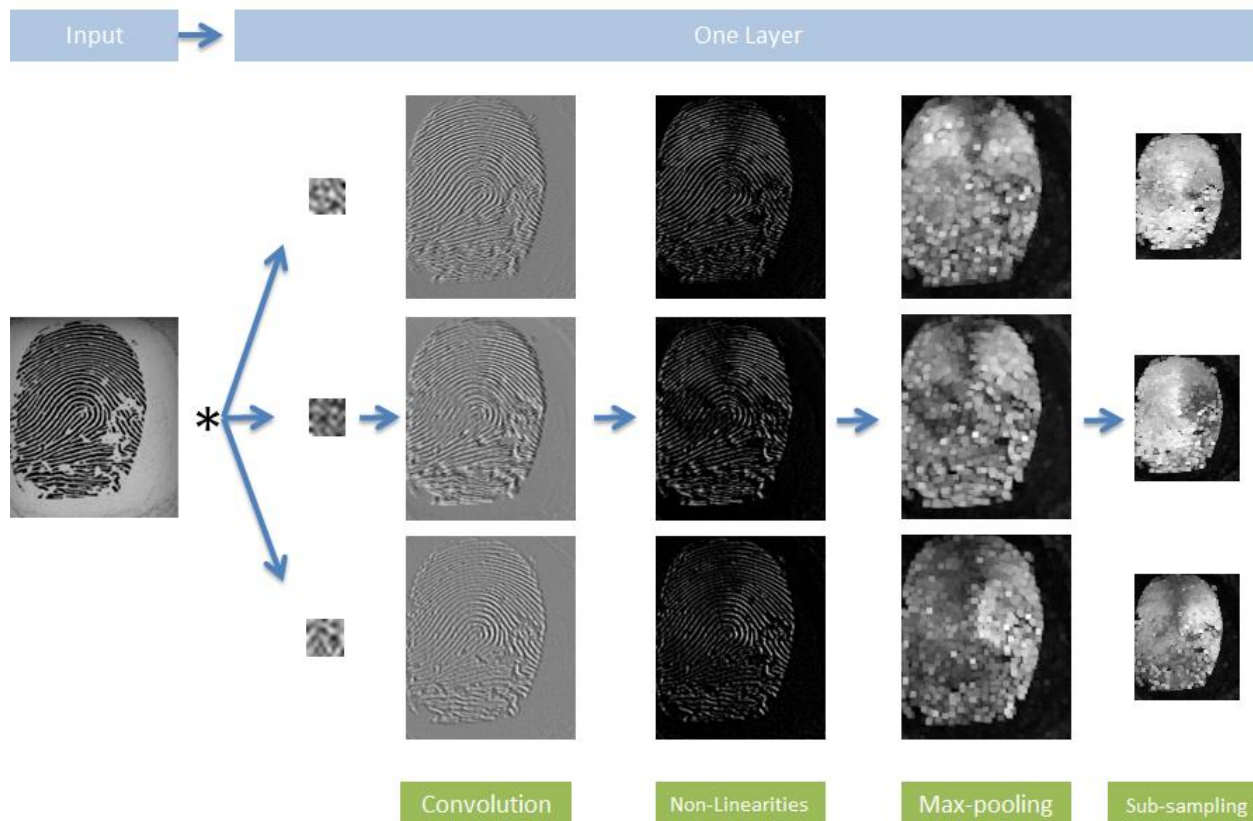


Figure 8 - Illustration of a sequence of operations performed by a single layer convolutional network in a sample image.

Figure 8 illustrates the feed-forward pass of a single layer convolutional network. The input sample is convoluted with three random filters of size 5x5 (enlarged to make visualization easier), generating 3 convoluted images, which are then subjected to a non-linear function  $\max(x,0)$ , followed by a max-pooling operation and then subsampled by a factor of 2.

Figure 9 illustrates a sample sequence of operations for a convolutional network with two layers (the non-linear and max-pooling operations are not shown). In the second layer, the outputted images from the first layer are convoluted with 9 random filters (only three are displayed), max-pooled, and sub-sampled. The outputted images are normally rasterized and concatenated forming a one-dimensional vector that will be fed in a classifier (not shown in the illustration).

Our convolutional networks use only random filters weights draw from a Gaussian distribution. Although the filter weights can be learned, as described in [43], filters with random weights can perform surprisingly well and they have the advantage that they do not need to be learned [44] [45] [46], decreasing the pipeline’s training time.

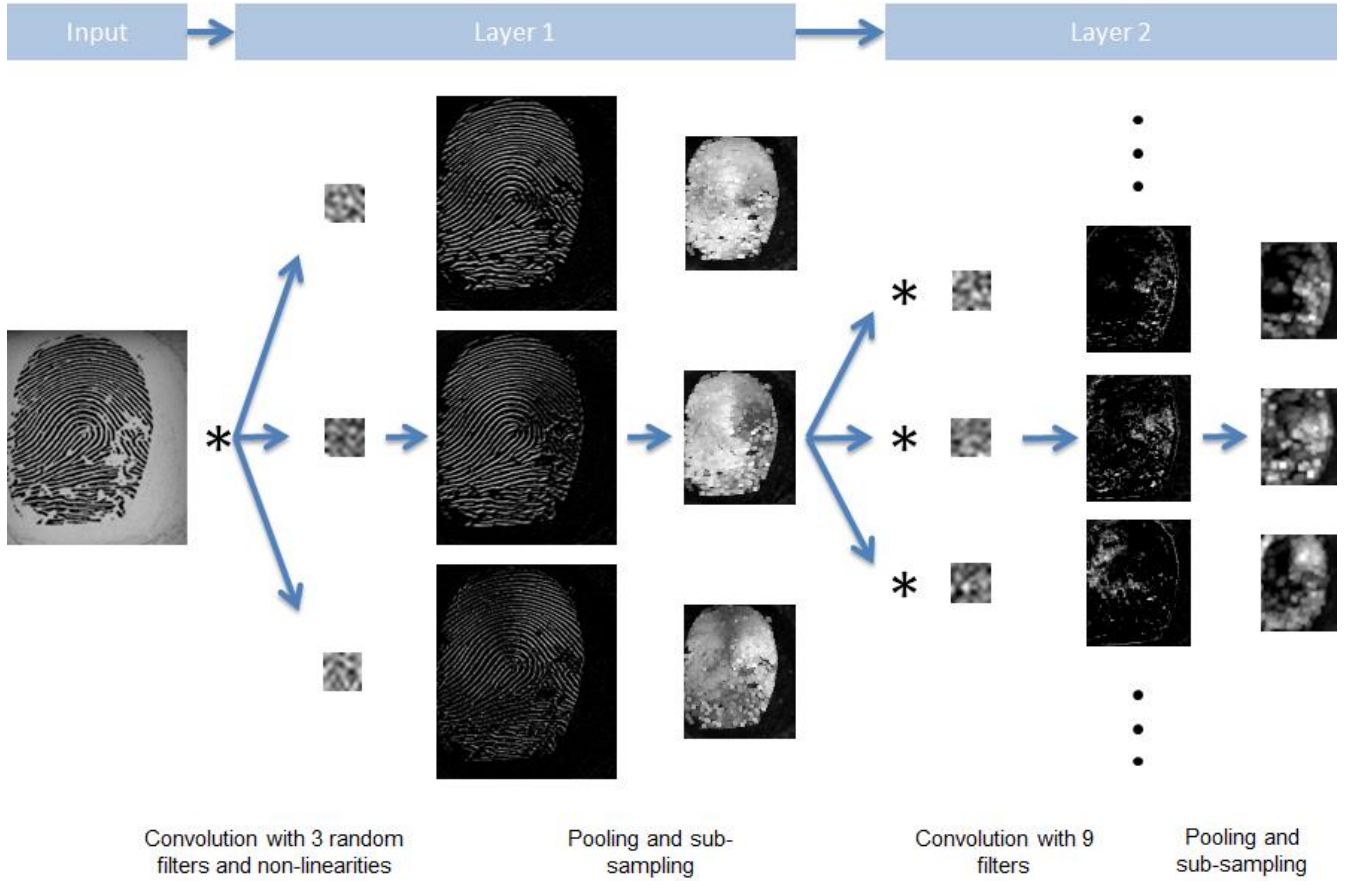


Figure 9 – Illustration of a sequence of operations performed by a two layers convolutional network in a sample image.

It is a common practice to have a Local Contrast Normalization layer (which is different from the Contrast Equalization previously described) between each convolution and pooling layer. The goal of this layer is to normalize pixels intensities based on its neighborhood. The operations of subtractive and divisive normalization described below are inspired by computational neuroscience models [47] [48] [49]. The subtractive normalization operation for a given 3D image patch  $x_{ijk}$  can be defined by:

$$V_{ijk} = x_{ijk} - \sum_{ipq} w_{pq} \cdot x_{i,j+p,k+q} \quad (10)$$

where  $w_{pq}$  is a Gaussian weighting window normalized so that

$$\sum_{ipq} w_{pq} = 1 \quad (11)$$

$i$  refers to the index of the third dimension of the image patch,  $j$  and  $k$  refer to the two dimensions of the image patch,  $p$  and  $q$  refer to the neighborhood region of the patch defined by  $j$  and  $k$ . The divisive normalization computes

$$y_{ijk} = v_{ijk} / \max(c, \sigma_{jk}) \quad (12)$$

where

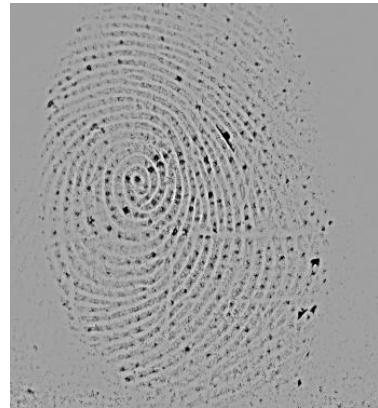
$$\sigma_{jk} = \left( \sum_{ipq} w_{pq} \cdot v_{i,j+p,k+q}^2 \right)^{1/2} \quad (\text{in our experiments } c = 1) \quad (13)$$

Figure 10 shows divisive normalization filtering with filter size of 9x9 applied to some fingerprint image samples.

(a)



(b)



(c)

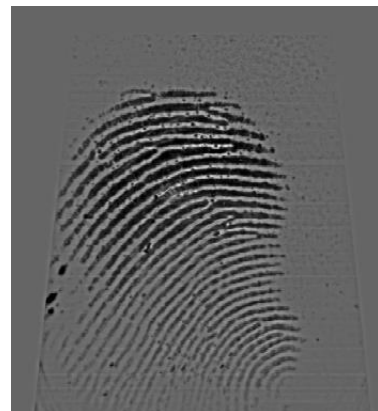




Figure 10 – Original (Right) and Divisive Normalized (Left) Images. Each row represents an image from a different dataset: (a) – Crossmatch 2013, (b) – Sagem 2011, (c) – Digital 2011 and (d) – Identix 2009.

### 2.5.2. Local Binary Pattern

Local Binary Patterns (LBP) are a local texture descriptor that have performed well in various computer vision applications, including texture classification and segmentation, image retrieval, surface inspection, and face detection [50]. The best current method for fingerprint liveness detection [18] uses this technique.

In its original version, the LBP operator assigns a label to every pixel of an image by thresholding each of the 8 neighbors of the 3x3-neighborhood with the center pixel value and considering the result as a unique 8-bit code representing the 256 possible neighborhood combinations. As the comparison with the neighborhood is done with the central pixel, the LBP is an illumination invariant descriptor. The operator can be extended to use neighborhoods of different sizes [19].

In mathematical terms, the LBP label for the center pixel  $(x,y)$  of image  $f(x,y)$  is obtained through

$$LBP_{P,R}(x,y) = \sum_{p=0}^{P-1} s(f(x,y) - f(x_p, y_p)) 2^p \quad (14)$$

where  $P$  represents the number of sampling points,  $R$  is the radius of the neighborhood,  $s(z)$  is the thresholding function and

$$s(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \quad (15)$$

Normally, the normalized histogram of the LBPs is used as a feature vector for an image or a ROI of an image, as the histogram gives the frequency distribution of each particular pattern in the image.





Figure 11 – Original image (Left) and LBP filtered image (Right).

Another extension to the original operator is the definition of so-called *uniform patterns*, which can be used to reduce the length of the feature vector and implement a simple rotation-invariant descriptor [19]. An LBP is called uniform if the binary pattern contains at most two bitwise transitions from 0 to 1 or vice versa when the bit pattern is considered circular. The number of different labels of LBP is reduced from 256 to just 10 in the uniform pattern.

The original rotation-invariant LBP operator based on uniform patterns is achieved by circularly rotating each bit pattern to the minimum value. For instance, the bit sequences 10000011, 11100000, and 00111000 arise from different rotations of the same local pattern, and they all correspond to the normalized sequence 00000111. Figure 12 shows the 58 possible different uniform patterns in the (8, R) neighborhood. In the uniform operator, all the patterns from one row are replaced with a single label, which results in 9 possible labels. The remaining non-uniform patterns are assigned to the 10<sup>th</sup> label.

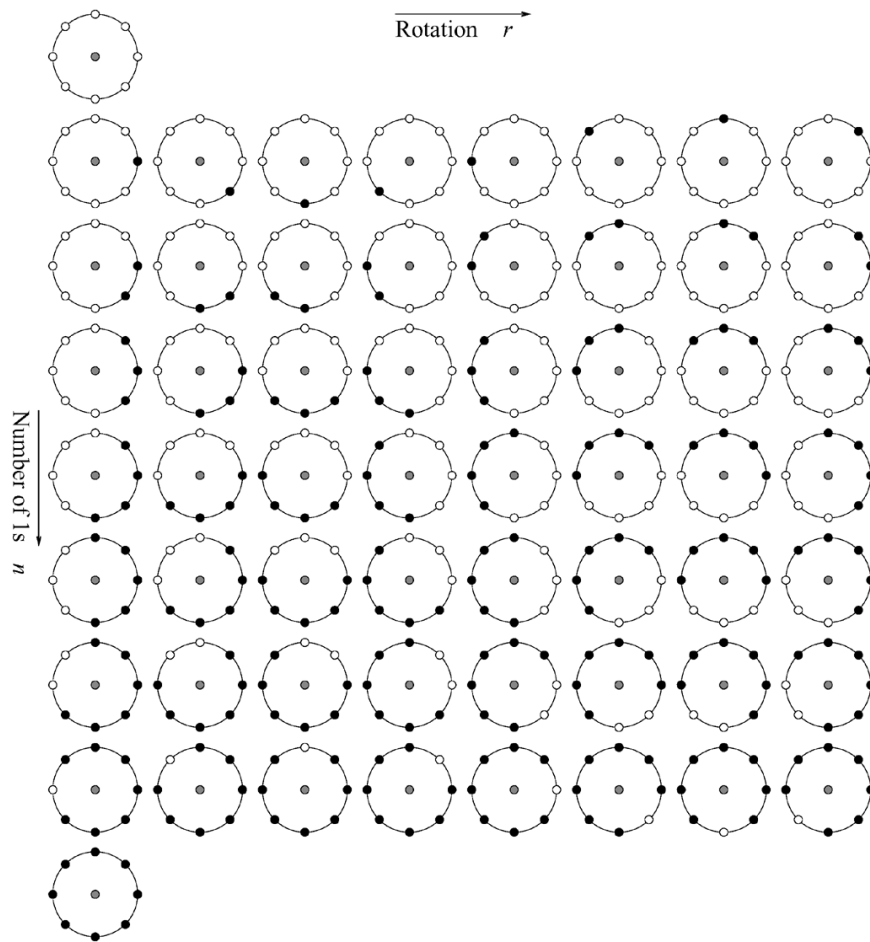


Figure 12 - Fifty-eight different uniform patterns in the  $(8, R)$  neighborhood. Source: [51].

The normalized histogram of the LBPs (with 256 and 10 bins for non-uniform and uniform operators, respectively) is used as a feature vector. The assumption underlying the computation of a histogram is that the distribution of patterns matters, but the exact spatial location does not. Thus, the advantage of extracting the histogram is the spatial invariance property. To investigate if location matters to our problem, we also implemented the method presented in [52], for face recognition, where the LBP filtered images are equally divided in rectangles and their histograms are concatenated to form a final feature vector, as exemplified in Figure 13.

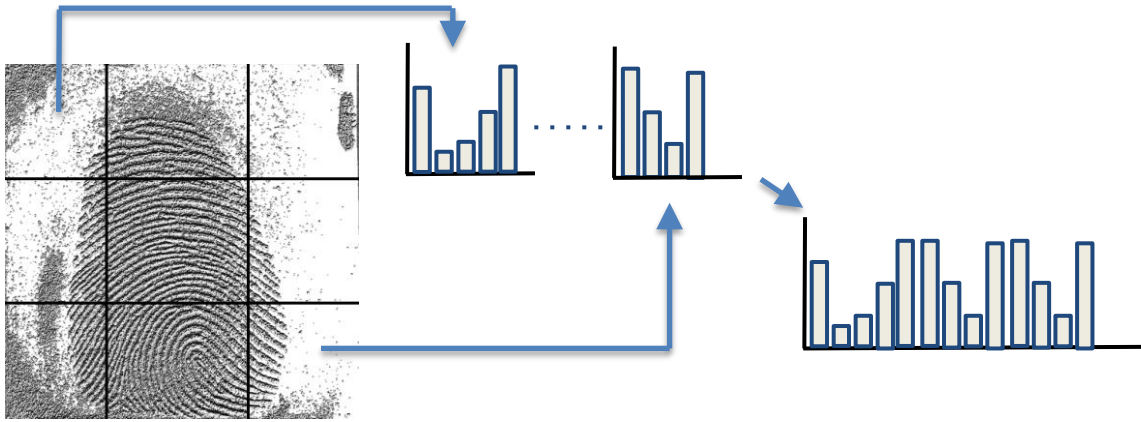


Figure 13 – the LBP filtered images are equally divided in rectangles and their histograms are concatenated to form a final feature vector.

## 2.6. Feature Normalization, Dimensionality Reduction and Whitening

After the feature extraction phase, each dimension of the dataset is independently normalized to zero mean and unit variance. This is normally required because many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines) assume that all features are centered on zero and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features as expected.

The normalized data is then subject to dimension reduction by using Principal Components Analysis (PCA), which is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [53]. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. The PCA can be computed by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition (SVD) of a data matrix. We will explain only the latter, as it is the most common implementation. The singular value decomposition of an  $n$ -by- $p$  data matrix  $X$ , where the  $n$  rows represents the samples and the  $p$  columns represents the features, is defined as

$$X = USV^T$$

where  $S$  is a  $n$ -by- $p$  rectangular diagonal matrix of positive numbers  $\sigma(k)$ , called the singular values of  $X$ ;  $U$  is an  $n$ -by- $n$  matrix, the columns of which are orthogonal unit vectors of length  $n$  called the left singular vectors of  $X$ ; and  $V$  is a  $p$ -by- $p$  matrix whose columns are orthogonal unit vectors of length  $p$  and called the right singular vectors of  $X$ . By convention, the ordering of the singular vectors is determined by high-to-low sorting of singular values, with the highest singular value in the upper left index of the  $S$  matrix. The principal components score matrix  $T$  can be written

$$\begin{aligned} T &= XV \\ &= USV^T V \end{aligned}$$

$$= US$$

so each column of  $T$  is given by one of the left singular vectors of  $X$  multiplied by the corresponding singular value. Keeping only the first  $L$  principal components, produced by using only the first  $L$  loading vectors, gives the truncated transformation

$$T_L = U_L S_L = X V_L$$

where the matrix  $T_L$  now has  $n$  rows but only  $L$  columns. By construction, of all the transformed data matrices with only  $L$  columns, this score matrix maximizes the variance in the original data that has been preserved, while minimizing the total squared reconstruction error  $\|TV^T - T_L V_L^T\|_2^2$  or  $\|X - X_L\|_2^2$ . In other words, the truncation of a matrix  $T$  using a truncated singular value decomposition in this way produces a matrix that is the nearest possible matrix of rank  $L$  to the original matrix.

In our implementation, we used the Randomized version of PCA [54] [55], which is faster than the original PCA because it limits computation to an approximated estimate of the singular vectors kept to actually perform the transformation. If we note

$$\begin{aligned} n_{\max} &= \max(n, p) \\ n_{\min} &= \min(n, p) \end{aligned}$$

the time complexity of Randomized PCA is  $O(n_{\max}^2 \cdot n_{\text{components}})$  instead of  $O(n_{\max}^2 \cdot n_{\min})$  for the exact PCA method. The memory footprint of Randomized PCA is also proportional to  $2 \cdot n_{\max} \cdot n_{\text{components}}$  instead of  $n_{\max}^2 \cdot n_{\min}$  for the exact method.

A decorrelation method called Whitening [56] [57], also known as Sphering, is applied after PCA to normalize the variances of the principal components, which has been shown to improve results in computer vision classification tasks [58]. It divides the principal components by their standard deviations, which yields an identity covariance matrix. Denoting the PCA rotated components by  $y_i$ , this means we compute

$$s_i = \frac{y_i}{\sqrt{\text{var}(y_i)}} \quad (16)$$

to get whitened components  $s_i$ . This is often useful if the classification model makes assumptions on the isotropy of the signal, which is the case for Support Vector Machines with the RBF kernel.

It may appear unnecessary to normalize the data before rotating and whitening it but we exemplify that this conclusion can be wrong: if one of the dimensions is orders of magnitude larger than the others, PCA will rotate the un-normalized data in the “wrong” direction, that is, in the direction of the dimension that has the greater (un-normalized) variance. After rotation, whitening will simply normalize the dimensions, but the data will be still rotated in the wrong direction.

## 2.7. Classifiers

As the final step of the pipeline, a classifier is used. Two classifiers were tested: K-Nearest-Neighbors (KNN), for comparison purposes, and Support Vector Machines (SVM), that is suitable to our problem because it is an inherently binary (two classes) classifier and it is widely used in large range of machine learning problems [59]. Two types of kernels were chosen for the SVM: linear and Gaussian Radial Basis Function (RBF) kernels. The linear kernel can be faster but the Gaussian kernel can find better separation hyperplanes when the number of features is not high [60], which is the case for the LBP pipelines.

The  $C$  parameter, common to all SVM kernels, was chosen during validation and it trades off misclassification of training examples against simplicity of the decision surface. Formally, the standard optimization problem for fitting a linear SVM is defined by [61]:

$$\min_{w,b,\delta} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \delta_i \quad (17)$$

$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \delta_i, \text{ for } i = 1, \dots, n$$

where  $n$  is the number of training samples,  $w$  and  $b$  are the coefficients that define the separation hyperplane, and  $\delta_i$  are non-negative slack variables that allow points to be on the wrong side of their ‘‘soft margin’’. A low  $C$  makes the decision surface smooth, while a high  $C$  aims at classifying all training examples correctly. If the data are separable, then for sufficiently large  $C$  the solution achieves the maximal margin separator; if not, the solution achieves the minimum overlap solution with largest margin.

In the Gaussian RBF kernel, the separating surface will be based on a combination of bell-shaped surfaces centered at each support vector. The width of each bell-shaped surface will be inversely proportional to hyper-parameter  $\gamma$ . More formally, from the dual optimization problem defined by

$$\max_a W(a) = \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i,j=1}^n y^{(i)} y^{(j)} a_i a_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{Subject to } a_i \geq 0, \text{ for } i = 1, \dots, n \quad (18)$$

$$\sum_{i=1}^n a_i y^{(i)} = 0$$

The dot product  $\langle x^{(i)}, x^{(j)} \rangle$  can be substituted by the RBF kernel, defined as

$$\langle x^{(i)}, x^{(j)} \rangle = e^{-\gamma |x^{(i)} - x^{(j)}|^2} \quad (19)$$

From (19), the value of the RBF kernel decreases with distance and ranges between zero (in the limit) and one (when  $x^{(i)} = x^{(j)}$ ). Hence, it has an interpretation of a similarity measure [62]. When  $\gamma$  is low, the expression above will be close to one even when  $x^{(i)}$  and  $x^{(j)}$  are far apart, meaning that a large number of support vectors influence the classification of a new sample. On the other hand, when  $\gamma$  is high, the expression will be close to one only when  $x^{(i)}$  and  $x^{(j)}$  are close, which characterizes overfitting. Another interpretation for  $\gamma$  is that it defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.

A question that may rise is why to use PCA before SVM if the latter is supposed to deal well with data in high dimensional space? It was shown in [63] that SVM is invariant to PCA and [64] showed that SVM performs better when using PCA as a feature extractor. Supported by these results, we used data dimensionality reduction through PCA to speed up SVM’s training time without losing accuracy.

## 2.8. Increasing Generalization through Dataset Augmentation

Dataset Augmentation is a technique that consists in artificially creating slightly modified samples from the original ones. Using them during training, it is expected that the classifier will become more robust against small variations that may be present in the data, forcing it to learn larger (and possibly more important) structures. It has been successfully used in computer vision benchmarks such as in [38], [65], and [66].

Our dataset augmentation implementation is similar to the one presented in [38]: from each image of the dataset five smaller images with 80% of each dimension of the original images are extracted: four patches from each corner and one at the center. For each patch, horizontal reflections are created. As a result, we obtain a dataset that is 10 times larger than the original one: 5 times are due to translations and 2 times are due to reflections.

In the training phase, the models of the pipeline are fitted using the samples of the augmented dataset. At test time, the input image is derived to ten translated and reflected patches followed by prediction for each of them. The prediction of the input image is made by averaging the individual predictions on the ten patches.

Other transformations, like image rotation, can be used to increase the dataset even more. However, we chose to use only translation and horizontal reflections in this work, mainly because of memory and training time limitations.

Due to the large amount of time to train the pipelines that use the artificial augmented dataset, model selection was first made only in the original dataset and considering a wide range of parameters, and then a second validation is performed together with the augmented data using only a subset of parameters from the previous validation.

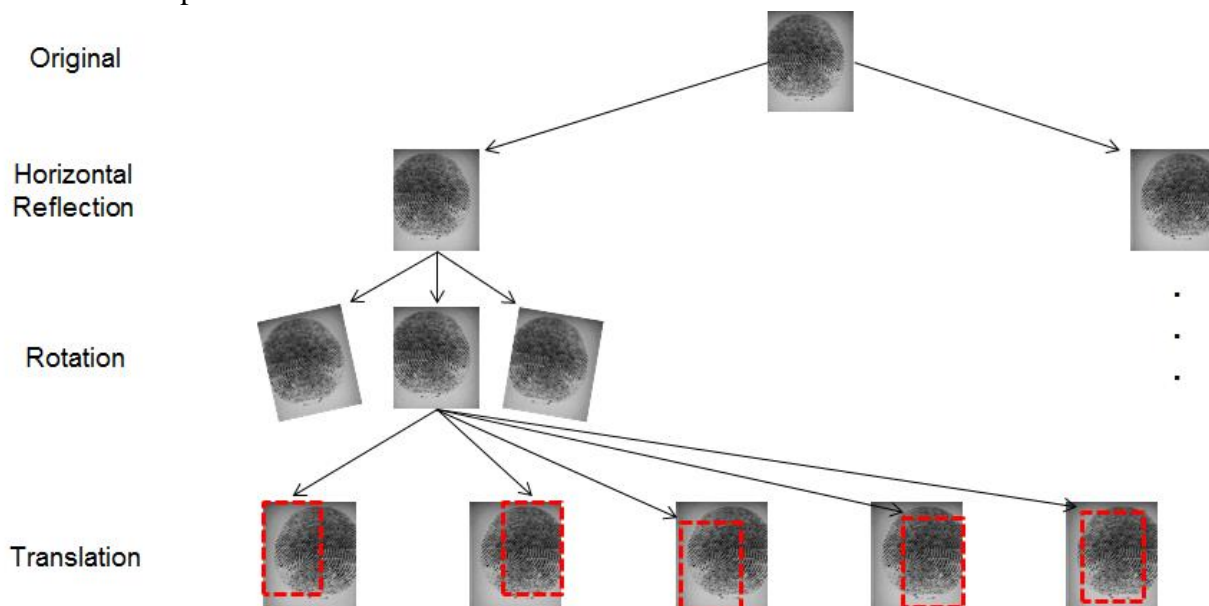


Figure 14 – Illustration of three types of transformations for dataset augmentation: Horizontal Reflections, Rotations, and Translations.

## 2.9. Performance Metrics

The classification results were evaluated by the Average Classification Error (ACE), which is the standard metric for evaluation in the LivDet competitions. It is defined as

$$ACE = (FPR + FNR)/2 \quad (20)$$

where *FPR* (False Positive Rate) is the percentage of misclassified live fingerprints and *FNR* (False Negative Rate) is the percentage of misclassified fake fingerprints.

## 2.10. Implementation Details

The algorithms were implemented in Python and most of the code uses build-in functions from Numpy, Scipy, Scikit-Image and Scikit-Learn packages, except for the Convolutional Networks, for which we used an efficient package from [67], and the Cross-Validation/Grid-Search algorithm, for which we wrote our own code using Numpy. NumPy is a general-purpose array-processing package designed to efficiently manipulate large multi-dimensional arrays of arbitrary records. Although Numpy is a python extension, its functions are written in C. Thus any algorithm that can be expressed primarily as operations on arrays and matrices can run almost as quickly as the equivalent C code.

### 2.10.1. Cross-Validation/Grid-Search Algorithm

We wrote an improved Cross-Validation/Grid-Search algorithm for choosing the best combination of hyper-parameters, in which each element of pipeline is computed/trained only when its training data is changed (the term “element” refers to operations such as preprocessing, feature extraction, dimensionality reduction or classification). This modification speeded-up the validation phase in approximately 10 times, although the gain can greatly vary as it depends on the computational cost of each element of the pipeline and the number of hyper-parameters chosen. The pseudo-code for the algorithm is presented in two parts: initialization of variables and call of the recursive function (Figure 15) and the definition of the recursive function (Figure 16).

<p>Begin</p> <p>Set <i>Lc</i> as a list of all combinations of the parameters for each element of the pipeline.</p> <p>Set <i>ce</i> with the first element of the pipeline.</p> <p>Set <i>dtr</i> with the training dataset.</p> <p>Call Function <i>run_element(Lc, ce, dtr)</i> and set <i>Lr</i> with the list of tuples (<i>scores, parameters</i>) returned by the function</p> <p>Order the list <i>Lr</i> by <i>score</i>. The <i>parameters</i> set with the lowest score are the best parameter's combination.</p> <p>End</p>
---

Figure 15 – Pseudo-code that describes the variable's initialization and the call of the recursive function in the Cross-Validation/Grid-Search algorithm

```

Begin Function run_element (Lc, ce, dtr, dval (optional)):
  For each parameter combination (called as cp) of element ce in the list of parameter's combinations Lc:
    If ce is the first element of the pipeline:
      Split the dtr dataset in the training and validation sets based on the cross-validation scheme used (e.g., k-fold) and store them as a list of tuples in dsets.
    Else:
      Set dsets with a list that has a single tuple (dtr, dval).
    For each training and validation set (called as cdtr and cdval, respectively) tuple in dsets:
      Transform (or fit and transform if the element supports both operations) cdtr and cdval datasets using element ce initialized with parameters cp.
    If ce is the last element of the pipeline:
      Predict the samples of cdval and computes the score.
      Append the tuple (score, cp) to the list Ls.
    Else:
      Set ce with the next element of the pipeline.
      Call run_element(Lc, ce, cdtr, cdval) and append the returned list to Ls.

  Return Ls
End Function

```

Figure 16 – Definition of the recursive function in the Cross-Validation/Grid-Search algorithm

Figure 17 illustrates a graph showing how the elements and parameters are reused in our improved grid search algorithm. For instance, the rounded *Node 3* represents an element *Elem 2* trained with the transformed data from the parent *Node 1*, with parameters'  $p_2$  set with values  $v_2$ . The algorithm starts by transforming the incoming data (illustrated as green square in the top of the figure) by *Node 1* (that represents *Elem 1* with parameters'  $p_1$  set with values  $v_1$ ) followed by the transformation by *Node 3* and so on until the last node of the branch, *Node i-1*, is reached and the score is computed. Next, the adjacent node, *Node i*, can be computed using the data transformed from its parent node. Note that a new branch is started only when the last node of the previous branch is reached.



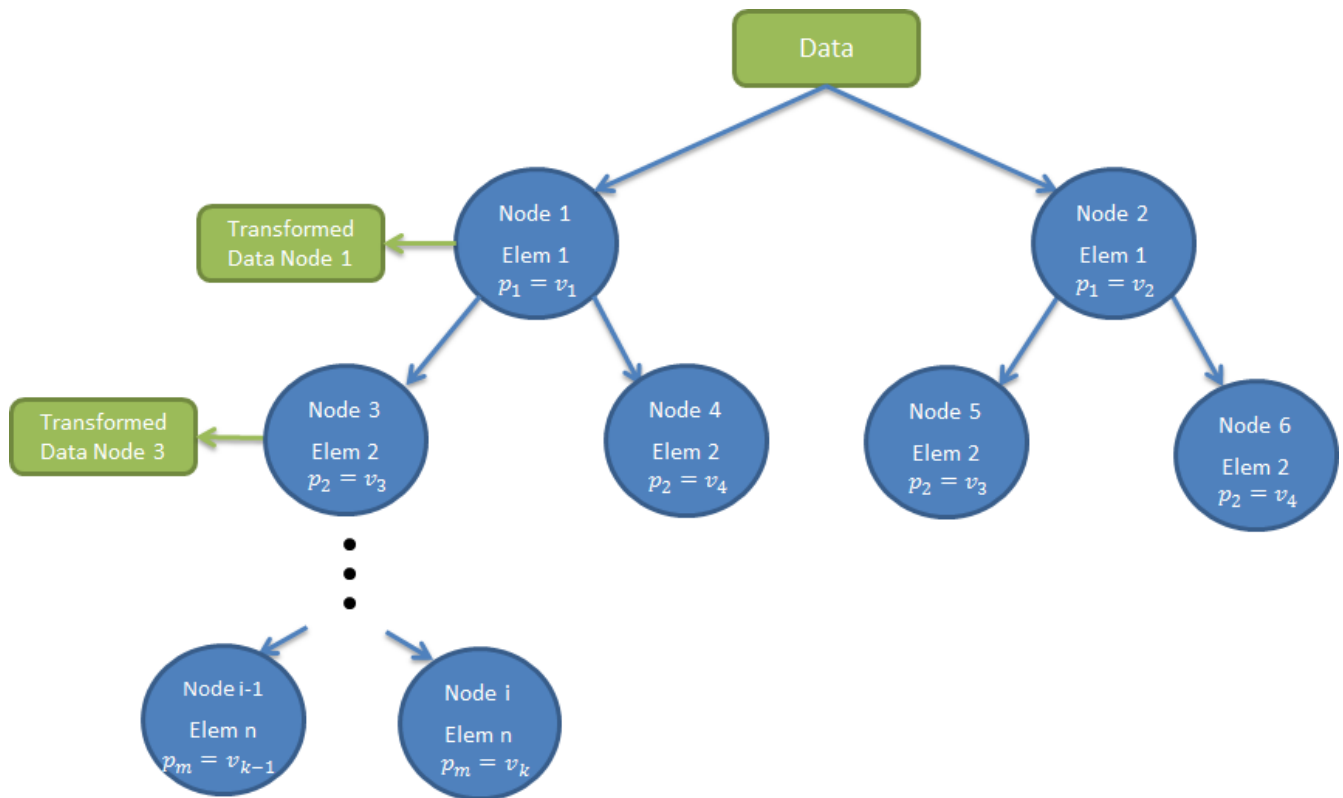


Figure 17 – A graph showing how the elements and parameters are reused in our improved grid search algorithm.

The search for the best hyper-parameters was made using 5x2-Fold [68] cross-validation scheme and it works as follows:

- 1- Divide the training dataset into 2 blocks, A and B.
- 2- Train on block A and evaluate on B.
- 3- Train on B and evaluate on A.
- 4- Repeat the last 3 steps 5 times, but choosing a different split location to create blocks A and B.
- 5- Compute the average score for all 10 (5x2) evaluations.

Although 10-Fold cross-validation is a common choice, we noticed in previous experiments that both methods yield similar choices for the hyper-parameters but 5x2-Fold is faster as it uses 50% of the validation dataset for training instead of 90% used by the 10-Fold method (assuming that training is slower than classifying and that both methods iterate 10 times over the validation dataset).

As explained before, in the cross-validation schemes like the 5x2 Fold, the dataset is divided in training and validation set multiple times, assuring that each sample will belong to the validation set at least once. These interactions are computationally expensive, as a new model need to be trained every time a different partition of the dataset is used. In our pipelines, the pre-processing and feature extraction phases do not need training, that is, the samples can be processed independently and in parallel. This resulted in another implementation improvement in Grid Search algorithm: the dataset is transformed by these two initial phases before being split in the cross-validation iterations, which

avoids repeated transformations that would have to be performed if the standard cross-validation was used.

### **2.10.2. Using Fast Computers in the Cloud**

An important aspect of this work is that the algorithms were run on cloud service computers, where the user can rent virtual computers and pay only for the hours that the machines are running. Among other advantages of this kind of service, they offer ready-to-use instances (with Machine Learning packages like Scikit-Learn already installed, for example), high availability (commonly greater than 99.9%), and computational optimized instances, commonly used for High performance front-end fleets and web-servers, on-demand batch processing, distributed analytics, and high performance science and engineering applications, batch processing, and video encoding. To train the algorithms, we used the fastest Amazon EC2 HPC instance available, with 32 cores and 60 GB of RAM, which allowed us to run dataset augmented experiments and search for a wide range of parameters in a few hours – otherwise it would take weeks to perform them.

### 3. Experiments

Table 2 lists the pipelines used in the experiments. The preprocessing step is omitted but it was executed in all pipelines. The list of hyper-parameters and ranges searched in the validation phase are shown in Table 3.

Pipeline	Description
KNN	Images reduced to 50% or 25% of their original size and then fed into a k-Nearest-Neighbor classifier. This pipeline is used only for comparison purposes.
PCA+KNN	Data dimensionality is reduced using PCA and then fed into a KNN classifier.
CN+PCA+LinearSVM	Features are extracted using Convolutional Networks. The feature vector is reduced using PCA and then fed into a SVM classifier using linear kernel.
CN+PCA+GaussianSVM	Features are extracted using Convolutional Networks. The feature vector is reduced using PCA and then fed into a SVM classifier using (Gaussian) RBF kernel.
LBP+PCA+GaussianSVM	Features are extracted using LBP. The feature vector is reduced using PCA and then fed into a SVM classifier with (Gaussian) RBF kernel.
AUG+LBP+PCA+GaussianSVM	Dataset is artificially augmented, and the pipeline follows as in the LBP+PCA+GaussianSVM pipeline.
AUG+CN+PCA+LinearSVM	Dataset is artificially augmented, and the pipeline follows as in the CN+PCA+LinearSVM pipeline.
AUG+CN+PCA+GaussianSVM	Dataset is artificially augmented, and the pipeline follows as in the CN+PCA+GaussianSVM pipeline.

Table 2 - Summary of the pipelines evaluated in this work.

Pipeline's Element	Hyper-parameter	Range
Preprocessing	Image Reduction Ratio	12,5%, 25%, 50%, 100% (original size)
	High-pass filtering	Apply or Not
	Low-pass filtering	Apply or Not
	Region of Interest	Apply or Not
	Contrast Equalization	Apply or Not
Convolutional Networks	Number of Layers	1, 2, 3, 4, 5
	Number of Filters (in each layer)	32, 64, ..., 2048
	Divisive Normalization	Apply or Not
	Filter Size for Divisive Normalization	5x5, 7x7, ..., 15x15
	Filter Size for Convolution	5x5, 7x7, ..., 15x15
	Filter Size for Pooling	3x3, 5x5, 7x7, 9x9
	Stride (reduction factor)	2, 3, ..., 7
LBP	Coding	Non-Uniform (standard) or Uniform
	Number of Image's Divisions	1x1 (no division), 3x3, 5x5, 7x7
PCA	Number of Components	30, 100, 300, 500, 800, 1000, 1300
GaussianSVM	Regularization Parameter C	0.1, 1, ..., $10^5$
	Kernel coefficient $\gamma$	$10^{-7}$ , ..., $10^{-1}$
LinearSVM	Regularization Parameter C	$10^{-5}$ , $10^{-4}$ , ..., $10^5$
KNN	Number of Neighbors	1, 3, 9, 15
	Metric Method	Distance or Uniform

Table 3 – List of hyper-parameters and ranges searched in the validation phase

For CN, max pooling is used, supported by [69], [70], and [71] that show its superiority over average pooling. We noticed that for our task, subtractive normalization seems to amplify noise, which decreases classification performance. Therefore, we choose to use only divisive normalization.

Apart from using PCA with Whitening in the dimensionality reduction phase, we performed experiments using PCA without Whitening, Linear Discriminant Analysis (LDA), and Independent Component Analysis (ICA) in 4 of the 11 datasets. In general, they presented a decrease of 0.5-2% on the overall validation accuracy. Additionally, [72] and [73] found that the choice for LDA, ICA or PCA depends on the nature of the task. Based on these works and the fact that PCA+Whitening showed the best preliminary results, we decided to use only this technique in the experiments.

Due to the increased amount of time to train the models using augmented datasets, hyper-parameters' selection was first made in the original dataset using a wide range of hyper-parameters values, and then a second validation was performed in the augmented dataset, but using a subset of hyper-parameters from the first validation. When using augmented datasets, it is important not to shuffle the samples. That is, images derived from the same image should not be separated in training and validation sets because classifier's generalization capabilities will decrease if similar data occurs in training and validation sets.

We would like to verify how a classifier would perform when unseen samples acquired from spoofing materials and individuals during training are presented at test time. For that, Cross-dataset experiments were performed, which consist in training a classifier using the dataset of one year of the LivDet competition and testing using the dataset of another year of the competition but using the same sensor type. For instance, a cross-dataset experiment would consist in training a classifier using LivDet2011 dataset for sensor Biometrika and testing it using LivDet2013 dataset for sensor Biometrika.

Since images from the same sensor are similar, it is expected that training independent classifiers for each sensor will help the classifier to find better separation hyperplanes. However, in order to test the hypothesis that the images share common characteristics for distinguishing fake fingerprints from real ones, that is, the important features for classification are independent from the acquisition device, Cross-device experiments were conducted, in which a classifier is trained with a dataset acquired with one type of sensor but tested with samples acquired with another sensor type. Additionally, we trained a single classifier with images from all sensors, excluding the Swipe sensor that produces images that are too different from the others.

In summary, these tests should reflect how well the classifier is able to learn relevant characteristics that distinguish real from fake fingerprints when samples acquired from different environments are presented.

## 4. Results

The average error in each testing dataset is shown on Table 4. The state-of-the-art (first column) results for LivDet 2009, 2011, and 2013 datasets were taken from [30], [18] and [32], respectively. The validation errors are shown on Appendix A (Table 8) and the parameters used and scores in the validation phase can be found in <http://adessowiki.fee.unicamp.br/adesso/wiki/Demo/fingerprint/view/>.

Technique		State-of-the-art	Aug LBP PCA Gaussian-SVM	LBP PCA Gaussian-SVM	Aug CN PCA Gaussian-SVM	Aug CN PCA Linear-SVM	CN PCA Gaussian-SVM	CN PCA Linear-SVM	PCA KNN	KNN
LivDet 2013	Crossmatch	31.2	49.45	49.87	<b>3.29</b>	3.64	5.2	3.69	17.27	28.8
	Swipe	14.07	<b>3.34</b>	4.02	7.67	6.87	5.97	5.13	29.12	40.76
	Italdata	3.5	2.3	55.45	2.45	<b>0.45</b>	47.65	49.95	46.3	32.6
	Biometrika	4.7	1.7	25.65	<b>0.8</b>	1.0	2.7	2.7	46.15	50.8
LivDet 2011	Italdata	14.8	12.34	23.68	9.27	15.17	<b>5.09</b>	5.35	34.35	41.99
	Biometrika	<b>7.3</b>	8.85	8.2	8.25	11.75	9.9	13	26	45.45
	Digital	2.5	4.15	3.85	3.65	4.25	<b>1.9</b>	2	11.75	23.3
	Sagem	5.3	7.54	5.56	<b>4.64</b>	6.74	7.86	8.3	14.5	42.54
Livdet 2009	Biometrika	18.1	10.44	50	9.23	6.48	9.49	<b>8.98</b>	46.55	34.15
	Crossmatch	15.2	3.65	6.81	<b>1.78</b>	2.68	3.76	3.83	14.85	14.67
	Identix	10.5	2.64	0.95	0.8	<b>0.68</b>	1.68	1.48	9.65	10.69
Griaule 2013		n/e	<b>0.33</b>	1.83	3.38	n/e	0.34	n/e	n/e	n/e
Average		11.56	9.67	21.28	<b>4.71</b>	5.43	9.20	9.49	26.95	33.25

Table 4 – Average Error Rate on testing datasets. “n/e” stands for “not executed”

The LBP+PCA+SVM pipeline seems to be highly prone to overfitting, since it has very low cross-validation errors (close to 0%) whereas large error rates in the testing datasets of Crossmatch 2013, Italdata 2013, Biometrika 2013, Italdata 2011 and Biometrika 2009. However, when dataset augmentation is used (AUG+LBP+PCA+SVM pipeline), the cross-validation error increases but the testing error decreases, except for Crossmatch 2013, which will be discussed latter. This is a good indication that dataset augmentation can be used to prevent overfitting.

When compared to the state-of-the-art technique in LivDet2011, a multi-scale LBP presented in [18], our LBP technique achieves better performance (9.67% error rate against 11.56%). The best operator (uniform or non-uniform/original) and number of tiles depend on the dataset.

Overfitting seems not to be a problem when using Convolutional Networks, except for the Italdata 2013 dataset. Overall, CN without the dataset augmentation have a similar performance to LBP with dataset augmentation. When using augmented datasets with convolutional networks, we achieved a test error rate of 4.75% (averaged from all datasets), which represents a reduction of 58% when compared to the best previously published results (11.56% error, on average).

Support Vector Machines with Gaussian kernel performed slightly better than the Linear Kernel (4.71% vs 5.43% on augmented datasets and 9.20% vs 9.49% on non-augmented datasets), but the former needs an extra hyper-parameter ( $\gamma$ ) to be tuned.

The best number of layers in the convolutional networks depends on the dataset: it varies from two to five layers. The fact that one layer networks were not selected confirms that the deep architectures perform better on the task than the shallow ones. On overall, the best convolution shapes are 9x9 for the first layers and 5x5 for the last layers. The best pooling shapes are 7x7 for the first layers and 5x5 for the last layers. The best quantity of filters was 256 or 512 for the first layers and 1024 or 2048 for the last layers. Unfortunately, we could not find a relation between architectures and dataset characteristics, such as image size and foreground/background ratio, that explain the choices for the best parameters. Surprisingly, divisive normalization did not improved results in the validation phase, despite its performance gain in natural images tasks reported in papers such as [69].

On average, the PCA models selected in the validation phase reduced the input vectors to 20% of their original dimensions, which represents an explained variance close to 100%. This is an indication that the vectors extracted using either CN or LBP still contain redundant information and reducing the dimensions using PCA can be advantageous. This statement is confirmed by empirical results: pipelines that use PCA have greater accuracy rates than the ones that do not use it.

For the majority of datasets and models, preprocessing operations (contrast equalization, ROI, etc) did not improved accuracy. For contrast equalization, this is not surprising, since both LBP and CN are illumination invariant, that is, each pixel is compared only with its local neighbors. Regarding ROI extraction, the backgrounds are mostly composed of white pixels or by a regular structure, like the trapezoidal background shape in the case of the Digital 4000B sensor, which results in components in the final extracted vector that have low variances and are probably discarded by the dimensionality reduction and the SVM classifier during training. In addition, the histogram extraction in the LBP pipeline and the pooling/sub-sampling operation in the CN offer translation invariance, so objects in the image center would not help much.

The hypothesis that the relevant information for liveness detection lies either in the low-frequency or in the high-frequency components of the image was not confirmed. Both low-pass and high-pass filtering decreased accuracy during validation, which suggests that the structures that differentiate false from real fingerprints do not have exclusively low or high frequency components.

A 50% image size reduction improved validation accuracy in some datasets (Italdata 2013 for both LBP and CN pipelines, Identix 2009 for Convets, Italdata 2011, Biometrika 2011, and Crossmatch 2009 for LBP). Reduction rate of 25% only helped for the Identix 2009 dataset in the LBP pipeline. Based on these differences, we conclude that there is not an optimal image size for classification; it

depends not only on the sensor type but also on the dataset and the transformations used. For instance, the best model in the LBP pipeline for Biometrika-2013 uses images in their original size while the best model for Biometrika-2011 uses reduction ratio of 50%, despite that both datasets were acquired using the same sensor type and resolution.

#### 4.1. Cross-dataset and Cross-device Experiments

As explained in chapter 0, we ran three types of experiments involving cross-training:

- 1- Cross-dataset: train with one dataset but test with another using the same sensor type;
- 2- Cross-device: train with one type of sensor but test with another type;
- 3- All-together: train and test a single classifier using all datasets, except for Swipe-2013 whose images are too different from the rest.

We chose to use only datasets Biometrika and Italdata of years of 2011 and 2013 of the LivDet competition, since executing all possible dataset combinations would be impractical. The experiments used LBP or CN as feature extractors and SVM with Gaussian kernel as classifier.

The error rates for the Cross-dataset experiments, shown in Table 5, vary from 10% to 50% and the training errors (not shown) were all very low (approx. 0%). The pipelines that use dataset augmentation have lower error in most of the cases, showing once again the benefits of the technique. However, there is a significant drop in performance in all pipelines when compared with error rates obtained from the standard training datasets, indicating that the classifiers were not able to generalize well to fake fingerprints created using unseen spoof techniques.

Table 5 - Cross-dataset error rates

<b>Train Dataset</b>	<b>Test Dataset</b>	<b>Aug LBP PCA Gaussian-SVM</b>	<b>LBP PCA Gaussian-SVM</b>	<b>Aug CN PCA Gaussian-SVM</b>	<b>CN PCA Gaussian-SVM</b>
<b>Biometrika 2011</b>	<b>Biometrika 2013</b>	<b>16.55</b>	20.3	20.4	26.05
<b>Biometrika 2013</b>	<b>Biometrika 2011</b>	<b>47.95</b>	48.55	48.0	48.45
<b>Italdata 2011</b>	<b>Italdata 2013</b>	<b>10.6</b>	13.0	21.0	50.0
<b>Italdata 2013</b>	<b>Italdata 2011</b>	46.08	<b>35.17</b>	46.82	46.03

In the Cross-device experiments, the testing error rates, shown in Table 6, range between 45-50% and the training error rates are all close to 0% (not shown), indicating overfitting. Similarly, [18] already reported that their multi-resolution LBP technique fail to learn good features when different sensors are used for training and testing.



Table 6 - Cross-device error rates

Train Dataset	Test Dataset	Aug LBP PCA Gaussian-SVM	LBP PCA Gaussian-SVM	Aug CN PCA Gaussian-SVM	CN PCA Gaussian-SVM
Biometrika 2013	Italdata 2013	43.7	50.0	47.9	45.3
Italdata 2013	Biometrika 2013	48.4	50.0	48.95	52.6

For the third experiment (“All Together”), it can be seen from the results shown in Figure 18 that training one classifier using all datasets yields error rates around 10% for both LBP and CN pipelines that do not use dataset augmentation. The error rate in training (not shown in the figure), is around 6%. Dataset augmentation was not used due to the larger training time, but one should expect lower errors.

From these results, we can conclude that the effort to design a liveness detection system can be considerably reduced if all datasets are used together, as the hyper-parameter fine tuning needs to be made for only one classifier.

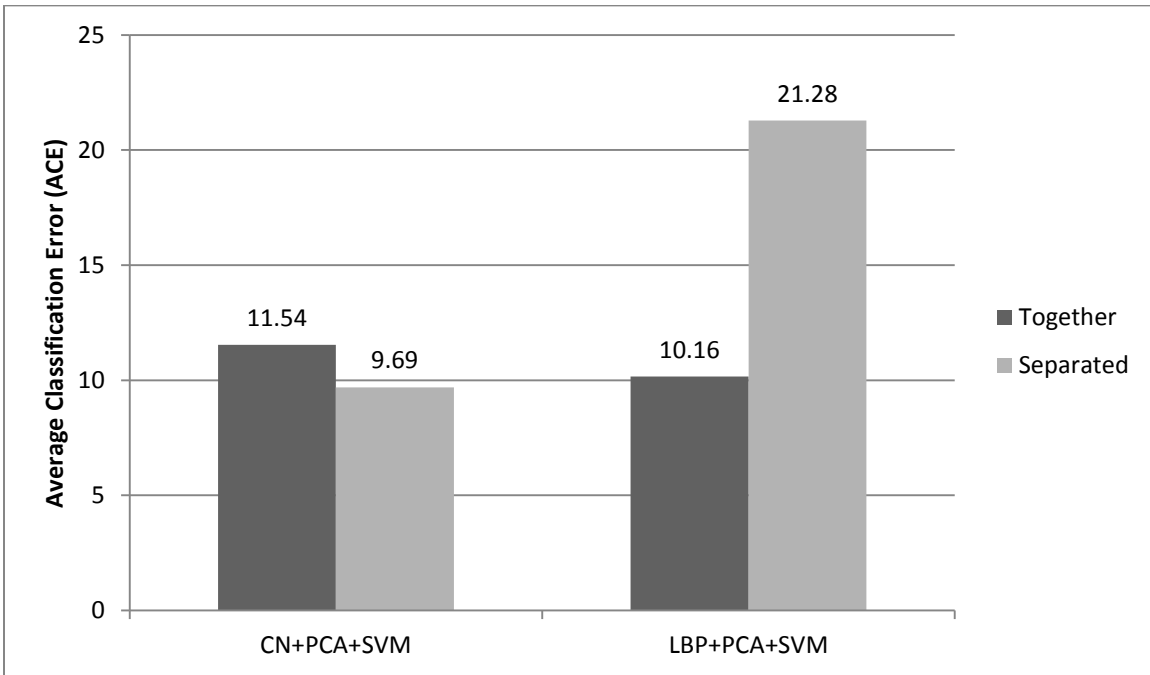


Figure 18 – Error rates in the testing when training one classifier using all datasets (Together) vs training one classifier for each dataset (Separated)

#### 4.2. Crossmatch 2013 dataset and Dataset Visualization

The results for Crossmatch 2013 dataset using LBP present error rates close to zero at validation time and around 50% at test time, even when using augmented datasets. It can be noticed from the results of the LivDet 2013 competition that this dataset is particularly difficult to generalize, since nine of the

eleven participants presented error rates greater than 45%. CN performs very well (3.28%), which suggests that the problem occurs mostly when extracting features with LBP.

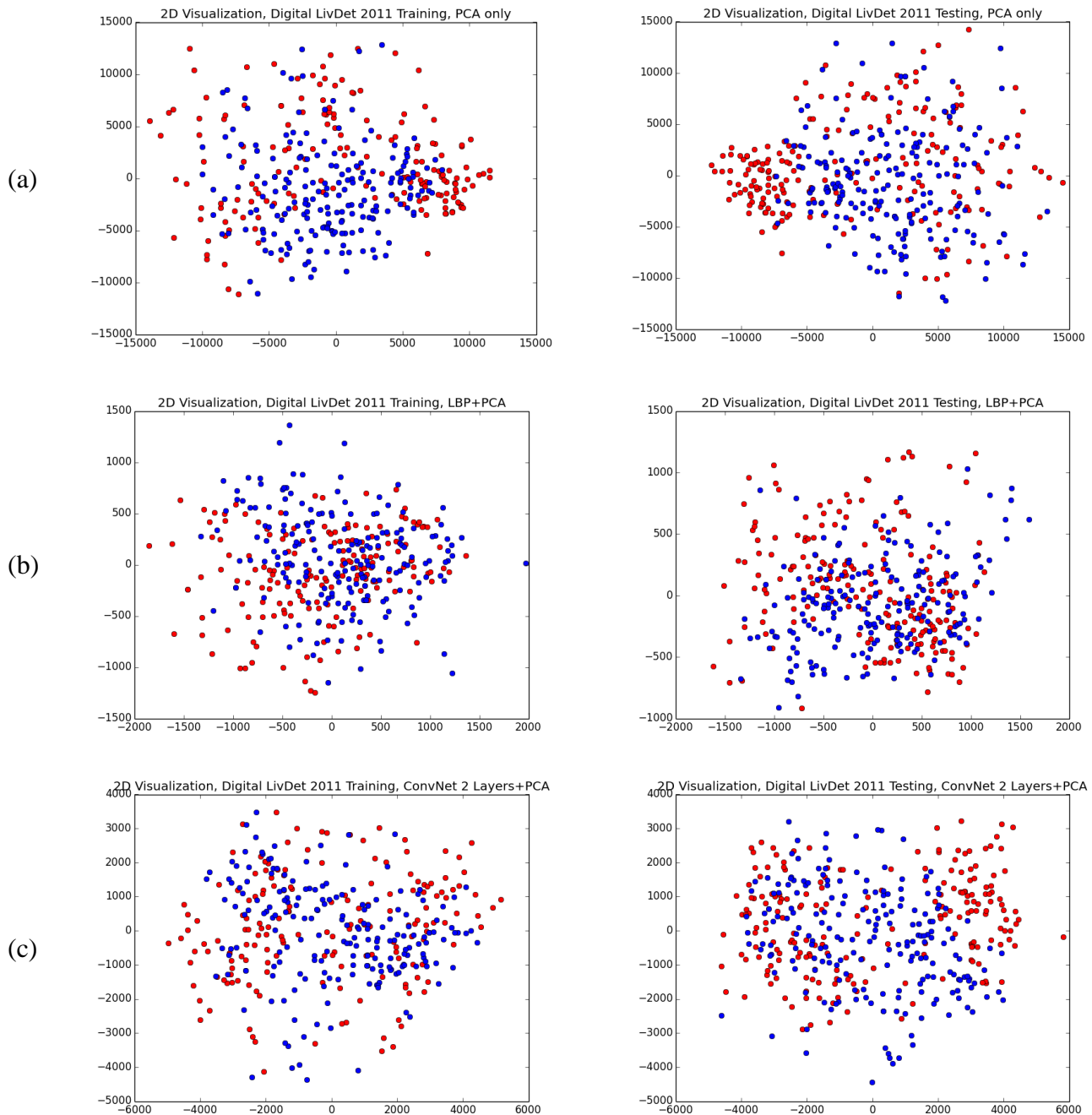


Figure 19 - 2D visualization of the Digital-2011 training (left) and testing (right) datasets. The rows (a), (b) and (c) represents dimensionality reduction using PCA, LBP+PCA and CN+PCA pipelines, respectively.

To help better understanding of how the feature extractors contribute to the classification system, and especially for the problem with the Crossmatch 2013 dataset, we reduced the images of the Biometrika-2009 dataset to 2 dimensions using three different pipelines: PCA, LBP+PCA and CN+PCA. The transformed datasets from each pipeline are shown in Figure 19. It can be seen that there is no clear separation between the fake and real fingerprints (represented by red and blue dots, respectively) in any of the pipelines.

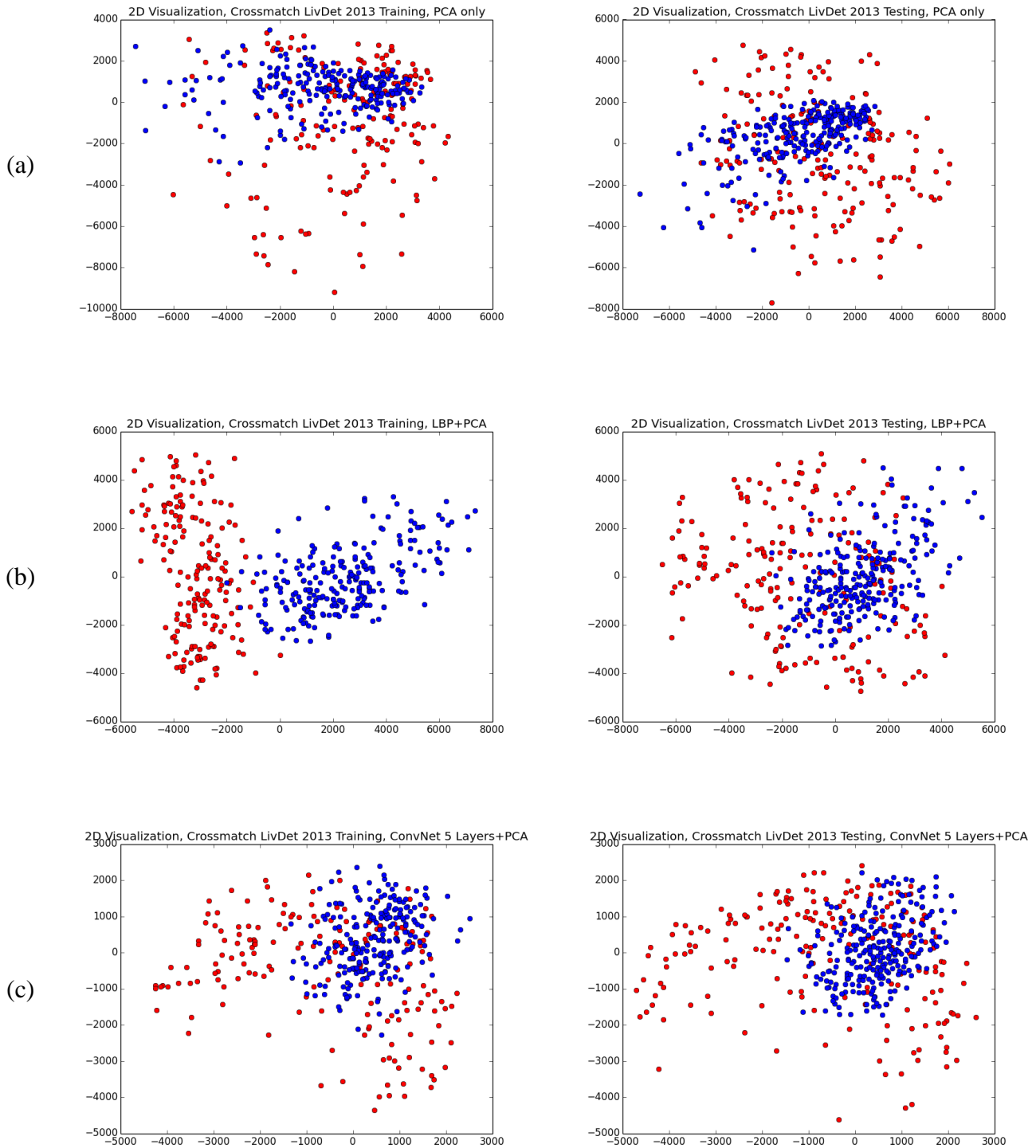


Figure 20 - 2D visualization of the Crossmatch-2013 training (left) and testing (right) datasets. The rows (a), (b) and (c) represents dimensionality reduction using PCA, LBP+PCA and CN+PCA pipelines, respectively.

The same steps were repeated for the Crossmatch 2013 training and testing datasets and the results are shown in Figure 20. The training data becomes more separable when using one of the feature extractors. However, we can see that there is a very low correspondence between training and

testing samples when using the LBP+PCA pipeline, especially for the fake (red dots) samples. In addition, there is a (rare) clear separation in the training dataset when the features are extracted using this pipeline, which can explain why the training errors are so low (~0%) and the testing error are so high (~50%) for that particular dataset.

The results on the Crossmatch 2013 testing dataset are improved to error rates around 20-30% when the images of the training set are filtered with low-pass, high-pass or adding Gaussian noise. One interpretation for these results is that, when no transformation is applied, the LBP filtering highlights some (still unknown) patterns that occurs only in the training dataset and that makes the false and real fingerprint very distinguishable. However, those patterns do not occur in the testing set, and the classifier fails to differentiate false from real fingerprint images. When the data is transformed by smoothing, sharpening, or adding noise, those patterns no longer occur in the training images, and the classifier is able to learn other (and possible more relevant) features.

When a single classifier trained all datasets is used (section 4.1), the testing error rate for the Crossmatch 2013 dataset is still around 50%, which indicates that the classifier was not able to generalize even when more training samples were used.

The problem was further investigated by trying to find which extracted features by the LBP+histogram are the most relevant for classification. For that, a decision tree classifier was used as it computes the feature's importance during training. We noticed that 3 out of 10 codes (0, 1 and 9) of the uniform coding and 30 out of 255 codes of the non-uniform coding are the main responsible for classifier's predictions. The visual inspection of these patterns in some sample images shows that they in fact occur much more frequently in the real fingerprints than in the fake ones. However, there is no visual similarity among the neighboring regions that forms these codes, and thus they provide no insight into the structure that can differentiate real from fake fingerprints.

### ***4.3. Average Processing Times and Memory Usage***

In real applications, a good fingerprint liveness detection system must be able to classify the images in a short amount of time and should have a small memory footprint when the algorithms are required to run in embedded devices. Table 7 shows the average processing times and memory usage for some pipelines to classify a single image on a single core computer (1.8 GHz). It is worth mentioning that those times can be decreased, since our code for the Convolutional Networks is not optimized and there are fast hardware implementations [74] [44]. The feature extraction phase (using either LBP or CN) represents 70% of the total processing time, on average, and PCA's rotation table represents more than 95% of the memory footprint. Removing the PCA from the pipeline can be an alternative for devices that have memory limitations, but an increase of 0.5 to 2% in the error rate should be expected. The training time of each model for LBP and CN are around 20 minutes and 1.5 hours, respectively.

Table 7 - Average processing time per sample in a single core CPU and memory usage.

	LBP PCA GaussianSVM	AUG LBP PCA GaussianSVM	CN PCA GaussianSVM	AUG CN PCA GaussianSVM
Avg. classification time (ms)	<b>60</b>	520	570	2540
Memory Footprint (MB)	52	<b>45</b>	410	385
Storage space required (MB)	27	<b>24</b>	212	190

## 5. Conclusion

A wide variety of models were implemented and compared for fingerprint liveness detection, from a simple k-Nearest Neighbor classifier to more complex pipelines that use feature extractor such as LBP and Convolutional Networks with dataset augmentation.

The Convolutional Networks presented the best performance. However, they are slower to train and more complex to design than LBP. LBP with dataset augmentation gave slightly better results than the state-of-the-art algorithms: 9.67% error against 11.56%, on average and the Convolution Networks achieved an average classification error of 4.71%. When a single classifier is trained using all datasets, both techniques perform well. For the LBP pipeline in particular, the error rate is half of the averaged error rate obtained with individual classifiers. This suggests that the effort to design a liveness detection system can be significantly reduced if different datasets are combined during training of a single classifier. However, there is still room for improvement, as the models suffer from a significant drop in accuracy in cross-dataset and cross-device experiments, indicating that they were not able to generalize when samples that were acquired from different sensor types and unseen spoof techniques are presented during testing.

Preprocessing operations such as Region of Interest extraction and Contrast Equalization did not help to improve accuracy, mainly because the feature extractors already offer some robustness against illumination and translation variances. PCA and Whitening are necessary, since the data has redundant dimensions after the feature extraction phase.

Dataset augmentation demonstrated to be one of the main contributors to increase accuracy and it is simple to implement. We claim that the method should always be considered if one has enough computational power.

We believe that the main contributions for the low error rates obtained were the large models and datasets used, like images in their original sizes, augmented datasets, and large number of layers and filters in the convolutional networks. With faster computers, we could execute a large number of experiments due to lower training/validation iteration times. The emerging high performance cloud computing platforms make the building of increasingly large experiments affordable by renting ready-to-run virtual computer infrastructure.

### 5.1. Future Work

Further experiments will include learning the filters' weights of the convolutional networks, as [69] reported that a better performance is achieved when the network is trained.

Given the promising results provided by the dataset augmentation, more types of image transformations should be included, such as artificially creating images with uneven illumination and with random noise. In particular, we want to know the limits of the technique: how many times can the dataset be artificially augmented with an improvement in performance?

A combination of convolutional networks and LBP can provide an effective scheme. The former offers the ability to represent more complex structures due to the deep architecture while the second is able to capture texture patterns, which seems to be important to the fingerprint liveness detection.

However, there is still no scheme to merge them, except for the trivial ensemble of two or more classifiers trained with the feature extractors separately, which also can be addressed in a future work.

## References

- [1] A. Wiehe, T. Søndrol, Olsen, O. K. and F. Skarderud, "Attacking fingerprint sensors," *Technical report, NISLAB Authentication Laboratory, Gjøvik University College*, 2004.
- [2] Y. Chen and A. Jain, "Fingerprint deformation for spoof detection," *Proc. IEEE Biometric Symposium*, pp. 19-21, 2005.
- [3] B. Tan and S. Schuckers, "Comparison of ridge-and intensity-based perspiration liveness detection methods in fingerprint scanners," *Defense and Security Symposium International Society for Optics and Photonics*, 2006.
- [4] P. Coli, G. L. Marcialis and F. Roli, "Fingerprint silicon replicas: static and dynamic features for vitality detection using an optical capture device," *International Journal of Image and Graphics* 8.04, pp. 495-512, 2008.
- [5] P. Lapsley, J. Lee, D. Pare and N. Hoffman, "Anti-fraud biometric scanner that accurately detects blood flow". US Patent 5,737,439, 1998.
- [6] M. R. Arneson, B. L. Blan, H. M. Carim and D. W. Osten, "Biometric, personal authentication system". U.S. Patent 5,719,950, 1998.
- [7] D. Baldisserra, A. Franco, D. Maio and D. Maltoni, "Fake fingerprint detection by odor analysis," in *Advances in Biometrics*, Berlin Heidelberg, Springer, 2005, pp. 265-272.
- [8] R. Derakhshani, S. Schuckers, L. Hornak and L. O’Gorman, "Determination of vitality from a non-invasive biomedical measurement for use in fingerprint scanners," *Pattern Recognition*, vol. 36, no. 2, pp. 383-396, 2003.
- [9] S. Parthasaradhi, R. Derakhshani, L. Hornak and S. Schuckers, "Time-series detection of perspiration as a liveness test in fingerprint scanners," *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, vol. 35, no. 3, pp. 335-343, 2005.
- [10] S. Schuckers and A. Abhyankar, "Detecting liveness in fingerprint scanners using wavelets: results of the test dataset," *Proceedings of BioAW*, pp. 100-110, 2004.
- [11] A. Antonelli, R. Cappelli, D. Maio and D. Maltoni, "Fake Finger Detection by Skin Distortion Analysis," *Information Forensics and Security*, pp. 360-373, 2006.
- [12] O. G. Martinsen, S. Clausen, J. B. Nysæther and S. Grimnes, "Utilizing characteristic electrical properties of the epidermal skin layers to detect fake fingers in biometric fingerprint systems—A pilot study," *Biomedical Engineering, IEEE Transactions on*, vol. 5, no. 54, pp. 891-894, 2007.
- [13] J. Galbally, F. Alonso-Fernandez, J. Fierrez and J. Ortega-Garcia, "A high performance fingerprint liveness detection method based on quality related features," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 311-321, 2012.
- [14] A. K. Jain, Y. Chen and M. Demirku, "Pores and ridges: high-resolution fingerprint matching using level 3 features," *Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 15-27, 2007.
- [15] P. Coli, G. S. Marcialis and F. Roli, "Power spectrum-based fingerprint vitality detection," *Proceedings of IEEE Workshop on Automatic Identification Advanced Technologies*, pp. 169-173, 2007.
- [16] Y. S. Moon, J. S. Chen, K. C. Chan, K. So and K. C. Woo, "Wavelet based fingerprint liveness detection," *Electronics Letters*, vol. 20, no. 41, pp. 1112-1113, 2005.
- [17] S. Nikam and S. Agarwal, "Local binary pattern and wavelet-based spoof fingerprint detection," *Int. J. Biometrics*, vol. 1, no. 2, pp. 141-159, 2008.



- [18] X. Jia, X. Yang, K. Cao, Y. Zang, N. Zhang, R. Dai and J. Tian, "Multi-scale Local Binary Pattern with Filters for Spoof Fingerprint Detection," *Information Sciences*, vol. 268, pp. 91-102, 2013.
- [19] T. Ojala, M. Pietikäinen and T. Mäenpää, "Multiresolution gray scale and rotation invariant texture analysis with local binary patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 971-987, Jul 2002.
- [20] L. Ghiani, G. L. Marcialis and F. Roli, "Fingerprint Liveness Detection by Local Phase Quantization," *21st International Conference on IEEE*, 2012.
- [21] L. Ghiani, H. A., G. L. Marcialis and F. Roli, "Fingerprint liveness detection using Binarized Statistical Image Features," *In Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*, pp. 1-6, September 2013.
- [22] [Online]. Available: [http://atvs.ii.uam.es/ffp\\_db.html](http://atvs.ii.uam.es/ffp_db.html).
- [23] E. R. Dougherty and R. A. Lotufo, *Hands-on morphological image processing*, Bellingham: SPIE press, 2003.
- [24] M. Mohri, A. Rostamizadeh and A. Talwalkar, *Foundations of Machine Learning*, The MIT Press, 2012.
- [25] M. I. Jordan and C. M. Bishop, "Neural Networks," in *Computer Science Handbook, Second Edition*, Chapman & Hall/CRC Press LLC, 2004.
- [26] P. Cunningham, "Dimension Reduction," University College Dublin.
- [27] E. B.S., *Cambridge Dictionary of Statistics*, 2002.
- [28] R. E. Bellman, *Dynamic programming*, Princeton University Press, 1957.
- [29] S. Geisser, *Predictive Inference*, New York, NY: Chapman and Hall, 1993.
- [30] G. L. Marcialis, A. Lewicke, B. Tan, P. Coli, D. Grimberg, A. Congiu and S. Schuckers, "First international fingerprint liveness detection competition—livdet 2009.," *Image Analysis and Processing—ICIAP 2009. Springer Berlin Heidelberg*, pp. 12-23, 2009.
- [31] D. Yambay, L. Ghiani, P. Denti, G. L. Marcialis, F. Roli and S. Schuckers, "LivDet 2011—Fingerprint liveness detection competition 2011.," *Biometrics (ICB), 2012 5th IAPR International Conference on*, pp. 208-215, 2012.
- [32] L. Ghiani, D. Yambay, V. Mura, S. Tocco, G. L. Marcialis, F. Roli and S. Schuckers, "LivDet 2013 fingerprint liveness detection competition 2013," *Biometrics (ICB), 2013 International Conference on*, pp. 1-6, 2013.
- [33] J. B. Zimmerman, S. M. Pizer, E. V. Staab, J. R. Perry, W. McCartney and B. C. Brenton, "An evaluation of the effectiveness of adaptive histogram equalization for contrast enhancement," *Medical Imaging, IEEE Transactions*, pp. 304-312, 1988.
- [34] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer vision, graphics, and image processing*, pp. 355-368, 1987.
- [35] Y. LeCun, "Generalization and network design strategies," in *Connections in Perspective*, 1989.
- [36] L. Wan, M. Zeiler, S. Zhang, Y. LeCun and R. Fergus, "Regularization of neural networks using dropconnect," *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058-1066, 2013.
- [37] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.
- [38] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *NIPS*, vol. 1, no. 2, 2012.

- [39] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The handbook of brain theory and neural networks*, 1995, pp. 33-61.
- [40] D. Baldisserra, A. Franco, D. Maio and D. Maltoni, "Fake fingerprint detection by odor analysis," in *Advances in Biometrics*, Berlin Heidelberg, Springer, 2005, pp. 265-272.
- [41] Y. L. Boureau, J. Ponce and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.
- [42] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013.
- [43] G. E. Hinton, S. Osindero and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation* 18.7, pp. 1527-1554, 2006.
- [44] Y. LeCun, K. Kavukcuoglu and C. Farabet, "Convolutional networks and applications in vision," *In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 253-256, May 2010.
- [45] K. Jarrett, K. Kavukcuoglu, M. Ranzato and Y. LeCun, "What is the best multi-stage architecture for object recognition?," *Computer Vision, 2009 IEEE 12th International Conference on. IEEE*, pp. 2143-2153, 2009.
- [46] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.
- [47] S. Lyu and E. P. Simoncelli, "Nonlinear image representation using divisive normalization," *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. IEEE*, 2008.
- [48] N. Pinto, D. D. Cox and J. J. DiCarlo, "Why is real-world visual object recognition hard?," *PLoS computational biology*, vol. 4, no. 1, p. e27, 2008.
- [49] S. Lyu, "Divisive Normalization: Justification and Effectiveness as Efficient Coding Transform," *NIPS*, 2010.
- [50] A. Hadid, M. Pietikainen and T. Ahonen, "A discriminative feature space for detecting and recognizing faces," *Computer Vision and Pattern Recognition*, 2004.
- [51] G. Zhao, T. Ahonen, J. Matas and M. Pietikainen, "Rotation-Invariant Image and Video Description With Local Binary Pattern Features," *IEEE Transactions On Image Processing*, vol. 21, no. 4, pp. 1465-1477, 2012.
- [52] T. Ahonen, A. Hadid and M. Pietikäinen, "Face recognition with local binary patterns," *Computer vision-eccv*, pp. 469-481, 2004.
- [53] J. Jackson, *A User's Guide to Principal Components*, Wiley, 1991.
- [54] N. Halko, P.-G. Martinsson and J. A. Tropp., "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.," *SIAM review* 53.2, pp. 217-288, 2011.
- [55] P.-G. Martinsson, V. Rokhlin and M. Tygert, "A randomized algorithm for the decomposition of matrices.," *Applied and Computational Harmonic Analysis* 30.1, pp. 47-68, 2011.
- [56] A. Hyvärinen, J. Hurri and P. O. Hoyer, "Principal Components and Whitening," in *Natural Image Statistics*, London, Springer, 2009, pp. 93-130.
- [57] "Whitening," [Online]. Available: <http://ufldl.stanford.edu/wiki/index.php/Whitening>. [Accessed 08 03 2014].
- [58] A. Coates, A. Y. Ng and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," *International Conference on Artificial Intelligence and Statistics*, 2011.

- [59] N. Cristianini and J. Shawe-Taylor, An introduction to support vector machines and other kernel-based learning methods., Cambridge university press, 2000.
- [60] C.-W. Hsu, C.-C. Chang and C.-J. Lin, "A practical guide to support vector classification.," 2003.
- [61] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 3, no. 20, pp. 273-297, 1995.
- [62] J.-P. Vert, K. Tsuda and B. Schölkopf, "A primer on kernel methods," in *Kernel Methods in Computational Biology*, 2004, pp. 35-70.
- [63] H. Lei and V. Govindaraju, "Speeding Up Multi-class SVM Evaluation by PCA and Feature Selection," *Feature Selection for Data Mining*, p. 72, 2005.
- [64] L. J. Cao, K. S. Chua, W. K. Chong, H. P. Lee and Q. M. Gu, "A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine," *Neurocomputing 55.1*, pp. 321-336, 2003.
- [65] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella and J. Schmidhuber, "High-performance neural networks for visual object classification," *arXiv:1102.0183*, 2011.
- [66] D. Cireşan, U. Meier and J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.*, pp. 3642-3649, June 2012.
- [67] G. Chiachia. [Online]. Available: <https://github.com/giovanichiachia/convnet-rfw>. [Accessed 17 05 2014].
- [68] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Computation*, vol. 10, no. 7, pp. 1895-1923, 1998.
- [69] K. Jarrett, K. Kavukcuoglu, M. Ranzato and Y. LeCun, "What is the best multi-stage architecture for object recognition?," *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2146-2153, 2009.
- [70] J. Yang, K. Yu, Y. Gong and T. Huang, "Linear spatial pyramid matching using sparse coding for image classification," *Computer Vision and Pattern Recognition*, pp. 1794-1801, 2009.
- [71] Y. L. Boureau, F. Bach, Y. LeCun and J. Ponce, "Learning mid-level features for recognition," *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2559-2566, 2010.
- [72] B. A. Draper, K. Baek, M. S. Bartlett and J. R. Beveridge, "Recognizing faces with PCA and ICA," *Computer vision and image understanding*, vol. 1, no. 91, pp. 115-137, 2003.
- [73] K. Delac, M. Grgic and S. Grgic, "Independent comparative study of PCA, ICA, and LDA on the FERET data set," *International Journal of Imaging Systems and Technology*, vol. 5, no. 15, pp. 252-260, 2005.
- [74] C. Farabet, Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod and S. Talay, "Large-scale FPGA-based convolutional networks," *Machine Learning on Very Large Data Sets*, 2011.
- [75] G. E. Hinton, S. Osindero and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation 18.7*, pp. 1527-1554, 2006.
- [76] A. Pacut and A. Czajka, "Aliveness detection for iris biometrics," *Carnahan Conferences Security Technology, Proceedings 2006 40th Annual IEEE International*, 2006.
- [77] K. Baek, B. A. Draper, J. R. Beveridge and K. She, "PCA vs. ICA: A comparison on the FERET data set," *JCIS*, pp. 824-827, 2002.
- [78] A. M. Martínez and A. C. Kak, "Pca versus lda," *Pattern Analysis and Machine Intelligence, IEEE*

*Transactions on*, vol. 2, no. 23, pp. 228-233, 2001.

## Appendix A

Table 8 shows the validation error in each dataset. The tag “n/e” stands for not executed, meaning that due to the large amount of time to search for parameters in augmented datasets, we did not execute the validation in the pipelines that use this technique. The parameters used in the testing dataset for those pipelines were the ones found in validation for the corresponding pipelines that do not use the augmentation technique.

		Aug LBP PCA Gaussian- SVM	LBP PCA Gaussian- SVM	Aug CN PCA Gaussian- SVM	CN PCA Gaussian- SVM	CN PCA Linear- SVM	CN Linear- SVM
<b>LivDet 2013</b>	<b>Crossmatch</b>	0	0	n/e	3.18	2.61	3.61
	<b>Swipe</b>	2.72	2.3	n/e	4.4	4.85	4.59
	<b>Italdata</b>	0.2	0.04	0.36	0.21	0.05	0.12
	<b>Biometrika</b>	0.96	0.1	n/e	1.07	0.69	0.94
<b>LivDet 2011</b>	<b>Italdata</b>	2.88	0.08	2.05	0.11	0.15	0.2
	<b>Biometrika</b>	n/e	0.37	n/e	2.17	3.58	3.71
	<b>Digital</b>	15.06	1.01	n/e	1.45	0.98	1.45
	<b>Sagem</b>	14.58	1.13	n/e	2.37	2.13	3.21
<b>Livdet 2009</b>	<b>Biometrika</b>	0	0	0	0	0.02	0
	<b>Crossmatch</b>	1.32	1.87	n/e	0.81	0.7	0.96
	<b>Identix</b>	0.6	0.67	n/e	0.31	1.15	0.96
	<b>Average</b>	3.83	0.69	0.18	1.41	1.54	1.80

**Table 8 - Validation Error**