Advances in Computer Vision and Pattern Recognition

Bir Bhanu
Ajay Kumar *Editors*

Deep Learning for Biometrics

Springer

Advances in Computer Vision and Pattern Recognition

More information about this series at http://www.springer.com/series/4205

Bir Bhanu · Ajay Kumar
Editors

Deep Learning for Biometrics

Springer

Editors
Bir Bhanu
University of California
Riverside, CA
USA

Ajay Kumar
Hong Kong Polytechnic University
Hong Kong
China

ISSN 2191-6586 ISSN 2191-6594 (electronic)
Advances in Computer Vision and Pattern Recognition
ISBN 978-3-319-61656-8 ISBN 978-3-319-61657-5 (eBook)
DOI 10.1007/978-3-319-61657-5

Library of Congress Control Number: 2017943828

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

With a very security-conscious society, biometrics-based authentication and identification have become the focus of many important applications as it is widely believed that biometrics can provide accurate and reliable identification. Biometrics research and technology continue to mature rapidly, driven by pressing industrial and government needs and supported by industrial and government funding. As the number and types of biometrics architectures, sensors, and techniques increases, the need to disseminate research results increases as well.

Advanced deep learning capabilities, and deep convolutional neural networks (CNN) in particular, are significantly advancing the state of the art in computer vision and pattern recognition. The deep CNN is a biologically inspired variant of multilayer perceptron and represents a typical deep learning architecture.

Since 2006, we have organized a series of high-quality Annual Biometrics Workshops under the auspices of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). This series has emerged as the premier forum for showcasing cutting-edge research from academia, industry, and government laboratories. During the past few years the CNN-based techniques, as evidenced by the increasing number of papers at CVPR and the biometrics workshop, have shown strong ability to learn effective feature representation from input data, especially for the perceptual and biometrics-related tasks.

The book is based on a selection of topics and authors from the proceedings of the 2016 biometrics workshop and a general call for chapters to the computer vision, pattern recognition, and the biometrics communities at large. The selection of chapters in this book is made after two rounds of rigorous review process.

Outline of the Book and Chapter Synopsis

Many of the biometrics applications require the highest level of accuracy that is being made available with the advanced deep learning capabilities. This book addresses many aspects of biometrics research issues relating to different biometrics

modalities with the sole goal of improving performance for the biometrics identification. A brief and orderly introduction to the chapters is provided in the following.

The book chapters in this book on *Deep Learning for Biometrics* are organized under four major parts. **Part I** deals with deep learning for face biometrics and it includes three chapters on this topic in the book.

Chapter 1 by Grill-Spector, Kay, and Weiner on the functional neuroanatomy of face processing: insights from neuroimaging and implications for deep learning provides insightful details on the connections between currently popular deep neural network architectures for the biometrics and neural architectures in the human brain. The authors in this chapter detail a range of findings from the neuroimaging techniques on the anatomical features of the face network and the computations performed by the face-selective regions in the human brain. These empirical findings can lead to more accurate deep neural networks for face recognition.

One of the challenges related to the applications of the deep learning-based biometrics solutions is the computational complexity of the selected network. Many applications of face recognition demand highly efficient search capabilities from the large databases and in a database which requires smaller template size and faster template-matching capabilities.

Chapter 2 by Vizilter, Gorbatsevich, Vorotnikov, and Kostromov on real-time face identification via a multi-convolutional neural network and boosted hashing forest describes the development of boosted hashing-based real-time face recognition system using convolutional neural networks. This chapter presents a new biometric-specific objective function for the binary hashing that enables joint optimization of the face verification and identification.

Despite significant improvement in the accuracy of the face detection algorithms in the past decade, their accuracy is still far from that displayed by humans, particularly for the images acquired under challenging imaging environments, like with large occlusions or off-poses. Therefore, Chap. 3 by Zhu, Zheng, Luu, and Savvides on CMS-RCNN: contextual multi-scale region-based CNN for unconstrained face detection introduces a new architecture, a contextual multi-scale region-based CNN, to accurately detect human faces from a complex background and under challenging imaging conditions.

Part II of the book deals with deep learning for fingerprint, finger vein, and iris recognition. It includes three chapters that are described below.

Some of the challenging open problems in the fingerprint identification are related to accurately segmenting latent fingerprint images. Chapter 4 by Ezeobiejesi and Bhanu on latent fingerprint image segmentation using deep neural networks describes how deep neural networks can be employed to accurately segment latent fingerprint regions from complex image backgrounds. The authors present a latent fingerprint image patch and noise image patch-based classification strategy that outperforms the results on publicly available latent fingerprint databases.

Vascular biometrics identification has attracted several applications and is believed to significantly improve the integrity of biometrics systems, while

preserving the privacy of an individual during authentication. Chapter 5 by Xie and Kumar on finger vein identification using convolutional neural networks and supervised discrete hashing presents a detailed deep learning-based investigation into finger vein identification. The development of competing deep learning-based solutions that can be operational using limited training data is among one of the open challenges in biometrics. Biometrics modalities like finger vein have very limited dataset in the public domain, primarily due to enhanced privacy concerns and/or due to cooperative imaging requirements. The experimental results presented in this chapter, using publicly available but limited two-session dataset indicate promises from supervised discrete hashing and provide insights into the comparative performance with state-of-the-art methods for finger vein identification.

Almost all the iris recognition systems deployed today operate using stop and stare mode. Such systems operate in constrained imaging environment and deliver remarkable accuracy. Application of iris recognition technologies for surveillance and at-a-distance applications require accurate segmentation capabilities from such images acquired with visible and near-infrared illuminations. Chapter 6 by Jalilian and Uhl on iris segmentation using fully convolutional encoder–decoder networks details the segmentation of challenging iris images using fully convolutional encoder–decoder networks.

Part III of the book deals with deep learning for soft biometrics and it includes four interesting chapters on this topic.

Accurate identification of soft biometrics features is vital for improving the accuracy of biometrics-based surveillance systems. Chapter 7 by Wu, Chen, Ishwar, and Konrad on two-stream CNNs for gesture-based verification and identification: learning user style details a deep learning framework that simultaneously leverages on the spatial and temporal information in video sequences. The experimental results presented by the authors for the identification and verification on two biometrics-oriented gesture datasets indicate results that outperform the state-of-art methods in the literature.

Developing accurate capabilities to automatically detect the soft biometrics features, like the gender, from the low resolution, off angle, and occluded face images is highly desirable for a range of biometrics applications. Chapter 8 by Juefei-Xu, Verma, and Savvides is on DeepGender2: a generative approach toward occlusion and low-resolution robust facial gender classification via progressively trained attention shift convolutional neural networks (PTAS-CNN) and deep convolutional generative adversarial networks (DCGAN). It describes the development of a deep learning-based gender classification approach. The authors describe how a progressive training strategy and the deep generative approach to recover the missing pixels can achieve excellent results for the gender classification using occluded face images.

Chapter 9 by Tapia and Aravena is on gender classification from near-infrared (NIR) iris images using deep learning. Like the previous chapter, it is also devoted

to the gender classification problem but using NIR iris images. Such images are typically available during the iris acquisition and authors use a pretrained deep belief network to identify gender from these images.

Several law enforcement departments use tattoos to identify the personal beliefs and characteristics, similar to many popular soft biometrics features. Chapter 10 by Di and Patel on deep learning for tattoo recognition describes how the Siamese networks with the conventional triplet function can be used to identify tattoos in publicly available databases, with very good results.

Part IV of the book deals with deep learning for biometrics security and template protection and it consists of two chapters.

Ensuring the security of biometrics templates and the systems is an integral part of biometrics infrastructure. Chapter 11 by Pandey, Zhou, Kota, and Govindaraju on learning representations for cryptographic hash-based face template protection introduces challenges and the techniques for the protection of biometrics template. The authors in this chapter introduce template representations that are learned by CNN for the cryptographic hash-based protection of face image templates.

The last chapter in this book, i.e., Chap. 12 by Pala and Bhanu on deep triplet embedding representations for liveness detection considers widely discussed problems relating to the protection of biometrics systems against fake biometrics samples. The authors introduce a metric learning strategy that uses a variant of triplet loss function to identify fake fingerprints in image patches. Experimental results presented on publicly available database indicate outperforming state-of-the-art results from this deep learning-based strategy.

Challenges for the Future

A brief summary of the book chapters in the above paragraphs indicates that there has been significant interest in the advancement of biometrics technologies using the deep learning architectures. The work underlines challenges in deep learning when the training sample size available from a particular biometrics modality is quite small. Several authors have underlined that the factors that most influence the accuracy from deep neural networks is the depth of the network, pretraining, and the data augmentation in terms of random crops and rotations.

Several classical biometrics feature extraction and matching algorithms work very well when the images are acquired under relatively constrained environments, e.g., fingerprint and iris, with no constraints on the need for huge training samples. It is unclear how learned deep neural networks could aid, improve, or replace such popular classical methods that have been matured in past three decades, like the popular *iriscode*-based iris recognition or the minutiae matching-based fingerprint recognition widely deployed today. In this context, it is important that emerging deep learning based algorithms for biometrics should also present careful performance

comparisons, on public datasets using appropriate protocols, and under open set environments or cross-dataset evaluations to make a convincing case for the benefits of biometrics community in academia, industry, and the Government.

Riverside, CA, USA Bir Bhanu
April 2017 Ajay Kumar

Contents

Chapter 4
Latent Fingerprint Image Segmentation Using Deep Neural Network

Jude Ezeobiejesi and Bir Bhanu

Abstract We present a deep artificial neural network (DANN) model that learns latent fingerprint image patches using a stack of restricted Boltzmann machines (RBMs), and uses it to perform segmentation of latent fingerprint images. Artificial neural networks (ANN) are biologically inspired architectures that produce hierarchies of maps through learned weights or filters. Latent fingerprints are fingerprint impressions unintentionally left on surfaces at a crime scene. To make identifications or exclusions of suspects, latent fingerprint examiners analyze and compare latent fingerprints to known fingerprints of individuals. Due to the poor quality and often complex image background and overlapping patterns characteristic of latent fingerprint images, separating the fingerprint region of interest from complex image background and overlapping patterns is very challenging. Our proposed DANN model based on RBMs learns fingerprint image patches in two phases. The first phase (unsupervised pre-training) involves learning an identity mapping of the input image patches. In the second phase, fine-tuning and gradient updates are performed to minimize the cost function on the training dataset. The resulting trained model is used to classify the image patches into fingerprint and non-fingerprint classes. We use the fingerprint patches to reconstruct the latent fingerprint image and discard the non-fingerprint patches which contain the structured noise in the original latent fingerprint. The proposed model is evaluated by comparing the results from the state-of-the-art latent fingerprint segmentation models. The results of our evaluation show the superior performance of the proposed method.

J. Ezeobiejesi (✉) · B. Bhanu
Center for Research in Intelligent Systems, University of California at Riverside, Riverside, CA 92521, USA
e-mail: jezeobie@cs.ucr.edu

B. Bhanu
e-mail: bhanu@cris.ucr.edu

B. Bhanu and A. Kumar (eds.), *Deep Learning for Biometrics*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-61657-5_4

4.1 Introduction

Deep learning is a technique for learning features using hierarchical layers of neural networks. There are usually two phases in deep learning. The first phase commonly referred to as pre-training involves unsupervised, layer-wise training. The second phase (fine-tuning) involves supervised training that exploits the results of the first phase. In deep learning, hierarchical layers of learned abstraction are used to accomplish high level tasks [3]. In recent years, deep learning techniques have been applied to a wide variety of problems in different domains [3]. Some of the notable areas that have benefited from deep learning include pattern recognition [21], computer vision [16], natural language processing, and medical image segmentation [17]. In many of these domains, deep learning algorithms outperformed previous state-of-the-art algorithms.

Latent fingerprints are fingerprint impressions unintentionally left on surfaces at a crime scene. Latent examiners analyze and compare latent fingerprints to known fingerprints of individuals to make identifications or exclusions of suspects [9]. Reliable latent fingerprint segmentation is an important step in the automation of latent fingerprint processing. Better latent fingerprint matching results can be achieved by having automatic latent fingerprint segmentation with a high degree of accuracy. In recent years, the accuracy of latent fingerprint identification by latent fingerprint forensic examiners has been the subject of increased study, scrutiny, and commentary in the legal system and the forensic science literature. Errors in latent fingerprint matching can be devastating, resulting in missed opportunities to apprehend criminals or wrongful convictions of innocent people. Several high-profile cases in the United States and abroad have shown that forensic examiners can sometimes make mistakes when analyzing or comparing fingerprints [14] manually. Latent fingerprints have significantly poor quality ridge structure and large nonlinear distortions compared to rolled and plain fingerprints. As shown in Fig. 4.1, latent fingerprint images contain background structured noise such as stains, lines, arcs, and sometimes text. The poor quality and often complex image background and overlapping patterns characteristic of latent fingerprint images make it very challenging to separate the fingerprint regions of interest from complex image background and overlapping patterns [29]. To process latent fingerprints, latent experts manually mark the regions of interest (*ROIs*) in latent fingerprints and use the *ROIs* to search large databases of reference full fingerprints and identify a small number of potential matches for manual examination. Given the large size of law enforcement databases containing rolled and plain fingerprints, it is very desirable to perform latent fingerprint processing in a fully automated way. As a step in this direction, this chapter proposes an efficient technique for separating latent fingerprints from the complex image background using deep learning. We learn a set of features using a hierarchy of RBMs. These features are then passed to a supervised learning algorithm to learn a classifier for patch classification. We use the result of the classification for latent fingerprint image segmentation. To the best of our knowledge, no previous work has used this strategy to segment latent fingerprints.

Fig. 4.1 Sample latent fingerprints from NIST SD27 showing three different quality levels **a** good, **b** bad, and **c** ugly

The rest of the chapter is organized as follows: Sect. 4.2.1 reviews recent works in latent fingerprint segmentation while Sect. 4.2.1.1 describes the contributions of this chapter. Section 4.3 highlights our technical approach and presents an overview of RBM as well discussion on learning with RBMs. The experimental results and performance evaluation of our proposed approach are presented in Sect. 4.4. Section 4.4.5 highlights the impacts of diffusing the training dataset with fractal dimension and lacunarity features on the performance of the network while Sect. 4.5 contains the conclusions and future work.

4.2 Related Work and Contributions

4.2.1 Related Work

Recent studies carried out on latent fingerprint segmentation can be grouped into three categories:

- Techniques based on classification of image patches
- Techniques based on clustering
- Techniques that rely on ridge frequency and orientation properties

The study presented in [9] falls into the first category. The authors performed image segmentation by extracting 8×8 nonoverlapping patches from a latent fingerprint image and classifying them into fingerprint and non-fingerprint patches using fractal dimension features computed for each image patch. They assembled the fingerprint patches to build the fingerprint portion (segmented region of interest) of the original image.

In the second category of approaches, Arshad et al. [4] used K-means clustering to divide the latent fingerprint image into nonoverlapping blocks and computed the standard deviation of each block. They considered a block as foreground if its standard

deviation is greater than a predefined threshold otherwise, it was a background block. They used morphological operations to segment the latent fingerprint.

The approaches that fall into the third category rely on the analysis of the ridge frequency and orientation properties of the ridge valley patterns to determine the area within a latent fingerprint image that contains the fingerprint [4, 7, 13, 27, 29]. Choi et al. [7] used orientation tensor approach to extract the symmetric patterns of a fingerprint and removed the structural noise in background. They used a local Fourier analysis method to estimate the local frequency in the latent fingerprint image and located fingerprint regions by considering valid frequency ranges. They obtained candidate fingerprint (foreground) regions for each feature (orientation and frequency) and then localized the latent fingerprint regions using the intersection of those candidate regions. Karimi et al. [13] estimated local frequency of the ridge/valley pattern based on ridge projection with varying orientations. They used the variance of frequency and amplitude of ridge signal as features for the segmentation algorithm. They reported segmentation results for only two latent fingerprint images and provided no performance evaluation. Short et al. [27] proposed the ridge template correlation method for latent fingerprint segmentation. They generated an ideal ridge template and computed cross-correlation value to define the local fingerprint quality. They manually selected six different threshold values to assign a quality value to each fingerprint block. They neither provided the size and number for the ideal ridge template nor reported evaluation criteria for the segmentation results. Zhang et al. [29] proposed an adaptive total variation (TV) model for latent fingerprint segmentation. They adaptively determined the weight assigned to the fidelity term in the model based on the background noise level. They used it to remove the background noise in latent fingerprint images.

Our approach uses a deep architecture that performs learning and classification in a two-phase approach. The first phase (unsupervised pre-training) involves learning an identity mapping of the input image patches. In the second phase (fine-tuning), the model performs gradient updates to minimize the cost function on the dataset. The trained model is used to classify the image patches and the results of the classification are used for latent fingerprint image segmentation.

4.2.1.1 Contributions

This chapter makes the following contributions:

- Modification of how the standard RBM learning algorithm is carried out to incorporate a weighting scheme that enables the RBM in the first layer to model the input data with near zero reconstruction error. This enabled the higher level weights to model the higher level data efficiently.
- A cost function based on weighted harmonic mean of missed detection rate and false detection rate is introduced to make the network learn the minority class better, and improve per class accuracy. By heavily penalizing the misclassication of minority (fingerprint) class, the learned model is tuned to achieve close to zero missed detection rate for the minority class.

- The proposed generative feature learning model and associated classifier yield state-of-the-art performance on latent fingerprint image segmentation that is consistent across many latent fingerprint image databases.

4.3 Technical Approach

Our approach involves partitioning a latent fingerprint image into 8×8 nonoverlapping blocks, and learning a set of stochastic features that model a distribution over image patches using a generative multilayer feature extractor. We use the features to train a single-layer perceptron classifier that classifies the patches into fingerprint and non-fingerprint classes. We use the fingerprint patches to reconstruct the latent fingerprint image and discard the non-fingerprint patches which contain the structured noise in the original latent fingerprint. The block diagram of our proposed approach is shown in Fig. 4.2, and the architecture of the feature learning, extraction, and classification model is shown in Fig. 4.3.

4.3.1 Restricted Boltzmann Machine

A restricted Boltzmann machine is a stochastic neural network that consists of visible layer, hidden layer, and a bias unit [11]. A sample RBM with binary visible and hidden units is shown in Fig. 4.4. The energy function E_f of RBM is linear in its free parameters and is defined as [11]:

$$E_f(\hat{x}, h) = -\sum_i b_i \hat{x}_i - \sum_j c_j h_j - \sum_i \sum_j \hat{x}_i w_{i,j} h_j, \tag{4.1}$$

where $\hat{x}$ and h represent the visible and hidden units, respectively, W represents the weights connecting $\hat{x}$ and h, while b and c are biases of the visible and hidden units, respectively. The probability distributions over visible or hidden vectors are defined in terms of the energy function [11]:

$$P(\hat{x}, h) = \frac{1}{\omega} e^{-E_f(\hat{x}, h)}, \tag{4.2}$$

where ω is a partition function that ensures the probability distribution of over all possible configurations of the hidden or visible vectors sum to 1. The marginal probability of a visible vector $P(\hat{x})$ is the sum over all possible hidden layer configurations [11] and is defined as:

$$P(\hat{x}) = \frac{1}{\omega} \sum_h e^{-E_f(\hat{x}, h)} \tag{4.3}$$

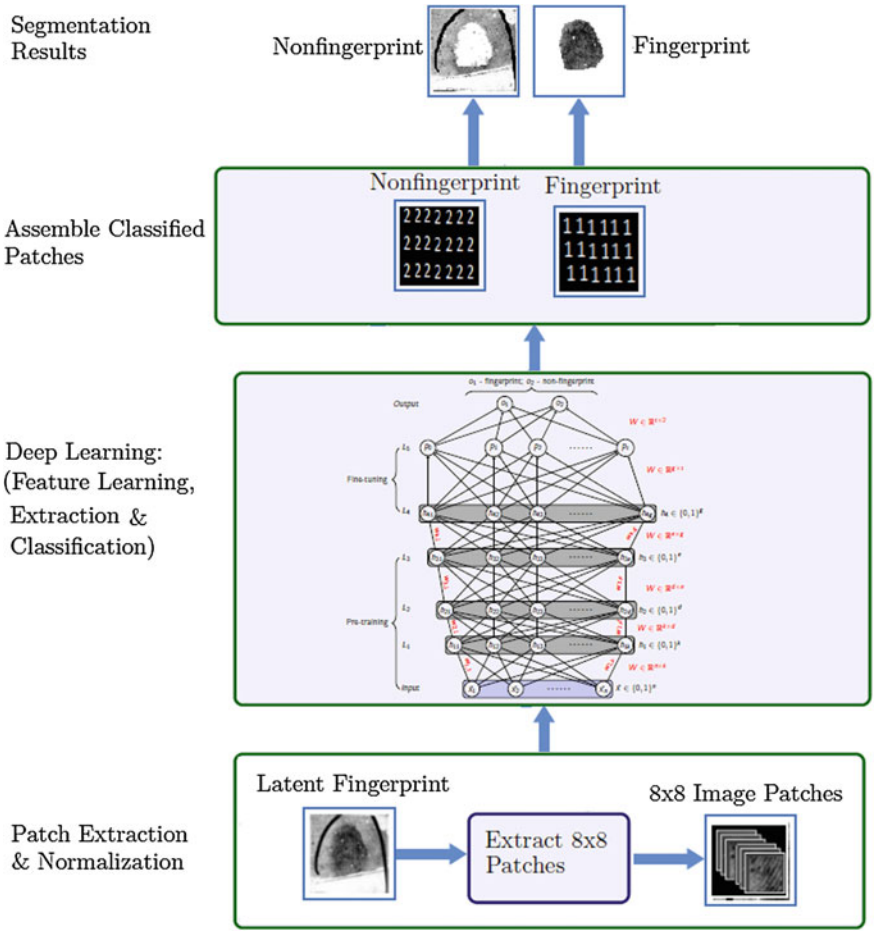

Fig. 4.2 Proposed approach

RBM has no intra-layer connections and given the visible unit activations, the hidden unit activations are mutually independent. Also the visible unit activations are mutually independent given the hidden unit activations [6]. The conditional probability of a configuration of the visible units is given by

$$P(\hat{x}|h) = \prod_{i=1}^{n} P(\hat{x}_i|h), \tag{4.4}$$

where n is the number of visible units. The conditional probability of a configuration of hidden units given visible units is

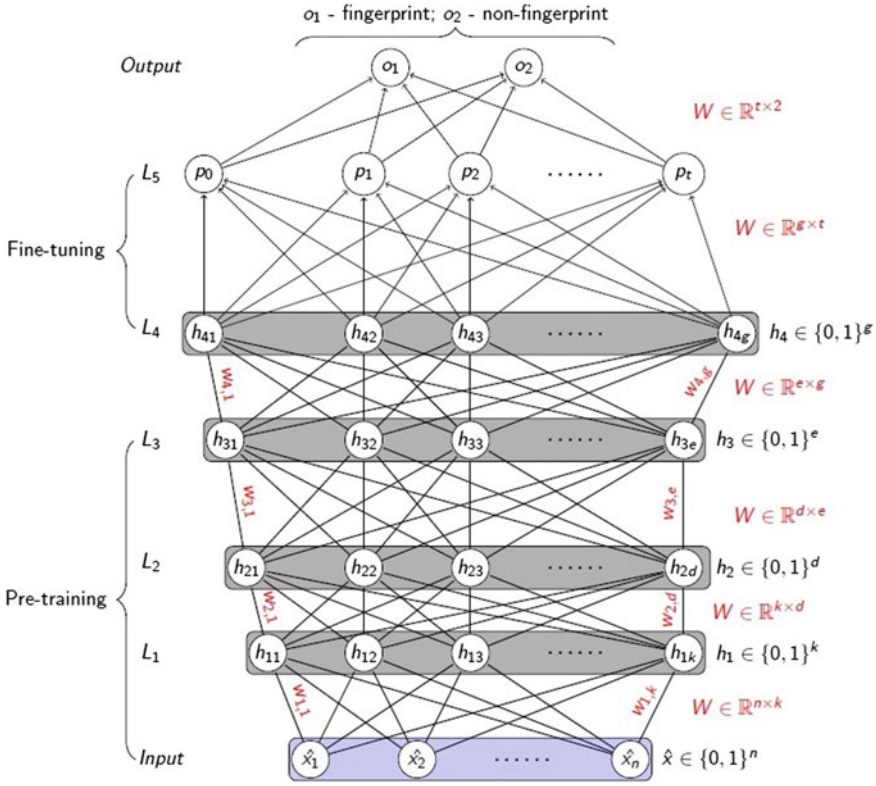

Fig. 4.3 Feature learning, extraction, and classification using a multilayer neural network. The pre-training phase uses the input layer (visible units), and three hidden layers of RBM (L_1, L_2, L_3). The fine-tuning phase uses an RBM layer (L_4) and a single-layer perceptron (L_5). The output layer has two output neurons (fingerprint and non-fingerprint). All the units are binary. $h_{i,j}$ is the jth node in L_i, $w_{i,j}$ is the weight connecting the ith node in layer L_i to the jth node in layer L_{i-1}. We set $n = 81$ (64 from 8×8 and 17 from diffusion), $k = 800$, $d = 1000$, $e = 1200$, $g = 1200$, $t = 1200$, where n, k, d, e, g, t are the number of nodes in the input layer, L_1, L_2, L_3, L_4, L_5, respectively

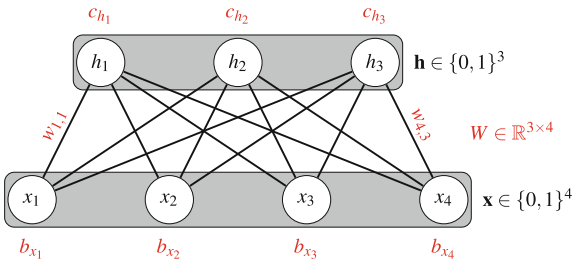

Fig. 4.4 Graphical depiction of RBM with binary visible and hidden units. x_i, $i = 1, \ldots, 4$, are the visible units while h_k, $k = 1, \ldots, 3$, are the hidden units. b_{x_i}, $i = 1, \ldots, 4$, are the biases for the visible units and c_{h_k}, $k = 1, \ldots, 3$, are the biases for the hidden units

$$P(h|\hat{x}) = \prod_{j=1}^{m} P(h_j|\hat{x}), \tag{4.5}$$

where m is the number of hidden units. The individual activation probabilities are given by

$$P(h_j = 1|\hat{x}) = \sigma\left(b_j + \sum_{i=1}^{n} w_{i,j}\hat{x}_i\right) \tag{4.6}$$

and

$$P(\hat{x}_i = 1|h) = \sigma\left(c_i + \sum_{j=1}^{m} w_{i,j}h_j\right), \tag{4.7}$$

where c_i is the ith hidden unit bias, b_j is the jth visible unit bias, $w_{i,j}$ is the weight connecting the ith visible unit and jth hidden unit, and σ is the logistic sigmoid.

4.3.2 Learning with RBM

Learning with RBM involves several steps of sampling hidden variables given visible variables, sampling visible variables given hidden variables, and minimizing reconstruction error by adjusting the weights between the hidden unit and visible layers. The goal of learning with RBM is to identify the relationship between the hidden and visible variables using a process akin to identity mapping. We performed the sampling step using Gibbs sampling technique enhanced with contrastive divergence.

4.3.2.1 Gibbs Sampling

A sampling algorithm based on Monte Carlo Markov Chain (MCMC) technique used in estimating desired expectations in learning models. It allows for the computation of statistics of a posterior distribution of given simulated samples from that distribution [28]. A Gibbs sampling of the joint of R random variables $R = (R_1, R_2, \ldots, R_n)$ involves a sequence of R sampling sub-steps of the form $R_i \sim p(R_i|R_{-i})$ where R_i contains the *n-1* other random variables in R excluding R_i. For RBMs, $R = Q_1 \cup Q_1$ where $Q_1 = \{\hat{x}_i\}$ and $Q_2 = \{h_i\}$. Given that the sets Q_1 and Q_2 are conditionally independent, the visible units can be sampled simultaneously given fixed values of the hidden units using block Gibbs sampling. Similarly, the hidden units can be sampled simultaneously given the visible units. The following is a step in the Markov chain:

$h^{(k+1)} \sim \sigma(W^T v^{(k)} + c)$

$\hat{x}^{(k+1)} \sim \sigma(W h^{(k+1)} + b)$,

where $h^{(k)}$ refers to the set of all hidden units at the kth step of the Markov chain and σ denotes logistic sigmoid defined as

$$o(x) = \frac{1}{1 + e^{-W_v z(x) - b}} \tag{4.8}$$

$$\text{with } z(x) = \frac{1}{1 + e^{-W_h x - c}}, \tag{4.9}$$

where W_h and c are the weight matrix and bias for the hidden layers excluding the first layer, and $z(x)$ is the activation of the hidden layer in the network. W_v is a weight matrix connecting the visible layer to the first hidden layer, and b is a bias for the visible layer.

4.3.2.2 Contrastive Divergence (CD-k)

This algorithm is used inside gradient descent procedure to speed up Gibbs sampling. It helps in optimizing the weight W during RBM training. CD-k speeds up Gibbs sampling by taking sample after only k-steps of Gibbs sampling, without waiting for the convergence of the Markov chain. In our experiments we set $k = 1$.

4.3.2.3 Stochastic Gradient Descent

With large datasets, computing the cost and gradient for the entire training set is usually very slow and may be intractable [24]. This problem is solved by Stochastic Gradient Descent (SGD) by following the negative gradient of the objective function after seeing a few training examples. SGD is used in neural networks to mitigate the high cost of running backpropagation over the entire training set [24].

Given an objective function $J(\phi)$, the standard gradient descent algorithm updates the parameters ϕ as follows:

$$\phi = \phi - \alpha \nabla_\phi E[J(\phi], \tag{4.10}$$

where the expectation $E[J(\phi]$ is obtained through an expensive process of evaluating the cost and gradient over the entire training set. With SGD, the gradient of the parameters are computed using a few training examples with no expectation to worry about. The parameters are update as,

$$\phi = \phi - \alpha \nabla_\phi J(\phi; x^{(i)}, y^{(i)}) \tag{4.11}$$

where the pair $(x^{(i)}, y^{(i)})$ are from the training set. Each parameter update is computed using a few training examples. This reduces the variance in the parameter update with the potential of leading to more stable convergence. Prior to each training epoch, we

randomly shuffled the training data to avoid biasing the gradient. Presenting the training data to the network in a nonrandom order could bias the gradient and lead to poor convergence.

One of the issues with learning with stochastic gradient descent is the tendency of the gradients to decrease as they are backpropagated through multiple layer of nonlinearity. We worked around this problem by using different learning rates for each layer in the proposed network.

4.3.2.4 Cost Function

Our goal is to classify all fingerprint patches (minority class) correctly to meet our segmentation objective of extracting the region of interest (fingerprint part) from the latent fingerprint image. We introduced a cost function based on the weighted harmonic mean of missed detection rate and false detection rate. We adopted a weight assignment scheme that was skewed in favor of the minority class to make the neural network learn the minority class better. Given a set of weights $w_1, w_2, \ldots, w_n$ associated with a dataset $x_1, x_2, \ldots, x_n$, the weighted harmonic mean $\mathbb{H}$ is defined as

$$\mathbb{H} = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{x_i}} = \left(\frac{\sum_{i=1}^{n} w_i x_i^{-1}}{\sum_{i=1}^{n} w_i} \right)^{-1}. \tag{4.12}$$

By penalizing the misclassification of minority class more, the model learned to detect minority class with a high degree of accuracy. The cost function is defined as:

$$\mathbb{C} = \frac{2}{\frac{1}{\tau MDR} + \frac{1}{\tau FDR}}, \tag{4.13}$$

where τMDR and τFDR are the weighted missed detection rate and weighted false detection rate, respectively. They are computed as: $\tau MDR = \tau_1 * MDR$ and $\tau FDR = \tau_2 * FDR$, where $\tau_1 = \frac{P_s + N_s}{P_s}$ and $\tau_2 = \frac{P_s + N_s}{N_s}$ are the weights assigned to positive class samples P_s and negative class samples N_s, respectively.

Table 4.1 shows a comparison of the error cost during the fine-tuning phase of our model with cross entropy cost function, and the proposed cost function.

4.3.3 Choice of Hyperparameters

We selected the value of the hyperparameters used in the proposed network based on the performance of the network on the validation set. The parameters and their values are shown in Table 4.2.

Table 4.1 Comparison of model performance using regular cost function (cross entropy) and proposed cost function. The mean, maximum, and minimum error costs are better (smaller is better) with the proposed cost function. With the proposed cost function, the model is tuned to achieve a low missed detection rate

Cost function	Min. error cost	Max. error cost	Mean error cost
Cross entropy	3.53E-03	1.041E+00	6.29E-02
Proposed	6.00E-04	1.10E-02	2.03E-03

Table 4.2 Parameters and values

Parameter	L_0	L_1	L_2	L_3	L_4	L_5	L_6
Number of neurons	81	800	1000	1200	1200	1200	2
Batch size	–	100	100	100	100	100	–
Epochs	–	50	50	50	50	–	–
Learning rate	–	1e-3	5e-4	5e-4	5e-4	–	–
Momentum	–	0.70	0.70	0.70	0.70	–	
Number of iterations	–	–	–	–	–	50	–

4.3.3.1 Unsupervised Pre-training

We adopt unsupervised layer-wise pre-training because of its power in capturing the dominant and statistically reliable features present in the dataset. The output of each layer is a representation of the input data embodying those features. According to [8], greedy layer-wise unsupervised pre-training overcomes the challenges of deep learning by introducing a useful prior to the supervised fine-tuning training procedure. After pre-training a layer, its input sample is reconstructed and the mean square reconstruction error is computed. The reconstruction step entails guessing the probability distribution of the original input sample in a process referred to as generative learning. Unsupervised pre-training promotes input transformations that capture the main variations in the dataset distribution [8]. Since there is a possibility that only a small subset of these variations may be relevant for predicting the class label of a sample, using a small number of nodes in the hidden layers will make it less likely for the transformations necessary for predicting the class label to be included in the set of transformations learned by unsupervised pre-training. This idea is reflected in our choice of the number of nodes in the pre-training layers. We ran several experiments to determine the optimal nodes in each of the three pre-training layers. As shown in Table 4.2, the number of nodes in the pre-training layers L_1, L_2, and L_3 are 800, 1000, and 1200, respectively.

4.3.3.2 Supervised Fine-Tuning

Supervised fine-tuning is the process of backpropagating the gradient of a classifier's cost through the feature extraction layers. Supervised fine-tuning boosts the performance of neural networks with unsupervised pre-training [19]. In our model, supervised fine-tuning is done with a layer of RBM and a single-layer perceptron depicted as L_4 and L_5, respectively in Fig. 4.3. During the fine-tuning phase, we initialized L_4, with the pre-trained weights of the top-most pre-training layer L_3.

4.4 Experiments and Results

We implemented our algorithms in MATLAB R2014a running on Intel Core i7 CPU with 8GB RAM and 750GB hard drive. Our implementation relied on NNBox, a MATLAB toolbox for neural networks. The implementation uses backpropagation, contrastive divergence, Gibbs sampling, and hidden units sparsity based optimization techniques.

4.4.1 Latent Fingerprint Databases

We tested our model on the following databases:
• **NIST SD27:** This database was acquired from the National Institute of Standards and Technology. It contains images of **258** latent crime scene fingerprints and their matching rolled tenprints. The images in the database are classified as good, bad, or ugly based on the quality of the image. The latent prints and rolled prints are at **500** ppi.
• **WVU Database:** This database is jointly owned by West Virginia University and the FBI. It has **449** latent fingerprint images and matching **449** rolled fingerprints. All images in this database are at **1000** ppi.
• **IIITD Database:** The IIITD was obtained from the Image Analysis and Biometrics lab at the Indraprastha Institute of Information Technology, Delhi, India [25]. There are **150** latent fingerprints and **1,046** exemplar fingerprints. Some of the fingerprint images are at **500** ppi while others are at **1000** ppi.

4.4.2 Performance Evaluation and Metrics

We used the following metrics to evaluate the performance of our network.

- **Missed Detection Rate** (***MDR***): This is the percentage of fingerprint patches classified as non-fingerprint patches and is defined as.

$$MDR = \frac{FN}{TP + FN} \tag{4.14}$$

where *FN* is the number of false negatives and *TP* is the number of true positives.

- **False Detection Rate** (***FDR***): This is the percentage of non-fingerprint patches classified as fingerprint patches.It is defined as

$$FDR = \frac{FP}{TN + FP} \tag{4.15}$$

where *FP* is the number of false positives and *TN* is the number of true negatives.

- **Segmentation Accuracy (SA):** It gives a good indication of the segmentation reliability of the model.

$$SA = \frac{TP + TN}{TP + FN + TN + FP} \tag{4.16}$$

4.4.3 Stability of the Architecture

To investigate the stability of the proposed architecture, we performed five runs of training the network using 50,000 training samples. All the model parameters (number of epochs, number of iterations etc.) shown in Table 4.2 remained unchanged across the runs. The mean square reconstruction error (msre), mean error cost, and standard deviation for the five runs are shown in Table 4.3. Plots of the reconstruction errors against number of training epochs as well as that of error cost against number or iterations during each run are shown in Fig. 4.5. These results show that our model is stable.

Table 4.3 Network Stability: The msre, error cost, MDR, FDR, and training accuracy for the five different runs are close. The mean and standard deviation indicate stability across the five runs

Run #	MSRE	Error cost	MDR	FDR	Training accuracy
1	0.0179	5.469e-04	2.010e-04	0.00	4.00e-05
2	0.0183	5.406e-04	3.020e-04	0.00	6.00e-05
3	0.0178	5.560e-04	1.010e-04	0.00	2.00e-05
4	0.0179	5.438e-04	2.020e-04	0.00	5.00e-05
5	0.0178	6.045e-04	1.010e-04	0.00	2.00e-05
Mean	0.0179	5.584e-04	1.814e-04	0.00	3.800e-05
Standard deviation	0.0002	2.643e-05	8.409e-05	0.00	1.789e-05

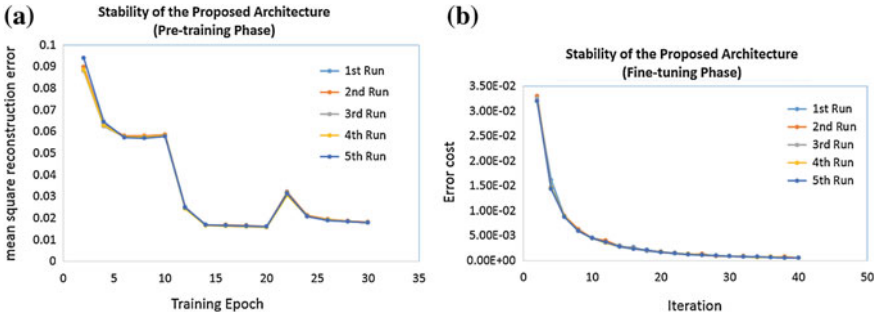

Fig. 4.5 Network Stability: **a** shows that the mean square reconstruction error (msre) followed the same trajectory during the five different runs, converging close to 0.02% msre. Similarly, **b** shows that the error cost during the fine-tuning phase followed the same trajectory for the five runs, converging to about 5.5E-04 error cost. The results are indicative of the stability of the network

4.4.4 Segmentation Using the Trained Network

To segment a latent fingerprint image using the trained network we proceed as follows:

- Split the image into 8×8 nonoverlapping patches and augment each patch data with its fractal dimension and lacunarity features to create a segmentation dataset.
- Normalize the segmentation dataset to have 0 mean and unit standard deviation.
- Load the trained network and compute activation value for each neuron: $a = \sum Wx$
- Feed the activation value to the activation function to normalize it.
- Apply the following thresholding function to obtain the classification results:

$$\theta(x) = \begin{cases} 1 : z > T \\ 0 : z \leq T \end{cases} \tag{4.17}$$

where z is the decision value, x is an example from the segmentation dataset, T is a threshold that gave the best segmentation accuracy on a validation set and was obtained using fivefold cross validation described in Algorithm 1.

4.4.4.1 Searching for Threshold *T*

We implemented a *hook* into the logic of output neurons to access the real-valued output of the activation function. To obtain the percentage of the activation function output for a given neuron, we divided its activation function value by the sum of all activation function values. For each label $y \in 1, 2$, we ordered the validation examples according to their decision values (percentages) and for each pair of adjacent decision values, we checked the segmentation accuracy using their average as T. The algorithm used was inspired by [10], and is shown as Algorithm 1.

Algorithm 1 Searching for threshold T

procedure THRESHOLD(X, Y) ▷ X is the patch dataset, Y is a set of corresponding labels
  $num\_class \leftarrow Unique(Y)$ ▷ Get unique labels from Y
  for $cs = 1, \ldots, num\_class$ **do** ▷ Iterate over the number of classes
    (a) $folds \leftarrow Split(X, 5)$ ▷ Split the validation set into five folds
    for $f = 1, \ldots, folds$ **do** ▷ Iterate over the folds
      (i) Run Compute(.) on four folds of validation set ▷ Run four folds through the trained network
      (ii)$T_c^f \leftarrow Best()$ ▷ Set T_c^f to the decision value that achieved the best *MDR*
    (b) Run Compute(.) on X ▷ Run the validation set through the trained network
    $T_c \leftarrow \frac{1}{5} \sum_{k=1}^{folds} T_c^k$ ▷ Set the threshold to the average of the five thresholds from cross validation
  return T ▷ Return the threshold

4.4.5 Dataset Diffusion and Impact on Model Performance

Given a dataset $X = \{x_1, x_2, \ldots, x_k\}$, we define the diffusion of X as $\hat{X} = \{x_1, x_2, \ldots, x_m\}$, where $m > k$ and each $x_i, k < i < m$ is an element from R^n. In other words, $\hat{X}$ is obtained by *expanding* X with new elements from R^n. A similar idea based on the principle of information diffusion has been used by researchers in situations, where the neural network failed to converge despite adjustments of weights and thresholds [12, 22]. We used features based on fractal dimension and lacunarity to diffuse X. These features help to characterize complex texture in latent fingerprint images [9].

4.4.5.1 Fractal Dimension

Fractal dimension is an index used to characterize texture patterns by quantifying their complexity as a ratio of the change in detail to the change in the scale used. It was defined by Mandelbrot [23] and was first used in texture analysis by Keller et al. [15]. Fractal dimension offers a quantitative way to describe and characterize the complexity of image texture composition [18].

We compute the fractal dimension of an image patch P using a variant of differential box counting (DBC) algorithm [2, 26]. We consider P as a *3-D* spatial surface with *(x,y)* axis as the spatial coordinates and z axis for the gray level of the pixels. Using the same strategy as in *DBC*, we partition the $N \times N$ matrix representing P into nonoverlapping $d \times d$ blocks where $d \in [1, N]$. Each block has a column of boxes of size $d \times d \times h$, where h is the height defined by the relationship $h = \frac{\mathscr{T} d}{N}$, where $\mathscr{T}$ is the total gray levels in P, and d is an integer. Let $\mathscr{T}_{min}$ and $\mathscr{T}_{max}$ be the minimum and maximum gray levels in grid *(i, j)*, respectively. The number of boxes covering block *(i, j)* is given by:

$$n_d(i,j) = floor[\frac{\mathscr{T}_{max} - \mathscr{T}_{min}}{r}] + 1, \tag{4.18}$$

where $r = 2, \ldots, N-1$, is the scaling factor and for each block $r = d$. The number of boxes covering all $d \times d$ blocks is:

$$N_d = \sum_{i,j} n_d(i,j) \tag{4.19}$$

We compute the values N_d for all $d \in [1, N]$. The fractal dimension of each pixel in P is by given by the slope of a plot of the logarithm of the minimum box number as a function of the logarithm of the box size. We obtain a fractal dimension image patch P' represented by an $M \times N$ matrix whose entry *(i, j)* is the fractal dimension FD_{ij} of the pixel at *(i, j)* in P.

$$FD_P = \sum_{i=1,j=1}^{MN} FD_{ij} \tag{4.20}$$

♦ **Fractal Dimension Features**: We implemented a variant of the *DBC* algorithm [2, 26], to compute the following statistical features from the fractal dimension image P'.

• ***Average Fractal Dimension***:

$$FD_{avg} = \frac{1}{MN} \sum_{i=1,j=1}^{MN} FD_{ij} \tag{4.21}$$

• ***Standard Deviation of Fractal Dimension:*** The standard deviation of the gray levels in an image provides a degree of image dispersion and offers a quantitative description of variation in the intensity of the image plane. Therefore

$$FD_{std} = \frac{1}{MN} \sum_{i=1,j=1}^{MN} (FD_{ij} - FD_{avg})^2, \tag{4.22}$$

• ***Fractal Dimension Spatial Frequency:*** This refers to the frequency of change per unit distance across fractal dimension (FD) processed image. We compute it using the formula for (spatial domain) spatial frequency [20]. Given an $N \times N$ FD processed image patch P', let *G(x,y)* be the FD value of the pixel at location *(x,y)* in P'. The row frequency R_f and column frequency C_f are given by

$$R_f = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=1}^{N-1} [G(x,y) - G(x,y-1)]^2} \tag{4.23}$$

$$C_f = \sqrt{\frac{1}{MN}\sum_{y=0}^{M-1}\sum_{x=1}^{N-1}[G(x,y) - G(x-1,y)]^2} \tag{4.24}$$

The FD spatial frequency FD_{sf} of P' is defined as

$$FD_{sf} = \sqrt{R_f^2 + C_f^2} \tag{4.25}$$

From signal processing perspective, Eqs. (4.23) and (4.24) favor high frequencies and yield values indicative of patches with fingerprint.

4.4.5.2 Lacunarity

Lacunarity is a second-order statistic that provides a measure of how patterns fill space. Patterns that have more or larger gaps have higher lacunarity. It also quantifies rotational invariance and heterogeneity. A spatial pattern that has a high lacunarity has a high variability of gaps in the pattern, and indicates a more heterogeneous texture [5]. Lacunarity (FD_{lac}) is defined in terms of the ratio of variance over mean value [2].

$$FD_{lac} = \frac{\frac{1}{MN}(\sum_{i=1}^{M-1}\sum_{j=1}^{N-1}P(i,j)^2)}{\{\frac{1}{MN}\sum_{i=1}^{M-1}\sum_{j=1}^{N-1}P(i,j)\}^2} - 1, \tag{4.26}$$

where M and N are the sizes of the fractal dimension image patch P.

4.4.5.3 Diffusing the Dataset

We followed standard engineering practice to select the architecture of our model. To improve the performance of the model, we tried various data augmentation techniques such as label preserving transformation and increasing/decreasing the number minority/majority samples to balance the dataset. We also tried other learning techniques such as one class learning. None of those techniques yielded the desired segmentation results.

Due to discriminative capabilities of fractal dimension and lacunarity features, we used them to diffuse the patch dataset. From experiments, we observed that by diffusing the dataset with these features before normalizing the data yielded a trained model that has better generalization on unseen examples. A comparison of the results obtained with and without dataset diffusion is shown in Fig. 4.6. As can be seen from Table 4.4, when the training dataset was augmented with FD features, there was a huge drop in both error cost during fine-tuning and the classification error during

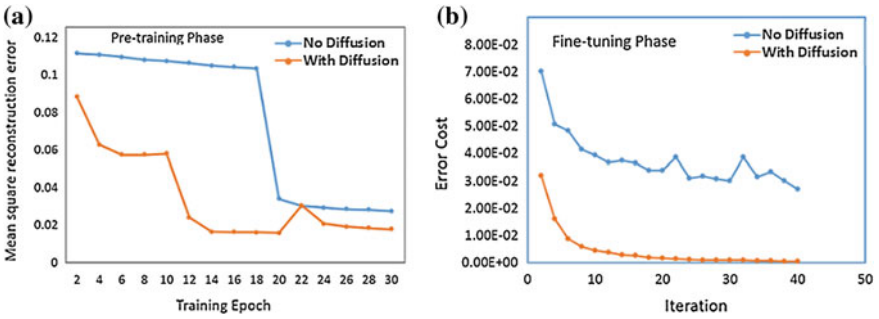

Fig. 4.6 Impact of Data Diffusion on Model Performance. **a** shows that during the pre-training phase, the network achieves lower mean square reconstruction error (msre) when the dataset is diffused with fractal dimension features. Also, as can be seen from **b**, diffusing the dataset leads to faster convergence and lower error cost during the fine-tuning phase

Table 4.4 Data diffusion and network performance

	MSRE	Error cost	Classification error (Training) (%)
Without diffusion	0.0179	7.97e-01	18.51
With diffusion	0.0178	6.0456e-04	0.006

training. It is interesting to note that the reconstruction error almost remained the same in both cases.

4.4.6 Classification and Segmentation Results

4.4.6.1 Training, Validation, and Testing

We studied the performance of our model when trained on one latent fingerprint database and tested on another using 3 sets of 20,000 patches, 40% drawn from good, 30% from bad, and 30% from ugly images from NIST, WVU, and IIITD databases. In each of the three experiments, 10,000 patches from a set were used for training, 4,000 for validation, and 6,000 for testing. The results are shown in Table 4.5.

The final training, validation and testing of the model was done with 233,200 patches from the NIST SD27 database with 40% from good, 30% from bad, and 30% from ugly NIST image categories. 132,000 examples were used for training, 48,000 for validation, and 53,200 for testing. Table 4.6 shows the confusion matrix for NIST SD 27 and Table 4.7 shows the TP, TN, FP, and FN, MDR, FDR and classification accuracy on the training, validation, and testing datasets. There was no

Table 4.5 Model performance when trained and tested on different latent fingerprint databases. The numbers in bracket delimited with colon are the training, validation, and testing datasets, respectively. The three datasets are independent. The training and validation datasets shown in column 1 of the last row were obtained exclusively from NIST SD27 database. The testing sets are independent of the training set and were obtained from the target testing database in column 5. MDR_V and FDR_V are the validation MDR and FDR, respectively. Similarly, MDR_T and FDR_T are the testing MDR and FDR, respectively. As shown in the last row, there was a marked improvement in the model performance when more training data was used. When we tried more than 132,000 patches for training, there was no appreciable performance gain despite more training time required to achieve convergence

Train on	Validate on	MDR_V (%)	FDR_V (%)	Test on	MDR_T (%)	FDR_T (%)
NIST SD27 (10,000 : 4,000 : 6,000)	NIST SD27	2.95	1.92	NIST SD27	3.04	1.98
				WVU	3.75	2.25
				IIITD	3.63	2.19
WVU (10,000 : 4,000 : 6,000)	WVU	3.12	2.54	NIST SD27	3.61	3.01
				WVU	3.22	2.87
				IIITD	3.90	3.05
IIITD (10,000 : 4,000 : 6,000)	IIITD	3.32	2.66	NIST SD27	3.49	3.19
				WVU	3.86	3.16
				IIITD	3.28	2.80
NIST SD27 (132,000 : 48,000 : 53,200)	NIST SD27	1.25	0	NIST SD27	1.25	0
				WVU	1.64	0.60
				IIITD	1.35	0.54

Table 4.6 NIST SD27—Confusion matrix for training, validation, and testing

		Predicted patch class (Training)	
		Fingerprint	Non-Fingerprint
Actual patch class	Fingerprint	23,667	9
	Non-Fingerprint	0	108,324
		Predicted patch class (Validation)	
		Fingerprint	Non-Fingerprint
Actual patch class	Fingerprint	12,946	154
	Non-Fingerprint	0	34,900
		Predicted patch class (Testing)	
		Fingerprint	Non-Fingerprint
Actual patch class	Fingerprint	15,291	193
	Non-Fingerprint	0	37,716

Table 4.7 NIST SD27—Training, Validation and Testing Accuracy: Training: **132,000** 8×8 patches; Validation: **48,000** 8×8 patches; Testing: **53,200** 8×8 patches. $MDR = \frac{FN}{TP+FN}$; $FDR = \frac{FP}{TN+FP}$

	TP	TN	FP	FN	MDR (%)	FDR (%)	Classification accuracy (%)
Training	23,667	108,324	0	9	0.04	0	99.96
Validation	12,946	34,900	0	154	1.17	0	98.83
Testing	15,291	37,716	0	193	1.25	0	98.75

noticeable performance gain when the model was trained with more than 132,000 patches.

4.4.6.2 Segmentation Results

Figures 4.7, 4.8, and 4.9 show the segmentation results of our proposed method on sample good, bad, and ugly quality images from the NIST SD27 database. The figures show the original latent fingerprint images and the segmented fingerprints and non-fingerprints constructed using patches classified as fingerprints and non-fingerprints.

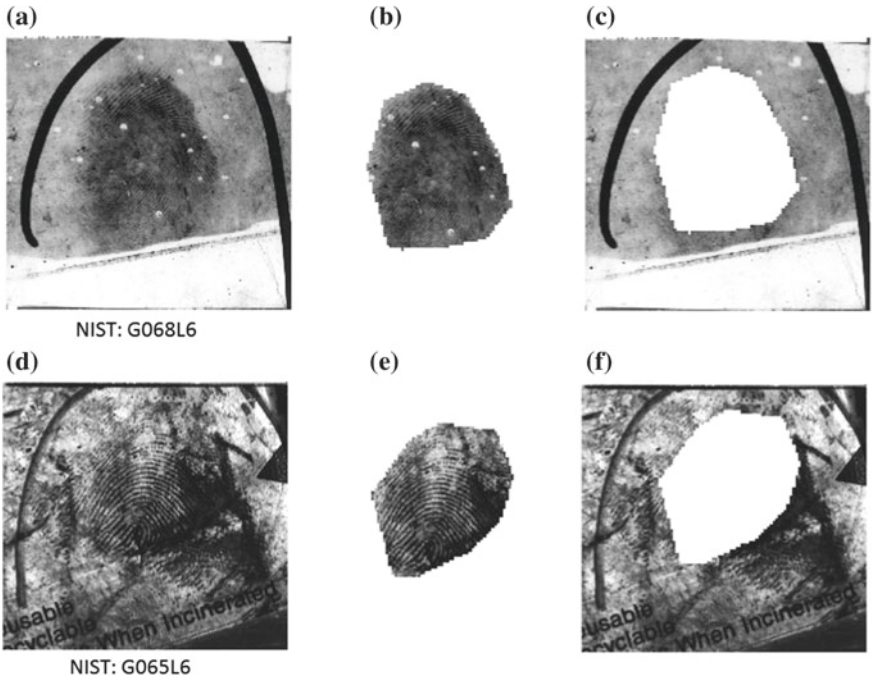

Fig. 4.7 NIST Good Category Latent fingerprint image and segmentation result without post classification processing. **a** and **d** Original images **b** and **e** Fingerprints **c** and **f** Non-fingerprints

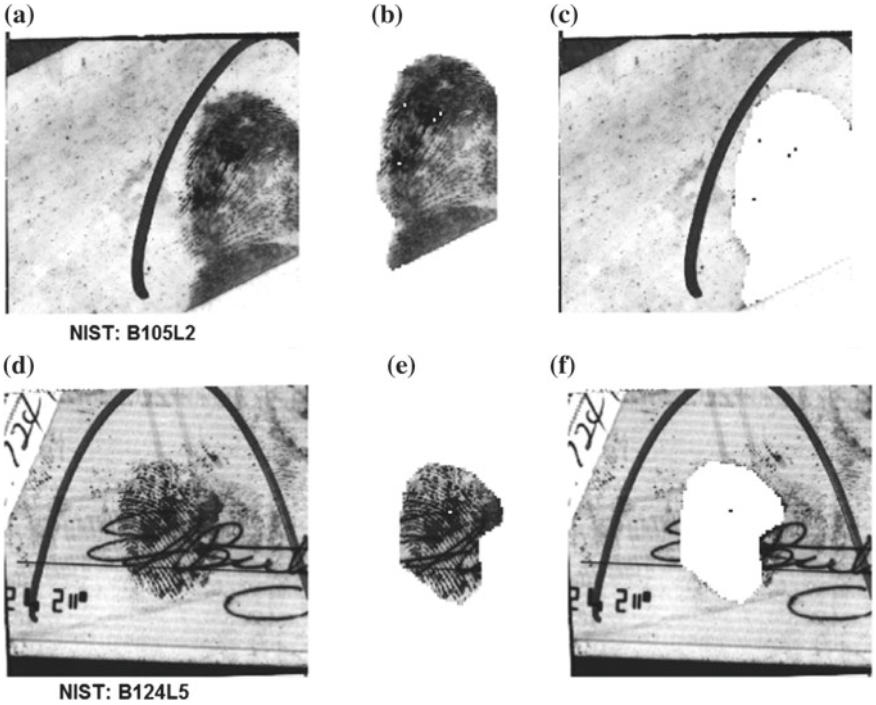

Fig. 4.8 NIST Bad Category [1] Latent Fingerprint Image and segmentation result without post classification processing. **g** and **j** Original images; **h** and **k** Fingerprints **i** and **l** Non-fingerprints

The segmentation results for WVU and IIITD are not shown due to restrictions in the database release agreement (Fig. 4.10).

4.4.7 Comparison with Current Algorithms

Table 4.8 shows the superior performance of our segmentation approach on the good, bad, and ugly quality latent fingerprints from NIST SD27 compared to the results from existing algorithms on the same database. It also shows the performance comparison of our model on WVU and IIITD with other algorithms that reported results on those latent fingerprint databases.

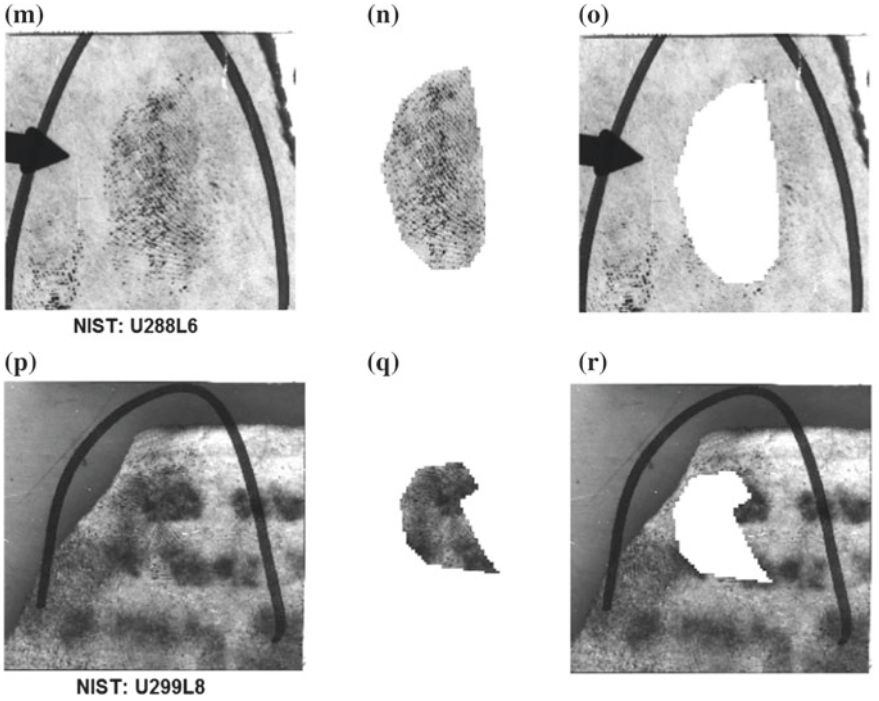

Fig. 4.9 NIST Ugly Category Latent Fingerprint Image and segmentation result without post classification processing. **m** and **p** Original images **n** and **q** Fingerprints **o** and **r** Non-fingerprints

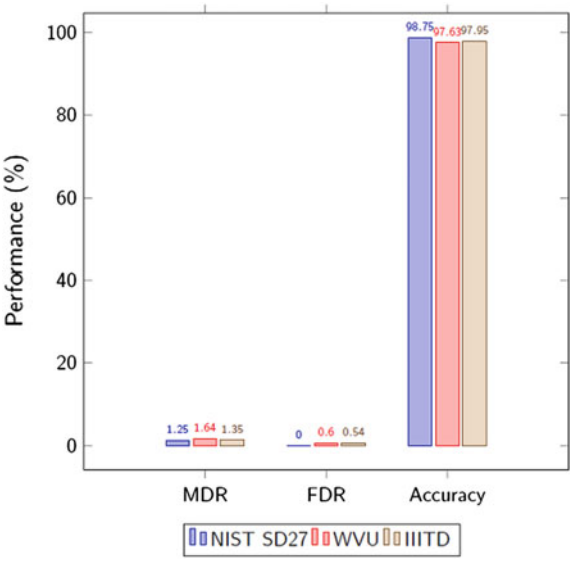

Fig. 4.10 Segmentation reliability in different databases for good quality images. This shows the results of training our model on NIST SD27 and testing on NIST SD27, WVU, and IIITD latent databases. The choice of latent fingerprint database used during training has small impact on the performance of our network. This assertion is also supported by the results in Table 4.5

Table 4.8 Comparison with other algorithms on various datasets

Author	Approach	Database	MDR %	FDR %	Average
Choi et al. [7]	Ridge orientation and frequency computation	NIST SD27	14.78	47.99	31.38
		WVU LDB	40.88	5.63	23.26
Zhang et al. [29]	Adaptive total variation model	NIST SD27	14.10	26.13	20.12
Arshad et al. [4]	K-means clustering	NIST SD27	4.77	26.06	15.42
Jude and Bhanu [9]	Fractal dimension & Weighted ELM	NIST SD27 (Good, Bad, Ugly)	9.22	18.7	13.96
		WVU LDB (Good, Bad, Ugly)	15.54	9.65	12.60
		IIITD LDB (Good)	6.38	10.07	8.23
This chapter	Deep learning	NIST SD27 (Good, Bad, Ugly)	1.25	0.04	0.65
		WVU LDB (Good, Bad, Ugly)	1.64	0.60	1.12
		IIITD (Good)	1.35	0.54	0.95

4.5 Conclusions and Future Work

We proposed a deep architecture based on restricted Boltzmann machine for latent fingerprint segmentation using image patches and demonstrated its performance on the segmentation of latent fingerprint images. The model learns a set of stochastic features that model a distribution over image patches. Using the features extracted from the image patches, the model classifies the patches into fingerprint and non-fingerprint classes. We use the fingerprint patches to reconstruct the latent fingerprint image and discard the non-fingerprint patches which contain the structured noise in the original latent fingerprint. We demonstrated the performance of our model in the segmentation of good, bad, and ugly latent fingerprints from the NIST SD27, as well as WVU and IIITD latent fingerprint databases. We showed that the overall performance of our deep model is superior to that obtained with the state-of-the-art latent fingerprint image segmentation algorithms. Our future work involves developing algorithms for feature extraction and matching for the segmented latent fingerprints.

Acknowledgements This research was supported in part by the Presley Center for Crime and Justice Studies, University of California, Riverside, California, USA.

References

1. NIST Special Database 27. *Fingerprint Minutiae from Latent and Matching Ten-print Images*, http://www.nist.gov/srd/nistsd27.htm
2. O. Al-Kadi, D. Watson, Texture analysis of aggressive and nonaggressive lung tumor CE CT images. IEEE Trans. Biomed. Eng. **55**(7), 1822–1830 (2008)
3. I. Arel, D.C. Rose, T.P. Karnowski, Deep machine learning - a new frontier in artificial intelligence research [research frontier]. IEEE Comput. Intell. Mag. **5**(4), 13–18 (2010)
4. I. Arshad, G. Raja, A. Khan, Latent fingerprints segmentation: feasibility of using clustering-based automated approach. Arabian J. Sci. Eng. **39**(11), 7933–7944 (2014)
5. M. Barros Filho, F. Sobreira, Accuracy of lacunarity algorithms in texture classification of high spatial resolution images from urban areas, in *XXI Congress of International Society of Photogrammetry and Remote Sensing* (2008)
6. M.A. Carreira-Perpinan, G. Hinton, On contrastive divergence learning, in *AISTATS*, vol. 10 (Citeseer, 2005), pp. 33–40
7. H. Choi, A.I.B.M. Boaventura, A. Jain, Automatic segmentation of latent fingerprints, in *2012 IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)* (2012), pp. 303–310
8. D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. **11**, 625–660 (2010)
9. J. Ezeobiejesi, B. Bhanu, Latent fingerprint image segmentation using fractal dimension features and weighted extreme learning machine ensemble, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2016)
10. R.-E. Fan, C.-J. Lin, *A Study on Threshold Selection for Multi-label Classification*. Department of Computer Science (National Taiwan University, 2007), pp. 1–23
11. G. Hinton. A Practical Guide to Training Restricted Boltzmann Machines, Version 1 (2010)
12. C. Huang, C. Moraga, A diffusion-neural-network for learning from small samples. Int. J. Approx. Reason. **35**(2), 137–161 (2004)
13. S. Karimi-Ashtiani, C.-C. Kuo, A robust technique for latent fingerprint image segmentation and enhancement, in *15th IEEE International Conference on Image Processing, 2008, ICIP 2008* (2008), pp. 1492–1495
14. D. Kaye, T. Busey, M. Gische, G. LaPorte, C. Aitken, S. Ballou, L.B. ..., K. Wertheim, Latent print examination and human factors: improving the practice through a systems approach, in *NIST Interagency/Internal Report (NISTIR) - 7842* (2012)
15. J. Keller, S. Chen, R. Crownover, Texture description and segmentation through fractal geometry. Comput. Vis. Graph. Image Process. **45**(2), 150–166 (1989)
16. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
17. M. Lai, Deep learning for medical image segmentation (2015), arXiv:1505.02000
18. A.D.K.T. Lam, Q. Li, Fractal analysis and multifractal spectra for the images, in *2010 International Symposium on Computer Communication Control and Automation (3CA)*, vol. 2 (2010), pp. 530–533
19. P. Lamblin, Y. Bengio, Important gains from supervised fine-tuning of deep architectures on large labeled sets, in *NIPS* 2010 Deep Learning and Unsupervised Feature Learning Workshop* (2010)
20. S. Li, J.T. Kwok, Y. Wang, Combination of images with diverse focuses using the spatial frequency. Inf. Fus. **2**(3), 169–176 (2001)
21. Y. Liu, E. Racah, P.J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, W.D. Collins, Application of deep convolutional neural networks for detecting extreme weather in climate datasets (2016), CoRR abs/1605.01156
22. Z. Makó, Approximation with diffusion-neural-network, in *6th International Symposium of Hungarian Researchers on Computational Intelligence* (2005), pp. 18–19

23. B. Mandelbrot, *The Fractal Geometry of Nature*. Einaudi paperbacks (Henry Holt and Company, New York, 1983)
24. A. Ng, J. Ngiam, C.Y. Foo, http://ufldl.stanford.edu/tutorial/supervised/optimizationstochasticgradientdescent/, *UFLDL Tutorial*
25. A. Sankaran, M. Vatsa, R. Singh, Hierarchical fusion for matching simultaneous latent fingerprint, in *2012 IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)* (2012), pp. 377–382
26. N. Sarkar, B.B. Chaudhuri, An efficient differential box-counting approach to compute fractal dimension of image. IEEE Trans. Syst. Man Cybern. **24**(1), 115–120 (1994)
27. N. Short, M. Hsiao, A. Abbott, E. Fox, Latent fingerprint segmentation using ridge template correlation, in *4th International Conference on Imaging for Crime Detection and Prevention 2011 (ICDP 2011)* (2011), pp. 1–6
28. I. Yildirim, Bayesian inference: Gibbs sampling. Technical Note, University of Rochester (2012)
29. J. Zhang, R. Lai, C.-C. Kuo, Latent fingerprint segmentation with adaptive total variation model, in *2012 5th IAPR International Conference on Biometrics (ICB)* (2012), pp. 189–195

Chapter 5
Finger Vein Identification Using Convolutional Neural Network and Supervised Discrete Hashing

Cihui Xie and Ajay Kumar

Abstract Automated personal identification using vascular biometrics, such as from the finger vein images, is highly desirable as it helps to protect the personal privacy and anonymity in during the identification process. The Convolutional Neural Network (CNN) has shown remarkable capability for learning biometric features that can offer robust and accurate matching. We introduce a new approach for the finger vein authentication using the CNN and supervised discrete hashing. We also systematically investigate comparative performance using several popular CNN architectures in other domains, i.e., Light CNN, VGG-16, Siamese and the CNN with Bayesian inference-based matching. The experimental results are presented using a publicly available two-session finger vein images database. Most accurate performance is achieved by incorporating supervised discrete hashing from a CNN trained using the triplet-based loss function. The proposed approach not only achieves outperforming results over other considered CNN architecture available in the literature but also offers significantly reduced template size as compared with those over the other finger vein images matching methods available in the literature to date.

5.1 Introduction

Automated personal identification using unique physiological characteristics of humans, like face, fingerprint, or iris, is widely employed for e-security in a range of applications. In the past decade, there has been significant increase in the detection of surgically altered fingerprints, fake iris stamps, or the usage of sophisticated face masks, to thwart integrity of deployed biometrics systems. Vascular biometrics identification, like using finger vein patterns which are located at about three millimetres below the skin surface, can help to preserve the integrity of biometrics system as it is extremely difficult to surgically alter vascular biometrics. Another advantage of

C. Xie · A. Kumar (✉)
Department of Computing, The Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong
e-mail: ajay.kumar@polyu.edu.hk

B. Bhanu and A. Kumar (eds.), *Deep Learning for Biometrics*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-61657-5_5

using finger vein biometrics is related to high degree of personal privacy as finger vein patterns are hidden underneath the skin surface and extremely difficult to acquire then covertly.

The possibility of personal identification using vascular patterns imaged using the light transmitted through hands was indicated in 1992 [6], but was not known to be demonstrated until 2000 [7]. Such earliest work demonstrated feasibility of finger vein identification using normalized-cross correlation. Miura et al. [12] later introduced repeated line tracking approach to improve the performance of finger vein identification, and they further enhanced the performance with maximum curvature [13]. Kumar and Zhou [8] introduced first publicly accessible finger vein database in public domain and comparatively evaluated a range of handcrafted features for the finger vein identification problem. The method introduced in [8] using Gabor filter-based enhancement and morphological operations is still regarded the best performing methods for matching finger vein images. A range of handcrafted features [2, 3, 8–13, 21], primarily obtained from the careful evaluation of the registered images, have been introduced in the literature to investigate finger-vein identification performance. Multiple features acquired from the two cameras [10] or using multiple feature extractors [22] can be combined to significantly improve performance for the vascular matching. One of the limitations of finger vein identification methods introduced in the literature is related to their large template size. Smaller template size is desirable to reduce storage and/or enhance the matching speed for the mobile and online applications. There have also been successful attempts to reduce the finger vein template size, like in [3, 9] or recently in [2] using sparse representation of enhanced finger vein images using the Gabor filters.

The finger vein matching methods available in the literature to date have judiciously introduced handcrafted features and demonstrated promising performance. However, the remarkable capabilities of the deep learning algorithms in automatically learning the most relevant vascular features are yet to be investigated or established. The objective of this work is to fairly investigate the effectiveness of self-learned features using popular convolutional neural network (CNN) architectures and develop more efficient and effective alternative for the automated finger vein identification. The experimental results and comparisons detailed in this chapter used light CNN [20], modified VGG-16 [16], CNN with Bayesian inference, and Siamese network with triplet loss function. Our reproducible [5] experimental results using *publicly* available database indicate that supervised discrete hashing in conjunction with CNN not only achieves outperforming results, but also significantly reduce the finger vein template size which offers increased matching speed. Table 5.1 in the following summarizes promising methods for the finger vein matching that have been introduced in the literature. This table also presents the template size in respective reference, which has been estimated from the details provided in respective reference, performance in terms of EER, and the database used for the performance evaluation. Reference [4] provides good summary of publicly available finger vein image databases introduced in the literature. The usage of two-session databases, particularly for the less constrained or contactless imaging setups as in [8], generates high intra-class variations and is highly desirable to generate fair evaluation of the matching algorithms

Table 5.1 Comparative summary of handcrafted finger vein features in the literature with this work

Ref.	Feature	Database	Two session	Template size (bytes)	EER	No. of subjects	No. of genuine scores	No. of impostor scores
[8]	Handcrafted (Even Gabor)	Public	Yes	106384	4.61%	105	1260	263,340
[8]	Handcrafted (Morphological)	Public	Yes	6710	4.99%	105	1260	263,340
[12]	Handcrafted (Repeated line tracking)	Proprietary	No	43200*	0.145%	678	678*	458,328*
[13]	Handcrafted (Maximum curvature)	Proprietary	No	43200*	0.0009%	678	678*	458,328*
[3]	Handcrafted (Local binary pattern)	Public	No	≤260*	3.53%	156	624	194,064
[2]	Handcrafted (Sparse representation using l1-norm)	Proprietary	No	630*	Unknown	17	Unknown	Unknown
[9]	Handcrafted (Extended local binary pattern)	Public	Yes	131328*	7.22%	105	1260	263,340
[21]	Handcrafted (Unknown algorithm)	Proprietary	Yes	20480*	0.77%	Unknown	10,000	499,500
[11]	Handcrafted (Histogram of salient edge orientation map)	Public	No	≤3496*	0.9%	100	3000	1,797,000
Ours	CNN with triplet similarity loss	Public	Yes	1024	13.16%	105	1260	263,340
Ours	Supervised discrete hashing with CNN	Public	Yes	250	9.77%	105	1260	263,340

*Computed by us from the details available in the respective reference

under more realistic usage/environments. Similarly, the usage of publicly available database can ensure reproducibility of results. Therefore, our all experiments in this chapter incorporate two-session and publicly available database from [8]. The last two rows in this table summarize best performing results from our investigation detailed in this work [19].

The rest of this chapter is organized as follows. Section 5.2 briefly describes on the preprocessing of the finger vein images and includes relevant steps for the image normalization, segmentation and enhancement. The CNN architectures, LCNN,

VGG, LCNN with triplet similarity loss function, and LCNN with joint Bayesian formulation investigated in this are introduced in Sect. 5.3 while Sect. 5.4 details the supervised discrete hashing algorithm investigated to enhance performance and reduce the template size. The experimental results are presented in Sect. 5.4 and includes discussion on our findings, comparison with earlier methods. Finally, the key conclusions from this work are summarized in Sect. 5.5.

5.2 Image Normalization for Finger Vein Images

5.2.1 Preprocessing

Acquisition of finger vein images can introduce translational and rotational changes in different images acquired from the same finger or subject. Therefore, automated extraction of fixed region of interest (ROI) that can minimize such intra-class variations is highly desirable. The method of ROI localization considered in this work is same as detailed in [8] as it works well in most cases. Figure 5.1 illustrates samples of the acquired images using the near infrared camera.

Once the region of interest is localized, we can recover the binary masks corresponding to the ROI which can be used for alignment of finger vein samples, so that the adverse influence from the rotational changes in fingers can be minimized. The method for estimating the rotation is same as described in [8]. This estimated angle is used to align ROI, before the segmentation, in a preferred direction.

5.2.2 Image Enhancement

The finger vein details from the normalized images are subjected to the contrast enhancement to enhance clarity in vascular patterns which can be more reliable for

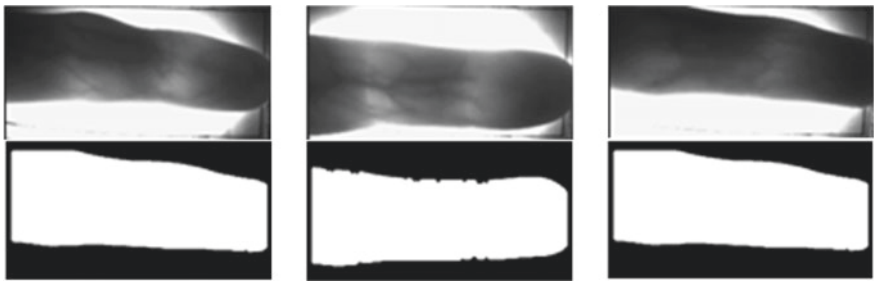

Fig. 5.1 Finger vein image samples before preprocessing (*first row*) and the binary images generated during the preprocessing (*second row*) stage

the training. Since vascular patterns are generally continuous, if a pixel belongs to the vascular pattern, there is a high possibility that its surrounding pixels are also part of the same vascular pattern and have similar gray level. Such observation is the same for nonvascular parts. Therefore, enhancement by computing the average gray level surrounding a pixel can help to enlarge the difference between the vascular parts and nonvascular parts, and makes the finger vein details more obvious as a result. After such enhancement, the vascular patterns become clearer with details as shown from sample images in Fig. 5.2.

The vascular patterns in the normalized image samples can be further enhanced by spatial filtering from orientation selective band pass filters, similar to as used in the enhancement of fingerprint images. We also attempted to ascertain usefulness of such enhanced finger vein images using the Gabor filters. These filters from the twelve different orientations are selected to generate enhanced finger vein images as shown in Fig. 5.4. Such enhanced images [8] using Gabor filters are effective in accentuating the vascular features and therefore its possible usage in automatically vascular features (Fig. 5.3) from CNN was also investigated in the experiments.

The finger vein image-processing operations essentially generates two kinds of enhanced images, ROI-A and ROI-B shown in Fig. 5.4, that were considered for the performance evaluation using the CNNs.

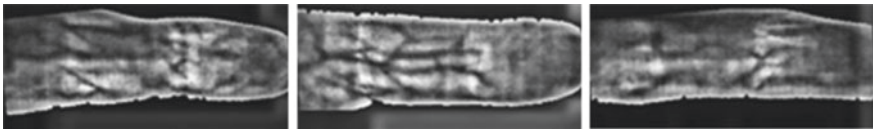

Fig. 5.2 Enhanced ROI vein images after rotation

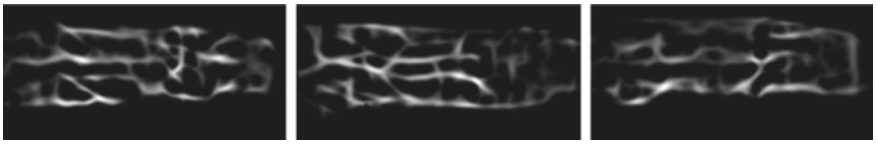

Fig. 5.3 Samples from even Gabor filtered finger vein images

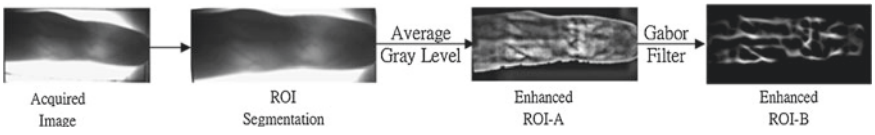

Fig. 5.4 Key steps in the generation of enhanced finger vein images for the CNNs

5.3 Convolutional Neural Network Architectures

A variety of models for the deep learning have been developed to learn useful feature representation but largely for the face biometric image patterns. A variety of such models using CNN have been introduced in the literature and were investigated to ascertain performance for the finger vein image matching. A brief introduction to various CNN architectures considered in this work is provided in the following sections.

5.3.1 Light CNN

The light CNN (LCNN) framework introduces a Max-Feature-Map (MFM) operation [20] between convolutional layers which establish a competitive relationship for superior generalization capability and reduce parameter space (compact feature representation). Such *maxout* activation (Fig. 5.5) function significantly reduces complexity and makes CNN lighter, where *conv* stands for convolutional layer.

For a convolutional layer without MFM, suppose that input size is $N_1 \times W_1 \times H_1$ and output size is $N_2 \times W_2 \times H_2$ then the required complexity using big 'O' notation can be represented as follows:

$$O\left(N_1 N_2 \Omega\right) \ \ where\ \Omega = W_1 \times H_1 \times W_2 \times H_2 \tag{5.1}$$

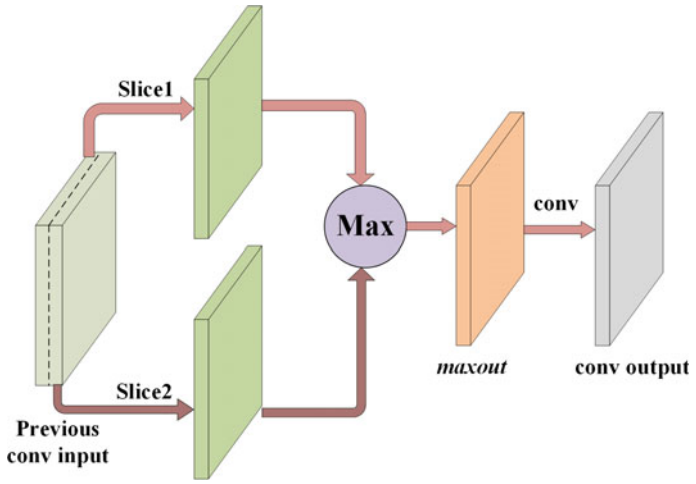

Fig. 5.5 Illustration for computing the Max-Feature Map in LCNN

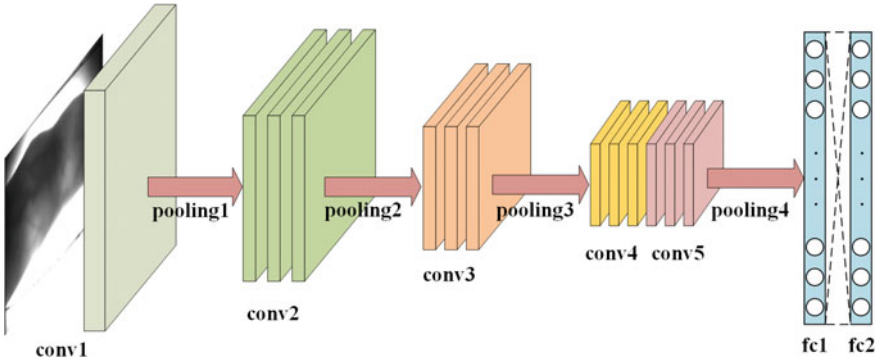

Fig. 5.6 The architecture for LCNN investigated in our experiments

For a convolutional layer with MFM, we could slice input into two equal-size parts, each with $\frac{N_1}{2} \times W_1 \times H_1$. Then for each corresponding element in two sliced parts we generate an output using *maxout* activation, which has a size of $\frac{N_1}{2} \times W_1 \times H_1$. With this smaller or lighter data as the input, complexity of convolutional layer reduces to

$$O\left(\frac{N_1 N_2 \Omega}{2}\right) \quad where\ \Omega = W_1 \times H_1 \times W_2 \times H_2 \tag{5.2}$$

Comparing (5.1) and (5.2) we can infer that the usage of MFM can help to significantly reduce the complexity or make CNN lighter.

The loss function we used in this structure is *softmax* loss function. The basic idea is to combine *softmax* function with a negative log-likelihood, and the last layer information is used to estimate the identity of the class.

The architecture of LCNN employed in our experiments is shown in Fig. 5.6 (MFM part is excluded to maintain the clarity). This network contains 9 convolutional layers (conv), 4 pooling layers (pooling) and 2 fully connected layers (fc) and some assistant layers which are summarized in Table 5.2.

5.3.2 LCNN with Triplet Similarity Loss Function

Deep Siamese networks have been successfully incorporated to learn a similarity metric between a pair of images. We incorporated similar triplet similarity loss function as detailed in [14] for LCNN to learn the similarity metric.

We randomly select an image x^r from training set as *random* sample in Fig. 5.7. Then, we choose image x^p which is from the same class referred to as *positive* sample and image x^n which is from a different class referred to as *negative* sample. After LCNN, we get the features $f(x^r)$, $f(x^n)$, and $f(x^p)$. Our objective is to decrease the

Table 5.2 Details of layer information of LCNN

Index	Type	Filter size	Num	Stride	Pad
1	conv1	5	96	1	2
2	MFM1	–	48	–	–
3	pooling1	2	–	2	0
4	conv2a	1	96	1	0
5	MFM2a	–	48	–	–
6	conv2	3	192	1	1
7	MFM2	–	96	–	–
8	pooling2	2	–	2	0
9	conv3a	1	192	1	1
10	MFM3a	–	96	–	–
11	conv3	3	384	1	1
12	MFM3	–	192	–	–
13	pooling3	2	–	2	0
14	conv4a	1	384	1	1
15	MFM4a	–	192	–	–
16	conv4	3	256	1	1
17	MFM4	–	128	–	–
18	conv5a	1	256	1	1
19	MFM5a	–	128	–	–
20	conv5	3	256	1	1
21	MFM5	–	128	–	–
22	pooling4	2	–	2	0
23	fc1	–	512	–	–
24	MFMfc	–	256	–	–
25	fc2	–	500	–	–

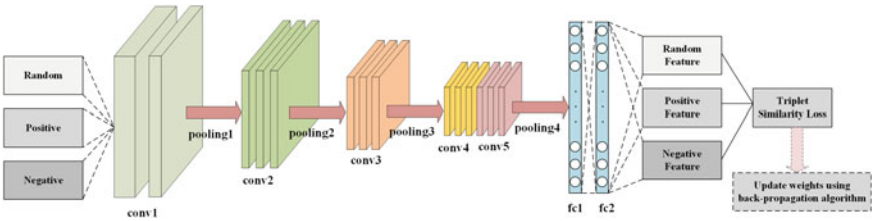

Fig. 5.7 The architecture for LCNN with triplet similarity loss function

similarity distance between *random* and *positive* features, and increase it between *random* and *negative* features, which indicates why it is named as triplet similarity loss. At the same time, we also need to ensure that there is a sufficient *margin* between them.

Suppose we have a random set $\mathbf{X^r} = \{x_i^r\}_{i=1}^N$ and its corresponding positive set $\mathbf{X^p} = \{x_i^p\}_{i=1}^N$ and negative set $\mathbf{X^n} = \{x_i^n\}_{i=1}^N$. Considering these notations, we can write our loss function as follows:

$$\sum_{i=1}^{N} [\| f\left(x_i^r\right) - f\left(x_i^p\right) \|^2 - \| f\left(x_i^r\right) - f\left(x_i^n\right) \|^2 + margin]_+ \tag{5.3}$$

where $[\cdot]_+$ presents that we maintain positive values and change others to zero. The architecture of LCNN with such triplet similarity loss function is shown in Fig. 5.7 and detailed in Table 5.3. When the set of input consists of n random samples, n positives and n negatives, we generate $3n \times 500$ features. These pairs were split into three parts, each with the size of $n \times 500$, and used as the input for computing triplet similarity loss for updating the neuron weights during the network training.

5.3.3 Modified VGG-16

The Visual Geometry Group architecture with 16 layers (VGG-16) [16] was modified for the CNN to directly recover the match scores, instead of the feature vectors, in our experiment. Our modification was motivated to fit the rectangular finger vein ROI images without introducing the distortion. We used pair of images rather than single image as input in conventional VGG-16 since we want to compare the similarity between two finger vein images. The input image size is also different from conventional VGG-16, which is 224×224, while it's 128×488 pixels for our finger vein ROI images. The training phase utilized the cross-entropy loss function which can be written as follows:

$$-\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log\left(\widehat{\boldsymbol{y}}_{\boldsymbol{l}}\right) + (1 - y_i) \log\left(1 - \widehat{\boldsymbol{y}}_{\boldsymbol{l}}\right) \right] \tag{5.4}$$

where $\widehat{\boldsymbol{y}}_{\boldsymbol{l}} = g(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{i}}) g(\cdot)$ is the logistic function, $\boldsymbol{x_i}$ is the extracted feature and $\mathbf{w}$ is the weight that needs optimized during the training. The architecture of Modified VGG-16 (MVGG) is shown in Fig. 5.8 and Table 5.4.

5.3.4 LCNN with Joint Bayesian Formulation

The principal component analysis (PCA) is a classical method to extract the most important features and is popular for the dimensionality reduction of the features. In another set of experiments, we incorporated PCA for the dimensionality reduction of features extracted from LCNN and then employed joint Bayesian [1] approach as distance metrics for matching finger vein images.

For any feature $\boldsymbol{f}$ extracted from LCNN, we regard it as combination of two parts $\boldsymbol{\mu}$ and $\boldsymbol{\varepsilon}$ where $\boldsymbol{\mu}$ is the average feature of the class to which $\boldsymbol{f}$ belongs and $\boldsymbol{\varepsilon}$ is

Table 5.3 Details of layer information of LCNN with triplet similarity loss

Index	Type	Input index	Output index	Filter size	Output size	Stride	Pad
1	DataR	–	4	–	N*W*H	–	–
2	DataP	–	4	–	N*W*H	–	–
3	DataN	–	4	–	N*W*H	–	–
4	Data	1,2,3	5	–	3N*W*H	–	–
5	conv1	4	6	5	3N*96	1	2
6	MFM1	5	7	–	3N*48	–	–
7	pooling1	6	8	2	–	2	0
8	conv2a	7	9	1	3N*96	1	0
9	MFM2a	8	10	–	3N*48	–	–
10	conv2	9	11	3	3N*192	1	1
11	MFM2	10	12	–	3N*96	–	–
12	pooling2	11	13	2	–	2	0
13	conv3a	12	14	1	3N*192	1	1
14	MFM3a	13	15	–	3N*96	–	–
15	conv3	14	16	3	3N*384	1	1
16	MFM3	15	17	–	3N*192	–	–
17	pooling3	16	18	2	–	2	0
18	conv4a	17	19	1	3N*384	1	1
19	MFM4a	18	20	–	3N*192	–	–
20	conv4	19	21	3	3N*256	1	1
21	MFM4	20	22	–	3N*128	–	–
22	conv5a	21	23	1	3N*256	1	1
23	MFM5a	22	24	–	3N*128	–	–
24	conv5	23	25	3	3N*256	1	1
25	MFM5	24	26	–	3N*128	–	–
26	pooling4	25	27	2	–	2	0
27	fc1	26	28	–	3N*512	–	–
28	MFMfc	27	29	–	3N*256	–	–
29	fc2	28	30,31,32	–	3N*500	–	–
30	SliceR	29	33	–	N*500	–	–
31	SliceP	29	33	–	N*500	–	–
32	SliceN	29	33	–	N*500	–	–
33	Loss	30,31,32	–	–	–	–	–

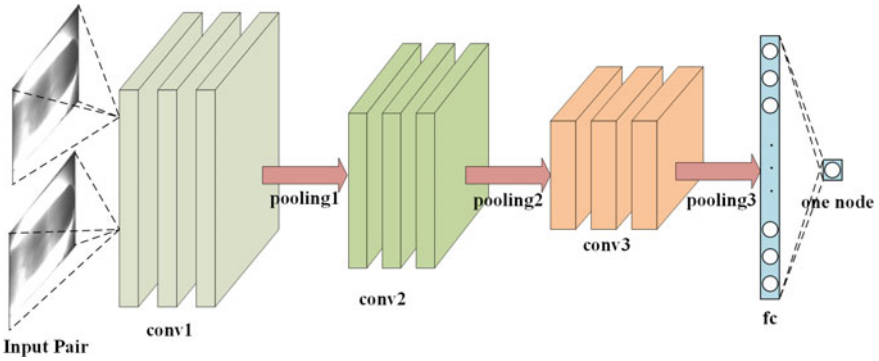

Fig. 5.8 The architecture for Modified VGG-16 for finger vein image matching

Table 5.4 Details of layer information of Modified VGG-16

Index	Type	Filter size	Num	Stride	Pad
1	conv1a	3	64	1	1
2	ReLU1a	–	–	–	–
3	conv1b	3	64	1	1
4	ReLU1b	–	–	–	–
5	conv1c	3	64	1	1
6	ReLU1c	–	–	–	–
7	pooling1	2	–	2	0
8	conv2a	3	128	1	1
9	ReLU2a	–	–	–	–
10	conv2b	3	128	1	1
11	ReLU2b	–	–	–	–
12	conv2c	3	128	1	1
13	ReLU2c	–	–	–	–
14	pooling2	2	–	2	0
15	conv3a	3	256	1	1
16	ReLU3a	–	–	–	–
17	conv3b	3	256	1	1
18	ReLU3b	–	–	–	–
19	conv3c	3	256	1	1
20	ReLU3c	–	–	–	–
21	pooling3	2	–	2	0
22	fc1	–	512	–	–
23	Dropout	–	–	–	–
24	fc2	–	1	–	–

the intra-class variation, and we suppose that $\boldsymbol{\mu}$ and $\boldsymbol{\varepsilon}$ are two independent variables with Gaussian distribution $N(0, \boldsymbol{\sigma}_{\boldsymbol{\mu}})$ and $N(0, \boldsymbol{\sigma}_{\boldsymbol{\varepsilon}})$.

Let I be the hypothesis that $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ are from the same class, and E mean that they are from different class. Thus we could write the goal as to enlarge $\frac{P(f_1 f_2|I)}{P(f_1 f_2|E)}$ where $P(\cdot)$ is the distribution. For simplicity, we use log formulation $r(\boldsymbol{f}_1, \boldsymbol{f}_2) = \log \frac{P(f_1 f_2|I)}{P(f_1 f_2|E)}$ and the computations are as detailed in [1].

Our objective has been to enlarge $r(\boldsymbol{f}_1, \boldsymbol{f}_2)$ and therefore we compute maximum of results rather than the minimum while using the L2-norm. The CNN training part is the same as LCNN. After extraction of features, we retain 80 dimensions instead of original 500 to accelerate computations and use these features to compute matrices for the joint Bayesian formulation based classification.

5.4 Supervised Discrete Hashing

One of the key challenges for the successful usage of biometrics technologies are related to efficient search speed (fast retrieval) and template storage/size. Hashing is one of the most effective approaches to address such challenges and can *efficiently* encode the biometrics templates using binary numbers (2000 in our experiments) that closely reflect similarity with the input data/templates. With such strategy we can only store the corresponding short/compact binary codes, instead of original feature templates, and significantly improve the search or the matching speed by highly efficient pairwise comparisons using the Hamming distance.

This framework for an effective supervised hashing scheme is introduced in [15] and the objective in the learning phase is to generate binary codes for the linear classification. We firstly define the problem and assume that we have n samples/features $\mathbf{X} = [\mathbf{x}_1 \mathbf{x}_2 \ldots \mathbf{x}_n]$ and our goal is to recover corresponding binary codes $\mathbf{B} = [\mathbf{b}_1 \mathbf{b}_2 \ldots \mathbf{b}_n]$ where $\mathbf{b}_i \in \{-1, 1\}, i = 1, 2, \ldots, n$. Since we have labels, in order to make good use of these information, we define a multi-class classification function:

$$\hat{\mathbf{y}} = \mathbf{W}^{\mathrm{T}} \mathbf{b} \text{ where } \mathbf{W} = [\mathbf{w}_1 \mathbf{w}_2 \ldots \mathbf{w}_C], \tag{5.5}$$

where C is the total number of classes, and $\hat{\mathbf{y}} \in \mathbb{R}^{C \times 1}$ is the label vector, where the maximum one indicates its class of input $\mathbf{x}$. Now we can formulate the hashing problem as follows:

$$\min_{\boldsymbol{B}, \boldsymbol{W}, F} \sum\nolimits_{i=1}^{n} L(\boldsymbol{y}_i, \mathbf{W}^{\mathbf{T}} \mathbf{b}_i) + \lambda \|\mathbf{W}\|^2, \text{ s.t. } \mathbf{b}_i = \operatorname{sgn}(F(\mathbf{x}_i)) \tag{5.6}$$

where $L(\cdot)$ represents the loss function used by us which is the L2-norm in our experiments, λ is the regularization parameter, and at the same time, $\mathbf{b}_i$ is generated by the hash function $\operatorname{sgn}(F(\mathbf{x}_i))$ where $\operatorname{sgn}(\cdot)$ is the sign function. With the help of Lagrange Multiplier, we can then rewrite (5.6) as:

$$\min_{\boldsymbol{B},\, \boldsymbol{W},\, F} \sum_{i=1}^{n} L(\boldsymbol{y}_i, \mathbf{W}^{\mathrm{T}}\mathbf{b}_i) + \lambda\|\mathbf{W}\|^2 + \mu \sum_{i=1}^{n} \|\mathbf{b}_i - F(\mathbf{x}_i)\|^2 \tag{5.7}$$

where μ is the Lagrange multiplier. We further select a nonlinear form of function for $F(\mathbf{x})$:

$$F(\mathbf{x} = \mathbf{U}^{\mathrm{T}}\phi(\mathbf{x}) \tag{5.8}$$

where $\mathbf{U}$ is the parameter matrix and $\phi(\mathbf{x})$ is a k-dimensional kernel that

$$\phi(\mathbf{x}) = \begin{bmatrix} \exp(\|\frac{\mathbf{x}-a_1\|^2}{\sigma}) \\ \cdots \\ \exp(\|\frac{\mathbf{x}-a_k\|^2}{\sigma}) \end{bmatrix} \tag{5.9}$$

$\boldsymbol{a}_j,\ j = 1, 2, \ldots, k$ are randomly selected anchor vectors from input. In order to compute $\boldsymbol{U}$ in the function, we can rewrite (5.6) as follows:

$$\min_{\mathbf{U}} \sum_{i=1}^{n} \|\mathbf{b}_i - F(\mathbf{x}_i)\|^2 = \min_{\mathbf{U}} \|\mathbf{U}^{\mathrm{T}}\Phi(\mathbf{X}) - \mathbf{B}\|^2 \tag{5.10}$$

where $\Phi(\mathbf{X}) = \{\phi(\mathbf{x}_i)\}_{i=1}^{n}$ and our purpose is to set the gradient to zero, which is

$$\nabla_{\mathbf{U}} \left(\|\mathbf{U}^{\mathrm{T}}\Phi(\mathbf{X}) - \mathbf{B}\|^2\right) = 2\left(\mathbf{U}^{\mathrm{T}}\Phi(\mathbf{X}) - \mathbf{B}\right)\Phi(\mathbf{X})^{\mathrm{T}} = 0 \tag{5.11}$$

It is simpler to achieve the final computation for $\mathbf{U}$ as follows:

$$\mathbf{U} = (\Phi(\mathbf{X})\,\Phi(\mathbf{X})^{\mathrm{T}})^{-1}\Phi(\mathbf{X})\,\mathbf{B}^{\mathrm{T}} \tag{5.12}$$

In order to solve for $\mathbf{W}$, we make use of the same method, first simplify (5.6) to

$$\min_{\mathbf{W}} \sum_{i=1}^{n} L(\boldsymbol{y}_i, \mathbf{W}^{\mathbf{T}}\mathbf{b}_i) + \lambda\|\mathbf{W}\|^2 = \min_{\mathbf{W}} \|\mathbf{Y} - \mathbf{W}^{\mathrm{T}}\mathbf{B}\|^2 + \lambda\|\mathbf{W}\|^2, \tag{5.13}$$

and then calculate its gradient based on $\mathbf{W}$

$$\nabla_{\mathbf{W}}(\|\mathbf{Y} - \mathbf{W}^{\mathrm{T}}\mathbf{B}\|^2 + \lambda\|\mathbf{W}\|^2) = 2\mathbf{B}\left(\mathbf{B}^{\mathrm{T}}\mathbf{W} - \mathbf{Y}^{\mathrm{T}}\right) + 2\lambda\mathbf{W}, \tag{5.14}$$

which can be set as zero and we get

$$\mathbf{W} = \left(\mathbf{B}\mathbf{B}^{\mathrm{T}} + \lambda\mathbf{I}\right)^{-1}\mathbf{B}\mathbf{Y}^{\mathrm{T}}, \tag{5.15}$$

Finally, we can solve for $\mathbf{B}$ and we exclude those variables which have no relation to $\mathbf{B}$ and then rewrite (5.6) as follows.

$$\min_{\mathbf{B}} \|\mathbf{Y} - \mathbf{W}^{\mathrm{T}}\mathbf{B}\|^2 + \mu\|\mathbf{B} - F(\mathbf{X})\|^2 = \min_{\mathbf{B}} \|\mathbf{Y}\|^2 - 2\mathrm{tr}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{B}\right) + \|\mathbf{W}^{\mathrm{T}}\mathbf{B}\|^2 + \mu(\|\mathbf{B}\|^2 + 2\mathrm{tr}\left(\mathbf{B}^{\mathrm{T}}F(\mathbf{X})\right) + \|F(\mathbf{X})\|^2) \tag{5.16}$$

or we can further simplify the above formulation as follows:

$$\min_{\mathbf{B}} \|\mathbf{W}^{\mathrm{T}}\mathbf{B}\|^2 - 2\mathrm{tr}\left(\mathbf{B}^{\mathrm{T}}(F(\mathbf{X}) + \mathbf{WY})\right) \tag{5.17}$$

$\|\mathbf{B}\|^2$ is excluded here because $\mathbf{b}_i \in \{-1, 1\}, i = 1, 2, \ldots, n$, indicating that $\|\mathbf{B}\|^2$ is some constant.

We can now solve this problem *bit-by-bit*. Let $\mathbf{p}^{\mathrm{T}}$ represent the lth row of $\mathbf{B}$, and $\mathbf{B}'$ is the matrix without $\mathbf{p}^{\mathrm{T}}$. Similarly let $\mathbf{v}^{\mathrm{T}}$ be the lth row of $\mathbf{W}$ and let $\mathbf{q}^{\mathrm{T}}$ be the lth row of $\mathbf{Q}$ where $\mathbf{Q} = F(\mathbf{X}) + \mathbf{WY}$ then we can ignore $\mathbf{W}'$ and $\mathbf{Q}'$ While moving a row to the end for all matrices would not cause problems but help to better understand the problem. In order to enhance clarity of the problem, we can move all the lth row to the end and rewrite $\mathbf{B} = \begin{bmatrix} \mathbf{B}' \\ \mathbf{p}^{\mathrm{T}} \end{bmatrix}$, and the same for $\mathbf{W}$ and $\mathbf{Q}$ We can then rewrite first term in (5.17) as follows.

$$\begin{aligned} \|\mathbf{W}^{\mathrm{T}}\mathbf{B}\|^2 &= \left\| \begin{bmatrix} \mathbf{W}'^{\mathrm{T}} & \mathbf{v} \end{bmatrix} \begin{bmatrix} \mathbf{B}' \\ \mathbf{p}^{\mathrm{T}} \end{bmatrix} \right\|^2 = \|\mathbf{W}'^{\mathrm{T}}\mathbf{B}'\|^2 + \|\mathbf{v}\mathbf{p}^{\mathrm{T}}\|^2 + 2\mathrm{tr}\left(\mathbf{B}'^{\mathrm{T}}\mathbf{W}'\mathbf{v}\mathbf{p}^{\mathrm{T}}\right) \\ &= \|\mathbf{W}'^{\mathrm{T}}\mathbf{B}'\|^2 + \mathrm{tr}\left(\mathbf{v}\mathbf{p}^{\mathrm{T}}\mathbf{p}\mathbf{v}^{\mathrm{T}}\right) + 2\left(\mathbf{W}'\mathbf{v}\right)^{\mathrm{T}}\mathbf{B}'\mathbf{p} \end{aligned} \tag{5.18}$$

While $\|\mathbf{W}'^{\mathrm{T}}\mathbf{B}'\|^2 + \mathrm{tr}\left(\mathbf{v}\mathbf{p}^{\mathrm{T}}\mathbf{p}\mathbf{v}^{\mathrm{T}}\right)$ is equal to some constant, and because our goal is to solve for $\mathbf{p}$ we can regard other parts as constant. Here $\|\mathbf{v}\mathbf{p}^{\mathrm{T}}\|^2$ is ignored because $\|\mathbf{v}\mathbf{p}^{\mathrm{T}}\|^2 = \|\mathbf{p}\mathbf{v}^{\mathrm{T}}\|^2 = \mathrm{tr}\left(\mathbf{v}\mathbf{p}^{\mathrm{T}}\mathbf{p}\mathbf{v}^{\mathrm{T}}\right) = n\,\mathrm{tr}\left(\mathbf{v}\mathbf{v}^{\mathrm{T}}\right)$ where $\mathbf{p}^{\mathrm{T}}\mathbf{p} = n$. The other part can be simplified as follows:

$$\mathrm{tr}\left(\mathbf{B}^{\mathbf{T}}\mathbf{Q}\right) = \mathrm{tr}\left(\mathbf{B}'^{\mathbf{T}}\mathbf{Q}' + \mathbf{p}\mathbf{q}^{\mathrm{T}}\right) = \mathrm{tr}\left(\mathbf{B}'^{\mathbf{T}}\mathbf{Q}'\right) + \mathrm{tr}(\mathbf{p}\mathbf{q}^{\mathrm{T}}) = \mathrm{tr}(\mathbf{B}'^{\mathbf{T}}\mathbf{Q}') + \mathbf{q}^{T}\boldsymbol{p} \tag{5.19}$$

Combining these terms, we can rewrite (5.17) as follows.

$$\min_{\mathbf{B}} \mathbf{v}^{\mathrm{T}}\mathbf{W}'^{\mathrm{T}}\mathbf{B}'\mathbf{p} - \mathbf{q}^{\mathrm{T}}\mathbf{p} = \min_{\mathbf{B}} (\mathbf{v}^{\mathrm{T}}\mathbf{W}'^{\mathrm{T}}\mathbf{B}' - \mathbf{q}^{\mathrm{T}})\mathbf{p} \tag{5.20}$$

This is an optimization problem, and $\mathbf{p} \in \{-1, 1\}^n$, therefore we just need to incorporate the opposite sign for its first argument.

$$\mathbf{p} = -\text{sgn}\left(\mathbf{v}^{\mathrm{T}}\mathbf{W}'^{\mathrm{T}}\mathbf{B}' - \mathbf{q}^{\mathrm{T}}\right) = \text{sgn}(\mathbf{q}^{\mathrm{T}} - \mathbf{v}^{\mathrm{T}}\mathbf{W}'^{\mathrm{T}}\mathbf{B}') \tag{5.21}$$

We can now *explicitly* outline computed parts in the following.

$$\mathbf{W} = \left(\mathbf{B}\mathbf{B}^{\mathrm{T}} + \lambda\mathbf{I}\right)^{-1}\mathbf{B}\mathbf{Y}^{\mathrm{T}}, \tag{5.15}$$

$$\mathbf{U} = (\Phi(\mathbf{X})\,\Phi(\mathbf{X})^{\mathrm{T}})^{-1}\Phi(\mathbf{X})\,\mathbf{B}^{\mathrm{T}} \tag{5.12}$$

$$\mathbf{p} = \text{sgn}(\mathbf{q}^{\mathrm{T}} - \mathbf{v}^{\mathrm{T}}\mathbf{W}'^{\mathrm{T}}\mathbf{B}') \tag{5.21}$$

Shen et al. [15] have provided another computation based on the hinge loss. However for the simplicity, we incorporated L2-norm in our experiments and therefore this part is excluded here. We now have the required equations here and can *summarize* this algorithm as follows.

Algorithm: Supervised Discrete Hashing

Input: Training data $\{\mathbf{X}, \mathbf{Y}\}$
Output: Binary codes $\mathbf{B}$

1. Randomly select k anchors $\boldsymbol{a}_j$, $j = 1, 2, \ldots, k$ from $\mathbf{X}$ and calculate $\Phi(\mathbf{X})$
2. Randomly initiate $\mathbf{B}$
3. Loop until converge or reach maximum iterations

   – Calculate $\mathbf{W}$ and $\mathbf{U}$ which are described in (5.15) and (5.12)
   – Learn $\mathbf{B}$ bit by bit, with the help of (5.21)

5.5 Experiments and Results

This section provides details on the experiments performed using various CNN architectures discussed in previous sections.

5.5.1 Database and Evaluation Protocol

In order to ensure reproducibility of experimental results, we utilized publicly available *two-session* finger vein images database from [17]. This database of 6264 images has been acquired from 156 different subjects and includes finger vein images from

two fingers for each subject. However, second session images are only from 105 different subjects. This database has high intra-class variations in the images and includes significant variations in the quality of images which makes it most suitable for benchmarking the finger vein matching algorithms for the real applications. In our experiments, we only used first session images to train different network architectures discussed in previous section, and initially excluded 51 subjects without second session images. The experimental results are presented using independent second session test data. Therefore, each of the receiver operating characteristics uses 1260 (210×6) genuine scores and 263340 ($210 \times 209 \times 6$) impostor scores.

We experimented on ROI images, enhanced ROI images, and even Gabor- filtered images separately. The ROI images have 256×513 pixels, enhanced ROI images have 218×488 pixels, and even Gabor filtered images have 218×488 pixels. The experimental results using ROC and CMC from the respective CNN architecture are presented in the following.

5.5.2 *Results Using LCNN*

We first performed the experiments using the LCNN trained using the ROI images, enhanced ROI images, and the enhanced images using even Gabor filters (Figs. 5.2, 5.3, and 5.4). The experimental results using respective second session dataset are shown in Fig. 5.9. The respective ROC and CMC illustrate that *enhanced* ROI images can achieve superior matching performance than those from using ROI images. The enhanced ROI images using even Gabor filters significantly helps to *suppress* the noisy pixels and accentuate the vascular regions and is the plausible reason for superior accuracy.

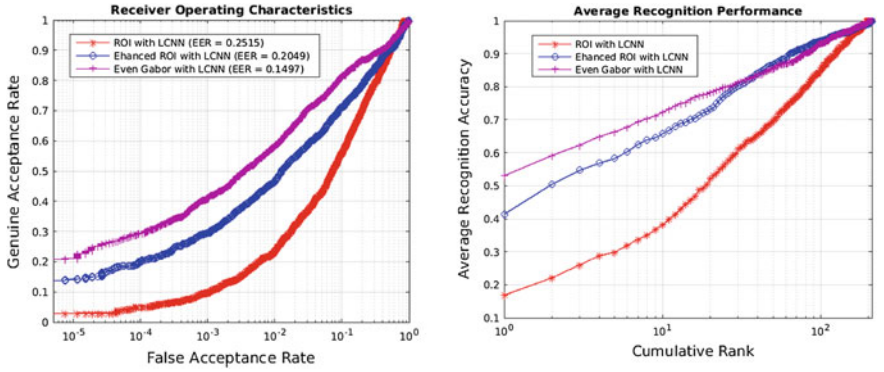

Fig. 5.9 Comparative ROC (*left*) and CMC (*right*) performance using LCNN

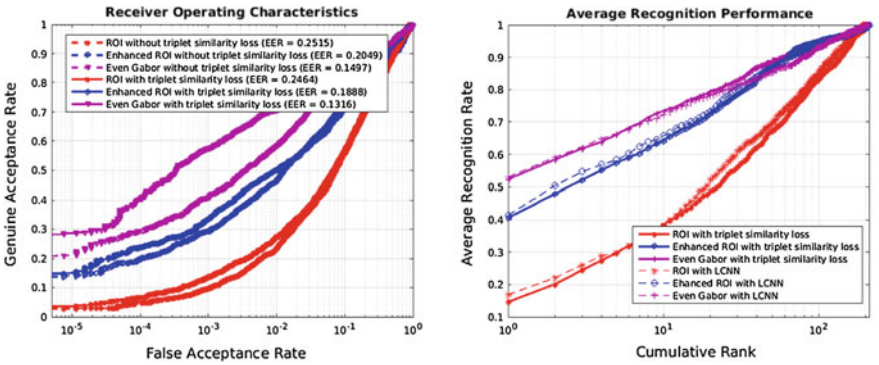

Fig. 5.10 Comparative ROC (*left*) and CMC (*right*) performance using LCNN with triplet similarity loss

5.5.3 *Results Using LCNN with Triplet Similarity Loss Function*

The experimental results using LCNN trained with Siamese triplet similarity loss function are presented in Fig. 5.10. These results consistently illustrate superior performance using the architecture than the LCNN. The performance from the ROC of enhanced ROI with Gabor filters is *significantly* superior and this observation is in line with the trend observed from results using LCNN in Fig. 5.9 (the dash lines in Fig. 5.10 are previous results using LCNN for ease in the comparison). However, this approach has little influence on CMC.

The LCNN *without* triplet similarity loss tries to match a sample with its label, while LCNN *with* similarity loss focuses on the similarities of the images which could contribute to the better ROC performance. However, at the same time, label information is not sufficiently exploited with the triplet similarity loss and therefore the CMC performance has not changed significantly.

5.5.4 *Results Using CNN and Joint Bayesian Formulation*

Another scheme that has shown superior performance for ocular classification in [23] uses joint Bayesian [1] instead of L2-norm as the metrics for the similarity. The LCNN with the joint Bayesian classification scheme was also attempted to ascertain the performance. The ROC using this approach is illustrated in Fig. 5.11 where dash lines are previous results using LCNN and indicates performance improvement over LCNN.

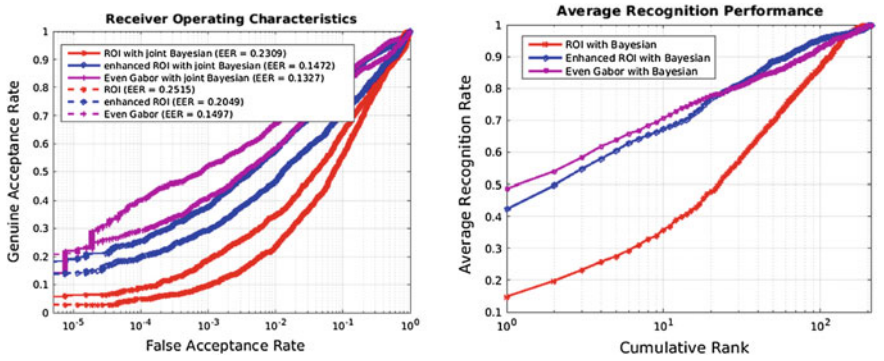

Fig. 5.11 Comparative ROC (*left*) and CMC (*right*) performance using LCNN with Joint Bayesian

5.5.5 *Comparisons and Results Using Supervised Discrete Hashing*

The supervised discrete hashing (SDH) scheme detailed in Sect. 5.4 was also investigated for the performance improvement. Only *first* session data was employed for the training part and the used for generating binarized bits that were used for matching using Hamming distance. The results using the ROC and CMC in Fig. 5.12 illustrates *consistent* performance improvement with the usage of SDH and the trends in the usage of enhanced ROI images are also consistent with our earlier observations.

The LCNN trained with triplet similarity loss function was also employed and used with the SDH to evaluate the performance. We attempted to ascertain the performance with different number of bits. Higher number of bits for SDH can be generally expected to offer superior results, but requires more training time. It should be noted that this method is actually a second-step training, and tries to map features from

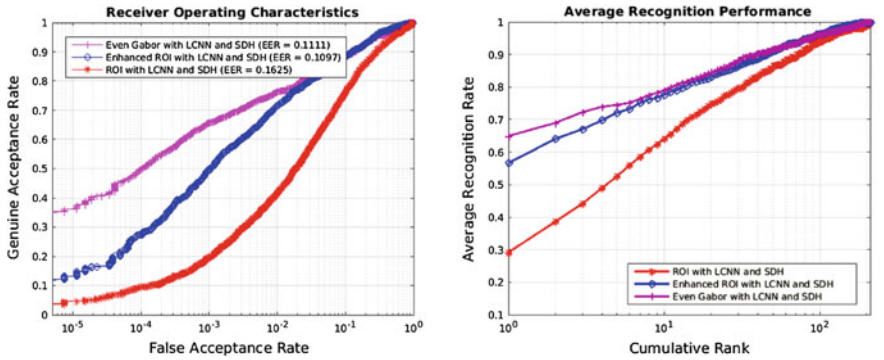

Fig. 5.12 Comparative ROC (*left*) and CMC (*right*) performance using LCNN with SDH

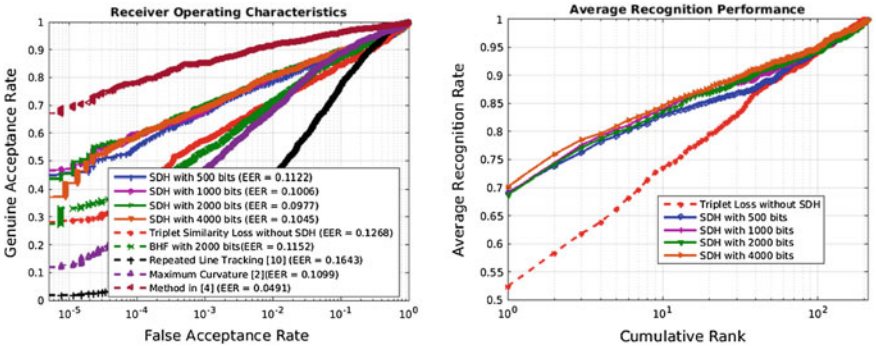

Fig. 5.13 Comparative ROC (*left*) and CMC (*right*) performance using SDH with triplet similarity loss function

the Euclidian space to the binary space. The training phase and hamming distance metrics can contribute to its superior performance. The combination of CNN with the hashing to reduce for the faster and real-time identification has also been attempted earlier [18] but for the face recognition problem. Authors in [18] introduced incorporated Boosted Hashing Forest (BHF) for the hashing and therefore we also attempted to incorporate BHF scheme to comparatively evaluate the performance. However, our results illustrated superiority of SDH over BHF and the template size using BHF was also fixed to 2000 bits. Although our results did not achieve significant improvement in the performance using BHF, its usage helps in remarkably reducing the template size. In order to ascertain comparative performance for matching finger vein images using the handcrafted features, we also performed additional experiments. The ROC from the same test images and matching protocols but using repeated line tracking [1] (also used as baseline in [21]) and curvatures [13] method is illustrated in Fig. 5.13. We can observe that the experimental results using SDH and LCNN offer superior performance and significantly reduced template size. Our results over the method using [8] are can be considered as competing and not yet superior but offers significantly reduced template size (∼26 times smaller) over the best of the methods in [8] which is still the state-of-the-art method to date.

5.5.6 Results Using Modified VGG-16

The experimental results using modified VGG-16, as detailed in Sect. 5.3.3 are presented in Fig. 5.14. It should be noted that this CNN architecture generates single match score and therefore we cannot use SDH scheme to the infer features. We can infer from the ROCs in Fig. 5.14, that modified VGG-16 architecture generates superior performance for matching finger vein images as compared to the network trained using LCNN. This architecture directly generates the matching scores and

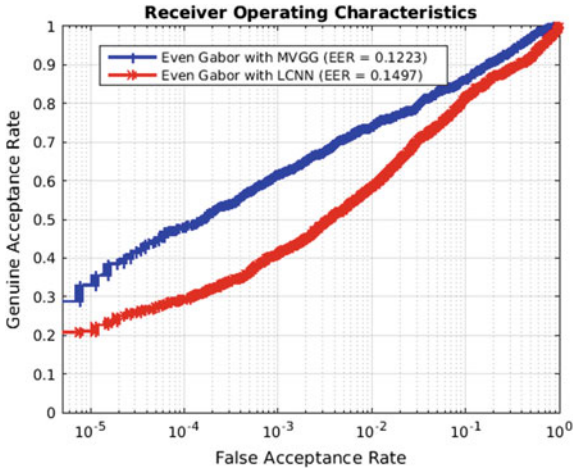

Fig. 5.14 Comparative performance using SDH with MVGG

therefore can minimize the problems from the inappropriate choice of distance metric (or weighting of features), which may be a convincible reason for its superior performance/results. It should however be noted that different training of the network can lead to variations in the results.

5.5.7 Results Using Single Fingers and Unregistered Fingers

Since we used both finger vein images from two fingers to form larger dataset (as in [8]) for above experiments, it is judicious to ascertain the performance when only one, i.e., index or middle, finger vein images are used in the training and test phase. The results using respective finger vein images from 105 different subjects are comparatively shown in Fig. 5.15 using the ROC. The performance using the images from both fingers (using 210 class formed by combination of index and middle finger for 105 different subjects) is superior to single finger, and index finger shows better performance than middle finger. Similar trends are also observed in [8] and can be attributed to the nature of dataset.

In earlier experiments, the first session data had images acquired from the same subjects who were providing their images during the second session and were used as test set for the performance. In order to ascertain robustness of self-learned features using the best scheme so far, we also evaluated the performance from the 51 subjects in this dataset which did not have any two-session finger vein images Therefore, images from none of these subject's images were employed during the training for CNN in any of the earlier experiments. The six images from these 51 subjects were used to ascertain performance using challenging protocol, i.e., all-to-all, so that we generated a total of 1530 genuine scores and 185436 impostor scores to ascertain such performance. The ROC corresponding to this independent test subjects finger

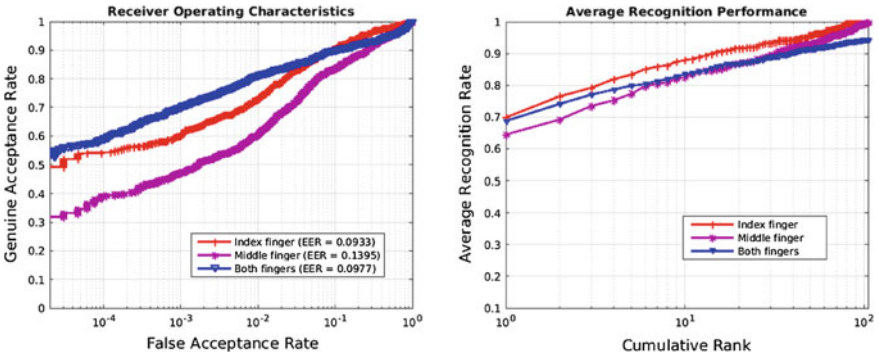

Fig. 5.15 Comparative ROC (*left*) and CMC (*right*) performance using SDH with triplet similarity loss experimented on single finger

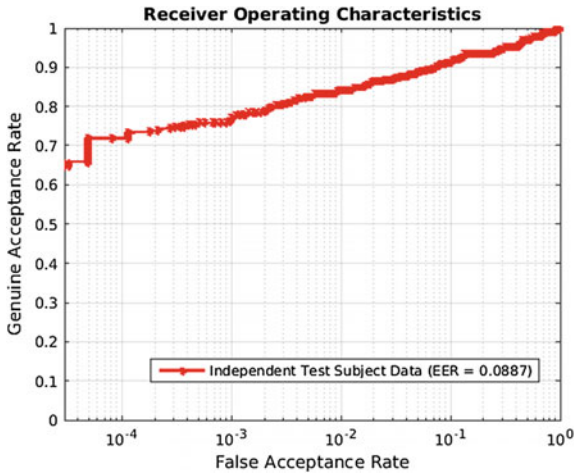

Fig. 5.16 The performance using independent test subjects in [17] for matching finger vein images

vein data is shown in Fig. 5.16 and the results indicate promising performance from the self-learned features using a model trained (in Sect. 5.3.2 and SDH) for matching finger vein images from unknown subjects.

5.6 Discussion

This chapter has investigated finger vein matching performance using various convolutional neural network architectures. Unlike earlier work on finger vein image matching which largely employed handcrafted features, our emphasis has been to investigate automatically learned features using the capabilities of deep learning. We systematically investigated the performance improvement using just the ROI images

and the systematically enhanced images that mainly emphasizes on subsurface vascular network. Our results consistently indicate superior performance from the CNN that are trained with images which have such enhanced vascular features.

According to our experimental results in Sect. 5.5.6, modified VGG-16 (MVGG) achieves superior performance than LCNN. However, MVGG requires significantly higher time for the training (also for the test phase). This can be largely attributed to the fact that it directly generates the match scores and therefore the loss function outputs propagate iteratively through the whole network to ascertain the similarity between a pair of finger vein images. At the same time, we cannot incorporate SDH (hashing scheme) with the MVGG, due to nonavailability of intermediate features, while the usage of SDH has shown to offer remarkable improvement in the performance.

It should be noted that the triplet similarity loss function helps to significantly improve the experimental performance using the LCNN. However, this approach cannot adequately make use of the label information, because it attempts to decrease the feature similarity between the pairwise images from the same subject, but cannot accurately locate the labels, i.e., identity of the subjects they are associated with. Supervised discrete hashing approach significantly improves performance and the retrieval speed, and decrease the storage which requires only 250 bytes (2000 bits) for the one template (feature vector). However, it should also be noted that this method needs a separate training phase and training time rapidly increases when the bit length or number of features are increased.

The work detailed in this chapter also had several constraints and therefore should be considered only preliminary. The database employed, although one of the largest two-session finger vein databases available in public domain, is still of smaller size for the deep learning based algorithms. There are several references in the literature that have shown promising performance but yet to demonstrate superior matching performance over the method in [8] using fair comparison or the same matching protocols. Therefore, we are justified in using the performance from [8], for this publicly available dataset, as the reference.

5.7 Conclusions and Further Work

This chapter has systematically investigated finger vein identification using the various CNN architectures. Unlike earlier work on finger vein matching which employed handcrafted features, our emphasis has been to investigate performance from the automatically learned features using the capabilities of deep learning. We systematically investigated the performance improvement using the ROI finger vein images, and the enhances images and consistently observed that the usage of ROI images with enhanced vascular features and attenuation of background (noise) can significantly improve the performance. The factors that most influence the accuracy of matching finger vein images is the depth of the network, the pretraining and the data augmentation in terms of random crops and rotations.

The LCNN architecture detailed in Sect. 5.3.2 that uses triplet similarity loss function achieves superior performance than those from the original *softmax* loss. Our experimental results using SDH illustrates that this hashing method significantly improves the matching performance and offers better alternative than BHF for the finger vein identification problem. Apart from two-session identification on same subjects, we also experimented on the datasets from subjects whose data was never available for training the CNN and this remarkable performance indicates high generalization capability for the finger vein identification problem. Our proposal for finger vein identification detailed in this chapter achieves smallest template size than using any other methods available in the literature to date. This work however had several constraints and therefore should be considered only preliminary. The database employed, although the largest two session finger vein images database available in public domain, is still of smaller size for the deep learning algorithms. Despite promising improvements in the accuracy from the publicly available (limited) training data, more work needs to be done to achieve significantly superior results than using the best performing method in [8]. Further work should use larger training dataset but should provide performance using the independent test data or using the publicly available dataset [17] to achieve more accurate alternative for the automated finger vein identification.

References

1. D. Chen, X. Cao, L. Wang, F. Wen, J. Sun, Bayesian face revisited: a joint formulation, in *Proceedings of ECCV 2012* (2012)
2. L. Chen, J. Wang, S. Yang, H. He, A finger vein image-based personal identification system with self-adaptive illuminance control. IEEE Trans. Instrum. Meas. **66**(2), 294–304 (2017)
3. L. Dong, G. Yang, Y. Yin, F. Liu, X. Xi, Finger vein verification based on a personalized best patches map, in *Proceedings of 2nd IJCB 2014*, Tampa, Sep.–Oct. 2014
4. F. Hillerstorm, A. Kumar, R. Veldhuis, Generating and analyzing synthetic finger-vein images, in *Proceedings of BIOSIG 2014*, Darmstadt, Germany, September 2014
5. https://polyuit-my.sharepoint.com/personal/csajaykr_polyu_edu_hk/_layouts/15/guestaccess.aspx?docid=11d706337994749759a2cc64cb70b604a&authkey=AcDq15geZ7942-7owNQfmYQ
6. M. Kono, H. Ueki, S. Umemura, A new method for the identification of individuals by using of vein pattern matching of a finger, in *Proceedings of 5th Symposium on Pattern Measurement*, pp. 9–12 (*in Japanese*), Yamaguchi, Japan, 2000
7. M. Kono, H. Ueki, S. Umemura, Near-infrared finger vein patterns for personal identification. Appl. Opt. **41**(35), 7429–7436 (2002)
8. A. Kumar, Y. Zhou, Human identification using finger images. IEEE Trans. Image Process. **21**, 2228–2244 (2012)
9. C. Liu, Y.H. Kim, An efficient finger-vein extraction algorithm based on random forest regression with efficient local binary patterns, in *2016 IEEE ICIP* (2016)
10. Y. Lu, S. Yoon, D.S. Park, Finger vein identification system using two cameras. Electron. Lett. **50**(22), 1591–1593 (2014)
11. Y. Lu, S. Yoon, S.J. Xie, J. Yang, Z. Wang, D.S. Park, Efficient descriptor of histogram of salient edge orientation map for finger vein recognition. Optic. Soc. Am. **53**(20), 4585–4593 (2014)

12. N. Miura, A. Nagasaka, T. Miyatake, Feature extraction of finger-vein patterns based on repeated line tracking and its application to personal identification, in *Machine Vision & Applications*, pp. 194–203, Jul, 2004
13. N. Miura, A. Nagasaka, T. Miyatake, Extraction of finger-vein patterns using maximum curvature points in image profiles, in *Proceedings of IAPR Conference on Machine Vision and Applications*, pp. 347–350, Tsukuba Science City, May 2005
14. F. Schroff, D. Kalenichenko, J. Philbin, Facenet: a unified embedding for face recognition and clustering (2015), arXiv:1503.03832
15. F. Shen, C. Shen, W. Liu, H.T. Shen, Supervised Discrete Hashing, in *Proceedings of CVPR, 2015*, pp. 37-45, Boston, June 2015
16. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2014), arXiv:1409.1556
17. The Hong Kong Polytechnic University Finger Image Database (Version 1.0) (2012), http://www.comp.polyu.edu.hk/~csajaykr/fvdatavase.htm
18. Y. Vizilter, V. Gorbatsevich, A. Vorotnikov, N. Kostromov, Real-time face identification via CNN and boosted hashing forest, in *Proceedings of CVPR 2016 Biometrics Workshop, CVPR'W 2016*, Las Vegas, June 2016
19. C. Xie and A. Kumar, 'Finger vein identification using convolutional neural networks', The Hong Kong Polytechnic University, Tech. Report No. COMP K-25, Jan 2017
20. X. Wu, R.He, Z. Sun, T. Tan, A light CNN for deep face representation with noisy labels (2016), arXiv:1151.02683v2
21. Y. Ye, L. Ni, H. Zheng, S. Liu, Y. Zhu, D. Zhang, W. Xiang, FVRC2016: the 2nd finger vein recognition competition, in *2016 ICB* (2016)
22. Y. Zhou, A. Kumar, Contactless palmvein identification using multiple representations, in *Proceedings of BTAS 2010*, Washington DC, USA, September 2010
23. Z. Zhao, A. Kumar, Accurate periocular recognition under less constrained environment using semantics-assisted convolutional neural network. IEEE Trans. Info. Forensics Secur. **12**(5), 1017–1030 (2017)

Chapter 11
Learning Representations for Cryptographic Hash Based Face Template Protection

Rohit Kumar Pandey, Yingbo Zhou, Bhargava Urala Kota and Venu Govindaraju

Abstract In this chapter, we discuss the impact of recent advancements in deep learning in the field of biometric template protection. The representation learning ability of neural networks has enabled them to achieve state-of-the-art results in several fields, including face recognition. Consequently, biometric authentication using facial images has also benefited from this, with deep convolutional neural networks pushing the matching performance numbers to all time highs. This chapter studies the ability of neural networks to learn representations which could benefit template security in addition to matching accuracy. Cryptographic hashing is generally considered most secure form of protection for the biometric data, but comes at the high cost of requiring an exact match between the enrollment and verification templates. This requirement generally leads to a severe loss in matching performance (FAR and FRR) of the system. We focus on two relatively recent face template protection algorithms that study the suitability of representations learned by neural networks for cryptographic hash based template protection. Local region hashing tackles hash-based template security by attempting exact matches between features extracted from local regions of the face as opposed to the entire face. A comparison of the suitability of different feature extractors for the task is presented and it is found that a representation learned by an autoencoder is the most promising. Deep secure encoding tackles the problem in an alternative way by learning a robust mapping of face classes to secure codes which are then hashed and stored as the secure template. This approach overcomes several pitfalls of local region hashing and other face template algorithms. It also achieves state-of-the-art matching performance with a high standard of template security.

R.K. Pandey (✉) · Y. Zhou · B.U. Kota · V. Govindaraju
Univeristy of Buffalo, SUNY, Buffalo, NY 14260, USA
e-mail: rpandey@buffalo.edu

Y. Zhou
e-mail: yingbozh@buffalo.edu

B.U. Kota
e-mail: buralako@buffalo.edu

V. Govindaraju
e-mail: govind@buffalo.edu

B. Bhanu and A. Kumar (eds.), *Deep Learning for Biometrics*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-61657-5_11

11.1 Introduction

Privacy is a growing concern in today's world given the large digital footprint we leave behind on a day-to-day basis. This may be in the form of web browsing history, photos we share via social media, or passwords we register on applications and websites. Given the sensitive nature of most of this data, there are concerns of it falling into the wrong hands. Modern cryptography provides good solutions for the protection of sensitive information such as passwords and other textual data, but there are still forms of sensitive data that remain challenging to secure. This chapter focuses on the protection of one critical such data type; face biometric templates.

Biometric authentication is increasingly finding its way into our daily lives through face and fingerprint recognition systems in mobile phones as well as computers. Authentication based on "who we are" as opposed to "something we remember" or "something we possess" offers convenience and often, stronger system security. Typically, a biometric authentication system extracts and stores a template from the user's biometric data during the enrollment phase. During verification, the user's biometric is presented and a template is extracted and matched to the stored template. Depending on the matching score, access is granted or denied. One crucial, and often overlooked, aspect to such authentication is the protection of the stored template. If an attacker comes in possession of the stored template, not only can he gain access to the system, but also possibly recover the original biometric data of the user from it. Given that physical biometric features like face, fingerprint, or iris are not replaceable, compromising them would prevent the user from using them for any biometric authentication in the future.

11.2 Password Protection

Before getting into biometric template protection, let us briefly consider the most common form of authentication today; the string password. During enrollment, a template is extracted from the presented text and stored in the database. For text password authentication, the template is a one way non-invertible transform in the form of a hash digest generally computed using a cryptographic hash function like SHA. During verification, the password is presented again, its hash digest is calculated and compared to the stored hash digest. If the two passwords matched exactly, their hash digests would match as well, and access would be granted. If an attacker comes in possession of the template, the properties of hash functions like SHA make sure that no information is revealed about the original password. In fact, the attacker would have to brute force through each possible text password in order to find the one that corresponds to the template. Furthermore, it is straightforward to ask the user to change their password if the database breach is detected. Such hash-based template protection may seem ideal but comes at the high cost of requiring an exact match between the enrollment and verification texts.

11.3 Cryptographic Hash Functions

Hash functions are functions that map data of arbitrary size, generally referred to as the message, to a fixed length bit string called a hash digest. Cryptographic hash functions are a special case of hash functions with certain properties that make them suitable for cryptography. An ideal cryptographic hash function has the following properties:

1. It should be quick to compute the hash digest for any type of data.
2. It should be infeasible to obtain the original data given its hash digest.
3. It should be infeasible to find two messages with the same hash digest.
4. A small change in the message must lead to a large change in its hash digest in a manner such that the hash digest corresponding to the altered message is uncorrelated with the original digest.

Thus, the key aspect of cryptographic hash functions that differentiate them from other encryption methods is that the original data cannot be recovered from the encrypted form. Combined with the other properties mentioned above, the only possible attack on a hashed digest is a brute force attack, i.e., the attacker must go through all possible values of the message, hash each one, and compare the hash to the stored digest. In practice this task is made even harder by salting and multiple iterations of hashing. Adding a user-specific salt to the message prior to applying the hash function significantly increases the search space for the attacker. Whereas, applying several iterations of the hash function to the message increases the forward computation time of the digest and thus, significantly increases the time complexity for a brute force attack.

11.4 Biometric Template Protection

The goal of biometric template protection is to protect the biometric data in a manner similar to how text passwords are protected. Since the original biometric data is not changeable like a text password, its security is arguable more crucial. An ideal biometric template protection algorithm should possess the following properties.

1. Security—It should be infeasible to extract the original biometric data given the protected template.
2. Cancelability—It should be feasible to generate a new protected template from the original biometric if the old template has been compromised.
3. Performance—The template protection algorithm should not lead to loss in matching performance (FAR and FRR) of the biometric system.

From the perspective of security, cryptographic hash functions would be the ideal choice for biometric templates as well, but certain properties of biometric data make hash-based security very challenging. Hash-based security would require an exact

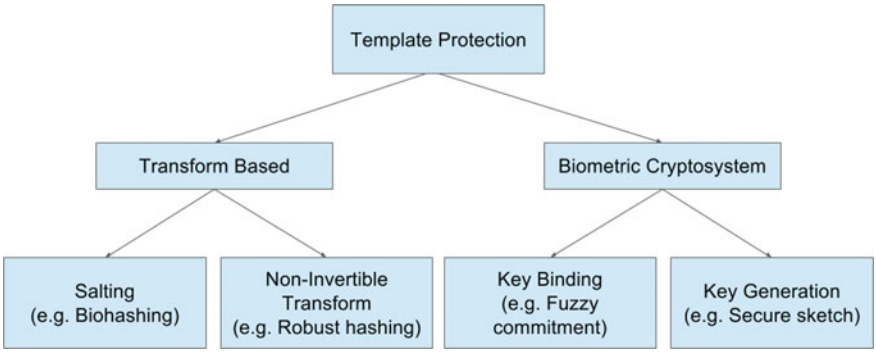

Fig. 11.1 Biometric template protection algorithms

match between the templates extracted from different readings of biometric data. Since biometric data shows significant inherent variations between readings, it is very challenging to extract a templates that would match exactly. Even a single bit variation between the templates would lead to very different hash digest values. Additionally, since biometric data cannot be changed in the event of the hash digest being compromised, cancelability also needs to be addressed in some way.

Despite the difficulties involved in achieving hash-based template security for biometric data, several approaches have been proposed. As shown in Fig. 11.1, biometric template protection algorithms can be broadly classified into two types [12], biometric cryptosystems and feature transforms.

Biometric cryptosystem approaches rely on the use of some publicly available data (referred to as helper data) derived from the original biometric. The helper data should, of course, not reveal significant information about the original biometric data. Depending on the way the helper data is derived, biometric cryptosystems are of two types; key binding and key generation schemes. Key binding schemes derive the helper data by binding an external key with the biometric data. For example, fuzzy commitment binds a key in the form of an error correcting output code C, with the biometric data X, and generates helper data $X - C$ and $Hash(C)$. When a new sample X' needs to be verified, it is combined with the helper data yielding $C' = X' - (X - C)$. If X' is close to X, C' would be close to C and thus, the error correcting decoder will be able to correct C' to C. This enables us to match the hash of the corrected code to the stored $Hash(C)$. On the other hand, key generation schemes try to extract stable keys from the biometric data itself. Examples include fuzzy extractors and secure sketches proposed in [6]. The idea behind fuzzy extractors is to extract a uniform random string from the biometric data in a noise tolerant way. Thus even if the biometric data is altered slightly, the uniform random string can be reproduced exactly and used as a key. A secure sketch S, is some publicly visible data that is extracted from the biometric X, but does not reveal significant information about X. Given X' (some value close to X) and S it is possible to recover the value

of X. Thus a secure sketch enables the exact recovery of the enrolled biometric but does not address uniformity like fuzzy extractors.

Feature transform based approaches simply try to transform the biometric data into an encoded space where matching is possible. They are of two types; non-invertible transforms and biometric salting. Non-invertible transforms apply a non-invertible function to the biometric data to generate a template from which it is infeasible to extract the original biometric. During verification the same function is applied to the new sample and matching is performed in the transformed space. Biometric salting based approaches transform the biometric data in an invertible manner using a secret key. Since it is possible to obtain the original data if the key is compromised, the security of salting-based approaches is entirely dependent on the security of the secret key.

Let us briefly go over some of these algorithms that have been applied to faces. Schemes that used cryptosystem based approaches include Fuzzy commitment schemes by Ao and Li [1], Lu et al. [16] and Van Der Veen et al. [30], and fuzzy vault by Wu and Qiu [32]. In general, the fuzzy commitment schemes suffered from limited error correcting capacity or short keys. In Fuzzy vault schemes the data is stored in the open between chaff points, and this also causes an overhead in storage space. Some quantization schemes were used by Sutcu et al. [23, 24] to generate somewhat stable keys. There were also several works that combine the face data with user specific keys. These include combination with a password by Chen and Chandran [4], user specific token binding by Ngo et al. [17, 28, 29], biometric salting by Savvides et al. [20], and user specific random projection schemes by Teoh and Yuang [27] and Kim and Toh [13]. Although the use of user-specific tokens boosts matching performance while enabling high template security, there are questions raised regarding the protection of the tokens themselves and the contribution of the biometric data towards the performance. Hybrid approaches that combine transform based cancelability with cryptosystem-based security like [8] have also been proposed but give out user specific information to generate the template, creating openings for masquerade attacks.

11.5 Deep Learning for Face Template Protection

In the last few years deep convolutional neural network (CNN) based algorithms like Facenet [21] and Deepface [25] have shown exceptional performance and hold the current state-of-the-art results for face recognition. The key reason behind the success of deep neural networks is their ability to learn representations suited to the task, as opposed to using hand-crafted features. The large availability of labeled data for faces coupled with recent advances in deep CNN architectures and training algorithms has made it possible to learn representations that far exceed the performance of traditional features for face recognition. Consequently biometric authentication using faces has also benefited from this and matching numbers have reached all time highs.

This representation learning ability of neural networks can also be used to learn features suited to template security in addition to discriminative ability. As discussed earlier, cryptographic hash based template security is theoretically ideal but achieving it without some form of information leakage or severe loss in matching performance is challenging. The next two sections discuss two recent approaches to face template security that utilize recent advances in deep learning to address the challenges related to achieving cryptographic hash based security.

11.6 Local Region Hashing

To the best of our knowledge, local region hashing [19] is one of the first approaches that studies the suitability of learned representations for the purpose of cryptographic hash based face template security. The algorithm seeks to achieve exact matching between features extracted from local regions of the face instead of the entire face. Since these regions are smaller, features extracted from them would arguably show lesser variations as compared to features extracted from the entire face. Let us discuss the algorithm and accompanying experiments in further detail.

11.6.1 Algorithm

This approach for template generation consists of the following. First, the face image is divided into a set of selected local regions. Next, features are extracted from each of them and the feature vectors are quantized to eliminate minor variations. Finally, the quantized feature vectors are hashed and stored. Thus, the protected face template consists of a set of hashed features. An overview of the algorithm is shown in Fig. 11.2.

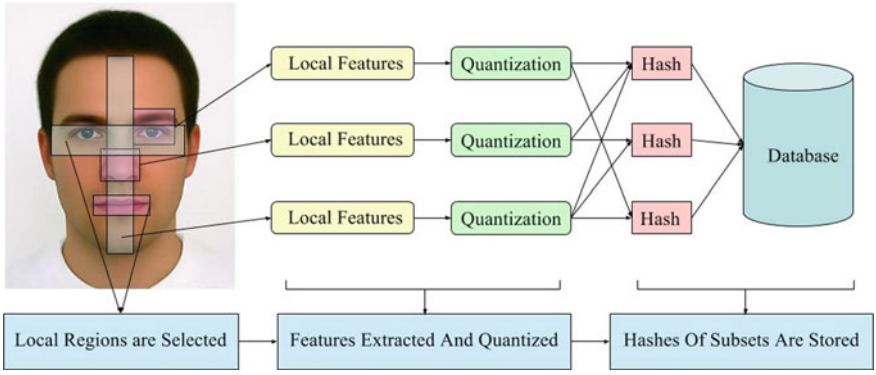

Fig. 11.2 Overview of the local region hashing

11.6.1.1 Facial Region Selection

The purpose of extracting features from fixed local regions of the face is twofold. First, this increases the chances of obtaining exact matches between features. Even though there are inherent variations in holistic features extracted from face samples belonging to different sessions, there exist local regions of the face that remain relatively unchanged, and thus yield features that show much smaller variations. Second, choosing fixed regions, along with tweaks to the design of the framework, ensures that features from a particular region are matched only to features from the same region. This imposes a spatial constraint on the features and reduces the chances of false positive resulting from features from one region being matched to those from others. Thus, the total region of interest is given by a union of V smaller local regions, r, as,

$$R_{interest} = \bigcup_{i=0}^{V-1} r_i \tag{11.1}$$

Desirable characteristics of the selected local regions are: (i) They should yield features that are discriminative. (ii) They should yield features that have higher chances of exact matches. At the pixel level this can be controlled by varying the size and location of the regions. $R_{interest}$ should be such that it spans either the entire face or at least the areas of higher statistical significance for discriminative power. The size of each r_i on the other hand affects the ability to yield features that match exactly. If the size is too large, the features extracted from them may show high variations and if the size is too small, the features extracted may not be discriminative enough. Thus, this can be seen as the first level of control of a two stage process which is geared towards yielding features suited to our task.

For the purpose of experimentation $R_{interest}$ is chosen as the entire face, and equally sized nonoverlapping blocks as the local regions. Hence for a face image of size $(d \times d)$, division into b nonoverlapping blocks would yield local regions of size $(d/b \times d/b)$.

11.6.1.2 Feature Extraction

The second level of control on the representation of the template comes in the form of high-level feature extractors applied to the low-level pixel regions. The higher level features that are used should possess the following characteristics: (i) They should be stable to minor variations and thus be appropriate for exact matching. (ii) They should be discriminative enough to distinguish users and thus have low FAR/FRR. (iii) They should be of sufficient length in order for their hashed versions to be resistant towards brute force attacks, and possess sufficient entropy to be resistant to smarter attacks. For now, the simpler requirements of stability and length are discussed, leaving the more involved entropy-related aspects for the security analysis section. The first feature extraction strategy uses standard feature extractors which

have been shown to be useful for face recognition. The second explores the use of a denoising autoencoder to learn a feature representation.

Standard feature extractors

The suitability of two standard feature extractors, namely, histogram of gradients (HoG) [5] and local binary pattern (LBP) histograms [18] is explored. The choice is driven by the motivation that these are the two most basic gradient and texture-based feature extractors and studying their performance for the task at hand would give us valuable insight into the kind of representations suitable for security as well as accuracy. The controllable parameters for HoG features are the number of cells per block, the number of pixels per cell, and the number of orientations. The number of pixels per cell and number of orientations would affect the feature length. Let o denote the number of orientations and $c = m/p$ denote the number of cells if $(p \times p)$ pixels per cell for a region of size $(m \times m)$ are chosen. The final feature length is given by $l_{feat} = o \times c$. For LBP histograms, the variable parameters are the radius and the number of neighbors. The radius affects the resolution of the LBP codes calculated while the number of neighbors determines the feature length. Thus, for LBP histograms calculated with n neighbors per pixel, the feature length is given by $l_{feat} = 2^n$.

The length of the feature vector not only affects the discriminative ability of the features but also the resilience of their hashed versions to brute force attacks. The effect of feature length of discriminative ability and template security are discussed in further detail in the experiments section.

Learned representation

The standard feature extractors are compared to a representation learned from the data itself. Not only does this generate features that are suited to the choice of local regions, but it also enables greater control on the desirable properties of the features. A stacked denoising autoencoder (SdA) [31] is used to learn this representation. The autoencoder is a neural network that maps the input back to itself through some hidden states. The loss between the reconstructed input and the original one is minimized to learn a representation of the size of the number of hidden states. In order to learn a robust representation and prevent the autoencoder from learning the identity function, a denoising autoencoder (dA) tries to reconstruct the input from a corrupted version of it. Thus, the input layer of the neural network is the data, x whose representation we wish to learn. The hidden layer consists of h nodes to which the input layer is fully connected via a set of weights, W along with some bias, b. The output of the hidden layer is a latent representation given by $y = s(Wx + b)$, where s is a nonlinearity such as a sigmoid function. The latent representation is mapped back to the reconstruction z given by $z = s(W'y + b')$. The parameters of this model W, W′, b, and b′ are optimized to minimize the reconstruction error between z and x. In our case this is given by the squared error, $L(xz) = \| x - z \|^2$. Multiple dA's can be stacked to form a SdA where the output of each layer is treated as the input to the next and mapped back to the respective layers and finally the reconstructed input.

Gaussian noise is applied to perturb the input during training. Training is done layer wise, in an unsupervised manner, minimizing the reconstruction error for each layer independently. The SdA is trained using the pixel level data from all the blocks seen during enrollment and the trained network is then used as a feature extractor. The feature would correspond to the hidden representation y of the final layer and its length is given by $l_{feat} = h$. In this way, the length of the feature vector can be controlled by varying the number of hidden nodes in the network.

11.6.1.3 Quantization and Combination

After extraction and normalization of each d dimensional feature vector f, such that $f \in [0, 1]^d$, they are quantized by binning each dimension into one of q bins, b of size $1/q$ yielding,

$$f_q \in \left\{b_0, b_1, \ldots, b_{q-1}\right\}^d \tag{11.2}$$

This is done to eliminate minor variations between vectors and increase chances of an exact match. The discriminative power and entropy of the features are also reduced during this step and the effects of this are analyzed in the experiments and security analysis sections. In addition to binning, a region identifier is appended to the feature vector to tag it with the region it was extracted from. Thus, if feature vector $f_q \in \left\{b_1, b_2, \ldots, b_q\right\}^d$ was extracted from local region r_i, the region number i would be appended to f, yielding,

$$f_{qr} \in \left\{b_0, b_1, \ldots, b_{q-1}\right\}^d \parallel i \tag{11.3}$$

This ensures that features would only match to others from the same region and imposes a spatial constraint on local matching.

Another way in which length can be increased for feature vectors is by using combinations of vectors from different regions. This would prove useful in scenarios where low length features are discriminative enough to yield good matching accuracy but not of sufficient length to be resilient to brute force attacks on their hashed versions. Given a set of features $F = \left\{f_{qr0}, f_{qr1}, \ldots, f_{qrV-1}\right\}$, of dimensionality d, and an assumption that, on average, at least k features from different regions match, indexing of all possible combinations of size k is possible. Now, each new feature f'_{qr} would be given by $\|_{j=0}^{k-1} f_{qrj}$ with a new dimensionality of $k \times d$. The feature set for a sample would then be $F' = \left\{f'_{qr0}, f'_{qr1}, \ldots, f'_{qr\binom{V-1}{k}}\right\}$. Thus, there is no loss of exact matching, gain in feature length and intuitively, a reduction in the chances of false positives.

11.6.1.4 Template Protection and Indexing

Given a set of features, F, per sample, the next objective is to hashing and store them in the form of a biometric template. A standard cryptographic hash function $h(x)$ (SHA-256 in this case) is used for hashing each feature in F. This yields a new set T, corresponding to the protected template for one sample, and given by,

$$T = \left\{h(f_{qr0}), h(f_{qr1}), \ldots, h(f_{qrV-1})\right\} \tag{11.4}$$

Thus, the final protected template is a set of hashed local features extracted from the face. If multiple images are enrolled, the template would be a set consisting of hash digests of all the non-repeated quantized feature vectors seen during enrollment.

During authentication, similar local features are extracted from the face image, quantized, and hashed yielding a set of V local features, T_{test}. In verification mode, the template corresponding to the user, $T_{enrolled}$ is retrieved and the matching score is given by the set intersection between the two templates as,

$$score = T_{test} \cap T_{enrolled} \tag{11.5}$$

The index can also be set up for identification where a given sample does not claim to be from any class. In this case, the templates for all users would be gone through and the sample would be identified as the user with the template which showed maximum matches.

11.6.2 Experiments

The framework is tested on the Multi-PIE [10] database using ROC curves and the EER as evaluation metrics. Frontal poses with neutral expressions of the 167 face classes that are common between session 1 and session 2 are used for evaluation. Ten randomly selected images of each user from session 1 are used for enrollment, and all the images from session 2 are used for verification. Thus, the test bed contains variations in lighting and inherent variations in samples taken from different sessions. These variations are arguably sufficient for deployment of the algorithm in scenarios of user compliance. Due to the strict requirements of exact matching, proper alignment of the images is crucial and handled by aligning eye centers for each image. Other specifics including the parameters and implementation libraries used for the experiments are described below.

Each face image is aligned to have their eyes positioned at roughly the same location and cropped to remove most of the background yielding a final face image of size 200×200. The face image is then divided into $V = 100$ local regions in the form of nonoverlapping blocks of size 20×20 with $R_{interest}$ from Eq. 11.1 being the entire image. Next, features f are extracted from each block followed by binning each dimension into $q = 4$ bins and appending the region identifier $i \in \{1, 2, \ldots, 100\}$

yielding f_{qr} from Eq. 11.3. For HoG features the number of cells are fixed to 4 and the number of orientations, o, are varied. For LBP features the number of neighbors, n, are varied. For the SdA, two layers are used, fixing the number of hidden states in the first layer to 576 and varying the number of hidden layers in the second hidden layer to the desired dimensionality of the learned representation. Zero mean Gaussian noise with 0.5 variance is used as noise, and training is done for 20 epochs for each layer, with a learning rate of 0.3. Note that blocks extracted from the enrollment set are used to train the autoencoder for simplicity but since we merely want to learn a feature representation for faces, the training could have been done on an entirely different database as well. Scikit-image is used for calculating HoG and LBP features. The autoencoder is implemented using Theano [2, 3]. Experimentation is done with feature lengths ranging from 8–32 for HoG, 16–256 for LBP and 16–256 for the autoencoder. Next each f_{qr} is hashed using SHA-256 as the hashing function h, yielding the set T from Eq. 11.4. During enrollment a set of non-repeating features is built for each user. For verification, the set intersection between the set generated from the test sample, T_{test_i} and the set enrolled for the claimed user, T_{user_i} is computed. This yields the $score$ from Eq. 11.5 as $T_{test_i} \cap T_{user_i}$. The score ranges from 0–100 in this case. Genuine scores are computed by comparing test samples against their true classes while impostor scores are computed against each of the other enrolled classes.

Figure 11.3 shows the ROC curves for HoG, LBP, and autoencoder features respectively (note that some of the curves are not smooth due to the discrete nature of the score and overlaps at varying thresholds). It can be seen that HoG features perform very poorly showing a reasonable EER of 15.3% only at a very low bits of security of 8, making it unsuitable for security. LBP features perform better with an EER of 21.3% at six neighbors or 128 bits of security. The feature representation learned by the SdA shows the best results at an acceptable 128 bits of security with an EER of 16.6%. Interestingly, the EER is not far behind even at 256 bits. It is worth noting that the matching score in all the cases was never more than 60–70% of the total number of regions, confirming that such a hashing-based security scheme would be infeasible if holistic features from the entire face were used.

11.6.3 Security Analysis

The security of the algorithm is studied in a stolen template scenario, where in, if an attacker gains access to the secure templates, it should be infeasible to extract the original biometric data from them. It is assumed that the hash function, SHA-256, is collision resistant and follows the random oracle model in which the hashed digests reveal no dependencies between them. In such a scenario, the only way to attack the system is by guessing values in the feature domain, hashing them and comparing them to the stored hashes in the database. A naive attacker would assume uniformity in the feature space and search through every possible value. Hence, in this case the bits of security offered by the system is proportional to the size of the

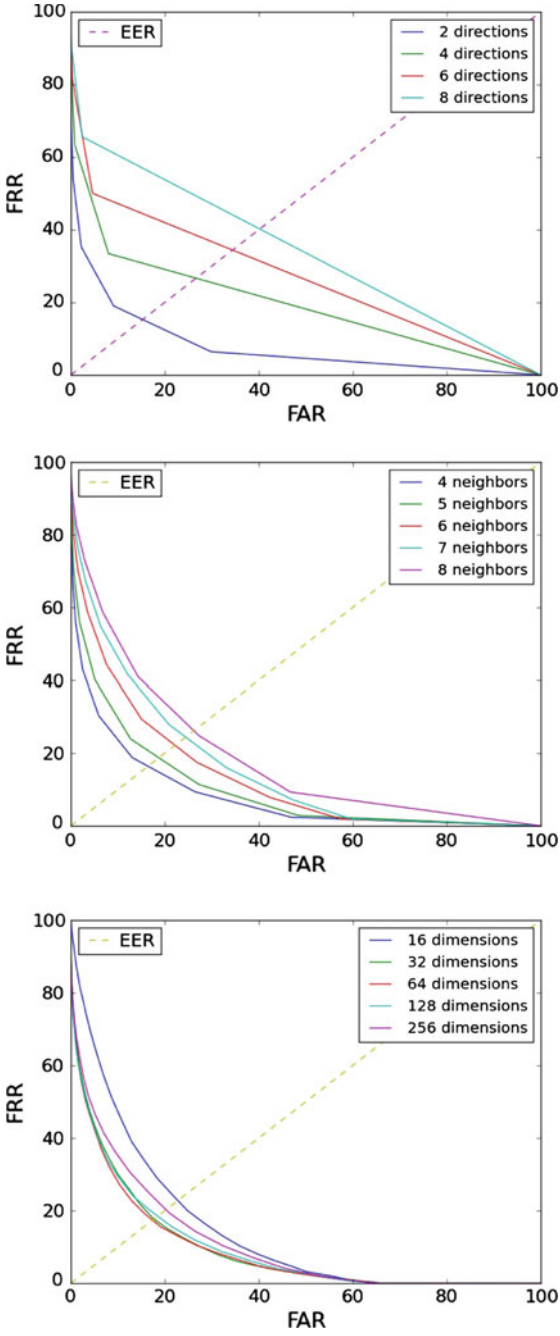

Fig. 11.3 ROC curves for HoG (*top*), LBP (*mid*) autoencoder (*bottom*) features

Table 11.1 Feature security

Feature extractor	$q^{l_{feat}}$	$MI(Db_{en}, Db_{at})$
HoG	2^{64}	0.1610
LBP	2^{64}	0.1068
	2^{128}	0.0962
	2^{256}	0.1029
SdA	2^{64}	0.0739
	2^{128}	0.0870
	2^{256}	0.1020

search space, given by $T \times q^{l_{feat}}$, where T is the selected score threshold. In reality, the feature space is far from uniform and approximating its true distribution in the lack of sufficient data is a daunting task. Instead, it is somewhat easier to quantify the amount by which a smart attacker can reduce the search space from $q^{l_{feat}}$. It is assumed that the attacker has knowledge of the algorithm, the feature extractor used, and access to face data, Db_{at} that is not enrolled in the system. The amount by which an attacker can reduce his search space using information leakage, is quantified in the form of mutual information (MI), between the features extracted from Db_{at} and the enrolled database, Db_{en}. It is hypothesized that due to the highly multimodal nature of the data from different regions and face classes, a smart attack will have to be made region wise and thus, the information leakage is reported in terms of the average MI between regions from the two databases.

$$MI(Db_{en}, Db_{at}) = \frac{(\sum_{i=1}^{V} MI(F_{qr_i}^{en}, F_{qr_i}^{at})}{V} \tag{11.6}$$

where F_{qr_i} is the set of quantized features extracted from region i.

For the purpose of this experiment Db_{en} is a set of 20 randomly chosen face classes while Db_{at} is a distinct set of rest of the 147 classes not in Db_{en}. The MI is calculated using the non-parametric entropy estimation toolkit (NPEET) which implements the MI estimator proposed by Kraskov et al. [14]. Since the non-parametric estimation is specific to the data distribution, the MI values are best seen as a comparative measure between the feature extractors, and not on a universal scale. The MI values for different feature extractors at varying feature lengths is shown in Table 11.1, and further details about the values themselves can be found in [14].

11.7 Deep Secure Encoding

Let us now discuss an alternative approach that addresses the shortcomings of local region hashing in terms of template security as well as matching accuracy. The algorithm uses a deep convolutional neural network (CNN) to learn a robust mapping of face classes to codes referred to as maximum entropy binary (MEB) codes. The

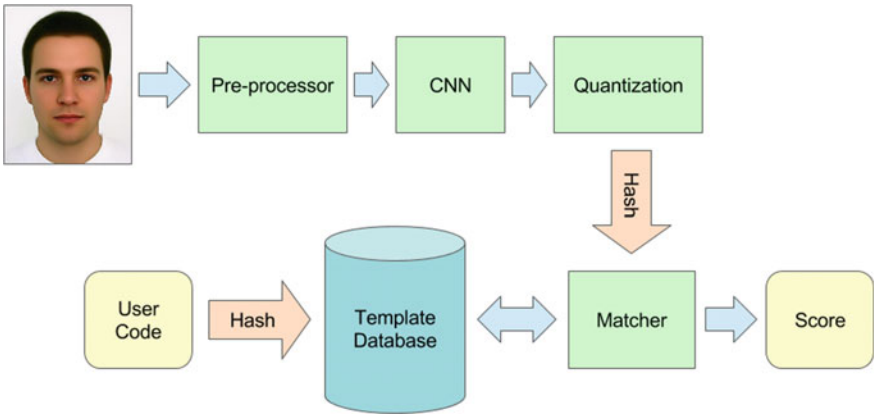

Fig. 11.4 Overview of deep secure encoding

mapping is robust enough to tackle the problem of exact matching, yielding the same code for new samples of a user as the code assigned during training. This exact matching makes it possible to store the cryptographic hash of the code as the template for the user. Once hashed, the template has no correlation with the code assigned to the user. Furthermore, the codes assigned to users are bit-wise randomly generated and thus, possess maximum entropy, and have no correlation with the original biometric modality (the user's face). These properties make attacks on the template very difficult, leaving brute force attacks in the code domain and complex dictionary attacks in the input domain as the only feasible options. Cancelability can also be easily achieved by changing the codes assigned to users and relearning the mapping.

11.7.1 Algorithm

We now describe the individual components of our architecture in more detail. An overview of the algorithm is shown in Fig. 11.4.

11.7.1.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) [15] are biologically inspired models, which contain three basic components: convolution, pooling, and fully connected layers. In the convolution layer one tries to learn a filter bank given input feature maps. The input of a convolution layer is a 3D tensor with d number of 2D feature maps of size $n_1 \times n_2$. Let x_{ijk} denote the component at row j and column k in the ith feature map, and $x_i^{(l)}$ denote the complete ith feature map at layer l. If one wants to learn h_f set of filters of size $f_1 \times f_2$, the output $x^{(l+1)}$ for the next layer will still be a 3D

tensor with h_f number of 2D feature maps of size $(n_1 - f_1 + 1) \times (n_2 - f_2 + 1)$. More formally, the convolution layer computes the following:

$$x_j^{(l+1)} = s\left(\sum_i F_{ij}^{(l)} * x_i^{(l)} + b_j^{(l)}\right) \tag{11.7}$$

where $F_{ij}^{(l)}$ denotes the filter that connects feature map x_i to output map $x_j^{(l)}$ at layer l, $b_j^{(l)}$ is the bias for the jth output feature map, $s(\cdot)$ is some element-wise nonlinearity function and $*$ denotes the discrete 2D convolution.

The pooling (or subsample) layer takes a 3D feature map and tries to downsample/summarize the content to lesser spatial resolution. Pooling is commonly done for every feature map independently and with nonoverlapping windows. The intuition behind such an operation is to have some built in invariance against small translations. Additionally this reduces the spatial resolution and thus saves computation for the upper layers. For average (mean) pooling, the output will be the average value inside the pooling window, and for max pooling the output will be the maximum value inside the pooling window.

The fully connected layer connects all the input units from a lower layer l to all the output units in the next layer $l + 1$. In more detail, the next layer output is calculated by

$$x^{(l+1)} = s(W^{(l)}x^{(l)} + b^{(l)}) \tag{11.8}$$

where $x^{(l)}$ is the vectorized input from layer l, $W^{(l)}$ and $b^{(l)}$ are the parameters of the fully connected layers at layer l.

A CNN is commonly composed of several stacks of convolution and pooling layers followed by a few fully connected layers. The last layer is normally associated with some loss to provide training signals, and the training for CNN can be done by doing gradient descent on the parameters with respect to the loss. For example, in classification the last layer is normally a softmax layer and cross-entropy loss is calculated against the 1 of K representation of the class labels. In more detail, let $x^{(L)} = Wx^{(L-1)} + b$ be the pre-activation of the last layer, $\mathbf{t}$ denotes the final output and t_k the kth component of $\mathbf{t}$, and $\mathbf{y}$ denote the target 1 of K vector and y_k the kth dimension of that vector, then

$$t_k = \frac{\exp\{x_k^{(L)}\}}{\sum_j \exp\{x_j^{(L)}\}} \tag{11.9}$$

$$L(\mathbf{t}, \mathbf{y}) = \sum_j y_j \log t_j \tag{11.10}$$

where L is the loss function.

11.7.1.2 Maximum Entropy Binary Codes

The first step of training is to assign unique codes to each user to be enrolled. Note that these codes are internally used for training during enrollment, and are not supplied to the user or retained in an unprotected form after training. From a template security point of view, these codes should ideally possess two properties. First, they should posses high entropy. Since a hash of these codes is the final protected template, the higher the entropy of the codes, the larger the search space for a brute force attack would be. In order to make brute force attacks in the code domain infeasible, we use binary codes with a minimum dimensionality $K = 256$ and experiment with values up to $K = 1024$. The second desirable property of the codes is that they should not be correlated with the original biometric modality. Any correlation between the biometric samples and the secure codes can be exploited by an attacker to reduce the search space during a brute force attack. One example to illustrate this can be to think of binary features extracted from faces. Even though the dimensionality of the feature vector may be high, given the feature extraction algorithm and type of data, the number of possible values the vector can take is severely reduced. In order to prevent such reduction of entropy, the codes used are bit-wise randomly generated and have no correlation with the original biometric samples. This makes the space to be hashed uniformly distributed. More precisely, let $c_i \sim \mathbf{B}(1, 0.5)$ be the binary variable for each bit of the code, where $\mathbf{B}(1, 0.5)$ is the maximum entropy Bernoulli distribution, and the resultant MEB code with K independent bits is thus $\mathbf{C} = [c_1, c_2, \ldots, c_K]$. The code for user u is denoted by $\mathbf{C}_u$.

11.7.1.3 Learning the Mapping

In order to learn a robust mapping of a user's face samples to the codes, some modifications are made to the standard CNN training procedure. As shown in Fig. 11.5, the 1 of K encoding of the class labels is replaced by the MEB codes $\mathbf{C}_u$ assigned to each user. Since several bits of the network output are going to be one instead of a single bit, sigmoid activation is used instead of softmax. In more detail

$$t_k = \frac{1}{1 + \exp\{-x_j^{(L)}\}} \tag{11.11}$$

$$L(\mathbf{t}, \mathbf{C}) = \sum_j \{c_j \log t_j + (1 - c_j) \log(1 - t_j)\} \tag{11.12}$$

where t_k is the kth output from the last layer and L is the binary cross-entropy loss.

Data Augmentation

Deep learning algorithms generally require a large number of training samples whereas, training samples are generally limited in the case of biometric data. In order to magnify the number of training samples per user, the following data

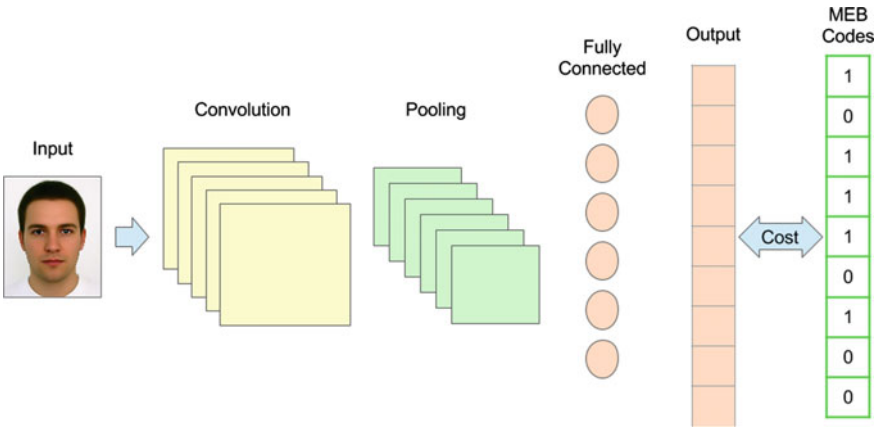

Fig. 11.5 Learning the mapping

augmentation is performed. For each training sample of size $m \times m$ all possible crops of size $n \times n$ are extracted. Each crop is also flipped along its vertical axis yielding a total of $2 \times (m - n + 1) \times (m - n + 1)$ crops. The crops are then resized back to $m \times m$ and used for training the CNN.

Regularization

The large learning capacity of deep neural networks comes with the inherent risk of overfitting. The number of parameters in the network are often enough to memorize the entire training set, and the performance of such a network does not generalize to new data. In addition to general concerns, mapping to MEB codes is equivalent to learning a highly complex function, where each dimension of the function output can be regarded as an arbitrary binary partition of the classes. This further increases the risk of overfitting and powerful regularization techniques need be employed to achieve good matching performance.

Dropout [11] is applied to all fully connected layers with 0.5 probability of discarding a hidden activation. Dropout is a very effective regularizer and can also be regarded as training an ensemble of an exponential number of neural networks that share the same parameters, therefore reducing the variance of the resulting model.

11.7.1.4 Protected Template

Even though MEB codes assigned to each user have no correlation with the original samples, another step of taking a hash of the code is required to generate the protected template. Given the parameters of the network, it is not possible to entirely recover the original samples from the code (due to the max pooling operation in the forward pass of the network) but, some information is leaked. Using a hash digest of the code

as the final protected template prevents any information leakage. The hash function used can be any function that follows the random oracle model. For experimentation SHA-512 is utilized, yielding the final protected template $\mathbf{T}_u = \text{SHA512}(\mathbf{C}_u)$.

During verification, a new sample of the enrolled user is fed through the network to get the network output $\mathbf{y}_{out} = \mathbf{t}$. This output is then binarized via a simple thresholding operation yielding the code for the sample $\mathbf{s}_{out} = [s_1, s_2, \ldots, s_K]$, where $s_i = \mathbf{1}(t_i > 0.5)$ and $\mathbf{1}(\cdot)$ is the indicator function. At this point, the SHA-512 hash of the code, $\mathbf{H}_{out} = \text{SHA512}(\mathbf{s}_{out})$ could be taken and compared with the stored hash $\mathbf{T}_u$ for the user. Due to the exact matching nature of the framework, this would yield a matching score of true/false nature. This is not ideal for a biometric-based authentication system since it is desirable to obtain a tunable score in order to adjust the false accept (FAR) and false reject rates (FRR). In order to obtain an adjustable score, several crops and their flipped counterparts are taken for the new sample (in the manner described in Sect. 11.7.1.3) and $\mathbf{H}_{out}$ is calculated for each one, yielding a set of hashes $\mathbb{H}$. The final matching score is defined as the number of $\mathbf{H}_{outs}$ in $\mathbb{H}$ that match the stored template, scaled by the cardinality of $\mathbb{H}$. Thus, the score for matching against user u is given by,

$$score = \frac{\sum_{\mathbf{H}_i \in \mathbb{H}} \mathbf{1}(\mathbf{H}_i = \mathbf{T}_u)}{|\mathbb{H}|} \tag{11.13}$$

Now the score can be set to achieve the desired value of FAR/FRR. Note that, the framework provides the flexibility to work in both verification and identification modes. For identification $\mathbb{H}$ can be matched against templates of all the users stored in the database.

11.7.2 Experiments

We now discuss the databases, evaluation protocols, and specifics of the parameters used for experimental evaluation.

11.7.2.1 Databases

This study tackles the problem of using faces as passwords and thus, utilizes face databases that have been collected in controlled environments for experimentation. The evaluation protocols include variations in lighting, session, and pose that would be typical to applications like face unlock since a reasonable degree of user compliance is expected.

The CMU PIE [22] database consists of 41,368 images of 68 people under 13 different poses, 43 different illumination conditions, and with four different expressions. Five poses (c27, c05, c29, c09 and c07) and all illumination variations are

used for experimentation. 10 images are randomly chosen for training and the rest are used for testing.

The extended Yale Face Database B [9] contains 2432 images of 38 subjects with frontal pose and under different illumination variations. The cropped version of the database is used for experimentation. Again, 10 randomly selected images are used for training and the rest for testing.

The CMU Multi-PIE [10] face database contains more than 750,000 images of 337 people recorded in four different sessions, 15 view points and 19 illumination conditions. This database is used to highlight the algorithm's robustness to changes in session and lighting conditions. Two sessions (3 and 4) which have the most number of common users (198) between them are chosen. Ten randomly chosen frontal faces from session 3 are used for enrollment and all frontal faces from session 4 are used for testing.

11.7.2.2 Evaluation Metrics

Genuine accept rate (GAR) at 0 false accept rate (FAR) is used as the evaluation metric. The equal error rate (EER) is also reported as an alternative operating point for the system. Since the train-test splits used are randomly generated, the mean and standard deviation of the results for 10 different random splits are reported.

11.7.2.3 Experimental Parameters

The same training procedure is used for all databases. The CNN architecture used is as follows: two convolutional layers of 32 filters of size 7×7 and 64 filters of size 7×7, each followed by max pooling layers of size 2×2. The convolutional and pooling layers are followed by two fully connected layers of size 2000 each, and finally the output. Rectifier activation function $s(x) = \max(x, 0)$ is used for all layers, and dropout is applied with 0.5 probability of discarding activations to both fully connected layers.

MEB codes of dimensionality $K = 256, 1024$ are assigned to each user. All training images are resized to $m \times m = 64 \times 64$ and roughly aligned using eye center locations. For augmentation $n \times n = 57 \times 57$ crops are used yielding 64 crops per image. Each crop is also illumination normalized using the algorithm in [26]. The network is trained by minimizing the cross-entropy loss against user codes for 20 epochs using mini-batch stochastic gradient descent with a batch size of 200. Five of the training samples are initially used for validation to determine the mentioned training parameters. Once the network is trained, the SHA-512 hashes of the codes are stored as the protected templates and the original codes are purged. During verification, crops are extracted from the new sample, preprocessed, and fed through the trained network. Finally, the SHA-512 hash of each crop is calculated and matched to the stored template, yielding the matching score in Eq. 11.3.

Table 11.2 Verification results obtained from various datasets

Database	K	GAR@0FAR (%)	EER (%)
PIE	256	93.22 ± 2.61	1.39 ± 0.20
	1024	90.13 ± 4.30	1.14 ± 0.14
Yale	256	96.74 ± 1.35	0.93 ± 0.18
	1024	96.49 ± 2.30	0.71 ± 0.17
Multi-PIE	256	95.93 ± 0.55	1.92 ± 0.27
	1024	97.12 ± 0.45	0.90 ± 0.13

11.7.2.4 Results

The results of our experiments are shown in Table 11.2. The mean and standard deviation of GAR at zero FAR, and EER are reported for the 10 different train-test splits at code dimensions, $K = 256, 1024$. GARs up to $\sim$90% on PIE, $\sim$96% on Yale, and $\sim$97% on Multi-PIE with up to $K = 1024$ are achieved at the strict operating point of zero FAR. During experimentation it was observed that the results were stable with respect to K, making the parameter selectable purely on the basis of desired template security. In order to get an idea of the system performance with respect to the operating point, the GAR and FAR of the system are shown with respect to the matching score for $K = 256$ in Fig. 11.6. It is noteworthy that the system has very low FAR values even at low matching scores due to the strict exact matching requirements.

A comparison of the results to other face template protection algorithms on the PIE database is shown in Table 11.3. GAR at an FAR of 1% is compared as this is the reported operating point in [7]. For security level, the code dimensionality parameter (K) is compared to the equivalent parameter in the shown approaches. In the absence of algorithm parameters, this is a good measure of the security level against brute force attacks.

11.7.3 Security Analysis

Once again, the security of the system is analyzed in a stolen template scenario. It is assumed that the attacker has possession of the templates, and knowledge of the template generation algorithm. Given the templates, the attacker's goal is to extract information about the original biometric of the users. Since the hash function used to generate the templates is a one way transformation function, no information about the MEB codes can be extracted from the protected templates. Thus, the only way in which the attacker can get the codes is by brute forcing through all possible values the codes can take, hash each one, and compare them to the templates. The search space for such brute force attacks is now analyzed.

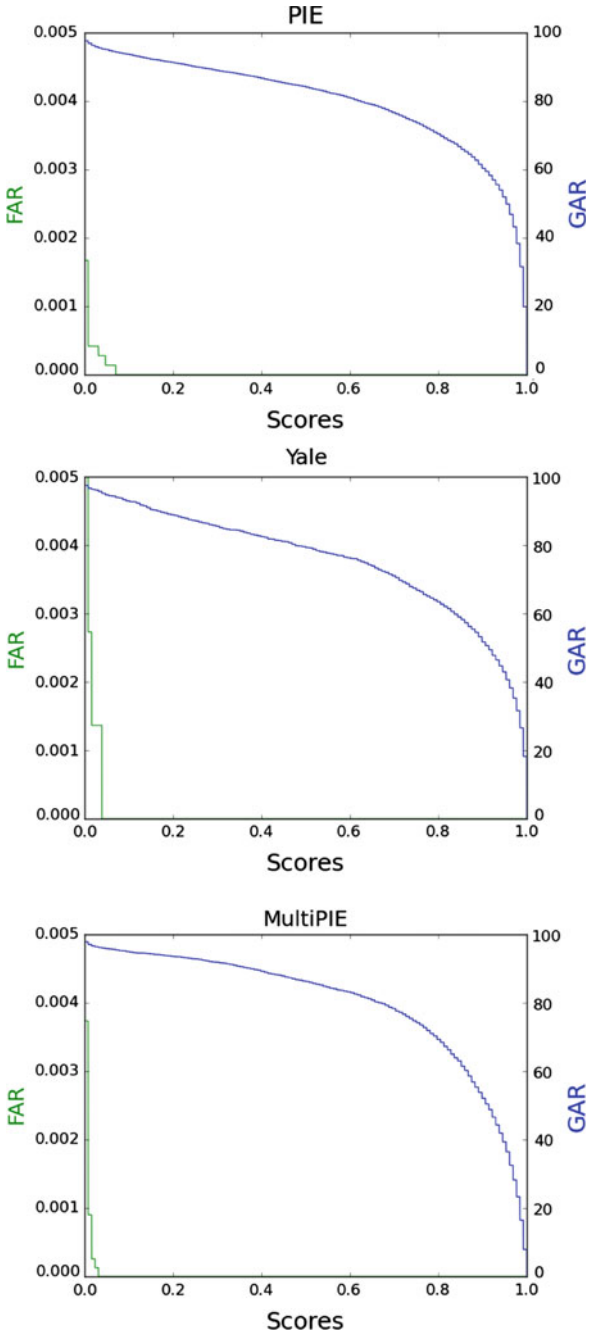

Fig. 11.6 GAR and FAR with respect to matching score at K = 256 for PIE (*top*), Yale (*mid*) and Multi-PIE (*bottom*) databases

Table 11.3 Performance comparison with other algorithms on PIE dataset

Method	K	GAR@1FAR (%)	EER (%)
Hybrid approach [8]	210	90.61	6.81
BDA [7]	76	96.38	–
MEB encoding	**1024**	**97.59**	**1.14**

In the absence of the CNN parameters, the search space for brute force attacks would be 2^K where K is the number of dimensions of the MEB code. This is because the MEB codes are bit-wise randomly generated and uncorrelated to the original biometric data. Since a minimum of $k = 256$ is used, the search space would be of the order of 2^{256} or larger, making brute force attacks computationally infeasible.

An attack given the CNN parameters is now analyzed. With the CNN parameters, it would make sense to generate attacks in the input domain of the network and try to exploit the FAR of the system. Brute forcing through all possible values in the input domain would yield a search space much larger than 2^K. Thus, in a practical scenario, attackers would most likely perform a dictionary attack using a large set of faces that is available to them. Even though it is not straightforward to analyze the reduction of the attacker's search space due to the knowledge of the system parameters, the FAR of the system under the aforementioned attack scenario is arguably a good indicator of the template security. The genuine and imposter score distributions when all other users other than the genuine are treated as imposter are shown in Fig. 11.7. It can be seen that the imposter scores are mostly zero, indicating that there are very few false accepts in this scenario. The genuine and imposter distributions under a dictionary attack using an attacker database consisting of all frontal images of the Multi-PIE database and genuine database consisting of the smaller Yale database is shown in Fig. 11.8. Again, it can be seen that there are minimal false accepts indicating that the model does not easily accept external faces even when they are preprocessed in the same manner as the enrolled ones. Separately, a large number of random noise samples are also used as an attack to verify that the CNN does not trivially learn how to map large portions of the input space to the learned codes. In this case, there are no false accepts showing that the model is indeed not trivially learning to map all inputs to assigned codes. Hence, even though it is not straightforward to quantify the reduction in the search space of codes due to knowledge of the CNN parameters, it has been empirically shown that false accepts are difficult due to the strict exact matching requirements of the system.

It is worth noting that even if a MEB code is obtained by the attacker, reconstructing the original biometric in an exact manner is not possible due to the pooling and dropout operations in the CNN. Furthermore, knowledge of one code reveals no information about the others since they are uncorrelated. Thus, if a security breach is detected, it is safe to use a new set of MEB codes to generate another set of templates.

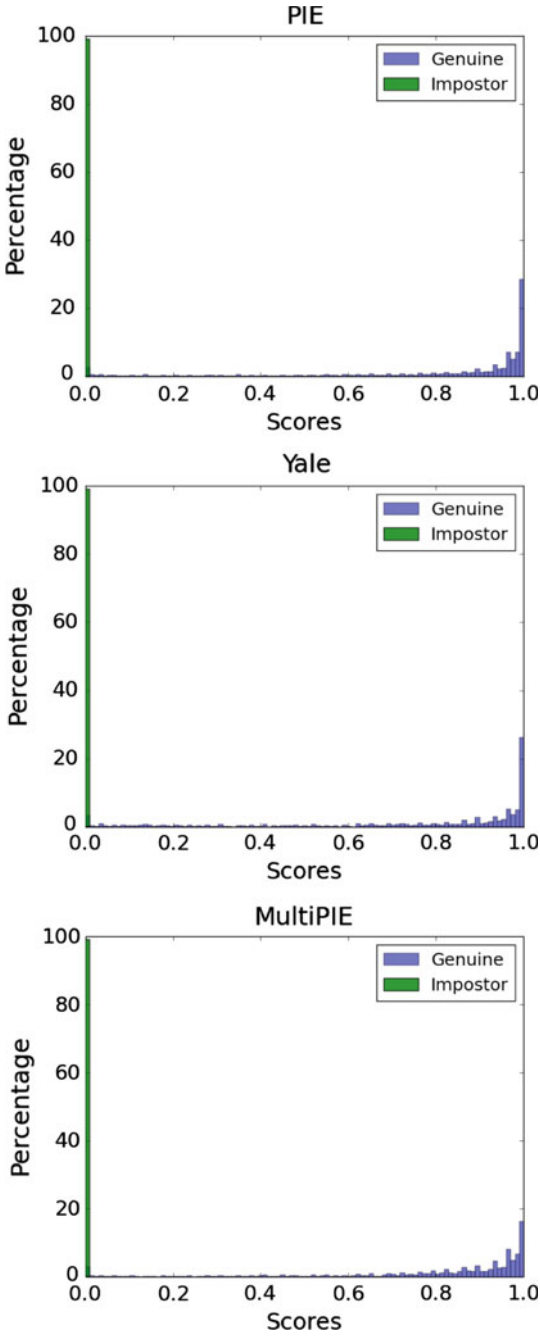

Fig. 11.7 Genuine and imposter distributions from PIE (*top*), Yale (*mid*) and Multi-PIE (*bottom*) databases

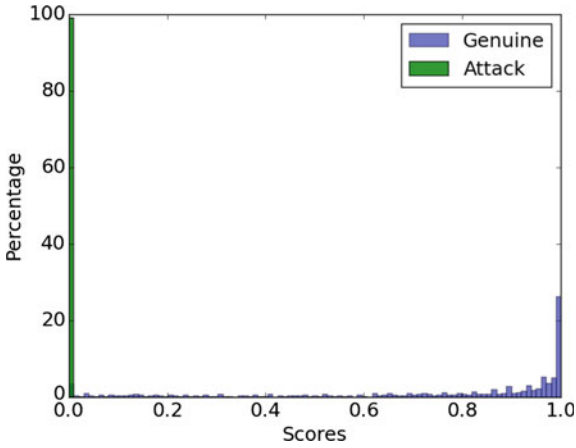

Fig. 11.8 Genuine and imposter distributions under a dictionary attack in the input space

11.8 Future Directions

We have discussed two very different approaches to enabling the use of cryptographic hash functions for face template protection. Local region hashing highlights the challenges associated with cryptographic hash based template security for face biometrics, and confirms the intuition regarding the need for learning representations suitable for security in addition to matching. Even though the algorithm offers a higher standard of template security, it suffers in terms of matching accuracy and uniformity of the hashed space. On the other hand, deep secure encoding takes a different approach by designing the ideal representation for template security first, and then learning a mapping to it. In the process, the algorithm eliminates all the downfalls of local region hashing as well and most previous approaches to face template protection. Even though deep secure encoding achieves state-of-the-art template security as well as matching accuracy, it does not come without its own pitfalls.

One of the disadvantages of deep secure encoding comes to light when we think about enrolling new subjects. Since the mapping to the MEB codes for all enrolled users is learned jointly, the network would need to be retrained in order to enroll new subjects. Deep neural network retraining can be a time consuming task making the process of new enrollment inefficient. That said, neural networks are generally trained by fine tuning a representation that has already been pretrained on relevant data, dramatically reducing the training time. In the context of deep secure encoding, a base network that has been trained on faces in general might be used for initialization. Thus, fine tuning on the classes to be enrolled would be considerably less time consuming. There is, however, a bigger issue with the need for retraining. Since the system does not retain the original face data, it would have to request samples from all the previous users in order to retrain the network. This may not always be feasible. One can think of some workarounds like maintaining different networks for different

sets of users and adding a new network when a new set of users needs to be enrolled, but they could still be complicated to practically deploy.

A possible solution to the above-mentioned issues could be to use a deep neural network to learn a universal locally sensitive hashing function for faces. Instead of learning a mapping to fixed codes like deep secure encoding, a binary code can be learned using a loss function that minimizes the distance between codes for the same user and maximizes the distance between codes for different users. One important detail to keep in mind during the implementation of such an approach is that the smoothness of the binary representation space can be exploited to perform hill climbing attacks on the learned codes. Needless to say, this approach would also require a much larger data pool for training. Whether this would work in a practice or not, is yet to be seen. Other improvements to deep secure encoding include pre-training before enrollment and possible combination with local region hashing to learn mappings for smaller facial regions.

In conclusion, deep learning based algorithms show significant potential for learning representations that are suited to biometric template protection. Although not perfect, algorithms like deep secure encoding achieve impressive results and pave the way to towards high standards of template security with minimal loss in matching accuracy. Furthermore, the generic nature of the presented algorithms enable their use for any task where neural networks are applicable; potentially motivating privacy-centric research in other domains.

References

1. M. Ao, S.Z. Li, Near infrared face based biometric key binding, in *Advances in Biometrics* (Springer, Berlin, 2009), pp. 376–385
2. F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I.J. Goodfellow, A. Bergeron, N. Bouchard, Y. Bengio, Theano: new features and speed improvements, in *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop* (2012)
3. J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, Theano: a CPU and GPU math expression compiler, in *Proceedings of the Python for Scientific Computing Conference (SciPy)* (2010) (Oral Presentation)
4. B. Chen, V. Chandran, Biometric based cryptographic key generation from faces, in *9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications* (IEEE, 2007), pp. 394–401
5. N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*, vol. 1 (IEEE, 2005), pp. 886–893
6. Y. Dodis, L. Reyzin, A. Smith, Fuzzy extractors: how to generate strong keys from biometrics and other noisy data, in *International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2004), pp. 523–540
7. Y.C. Feng, P.C. Yuen, Binary discriminant analysis for generating binary face template. IEEE Trans. Inf. Forensics Secur. **7**(2), 613–624 (2012)
8. Y.C. Feng, P.C. Yuen, A.K. Jain, A hybrid approach for generating secure and discriminating face template. IEEE Trans. Inf. Forensics Secur. **5**(1), 103–117 (2010)

9. A.S. Georghiades, P.N. Belhumeur, D.J. Kriegman, From few to many: illumination cone models for face recognition under variable lighting and pose. IEEE Trans. Pattern Anal. Mach. Intell. **23**(6), 643–660 (2001)
10. R. Gross, I. Matthews, J. Cohn, T. Kanade, S. Baker, Multi-pie. Image Vis. Comput. **28**(5), 807–813 (2010)
11. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors (2012). arXiv:1207.0580
12. A.K. Jain, K. Nandakumar, A. Nagar, Biometric template security. EURASIP J. Adv. Signal Process. **2008**, 113 (2008)
13. Y. Kim, K.-A. Toh, A method to enhance face biometric security, in *First IEEE International Conference on Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007* (IEEE, 2007), pp. 1–6
14. A. Kraskov, H. Stögbauer, P. Grassberger, Estimating mutual information. Phys. Rev. E **69**(6), 066138 (2004)
15. Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in *Proceedings of the IEEE* (1998), pp. 2278–2324
16. H. Lu, K. Martin, F. Bui, K.N. Plataniotis, D. Hatzinakos, Face recognition with biometric encryption for privacy-enhancing self-exclusion, in *2009 16th International Conference on Digital Signal Processing* (IEEE, 2009), pp. 1–8
17. D.C.L. Ngo, A.B.J. Teoh, A. Goh, Biometric hash: high-confidence face recognition. IEEE Trans. Circuits Syst. Video Technol. **16**(6), 771–775 (2006)
18. T. Ojala, M. Pietikainen, T. Maenpaa, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. Pattern Anal. Mach. Intell. **24**(7), 971–987 (2002)
19. R.K. Pandey, V. Govindaraju, Secure face template generation via local region hashing, in *2015 International Conference on Biometrics (ICB)* (IEEE, 2015), pp. 1–6
20. M. Savvides, B.V.K. Vijaya Kumar, P.K. Khosla, Cancelable biometric filters for face recognition, in *ICPR 2004*, vol. 3 (IEEE, 2004), pp. 922–925
21. F. Schroff, D. Kalenichenko, J. Philbin, Facenet: a unified embedding for face recognition and clustering, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 815–823
22. T. Sim, S. Baker, M. Bsat, The CMU pose, illumination, and expression (PIE) databas, in *Proceedings. Fifth IEEE International Conference on Automatic Face and Gesture Recognition, 2002* (IEEE, 2002), pp. 46–51
23. Y. Sutcu, H.T. Sencar, N. Memon, A secure biometric authentication scheme based on robust hashing, in *Proceedings of the 7th Workshop on Multimedia and Security* (ACM, 2005), pp. 111–116
24. Y. Sutcu, Q. Li, N. Memon, Protecting biometric templates with sketch: theory and practice. IEEE Trans. Inf. Forensics Secur. **2**(3), 503–512 (2007)
25. Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: closing the gap to human-level performance in face verification, in *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2014), pp. 1701–1708
26. X. Tan, B. Triggs, Enhanced local texture feature sets for face recognition under difficult lighting conditions. IEEE Trans. Image Process. **19**(6), 1635–1650 (2010)
27. A. Teoh, C.T. Yuang, Cancelable biometrics realization with multispace random projections. IEEE Trans. Syst. Man Cybern. Part B: Cybern. **37**(5), 1096–1106 (2007)
28. A.B.J. Teoh, D.C.L. Ngo, A. Goh, Personalised cryptographic key generation based on face-hashing. Comput. Secur. **23**(7), 606–614 (2004)
29. A.B.J. Teoh, A. Goh, D.C.L. Ngo, Random multispace quantization as an analytic mechanism for biohashing of biometric and random identity inputs. IEEE Trans. Pattern Anal. Mach. Intell. **28**(12), 1892–1901 (2006)
30. M. Van Der Veen, T. Kevenaar, G.-J. Schrijen, T.H. Akkermans, F. Zuo, et al., Face biometrics with renewable templates, in *Proceedings of SPIE*, vol. 6072 (2006), pp. 60720J

31. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**, 3371–3408 (2010)
32. Y. Wu, B. Qiu, Transforming a pattern identifier into biometric key generators, in *ICME 2010* (IEEE, 2010), pp. 78–82

Chapter 12
Deep Triplet Embedding Representations for Liveness Detection

Federico Pala and Bir Bhanu

Abstract Liveness detection is a fundamental element for all biometric systems that have to be safe against spoofing attacks at sensor level. In particular, for an attacker it is relatively easy to build a fake replica of a legitimate finger and apply it directly to the sensor, thereby fooling the system by declaring its corresponding identity. In order to ensure that the declared identity is genuine and it corresponds to the individual present at the time of capture, the task is usually formulated as a binary classification problem, where a classifier is trained to detect whether the fingerprint at hand is real or an artificial replica. In this chapter, unlike the binary classification model, a metric learning approach based on triplet convolutional networks is proposed. A representation of the fingerprint images is generated, where the distance between feature points reflects how much the real fingerprints are dissimilar from the ones generated artificially. A variant of the triplet objective function is employed, that considers patches taken from two real fingerprint and a replica (or two replicas and a real fingerprint), and gives a high penalty if the distance between the matching couple is greater than the mismatched one. Given a test fingerprint image, its liveness is established by matching its patches against a set of reference genuine and fake patches taken from the training set. The use of small networks along with a patch-based representation allows the system to perform the acquisitions in real time and provide state-of-the-art performances. Experiments are presented on several benchmark datasets for liveness detection including different sensors and fake fingerprint materials. The approach is validated on the cross-sensor and cross-material scenarios, to understand how well it generalizes to new acquisition devices, and how robust it is to new presentation attacks.

F. Pala (✉) · B. Bhanu
Center for Research in Intelligent Systems - University of California,
Riverside Winston Chung Hall Suite 216, 900 University Avenue,
Riverside, CA 92521-0425, USA
e-mail: fedpala@ucr.edu

B. Bhanu
e-mail: bhanu@cris.ucr.edu

B. Bhanu and A. Kumar (eds.), *Deep Learning for Biometrics*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-61657-5_12

12.1 Introduction

Biometrics authentication systems have become part of the daily routine for millions of people around the world. A large number of people use their fingerprints every day in order to pass the security checkpoints at airports, to access their personal mobile devices [1] and to access restricted areas. The popularity of this biometric with respect of others such as face and iris recognition lies on its reliability, that has been proven during the last decades, its implementation at an affordable cost, and especially on the simplicity of touching a surface to get immediately authenticated.

Unfortunately, all these advantages come with various security and privacy issues. Different attacks can be performed to the authentication system in order to grant access to some exclusive area or to steal confidential data. For instance, the software and the network configuration can have security holes or bugs, and the matching algorithms can be fooled if the attacker knows the software implementation details [2]. Moreover, whereas a physical key or badge can be replaced, fingerprints are permanent and the pattern on their surface can be easily captured and reproduced. It can be taken from a high-resolution photograph or from a print left on a surface such as a mug or even a piece of paper. A high-quality reproduction of the pattern on some gummy material can be simply applied to the scanner [3] so that the attacker can fool the authentication system by declaring its corresponding identity. Since the sensor device is inevitably at a direct contact of the user being captured, it is considered one of the weakest point in the entire biometrics system. Because of this, there is a growing interest in automatically analyzing the acquired fingerprint images in order to catch potential malignant users [4]. This kind of attacks are known as presentation attacks [5], and liveness detection techniques are designed to spot them by formulating a binary classification problem with the aim of establishing whether a given biometrics comes from the subject present at the time of capture [6].

The liveness of a fingerprint can be established by designing a software system that analyzes the same images used by the recognition algorithm, or by equipping the scanner with additional hardware. These last prevention measures are called hardware-based systems [7] and are generally more accurate since they take advantage of additional cues. Anyway, the software of a fingerprint scanner can be updated with no additional cost, and if a software technique is robust to a variety of attacks and does not annoy the users with too many false positives, it can be an alternative with regard to acquiring new sensing devices.

Recently, different studies [8–10] have shown the effectiveness of deep learning algorithms for the task of fingerprint liveness detection. Deep learning has seriously improved the state of the art in many fields such as speech recognition, natural language processing, and object recognition [11–13]. The ability to generate hierarchical representations and discover complex structures in raw images allows for better representations with respect to traditional methods based on handcrafted features. Software-based systems for fingerprint liveness detection can take advantage of the very broad literature where similar tasks have already been addressed. Among the recent works, we noticed that it has not yet directly modeled a notion of similarity

among real and fake fingerprints that can capture the underlying factors that explain their inter- and intra-class variations. We make a step in this direction by proposing a deep metric learning approach based on Triplet networks [14]. Specifically, these networks map the fingerprint images into a representation space, where the learned distance captures the similarity of the examples coming from the same class and push away the real samples from the fake ones. Unlike other metric learning approaches, such as the ones involving Siamese networks [15], the triplet objective function puts in direct comparison the relation among the classes, giving a notion of context that does not require a threshold selection in order make the prediction [14].

We propose a framework that learns a representation from fingerprint patches, starting from a set of real and fake samples given as a training set. Since at test time only a fingerprint image is given, we make our decision on the basis of a matching score, computed against a set of real and fake patches given as a reference. The similarity metric is learned using an improved version [16, 17] of the original triplet objective formulation [14] that adds a pairwise term that more firmly forces the closeness of two examples of the same class. We performed extensive experiments using ten datasets taken from the fingerprint liveness detection competitions LivDet[1] 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, held in the years 2009 [7], 2011 [18] and 2013 [19]. We compare our approach with respect to the state of the art, getting competitive performance for all the examined datasets. We also perform the cross-dataset and cross-material experiments, in order to evaluate if the obtained fingerprint representation can be reused in different settings or to spot materials that have not been seen during training.

The chapter is structured as follows. In Sect. 12.2 we present some of the current approaches for designing fingerprint liveness detection systems, and the current state of the art. In Sect. 12.3 we explain the details of the proposed framework and in Sect. 12.4 we provide the experimental results. The final Sect. 12.5 is dedicated to the conclusions.

12.2 Background and Previous Work

In this section, we describe various fingerprint liveness detection techniques, particularly with a focus on the ones related to our method, which can be considered as a static software-based technique [6]. Subsequently, we provide details on some of the most recently proposed software-based approaches in order to contextualize our method and highlight our contributions.

[1] http://livdet.org/.

12.2.1 Background

A first categorization of liveness detection systems can be made by distinguishing between hardware and software systems. As already mentioned, hardware-based systems, also called sensor-based [6], use additional information in order to spot the characteristics of living human fingerprints. Useful clues can be found for instance by detecting the pattern of veins underlying the fingerprints surface [20], measuring the pulse [21], by performing odor analysis [22] and by employing near-infrared sensors [23].

Software, also called feature-based systems, are algorithms that can be introduced into a sensor software in order to add the liveness detection functionality. Our approach falls in this category and can also be considered static, to distinguish it from methods that use multiple images taken during the acquisition process. For instance, dynamic methods can exploit the distortion of the fingertip skin since with respect to gummy materials, it differs in terms of flexibility. In the approach proposed by [24], the user is asked to rotate the finger while touching the sensor. The distortion of the different regions at a direct contact of the sensor are characterized in terms of optical flow and the liveness prediction is based on matching the encoded distortion code sequences over time.

In order to design a software-based system based on machine learning algorithms, a database of real and fake examples of fingerprints is needed. The more spoofing techniques are used, the more the algorithm will be able to generalize to new kind of attacks. In a second categorization of liveness detection systems, we consider how the fingertip pattern is taken from the victim. In the cooperative method, the victim voluntarily puts his/her finger on some workable material that is used to create a mold. From this mold, it is possible to generate high-quality reproductions by filling it with materials such as gelatin, silicone, and wooden glue. Figure 12.1 shows some photographs of artificial fingers. Noncooperative methods instead, capture the scenarios where the pattern has been taken from a latent fingerprint. After taking a high-resolution picture, it is reproduced by generating a three-dimensional surface,

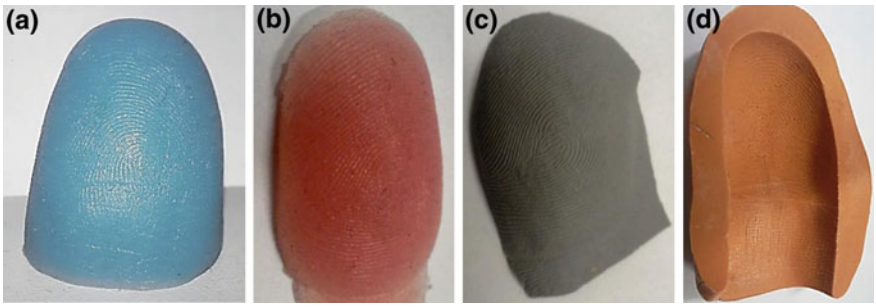

Fig. 12.1 Examples of artificial finger replicas made using different silicon rubber materials: **a** GLS, **b** Ecoflex, **c** Liquid Ecoflex and **d** a Modasil mold

Fig. 12.2 Examples of fake acquisitions from the LivDet 2011 competition (cooperative). From Biometrika **a** Latex, **b** Silicone, from Digital **c** Latex, **d** Gelatine, from Sagem **e** Silicone, **f** Play-doh

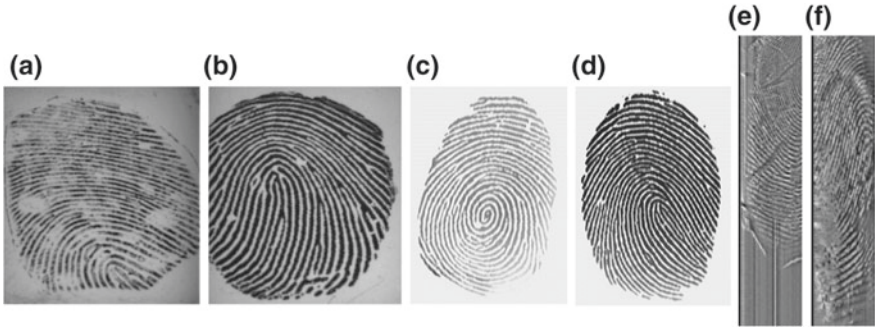

Fig. 12.3 Examples of fake acquisitions from the LivDet 2013 competition. Noncooperative: from Biometrika **a** Gelatine, **b** Wooden Glue, from Italdata **c** Ecoflex, **d** Modasil. Cooperative: from Swipe **e** Latex, **f** Wooden Glue

for instance, by printing it into a circuit board. At this point, a mold can be generated and filled with the above-mentioned materials. The quality of images is inferior as compared to the cooperative methods, and usually, software-based systems have better performance on rejecting these reproductions. Figures 12.2 and 12.3 show several acquisitions, where the fingertip pattern has been captured using the cooperative and noncooperative methods.

12.2.2 Previous Work

In this subsection, we discuss some of the previous work on static software-based fingerprint liveness detection systems. We start by presenting some hand crafted feature-based approaches and conclude with the more recently proposed deep learning techniques.

One of the first approaches to fingerprint liveness detection has been proposed by [25]. It is based on the perspiration pattern of the skin that manifests itself into static and dynamic patterns on the dielectric mosaic structure of the skin. The classification is based on a set of measures extracted from the data and classified using a neural

network. In [26], the same phenomenon is captured by a wavelet transform applied to the ridge signal extracted along the ridge mask. After extracting a set of measures from multiple scales, the decision rule is based on classification trees.

Other approaches build some descriptors from different kind of features that are conceived specifically for fingerprint images. In [27] Local binary pattern (LBP) histograms are proposed along with wavelet energy features. The classification is performed by a hybrid classifier composed of a neural network, a support vector machine (SVM) and a k-nearest neighbor classifier. Similar to LBP, the Binary Statistical Image Features (BSIF) [28], encode local textural characteristics into a feature vector and SVM is used for classification. In [29] a Local Contrast Phase Descriptor (LCPD) is extracted by performing image analysis in both the spatial and frequency domains. The same authors propose the use of the Shift-Invariant Descriptor (SID) [30], originally introduced by [31], which provides rotation and scale invariance properties. SID, along with LCPD provides competitive and robust performances on several datasets.

Recently, deep learning algorithms have been applied to fingerprint liveness detection with the aim of automatically finding a hierarchical representation of fingerprints directly from the training data. In [9] the use of convolutional networks has been proposed. In particular, the best results [9] are obtained by fine-tuning the AlexNet and VGG architectures proposed by [11, 32], previously trained on the Imagenet dataset of natural images [33]. From their experimental results, it seems that the factors that most influence the classification accuracy are the depth of the network, the pretraining and the data augmentation they performed in terms of random crops. Since we use a patch-based representation we employ a smaller, but reasonably deep architecture. The use of patches does not require resizing all the images to a fixed dimension, and at the same time, the number of examples is increased so that pretraining can be avoided.

In [8], deep representations are learned from fingerprint, face, and iris images. They are used as traditional features and fed into SVM classifiers to get a liveness score. The authors focus on the choice of the convolutional neural network parameters and architecture.

In [34] deep Siamese networks have been considered along with classical pretrained convolutional networks. This can be considered the most similar work to this chapter since they also learn a similarity metric between a pair of fingerprints. However, their use of metric learning is different since they assume a scenario where the enrollment fingerprint is available for each test image. That is, the decision is made by comparing fingerprints of the same individual. Our approach instead is more versatile and can be applied even if the enrollment fingerprint image is not available.

Different from the above studies, [10, 35] do not give the entire image to the deep learning algorithm but extract patches from the fingerprint acquisition after removing the background. [35] uses classical ConvNets with a binary cross-entropy loss, along with a majority voting scheme to make the final prediction. [10] proposes deep belief networks and use contrastive divergence [36] for pretraining and fine-tunes on the real and fake fingerprint images. The decision is based on a simple threshold applied to the output of the network. Our work is substantially different because it proposes

a framework where triplet architectures are used along with a triplet and pairwise objective function.

Summarizing, the contribution of this chapter are (i) a novel deep metric learning based framework, targeted to fingerprint liveness detection, able to work in real time with state-of-the-art performance, (ii) a patch-based and fine-grained representation of the fingerprint images that makes it possible to train a reasonably deep architecture from scratch, even with few hundreds of examples, and that shows superior performance even in settings different from the ones used for training.

12.3 A Deep Triplet Embedding for Fingerprint Liveness Detection

In this section, we describe the proposed method for fingerprint liveness detection based on triplet loss embedding. We start by describing the overall framework; subsequently, we introduce the network architecture and the training algorithm. Finally, we describe the matching procedure that leads to the final decision on the liveness of a given fingerprint image.

12.3.1 Framework

As depicted in Fig. 12.4, the proposed framework requires a collection of real and fake fingerprint images taken from a sensor and used as a training set. From each image, we randomly extract one fixed sized patch and arrange them in a certain number of triplets $\{x_i, x_j^+, x_k^-\}$, where x_i (anchor) and x_j^+ are two examples of the same class, and x_k^- comes from the other class. We alternatively set the anchor to be a real or a fake fingerprint patch.

The architecture is composed of three convolutional networks with shared weights so that three patches can be processed at the same time and mapped into a common feature space. We denote by $\mathbf{r}(\cdot)$ the representation of a given patch obtained from the output of one of the three networks. The deep features extracted from the live and fake fingerprints are compared in order to extract an intra-class distance $d(\mathbf{r}(x), \mathbf{r}(x^+))$ and an inter-class distance $d(\mathbf{r}(x), \mathbf{r}(x^-))$. The objective is to learn d so that the two examples of the same class are closer than two examples taken from different classes, and the distance between two examples of the same class is as short as possible. After training the networks with a certain number of triplets, we extract a new patch from each training sample and generate a new set of triplets. This procedure is repeated until convergence, see more details in Sect. 12.4.2.

After the training procedure is completed, the learned metric is used as a matching distance in order to establish the liveness of a new fingerprint image. Given a query fingerprint, we can extract p (possibly overlapping) patches and give them as input to

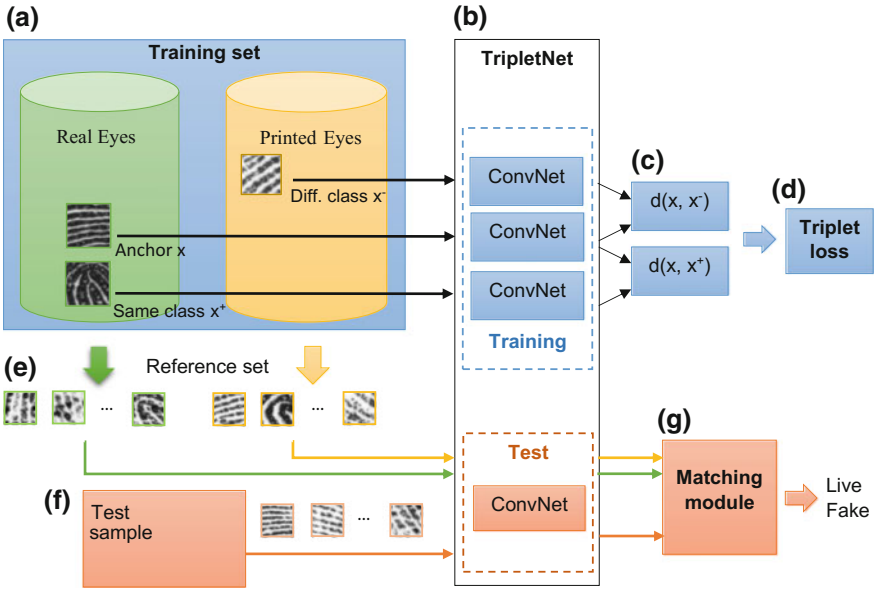

Fig. 12.4 The overall architecture of the proposed fingerprint liveness detection system. From the training set **a** of real and fake fingerprint acquisitions, we train a triplet network **b** using alternatively two patches of one class and one patch of the other one. The output of each input patch is used to compute the inter- and intra-class distances **c** in order to compute the objective function **d** that is used to train the parameters of the networks. After training, a set of real and a set of fake reference patches **e** are extracted from the training set (one for each fingerprint) and the corresponding representation is computed forwarding them through the trained networks. At test time, a set of patches is extracted from the fingerprint image **f** in order to map it to the same representation space as the reference gallery and are matched **g** in order to get a prediction on its liveness

the networks in order to get a representation $Q = \{\mathbf{r}(Q_1), \mathbf{r}(Q_2), \ldots, \mathbf{r}(Q_p)\}$. Since we are not directly mapping each patch to a binary liveness label, but generating a more fine-grained representation, the prediction can be made by a decision rule based on the learned metric d computed with respect to a set R_L and R_F of real and fake reference fingerprints:

$$R_L = \{\mathbf{r}(x_{L_1}), \mathbf{r}(x_{L_2}), \ldots, \mathbf{r}(x_{L_n})\} \tag{12.1a}$$

$$R_F = \{\mathbf{r}(x_{F_1}), \mathbf{r}(x_{F_2}), \ldots, \mathbf{r}(x_{F_n})\} \tag{12.1b}$$

where the patches x_{L_i} and x_{F_i} can be taken from the training set or from a specially made design set.

Table 12.1 Architecture of the proposed embedding network

Layer name	Layer description	Output
input	32×32 gray level image	
conv1	5×5 conv. filters, stride 1, $1 \rightarrow 64$ feat. maps	$64 \times 28 \times 28$
batchnorm1	Batch normalization	$64 \times 28 \times 28$
relu1	Rectifier linear unit	$64 \times 28 \times 28$
conv2	3×3 conv. filters, stride 2, padding 1, $64 \rightarrow 64$ feat. maps	$64 \times 14 \times 14$
conv3	3×3 conv. filters, stride 1, $64 \rightarrow 128$ feat. maps	$64 \times 12 \times 12$
batchnorm2	Batch normalization	$64 \times 12 \times 12$
relu2	Rectifier linear unit	$64 \times 12 \times 12$
conv4	3×3 conv. filters, stride 2, padding 1, $128 \rightarrow 128$ feat. maps	$128 \times 6 \times 6$
conv5	3×3 conv. filters, stride 1, $128 \rightarrow 256$ feat. maps	$256 \times 4 \times 4$
batchnorm3	Batch normalization	$256 \times 4 \times 4$
relu3	Rectifier linear unit	$256 \times 4 \times 4$
conv6	3×3 conv. filters, stride 2, padding 1, $256 \rightarrow 256$ feat. maps	$256 \times 2 \times 2$
fc1	Fully connected layer $4 \times 256 \rightarrow 256$	256
dropout	Dropout $p = 0.4$	256
relu5	Rectifier linear unit	256
fc2	Fully connected layer $256 \rightarrow 256$	256
output	Softmax	256

12.3.2 Network Architecture

We employ a network architecture inspired by [37] where max pooling units, widely used for downsampling purposes, are replaced by simple convolution layers with increased stride. Table 12.1 contains the list of the operations performed by each layer of the embedding networks.

The architecture is composed of a first convolutional layer that takes the 32×32 grayscale fingerprint patches and outputs 64 feature maps by using filters of size 5×5. Then, batch normalization [38] is applied in order to get a faster training convergence and rectified linear units (ReLU) are used as nonlinearities. Another convolutional layer with a stride equal to 2, padding of 1 and filters of size 3×3 performs a downsampling operation by a factor of two in both directions.

The same structure is replicated two times, reducing the filter size to 3×3 and increasing the number of feature maps from 64 to 128 and from 128 to 256. At this point, the feature maps have the size of $128 \times 2 \times 2$ and are further processed by two fully connected layers with 256 outputs followed by a softmax layer. This nonlinearity helps in getting a better convergence of the training algorithm and ensures that the

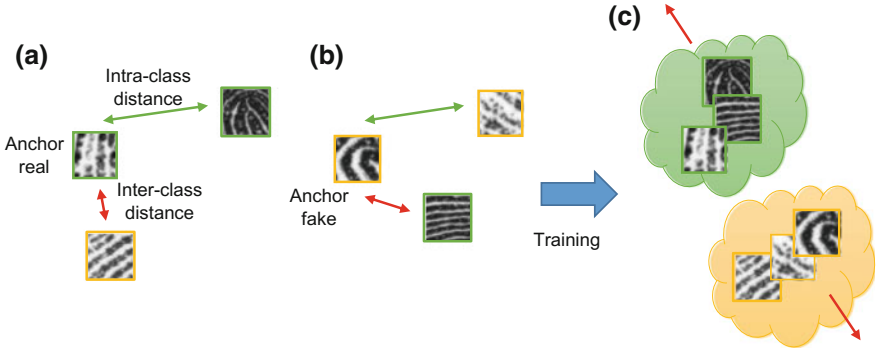

Fig. 12.5 The training procedure uses examples as triplets formed by **a** two real fingerprints (in *green*) and one impostor (in *yellow*) and **b** two impostors and one genuine. The training procedure using the triplet loss will result in an attraction for the fingerprints of the same class (either real or fake) so that their distance will be as close as possible. At the same time, real and fake fingerprints will be pushed away from each other (**c**)

distance among to outputs does not exceed one. Dropout [39] with probability 0.4 is applied to the first fully connected layer for regularization purposes.

The complete network is composed of three instances of this architecture: from three batches of fingerprint images we get the L2 distances between the matching and mismatching images. At test, we take the output of one of the three networks to obtain the representation for a given patch. If there are memory limitations, an alternative consists of using just one network, collapse the three batches into a single one, and computing the distances among the examples corresponding to the training triplets.

12.3.3 Training

As schematized in Fig. 12.5, the triplet architecture along with the triplet loss function aims to learn a metric that makes two patches of the same class closer with respect to two coming from different classes. The objective is to capture the cues that make two fingerprints both real or fake. The real ones come from different people and fingers, and their comparison is performed in order to find some characteristics that make them genuine. At the same time, fake fingerprints come from different people and can be built using several materials. The objective is to detect anomalies that characterize fingerprints coming from a fake replica, without regard to the material they are made of.

Given a set of triplets $\{x_i, x_j^+, x_k^-\}$, where x_i is the anchor and x_j^+ and x_k^- are two examples of the same and the other class, respectively, the objective of the original triplet loss [14] is to give a penalty if the following condition is violated:

$$d(\mathbf{r}(x_i), \mathbf{r}(x_j^+)) - d(\mathbf{r}(x_i), \mathbf{r}(x_k^-)) + 1 \leq 0 \tag{12.2}$$

At the same time, we would like to have the examples of the same class as close as possible so that, when matching a new fingerprint against the reference patches of the same class, the distance $d(\mathbf{r}(x_i), \mathbf{r}(x_j^+))$ is as low as possible. If we denote by $y(x_i)$ the class of a generic patch x_i, we can obtain the desired behavior by formulating the following loss function:

$$L = \sum_{i,j,k} \left\{ c(x_i, x_j^+, x_k^-) + \boldsymbol{\beta} c(x_i, x_j^+) \right\} + \lambda \|\boldsymbol{\theta}\|_2 \tag{12.3}$$

where θ is a one-dimensional vector containing all the trainable parameters of the network, $y(x_i) = y(x_j)$, $y(x_k^-) \neq y(x_i)$ and

$$c(x_i, x_j^+, x_k^-) = \max \left\{ 0, d(\mathbf{r}(x_i), \mathbf{r}(x_j^+)) - d(\mathbf{r}(x_i), \mathbf{r}(x_k^-)) + 1 \right\} \tag{12.4a}$$

$$c(x_i, x_j^+) = d(\mathbf{r}(x_i), \mathbf{r}(x_j^+)) \tag{12.4b}$$

During training, we compute the subgradients and use backpropagation through the network in order to get the desired representation. Contextualizing to what depicted in Fig. 12.5, $c(x_i, x_j^+, x_k^-)$ is the inter-class and $c(x_i, x_j^+)$ the intra-class distance term. $\lambda \|\theta\|_2$ is an additional weight decay term added to the loss function for regularization purposes.

After a certain number of iterations k, we periodically generate a new set of triplets by extracting a different patch from each training fingerprint. It is essential to not update the triplets after too many iterations because it can result in overfitting. At the same time, generating new triplets too often or mining hard examples can cause convergence issues.

12.3.4 Matching

In principle, any distance among bag of features can be used in order to match the query fingerprint $Q = \{\mathbf{r}(Q_1), \mathbf{r}(Q_2), \ldots, \mathbf{r}(Q_p)\}$ against the reference sets R_L and R_F. Since the training objective drastically pushes the distances to be very close to zero or to one, a decision on the liveness can be made by setting a simple threshold $\tau = 0.5$. An alternative could consists of measuring the Hausdorff distance between bags, but it would be too much sensitive to outliers since it involves the computation of the minimum distance between a test patch and each patch of each reference set. Even if using the k-th Hausdorff distance [40], that considers the k-th value instead of the minimum, we obtained better performance by following a simple majority voting strategy. It is also faster since it does not involve sorting out the distances.

Given a fingerprint Q, for each patch Q_j we count how many distances for each reference set are below the given threshold

$$D(R_L, Q_j) = |\{i \in \{1, \ldots, n\} : d(R_{L_i}, Q_j) < \tau\}| \tag{12.5a}$$

$$D(R_F, Q_j) = |\{i \in \{1, \ldots, n\} : d(R_{F_i}, Q_j) < \tau\}| \tag{12.5b}$$

then we make the decision evaluating how many patches belong to the real or the fake class:

$$y(Q) = \begin{cases} \text{real} & \text{if } \sum_{j=1}^{p} D(R_L, Q_j) \geq \sum_{j=1}^{p} D(R_F, Q_j) \\ \text{fake} & \text{otherwise} \end{cases} \tag{12.6}$$

The above method can also be applied in scenarios where multiple fingerprints are acquired from the same individual, as usually happens on passport checks at airports. For instance, the patches coming from different fingers can be accumulated in order to apply the same majority rule of Eq. 12.6 or the decision can be made on the most suspicious fingerprint.

12.4 Experiments

We evaluated the proposed approach with ten of the most popular benchmark for fingerprint liveness detection, coming from the LivDet competitions held in 2009 [7], 2011 [18] and 2013 [19]. We compare our method with the state of the art, specifically the VGG pretrained network of [9], the Local Contrast Phase Descriptor LCPD [29], the dense Scale Invariant Descriptor SID [30] and the Binarized Statistical Image Features [28]. For the main experiments, we strictly follow the competition rules using the training/test splits provided by the organizers while for the cross-dataset and cross-material scenarios, we follow the setup of [9].

The network architecture along with the overall framework have been implemented using the Torch7 computing framework [41] on an NVIDIA® DIGITS™ DevBox with four TITAN X GPUs with seven TFlops of single precision, 336.5 GB/s of memory bandwidth, and 12 GB of memory per board. MATLAB® has been used for image segmentation.

12.4.1 Datasets

The LivDet 2009 datasets [7] were released with the first international fingerprint liveness detection competition, with the aim of becoming a reference and allowing researchers to compare the performance of their algorithms or systems. The fingerprints were acquired using the cooperative approach (see Sect. 12.2.1) and the replicas are created using the materials: gelatin, silicone, and play-doh. The organizers released three datasets, acquired using three different sensors: Biometrika (FX2000), Identix (DFR2100), and Crossmatch (Verifier 300 LC).

Table 12.2 Details of the LivDet 2009 and 2013 competitions. The last row indicates the spoof materials: S = Silicone, G = Gelatine, P = Play-Doh, E = Ecoflex, L = Latex, M = Modasil, B = Body Double, W = Wooden glue

Competition	LivDet2009			LivDet2013		
Dataset	Biometrika	CrossMatch	Identix	Biometrika	Italdata	Swipe
Size	312 × 372	480 × 640	720 × 720	312 × 372	480 × 640	1500 × 208
DPI	569	500	686	569	500	96
Subjects	50	254	160	50	50	100
Live samples	2000	2000	1500	2000	2000	2500
Spoof samples	2000	2000	1500	2000	2000	2000
Materials	S	GPS	GPS	EGLMW	EGLMW	BLPW

The LivDet 2011 competition [18] released four datasets, acquired using the scanners Biometrika (FX2000), Digital Persona (4000B), ItalData (ETT10) and Sagem (MSO300). The materials used for fake fingerprints are gelatin, latex, Ecoflex (platinum-catalyzed silicone), silicone and wooden glue. The spoof fingerprints have been obtained as in LivDet 2009 with the cooperative method.

The LivDet 2013 competition [19] consists of four datasets acquired using the scanners Biometrika (FX2000), ItalData (ETT10), Crossmatch (L SCAN GUARDIAN) and Swipe. Differently from LivDet 2011, two datasets, Biometrika and Italdata, have been acquired using the non-cooperative method. That is, latent fingerprints have been acquired from a surface, and then printed on a circuit board (PCB) in order to generate a three-dimensional structure of the fingerprint that can be used to build a mold. To replicate the fingerprints they used Body Double, latex, PlayDoh and wood glue for the Crossmatch and Swipe datasets and gelatin, latex, Ecoflex, Modasil and wood glue for Biometrika and Italdata.

The size of the images, the scanner resolution, the number of acquired subject and of live and fake samples are detailed in Tables 12.2 and 12.3. The partition of training and test examples is provided by the organizers of the competition.

12.4.2 Experimental Setup

For all the experiments we evaluate performance in terms of average classification error. This is the measure used to evaluate the entries in the LivDet competitions and is the average of the Spoof False Positive Rate (SFPR) and the Spoof False Negative Rate (SFNR). For all the experiments on the LivDet test sets we follow the standard protocol and since a validation set is not provided, we reserved a fixed amount of 120 fingerprints. For the cross-dataset experiments, we used for validation purposes the Biometrika 2009 and Crossmatch 2013 datasets.

Table 12.3 Details of the LivDet 2011 competition. The last row indicates the spoof materials: Sg = Silgum, the others are the same as in Table 12.2

Competition	LivDet2011			
Dataset	Biometrika	Digital persona	Italdata	Sagem
Size	312 × 372	355 × 391	640 × 480	352 × 384
DPI	500	500	500	500
Subjects	200	82	92	200
Live samples	2000	2000	2000	2000
Spoof samples	2000	2000	2000	2000
Materials	EGLSgW	GLPSW	EGLSgW	GLPSW

The triplets set for training is generated by taking one patch from each fingerprint and arranging them alternatively in two examples of one class and one of the other class. The set is updated every $k = 100{,}000$ triplets that are fed to the networks in batches of 100. In the remainder of the chapter, we refer to each update as the start of a new iteration. We use stochastic gradient descent to minimize the triplet loss function, setting a learning rate of 0.5 and a momentum of 0.9. The learning rate η_0 is annealed by following the form:

$$\eta = \frac{\eta_0}{1 + 10^{-4} \cdot b} \tag{12.7}$$

where b is the progressive number of batches that are being processed. That is, after ten iterations the learning rate is reduced by half. The weight decay term of Eq. 12.3 is set to $\lambda = 10^{-4}$ and $\beta = 0.002$ as in [17].

After each iteration, we check the validation error. Instead of using the same accuracy measured at test (the average classification error), we construct 100,000 triplets using the validation set patches, but taking as anchor the reference patches taken from the training set and used to match the test samples. The error consists of the number of violating triplets and reflects how much the reference patches failed to classify patches never seen before. Instead of fixing the number of iterations, we employ early stopping based on the concept of patience [42]. Each time the validation error decrease, we save a snapshot of the network parameters, and if in 20 consecutive iterations the validation error is not decreasing anymore, we stop the training and evaluate the accuracy on the test set using the last saved snapshot.

12.4.3 Preprocessing

Since the images coming from the scanners contain a wide background area surrounding the fingerprint, we segmented the images in order to avoid extracting background patches. The performance is highly affected by the quality of the background

subtraction, therefore, we employed an algorithm [43], that divides the fingerprint image into 16×16 blocks, and classifies a block as foreground only if its standard deviation is more than a given threshold. The rationale is that a higher standard deviation corresponds to the ridge regions of a fingerprint. In order to exclude background noise that can interfere with the segmentation, we compute the connected components of the foreground mask and take the fingerprint region as the one with the largest area. In order to get a smooth segmentation, we generate the convex hull image from the binary mask using morphological operations.

We also tried to employ data augmentation techniques in terms of random rotations, flipping and general affine transformation. Anyway, they significantly slowed down the training procedure and we did not get any performance improvement on either the main and cross-dataset experiments.

12.4.4 Experimental Results

In this section, we present the performance of the proposed fingerprint liveness detection system in different scenarios. In Table 12.4 we list the performance in terms of average classification error on the LivDet competition test sets. With respect to the currently best-performing methods [9, 29, 30] we obtained competitive performance for all the datasets, especially on Italdata 2011, and Swipe 2013. This means that the approach works properly also on the images coming from swipe scanners, where the fingerprints are acquired by swiping the finger across the sensor surface (see Fig. 12.3e, f). Overall, our approach has an average error of 1.75% in comparison

Table 12.4 Average classification error for the LivDet Test Datasets. In column 2 our TripletNet based approach, in column 2 the VGG deep network pretrained on the Imagenet dataset and fine-tuned by [9], in column 3 the Local Contrast Phase Descriptor [29] based approach, in column 4 the dense Scale Invariant Descriptor [30] based approach and in column 5 the Binarized Statistical Image Features [28] based approach

Dataset	TripletNet	VGG [9]	LCPD [29]	SID [30]	BSIF [28]
Biometrika 2009	**0.71**	4.1	1	3.8	9
CrossMatch 2009	1.57	**0.6**	3.4	3.3	5.8
Identix 2009	**0.044**	0.2	1.3	0.7	0.7
Biometrika 2011	5.15	5.2	**4.9**	5.8	6.8
Digital 2011	**1.85**	3.2	4.7	2.0	4.1
Italdata 2011	**5.1**	8	12.3	11.2	13.9
Sagem 2011	**1.23**	1.7	3.2	4.2	5.6
Biometrika 2013	**0.65**	1.8	1.2	2.5	1.1
Italdata 2013	0.5	**0.4**	1.3	2.7	3
Swipe 2013	**0.66**	3.7	4.7	9.3	5.2
Average	**1.75%**	2.89%	3.8%	4.5%	5.5%

to the 2.89% of [9] which results in a performance improvement of 65%. We point out that we did not use the dataset CrossMatch 2013 for evaluation purposes because the organizers of the competition found anomalies in the data and discouraged its use for comparative evaluations [4]. In Fig. 12.6 we depict a 2D representation of the test set of Biometrika 2013, specifically one patch for every fingerprint image, computed from an application of t-SNE [44] to the generated embedding. This dimensionality reduction technique is particularly insightful since it maps the high-dimensional representation in a space where the vicinity of points is preserved. We can see that the real and fake fingerprints are well separated and only a few samples are in the wrong place, for the major part Wooden Glue and Modasil. Ecoflex and gelatin replicas seem more easy to reject. Examining the patch images, we can see that going top to bottom, the quality of the fingerprint pattern degrades. This may be due to the perspiration of the fingertips that makes the ridges not as uniform as the fake replicas.

12.4.4.1 Cross-Dataset Evaluation

As in [9], we present some cross-dataset evaluation and directly compare our performance with respect to their deep learning and Local Binary Pattern approach. The results are shown in Table 12.5 and reflect a significant drop in performance with respect to the previous experiments. With respect to [9] the average classification error is slightly better, anyway it is too high to possibly consider doing liveness detection in the wild. Similar results have been obtained by [34]. We point out that different sensors, settings and climatic conditions can extremely alter the fingerprint images, and if the training set is not representative of the particular conditions, any machine learning approach, not just deep learning algorithms, would not be effective at generalization.

12.4.4.2 Cross-Material Evaluation

We also evaluated the robustness of our system to new spoofing materials. We followed the protocol of [9] by training the networks using a subset of materials and testing on the remaining ones. The results are shown in Table 12.6. With respect to the cross-dataset experiments, the method appears to be more robust to new materials rather than a change of the sensor. Also in this scenario, if we exclude the Biometrika 2011 dataset, our approach has a significative improvement with respect to [9].

12.4.4.3 Computational Efficiency

One of the main benefits of our approach is the computational time since the architecture we employed is smaller in comparison to other deep learning approaches such as [9, 34]. Moreover, the patch representation allows for scaling the matching

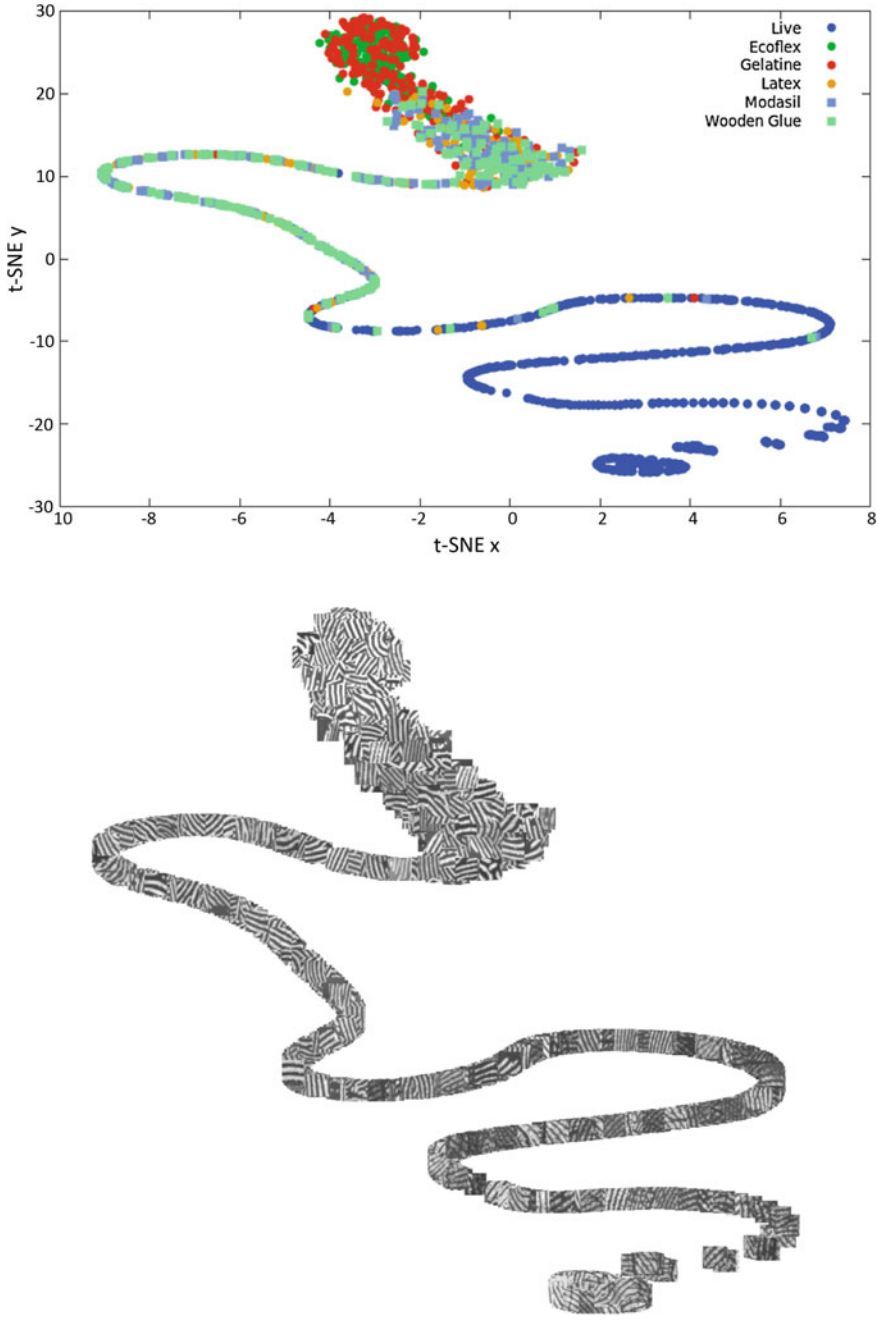

Fig. 12.6 T-SNE visualization of the embedding generated from the live and fake fingerprints composing the test set of Biometrika 2013 (one patch for each acquisition). The high dimensional representation is mapped into a two-dimensional scatter plot where the vicinity of points is preserved

Table 12.5 Average classification error for the cross-dataset scenarios. The first column is the dataset used for training and the second the one used for the test. The third column is our TripletNet approach, the fourth and the fifth are the deep learning and the Local Binary Patterns (LBP) based approaches proposed by [9]

Training set	Test set	TripletNet	VGG	LBP
Biometrika 2011	Biometrika 2013	14	15.5	16.5
Biometrika 2013	Biometrika 2011	34.05	46.8	47.9
Italdata 2011	Italdata 2013	8.3	14.6	10.6
Italdata 2013	Italdata 2011	44.65	46.0	50.0
Biometrika 2011	Italdata 2011	29.35	37.2	47.1
Italdata 2011	Biometrika 2011	27.65	31.0	49.4
Biometrika 2013	Italdata 2013	1.55	8.8	43.7
Italdata 2013	Biometrika 2013	3.8	2.3	48.4

Table 12.6 Average Classification Error for the cross-material scenario. In column 2 are the materials used for training and in column 3 the ones used for the test. The abbreviations are the same as in Tables 12.2 and 12.3

Dataset	Train materials	Test materials	TripletNet	VGG	LBP
Biometrika 2011	EGL	SgW	13.1	10.1	17.7
Biometrika 2013	MW	EGL	2.1	4.9	8.5
Italdata 2011	EGL	SgW	7	22.1	30.9
Italdata 2013	MW	EGL	1.25	6.3	10.7

procedure on different computational units, so that it can be used also in heavily populated environments. In our experiments, we extract 100 patches from each test fingerprint and the time to get their corresponding representation is about 0.6ms using a single GPU and 0.3 s using a Core i7-5930K 6 Core 3.5 GHz desktop processor (single thread). Considering the most common dataset configuration of 880 real and 880 fake reference patches, the matching procedure takes 5.2 ms on a single GPU and 14 ms on the CPU. Finally, the training time varies depending on the particular dataset, and on the average, the procedure converges in 135 iterations. A single iteration takes 84 and 20 s are needed to check the validation error.

12.5 Conclusions

In this chapter, we introduced a novel framework for fingerprint liveness detection which embeds the recent advancements in deep metric learning. We validated the effectiveness of our approach in a scenario where the fingerprints are acquired using the same sensing devices that are used for training. We also presented quantified

results on the generalization capability of the proposed approach for new acquisition devices, and unseen spoofing materials. The approach is able to work in real time and surpasses the state-of-the-art on several benchmark datasets.

In conclusion, we point out that the employment of software-based liveness detection systems should never give a sense of false security to their users. As in other areas such as cyber-security, the attackers become more resourceful every day and new ways to fool a biometric system will be discovered. Therefore, such systems should be constantly updated and monitored, especially in critical applications such as airport controls. It would be desirable to have large datasets that contain fingerprint images of people with different age, sex, ethnicity, and skin conditions and that are acquired under different time periods, environments and using a variety of sensors with a multitude of spoofing materials.

Acknowledgements This work was supported in part by grant N00014-12-1-1026.

References

1. A.K. Jain, A. Ross, S. Pankanti, Biometrics: a tool for information security. IEEE Trans. Inf. Forensics Secur. **1**(2), 125–143 (2006)
2. M. Martinez-Diaz, J. Fierrez, J. Galbally, J. Ortega-Garcia, An evaluation of indirect attacks and countermeasures in fingerprint verification systems. Pattern Recogn. Lett. **32**(12), 1643–1651 (2011)
3. T. Matsumoto, H. Matsumoto, K. Yamada, S. Hoshino, Impact of artificial "gummy" fingers on fingerprint systems. Proc. Conf. Opt. Secur. Counterfeit Deterrence Tech. **4677**, 275–289 (2002)
4. L. Ghiani, D.A. Yambay, V. Mura, G.L. Marcialis, F. Roli, S.A. Schuckers, Review of the fingerprint liveness detection (livdet) competition series: 2009 to 2015. *Image and Vision Computing* (2016) (in press)
5. C. Sousedik, C. Busch, Presentation attack detection methods for fingerprint recognition systems: a survey. IET Biom. **3**(4), 219–233 (2014)
6. J. Galbally, S. Marcel, J. Fierrez, Biometric antispoofing methods: a survey in face recognition. IEEE Access **2**, 1530–1552 (2014)
7. G.L. Marcialis, A. Lewicke, B. Tan, P. Coli, D. Grimberg, A. Congiu, A. Tidu, F. Roli, S. Schuckers, First international fingerprint liveness detection competition – livdet 2009, in *Proceedings of the International Conference on Image Analysis and Processing (ICIAP, 2009)* (Springer, Berlin, 2009), pp. 12–23
8. D. Menotti, G. Chiachia, A. Pinto, W.R. Schwartz, H. Pedrini, A.X. Falcao, A. Rocha, Deep representations for iris, face, and fingerprint spoofing detection. IEEE Trans. Inf. Forensics Secur. **10**(4), 864–879 (2015)
9. R.F. Nogueira, R. de Alencar Lotufo, R.C. Machado, Fingerprint liveness detection using convolutional neural networks. IEEE Trans. Inf. Forensics Secur. **11**(6), 1206–1213 (2016)
10. S. Kim, B. Park, B.S. Song, S. Yang, Deep belief network based statistical feature learning for fingerprint liveness detection. Pattern Recogn. Lett. **77**, 58–65 (2016)
11. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Conference on Advances in Neural Information Processing Systems (NIPS, 2012)* (2012), pp. 1097–1105
12. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436–444 (2015)
13. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016), http://www.deeplearningbook.org

14. E. Hoffer, N. Ailon, Deep metric learning using triplet network, in *Proceedings of the International Workshop on Similarity-Based Pattern Recognition (SIMBAD, 2015)* (Springer, Berlin, 2015), pp. 84–92
15. J. Bromley, J.W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, R. Shah, Signature verification using a "siamese" time delay neural network. Int. J. Pattern Recognit. Artif. Intell. **7**(04), 669–688 (1993)
16. P. Wohlhart, V. Lepetit, Learning descriptors for object recognition and 3D pose estimation, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR, 2015)* (2015), pp. 3109–3118
17. D. Cheng, Y. Gong, S. Zhou, J. Wang, N. Zheng, Person re-identification by multi-channel parts-based cnn with improved triplet loss function, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR, 2016)* (2016)
18. D. Yambay, L. Ghiani, P. Denti, G.L. Marcialis, F. Roli, S. Schuckers, Livdet 2011 - fingerprint liveness detection competition 2011, in *Proceedings of the IAPR International Conference on Biometrics (ICB, 2011)* (2012), pp. 208–215
19. L. Ghiani, D. Yambay, V. Mura, S. Tocco, G.L. Marcialis, F. Roli, S. Schuckcrs, Livdet 2013 fingerprint liveness detection competition 2013, in *Proceedings of the International Conference on Biometrics (ICB, 2013)* (2013), pp. 1–6
20. A. Kumar, Y. Zhou, Human identification using finger images. IEEE Trans. Image Process. **21**(4), 2228–2244 (2012)
21. K. Seifried, Biometrics-what you need to know. Secur. Portal **10** (2001)
22. D. Baldisserra, A. Franco, D. Maio, D. Maltoni, Fake fingerprint detection by odor analysis, in *Proceedings of the International Conference on Biometrics (ICB, 2006)* (Springer, Berlin, 2006), pp. 265–272
23. P.V. Reddy, A. Kumar, S.M.K. Rahman, T.S. Mundra, A new antispoofing approach for biometric devices. IEEE Trans. Biomed. Circuits Syst. **2**(4), 328–337 (2008)
24. A. Antonelli, R. Cappelli, D. Maio, D. Maltoni, Fake finger detection by skin distortion analysis. IEEE Trans. Inf. Forensics Secur. **1**(3), 360–373 (2006)
25. R. Derakhshani, S.A.C. Schuckers, L.A. Hornak, L. O'Gorman, Determination of vitality from a non-invasive biomedical measurement for use in fingerprint scanners. Pattern Recogn. **36**(2), 383–396 (2003)
26. B. Tan, S. Schuckers, Liveness detection for fingerprint scanners based on the statistics of wavelet signal processing, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW, 2006)* (IEEE, 2006), p. 26
27. S.B. Nikam, S. Agarwal, Texture and wavelet-based spoof fingerprint detection for fingerprint biometric systems, in *Proceedings of the International Conference on Emerging Trends in Engineering and Technology (ICETET, 2008)* (IEEE, 2008), pp. 675–680
28. L. Ghiani, A. Hadid, G.L. Marcialis, F. Roli, Fingerprint liveness detection using binarized statistical image features, in *Proceedings of the International Conference on Biometrics Theory, Applications and Systems (BTAS, 2013)* (IEEE, 2013), pp. 1–6
29. D. Gragnaniello, G. Poggi, C. Sansone, L. Verdoliva, Local contrast phase descriptor for fingerprint liveness detection. Pattern Recogn. **48**(4), 1050–1058 (2015)
30. D. Gragnaniello, G. Poggi, C. Sansone, L. Verdoliva, An investigation of local descriptors for biometric spoofing detection. IEEE Trans. Inf. Forensics Secur. **10**(4), 849–863 (2015)
31. I. Kokkinos, M. Bronstein, A. Yuille, Dense Scale Invariant Descriptors for Images and Surfaces. Research report RR-7914, INRIA (2012)
32. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2014), arXiv.org/abs/1409.1556
33. R. Olga, D. Jia, S. Hao, K. Jonathan, S. Sanjeev, M. Sean, H. Zhiheng, K. Andrej, K. Aditya, B. Michael, C.B. Alexander, F. Li, Imagenet large scale visual recognition challenge. Int. J. Comput. Vision **115**(3), 211–252 (2015)
34. E. Marasco, P. Wild, B. Cukic, Robust and interoperable fingerprint spoof detection via convolutional neural networks (hst 2016), in *Proceedings of the IEEE Symposium on Technologies for Homeland Security* (2016), pp. 1–6

35. C. Wang, K. Li, Z. Wu, Q. Zhao, *A DCNN Based Fingerprint Liveness Detection Algorithm with Voting Strategy* (Springer International Publishing, Cham, 2015). pp. 241–249
36. G.E. Hinton, Training products of experts by minimizing contrastive divergence. Neural Comput. **14**(8), 1771–1800 (2002)
37. J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: the all convolutional net (2014), arXiv.org/abs/1412.6806
38. S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in *Proceedings of the International Conference on Machine Learning (ICML, 2015)* (2015), p. 37
39. N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
40. J. Wang, J. Zucker, Solving multiple-instance problem: a lazy learning approach (2000), pp. 1119–1126
41. R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: a matlab-like environment for machine learning, in *BigLearn, NIPS Workshop* (2011)
42. Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in *Neural Networks: Tricks of the Trade* (Springer, Berlin, 2012), pp. 437–478
43. P.D. Kovesi, MATLAB and Octave functions for computer vision and image processing (2005), http://www.peterkovesi.com/matlabfns
44. L. van der Maaten, G. Hinton, Visualizing data using t-sne. J. Mach. Learn. Res. **9**, 2579–2605 (2008)

Index

B. Bhanu and A. Kumar (eds.), *Deep Learning for Biometrics*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-61657-5