Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Deep learning compact binary codes for fingerprint indexing*

Chao-chao BAI[1], Wei-qiang WANG[1], Tong ZHAO[†‡2,3], Ru-xin WANG[2], Ming-qiang LI[2]

[1]*School of Computer and Control, University of Chinese Academy of Sciences, Beijing 101408, China*
[2]*School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China*
[3]*Key Laboratory of Big Data Mining and Knowledge Management, Chinese Academy of Sciences, Beijing 100080, China*
[†]E-mail: zhaotong@ucas.ac.cn

Received June 26, 2017; Revision accepted Sept. 30, 2017; Crosschecked June 7, 2018; Published online Sept. 12, 2018

Abstract: With the rapid growth in fingerprint databases, it has become necessary to develop excellent fingerprint indexing to achieve efficiency and accuracy. Fingerprint indexing has been widely studied with real-valued features, but few studies focus on binary feature representation, which is more suitable to identify fingerprints efficiently in large-scale fingerprint databases. In this study, we propose a deep compact binary minutia cylinder code (DCBMCC) as an effective and discriminative feature representation for fingerprint indexing. Specifically, the minutia cylinder code (MCC), as the state-of-the-art fingerprint representation, is analyzed and its shortcomings are revealed. Accordingly, we propose a novel fingerprint indexing method based on deep neural networks to learn DCBMCC. Our novel network restricts the penultimate layer to directly output binary codes. Moreover, we incorporate independence, balance, quantization-loss-minimum, and similarity-preservation properties in this learning process. Eventually, a multi-index hashing (MIH) based fingerprint indexing scheme further speeds up the exact search in the Hamming space by building multiple hash tables on binary code substrings. Furthermore, numerous experiments on public databases show that the proposed approach is an outstanding fingerprint indexing method since it has an extremely small error rate with a very low penetration rate.

Key words: Fingerprint indexing; Minutia cylinder code; Deep neural network; Multi-index hashing
https://doi.org/10.1631/FITEE.1700420 **CLC number:** TP311

1 Introduction

Fingerprints, the most famous biometric traits, are widely used for personal identification (Maltoni et al., 2009). For personal identification, a query fingerprint is used to identify the matched template fingerprint in a fingerprint database using a $1:N$ matching process. With the explosive growth of large-scale fingerprint databases and the frequent access demands, the development of an automatic fingerprint identification system (AFIS) has become more challenging than ever. Specifically, identifying a query fingerprint within such a tremendous fingerprint database often incurs significant consumption of running time and memory. Moreover, because of the extreme similarity of fingerprints within such a gigantic fingerprint database, a reliable and precise conclusion might not be obtained.

To solve these problems, fingerprint indexing is the most common solution (Maltoni et al., 2009). First, fingerprint indexing is employed to promptly select a series of candidate similar templates. This narrows the large-scale database effectively. Then a precise but slow fingerprint matching approach is used to return the truly matched one. Recently,

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 11331012, 11571014, and 11731013)
ORCID: Tong ZHAO, http://orcid.org/0000-0002-7587-3573

extensive fingerprint indexing approaches have been proposed and they can be grouped mainly into two categories: level-1 and level-2 fingerprint indexing algorithms. Level-1 features refer to the ridge orientation map (Jiang et al., 2006; Liu et al., 2007; Wang et al., 2007; Liu and Yap, 2012) and the ridge frequency map (Lee et al., 2005). Generally, level-1 features can produce a numerical vector with a similarity-preserving transformation and similar fingerprints are projected as near points in the multidimensional space. Level-2 approaches extract local geometric invariants among the level-2 features (i.e., minutiae). Examples of invariant features include minutia triplets (Germain et al., 1997; Bhanu and Tan, 2003; Liang et al., 2006, 2007), minutia quadruplets (Iloanusi et al., 2011; Iloanusi, 2014), K-plet structures (Bai et al., 2015), and minutia cylinder code (MCC) (Cappelli et al., 2011). However, most of these earlier methods focus on the real-valued feature representations, which are computationally expensive, time consuming, and memory consuming.

Compared with real-valued features, a binary representation, which reduces the loss of memory, is fast in terms of computation and is robust to local variations. As a remedy, MCC (the state-of-the-art binary fingerprint feature) is elaborately designed for fingerprint identification (Cappelli et al., 2010) and indexing (Cappelli et al., 2011). However, MCC also has room for further improvement since MCC is a high-dimensional and redundant code. Additionally, quantization loss can be introduced by quantizing the binary bit directly through the empirical threshold. Therefore, the opportunity exists to propose a set of more effective and discriminative binary indexing features and to design a fast and exact fingerprint indexing scheme.

In this study, we first adopt the MCC structure and further improve it. Accordingly, we propose a novel deep neural network and an alternating iterative learning algorithm to learn the binary fingerprint indexing features. Specifically, the real-valued MCC is the input of the network and we constrain the penultimate layer (layer $n-1$) to directly output the binary codes, which are converted to the final deep compact binary minutia cylinder code (DCBMCC). The objective function of this network is carefully designed to include the criteria for producing good binary codes, which have small quantization error, similarity-preservation, independence, and balance properties. The deep neural network with nonlinear activation functions can capture the complex structure from the original inputs. Finally, we propose an alternating iterative method and a discrete cyclic coordinate technique to solve the objective function.

Thus, we can achieve the binary fingerprint descriptor, and the most convincing reason for binary codes is that they can be directly used as the addresses (indices) of hash tables, which results in a sharp increase in the search speed. However, it is not necessarily efficient to adopt binary codes as direct hash indices. To find close neighbors, it is necessary to inspect all the hash table buckets within some Hamming balls around the query. This will cause a severe problem wherein the number of buckets, which is often larger than the number of items in the database, increases near-exponentially with the search radius.

To solve this problem, we design a fast and exact fingerprint indexing scheme based on the multi-index hashing (MIH) algorithm (Norouzi et al., 2012, 2014) in the high-dimensional Hamming space. With this approach, each binary code in the database is indexed m times into m different hash tables, corresponding to its m disjoint binary substrings. Given a query, we obtain exact k nearest neighbors through the MIH algorithm and select the maximum votes of each template as the scores. Finally, the candidate similar fingerprints are easily produced by sorting the scores in descending order. Note that there also exist several approximate nearest neighbor (ANN) algorithms, while we focus only on an exact search since fingerprint indexing requires extremely high accuracy for critical security applications.

In summary, Fig. 1 shows the flowchart of our approach and the contributions of this paper are summarized as follows:

1. According to the characteristics of MCC, we propose a creative and novel feature learning method based on deep learning techniques and learn to obtain an effective and discriminative DCBMCC for fingerprint indexing. It combines independence, balance, and similarity-preservation properties.

2. We design a fast and exact fingerprint indexing scheme based on the MIH algorithm in the Hamming space. This approach is quite suitable for fingerprint indexing, since it has the large-scale database and extremely high accuracy, which are required in critical security applications.

3. Extensive experiments on public databases show that the proposed approach is an outstanding fingerprint indexing method, since it has a small error rate with a low penetration rate.

2 Minutia cylinder code representation

For the MCC representation, the minutia is represented as a structure that can encode the spatial and directional local information of its neighboring minutiae within the fixed radius. Specifically, this local structure can be represented as a cylinder. Its height and base denote the directional and spatial information, respectively.

In the minutia template T (ISO/IEC 19794-2), let m be the minutia, which is represented as $m = (x_m, y_m, \theta_m)$, where x_m and y_m are the minutia position, and θ_m is the direction of the minutia. To construct the MCC representation, minutia m needs to be represented as a cylinder, which contains a set of sections. Each section means a certain direction within the range $[-\pi, \pi]$; sections continue to be divided into a number of cells. Each cell can be uniquely identified by three indices (i, j, k). Let $d\varphi_k$ be the angle associated to all cells at height k in the cylinder. Let $p_{i,j}^m$ be the two-dimensional point corresponding to the center of the cells with indices (i, j) projected onto the base of cylinder m.

For cell (i, j, k), the value $C_m(i, j, k)$ is computed by summing the contributions of minutia m_t within the neighborhood region $N_{p_{i,j}^m}$ of $p_{i,j}^m$. The function $C_m(i, j, k)$ is defined as follows:

$$C_m(i,j,k) = \begin{cases} \Psi\Big(\sum_{m_t \in N_{p_{i,j}^m}} G(m_t, p_{i,j}^m, d\varphi_k)\Big), & \\ & \text{if } p_{i,j}^m \text{ is valid}, \\ \text{invalid}, & \text{otherwise}, \end{cases} \tag{1}$$

where $G(m_t, p_{i,j}^m, d\varphi_k) = C_m^{\mathrm{S}}(m_t, p_{i,j}^m) C_m^{\mathrm{D}}(m_t, d\varphi_k)$, $C_m^{\mathrm{S}}(m_t, p_{i,j}^m)$ is a Gaussian function to sum the spatial contributions, and $C_m^{\mathrm{D}}(m_t, d\varphi_k)$ is the Gaussian function to accumulate the directional contributions. Thus, we obtain the real-valued MCC representation. Moreover, the binary MCC implementation is adopted by replacing the sigmoid function $\Psi(v)$ with a unit step function:

$$\Psi(v) = \begin{cases} 1, & \text{if } v \geq \mu, \\ 0, & \text{otherwise}. \end{cases} \tag{2}$$

Refer to Cappelli et al. (2010) for more details about the MCC representation.

The main advantage of MCC is that the constructed cylinder is independent of rotation and translation, robust against image distortion (which is small at a local level), robust against the subtle errors of feature extraction, with a fixed length code. However, because MCC is a high-dimensional redundant code, it is not a sufficiently compact and discriminative binary representation. Specifically, there are fewer ones than zeros in each bit of the MCC representation. For example, we calculate the statistics of all MCC representations from the FVC2000DB1 dataset. The average bit value is approximately 0.12; i.e., approximately 88% of the MCC bits are zeros on average. The intra-bit variance of MCC is very small; at the same time, the correlation between neighboring bits of MCC is very large. Additionally, quantization loss can be introduced by quantizing the binary bit directly through Eq. (2) with the empirical threshold.

3 Learning deep compact binary minutia cylinder codes

To obtain an effective DCBMCC, we require a balanced probability of one or zero in each bit and an independence from each other for the different bits. Meanwhile, the loss must be minimized and the similarity must be maintained after quantization. Therefore, we propose a deep neural network as the hash functions and design the objective function of this network to ensure the above properties.

3.1 Formulation

Our main idea is to define the hash functions as an unsupervised deep neural network model and the parameters of this network are what we need to learn. A deep neural network with nonlinear activation functions can capture the complex structure from the inputs. We input the real-valued MCC into the network with n layers (including the input and output layers) and constrain the penultimate layer (layer $n-1$) to produce the binary codes, which can make a good reconstruction of the inputs. Fig. 2 shows the proposed network in a simple case.

First, let us introduce some basic notations. There are m real-valued D-dimensional MCC samples $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ ($\boldsymbol{x}_i \in \mathbb{R}^{D\times 1}$), which

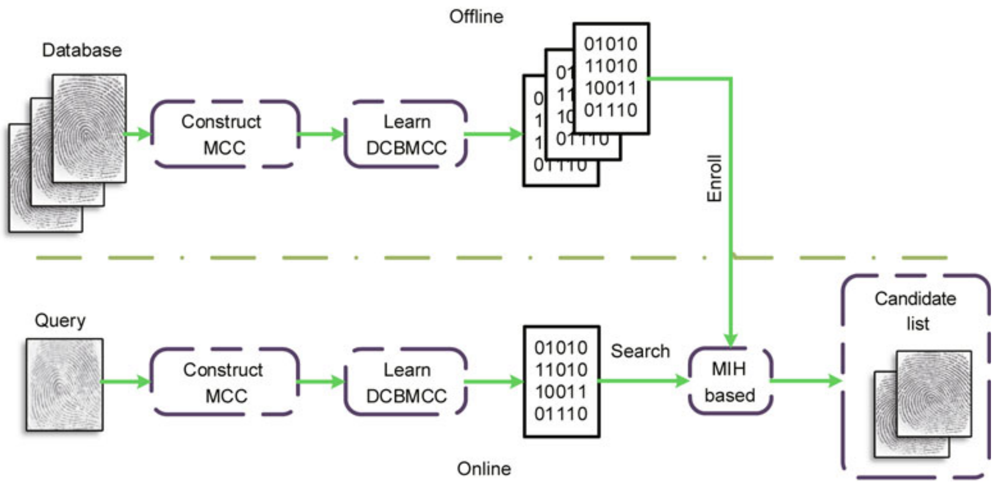

Fig. 1 Flowchart of the proposed deep compact binary minutia cylinder code based on the multi-index hashing (DCBMCC-MIH) approach. First, we extract and construct the real-valued MCC. Then we learn to obtain the binary DCBMCC. Eventually, an MIH-based fingerprint indexing scheme further speeds up the exact search in the Hamming space

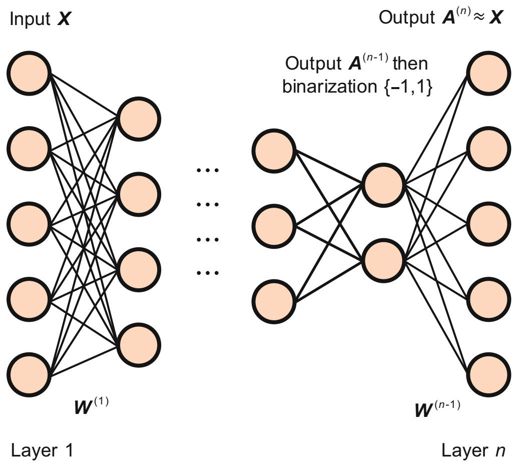
Input X
Output $A^{(n)} \approx X$
Output $A^{(n-1)}$ then binarization {−1,1}
$W^{(1)}$
$W^{(n-1)}$
Layer 1
Layer n

Fig. 2 A simple case for the proposed network ($D = 5$ and $L = 2$). We input a D-dimensional real-valued minutia cylinder code into the network and constrain layer $n-1$ to produce an L-dimensional binary code, which can make a reconstruction of the input

constitute the columns of the data matrix $\boldsymbol{X} \in \mathbb{R}^{D\times m}$. Without loss of generality, we require that the samples be zero-centered; i.e., $\sum_{i=1}^{m} \boldsymbol{x}_i = \boldsymbol{0}$. Let $\boldsymbol{B} = \{\boldsymbol{b}_i\}_{i=1}^{m} \in \mathbb{R}^{L\times m}$ be the binary matrix of $\boldsymbol{X}$, where L is the number of bits to encode a sample (the number of neurons in layer $n-1$ of the network). In the network, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{s_{l+1}\times s_l}$ is the weight matrix connecting layer $l+1$ and layer l; s_l denotes the number of units in layer l; $\boldsymbol{c}^{(l)} \in \mathbb{R}^{s_{l+1}}$ is the bias vector in layer $l+1$; $\boldsymbol{A}^{(l)} \in \mathbb{R}^{s_l\times m}$ represents the output activation value of layer l and $\boldsymbol{A}^{(l)} = f^{(l)}(\boldsymbol{W}^{(l-1)}\boldsymbol{A}^{(l-1)} + \boldsymbol{c}^{(l-1)}\boldsymbol{1}_{1\times m})$, where $f^{(l)}(\cdot)$ is the activation function of layer l and $\boldsymbol{1}_{a\times b}$ is the $a \times b$ matrix whose elements are always 1. The activation functions of all the layers are not the same. Specifically, we employ sigmoid functions (3) at layers $\{2, 3, \ldots, n-2\}$ and identity functions (4) at layers $\{n-1, n\}$:

$$f^{(l)}(\boldsymbol{x}) = \frac{1}{1+\mathrm{e}^{-\boldsymbol{x}}}, \quad \forall\, l = 2, 3, \ldots, n-2, \tag{3}$$

$$f^{(l)}(\boldsymbol{x}) = \boldsymbol{x}, \quad \forall\, l = n-1, n. \tag{4}$$

Inspired by the auto-encoder and network (Do et al., 2016), we first design the objective function of this network to include the criteria of reconstruction and parameter regularization as follows:

$$\min_{\boldsymbol{W},\boldsymbol{c}} J = \frac{1}{2m}\Big\|\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\mathrm{sgn}(\boldsymbol{A}^{(n-1)}) - \boldsymbol{c}^{(n-1)}\boldsymbol{1}_{1\times m}\Big\|_{\mathrm{F}}^{2} + \frac{\lambda_1}{2}\sum_{l=1}^{n-1}\big\|\boldsymbol{W}^{(l)}\big\|_{\mathrm{F}}^{2}, \tag{5}$$

where the sgn(·) function makes $\boldsymbol{A}^{(n-1)}$ be $\{-1, 1\}$. Note that we make the samples and binary codes zero-centered and first set $\boldsymbol{B}$ as $\{-1, 1\}$ rather than $\{0, 1\}$. Finally, we transform $\boldsymbol{B}$ to be $\{0, 1\}$ as the learned DCBMCC. In model (5), the first item can make a good reconstruction of the inputs $\boldsymbol{X}$ with the binary codes $\mathrm{sgn}(\boldsymbol{A}^{(n-1)})$. The second term is the regularization of the deep neural network model, which minimizes the network parameter values and avoids over-fitting.

To simply solve model (5), we introduce the auxiliary variable $\boldsymbol{B}$. Then we can decompose model (5) into two sub-models and alternatively iterate to optimize variables $(\boldsymbol{W}, \boldsymbol{c})$ and $\boldsymbol{B}$. As a result, model (5) can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{c},\boldsymbol{B}} \ J = & \frac{1}{2m}\left\|\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\boldsymbol{B} - \boldsymbol{c}^{(n-1)}\mathbf{1}_{1\times m}\right\|_{\mathrm{F}}^2 \\ & + \frac{\lambda_1}{2}\sum_{l=1}^{n-1}\left\|\boldsymbol{W}^{(l)}\right\|_{\mathrm{F}}^2 \qquad (6) \\ \text{s.t.} \quad & \boldsymbol{B} = \operatorname{sgn}(\boldsymbol{A}^{(n-1)}) \in \{-1,1\}^{L\times m}. \end{aligned}$$

In addition, a good binary code should have these important properties: balance, independence, and small quantization error. That is to say, we need to learn DCBMCC, whose intra-bit variance is maximized and inter-bit correlation is minimized. Equivalently, each bit has an even probability of being -1 or 1 and different bits are pairwise independent of each other. Moreover, it is necessary to minimize the quantization loss between quantized binary codes and real-valued features. Therefore, we add the above properties to model (6) and then the new model can be written as

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{c},\boldsymbol{B}} \ J = & \frac{1}{2m}\left\|\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\boldsymbol{B} - \boldsymbol{c}^{(n-1)}\mathbf{1}_{1\times m}\right\|_{\mathrm{F}}^2 \\ & + \frac{\lambda_1}{2}\sum_{l=1}^{n-1}\left\|\boldsymbol{W}^{(l)}\right\|_{\mathrm{F}}^2 + \frac{\lambda_2}{2m}\left\|\boldsymbol{A}^{(n-1)}\mathbf{1}_{m\times 1}\right\|_2^2 \\ & + \frac{\lambda_3}{2}\left\|\frac{1}{m}\boldsymbol{A}^{(n-1)}(\boldsymbol{A}^{(n-1)})^{\mathrm{T}} - \boldsymbol{I}\right\|_{\mathrm{F}}^2 \qquad (7) \\ & + \frac{\lambda_4}{2m}\left\|\boldsymbol{A}^{(n-1)} - \boldsymbol{B}\right\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{B} \in \{-1,1\}^{L\times m}. \end{aligned}$$

Last but not the least, we require binary codes maintain similarity after the quantization; i.e., it encourages similar high-dimensional real-valued MCC samples to learn similar binary codes while dissimilar MCC samples to learn dissimilar binary codes. Here, we ensure the preservation of similarity with the Laplacian matrix.

In advance, we calculate the weight matrix $\boldsymbol{R} \in \mathbb{R}^{m\times m}$ within the input samples $\boldsymbol{X}$, and its element r_{ij} denotes the distance between samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$:

$$r_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{2\sigma^2}\right). \qquad (8)$$

To obtain the Laplacian matrix, we then define a diagonal matrix $\boldsymbol{D}$. The element value of $\boldsymbol{D}$ is the row (or column) of matrix $\boldsymbol{R}$, because $\boldsymbol{R}$ is a symmetric matrix; i.e., $d_{jj} = \sum_i r_{ij}$. Finally, the Laplacian matrix can be computed as $\boldsymbol{P} = \boldsymbol{D} - \boldsymbol{R}$, which stores our prior knowledge about the relationship (similarity) among high-dimensional MCC representations. Thus, the optimization model (7) can be written as

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{c},\boldsymbol{B}} \ J = & \frac{1}{2m}\left\|\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\boldsymbol{B} - \boldsymbol{c}^{(n-1)}\mathbf{1}_{1\times m}\right\|_{\mathrm{F}}^2 \\ & + \frac{\lambda_1}{2}\sum_{l=1}^{n-1}\left\|\boldsymbol{W}^{(l)}\right\|_{\mathrm{F}}^2 + \frac{\lambda_2}{2m}\left\|\boldsymbol{A}^{(n-1)}\mathbf{1}_{m\times 1}\right\|_2^2 \\ & + \frac{\lambda_3}{2}\left\|\frac{1}{m}\boldsymbol{A}^{(n-1)}(\boldsymbol{A}^{(n-1)})^{\mathrm{T}} - \boldsymbol{I}\right\|_{\mathrm{F}}^2 \\ & + \frac{\lambda_4}{2m}\left\|\boldsymbol{A}^{(n-1)} - \boldsymbol{B}\right\|_{\mathrm{F}}^2 \qquad (9) \\ & + \frac{\lambda_5}{2m}\operatorname{tr}(\boldsymbol{A}^{(n-1)}\boldsymbol{P}(\boldsymbol{A}^{(n-1)})^{\mathrm{T}}) \\ \text{s.t.} \quad & \boldsymbol{B} \in \{-1,1\}^{L\times m}. \end{aligned}$$

Finally, we formulate our final optimization model (9) gradually, and it includes all the important criteria for producing good binary codes. The first item of model (9) is the reconstruction term; i.e., to guarantee that the original input $\boldsymbol{X}$ can be reconstructed well with the compact binary outputs of layer $n-1$. The second item is the regularization of the deep neural network model, which minimizes the network parameter values and avoids over-fitting. The third item is the balance term to make the variance of each bit maximal so that each bit has an even probability of being -1 or 1. The fourth item is the independence term, to make sure that the binary bits are pairwise independent of each other. The fifth item is the quantization error, to minimize the quantization loss between binary codes $\boldsymbol{B}$ and real-valued features $\boldsymbol{A}^{(n-1)}$. The sixth item is the similarity-preserving term to maintain the similarity of data points; i.e., it encourages similar high-dimensional MCC samples to learn similar binary codes while dissimilar MCC samples to learn dissimilar binary codes.

3.2 Optimization

In optimization model (9), we divide the variables that need to be solved into two groups: binary codes $\boldsymbol{B}$ and network parameters $(\boldsymbol{W}, \boldsymbol{c})$. Then we propose an algorithm to optimize $(\boldsymbol{W}, \boldsymbol{c})$ and $\boldsymbol{B}$ alternately.

1. Fix $\boldsymbol{B}$ and update $(\boldsymbol{W}, \boldsymbol{c})$

When $\boldsymbol{B}$ is fixed, model (9) can be converted to an unconstrained optimization problem. We apply the back-propagation L-BFGS algorithm to optimize the problem. The gradients of objective function J for different variables are as follows:

For layer $n-1$, we define

$$\frac{\partial J}{\partial \boldsymbol{W}^{(n-1)}} = -\frac{1}{m}\left(\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\boldsymbol{B} - \boldsymbol{c}^{(n-1)}\boldsymbol{1}_{1\times m}\right)\boldsymbol{B}^{\mathrm{T}} + \lambda_1 \boldsymbol{W}^{(n-1)}, \tag{10}$$

$$\frac{\partial J}{\partial \boldsymbol{c}^{(n-1)}} = -\frac{1}{m}\left((\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\boldsymbol{B})\boldsymbol{1}_{m\times 1} - m\boldsymbol{c}^{(n-1)}\right). \tag{11}$$

For other layers, we define

$$\begin{aligned}\boldsymbol{\Upsilon}^{(n-1)} = \Bigg[&\frac{\lambda_2}{m}(\boldsymbol{A}^{(n-1)}\boldsymbol{1}_{m\times m}) \\ &+ \frac{2\lambda_3}{m}\left(\frac{1}{m}\boldsymbol{A}^{(n-1)}(\boldsymbol{A}^{(n-1)})^{\mathrm{T}} - \boldsymbol{I}\right)\boldsymbol{A}^{(n-1)} \\ &+ \frac{\lambda_4}{m}(\boldsymbol{A}^{(n-1)} - \boldsymbol{B}) + \frac{\lambda_5}{m}\boldsymbol{A}^{(n-1)}\boldsymbol{P}\Bigg] \\ &\odot f^{(n-1)'}(\boldsymbol{Z}^{(n-1)}),\end{aligned} \tag{12}$$

$$\boldsymbol{\Upsilon}^{(l)} = ((\boldsymbol{W}^{(l)})^{\mathrm{T}}\boldsymbol{\Upsilon}^{(l+1)}) \odot f^{(l)'}(\boldsymbol{Z}^{(l)}), \quad \forall\, l = n-2, n-3, \ldots, 2, \tag{13}$$

where $\boldsymbol{Z}^{(l)} = \boldsymbol{W}^{(l-1)}\boldsymbol{A}^{(l-1)} + \boldsymbol{c}^{(l-1)}\boldsymbol{1}_{(1\times m)}$ $(l = 2, 3, \ldots, n)$ and '$\odot$' denotes the Hadamard product.

Then for layers $l = n-2, n-3, \ldots, 1$, we have

$$\frac{\partial J}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\Upsilon}^{(l+1)}(\boldsymbol{A}^{(l)})^{\mathrm{T}} + \lambda_1\boldsymbol{W}^{(l)}, \tag{14}$$

$$\frac{\partial J}{\partial \boldsymbol{c}^{(l)}} = \boldsymbol{\Upsilon}^{(l+1)}\boldsymbol{1}_{m\times 1}. \tag{15}$$

After obtaining the gradients of objective function J, we update parameters $\boldsymbol{W}$ and $\boldsymbol{c}$ according to

$$\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \alpha\frac{\partial J}{\partial \boldsymbol{W}^{(l)}}, \tag{16}$$

$$\boldsymbol{c}^{(l)} = \boldsymbol{c}^{(l)} - \alpha\frac{\partial J}{\partial \boldsymbol{c}^{(l)}}, \tag{17}$$

where α is the learning rate, $l = 1, 2, \ldots, n-1$.

2. Fix $(\boldsymbol{W}, \boldsymbol{c})$ and update $\boldsymbol{B}$

When $(\boldsymbol{W}, \boldsymbol{c})$ is fixed, optimization model (9) can be rewritten as

$$\begin{aligned}\min_{\boldsymbol{B}}\ J = &\left\|\boldsymbol{X} - \boldsymbol{W}^{(n-1)}\boldsymbol{B} - \boldsymbol{c}^{(n-1)}\boldsymbol{1}_{1\times m}\right\|_{\mathrm{F}}^2 \\ &+ \lambda_4\left\|\boldsymbol{A}^{(n-1)} - \boldsymbol{B}\right\|_{\mathrm{F}}^2 \\ \text{s.t.}\quad &\boldsymbol{B} \in \{-1, 1\}^{L\times m}.\end{aligned} \tag{18}$$

It is very difficult to directly solve $\boldsymbol{B}$, since there is the binary constraint on $\boldsymbol{B}$ and we use the discrete cyclic coordinate descent algorithm (Shen et al., 2015) to solve it. Specifically, we solve $\boldsymbol{B}$ in a row-by-row iteration; i.e., we fix the $L-1$ rows in $\boldsymbol{B}$ and solve only the remaining one row, thus obtaining the closed-form solution for that row.

In advance, we define

$$\boldsymbol{O} = \boldsymbol{X} - \boldsymbol{c}^{(n-1)}\boldsymbol{1}_{1\times m}, \tag{19}$$

$$\boldsymbol{G} = (\boldsymbol{W}^{(n-1)})^{\mathrm{T}}\boldsymbol{O} + \lambda_2\boldsymbol{A}^{(n-1)}. \tag{20}$$

$\forall\, k$ $(k = 1, 2, \ldots, L)$, let $\boldsymbol{g}_k$ be the kth column in matrix $\boldsymbol{G}^{\mathrm{T}}$. Let $\boldsymbol{w}_k$ be the kth column in matrix $\boldsymbol{W}^{(n-1)}$. Matrix $\boldsymbol{W}_r$ means that $\boldsymbol{W}^{(n-1)}$ removes the kth column. Let $\boldsymbol{b}_k^{\mathrm{T}}$ be the kth row in matrix $\boldsymbol{B}$. Matrix $\boldsymbol{B}_r$ means that $\boldsymbol{B}$ removes the kth row. Thus, we can calculate $\boldsymbol{b}_k^{\mathrm{T}}$ by

$$\boldsymbol{b}_k^{\mathrm{T}} = \mathrm{sgn}\left(\boldsymbol{g}_k^{\mathrm{T}} - \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{W}_r\boldsymbol{B}_r\right). \tag{21}$$

To sum up, Algorithm 1 summarizes the proposed DCBMCC learning algorithm.

Algorithm 1 Learning the DCBMCC algorithm

```
Input: real-valued MCC samples $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{m} \in \mathbb{R}^{D\times m}$, the length of binary codes $L$, and the number of iterations $T$.
Output: network parameters $\{\boldsymbol{W}^{(l)}, \boldsymbol{c}^{(l)}\}_{l=1}^{n-1}$.
Initialize: $\boldsymbol{B}_{(0)} \in \{-1, 1\}^{L\times m}$ with the ITQ algorithm (Gong et al., 2013) and Laplacian matrix $\boldsymbol{P}$.
1: Fix $\boldsymbol{B}_{(0)}$ and calculate $(\boldsymbol{W}, \boldsymbol{c})_{(0)}$ with the initialized network parameters
2: for $t = 1, 2, \ldots, T$ do
3:     Fix $(\boldsymbol{W}, \boldsymbol{c})_{(t-1)}$ and update $\boldsymbol{B}_{(t)}$
4:     Fix $\boldsymbol{B}_{(t)}$ and update $(\boldsymbol{W}, \boldsymbol{c})_{(t)}$
5: end for
6: Return $(\boldsymbol{W}, \boldsymbol{c})_{(T)}$
```

4 Fingerprint indexing scheme

In this section, we describe our design for a fast and exact fingerprint indexing scheme based on the MIH algorithm in the Hamming space for DCBMCC. First, we introduce MIH for exact k nearest neighbors (KNN). Then we describe two steps of fingerprint indexing, which are enrollment of template fingerprints and searching for a query fingerprint.

4.1 Multi-index hashing for k nearest neighbors

So far, we have obtained DCBMCC and it can be rapidly compared linearly. However, the compelling reason for compact binary codes is the direct address of a hash table, resulting in a speed increase compared to a linear scan. Nevertheless, for n binary codes of q ($q > 32$) bits, the number $L(q, r)$ of hash buckets within the search radius r to be examined increases with the search radius r near-exponentially, where $L(q,r) = \sum_{z=0}^{r} C_q^z$. That is, the vast number of 2^q possible buckets in the full table will be empty, since $2^q \gg n$. Consequently, examining $L(q, r)$ buckets is more expensive than a linear scan, since most of them have no item. To solve this problem, we adopt MIH for an exact k nearest neighbors search among the binary codes.

Specifically, each q-bit binary code $\boldsymbol{h}$ is divided into m q/m-bit disjoint substrings, denoted as $\{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(m)}\}$. When two binary codes $\boldsymbol{h}$ and $\boldsymbol{g}$ differ by r bits or less, in at least one of their m substrings, they must differ by at most $r' = \lfloor q/m \rfloor$ bits. As a consequence, it suffices to examine $L(q/m, r')$ buckets in each of the m substring hash tables (i.e., a total of $m \times L(q/m, r')$ buckets), rather than $L(q, r)$ hash buckets. In this way, we can obtain a set of candidate neighbors $N_j(\boldsymbol{g})$ from the hash table corresponding to the j^{th} substring. Then the union of the m sets, $N(\boldsymbol{g}) = \cup_j N_j(\boldsymbol{g})$, is necessarily a superset of the r neighbors of $\boldsymbol{g}$. The final step is checking the full Hamming distance between each candidate neighbor in $N(\boldsymbol{g})$ and $\boldsymbol{g}$, and then retaining codes that are real r neighbors of $\boldsymbol{g}$.

Additionally, the multi-index hashing algorithm is easily adapted to accommodate the query-dependent search radius. Given a query, we gradually increase the Hamming search radius of each substring so that we can obtain a certain number of k neighbors. Refer to Norouzi et al. (2012, 2014) for more details about the MIH algorithm.

4.2 Enrollment of template fingerprints

An offline enrollment is to fill all features of the template fingerprints into the hash tables before indexing the query fingerprint. First, m duplicate hash tables are constructed for initialization. Then these minutiae of template fingerprints are transformed as DCBMCC vectors by Algorithm 1. According to the MIH method, they continue to be split as the m substrings, which have one-to-one correspondence with the m hash tables. In each hash table, the pair (i, j), which means template T_i but also vector $\boldsymbol{v}_j$ (corresponding to minutia m_j of template T_i), is registered into the corresponding bucket.

Algorithm 2 gives the summary of the offline enrollment process.

Algorithm 2 Enrollment algorithm

```
Input: database of minutiae templates DB = $\{T_1, T_2, \ldots, T_n\}$.
Output: hash tables HT = $\{H_1, H_2, \ldots, H_m\}$.
Initialize: each hash table $H_k$, $k = 1, 2, \ldots, m$.
1: for all minutiae templates $T_i \in$ DB do
2:    Use Algorithm 1 to create DCBMCC vector set $\boldsymbol{V}_i$ from $T_i$
3:    for all vectors $\boldsymbol{v}_j \in \boldsymbol{V}_i$ do
4:       Split $\boldsymbol{v}_j$ as substring set $\mathbf{sv}_j = \{\boldsymbol{v}_j^1, \boldsymbol{v}_j^2, \ldots, \boldsymbol{v}_j^m\}$
5:       for all substring vectors $\boldsymbol{v}_j^k \in \mathbf{sv}_j$ do
6:          Enroll $(i, j)$ in the corresponding bucket of hash table $H_k$
7:       end for
8:    end for
9: end for
```

4.3 Searching for a query fingerprint

Given query T_1 and template T_2, let $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ be the corresponding sets of binary vectors. Moreover, a simple but effective similarity measure between T_1 and T_2 can be defined as follows:

$$\text{hds}(T_1, T_2) = \frac{\text{score}(\boldsymbol{V}_1, \boldsymbol{V}_2)}{|\boldsymbol{V}_1|}, \tag{22}$$

where score$(\boldsymbol{V}_1, \boldsymbol{V}_2)$ means the similarity scores of two sets of binary vectors and hds$(\cdot)$ is normalized between zero and one by dividing $|\boldsymbol{V}_1|$ (the cardinality of set $\boldsymbol{V}_1$). Here, score$(\boldsymbol{V}_1, \boldsymbol{V}_2)$ is defined as

$$\text{score}(\boldsymbol{V}_1, \boldsymbol{V}_2) = \sum_{\boldsymbol{v} \in \boldsymbol{v}_1} \max_{\boldsymbol{v}_j \in \boldsymbol{V}_2} \text{sim}(\boldsymbol{v}, \boldsymbol{v}_j), \tag{23}$$

where sim$(\boldsymbol{v}, \boldsymbol{v}_j)$ denotes the similarity between two DCBMCC vectors $\boldsymbol{v}$ and $\boldsymbol{v}_j$. It is calculated as

$$\text{sim}(\boldsymbol{v}, \boldsymbol{v}_j) = 1 - \frac{\text{dis}(\boldsymbol{v}, \boldsymbol{v}_j)}{q}, \tag{24}$$

where q is the number of code bits and dis$(\boldsymbol{v}, \boldsymbol{v}_j)$ is the Hamming distance between $\boldsymbol{v}$ and $\boldsymbol{v}_j$, which can be returned by multi-index hashing for the KNN algorithm.

Accordingly, searching for a query fingerprint can obtain similar templates by looking up m established hash tables and calculating the above similarities. At query time, for each of the given DCBMCC vectors, we search and obtain k nearest neighbors through the MIH algorithm. Then these k nearest neighbors are obtained with the number of distance votes. For each query vector, we select the maximum votes of each template as the scores. In this way, the most similar database templates are efficiently determined and the candidate list is easily produced.

Algorithm 3 shows the summary of the online searching approach.

Algorithm 3 Searching algorithm

```
Input: query template $Q$ and hash tables HT = $\{H_1, H_2, \ldots, H_m\}$.
Output: list of candidates CL = $\{(i, \text{score})\}$.
Initialize: score accumulator $\mathbb{S}$.
 1: Use Algorithm 1 to create DCBMCC vector set $\boldsymbol{V}$ from $Q$
 2: for all vectors $\boldsymbol{v} \in \boldsymbol{V}$ do
 3:     Let $m$ be the minutia of $T$ corresponding to $\boldsymbol{v}$
 4:     Reset collision accumulator $\mathbb{A}$
 5:     Use MIH to search and obtain KNN of $\boldsymbol{v}$
 6:     Let $\boldsymbol{v}_j^i$ be one of the KNN, representing the $j^{\text{th}}$ minutia of template $T_i$
 7:     $\mathbb{A}[i,j] = \text{sim}(\boldsymbol{v}, \boldsymbol{v}_j^i)$  // Corresponding to Eq. (24)
 8:     for all $T_i$ with at least one collision in $\mathbb{A}$ do
 9:         Let $\text{CF}_{\max} = \max_j\{\mathbb{A}[i,j]\}$
10:         $\mathbb{S}[i] = \mathbb{S}[i] + \text{CF}_{\max}$  // Corresponding to Eq. (23)
11:     end for
12: end for
13: for all template index $i \in \mathbb{S}$ do
14:     $\mathbb{S}[i] = \mathbb{S}[i]/|\boldsymbol{V}|$  // Corresponding to Eq. (22)
15: end for
16: Create CL in descending order of $(i, \mathbb{S}[i])$
```

5 Experiments

In this section, we present experiments to evaluate our approach and its comparisons with other existing methods. First, we introduce the preparations for the experiments, including dataset information, implementation notes, and experiment protocols. Then extensive experiments on fingerprint indexing are performed on numerous public datasets.

5.1 Preparations for experiments

1. Dataset information

Experimental evaluation of DCBMCC and the comparisons were conducted on the FVC2000, FVC2002, FVC2004, and NIST DB4, DB14 public datasets. Table 1 lists the detailed information of the datasets, including the size, resolution, number of impressions, number of subjects, and sensor. Similar to Cappelli et al. (2011), we chose the last 2700 fingerprint pairs from NIST DB14 to conduct our experiments.

2. Implementation note

In all experiments, an open-source software package, the NIST biometric image software (NBIS) v5.0.0 (http://www.nist.gov/itl/iad/ig/nbis.cfm), was employed to extract minutiae from the fingerprint images. Then we employed the MCC SDK v2.0 (http://biolab.csr.unibo.it/mccsdk.html) with the same parameters reported in Cappelli et al. (2011) to create the real-valued MCC. We employed the open-source software (http://github.com/norouzi/mih/) to implement the MIH based indexing scheme.

We used the first 60 000 real-valued MCC (384 dimensions) in NIST DB14 as the training sets and the datasets in Table 1 are the test sets. The neural network adopts a representative stacked auto-encoder structure (Fig. 2) and it consists of five layers including the input layer and the output layer. In addition, we employed sigmoid functions at layers 2 and 3, and identity functions at layers 4 and 5. We implemented 64 and 128 dimensions of DCBMCC, and the corresponding network configurations were [384-312-256-128-384] and [384-256-128-64-384], respectively. Empirically, we set parameters λ_1, λ_2, λ_3, λ_4, and λ_5 as 10^{-4}, 10^{-2}, 10^{-2}, 10^{-5}, and 10^{-4}, respectively. Moreover, we set the maximum iteration number to be 20.

3. Experiment protocol

Typically, we adopted the first image for enrollment and the remaining seven for searching on all the FVC datasets. While the first image was used for index creation, the second one was used for searching on all the NIST datasets.

For indicators, the trade-off between the penetration rate and error rate (ER) was employed to show the efficiency and accuracy of a fingerprint indexing system. Here, the penetration rate denotes the proportion of the database that the system has to search and ER = $(N_{\text{er}}/N_{\text{d}}) \times 100\%$, where N_{er} is the number of unfound query fingerprints and N_{d} is the total number of input query fingerprints.

Table 1 Detailed information of the FVC2000, FVC2002, FVC2004, and NIST DB4, DB14 public datasets

Database	Set	Size	Resolution (dpi)	Number of impressions	Number of subjects	Sensor
FVC2000	DB2a	256×364	500	8	100	Capacitive
	DB3a	448×478	500	8	100	Optical
FVC2002	DB1a	388×374	500	8	100	Optical
	DB4a	288×384	500	8	100	Synthetic
FVC2004	DB2a	328×364	500	8	100	Optical
	DB3a	300×480	512	8	100	Sweeping
NIST	DB4	512×512	500	2	2000	Ink-rolled
	DB14	832×768	500	2	27 000	Ink-rolled

Moreover, we considered the single performance indicator of the average penetration rate. It is a retrieval scenario in which an ideal comparison algorithm is adopted to stop the search, once the correct candidate is retrieved.

5.2 Fingerprint indexing experiments

In this subsection, we verify the performance of DCBMCC-MIH fingerprint indexing and make the comparisons. The MCC-based fingerprint indexing algorithm (MCC-LSH) (Cappelli et al., 2011) has already outperformed most previous fingerprint indexing methods on several public datasets. To show our performance adequately and simply, MCC-LSH was chosen as the baseline method. The MCC-LSH results were produced with the MCC SDK v2.0 software with the same parameters as those reported in Cappelli et al. (2011). For more comparisons, we implemented the MCC-MIH combination method, where an MIH indexing approach was substituted for LSH. The PMCC-MIH approach was changed with 128-bit PMCC (Ferrara et al., 2012) as the indexing feature representation.

To better demonstrate the outstanding performance of the proposed method, Figs. 3–6 exhibit the fingerprint indexing performance and all comparisons with the indicators for the penetration rate and error rate on FVC2000, FVC2002, FVC2004, NIST DB4, and NIST DB14. From all of these figures, the fingerprint indexing performance of DCBMCC-MIH(128) is always the best, achieving a small error rate with a low penetration rate. Moreover, the performance of MCC-LSH(384) is the worst although it has the longest code representation, because its small intra-bit variance, large inter-bit correlation, and non-optimal binary quantization make MCC insufficiently discriminative. In addition, the LSH method in Cappelli et al. (2011) essentially searches the approximate nearest neighbors, resulting in the loss of precision to some extent.

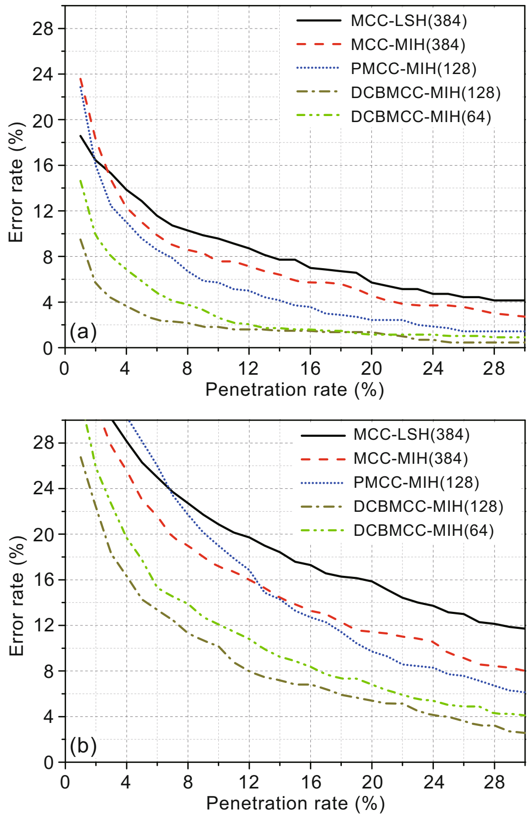

Fig. 3 Fingerprint indexing performance on the FVC2000 DB2 (a) and DB3 (b)

In essence, the results of fingerprint indexing are affected mainly by two factors: indexing feature and indexing scheme. MCC-LSH and MCC-MIH are contrasted in these figures and they employ the same MCC as the indexing feature; thus, the conclusion is that the indexing performance of MIH is better than that of LSH. It can be seen that MIH is more appropriate for fingerprint indexing, which requires ex-

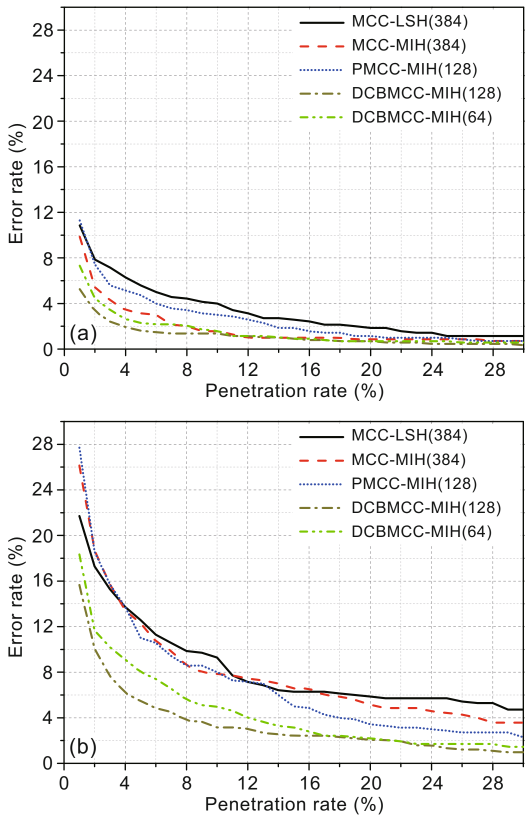

Fig. 4 Fingerprint indexing performance on the FVC2002 DB1 (a) and DB4 (b)

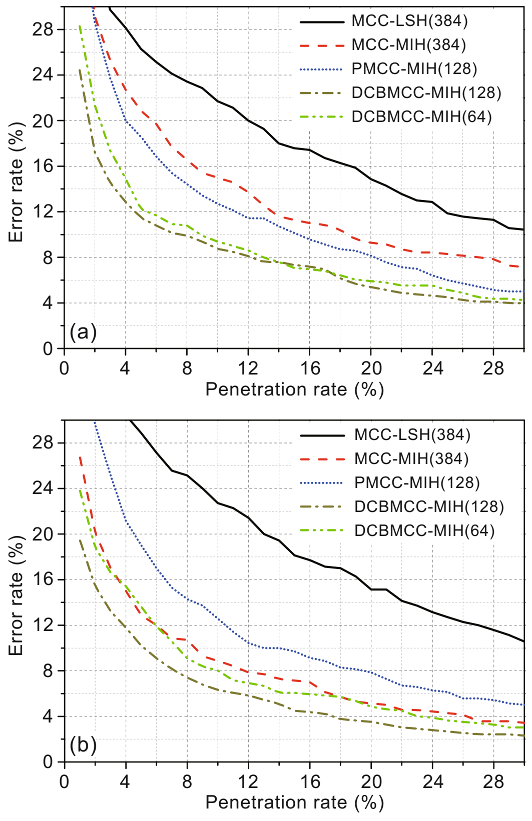

Fig. 5 Fingerprint indexing performance on the FVC2004 DB2 (a) and DB3 (b)

tremely high precision. By comparing the different features, we can draw some conclusions. The 128-bit DCBMCC is obviously more discriminative and effective than PMCC(128), since DCBMCC has outstanding indicators of intra-bit variance and inter-bit correlation, especially the minimum quantization loss and similarity preservation. In addition, the performance of DCBMCC-MIH(64) is similar to, or even better than, that of MCC-MIH(384), and what deserves our consideration is that DCBMCC(64) has one-sixth of length, consuming less computational time and storage than MCC(384). In particular, it can be seen that DCBMCC with more bits is more effective and discriminative than that with fewer bits. However, when the limitations in computational time and memory capacity are required, DCBMCC(64) or a shorter DCBMCC can be the solution with a tolerable precision loss.

Moreover, we ran the retrieval-scenario experiments with the average penetration rate and compared it with some learning-to-hash (L2H) and our previous CBMCC methods (Bai et al., 2016). While most existing L2H approaches were designed for visual search, they may also be suitable for fingerprint recognition, even if no such study has been done before. In this part, some widely used binary-code learning methods were employed, such as spectral hashing (SH) (Weiss et al., 2009), iterative quantization (ITQ) (Gong and Lazebnik, 2011; Gong et al., 2013), and spherical hashing (SPH) (Heo et al., 2012, 2015), to learn binary fingerprint-feature representations. All of these methods were implemented with open-source codes. We used these methods to learn binary codes by replacing DCBMCC, and all other steps remained the same for a fair comparison. Table 2 reports the average penetration rates of the different methods. Clearly, our DCBMCC outperforms the existing binary-code learning methods in the fingerprint indexing task. Furthermore, we can see that the performance of our DCBMCC is better than that of CBMCC, since the DCBMCC approach takes more comprehensive properties into consideration, especially in the case of similarity preservation and the deep neural network with nonlinear activation functions which can capture the complex structure in the inputs well.

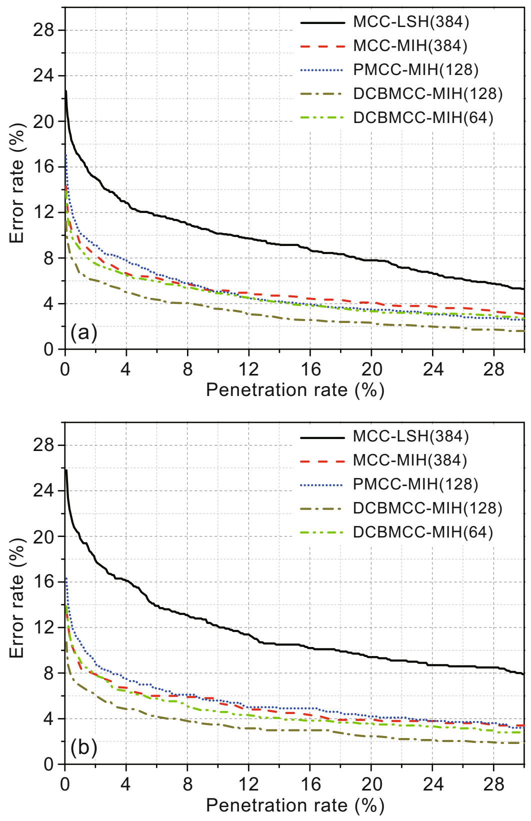

Fig. 6 Fingerprint indexing performance on the NIST DB4 (a) and DB14 (b)

Table 2 Comparison with the existing L2H and CBMCC methods

Dataset	Average penetration rate (%)				
	SH	ITQ	SPH	CBMCC	Ours
FVC2000 DB2	4.10	3.05	2.89	2.28	**1.63**
FVC2000 DB3	7.74	5.04	5.11	4.89	**4.07**
FVC2002 DB1	2.25	2.11	2.16	1.58	**1.49**
FVC2002 DB4	4.07	3.82	3.71	2.64	**2.32**
FVC2004 DB2	7.39	5.79	6.07	5.14	**4.58**
FVC2004 DB3	6.29	4.53	4.39	3.80	**3.59**
NIST DB4	3.95	2.66	2.62	2.15	**1.95**
NIST DB14	4.07	2.84	2.90	2.17	**1.97**

Bold numbers denote the best results

To show the time factor, we ran the time test on the whole set of 27 000-pair fingerprint images from the NIST DB14, which is the largest public dataset on hand. We tested the DCBMCC-MIH execution time with respect to different dimensions and their comparisons with a linear scan (DCBMCC-LinScan) on the NIST DB14. Table 3 lists the search time for a query fingerprint search on the NIST DB14. It is evident that the DCBMCC-MIH algorithm is very efficient and much faster than the linear scan in 64-bit cases. The speedup to 128-bit is not obvious since the size of NIST DB14 is still far more less with respect to the 128-bit search space. If the dataset is in its true sense large-scale, the 128-bit speedup will be much more obvious (Norouzi et al., 2014). The time refers to a Matlab and C++ implementation with no particular optimization, running on an Intel Core CPU @ 2.40 GHz. Note that the execution time will be further reduced with a GPU and proper optimization.

Table 3 Time test on the NIST DB14

Dimension	DCBMCC-MIH (s)	DCBMCC-LinScan (s)
128-bit	5.89	7.25
64-bit	0.76	5.32

6 Conclusions

We have detailedly analyzed the MCC representation, an excellent binary representation of fingerprint features. It has room for further improvement since it is high-dimensional, strongly redundant, less discriminative, and it has a lossy binary quantization. To solve these issues, we have proposed a deep neural network model and a learning algorithm to learn binary fingerprint indexing features (DCBMCC). The network constrains the penultimate layer to directly output the binary codes and incorporates small quantization error, similarity preservation, independence, and balance properties in the learning process. Then the MIH-based fingerprint indexing method further speeds up the exact search in the Hamming space. Finally, numerous experiments illustrate that the proposed approach has excellent performance in fingerprint indexing.

References

Bai C, Zhao T, Wang W, et al., 2015. An efficient indexing scheme based on K-plet representation for fingerprint database. Int Conf on Intelligent Computing, p.247-257. https://doi.org/10.1007/978-3-319-22180-9-25

Bai C, Wang W, Zhao T, et al., 2016. Learning compact binary quantization of minutia cylinder code. Int Conf on Biometrics, p.1-6. https://doi.org/10.1109/ICB.2016.7550054

Bhanu B, Tan X, 2003. Fingerprint indexing based on novel features of minutiae triplets. *IEEE Trans Patt Anal Mach Intell*, 25(5):616-622. https://doi.org/10.1109/TPAMI.2003.1195995

Cappelli R, Ferrara M, Maltoni D, 2010. Minutia cylinder-code: a new representation and matching technique for fingerprint recognition. *IEEE Trans Patt Anal Mach Intell*, 32(12):2128-2141. https://doi.org/10.1109/TPAMI.2010.52

Cappelli R, Ferrara M, Maltoni D, 2011. Fingerprint indexing based on minutia cylinder-code. *IEEE Trans Patt Anal Mach Intell*, 33(5):1051-1057. https://doi.org/10.1109/TPAMI.2010.228

Do T, Doan A, Cheung N, 2016. Learning to hash with binary deep neural network. European Conf on Computer Vision, p.219-234. https://doi.org/10.1007/978-3-319-46454-1-14

Ferrara M, Maltoni D, Cappelli R, 2012. Noninvertible minutia cylinder-code representation. *IEEE Trans Inform Forens Secur*, 7(6):1727-1737. https://doi.org/10.1109/TIFS.2012.2215326

Germain R, Califano A, Colville S, 1997. Fingerprint matching using transformation parameter clustering. *IEEE Comput Sci Eng*, 4(4):42-49. https://doi.org/10.1109/99.641608

Gong Y, Lazebnik S, 2011. Iterative quantization: a procrustean approach to learning binary codes. IEEE Int Conf on Computer Vision and Pattern Recognition, p.817-824. https://doi.org/10.1109/CVPR.2011.5995432

Gong Y, Lazebnik S, Gordo A, et al., 2013. Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans Patt Anal Mach Intell*, 35:2916-2929. https://doi.org/10.1109/TPAMI.2012.193

Heo J, Lee Y, He J, et al., 2012. Spherical hashing. IEEE Int Conf on Computer Vision and Pattern Recognition, p.2957-2964. https://doi.org/10.1109/CVPR.2012.6248024

Heo J, Lee Y, He J, et al., 2015. Spherical hashing: binary code embedding with hyperspheres. *IEEE Trans Patt Anal Mach Intell*, 37(11):2304-2316. https://doi.org/10.1109/TPAMI.2015.2408363

Iloanusi O, 2014. Fusion of finger types for fingerprint indexing using minutiae quadruplets. *Patt Recogn Lett*, 38:8-14. https://doi.org/10.1016/j.patrec.2013.10.019

Iloanusi O, Gyaourova A, Ross A, 2011. Indexing fingerprints using minutiae quadruplets. IEEE Int Conf on Computer Vision and Pattern Recognition Workshops, p.127-133. https://doi.org/10.1109/CVPRW.2011.5981825

Jiang X, Liu M, Kot A, 2006. Fingerprint retrieval for identification. *IEEE Trans Inform Forens Secur*, 1(4):532-542. https://doi.org/10.1109/TIFS.2006.885021

Lee S, Kim Y, Park G, 2005. A feature map consisting of orientation and inter-ridge spacing for fingerprint retrieval. Int Conf on Audio- and Video-Based Biometric Person Authentication, p.184-190. https://doi.org/10.1007/11527923-19

Liang X, Asano T, Bishnu A, 2006. Distorted fingerprint indexing using minutia detail and Delaunay triangle. IEEE Int Symp on Voronoi Diagrams in Science and Engineering, p.217-223. https://doi.org/10.1109/ISVD.2006.42

Liang X, Bishnu A, Asano T, 2007. A robust fingerprint indexing scheme using minutia neighborhood structure and low-order Delaunay triangles. *IEEE Trans Inform Forens Secur*, 2(4):721-733. https://doi.org/10.1109/TIFS.2007.910242

Liu M, Yap P, 2012. Invariant representation of orientation fields for fingerprint indexing. *Patt Recogn*, 45(7):2532-2542. https://doi.org/10.1016/j.patcog.2012.01.014

Liu M, Jiang X, Kot A, 2007. Efficient fingerprint search based on database clustering. *Patt Recogn*, 40(6):1793-1803. https://doi.org/10.1016/j.patcog.2006.11.007

Maltoni D, Maio D, Jain A, et al., 2009. Handbook of Fingerprint Recognition. Springer-Verlag, London, UK. https://doi.org/10.1007/978-1-84882-254-2

Norouzi M, Punjani A, Fleet D, 2012. Fast search in hamming space with multi-index hashing. IEEE Int Conf on Computer Vision and Pattern Recognition, p.3108-3115. https://doi.org/10.1109/CVPR.2012.6248043

Norouzi M, Punjani A, Fleet D, 2014. Fast exact search in hamming space with multi-index hashing. *IEEE Trans Patt Anal Mach Intell*, 36(6):1107-1119. https://doi.org/10.1109/TPAMI.2013.231

Shen F, Shen C, Liu W, et al., 2015. Supervised discrete hashing. IEEE Int Conf on Computer Vision and Pattern Recognition, p.37-45. https://doi.org/10.1109/CVPR.2015.7298598

Wang Y, Hu J, Phillips D, 2007. A fingerprint orientation model based on 2D Fourier expansion (FOMFE) and its application to singular-point detection and fingerprint indexing. *IEEE Trans Patt Anal Mach Intell*, 29(4):573-585. https://doi.org/10.1109/TPAMI.2007.1003

Weiss Y, Torralba A, Fergus R, 2009. Spectral hashing. Advances in Neural Information Processing Systems, p.1753-1760.