

Chapter 2

Real-Time Face Identification via Multi-convolutional Neural Network and Boosted Hashing Forest

Yury Vizilter, Vladimir Gorbatshevich, Andrey Vorotnikov and Nikita Kostromov

Abstract The family of real-time face representations is obtained via Convolutional Network with Hashing Forest (CNHF). We learn the CNN, then transform CNN to the multiple convolution architecture and finally learn the output hashing transform via new Boosted Hashing Forest (BHF) technique. This BHF generalizes the Boosted Similarity Sensitive Coding (SSC) approach for hashing learning with joint optimization of face verification and identification. CNHF is trained on CASIA-WebFace dataset and evaluated on LFW dataset. We code the output of single CNN with 97% on LFW. For Hamming embedding we get CBHF-200 bit (25 byte) code with 96.3% and 2,000-bit code with 98.14% on LFW. CNHF with $2,000 \times 7$ -bit hashing trees achieves 93% rank-1 on LFW relative to basic CNN 89.9% rank-1. CNHF generates templates at the rate of 40+ fps with CPU Core i7 and 120+ fps with GPU GeForce GTX 650.

2.1 Introduction

Various face recognition applications presume different priorities of template size, template generation speed, template matching speed, and recognition rates. We know that the fastest search in a database is provided by binary templates with Hamming distance [1, 7–10, 12, 14, 18, 20, 21, 30, 34]. On the other hand, the best face recognition rates are achieved by deep convolutional neural networks (CNN) with non-binary face representations [3, 5, 23–25, 27, 29, 31, 35]. These approaches can

Y. Vizilter (✉) · V. Gorbatshevich · A. Vorotnikov · N. Kostromov
State Research Institute of Aviation Systems (GosNIIAS), Moscow, Russia
e-mail: viz@gosniias.ru

V. Gorbatshevich
e-mail: gvs@gosniias.ru

A. Vorotnikov
e-mail: vorotnikov@gosniias.ru

N. Kostromov
e-mail: nikita-kostromov@yandex.ru

be fused in the special CNN architecture with binary output layer, which we refer as Convolutional Network with Hashing Layer (CNHL). The most promising CNHL is described in [6], where CNN and hashing layer are learned together via back propagation technique. But now we need the family of face representations, which continuously varies from small Hamming codes to coded features with larger size, better metrics and higher recognition rates. So, in this chapter we propose to combine the CNN and additional hashing transform based on Hashing Forest (HF). Our HF forms the vector of features coded by binary trees. HF with different depth of trees and different coding objectives allows obtaining the family of face representations based on the same CNN. We refer such CNN+HF architecture as Convolutional Network with Hashing Forrest (CNHF). In case of 1-bit coding trees CNHF degrades to CNHL and provides the Hamming embedding.

The architecture of our CNHF is based on the Max-Feature-Map (MFM) CNN architecture proposed by Xiang Wu [31]. For real-time implementation we accelerate our CNN via transforming to the multiple convolution architecture.

We propose the new Boosted Hashing Forest (BHF) technique, which generalizes the Boosted Similarity Sensitive Coding (Boosted SSC) [20, 21] for discriminative data coding by forest hashing with direct optimization of objective function and given properties of coded feature space. We also introduce and implement the new biometric-specific objective function for joint optimization of face verification and identification.

Proposed CNHF face representations are trained on CASIA-WebFace dataset and evaluated on LFW dataset. Our experiments demonstrate both compact binary face representations and increasing of face verification and identification rates. In the Hamming embedding task BHF essentially outperforms the original Boosted SSC. Our CNHF 200 bit (25 byte) hash achieves 96.3% on LFW with 70-time gain in a matching speed. CNHF 2,000 bit hash provides 98.14% on LFW. CNHF with $2,000 \times 7$ -bit hashing trees achieves 93% rank-1 on LFW relative to basic CNN 89.9% rank-1.

The remainder of this chapter is organized as follows. Section 2.2 briefly describes the related work. Section 2.3 describes the architecture and learning of our CNHF with multiple convolutional layers. Section 2.4 contains the outline of proposed BHF technique and its implementation for face hashing. Experimental results are presented in Sect. 2.5. Conclusion and discussion are presented in Sect. 2.6.

2.2 Related Work

A lot of face representation techniques were proposed [4, 16, 26], but all state-of-the-art results are obtained now via deep CNN. One can learn CNN for multi-class face identification with classes corresponding to persons [27, 35], or learn the similarity metric by training two identical CNNs (Siamese Architecture [5, 29], or combine these approaches [23, 25, 32]). Best modern results on LFW are obtained by ensembles of deep nets learned on different parts (patches) of face [13, 23, 25].

Nevertheless, some single nets can be efficient enough with essentially lower computational cost [3, 31]. Most frequently the CNN-based face representation is formed as an output of top hidden layer [5, 23, 27, 29, 31, 35]. Sometimes the PCA is applied for size reduction [23, 24]. The L2-distance [4, 29] or cosine similarity [23, 27, 31] are of use for matching of face representations.

Binary hashing means the assigning of binary code to each input feature vector. The review of classical hashing techniques is presented in [9]. The simplest binary hashing idea is to use some dimensionality reduction transform and then apply some quantization technique. The optimization-based hashing approach presumes the similarity-driven data embedding into the Hamming space. In [7] the similarity search is proposed based on linear binary coders and vectors of weights obtained by random rotations. The Iterative Quantization (ITQ) technique [8] considers the hashing problem as a search of rotation, which minimizes the quantization error. Kernel-Based Supervised Hashing (KSH) [14] utilizes a kernel formulation for the target hash functions. The affinity-preserving algorithm [10] performs k-means clustering and learns the binary indices of the quantized cells. The manifold hashing techniques follow the ideas of manifold learning. The Spectral Hashing [30] relaxes the hashing problem in the manner of Laplacian Eigenmaps [1]. Topology Preserving Hashing (TPH) [34] performs the Hamming embedding with additional preserving the neighbor ranks. Locally Linear Hashing (LLH) [12] presumes both preserving distances and reconstructing the locally linear structures. The Semantic Hashing (SH) [18] solves the hashing problem with the use of Restricted Boltzmann Machines (RBM). Boosted Similarity Sensitive Coding (Boosted SSC) proposed by Shaknarovich, Voila and Darrell [20, 21] performs the sequential bit-by-bit growing of the hash code with reweighting of samples in the manner of AdaBoost and forming the weighted Hamming space.

The idea of binary face coding based on deep learning is well implemented in [6]. The CNN and hashing layer are learned together via back propagation technique, and 32-bit binary face representation is generated with 91% verification on LFW. Unfortunately, the direct optimization of more complex face coding criterions is not available in this one-step CNHL learning framework. In particular, it cannot provide the immediate optimization of Cumulative Matching Curve (CMC). Due to this we implement the two-step CNHF learning procedure: learning basic CNN first and hashing transform second.

Our hashing transform is based on hashing forest. Look at some previous forest hashing techniques. Qiu, Sapiro, and Bronstein [17] propose the random forest semantic hashing scheme with information-theoretic code aggregation for large-scale data retrieval. The feature induction based on random forest for learning regression and multi-label classification is proposed by Vens and Costa [28]. Yu and Yuan [33] implement a forest hashing with special order-sensitive Hamming distance. The forest hashing by Springer et al. [22] combines *kd*-trees with hashing technique. The Boosted Random Forest algorithm proposed by Mishina, Tsuchiya, and Fujiyoshi [15] is out of the binary hashing topic. Our approach performs the feature space coding via boosted forest hashing in the manner of Boosted SSC with optimizing

of task-specific objective function. So, we mainly consider our BHF technique as a generalization of Boosted SSC.

2.3 CNHF with Multiple Convolution CNN

Our CNHF contains the basic deep CNN and additional hashing transform based on Hashing Forrest (HF). This hashing forest forms the output CNHF binary face representation, which semantically corresponds to some objective vector of features coded by these binary trees (Fig. 2.1). For obtaining the family of optimized face representations based on the same CNN we use the two-step CNHF learning procedure. At the first step the CNN is formed and trained for multi-class face identification. At the second step the hashing transform is trained for combined face verification and identification. We start from learning the source CNN with softmax output layer for face identification. Then we transform its convolutional layers to the multiple convolution form. Finally we cut the output softmax layer and use the activations of top hidden layer as a basic face representation for further hashing. In this chapter, we use the Max-Feature-Map (MFM) CNN architecture proposed by Xiang Wu [31]. It is based on the Max-Feature-Map activation function instead of ReLU. Reference [31] demonstrates that Max-Feature-Map can get the compact and discriminative feature vectors. The source network architecture contains four convolutional layers, four layers of pooling + MFM pooling, one fully connected layer and the softmax

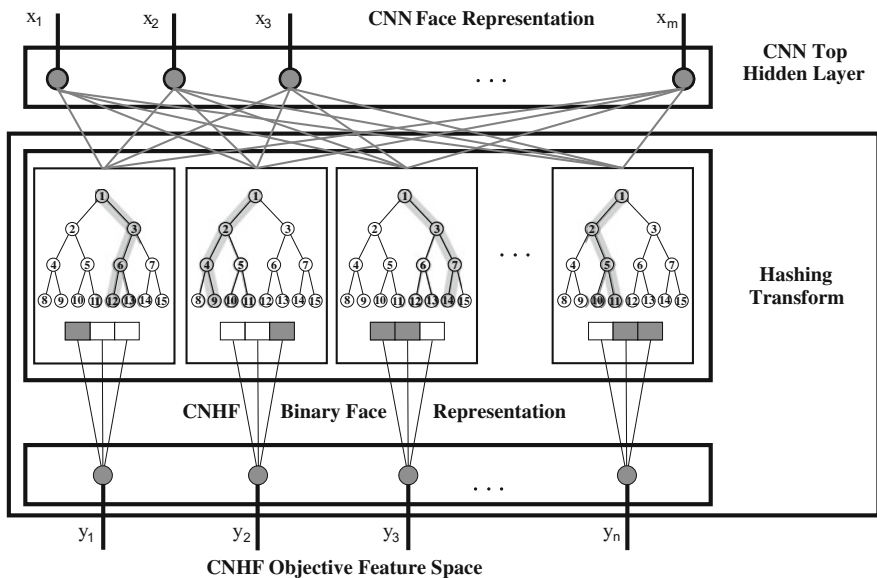


Fig. 2.1 Architecture of CNHF: CNN + hashing transform based on hashing forest

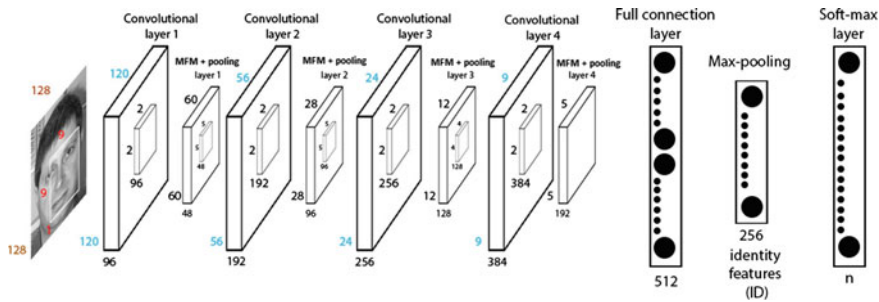


Fig. 2.2 Architecture of source MFM deep net [25]

layer (Fig. 2.2). Following the approach of Xiang Wu [31] we start from learning this source MFM deep net for multi-class face identification with classes corresponding to persons in the manner [24, 31] using the backpropagation technique.

Unfortunately, we cannot directly implement the architecture (Fig. 2.2) for the real-time face identification with CPU. We need to optimize this architecture in order to obtain essentially higher calculation speed. So, we propose and apply the new approach for sequential transformation of deep network topology based on the following tricks:

1. We use the small-sized filters instead of large-sized filters in the convolutional layers. For example, we substitute one layer with 5×5 filters by the sequence of two layers with 3×3 filters, which is 1.38 times faster on CPU.
2. We decrease the number of filters in each layer. For example, the first layer of source net (Fig. 2.2) contains 96 filters, but the first layer of our transformed net contains 20 filters only, which is more than 4 times faster on CPU.
3. The each layer is transformed and relearned separately. For this purpose we need to provide the equal input and output dimensionalities for the source layer and corresponding part of transformed net, which is used for its substitution. We do this by adding the $1 \times 1 \times n$ layers to the transformed net, where n is the number of filters in the substituted source layer. For example, we substitute the one $9 \times 9 \times 96$ layer of source network (Fig. 2.2) by the sequence of two layers $9 \times 9 \times 20$ and $1 \times 1 \times 96$, which is still more than 4 times faster on CPU.

Thus, we simplify the network topology sequentially, layer by layer, without the relearning of the whole CNN. In this process we represent the each convolution as a combination of convolutions, so, we refer the resultant architecture of transformed net as “multiple convolutional” or briefly “multiconv”. At the each step of one layer substitution we use the Euclidean loss for minimization of difference between the output response of this source layer and corresponding part of transformed net, which is used for its substitution, for the same input values. Figure 2.3 illustrates the proposed scheme for topology transformation and relearning. The training set at this stage contains face images without pointing to persons. We use the open source

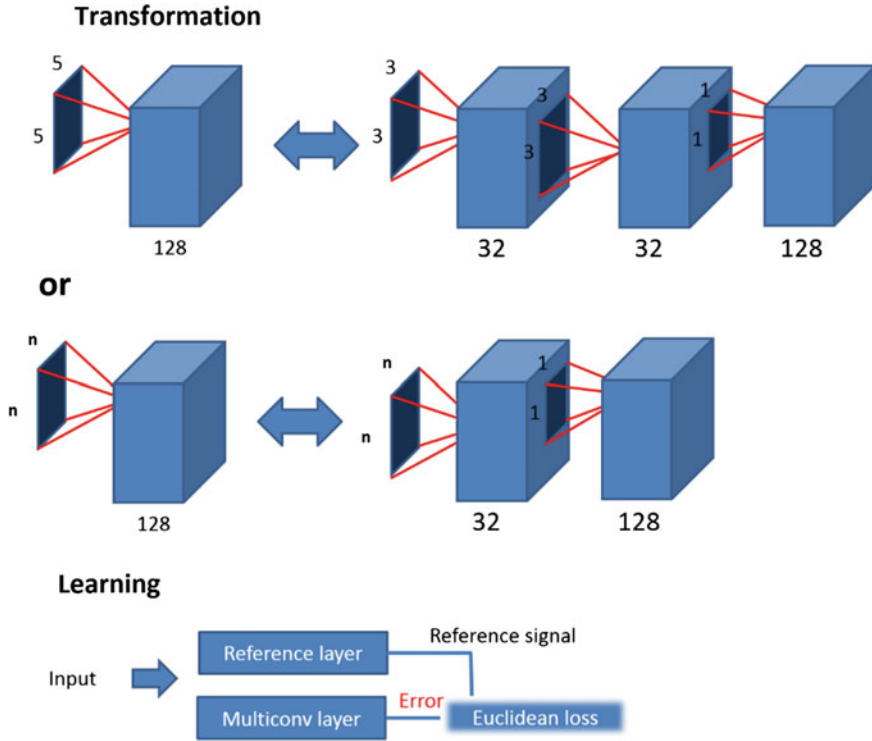


Fig. 2.3 The proposed scheme for network topology transformation and relearning

framework Caffe for learning of transformed layers by standard back propagation technique as well as for the whole network training (see Sect. 2.5.1).

Finally, we could represent the proposed process for deep net architecture sequential transformation as the following informal Algorithm 0.

Algorithm 0: CNN transform to multiconv net

Input data: *conv net*.

Output data: *multiconv net*.

Initialization:

multiconv net := *conv net*.

Repeat iterations:

Step 1. Find the slowest conv layer L in *multiconv net*;

Step 2. Replace L by sequence of layers S with less summarized number of convolutions;

Step 3. Learn S using the Euclidean loss for imitating the output of substituted layer L ;

while *speed grows and accuracy is still high enough*

Table 2.1 Iteration 1

Layer	Convolutional layer 2 (Fig. 2.2)	Convolutional layer 3 (Fig. 2.4)	Convolutional layer 4 (Fig. 2.4)	Convolutional layer 5 (Fig. 2.4)	Convolutional layer 6 (Fig. 2.4)
Filter size	$5 \times 5 \times 48$	$3 \times 3 \times 48$	$1 \times 1 \times 24$	$3 \times 3 \times 32$	$1 \times 1 \times 32$
Number of filters	192	24	32	32	192
Number of operations (mult.)	722,534,400	34,877,952	2,583,552	28,901,376	19,267,584

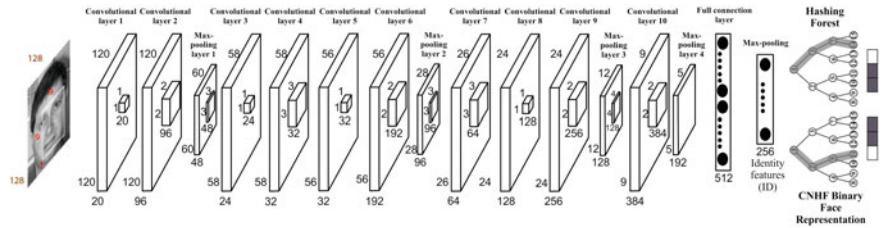


Fig. 2.4 Architecture of CNHF based on MFM net with multiple convolutions

Table 2.2 Iteration 2

Layer	Convolutional layer 3 (Fig. 2.2)	Convolutional layer 7 (Fig. 2.4)	Convolutional layer 8 (Fig. 2.4)	Convolutional layer 9 (Fig. 2.4)
Filter size	$5 \times 5 \times 96$	$3 \times 3 \times 96$	$3 \times 3 \times 64$	$1 \times 1 \times 128$
Number of filters	256	64	128	256
Number of operations (mult.)	353,894,400	37,380,096	42,467,328	18,874,368

Actually, only three iterations of the Algorithm 0 were used.

Iteration 1: Convolutional layer 2 (Fig. 2.2) was replaced by layers 3, 4, 5, 6 (see the Table 2.1):

The original layer (layer 2 in Fig. 2.2) requires more than 700 million multiplications and produces the output with the dimensions of $56 \times 56 \times 192$, the layer sequence (layers 3, 4, 5, 6 in Fig. 2.4) requires about 90 million of multiplications (about 7 times less than the original layer) with the same output size.

Iteration 2: Convolutional layer 3 (Fig. 2.2) was replaced by layers 7, 8, 9 (Table 2.2):

The original layer (layer 3 in Fig. 2.2) requires more than 350 million multiplications and produces the output with the dimensions of $24 \times 24 \times 256$, the layer sequence (layers 7, 8, 9 in Fig. 2.4) requires about 100 million multiplications (about 3 times less than the original layer) with the same output size.

Table 2.3 Iteration 3

Layer	Convolutional layer 1 (Fig. 2.2)	Convolutional layer 1 (Fig. 2.4)	Convolutional layer 2 (Fig. 2.4)
Filter size	$9 \times 9 \times 1$	$9 \times 9 \times 1$	$1 \times 1 \times 20$
Number of filters	96	20	96
Number of operations (mult.)	111,974,400	23,328,000	27,648,000

Iteration 3: Convolutional layer 1 (Fig. 2.2) was replaced by layers 1, 2 (Table 2.3):

The original layer (layer 1 in Fig. 2.2) requires more than 110 million multiplications and produces the output with the dimensions of $120 \times 120 \times 96$, the layer sequence (layers 1, 2 in Fig. 2.4) requires about 50 million multiplications (about 2 times less than the original layer) with the same output size.

Unfortunately, all our attempts to replace layer 4 from the original network led to significant loss in accuracy (greater than 10%), but this layer requires relatively less computations – about 90 million multiplications.

The main advantage of this approach is a relatively high speed of learning at the steps of layer substitutions. We used the GTX 1080 card for this learning and trained the one multiconv substitution approximately in 10–15 min. This allows performing the multiconv transformation of any source CNN architecture in the very convenient and partially automated way.

After all simplifying substitutions, the transformed CNN is trained again for multi-class face identification with classes corresponding to persons in the manner [24, 31] using the backpropagation technique. Finally the output softmax layer of transformed MFM net is replaced by hashing forest, and we obtain the CNHF based on MFM with multiple convolutional layers (Fig. 2.4). In result our CNHF contains 10 convolutional layers, four layers of MFM+pooling, fully connected layer and hashing forest. This CNHF generates face templates at the rate of 40+ fps with CPU Core i7 and 120+ fps with GPU GeForce GTX 650. Thus, we can conclude that the proposed multiconv approach makes our CNHF 5 times faster on CPU than source CNN. It is enough for the real-time operation.

2.4 Learning Face Representation via Boosted Hashing Forest

2.4.1 Boosted SSC, Forest Hashing and Boosted Hashing Forest

We learn our hashing transform via the new Boosted Hashing Forest (BHF) technique, which combines the algorithmic structure of Boosted SSC [20, 21] and the binary code structure of forest hashing [15, 17, 22, 28, 33].

Boosted SSC algorithms optimize the performance of L1 distance in the embedding space as a proxy for the pairwise similarity function, which is conveyed by a set of examples of positive (similar) and negative (dissimilar) pairs. The *SSC algorithm* takes pairs labeled by similarity and produces a binary embedding space. The embedding is learned by independent collecting thresholded projections of the input data. The threshold is selected by optimal splitting the projections of negative pairs and non-splitting the projections of positive pairs. *Boosted SSC algorithm* collects the embedding dimensions greedily with adaptive weighting of samples and dimensions in the manner of AdaBoost. *BoostPro algorithm* uses a soft thresholding for gradient-based learning of projections.

The differences of proposed BHF w.r.t. Boosted SSC are the following:

1. BHF performs the binary coding of output feature space, which is not binary in general, but can be binary Hamming, if required.
2. BHF performs the direct optimization of any given objective function of output features.
3. BHF learns the objective-driven data projections via RANSAC algorithm without gradient-based optimization.
4. BHF performs the recursive coding by binary trees and forms the hashing forest, while Boosted SSC performs the iterative feature coding and forms hashing vector.
5. BHF performs the adaptive reweighting of training pairs based on their contribution to the objective function, unlike the AdaBoost-style reweighting of Boosted SSC.
6. Boosted SSC forms the weighted Hamming space. Our BHF forms the any given metric space, including non-weighted Hamming space for fastest data search.

Algorithm 1: Greedy ORC

Input data: X, \mathcal{J}, n_{ORC} .

Output data: $\mathbf{h}(\mathbf{x}): \mathbf{x} \in R^m \rightarrow \mathbf{y} \in \{0,1\}^{n_{ORC}}, \mathbf{h}(\mathbf{x}) \in H$.

Initialization:

Step 0. $k:=0; \mathbf{h}^{(k)} := ()$.

Repeat iterations:

$k:=k+1;$

Learn k -th elementary coder:

$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) := \text{Learn1BitHash}(\mathcal{J}, X, \mathbf{h}^{(k-1)});$

Add k -th elementary coder to the hashing function:

$\mathbf{h}^{(k)}(\mathbf{x}) := (\mathbf{h}^{(k-1)}(\mathbf{x}), h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}));$

while $k < n_{ORC}$. // stop if the given size of coder is got

The main differences of proposed BHF w.r.t. other forest hashing techniques: we obtain the hashing forest via RANSAC projections and boosting process in the manner of Boosted SSC; we optimize the task-specific objective function in the coded feature space, but not the similarity in the binary code space.

BHF implementation for face recognition has some additional original features: new biometric-specific objective function with joint optimization of face verification and identification; selection and processing of subvectors of the input feature vector;

creation of ensemble of independent hash codes for overcoming the limitations of greedy learning. In the next subsections we describe our BHF algorithms in detail.

2.4.2 BHF: Objective-Driven Recurrent Coding

Let the training set $X = \{\mathbf{x}_i \in R^m\}_{i=1,\dots,N}$ contains N objects described by m -dimensional feature vectors. Map X to the n -dimensional binary space: $X = \{\mathbf{x}_i \in R^m\}_{i=1,\dots,N} \rightarrow B = \{\mathbf{b}_i \in \{0, 1\}^n\}_{i=1,\dots,N}$. This mapping is an n -bit coder:

$$\mathbf{h}(\mathbf{x}) : \mathbf{x} \in R^m \rightarrow \mathbf{b} \in \{0, 1\}^n \quad (2.1)$$

The elementary coder is called the *1-bit hashing function*

$$h(\mathbf{x}) : \mathbf{x} \in R^m \rightarrow b \in \{0, 1\} \quad (2.2)$$

Let some *objective function* (coding criterion) is given and required to be minimized

$$\mathcal{J}(X, \mathbf{h}) \rightarrow \min(\mathbf{h}). \quad (2.3)$$

Algorithm 2: RANSAC Learn1ProjectionHash

Input data: $\mathcal{J}, X, \mathbf{h}^{(k-1)}, k_{RANSAC}$.

Output data: $h(\mathbf{w}, t, \mathbf{x})$.

Initialization:

Step 0. $k:=0; \mathcal{J}_{min}:=+\infty$.

Repeat iterations:

$k:=k+1;$

Step 1. Take the random dissimilar pair $(\mathbf{x}_i, \mathbf{x}_j)$ in X .

Step 2. Get vector $\overrightarrow{(\mathbf{x}_i, \mathbf{x}_j)}$ as a vector of hyperplane direction: $\mathbf{w}_k := \mathbf{x}_j - \mathbf{x}_i$.

Step 3. Calculate the threshold t_k minimizing $\mathcal{J}(6)$ by t with $\mathbf{w}=\mathbf{w}_k$: $t_k := \operatorname{argmin}_t \mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t)$.

Step 4. If $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t_k) < \mathcal{J}_{min}$, then

$\mathcal{J}_{min} := \mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t_k); \mathbf{w} := \mathbf{w}_k; t := t_k$.

while $k < k_{RANSAC}$. // stop if the given number of RANSAC iterations is achieved

Denote $\mathbf{h}^{(k)}(\mathbf{x}) = (h^{(1)}(\mathbf{x}), \dots, h^{(k)}(\mathbf{x}))$. The operation of coders concatenation is $\mathbf{h}^{(k)}(\mathbf{x}) := (\mathbf{h}^{(k-1)}(\mathbf{x}), h^{(k)}(\mathbf{x}))$. The Greedy Objective-driven Recurrent Coding (Greedy ORC) algorithm (Algorithm 1) sequentially forms the bits of our coder in a recurrent manner: $h^{(k)}(\mathbf{x}) = h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)})$. The proper procedure for learning the each k th bit is described in the next subsections.

2.4.3 BHF: Learning Elementary Projection via RANSAC Algorithm

At the k th step of coder growing

$$\mathcal{J}(X, \mathbf{h}^{(k)}) = \mathcal{J}(X, \mathbf{h}^{(k-1)}, h^{(k)}) \rightarrow \min\{h^{(k)} \in \mathbf{H}\}, \quad (2.4)$$

Algorithm 3: Optimal threshold selection

Input data: $\mathcal{J}, X, \mathbf{h}^{(k-1)}, \mathbf{w}, N$.

Output data: t .

Operations:

Step 1. For all $\mathbf{x}_i \in X$ calculate projections $t_i = (\mathbf{x}_i, \mathbf{w})$;

Step 2. Arrange samples \mathbf{x}_i and projections t_i by increasing of t_i . For all arranged indices $i=1..N$ do:

$\Delta \mathcal{J}_i := 0$;

Step 3. For all pairs $(\mathbf{x}_i, \mathbf{x}_j)$ with $t_i < t_j$ do:

// increment the step values at projection points

$\Delta \mathcal{J}_i := \Delta \mathcal{J}_i + \mathcal{J}_{ij}^{(out)}$;

$\Delta \mathcal{J}_i := \Delta \mathcal{J}_i + \mathcal{J}_{ij}^{(in)} - \mathcal{J}_{ij}^{(out)}$;

$\Delta \mathcal{J}_j := \Delta \mathcal{J}_j + \mathcal{J}_{ij}^{(out)} - \mathcal{J}_{ij}^{(in)}$;

Step 4. Recover the stepwise objective function and find the optimal threshold

$\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t_1) := 0$; // recover the first value

$i:=0$; $\mathcal{J}_{min}:=+\infty$.

Repeat iterations:

$i:=i+1$;

$\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t_i) := \mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t_{i-1}) + \Delta \mathcal{J}_i$;

// accumulate step values from left to right

If $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t_i) < \mathcal{J}_{min}$, then

$\mathcal{J}_{min} := \mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t_i)$; $t := (t_i + t_{i+1})/2$;

while $i < N-1$. // stop if all step points are tested

where \mathbf{H} is a class of coders. Consider the class of elementary coders based on thresholded linear projections

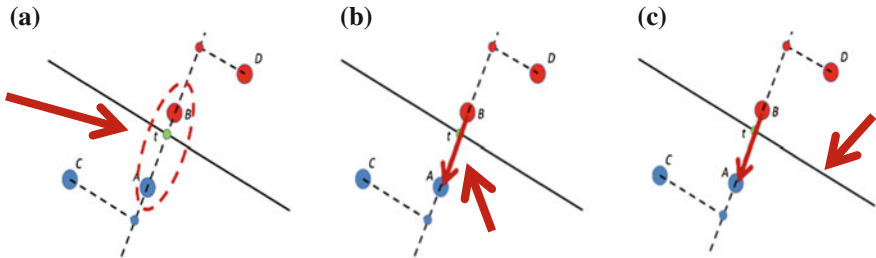


Fig. 2.5 RANSAC Learn1ProjectionHash: **a** Step 1, **b** Steps 2, **c** Steps 3 of Algorithm 2

$$h(\mathbf{w}, t, \mathbf{x}) = \text{sgn}(\sum_{k=1, \dots, m} w_k x_k + t), \quad (2.5)$$

where \mathbf{w} – vector of weights, t – threshold of hashing function, $\text{sgn}(u) = \{1, \text{ if } u > 0; 0 - \text{ otherwise}\}$. In case of (2.5) function (2.4) takes the form

$$\mathcal{J}(X, \mathbf{h}^{(k-1)}, h^{(k)}) = \mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t) \rightarrow \min\{\mathbf{w} \in R^m, t \in R\}. \quad (2.6)$$

We use the RANSAC algorithm for approximate solving (2.6). RANSAC hypotheses about \mathbf{w} parameters are generated based on the random choice of dissimilar pairs in a training set (Algorithm 2, Fig. 2.5).

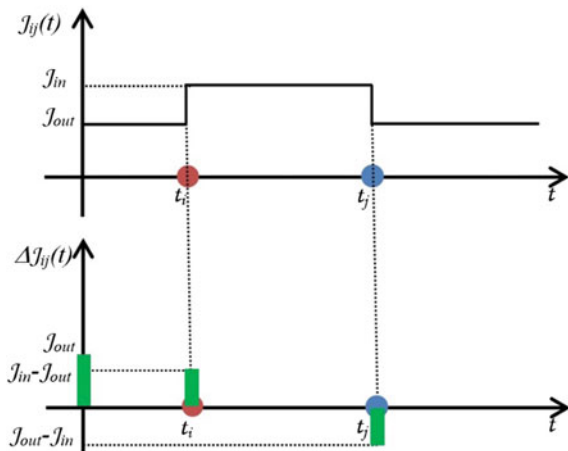
In general case the determination of optimal threshold at the step 3 of this Algorithm 2 could require a lot of time. But in important particular case, the objective function can be represented as a sum of some values corresponded to all pairs of samples from the training set. If these values for each pair depend only on the fact, whether the threshold separates the projections of these samples, or not, then the objective function will be a stepwise function

$$\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}, t) = \sum_{i=1, \dots, N} \sum_{j=1, \dots, N} \mathcal{J}_{ij}(\mathbf{h}^{(k-1)}, \mathbf{w}, t), \quad (2.7)$$

$$\mathcal{J}_{ij}(\mathbf{h}^{(k-1)}, \mathbf{w}, t) = \begin{cases} \mathcal{J}_{ij}^{(in)}, & \text{if } t \in [(\mathbf{x}_i, \mathbf{w}), (\mathbf{x}_j, \mathbf{w})]; \\ \mathcal{J}_{ij}^{(out)}, & \text{otherwise;} \end{cases}$$

and the procedure for optimal threshold search can be implemented more efficiently. For this case (2.7) we propose the special algorithm (Algorithm 3, Fig. 2.6), which requires $O(N^2)$ computations, and the number of computations for each pair from the training set is low enough. For the fixed hypothesis $\mathbf{w} = \mathbf{w}_k$, we arrange

Fig. 2.6 Optimal threshold selection (Algorithm 3): stepwise objective function recovering via accumulation of step values from left to right



projections $t^{(k)}_i = (\mathbf{x}_i, \mathbf{w}_k)$ by increasing and test them as possible threshold values via calculating the $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t^{(k)}_i)$. The idea of this algorithm is to calculate the step values at each projection point and then recover the stepwise objective function via accumulation of step values from left to right.

2.4.4 BHF: Boosted Hashing Forest

Our Learn1BitHash procedure (see Algorithm 1) contains the recursive call of Learn1ProjectionHash procedure (Algorithm 2). Consider the tessellation of X by n -bit coder: $\mathbf{X}_B = \{X_{\mathbf{b}}, \mathbf{b} \in \{0,1\}^n\}$, $X_{\mathbf{b}} = \{\mathbf{x} \in X: \mathbf{h}(\mathbf{x}) = \mathbf{b}\}$, $X = \cup_{\mathbf{b} \in \{0,1\}^n} X_{\mathbf{b}}$. The process of recursive coding is a dichotomy splitting of training set with finding the optimized elementary coder for each subset at each level of tessellation. So, the recursive coder for k th bit

$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = h(\mathbf{w}(\mathbf{h}^{(k-1)}(\mathbf{x})), t(\mathbf{h}^{(k-1)}(\mathbf{x})), \mathbf{x}), \quad (2.8)$$

Algorithm 4: Boosted Hashing Forest

Input data: $X, \mathcal{J}, n_{ORC}, n_{BHF}$.

Output data: $\mathbf{h}(\mathbf{x}): \mathbf{x} \in R^m \rightarrow \mathbf{y} \in \{0,1\}^n$.

Initialization:

$l := 0; \mathbf{h}^{[1,0]} := ()$.

Repeat iterations:

$l := l + 1;$

Form the objective as a function of l -th coding tree:

$\mathcal{J}^{[l]}(X, \mathbf{h}^{[l,l]}) = \mathcal{J}(X, \mathbf{h}^{[1,l-1]}, \mathbf{h}^{[l,l]})$;

Learn l -th coding tree:

$\mathbf{h}^{[l,l]} := \text{GreedyORC}(\mathcal{J}^{[l]}, X, n_{ORC})$;

Add l -th coding tree to the hashing forest:

$\mathbf{h}^{[1,l]}(\mathbf{x}) := (\mathbf{h}^{[1,l-1]}(\mathbf{x}), \mathbf{h}^{[l,l]}(\mathbf{x}))$;

while $l < n_{ORC}$. // stop if the given size of coder is got

is a combination of $2^{(k-1)}$ thresholded projections

$$\begin{aligned} h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) &= \text{Learn1BitHash}(\mathcal{J}, X, \mathbf{h}^{(k-1)}) \\ &= \{\text{Learn1ProjectionHash}(\mathcal{J}, X(\mathbf{h}^{(k-1)}, \mathbf{b}), \mathbf{h}^{(k-1)}), \mathbf{b} \in \{0, 1\}^{(k-1)}\}. \end{aligned} \quad (2.9)$$

Such recursive n -bit coder $\mathbf{h}(\mathbf{x})$ is a *tree* of thresholded projections (Fig. 2.7), which has much more recognition power relative to the n -bit sequence of thresholded projections.

We know that one coding tree cannot provide the fine recognition rate. Besides, the number of projections in a tree grows exponentially with tree depth. So, the training set of some fixed size allows learning the trees with some limited depth only. Due to this, we form the hashing forest via the boosting of hashing trees with optimization

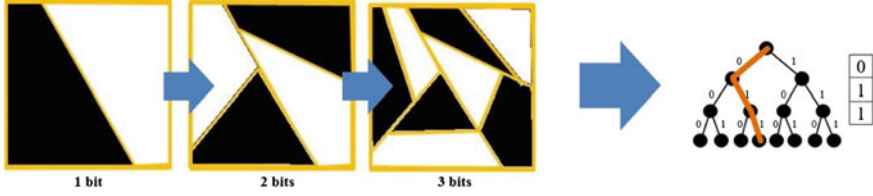
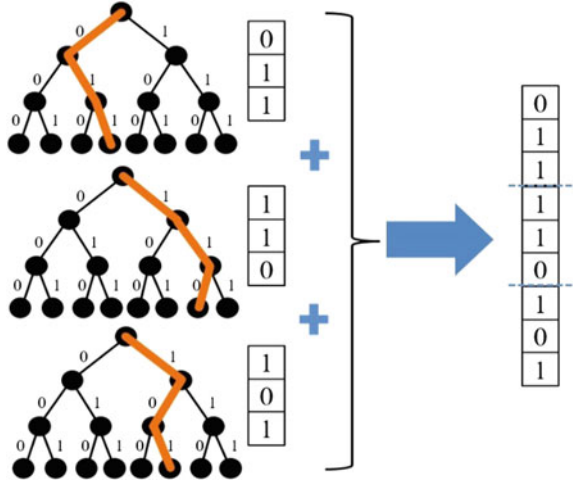


Fig. 2.7 The scheme of recursive coding by binary trees

Fig. 2.8 Output binary code forming via concatenation of binary codes formed by trees of hashing forest (Algorithm 4)



of joint objective function for all trees. We call such approach as Boosted Hashing Forest (BHF) (Algorithm 4, Fig. 2.8).

Here we use the following notation: $n_{ORC} = p$ is a depth of coding tree; $n_{BHF} = n/p$ is a number of trees; $\mathbf{h}^{[1,l]} = (h^{(1)}(\mathbf{x}), \dots, h^{(l)}(\mathbf{x}))$, $\mathbf{h}^{[1,l-1]} = (h^{(1)}(\mathbf{x}), \dots, h^{(l-p)}(\mathbf{x}))$, $\mathbf{h}^{[l,l]} = (h^{(l-p+1)}(\mathbf{x}), \dots, h^{(l)}(\mathbf{x}))$.

2.4.5 BHF: Hashing Forest as a Metric Space

We call the metric space (Y, d_Y) with $d_Y: Y \times Y \rightarrow \mathbb{R}^+$ as n -bit binary coded, if the each $y \in Y$ corresponds to unique $\mathbf{b} \in \{0,1\}^n$, and two decoding functions are given: feature decoder $f_y(\mathbf{b}): \{0,1\}^n \rightarrow Y$ and distance decoder $f_d(\mathbf{b}_1, \mathbf{b}_2): \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{R}^+$, $f_d(\mathbf{b}_1, \mathbf{b}_2) = d_Y(f_y(\mathbf{b}_1), f_y(\mathbf{b}_2))$. This allows define the distance-based objective function (DBOF) for coder $\mathbf{h}(\mathbf{x})$ of the form

$$(X, \mathbf{h}) \rightarrow \min(\mathbf{h}) \Leftrightarrow \mathcal{J}(D_Y) \rightarrow \min(D_Y), \quad (2.10)$$

$$D_Y = \{d_{ij} = f_d(\mathbf{h}(\mathbf{x}_i), \mathbf{h}(\mathbf{x}_j)), \mathbf{x}_i, \mathbf{x}_j \in X, \mathbf{h}(\mathbf{x}) \in \mathbf{H}\}_{i,j=1,\dots,N}.$$

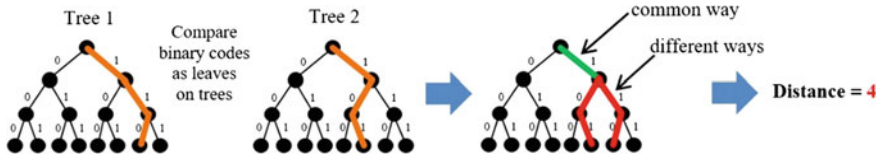


Fig. 2.9 Search index distance as a geodesic distance between codes as corresponding leaves on a coding tree

Such objective function depends on the set of coded distances d_{ij} only. In our current implementation of BHF we match p -bit binary trees via the *search index distance* (Fig. 2.9). It is a geodesic distance between codes as corresponding leaves on a coding tree

$$d_T(y_1, y_2) = f_{dT}(\mathbf{b}_1, \mathbf{b}_2) = 2 \sum_{k=1, \dots, p} (1 - \prod_{l=1, \dots, k} (1 - |b_1^{(l)} - b_2^{(l)}|)). \quad (2.11)$$

Finally, we form a matching distance for total n -dimensional forest containing $q = n/p$ trees as a sum of distances between individual p -bit trees

$$d_{ij} = \sum_{l=1, \dots, q} f_{dT}(\mathbf{h}^{[l, l]}(\mathbf{x}_i), \mathbf{h}^{[l, l]}(\mathbf{x}_j)). \quad (2.12)$$

2.4.6 BHF: Objective Function for Face Verification and Identification

Let the similarity function s describes positive (authentic) and negative (imposter) pairs

$$s_{ij} = \begin{cases} 1, & \text{if class}(\mathbf{x}_i) = \text{class}(\mathbf{x}_j), \\ 0, & \text{otherwise.} \end{cases} \quad (2.13)$$

The “ideal” distance for k -bit binary code, is

$$g^{(k)}_{ij} = \begin{cases} 0, & \text{if } s_{ij} = 1, \\ d_{\max}(k), & \text{otherwise,} \end{cases} \quad (2.14)$$

where $d_{\max}(k)$ is a maximal possible distance. So, the *distance supervision objective function* can be formed as

$$\mathcal{J}_{Dist}(D_Y) = \sum_{i=1, \dots, N} \sum_{j=1, \dots, N} v_{ij} (d_{ij} - g_{ij})^2 \rightarrow \min(D_Y = \{d_{ij}\}_{i,j=1, \dots, N}), \quad (2.15)$$

where v_{ij} are the different weights for authentic and imposter pairs. This objective function (2.15) controls the verification performance (FAR and FRR).

In the identification-targeted biometric applications we need to control both distances and ordering of distances. Let $d^1_k = \max_l \{d_{kl} : s_{kl} = 1\}$ is a distance to the most far authentic and $d^0_k = \min_l \{d_{kl} : s_{kl} = 0\}$ is a distance to the closest imposter for the query $\mathbf{h}(\mathbf{x}_k)$. Then the ordering error e_{ij} for a pair $(\mathbf{x}_i, \mathbf{x}_j)$ can be expressed as

$$e_{ij} = \begin{cases} 1, & \text{if } (s_{ij} = 0 \text{ and } h_{ij} < \max(d_i^1, d_j^1)) \\ & \text{or } (s_{ij} = 1 \text{ and } h_{ij} > \min(d_i^0, d_j^0)) \\ 0, & \text{otherwise} \end{cases} \quad (2.16)$$

The ordering error occurs if imposter is closer than authentic or authentic is more far than imposter. So, the *distance order supervision objective function* can be formed as

$$\mathcal{J}_{Ord}(D_Y) = \sum_{i=1, \dots, N} \sum_{j=1, \dots, N} v_{ij} (d_{ij} - g_{ij})^2 e_{ij} \rightarrow \min(D_Y = \{d_{ij}\}_{i,j=1, \dots, N}). \quad (2.17)$$

Here we penalize the difference between d_{ij} and objective distance g_{ij} like in (2.15), but only in case that the ordering error (2.16) occurs for this pair. So, criterion (2.17) directly controls the face identification characteristics (CMC).

Finally, for obtaining both verification and identification we combine the (2.15) and (2.17) resulting in

$$\begin{aligned} (D_Y) &= \alpha \mathcal{J}_{Dist}(D_Y) + (1 - \alpha) \mathcal{J}_{Ord}(D_Y) \\ &= \sum_{i=1, \dots, N} \sum_{j=1, \dots, N} v_{ij} (d_{ij} - g_{ij})^2 (e_{ij} + \alpha(1 - e_{ij})) \\ &\rightarrow \min(D_Y = \{d_{ij}\}_{i,j=1, \dots, N}), \end{aligned} \quad (2.18)$$

where $\alpha \in [0, 1]$ is a tuning parameter.

2.4.7 BHF Implementation for Learning Face Representation

For enhancement of our face representation learning we use some additional semi-heuristic modifications of described scheme. The goal distance (2.14) is modified

$$g^{(k)}_{ij} = \begin{cases} 0, & \text{if } s_{ij} = 1, \\ m^{(k-1)}_1 + 3\sigma^{(k-1)}_1, & \text{otherwise,} \end{cases} \quad (2.19)$$

where $m^{(k-1)}_1$ and $\sigma^{(k-1)}_1$ are the mean value and standard deviation of authentic coded distances. Such goal distance (2.19) excludes the penalizing of imposter pairs,

which could not be treated as authentic. In (2.18) we use the adaptive weighting of pairs at each k th step of boosting

$$v_{ij}^{(k)} = \begin{cases} \gamma / a^{(k)}, & \text{if } s_{ij} = 1, \\ 1/b^{(k)}, & \text{otherwise,} \end{cases} \quad (2.20)$$

$$\begin{aligned} a^{(k)} &= \sum_{i=1, \dots, N} \sum_{j=1, \dots, N} s_{ij} (d_{ij} - g_{ij})^2 (e_{ij} + \alpha(1 - e_{ij})), \\ b^{(k)} &= \sum_{i=1, \dots, N} \sum_{j=1, \dots, N} (1 - s_{ij}) (d_{ij} - g_{ij})^2 (e_{ij} + \alpha(1 - e_{ij})), \end{aligned} \quad (2.21)$$

where $a^{(k)}$ and $b^{(k)}$ provide the basic equal weight for all authentic and imposter pairs, and tuning parameter $\gamma > 1$ gives the slightly larger weights to authentic pairs.

We split the input m -dimensional feature vector to the set of independently coded subvectors with fixed sizes from the set $\mathbf{m} = \{m_{\min}, \dots, m_{\max}\}$. At the each step of boosting we get the subvector with corresponding BHF elementary coder providing the best contribution to the objective function. The output binary vector of size n consists of some independently grown parts of size $n_{BHF} < n$. Such learning strategy prevents the premature saturation of objective function.

So, our binary face hashing is implemented with the following set of free parameters: \mathbf{m} , n_{ORC} , n_{BHF} , k_{RANSAC} , α and γ . The type of coded metrics is a free parameter of our approach too.

2.5 Experiments

In this section, we describe our methodology for learning and testing CNHF, report our results in Hamming embedding task, compare proposed BHF to original Boosted SSC, explore the CNHF performance w.r.t. depth of coding trees and compare CNHL and CNHF to best methods on LFW. We test the verification accuracy by the standard LFW unrestricted with outside labeled data protocol. Our CMC and rank-1 tests follow the methodology described in [2].

2.5.1 Methodology: Learning and Testing CNHF

The basic CNN is trained on CASIA-WebFace dataset. Face images are aligned by rotation of eye points to horizontal position with fixed eye-to-eye distance and crop to 128×128 size. The open source deep learning framework Caffe (<http://caffe.berkeleyvision.org/>) is used for training the basic CNN for multi-class face identification in the manner [24, 31]. The hashing forest is trained on the dataset containing 1,000 authentic pairs and correspondingly 999,000 imposter pairs of Faces in the Wild images (not from the testing LFW set). Finally, the family of CNHF coders is formed by proposed BHF: Hamming embedding coders $2,000 \times 1$ bit (250 byte), 200×1 bit (25 byte) and 32×1 bit (4 byte) of size; Hashing forest coders containing

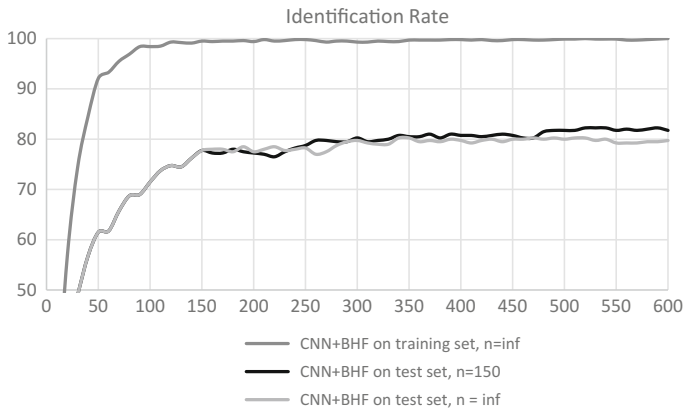


Fig. 2.10 Example of n_{BHF} parameter selection

2,000 trees with 2–7 bits depth (0.5–1.75 Kbyte of size). We used the common setting of BHF parameters: $\mathbf{m} = \{8, 16, 32\}$, $k_{RANSAC} = 50$, $\alpha = 0.25$, $\gamma = 1.1$. But we set $n_{BHF} = 200$ for CNN+BHF- 200×1 , $n_{BHF} = 500$ for CNN+BHF- $2,000 \times 1$ and $n_{BHF} = 100$ for CNHF- $2,000 \times 7$. Such n_{BHF} parameter values are determined experimentally based on the analysis of the speed of identification rate growing w.r.t. number of code bits in the hashing process. We determine the minimal number of generated code bits, which provides the best identification rate on training database in the hashing process. Figure 2.10 demonstrates the example of n_{BHF} parameter selection. Graphs for identification score w.r.t. number of coding trees are shown both for training and for testing set. One can see that on the training set the identification stabilizes approximately at the level of 150 coding trees. Correspondingly in testing the identification rate for $n_{BHF} = 150$ (600 coding trees are divided into four independent coding forests) outperforms the identification rate for $n_{BHF} = \infty$ (600 coding trees are not divided to independent parts) by $\approx 2\%$. The evaluation is performed on the Labeled Faces in the Wild (LFW) dataset. All the images in LFW dataset are processed by the same pipeline as in [11] and normalized to 128×128 .

2.5.2 Hamming Embedding: CNHL Versus CNN, BHF Versus Boosted SSC

In this subsection, we test our approach in Hamming embedding task, so, CNHF degrades to CNHL. We compare CNHL to basic CNN on LFW via verification accuracy and ROC curve (Table 2.4 and Fig. 2.11a). The CNN face representation is formed like in [34] as a vector of activations of 256 top hidden layer neurons. The cosine similarity (CNN+CS) and L2-distance (CNN+L2) are applied for matching. CNHL coders 2,000 and 200 bit of size are trained by BHF and matched by

Table 2.4 Verification accuracy on LFW, code size, and matching speed of CNN and CNHL

Solution	Accuracy	Template size	Matches in sec
CNN+L2	0.947	8,192 bit	2,713,222
CNN+BHF-200×1	0.963	200 bit	194,986,071
CNN+CS	0.975	8,192 bit	2,787,632
CNN+BHF-2000×1	0.9814	2,000 bit	27,855,153

Hamming distance (CNN+BHF-2,000 × 1 and CNN+BHF-200 × 1 correspondingly). Our solution CNN+BHF-2,000 × 1 achieves verification accuracy 98.14% on LFW, which outperforms all other CNN-based solutions. Moreover, our 25-byte length solution CNN+BHF-200 × 1 outperforms CNN+L2. Table 2.4 additionally demonstrates the gain in template size and matching speed.

We compare CNHL trained by BHF to CNHL trained by original Boosted SSC. Figure 2.11c demonstrates that proposed BHF essentially outperforms Boosted SSC in identification (rank-1) on LFW for all binary template sizes. The maximal rank-1 is 0.91 for BHF-2,000 × 1 and 0.865 for BoostSSC-2,000 × 1 (relative to 0.899 for CNN+CS). The ROC graph for CNN+BHF is monotonously better than for CNN+BoostSSC with same template size (Fig. 2.11a). Figure 2.11b contains the CMC graphs (ranks 1–10), which demonstrate that BHF outperforms Boosted SSC with same template size (additionally note that CNN+BHF-2,000 × 1 outperforms CNN+CS).

2.5.3 CNHF: Performance w.r.t. Depth of Trees

CNHF with 2,000 output features formed by 7-bit coding trees (CNHF-2,000 × 7) achieves 98.59% on LFW. The identification result of CNHF-2,000 × 7 is 93% rank-1 on LFW relative to 89.9% rank-1 for CNN+CS. Figure 2.11f presents the ROC curves for CNHF with different depth coding trees. The forest with 7-bit coding trees is the best by ROC, but 6-bit and 5-bit depth solutions are very close. We suppose that the reason of this result is a limited amount of hashing forest training set. Figure 2.11d, e demonstrates that CNHF-2,000 × 7 outperforms basic CNN+CS and CNHF-2,000 × 1 both in verification (ROC) and in identification (CMC). So, we can conclude that the adding of hashing forest on the top of CNN allows both generating the compact binary face representation and increasing the face verification and especially identification rates.

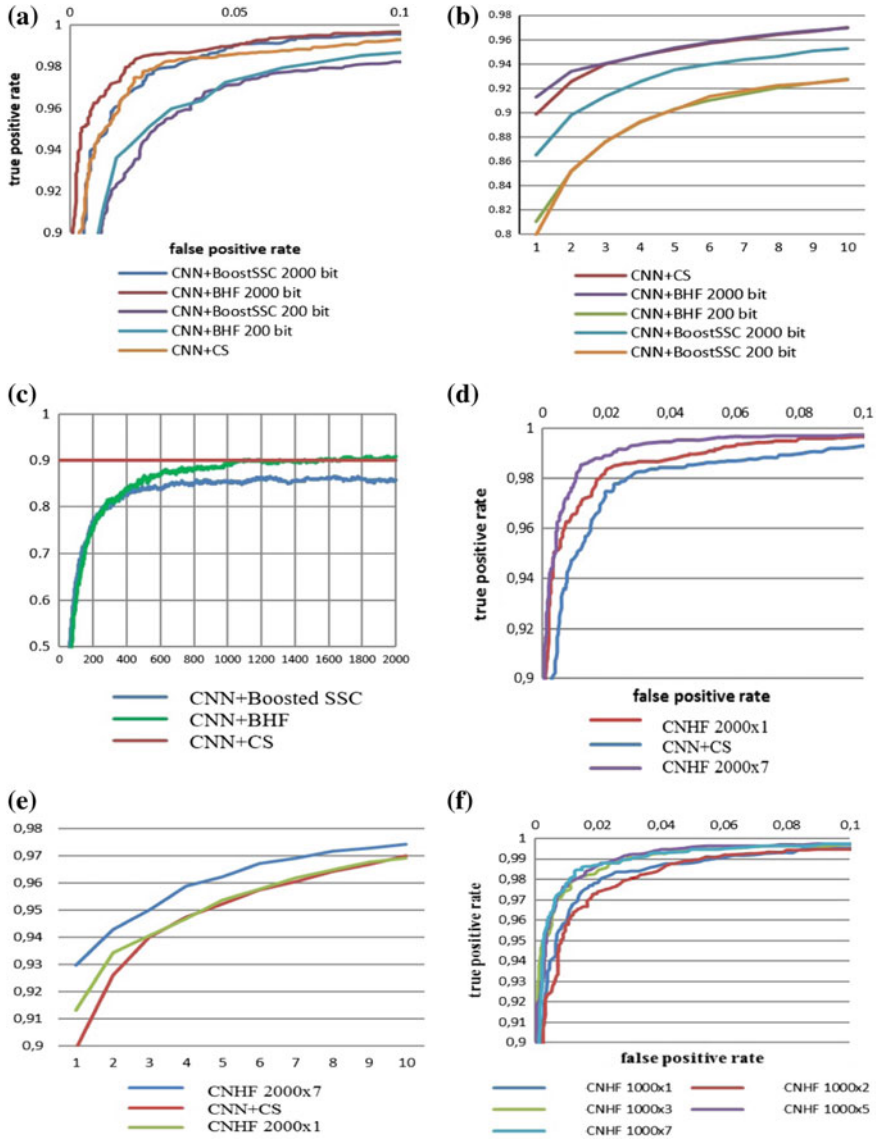


Fig. 2.11 **a** ROC curves, **b** CMC curves, **c** identification performance (rank 1) on LFW relative to the size of biometric template in bits for proposed BHF(CNN+BHF) and original Boosted SSC(CNN +BoostSSC) and best basic CNN solution without hashing - CNN + Last hidden layer + cosine similarity (CNN+CS), **d** ROC curves, **e** CMC curves for CNN+CS, CNHF-2000 \times 1, CNHF-2000 \times 7, **f** ROC curves for CNHF-1000 \times p-bit trees

Table 2.5 Verification accuracy on LFW

Method	Accuracy
WebFace [24]	0.9613
CNHL-200×1	0.963 ± 0.00494
DeepFace-ensemble [21]	0.9730 ± 0.0025
DeepID [19]	0.9745 ± 0.0026
MFM Net [25]	0.9777
CNHL-2000×1	0.9814
CNHF-2000×7	0.9859
DeepID2 [17]	0.9915 ± 0.0013
DeepID3 [18]	0.9953 ± 0.0010
Baidu [11]	0.9977 ± 0.0006

2.5.4 CNHL and CNHF Versus Best Methods on LFW

We compare our CNHF solutions to state-of-the-art methods (best on LFW) via verification accuracy (Table 2.5). CNHF-2,000 × 1 outperforms DeepFace-ensemble [30], DeepID [27], WebFace [35] and MFM Net [34]. The DeepID2 [24], DeepID3 [26] and Baidu [14] multi-patch CNNs outperform our CNHF-2,000 × 1 based on single net.

Note that our CNHF-200 × 1 (25 byte) hash demonstrates 96.3% on LFW. Compare this result to previous best CNHL result [6]. On the one hand, the extreme-short 32-bit binary face representation [6] achieves 91% verification on LFW. Our CNHF 32 × 1 provides 90% only. On the other hand, face representation [6] requires 1000 bit for achieving the 96% verification on LFW. So, our CNHF-200 × 1 solution improves this face packing result in 5 times.

The identification result (rank-1) of our real-time coder CNHF-2,000 × 7 is 0.93 on LFW. It is close enough to best reported identification result of essentially deeper and slower multi-patch DeepID3 CNN [25] (0.96 rank-1 on LFW). Baidu [13] declares even better result (0.98 rank-1 on LFW), but they use the training set 1.2 million images of size w.r.t. 400 thousand images in our case.

2.6 Conclusion and Discussion

We develop the family of CNN-based binary face representations for real-time face identification. Our Convolutional Network with Hashing Forest (CNHF) generates binary face templates at the rate of 40+ fps with CPU Core i7 and 120+ fps with GPU GeForce GTX 650. Our 2,000 × 1-bit face coder provides the compact face coding (250 byte) with simultaneous increasing of verification (98.14%) and identification (91% rank-1) on LFW. Our 200 × 1-bit face coder provides the 40-time gain in

template size and 70-time gain in a matching speed with 1% decreasing of verification accuracy relative to basic CNN (96.3% on LFW). Our CNHF with 2000 output 7-bit coding trees ($\text{CNHF-}2,000 \times 7$) achieves 98.59% verification accuracy and 93% rank-1 on LFW (add 3% to rank-1 of basic CNN).

We use the multiple convolution deep network architecture for acceleration of source Max-Feature-Map (MFM) CNN architecture [31]. We propose and implement the new binary hashing technique, which forms the output feature space with given metric properties via joint optimization of face verification and identification. This Boosted Hashing Forest (BHF) technique combines the algorithmic structure of Boosted SSC approach and the binary code structure of forest hashing. Our experiments demonstrate that BHF essentially outperforms the original Boosted SSC in face identification test.

In the future we will try to achieve the better recognition rates via CNHF based on multi-patch CNN, which we can use for nonreal-time applications. We will evolve and apply the proposed BHF technique for different data coding and dimension reduction problems (supervised, semi-supervised and unsupervised). Additionally, we will investigate the influence of the output metric space properties in the process of hashing forest learning.

Acknowledgements This work is supported by grant from Russian Science Foundation (Project No. 16-11-00082).

References

1. M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in *Proceedings of the NIPS*, vol. 14 (2001), pp. 585–591
2. L. Best-Rowden, H. Han, C. Otto, B. Klare, A.K. Jain, Unconstrained face recognition: identifying a person of interest from a media collection. *IEEE Trans. Inf. Forensic Secur.* **9**(12), 2144–2157 (2014)
3. Z. Cao, Q. Yin, X. Tang, J. Sun, Face recognition with learning-based descriptor, in *Proceedings of the CVPR* (2010), pp. 2707–2714
4. D. Chen, X. Cao, F. Wen, J. Sun, Blessing of dimensionality: high-dimensional feature and its efficient compression for face verification, in *Proceedings of the CVPR* (2013), pp. 3025–3032
5. H. Fan, Z. Cao, Y. Jiang, Q. Yin, C. Doudou, Learning deep face representation (2014), [arXiv:1403.2802](https://arxiv.org/abs/1403.2802)
6. H. Fan, M. Yang, Z. Cao, Y. Jiang, Q. Yin, Learning compact face representation: packing a face into an int32, in *Proceedings of the ACM International Conference on Multimedia* (2014), pp. 933–936
7. A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in *Proceedings of the VLDB* (1999), pp. 518–529
8. Y. Gong, S. Lazebnik, A. Gordo, F. Perronnin, Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(12), 2916–2929 (2012)
9. K. Grauman, R. Fergus, Learning binary hash codes for large-scale image search, *Machine Learning for Computer Vision* (Springer, Berlin, 2013), pp. 49–87
10. K. He, F. Wen, J. Sun, K-means hashing: an affinity-preserving quantization method for learning binary compact codes, in *Proceedings of the CVPR* (2013), pp. 2938–2945

11. G.-B. Huang, M. Mattar, H. Lee, E. Learned-Miller, Learning to align from scratch, in *Proceedings of the NIPS*, vol. 25 (2012)
12. G. Irie, L. Zhenguo, W. Xiao-Ming, C. Shih-Fu, Locally linear hashing for extracting non-linear manifolds, in *Proceedings of the CVPR* (2014), pp. 2115–2122
13. J. Liu, Y. Deng, T. Bai, Z. Wei, C. Huang, Targeting ultimate accuracy: face recognition via deep embedding (2015), [arXiv:1506.07310](#)
14. W. Liu, J. Wang, R. Ji, Y.-G. Jiang, S.-F. Chang, Supervised hashing with kernels, in *Proceedings of the CVPR* (2012), pp. 2074–2081
15. Y. Mishina, M. Tsuchiya, H. Fujiyoshi, Boosted random forest. *IEICE Trans.* **E98D**(9), 1630–1636 (2015)
16. H.-V. Nguyen, L. Bai, Cosine similarity metric learning for face verification, in *Proceedings of the ACCV* (2010), pp. 709–720
17. Q. Qiu, G. Sapiro, A. Bronstein, Random forests can hash (2014), [arXiv:1412.5083](#)
18. R. Salakhutdinov, G. Hinton, Semantic hashing. *Int. J. Approx. Reason.* **50**(7), 969–978 (2009)
19. F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: a unified embedding for face recognition and clustering, in *Proceedings of the CVPR* (2015), pp. 815–823
20. G. Shakhnarovich, Learning task-specific similarity, Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge (2005)
21. G. Shakhnarovich, P. Viola, T. Darrell, Fast pose estimation with parameter sensitive hashing. *Proc. Comput. Vis.* **2**, 750–757 (2003)
22. J. Springer, X. Xin, Z. Li, J. Watt, A. Katsaggelos, Forest hashing: expediting large scale image retrieval, in *Proceedings of the ICASSP* (2013), pp. 1681–1684
23. Y. Sun, X. Wang, X. Tang, Deep learning face representation by joint identification-verification, in *Proceedings of the NIPS*, vol. 27 (2014)
24. Y. Sun, X. Wang, X. Tang, Deep learning face representation from predicting 10,000 classes, in *Proceedings of the CVPR* (2014), pp. 1891–1898
25. Y. Sun, X. Wang, X. Tang, DeepID3: face recognition with very deep neural networks (2015), [arXiv:1502.00873](#)
26. Y. Taigman, L. Wolf, T. Hassner, Multiple one-shots for utilizing class label information, in *Proceedings of the BMVC* (2009)
27. Y. Taigman, M. Yang, M. Ranzato, L. Wolf, DeepFace: closing the gap to human-level performance in face verification, in *Proceedings of the CVPR* (2014), pp. 1701–1708
28. C. Vens, F. Costa, Random forest based feature induction, in *Proceedings of the ICDM* (2011), pp. 744–753
29. W. Wang, J. Yang, J. Xiao, S. Li, D. Zhou, Face recognition based on deep learning, in *Proceedings of the HCC*, vol. 8944 (2015), pp. 812–820
30. Y. Weiss, A. Torralba, R. Fergus, Spectral hashing, in *Proceedings of the NIPS*, vol. 21 (2008)
31. X. Wu, Learning robust deep face representation (2015), [arXiv:1507.04844](#)
32. D. Yi, Z. Lei, S. Liao, S.Z. Li, Learning face representation from scratch (2014), [arXiv:1411.7923](#)
33. G. Yu, J. Yuan, Scalable forest hashing for fast similarity search, in *Proceedings of the ICME* (2014), pp. 1–6
34. L. Zhang, Y. Zhang, X. Gu, J. Tang, Q. Tian, Topology preserving hashing for similarity search, in *Proceedings of the ACM International Conference on Multimedia* (2013), pp. 123–132
35. E. Zhou, Z. Cao, Q. Yin, Naive-deep face recognition: touching the limit of LFW benchmark or not? (2015), [arXiv:1501.04690](#)

Deep Learning for Biometrics

Bhanu, B.; Kumar, A. (Eds.)

2017, XXXI, 312 p. 117 illus., 96 illus. in color.,

Hardcover

ISBN: 978-3-319-61656-8