Real-Time Face Identification via CNN and Boosted Hashing Forest

Vladimir Gorbatsevich, *Yury Vizilter, Andrey Vorotnikov and Nikita Kostromov*
State Research Institute of Aviation Systems (GosNIIAS), Moscow, Russia
viz@gosniias.ru, gvs@gosniias.ru, vorotnikov@gosniias.ru, nikita-kostromov@yandex.ru

IDP-2016

MOTIVATION: real-time face identification

- Real-time means **smallest templates and fastest search** ⇒ binary templates with Hamming distance;
- **State-of-the-art recognition rates** ⇒ convolutional neural networks (CNN) with non-binary output features compared by better metrics (L2, cosine, etc.).

Various applications ⇒ different requirements to:
• template size, • template generation speed,
• template matching speed, • recognition rate.

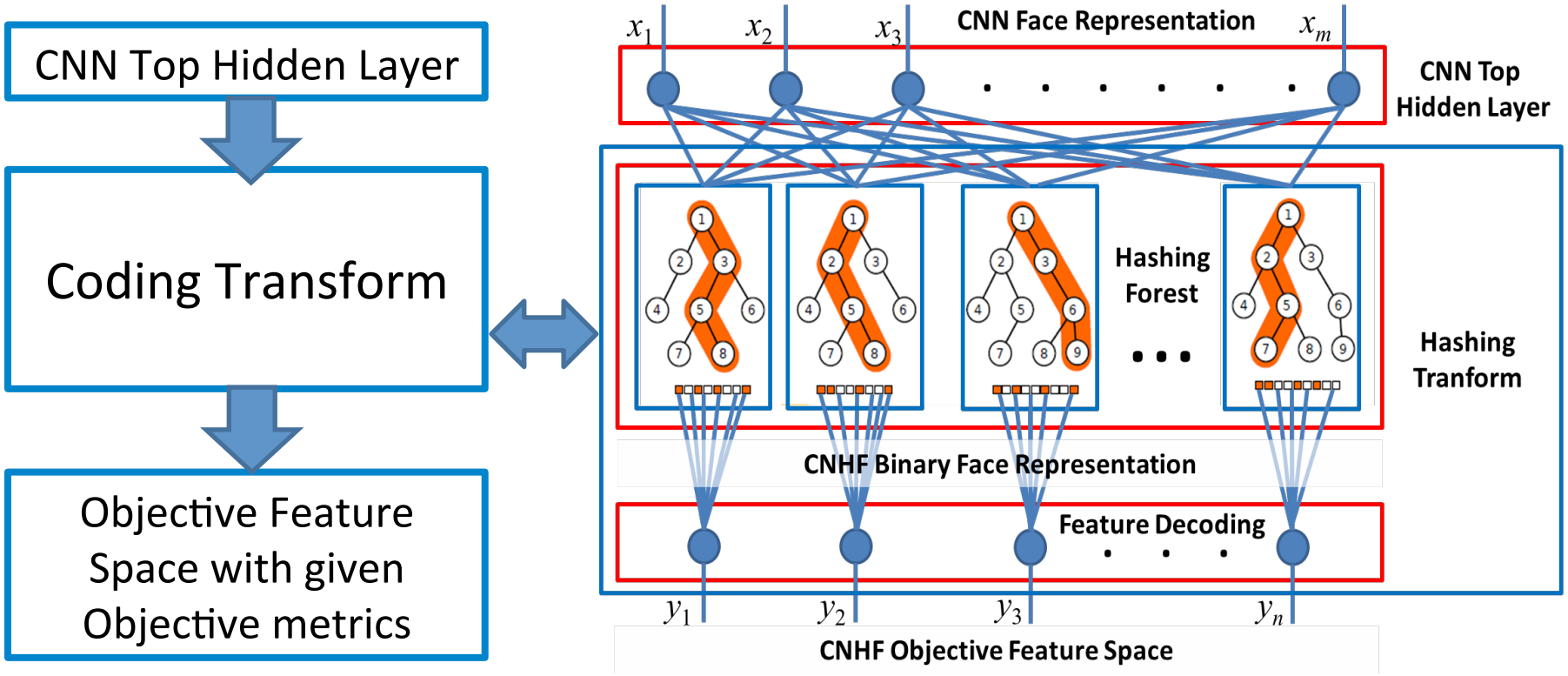
Our purpose: construct the **family of face representations**, which **continuously varies from "compact & fast" to "large & powerful"…**

…with the same engine.

Is it possible?

MAIN IDEA: Convolutional Network with Hashing Forrest (CNHF)

CNHF = CNN + Hashing Transform based on Hashing Forest (HF)



(Depth of trees × Coded metrics × Coding objective) = Family of face representations based on the same CNN.

**In case of 1-bit coding "trees" and Hamming distance CNHF provides the Hamming embedding.*

RELATED WORK

Related work topics:

- CNN for face recognition
  - learning approach
  - face representation
  - matching metrics
- Binary Hashing and Hamming Embedding
- Convolutional Networks + Binary Hashing
- Forest Hashing and Boosted Forest

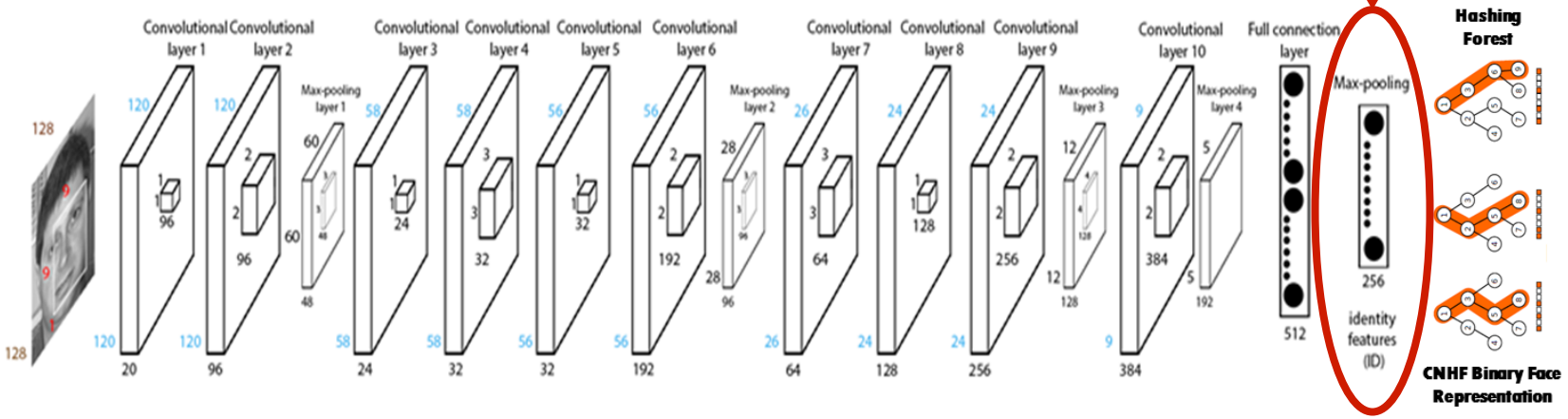
RELATED WORK: CNN for face recognition

CNN learning approaches:

- learn CNN for multi-class face identification with classes corresponding to persons
  Y. Taigman at al., "DeepFace: closing the gap to human-level performance in face verification," 2014.
  E. Zhou at al., "Naive-deep face recognition: Touching the limit of LFW benchmark or not?" 2015.
- learn the similarity metrics by training two identical CNNs (Siamese Architecture)
  H. Fan at al., "Learning deep face representation," 2014.
  W. Wang at al., "Face recognition based on deep learning," 2015.
- combine these approaches
  Y. Sun at al., "Deep learning face representation by joint identification-verification," 2014.
  Y. Sun at al., "DeepID3: Face recognition with very deep neural networks," 2015.

Face representation:

- output of the (top) hidden layer:



RELATED WORK: CNN for face recognition

Matching metrics:

- L1-distance/Hamming distance
  H. Fan, M. Yang, Z. Cao, Y. Jiang and Q. Yin, "Learning Compact Face Representation: Packing a Face into an int32," Proc. ACM Int. Conf. Multimedia, pp. 933-936, 2014.
- L2-distance
  D. Chen at al., "Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification," 2013.
  W. Wang at al., "Face recognition based on deep learning," 2015.
- cosine similarity
  Y. Sun at al., "Deep learning face representation by joint identification-verification," 2014.
  Y. Taigman at al., "DeepFace: closing the gap to human-level performance in face verification," 2014.
  X. Wu, "Learning robust deep face representation," 2015.
- other

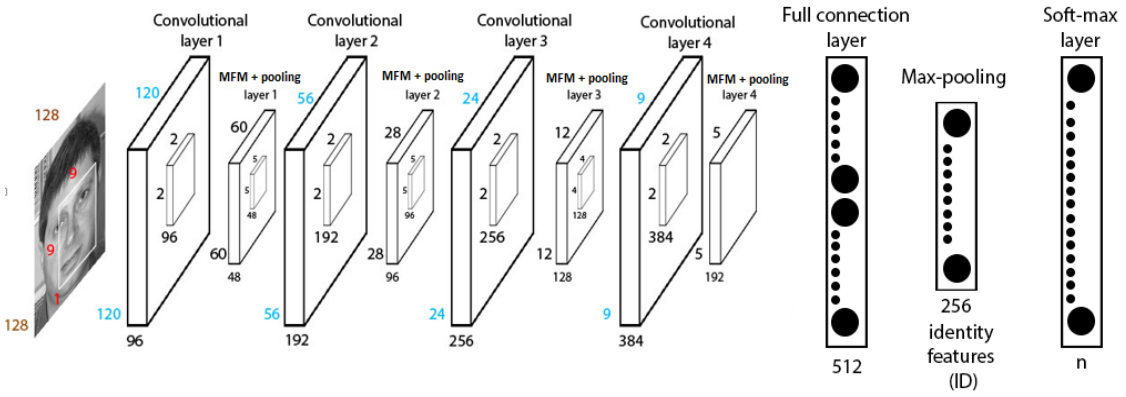
Our approach:

- any given metrics (including Hamming distance and special metrics for forest of binary trees matching)

RELATED WORK: CNN for face recognition

CNN architectures:

- **multi-patch** deep nets for different parts of face (state-of-the-art rates!)
  J. Liu at al., "Targeting ultimate accuracy: face recognition via deep embedding," 2015.
  Y. Sun at al., "Deep learning face representation by joint identification-verification," 2014.
  Y. Sun at al., "DeepID3: Face recognition with very deep neural networks," 2015.
- **single nets** (can be efficient enough with essentially lower computational cost)
  Z. Cao at al., "Face Recognition with Learning-based Descriptor" 2010.
  Omkar M. Parkhi at al., "Deep Face Recognition", 2015
  - Max-Feature-Map (MFM) architecture
    X. Wu, "Learning robust deep face representation," 2015.



Our architecture: (*based on the MFM architecture*)

- learning the **basic single CNN** with Max-Feature-Map (**MFM**) architecture
- transforming the layers of learned CNN to the **multiple convolution architecture** for real-time implementation

RELATED WORK: Binary Hashing and Hamming Embedding

Binary hashing techniques:

A. Gionis at al., "Similarity search in high dimensions via hashing," 1999.

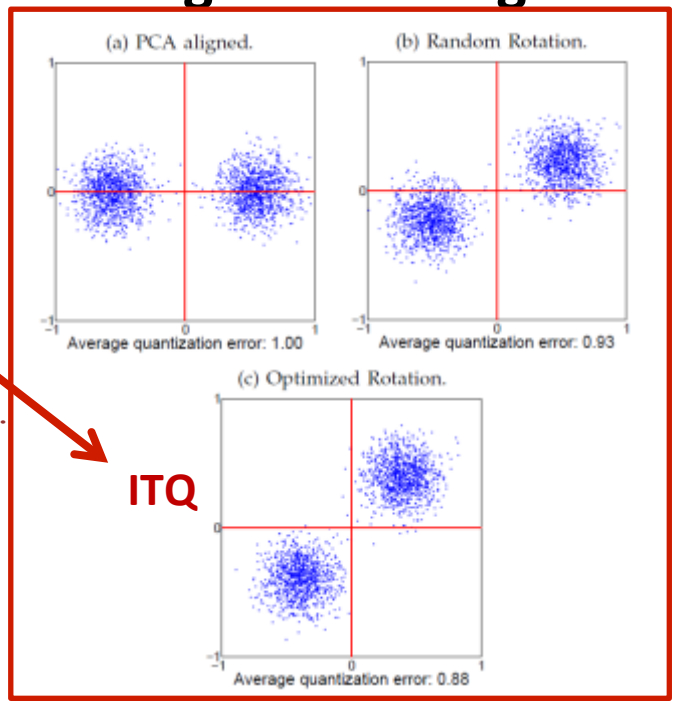
Y. Gong at al., "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," 2012.

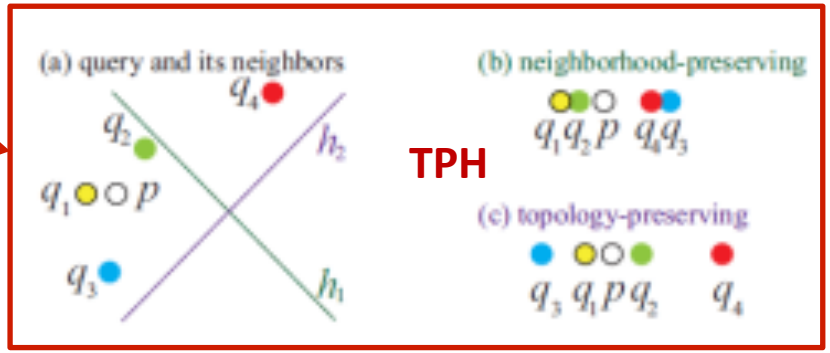
K. He at al., "K-means Hashing: An affinity-preserving quantization method for learning binary compact codes," 2013.

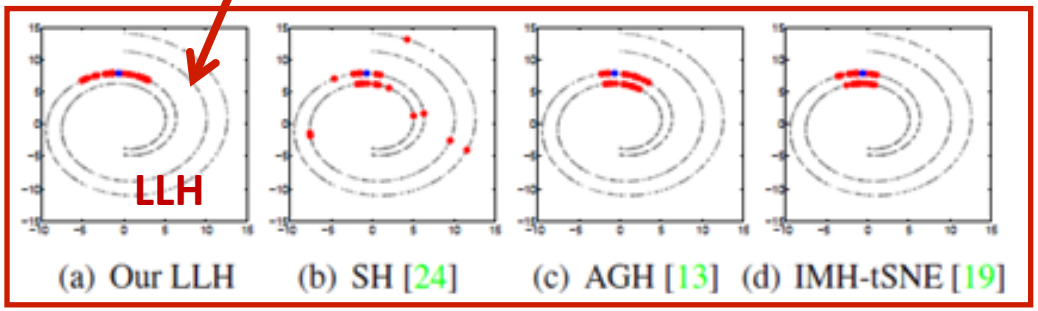
W. Liu at al., "Supervised hashing with kernels," 2012.

Manifold hashing techniques:

- Spectral Hashing
  Y. Weiss at al., "Spectral Hashing," 2008.
- Topology Preserving Hashing (TPH)
  L. Zhang at al., "Topology preserving hashing for similarity search," 2013.
- Locally Linear Hashing (LLH)
  G. Irie at al., "Locally linear hashing for extracting non-linear manifolds," 2014.





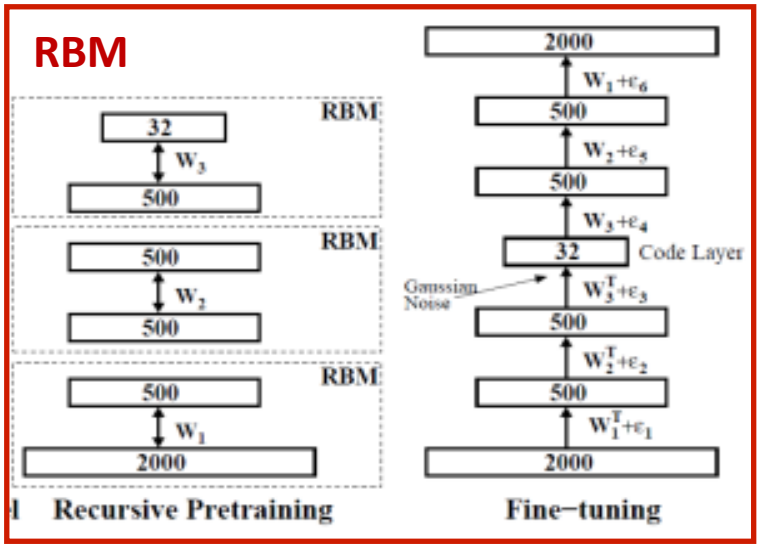


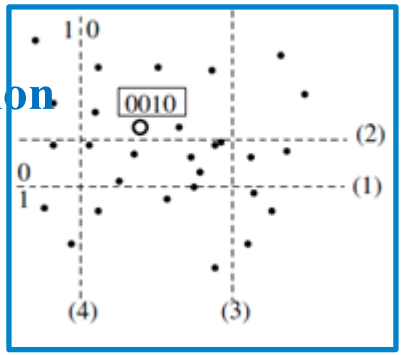
**Figures are taken from cited papers!*

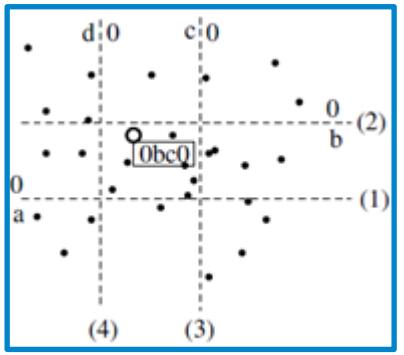
RELATED WORK: Binary Hashing and Hamming Embedding

Closest approaches:

- Restricted Boltzmann Machines (RBM)
  R. Salakhutdinov and G. Hinton, "Semantic hashing," 2009.
- Boosted Similarity Sensitive Coding (Boosted SSC)
  G. Shakhnarovich, "Learning task-specific similarity," 2005.
  G. Shakhnarovich at al., "Fast pose estimation with parameter sensitive hashing," 2003.

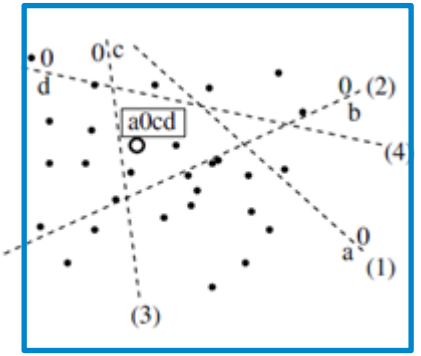






Our approach:

- Boosted Hashing Forest (*generalization of Boosted SSC*):
  - boosting the **hashing forest** in the manner of Boosted SSC;
  - induction of **any given metrics** in the coded feature space;
  - optimizing **any given task-specific objective function**.



RELATED WORK: Binary Hashing via Convolutional Networks

Closest approach:

- **binary face coding via CNN with *hashing layer* (CNHL)**
- **learning CNN and hashing layer *together* via back propagation technique**

  H. Fan, M. Yang, Z. Cao, Y. Jiang and Q. Yin, "Learning Compact Face Representation: Packing a Face into an int32," *Proc. ACM Int. Conf. Multimedia*, pp. 933-936, 2014.

Great result: 32-bit binary face representation provides 91% verification on LFW!

Problems:

- results for **larger templates** are too far from state-of-the-art;
- direct optimization of **more complex face coding criterions** is not available;
- we **cannot perform the learning if CNHF** via back propagation.

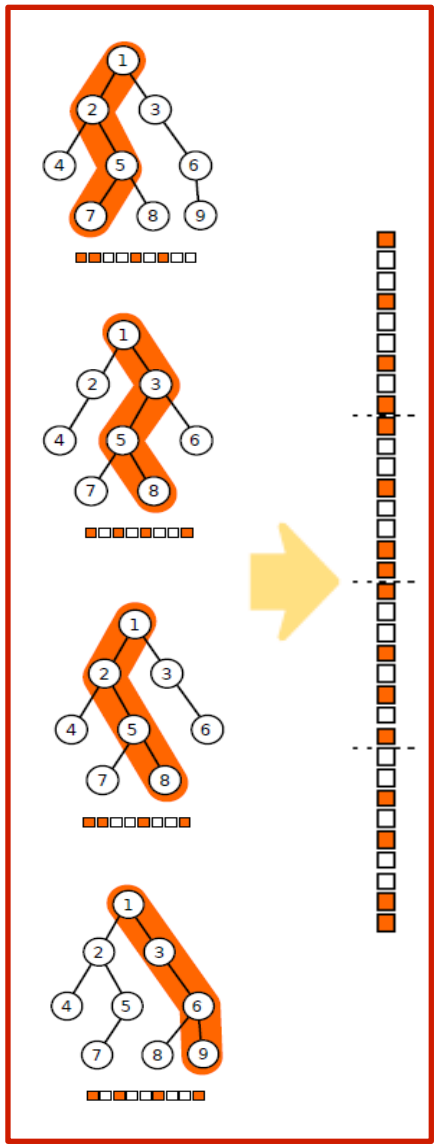
Our approach:

- binary face coding via CNN with *hashing transform*
- **go back to the *two-step learning procedure*:**
  - learn basic CNN first,
  - learn hashing transform second.

RELATED WORK: Forest Hashing and Boosted Forest

Previous Forest Hashing techniques (*non-boosted*):

- random forest semantic hashing scheme with information-theoretic code aggregation
  Q. Qiu at al., "Random Forests Can Hash," 2014.
- feature induction based on random forest for learning regression and multi-label classification
  C. Vens and F. Costa, "Random Forest Based Feature Induction", 2011.
- forest hashing with special order-sensitive Hamming distance
  G. Yu and J. Yuan, "Scalable forest hashing for fast similarity search", 2014.
- combination of *kd*-trees with hashing technique
  J. Springer at al., "Forest hashing: Expediting large scale image retrieval," 2013.



Previous Boosted Forest approach:

- Boosted Random Forest (*out of the binary hashing topic*)
  Y. Mishina at al., "Boosted Random Forest," 2015.

Our approach:

- **Boosted Hashing Forest (generalization of Boosted SSC):**
  - metric feature space induction via forest hashing;
  - **hashing forest boosting in the manner of Boosted SSC;**
  - **optimizing the biometric-specific objective function.**

SUMMARY OF INTRODUCTION

Contributions of this paper:

(1) The family of **real-time** face representations based on **multiple convolution CNN** with **hashing forest** (CNHF);

(2) New **biometric-specific objective function** for joint optimization of face verification and identification;

(3) **Boosted Hashing Forest (BHF) technique** for optimized feature induction with generic form of coding objective, coded feature space and hashing function.

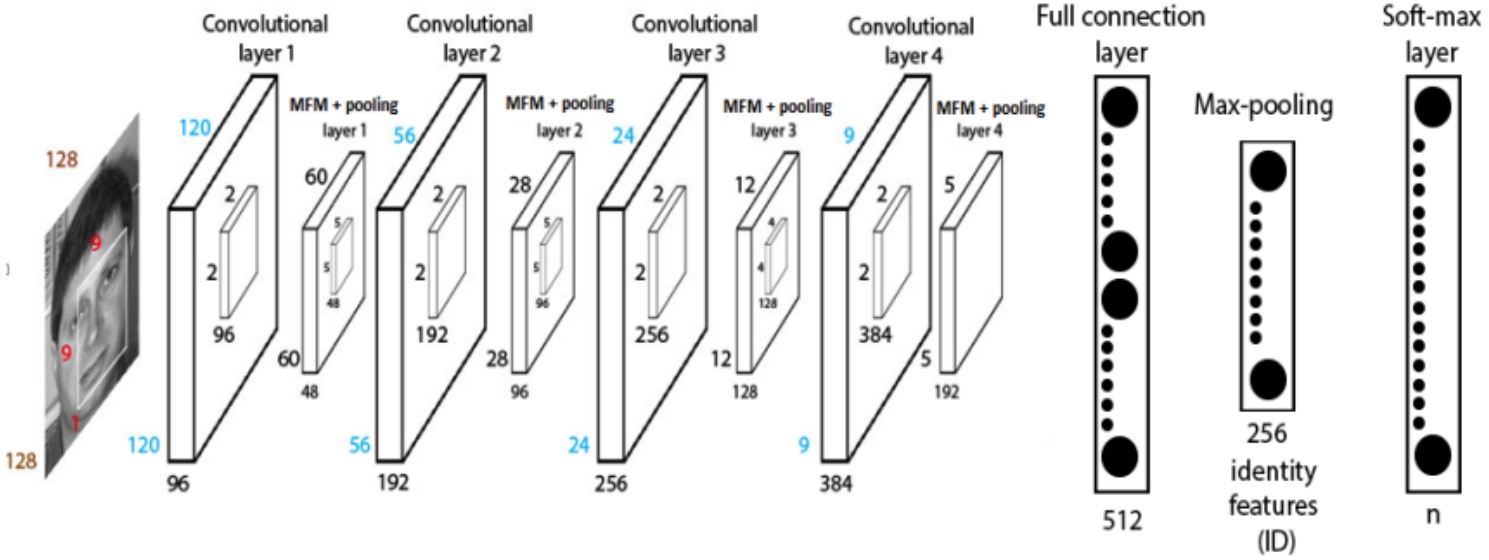
Content of presentation reminder:

- Architecture and learning of CNHF with multiple convolution layers;
- Boosted Hashing Forest technique and its implementation for face coding;
- **Experimental results on LFW**
- Conclusion and discussion

CNHF: basic single net with MFM architecture

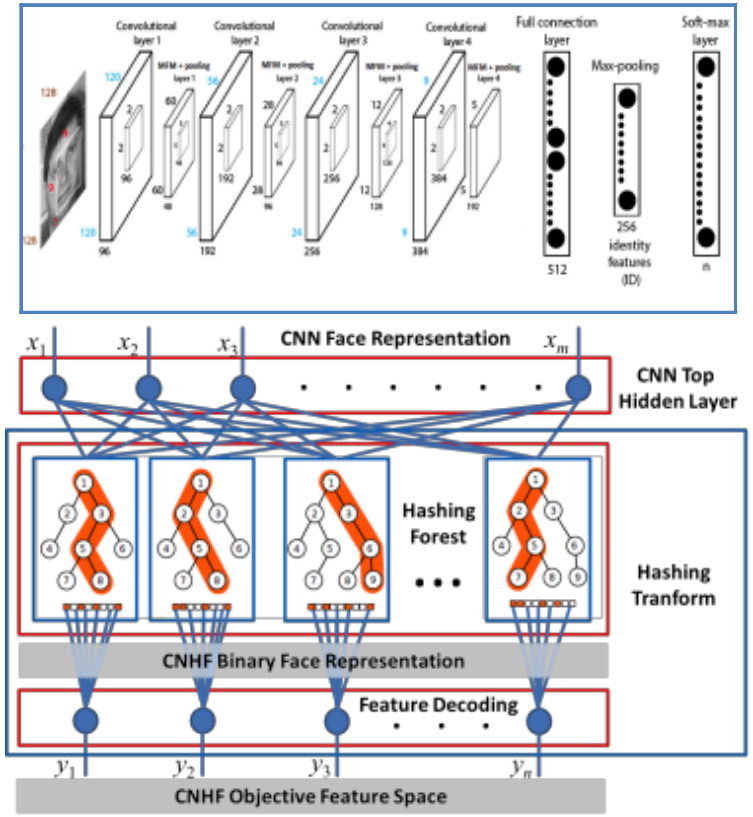
Original Max-Feature-Map (MFM) architecture:

- Max-Feature-Map instead of ReLU activation function
- **4 convolutional layers**
- 4 layers of pooling + MFM pooling
- 1 fully connected layer
- 1 sofmax layer



X. Wu, "Learning robust deep face representation," 2015.

CNHF: basic CNN + Hashing Forest (HF)



1 coding tree ↔ 1 coded feature

Hashing forest ↔ Objective feature space with Objective metrics

(Depth of trees × Objective metrics × Objective function) =

= **Family of face representations***

**1-bit coding "trees" + Hamming distance = CNHL for Hamming embedding*

CNHF: forming and learning

Two-step CNHF learning scheme:

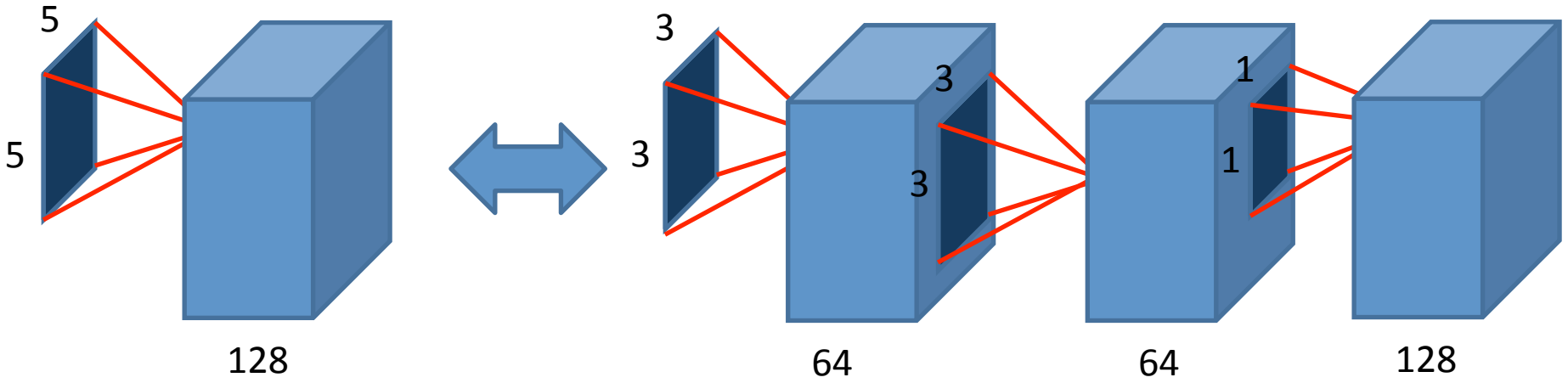
- learn basic CNN for multi-class face identification;
- learn hashing transform for joint face verification and identification.

CNHF forming and learning (more details about CNN implementation):

1. learn the source CNN for multi-class face identification with classes corresponding to persons via back-propagation;
2. transform CNN to the multiple convolution architecture via **substitution of each convolutional layer by the superposition of some (2-4) simpler convolutional layers** (decreasing the number of multiplication operations);
3. train again the transformed CNN for multi-class face identification via back-propagation;
4. **replace the output soft-max layer of transformed CNN by hashing forest** and train the hashing forest.

Steps 2&3: Multiple convolution CNN transformation

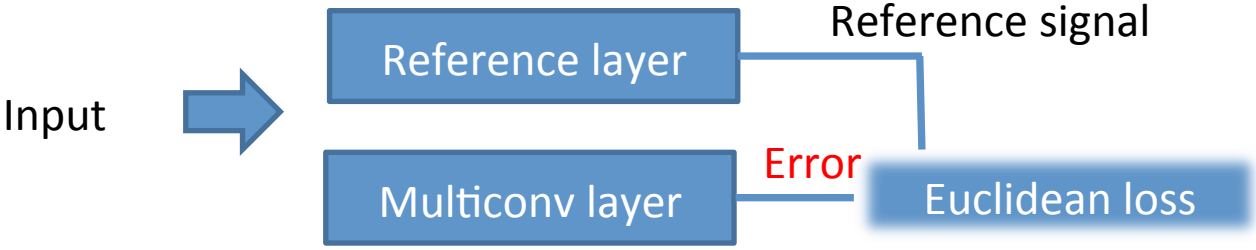
Idea: Replace big convolutional filters by sequences of small filters and 1x1xN filters



Input and output dimensions are equal

Fast relearning (small layer sizes)!
No need to relearn all network!

Transformed layer relearning by back propagation:



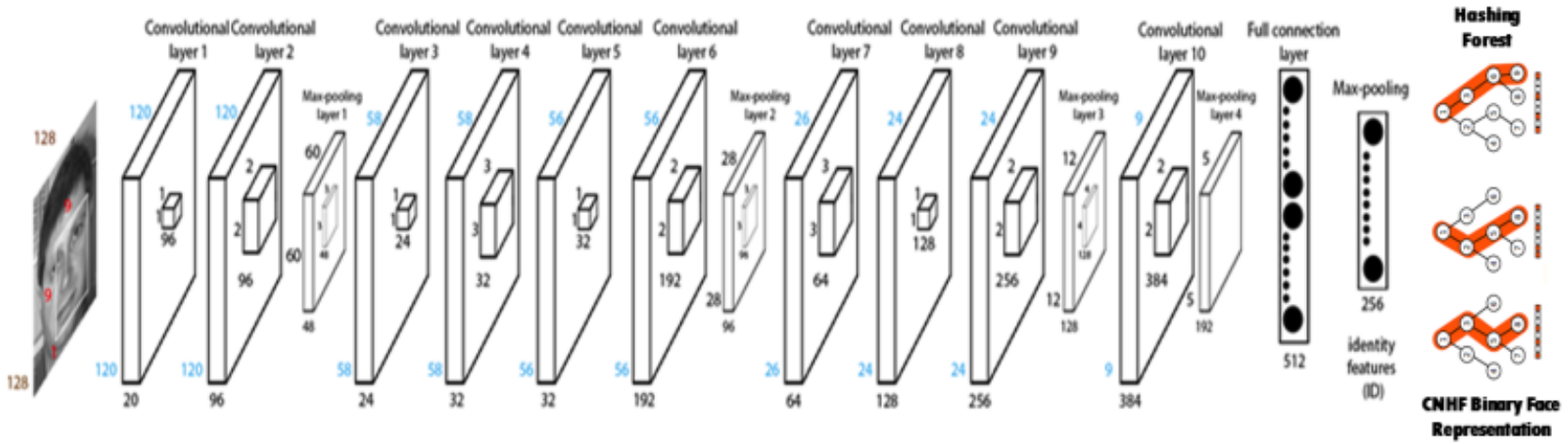
Finally: fine tune the whole network after replacing all convolutional layers

CNHF: multiple convolution architecture and performance

Our CNHF architecture:

- Max-Feature-Map instead of ReLU activation function
- **10 convolutional layers**
- 4 layers of pooling + MFM pooling
- 1 fully connected layer
- **hashing forest**

5 times faster!



CNHF performance:

- 40+ fps with CPU Core i7 (***real-time without GPU***)
- 120+ fps with GPU GeForce GTX 650

FACE CODING: Boosted Hashing Forest (BHF)

Boosted Hashing Forest (BHF):

- *algorithmic structure* of Boosted SSC;
- *binary code structure* of Forest Hashing;
- *support of any objective metrics and objective functions*

Original Boosted SSC

- optimizes 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*.

FACE CODING: Boosted Hashing Forest (BHF)

Main parts of original Boosted SSC:

- *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 features like in AdaBoost.
- *BoostPro algorithm* uses a soft thresholding for gradient-based learning of projections.

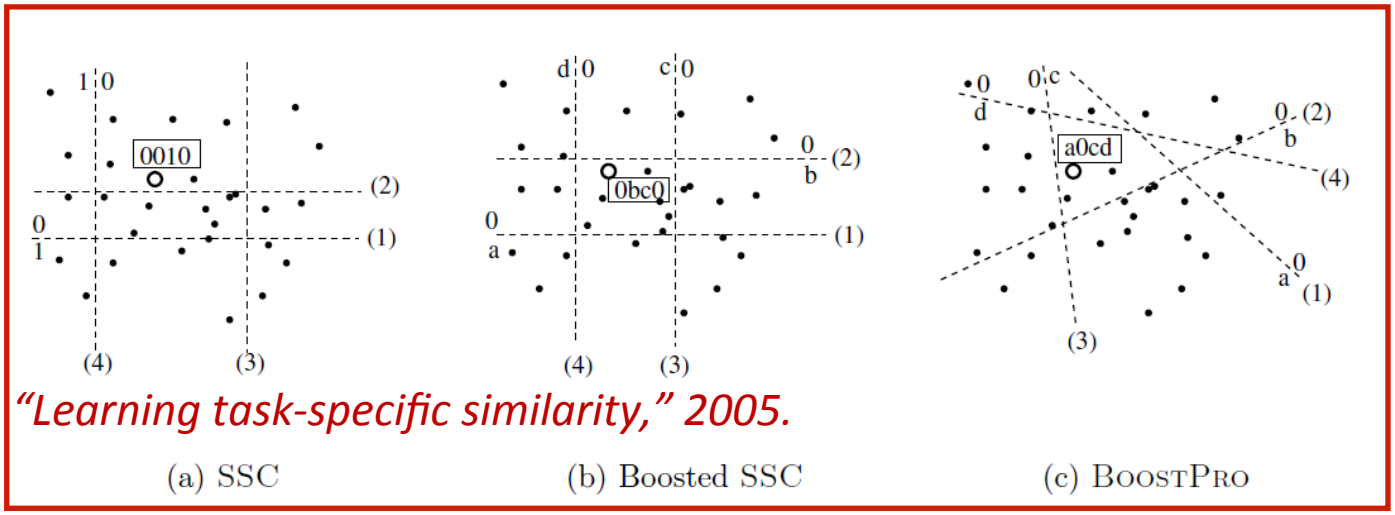


**Figure from G. Shakhnarovich, "Learning task-specific similarity," 2005.*

FACE CODING: Boosted Hashing Forest (BHF)

Proposed BHF w.r.t. original Boosted SSC:

	Original Boosted SSC	**Proposed BHF**
output features	binary features	binary coded non-binary features
data coding structure	binary hashing vector	hashing forest of binary trees
objective function	pairwise similarity function	any given objective function
boosting algorithm	iterative binary vector growing by adding thresholded projections	iterative hashing forest growing with recursive growing of each tree by adding thresholded projections
learning data projections	gradient-based optimization	objective-driven RANSAC search
adaptive reweighting of training pairs	AdaBoost-style reweighting	directly based on the contribution of this pair to the objective function
output metric space (matching metrics)	weighted Hamming space	any given metric space (including Hamming space, if required)

Our BHF implementation for face coding:

- 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 output hash codes for overcoming the limitations of greedy learning.

BHF details: Objective-driven Recurrent Coding

Training set: $X = \{\mathbf{x}_i \in R^m\}_{i=1,\ldots,N}$ (N objects described by m-dimensional feature space).

Mapping X to the n-dimensional binary space (*n-bit coder*):

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

Elementary coder (1-*bit hashing function*):

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

Objective function to be minimized via learning:

$$\mathcal{J}(X, \mathbf{h}^{(n)}) \rightarrow min(\mathbf{h}^{(n)}).$$

Formal statement

BHF details: Objective-driven Recurrent Coding

Greedy Objective-driven Recurrent Coding (Greedy ORC) algorithm sequentially forms the bits of coder in a recurrent manner (Algorithm 1)

Algorithm 1: Greedy ORC

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

Output data:

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

Initialization:

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

Repeat iterations:

k:= k+1;

Learn k-th elementary coder:

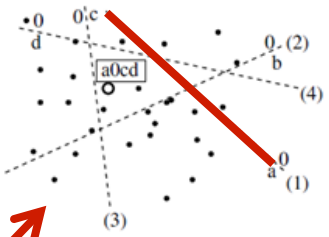
$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)})$:= Learn1BitHash($\mathcal{J}$, X, $\mathbf{h}^{(k-1)}$);

Add k-th elementary coder to the hashing function:

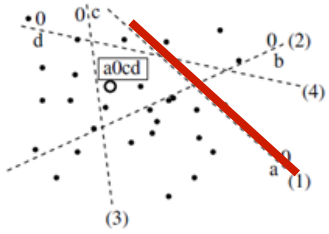
$\mathbf{h}^{(k)}(\mathbf{x})$:= ($\mathbf{h}^{(k-1)}(\mathbf{x})$, $h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)})$);

// concatenation

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



=



1

BHF details: Objective-driven Recurrent Coding

Greedy Objective-driven Recurrent Coding (Greedy ORC) algorithm sequentially forms the bits of coder in a recurrent manner (Algorithm 1)

Algorithm 1: Greedy ORC

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

Output data:

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

Initialization:

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

Repeat iterations:

k:= k+1;

Learn k-th elementary coder:

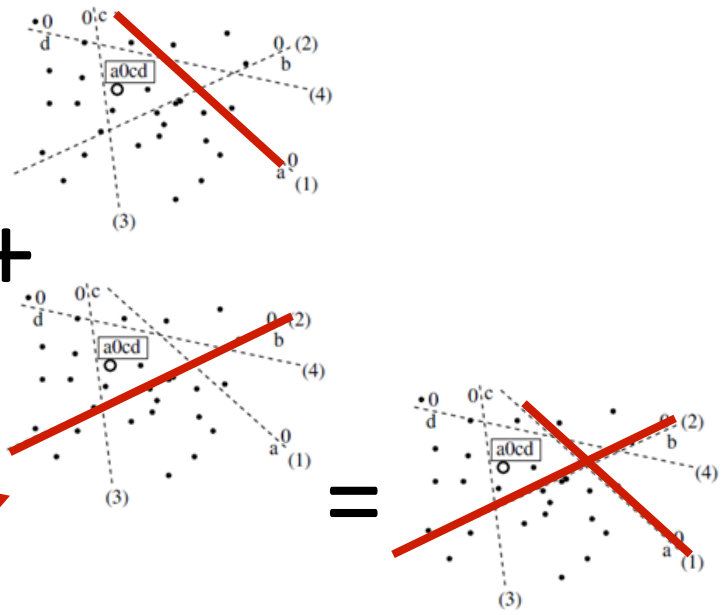
$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)})$:= Learn1BitHash($\mathcal{J}$, X, $\mathbf{h}^{(k-1)}$);

Add k-th elementary coder to the hashing function:

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

// concatenation

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



BHF details: Objective-driven Recurrent Coding

Greedy Objective-driven Recurrent Coding (Greedy ORC) algorithm sequentially forms the bits of coder in a recurrent manner (Algorithm 1)

Algorithm 1: Greedy ORC

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

Output data:

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

Initialization:

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

Repeat iterations:

k:= k+1;

Learn k-th elementary coder:

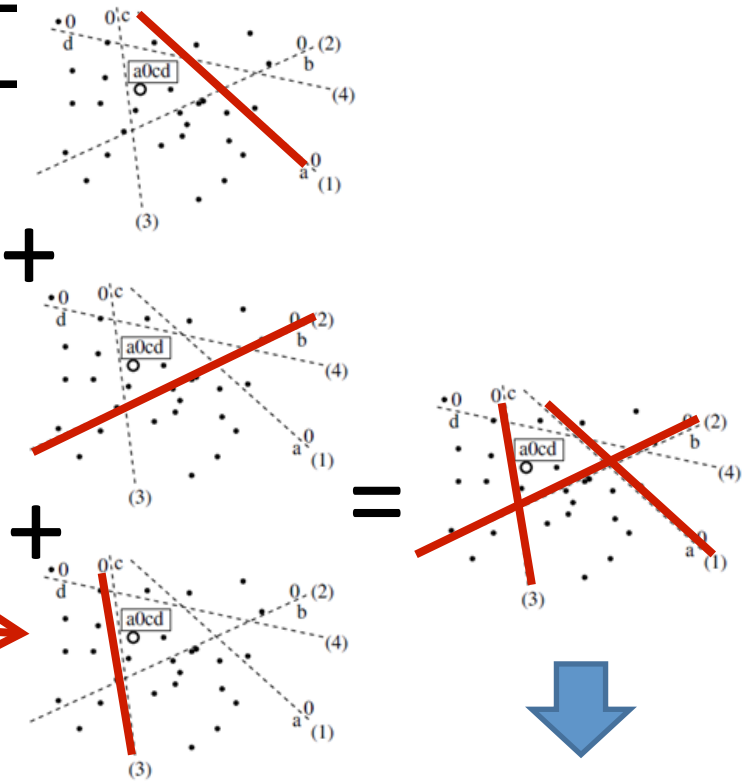
$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)})$:= Learn1BitHash($\mathcal{J}$, X, $\mathbf{h}^{(k-1)}$);

Add k-th elementary coder to the hashing function:

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

// concatenation

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



BHF details: Objective-driven Recurrent Coding

Greedy Objective-driven Recurrent Coding (Greedy ORC) algorithm sequentially forms the bits of coder in a recurrent manner (Algorithm 1)

Algorithm 1: Greedy ORC

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

Output data:

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

Initialization:

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

Repeat iterations:

k:= k+1;

Learn k-th elementary coder:

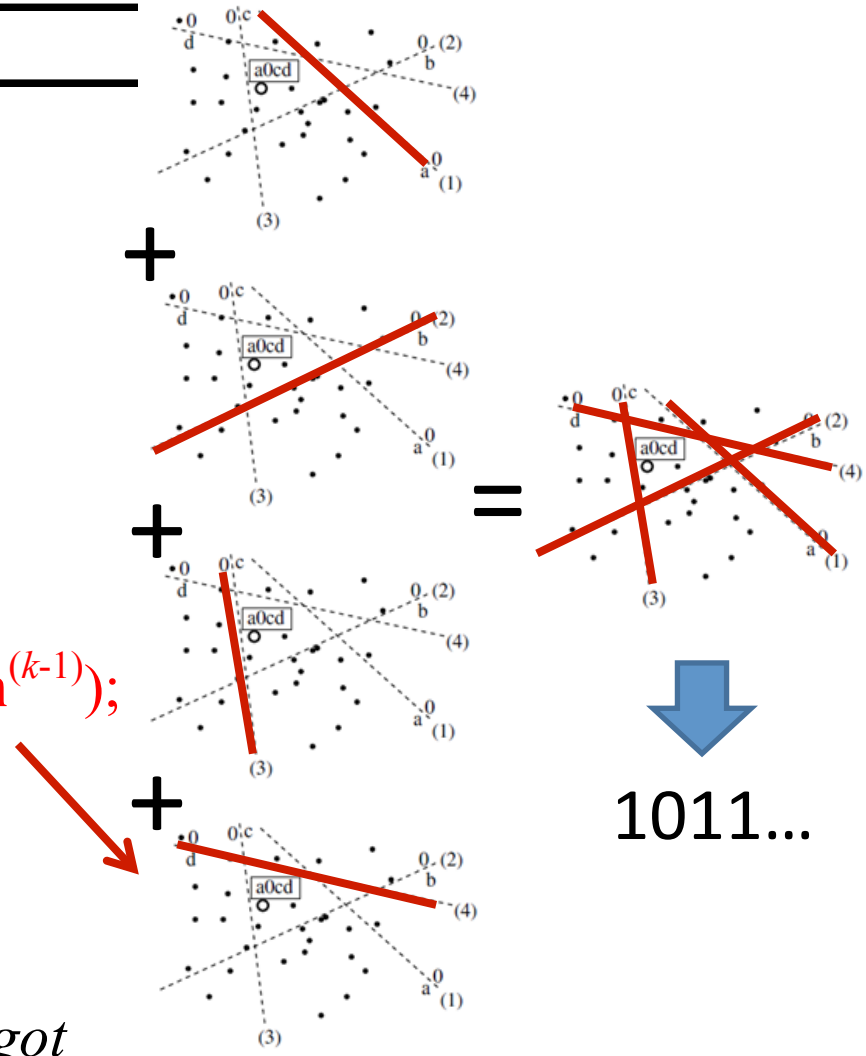
$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)})$:= Learn1BitHash($\mathcal{J}$, X, $\mathbf{h}^{(k-1)}$);

Add k-th elementary coder to the hashing function:

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

// concatenation

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



BHF details: Learning elementary linear classifier via RANSAC algorithm

At the k-th step of coder growing we select the k-th elementary coder

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

Let **H** is a class of **binary linear classifiers** of the form

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

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

In this case objective function depends on $\mathbf{w}$ and t only:

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

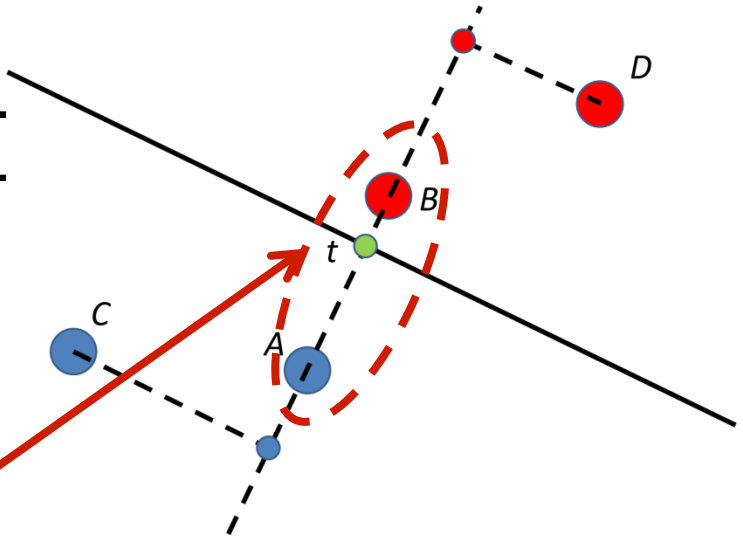
We use RANSAC for finding the projection $\mathbf{w}$ and threshold t, which approximately minimizes the objective function.

Formal statement

BHF details: Learning elementary linear classifier via RANSAC algorithm

Projection determination:

- Iterative random selection of dissimilar pairs in a training set as vectors of hyperplane direction and searching for corresponding optimal threshold;
- taking the projection and threshold, which provide the best value of objective function.



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; $\mathcal{J}_{max}$:=-∞.

Repeat iterations:

k:= k+1;

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

Step 2. Get vector ($\vec{x_i x_j}$) 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:=argmin$_t$ $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t)$.

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

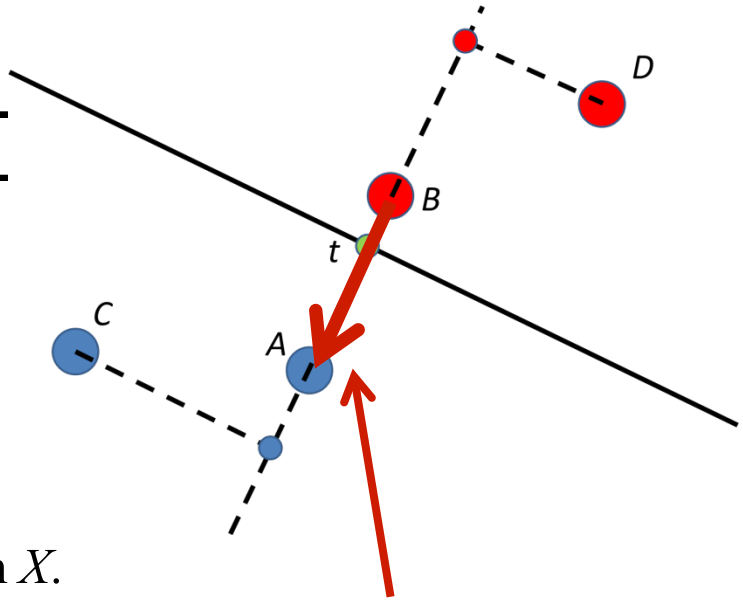
$\mathcal{J}_{max}$:= $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t_k)$; $\mathbf{w}$:= $\mathbf{w}_k$; t:= t_k.

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

BHF details: Learning elementary linear classifier via RANSAC algorithm

Projection determination:

- Iterative random selection of dissimilar pairs in a training set as vectors of hyperplane direction and searching for corresponding optimal threshold;
- taking the projection and threshold, which provide the best value of objective function.



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; $\mathcal{J}_{max}$:=-∞.

Repeat iterations:

k:= k+1;

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

Step 2. Get vector $(\vec{x_i x_j})$ 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:=argmin$_t$ $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t)$.

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

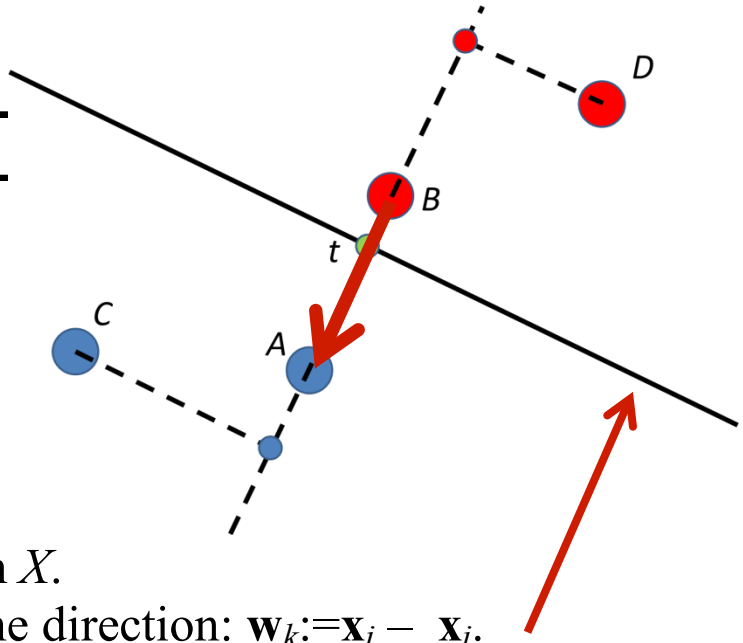
$\mathcal{J}_{max} := \mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t_k)$; $\mathbf{w} := \mathbf{w}_k$; $t := t_k$.

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

BHF details: Learning elementary linear classifier via RANSAC algorithm

Projection determination:

- Iterative random selection of dissimilar pairs in a training set as vectors of hyperplane direction and searching for corresponding optimal threshold;
- taking the projection and threshold, which provide the best value of objective function.



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; $\mathcal{J}_{max}$:=-∞.

Repeat iterations:

k:= k+1;

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

Step 2. Get vector $(\vec{x_i x_j})$ 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:=argmin$_t$ $\mathcal{J}(X, \mathbf{h}^{(k-1)}, \mathbf{w}_k, t)$.

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

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

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

BHF details: Learning elementary linear classifier via RANSAC algorithm

Projection determination:

- random selection of dissimilar pairs in a training set as a vector of hyperplane direction.

Threshold determination:

- The idea of Boosted SSC "ThresholdRate" algorithm is implemented for direct optimization of the global objective function:
  - **Sort projections of objects of training set onto the current projection direction *w***
  - **For each *t* accumulate penalties for separated similar and non-separated dissimilar pairs**

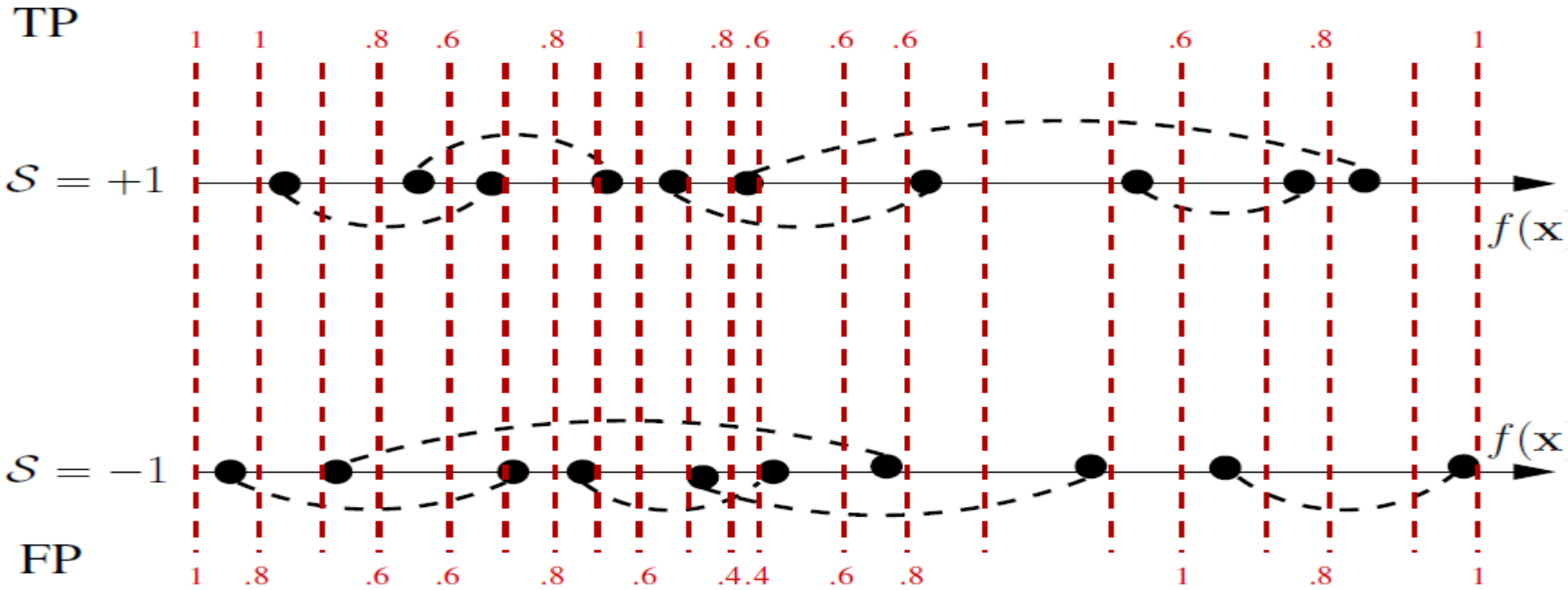


**Figure from G. Shakhnarovich, "Learning task-specific similarity," 2005.*

BHF details: Recursive coding and Trees of coders

Consider the tessellation of X by n-bit coder:

$$\mathbf{X}_B = \{X_{\mathbf{b}}, \mathbf{b}\in\{0,1\}^n\}, X_{\mathbf{b}} = \{\mathbf{x}\in X: \mathbf{h}(\mathbf{x})=\mathbf{b}\}, X = \cup_{\mathbf{b}\in\{0,1\}^n} X_{\mathbf{b}}.$$

Recursive coding is a **dichotomy of training set** with finding the optimized elementary coder for each subset **at each level of tessellation**:

1 bit

Elementary recursive coder for k-th bit:

$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = h(\mathbf{w}(\mathbf{h}^{(k-1)}(\mathbf{x})), t(\mathbf{h}^{(k-1)}(\mathbf{x})), \mathbf{x}),$$
$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = \text{Learn1BitHash}(\mathcal{J}, X, \mathbf{h}^{(k-1)}) =$$
$$= \{\text{Learn1ProjectionHash}(\mathcal{J}, X(\mathbf{h}^{(k-1)},\mathbf{b}), \mathbf{h}^{(k-1)}), \mathbf{b}\in\{0,1\}^{(k-1)}\}.$$

//set of $2^{(k-1)}$ thresholded projections formed by RANSAC

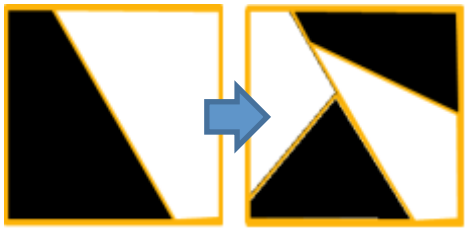
Tree of binary coders: recursive n-bit coder $\mathbf{h}^{(n)}(\mathbf{x})$.

BHF details: Recursive coding and Trees of coders

Consider the tessellation of X by n-bit coder:

$$\mathbf{X}_B = \{X_\mathbf{b}, \mathbf{b}\in\{0,1\}^n\}, X_\mathbf{b} = \{\mathbf{x}\in X: \mathbf{h}(\mathbf{x})=\mathbf{b}\}, X = \cup_{\mathbf{b}\in\{0,1\}^n} X_\mathbf{b}.$$

Recursive coding is a dichotomy of training set with finding the optimized elementary coder for each subset at each level of tessellation:



1 bit 2 bits

Elementary recursive coder for k-th bit:

$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = h(\mathbf{w}(\mathbf{h}^{(k-1)}(\mathbf{x})), t(\mathbf{h}^{(k-1)}(\mathbf{x})), \mathbf{x}),$$
$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = \text{Learn1BitHash}(\mathcal{J}, X, \mathbf{h}^{(k-1)}) =$$
$$= \{\text{Learn1ProjectionHash}(\mathcal{J}, X(\mathbf{h}^{(k-1)},\mathbf{b}), \mathbf{h}^{(k-1)}), \mathbf{b}\in\{0,1\}^{(k-1)}\}.$$

//set of $2^{(k-1)}$ thresholded projections formed by RANSAC

Tree of binary coders: recursive n-bit coder $\mathbf{h}^{(n)}(\mathbf{x})$.

BHF details: Recursive coding and Trees of coders

Consider the tessellation of X by n-bit coder:

$$\mathbf{X}_B = \{X_{\mathbf{b}}, \mathbf{b}\in\{0,1\}^n\}, X_{\mathbf{b}} = \{\mathbf{x}\in X: \mathbf{h}(\mathbf{x})=\mathbf{b}\}, X = \cup_{\mathbf{b}\in\{0,1\}^n} X_{\mathbf{b}}.$$

Recursive coding is a dichotomy of training set with finding the optimized elementary coder for each subset at each level of tessellation:

Elementary recursive coder for k-th bit:

$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = h(\mathbf{w}(\mathbf{h}^{(k-1)}(\mathbf{x})), t(\mathbf{h}^{(k-1)}(\mathbf{x})), \mathbf{x}),$$
$$h^{(k)}(\mathbf{x}, \mathbf{h}^{(k-1)}) = \text{Learn1BitHash}(\mathcal{J}, X, \mathbf{h}^{(k-1)}) =$$
$$= \{\text{Learn1ProjectionHash}(\mathcal{J}, X(\mathbf{h}^{(k-1)},\mathbf{b}), \mathbf{h}^{(k-1)}), \mathbf{b}\in\{0,1\}^{(k-1)}\}.$$

//set of $2^{(k-1)}$ thresholded projections formed by RANSAC

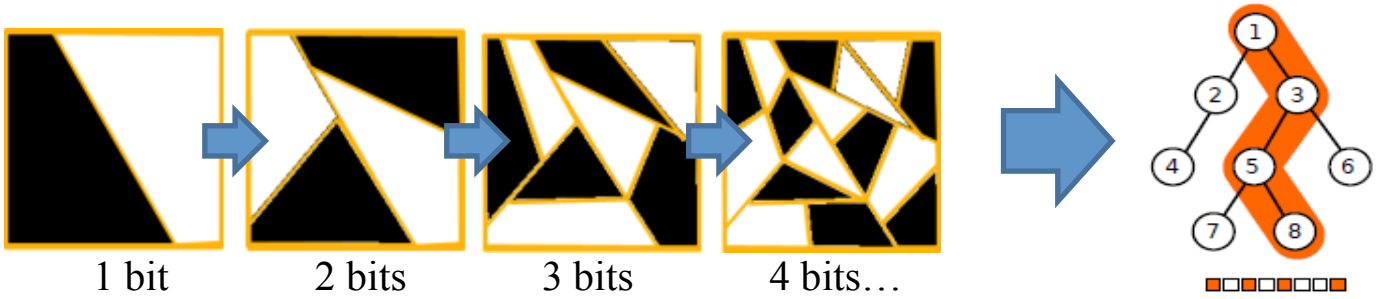
Tree of binary coders: recursive n-bit coder $\mathbf{h}^{(n)}(\mathbf{x})$.

BHF details: Recursive coding and Trees of coders

Consider the tessellation of X by n-bit coder:

$$\mathbf{X}_B = \{X_\mathbf{b}, \mathbf{b}\in\{0,1\}^n\},\ X_\mathbf{b} = \{\mathbf{x}\in X\colon \mathbf{h}(\mathbf{x})=\mathbf{b}\},\ X = \cup_{\mathbf{b}\in\{0,1\}^n} X_\mathbf{b}.$$

Recursive coding is a dichotomic splitting of training set with finding the optimized elementary coder for each subset at each level of tessellation:



Elementary recursive coder for k-th bit:

$$h^{(k)}(\mathbf{x},\ \mathbf{h}^{(k-1)}) = h(\mathbf{w}(\mathbf{h}^{(k-1)}(\mathbf{x})),\ t(\mathbf{h}^{(k-1)}(\mathbf{x})),\ \mathbf{x}),$$
$$h^{(k)}(\mathbf{x},\ \mathbf{h}^{(k-1)}) = \text{Learn1BitHash}(\mathcal{J},\ X,\ \mathbf{h}^{(k-1)}) =$$
$$= \{\text{Learn1ProjectionHash}(\mathcal{J},\ X(\mathbf{h}^{(k-1)},\mathbf{b}),\ \mathbf{h}^{(k-1)}),\ \mathbf{b}\in\{0,1\}^{(k-1)}\}.$$

//set of $2^{(k-1)}$ thresholded projections formed by RANSAC

Tree of binary coders: recursive n-bit coder $\mathbf{h}^{(n)}(\mathbf{x})$.

BHF details: Boosted Hashing Forest

Boosted Hashing Forest (BHF) is formed via the boosting of hashing trees with optimization of joint objective function for all trees (Algorithm 3).

Algorithm 3: 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:

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

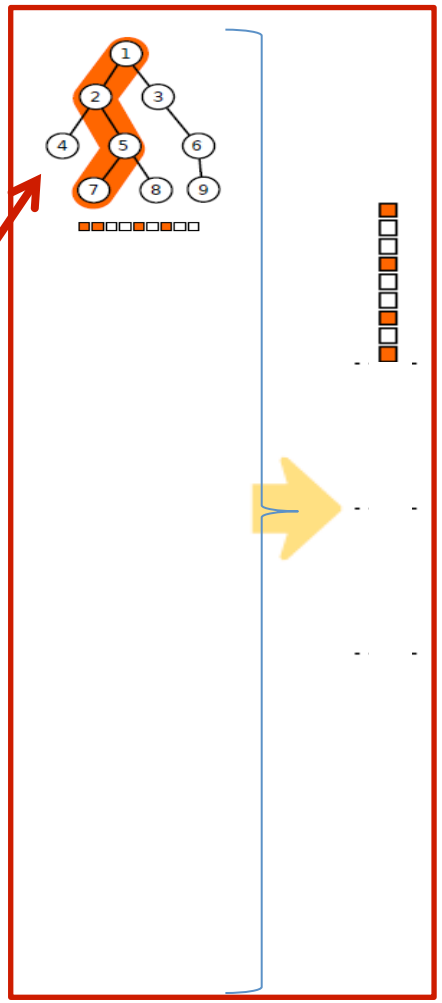
Learn l-th coding tree:

$\mathbf{h}^{[l,l]}$:= 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*



BHF parameters and notation:

$n_{ORC} = p$ is a depth of coding tree; $n_{BHF} = n/p$ is a number of trees;

$\mathbf{h}^{[1,l]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$, $\mathbf{h}^{[1,l-1]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp-p)}(\mathbf{x}))$, $\mathbf{h}^{[l,l]} = (h^{(lp-p+1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$.

BHF details: Boosted Hashing Forest

Boosted Hashing Forest (BHF) is formed via the boosting of hashing trees with optimization of joint objective function for all trees (Algorithm 3).

Algorithm 3: 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:

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

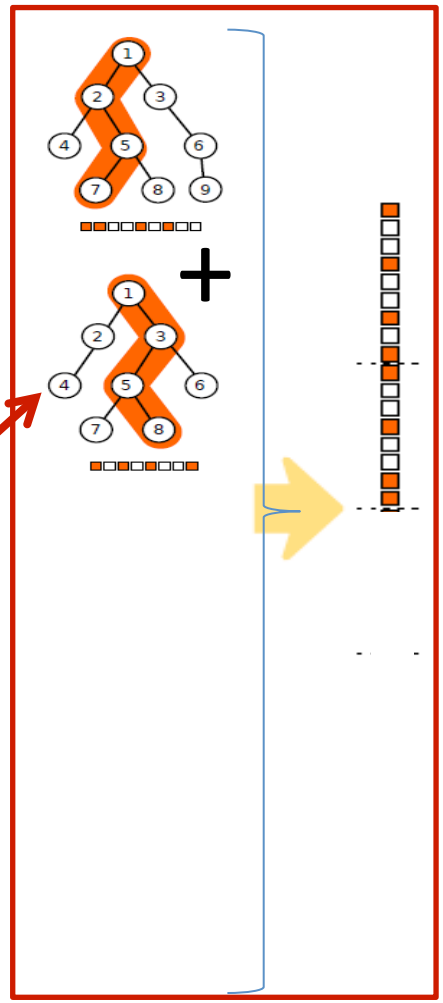
Learn l-th coding tree:

$\mathbf{h}^{[l,l]}$:= 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*



BHF parameters and notation:

$n_{ORC} = p$ is a depth of coding tree; $n_{BHF} = n/p$ is a number of trees;

$\mathbf{h}^{[1,l]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$, $\mathbf{h}^{[1,l-1]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp-p)}(\mathbf{x}))$, $\mathbf{h}^{[l,l]} = (h^{(lp-p+1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$.

BHF details: Boosted Hashing Forest

Boosted Hashing Forest (BHF) is formed via the boosting of hashing trees with optimization of joint objective function for all trees (Algorithm 3).

Algorithm 3: 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:

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

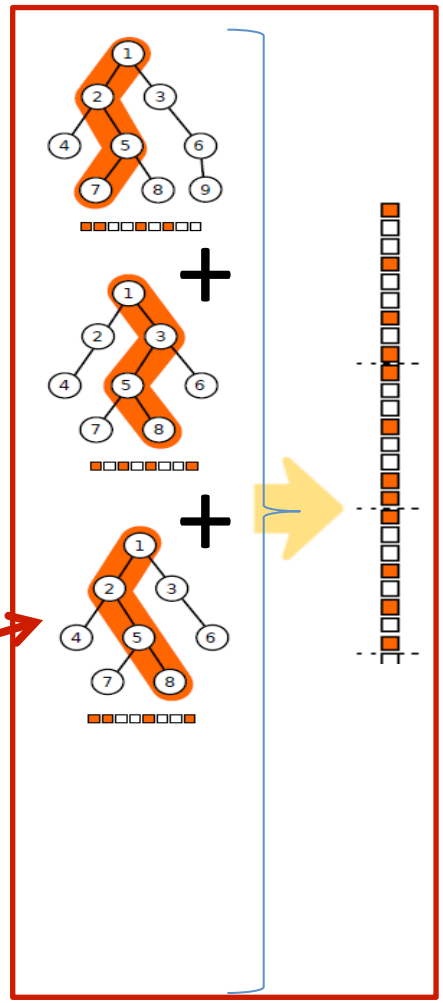
Learn l-th coding tree:

$\mathbf{h}^{[l,l]}$:= 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*



BHF parameters and notation:

$n_{ORC} = p$ is a depth of coding tree; $n_{BHF} = n/p$ is a number of trees;

$\mathbf{h}^{[1,l]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$, $\mathbf{h}^{[1,l-1]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp-p)}(\mathbf{x}))$, $\mathbf{h}^{[l,l]} = (h^{(lp-p+1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$.

BHF details: Boosted Hashing Forest

Boosted Hashing Forest (BHF) is formed via the boosting of hashing trees with optimization of joint objective function for all trees (Algorithm 3).

Algorithm 3: 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:

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

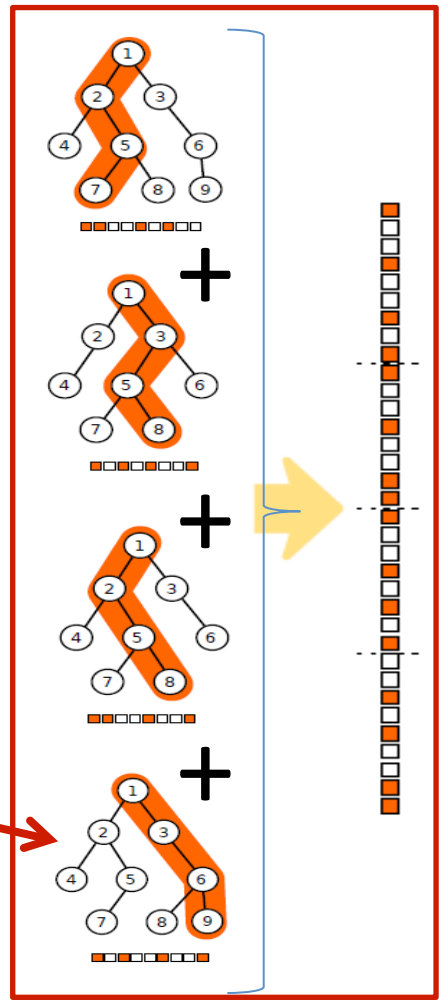
Learn l-th coding tree:

$\mathbf{h}^{[l,l]}$:= 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*



BHF parameters and notation:

$n_{ORC} = p$ is a depth of coding tree; $n_{BHF} = n/p$ is a number of trees;

$\mathbf{h}^{[1,l]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$, $\mathbf{h}^{[1,l-1]} = (h^{(1)}(\mathbf{x}),\ldots,h^{(lp-p)}(\mathbf{x}))$, $\mathbf{h}^{[l,l]} = (h^{(lp-p+1)}(\mathbf{x}),\ldots,h^{(lp)}(\mathbf{x}))$.

BHF details: Coding metric space

Metric space $(Y,\ d_Y: Y\times Y \to R^+)$ is *n-bit binary coded*, if

- the each $y \in Y$ corresponds to unique $\mathbf{b} \in \{0,1\}^n$,
- two decoding functions are given:
  - *feature decoder* $f_y(\mathbf{b})$: $\{0,1\}^n \to Y$
  - *distance decoder* $f_d(\mathbf{b}_1,\mathbf{b}_2)$: $\{0,1\}^n \times \{0,1\}^n \to R^+$,

$$f_d(\mathbf{b}_1,\mathbf{b}_2) = d_Y(f_y(\mathbf{b}_1), f_y(\mathbf{b}_2)).$$

Metric space coding:

- optimization of *Distance-based objective function* (DBOF)

$$\mathcal{J}(X,\mathbf{h}) \to min(\mathbf{h}) \Leftrightarrow \mathcal{J}(D_Y) \to min(D_Y), \quad (7)$$

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

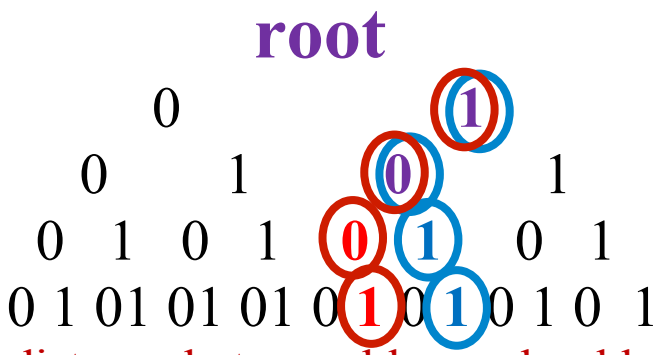
Such objective function depends on the set of coded distances d_{ij} only.

BHF implementation: Forest code matching via Sum of Search Index Distances

We match tree codes via *Search Index Distance (SID)* – geodesic distance between binary codes as corresponding leaves on the coding tree:

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

Example. Let p=4, $\mathbf{b}_1 = (1,0,1,1)$ and $\mathbf{b}_2 = (1,0,0,1)$.
Corresponding vertices on the coding tree are marked as blue ($\mathbf{b}_1$), red ($\mathbf{b}_2$) and purple (joint):



The distance between blue and red leaves is 4 (2 levels up + 2 levels down):

$d_T((1,0,1,1),(1,0,0,1)) = 2 (1 – (1 – 0) +$
$1 – (1 – 0)\times(1 – 0) +$
$1 – (1 – 0)\times(1 – 0) \times(1 – 1) +$
$1 – (1 – 0)\times(1 – 0) \times(1 – 1)\times(1 – 1)) = 2 (0 + 0 + 1 + 1) = 4.$

End of example.

Finally, **we match forest codes** via *Sum of Search Index Distances* (*SSID*) between trees:

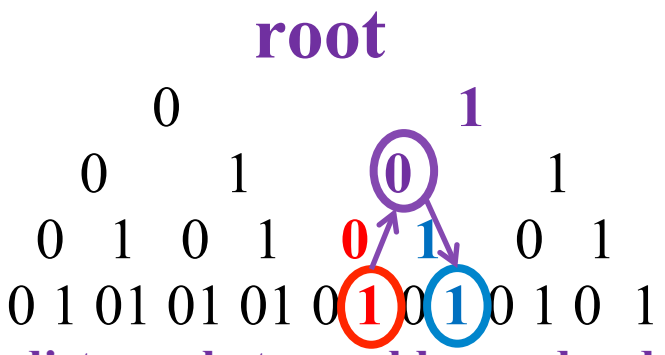
$$d_{ij} = \sum_{l=1,\dots,q} f_{dT}(\mathbf{h}^{[l,l]}(\mathbf{x}_i), \mathbf{h}^{[l,l]}(\mathbf{x}_j)).$$

BHF implementation: Forest code matching via Sum of Search Index Distances

We match tree codes via *Search Index Distance (SID)* – geodesic distance between binary codes as corresponding leaves on the coding tree:

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

Example. Let p=4, $\mathbf{b}_1$ = (1,0,1,1) and $\mathbf{b}_2$ = (1,0,0,1).
Corresponding vertices on the coding tree are marked as blue ($\mathbf{b}_1$), red ($\mathbf{b}_2$) and purple (joint):



The distance between blue and red leaves is 4 (2 levels up + 2 levels down):

$d_T((1,0,1,1),(1,0,0,1)) = 2 (1 – (1 – 0) +$
$1 – (1 – 0)\times(1 – 0) +$
$1 – (1 – 0)\times(1 – 0) \times(1 – 1) +$
$1 – (1 – 0)\times(1 – 0) \times(1 – 1)\times(1 – 1)) = 2 (0 + 0 + 1 + 1) = 4.$

End of example.

Finally, **we match forest codes** via *Sum of Search Index Distances* (*SSID*) between trees:

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

BHF implementation: Objective function for face verification

Similarity function s describes positive (authentic) and negative (imposter) pairs:

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

Goal distance for k-bit binary code:

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

***Distance supervision objective function*:**

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

where v_{ij} – different weights for authentic and imposter pairs.

Such objective function controls the verification performance (FAR-FRR).

BHF implementation: Objective function for face identification

Identification task requires controlling both distances and ordering of distances.

Let for the query $\mathbf{h}(\mathbf{x}_k)$:

$d^1{}_k = \max_l\{d_{kl}: s_{kl} = 1\}$ – distance to the most far authentic;

$d^0{}_k = \min_l\{d_{kl}: s_{kl} = 0\}$ – distance to the closest imposter

Ordering error e_{ij} for a pair $(\mathbf{x}_i,\mathbf{x}_j)$:

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

(occurs if imposter is closer than authentic or authentic is more far than imposter)

***Distance order supervision objective function*:**

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

penalizes the difference between d_{ij} and objective distance g_{ij},
but only in case that the ordering error e_{ij} occurs for this pair.

Such objective function (12) controls the face identification characteristics (CMC).

BHF implementation: Objective function for face verification and identification

***Distance and Distance order supervision objective function*:**

$$\mathcal{J}(D_Y) = \alpha\, \mathcal{J}_{Dist}(D_Y) + (1-\alpha)\, \mathcal{J}_{Ord}(D_Y) =$$
$$= \sum_{i=1,\ldots,N} \sum_{j=1,\ldots,N} v_{ij} (d_{ij} - g_{ij})^2 (e_{ij} + \alpha(1 - e_{ij})) \rightarrow$$
$$\rightarrow min(D_Y = \{d_{ij}\}_{i,j=1,\ldots,N}),$$

where $\alpha \in [0,1]$ is a tuning parameter for balancing distance and distance order influence.

Such objective function controls both the face verification and face identification characteristics.

BHF implementation: semi-heuristic tricks

Modification of goal distance:

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

where $m^{(k\text{-}1)}{}_1$ and $\sigma^{(k\text{-}1)}{}_1$ are the mean value and standard deviation of authentic coded distances.

Goal distance excludes the penalizing of imposter pairs, which could not be treated as authentic.

For possible questions

BHF implementation: semi-heuristic tricks

We use the **adaptive weighting of pairs** at each k-th step of boosting:

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

$$a^{(k)} = \sum_{i=1,\ldots,N} \sum_{j=1,\ldots,N} s_{ij} \, (d_{ij} - g_{ij})^2 \, (e_{ij} + \alpha(1 - e_{ij})),$$
$$b^{(k)} = \sum_{i=1,\ldots,N} \sum_{j=1,\ldots,N} (1 - s_{ij}) \, (d_{ij} - g_{ij})^2 \, (e_{ij} + \alpha(1 - e_{ij})),$$

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.

For possible questions

BHF implementation: semi-heuristic tricks

Selection and processing of subvectors of the input feature vector:
We split the input m-dimensional feature vector to the set of independently coded subvectors with fixed sizes from the set $\mathbf{m} = \{m_{\min},\ldots,m_{\max}\}$. At the each step of boosting we get the subvector with corresponding BHF elementary coder providing the best contribution to the objective function.

Creation of ensemble of independent hash codes:
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.

For possible questions

BHF implementation: tuning parameters

Set of implemented BHF free parameters:

- $\mathbf{m} = \{m_{\min},\ldots,m_{\max}\}$ – set of sizes for independently coded input subvectors;
- n_{ORC} – depth of hashing trees;
- n_{BHF} – number of trees in the hashing forests;
- k_{RANSAC} – number of RANSAC iterations for each projection;
- α – objective function tuning parameter for balancing distance and distance order influence;
- γ – tuning parameter, which gives slightly larger weights to authentic pairs;

In general, coded metrics is a free parameter of our approach too, but in this paper we use the *Sum of Search Index Distances (SSID)* only.

EXPERIMENTS

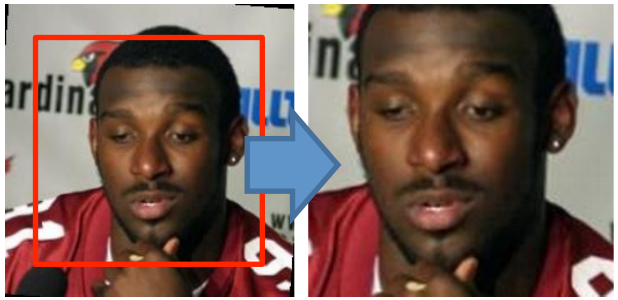
Content of experimental part

- Methodology: learning and testing CNHF;
- Hamming embedding: CNHL vs. CNN;
- Hamming embedding: BHF vs. Boosted SSC;
- Proposed BHF w.r.t. original Boosted SSC;
- CNHF performance w.r.t. depth of coding trees;
- CNHL and CNHF vs. best methods on LFW.

EXPERIMENTS: learning and testing CNHF

CNN Learning

- **training dataset:** CASIA-WebFace;
- **face alignment:**
  - rotation of eye points to horizontal position with fixed eye-to-eye distance
  - crop to 128x128 size;
- **training framework:** open source Caffe (http://caffe.berkeleyvision.org/);
- **training technique:** training for multi-class face identification in the manner of



Y. Sun at al., "Deep learning face representation from predicting 10,000 classes," 2014.

X. Wu, "Learning robust deep face representation," 2015.

EXPERIMENTS: learning and testing CNHF

Hashing forest learning

- **training dataset:** 1000 authentic and 999000 imposter pairs of Faces in the Wild images (not from the testing LFW set);
- **formed family of CNHF coders:**
  - *Hamming embedding coders* 2000×1 bit (250 byte), 200×1 bit (25 byte) and 32×1 bit (4 byte) of size;
  - *Hashing forest coders* – 2000 trees with 2-7 bits depth (0.5 – 1.75 Kbyte of size);
- **BHF parameter settings:**
  - common settings for all CNHFs:
    - $\mathbf{m} = \{8, 16, 32\}$, $k_{RANSAC} = 500$, $\alpha = 0.25$, $\gamma = 1.1$;
  - individual settings for number of trees in HF (determined experimentally based on the analysis of the speed of identification rate growing w.r.t. number of code bits in the hashing process):
    - n_{BHF}=200 for CNN+BHF-200×1,
    - n_{BHF}=500 for CNN+BHF-2000×1
    - n_{BHF}=100 for CNHF-2000×7.

EXPERIMENTS: learning and testing CNHF

CNHF and CNN evaluation

- **testing dataset:** Labeled Faces in the Wild (LFW);
- **face alignment:** all the LFW images are processed and normalized to 128x128 as in
  G.-B. Huang at al., "Learning to align from scratch," 2012;
- **verification test: accuracy** by the standard LFW unrestricted with outside labeled data protocol and **ROC**;
- **identification tests: CMC** and **rank-1** following the methodology
  L. Best-Rowden at al., "Unconstrained face recognition: Identifying a person of interest from a media collection," 2014.

EXPERIMENTS: Hamming embedding*

**CNHF degrades to CNHL*

Hamming embedding: CNHL vs. CNN

- **CNN face representation:** vector of activations of 256 top hidden layer neurons;
- **CNN matching metrics:** cosine similarity (CNN+CS) and L2-distance (CNN+L2);
- **CNHL face representations:** 2000 and 200 bit-coders trained by BHF (CNN+BHF-2000×1 and CNN+BHF-200×1);
- **CNHL matching metrics:** Hamming distance.

Table 1. Verification accuracy on LFW, code size and matching speed of CNN and CNHL

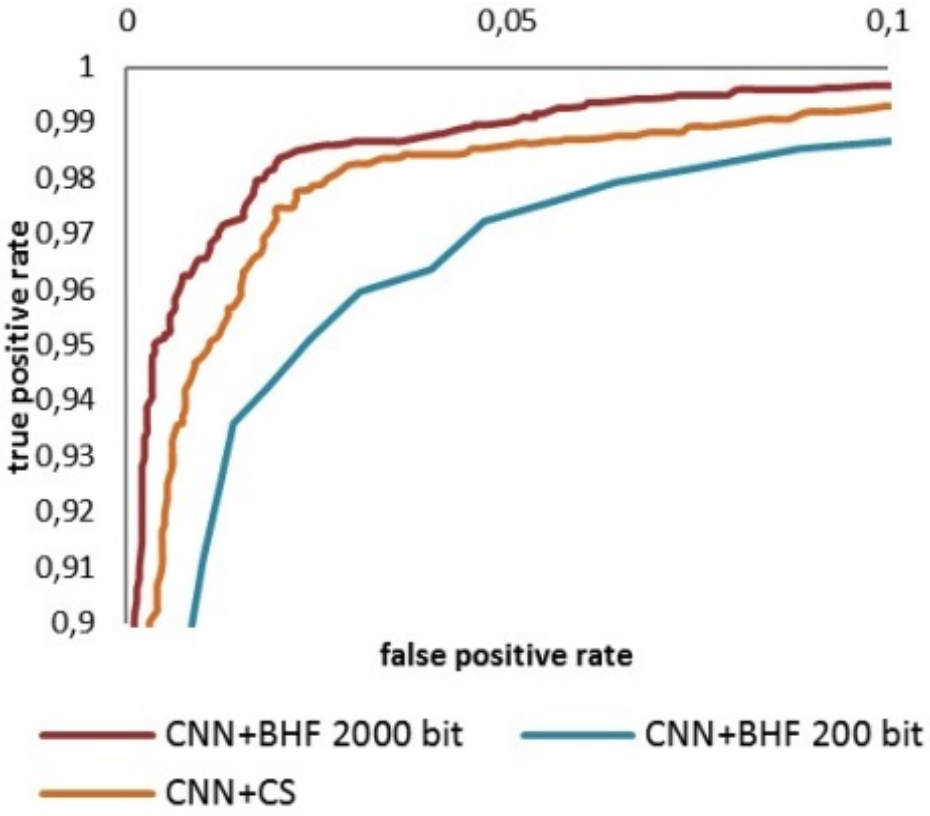
Solution	Accuracy on LFW	Template size	Matches in sec
CNN+L2	0.947	8192 bit	2713222
CNN+BHF-200×1	0.963	**200 bit**	**194986071**
CNN+CS	0.975	8192 bit	2787632
CNN+BHF-2000×1	**0.9814**	**2000 bit**	**27855153**

Our **200x1-bit face coder** provides **40-times smaller template size** and **70-times faster matching** with only **1% decreasing of accuracy** relative to basic CNN (**96.3% on LFW**)!

EXPERIMENTS: Hamming embedding*

**CNHF degrades to CNHL*

Hamming embedding: CNHL vs. CNN



Results:

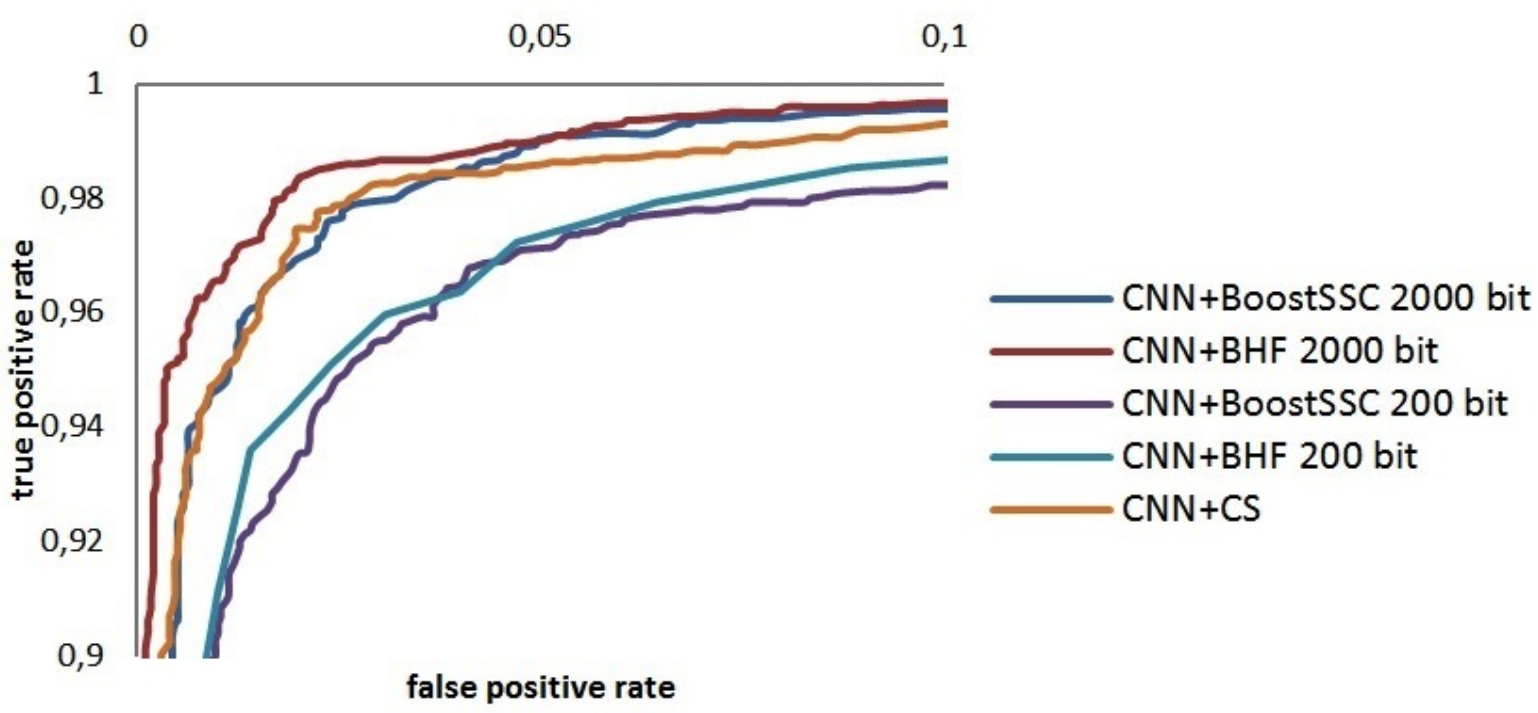
- Our solution CNN+BHF-2000×1 achieves 98.14% on LFW, which outperforms all other solutions based on this CNN.
- Our 25-byte solution CNN+BHF-200×1 outperforms CNN+L2.
- Table 1 additionally demonstrates the gain in template size and matching speed.

EXPERIMENTS: Hamming embedding*

**CNHF degrades to CNHL*

Hamming embedding: Proposed BHF vs. Boosted SSC

Verification: ROC



Results:

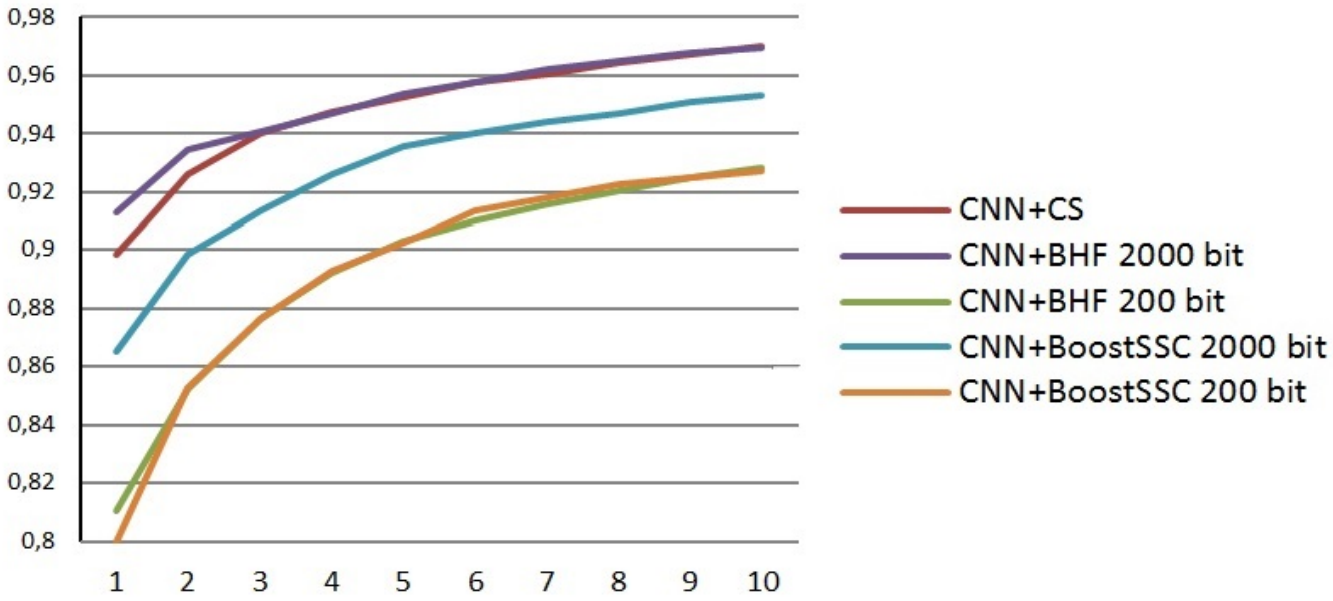
- ROC graph for CNN+BHF is monotonously better than for CNN+BoostSSC;

EXPERIMENTS: Hamming embedding*

**CNHF degrades to CNHL*

Hamming embedding: Proposed BHF vs. Boosted SSC

Identification: CMC



Results:

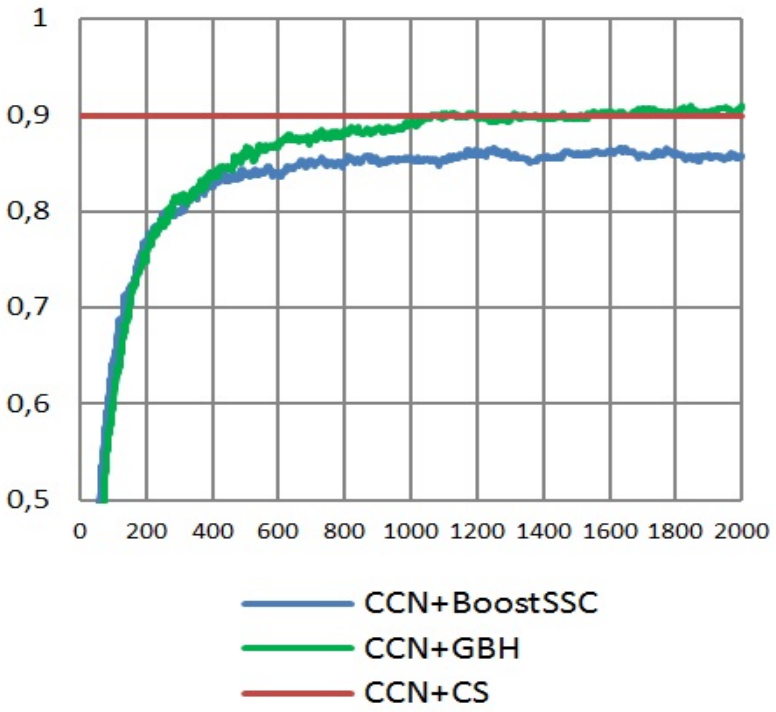
- ROC graph for CNN+BHF is monotonously better than for CNN+BoostSSC;
- CMC graphs (ranks 1-10) BHF outperforms BoostSSC (CNN+BHF-2000×1 outperforms even CNN+CS).

EXPERIMENTS: Hamming embedding*

**CNHF degrades to CNHL*

Hamming embedding: Proposed BHF vs. Boosted SSC

Identification: Rank-1



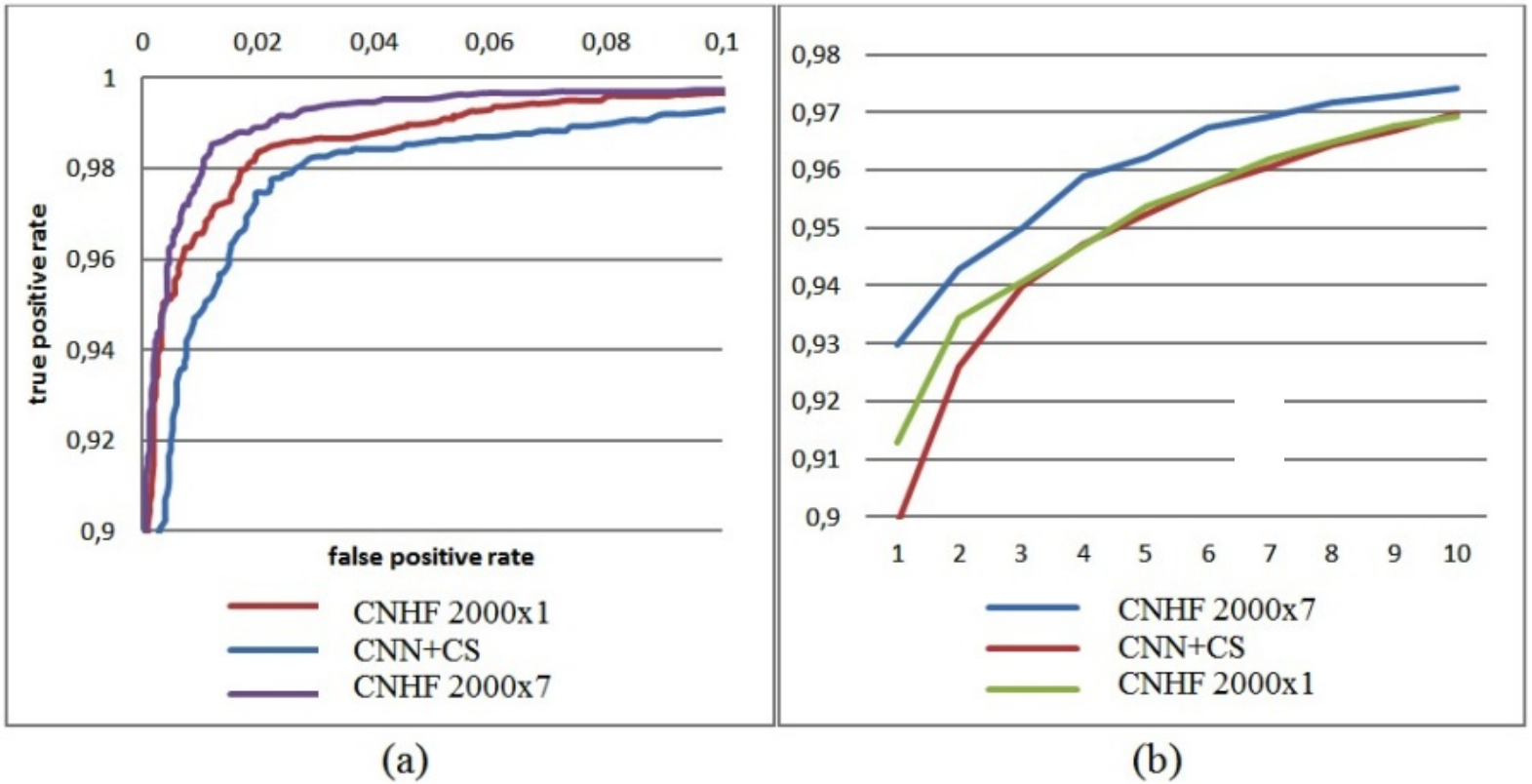
Results:

- ROC graph for CNN+BHF is monotonously better than for CNN+BoostSSC;
- CMC graphs (ranks 1-10) BHF outperforms BoostSSC (CNN+BHF-2000×1 outperforms even CNN+CS).
- **BHF outperforms Boosted SSC** in identification (rank-1) on LFW for all binary template sizes (**outperforms even CNN+CS**);
- maximal rank-1 is 0.91 for BHF-2000×1 and 0.865 for BoostSSC-2000×1;

EXPERIMENTS: Hashing Forest vs. Hashing Layer

CNHF: 7-bit trees w.r.t. 1-bit Hamming embedding and CNN

CNHF with N output features coded each by M-bit coding trees = CNHF-N×M



(a) (b)

ROC (a) and CMC (b) curves for CNN+CS, CNHF-2000×1 and CNHF-2000×7.

Results:

- CNHF-2000×7 achieves 98.59% on LFW.
- CNHF-2000×7 is 93% rank-1 on LFW relative to 89.9% rank-1 for CNN+CS.
- CNHF-2000×7 outperforms CNHF-2000×1 and basic CNN+CS both in verification (ROC) and in identification (CMC).

SUMMARY of EXPERIMENTS: CNHF vs. CNHL, BHF vs. Bossted SSC, CNHL two-step learning vs. one-step learning

Notation: CNHF with N output features coded each by M-bit coding trees = CNHF-NxM

1) For all characteristics (accuracy, ROC, rank-1, CMC):

CNN+BHF-2000×7 > CNN+BHF-2000×1 > CNN+CS > CNN+L2

2) For all characteristics (accuracy, ROC, rank-1, CMC) and all N:

CNN+BHF×1 > CNN+BoostedSSC

3) If accuracy > 96% on LFW (*other characteristics are unknown*) :

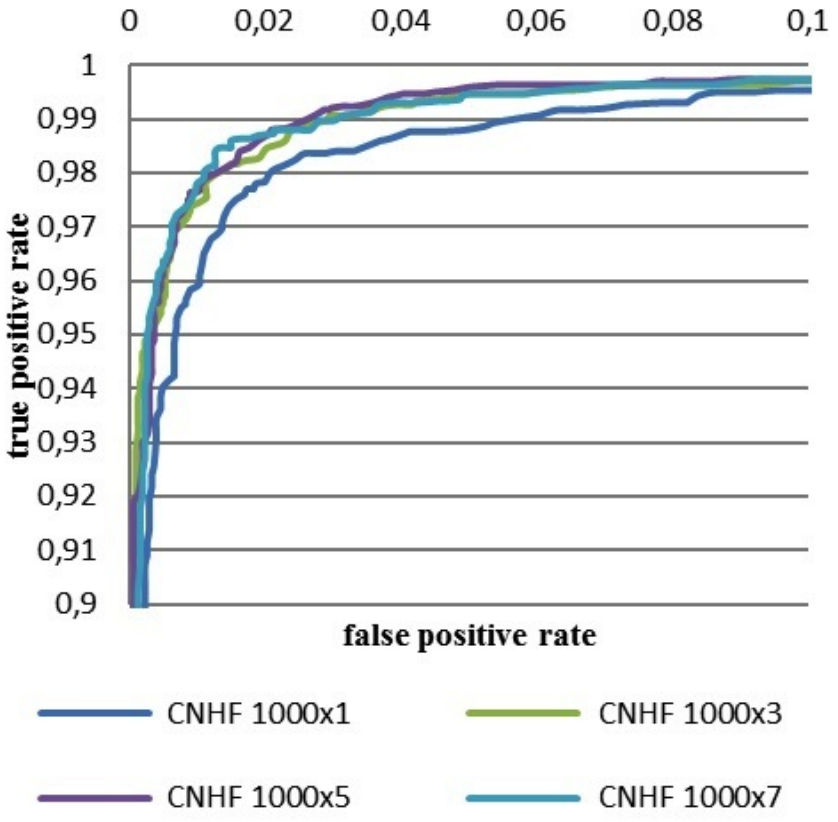
CNN+BHF-200x1 > CNHL-1000x1 (H. Fan at al., 2014)

(HL learned after CNN) is 5 times more compact than (HL+CNN learned together)

Main conclusion: adding hashing forest on the top of CNN allows both generating the compact binary face representations and increasing the face verification and especially identification rates.

EXPERIMENTS: CNHF performance w.r.t. depth of trees

ROC curves for CNHF with different depth of coding trees:



Results:

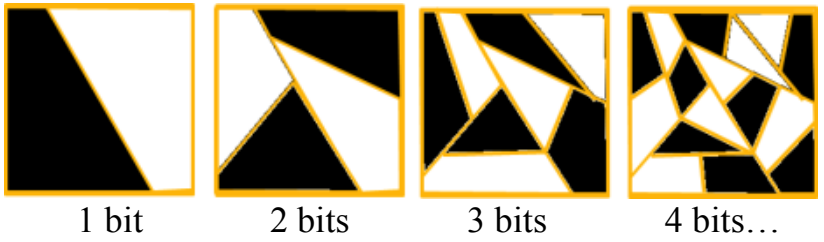
- performance grows with growing depth of trees;
- forest with 7-bit coding trees is the best by ROC…

…but 5-bit depth solution is *very close*.

Supposed reason:

- limited amount of training set for forest hashing.

(too small number of authentic pairs in the each cell of space tessellation)



It's a topic for further research!

EXPERIMENTS: CNHL and CNHF vs. best methods on LFW

Verification accuracy on LFW:

Method	**Accuracy**	
WebFace [25]	0.9613	
CNHL-200×1	**0.963±0.00494**	**25 bytes!**
DeepFace-ensemble[21]	0.9730±0.0025	
DeepID[19]	0.9745± 0.0026	
MFM Net[24]	0.9777	
CNHL-2000×1	**0.9814**	**250 bytes**
CNHF-2000×7	**0.9859**	
DeepID2[17]	0.9915 ± 0.0013	
DeepID3[18]	0.9953 ± 0.0010	
Baidu[11]	0.9977 ± 0.0006	

EXPERIMENTS: CNHL and CNHF vs. best methods on LFW

CNHF identification results:

- Our CNHF-2000×7 result is 0.93 rank-1 on LFW (*real-time single net with hashing forest*).
- Best reported* DeepID3 result is 0.96 rank-1 on LFW (*essentially deeper and slower multi-patch CNN*).

*Baidu 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.

CNHL verification results:

- Our CNHF-2000×1 outperforms DeepFace-ensemble [30], DeepID [27], WebFace [35] and MFM Net [34].
- DeepID2 [25], DeepID3 [26] and Baidu [14] *multi-patch* CNNs outperform our CNHF-2000×1 *based on single net*.

Conclusion: Our real-time CNHF-2000 solutions outperforms all single nets* and close enough to multi-patch nets.

*Google's FaceNet is formally a single net too, but it is too far from real-time

CONCLUSIONS

1. We develop the family of CNN-based binary face representations for real-time face identification:

- Our 2000×1-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 (CNHF-2000×7) achieves 98.59% verification accuracy and 93% rank-1 on LFW (add 3% to rank-1 of basic CNN).

2. We propose the multiple convolution deep network architecture for acceleration of source Max-Feature-Map (MFM) CNN architecture:

- Our CNHF generates binary face templates at the rate of 40+ fps with CPU Core i7
- Our CNHF generates binary face templates at the rate of 120+ fps with GPU GeForce GTX 650

CONCLUSIONS

3. 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.

- Our 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.

Ideas and plans for the future:

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

Acknowledgement

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

Thank you for your attention!

Real-Time Face Identification via CNN and Boosted Hashing Forest

Yury Vizilter, Vladimir Gorbatsevich, Andrey Vorotnikov, Nikita Kostromov

State Research Institute of Aviation Systems (GosNIIAS), Moscow, Russia

viz@gosniias.ru, gvs@gosniias.ru, vorotnikov@gosniias.ru, nikita-kostromov@yandex.ru

EXPERIMENTS: CNHL and CNHF vs. best methods on LFW

CNHL: two-step vs. one-step learning

- **32-bit binary face representation:**
  - Best one-step result – **91%** verification on LFW
    H. Fan at al., "Learning Compact Face Representation: Packing a Face into an int32," 2014.
  - Our two-step learned CNHF 32×1 provides **90%** only.
    (H. Fan at al., 2014)
- **96% accuracy on LFW:**
  - Our two-step learned CNHF-200×1 (**200 bit**) hash demonstrates 96.3% on LFW;
  - Best one-step result requires **1000 bit** for achieving the 96% verification on LFW (our CNHF-200×1 solution improves this face packing result in 5 times).

For possible questions