# Fast Training of Triplet-based Deep Binary Embedding Networks

Bohan Zhuang, Guosheng Lin, Chunhua Shen*, Ian Reid
The University of Adelaide; and Australian Centre for Robotic Vision

## Abstract

*In this paper, we aim to learn a mapping (or embedding) from images to a compact binary space in which Hamming distances correspond to a ranking measure for the image retrieval task. We make use of a triplet loss because this has been shown to be most effective for ranking problems. However, training in previous works can be prohibitively expensive due to the fact that optimization is directly performed on the triplet space, where the number of possible triplets for training is cubic in the number of training examples. To address this issue, we propose to formulate high-order binary codes learning as a multi-label classification problem by explicitly separating learning into two interleaved stages. To solve the first stage, we design a large-scale high-order binary codes inference algorithm to reduce the high-order objective to a standard binary quadratic problem such that graph cuts can be used to efficiently infer the binary codes which serve as the labels of each training datum. In the second stage we propose to map the original image to compact binary codes via carefully designed deep convolutional neural networks (CNNs) and the hashing function fitting can be solved by training binary CNN classifiers. An incremental/interleaved optimization strategy is proffered to ensure that these two steps are interactive with each other during training for better accuracy. We conduct experiments on several benchmark datasets, which demonstrate both improved training time (by as much as two orders of magnitude) as well as producing state-of-the-art hashing for various retrieval tasks.*

## 1. Introduction

With the rapid development of big data, large-scale nearest neighbor search with binary hash codes has attracted much more attention. Hashing methods aim to map the original features to compact binary codes that are able to preserve the *semantic* structure of the original features in the Hamming space. Compact binary codes are extremely suitable for efficient data storage and fast search.

*Corresponding author, e-mail: chunhua.shen@adelaide.edu.au


**Figure 1:** The Hamming distances calculated using the proposed hashing framework between pairs of faces. Each row represents a triplet of samples and the face pairs enclosed by a rectangle are from the same identity. Here each face image is represented by a 128-dimensional binary codes vector. We can see that a threshold of about 63 can correctly classify same-identity and different-identity pairs of faces.

A few hashing methods in the literature incorporate the triplet ranking loss to learn codes that preserve relative similarity relations [15, 16, 22, 38, 39]. In these works usually a triplet ranking loss is defined, followed by solving an expensive optimization problem. For instance, Lai *et al*. [15] and Zhao *et al*. [39] map original features into binary codes via deep convolutional neural networks (CNNs). Both use a triplet ranking loss designed to preserve relative similarities, with the key difference being in the exact form of the loss function used. Similarly, FaceNet [25] uses the triplet loss to learn a real-valued compact embedding of faces. All these methods suffer from huge training complexity, because they directly train the CNNs using the triplets, the number of which scales cubically with the number of images in the training set. For example, the training of FaceNet [25] took a few months on Google's computer clusters. Other work like [32] simply subsamples a small subset to reduce the computation complexity.

To address this issue, we employ a collaborative two-step approach, originally proposed in [18], to avoid directly learning hash functions based on the triplet ranking loss. This two-step approach enables us to convert triplet-based hashing into an efficient combination of solving binary quadratic programs and learning conventional CNN

classifiers. Hence, we don't need to directly optimize the loss function with huge number of triplets to learn deep hash functions. The result is an algorithm with computational complexity that is orders of magnitude lower than existing work such as [25, 39], but without sacrificing accuracy.

The two-step approach to hashing advocated by [17, 18] uses decision trees as hash functions in combination with the design of efficient binary code inference methods. The main difference of our work is as follows. The work in [17, 18] only preserves the *pairwise* similarity relations which do not directly encode relative semantic similarity relationships that are important for ranking-based tasks. In contrast, we use a triplet-based ranking loss to preserve relative semantic relationships. However it is not trivial to extend the first step (binary code inference) in [17] to triplet-based loss functions. The formulated binary quadratic problem (BQP) in [17] can be viewed as a pairwise Markov random field (MRF) inference problem, while in our case we need to solve large-scale *high-order* MRF inference. We here propose an efficient high-order binary code inference algorithm, in which we equivalently convert the binary high-order inference into the second-order binary quadratic problem, and graph cuts based block search method can be applied. In the second step of hash function learning, the work of [17, 18] relies on training classifiers such as linear SVM or decision trees on handcrafted features. We instead fit deep CNNs with incremental optimization to simultaneously learn feature representations and hash codes.

Our contributions are summarized as follows.

- To address the issue of prohibitively high computational complexity in triplet-based binary code learning, we propose a new efficient and flexible framework for interactively inferring binary codes and learning the deep hash functions, using a triplet-based loss function. We show how to convert the high-order loss introduced by the triplets into a binary quadratic problem that can be optimized efficiently in the manner of [17], using block-coordinate descent with graph-cuts. To learn the mapping from images to hash codes, we design deep CNNs capable of preserving their semantic ranking information of the data.
- We propose a novel incremental group-wise training approach, that interleaves finding groups of bits of the hash codes, with learning the hash functions. We show experimentally that this approach improves the quality of hash functions while retaining the advantage of efficient training.
- We demonstrate that our method outperforms many existing state-of-the-art hashing methods on several benchmark datasets by a large margin. We also demonstrate our hashing method in the context of a face search/retrieval system. We achieve the best reported results on face search under the IJB-A protocol.

### 1.1. Related work

Hashing methods may be roughly categorized into data-dependent and data-independent schemes. Data-independent methods [6, 10, 14] focus on using random projections to construct random hash functions. The canonical example is the locality-sensitive hashing (LSH) [6], which offers guarantees that metric similarity is preserved for sufficiently long codes based on random projections. Recent research focuses have been shifted to data-dependent methods, which learn hash functions in a either unsupervised, semi-supervised, or supervised learning fashion. Unsupervised hashing methods [2, 7, 20, 27, 34, 35] try to map the original features into hamming space while preserving similarity relations between the original features using unlabelled data. Supervised methods [5, 13, 16, 19, 26] use labelled training data for the similarity relations, aiming to preserve the "ground truth" similarity in the hash codes. Semi-supervised hashing methods incorporate ground-truth similarity information for the subset of the training data for which it is available, but also use unlaballed data. Our proposed method belongs to the supervised hashing framework.

Recently hashing using deep learning has shown great promise. The authors of [15, 39] learn hash bits such that multilevel semantic similarities are kept, taking raw pixels as input and training a deep CNN. This has the effect of simultaneously learning an image feature representation (in the early layers of the network) and the hash bits, which are obtained by thresholding the outputs of the last network layer, or *hash layer* at 0.5. Note that these methods suffer from huge computation complexity introduced by the triplet ranking loss for hashing. In contrast, our proposed method is much more efficient in training, as shown in our experiments.

## 2. The proposed approach

Our general problem formulation is as follows. Let $\mathcal{D} = \{(i,j,k) \,|\, s(\mathbf{x}_i, \mathbf{x}_j) > s(\mathbf{x}_i, \mathbf{x}_k)\}$ be a set of training triplet samples, in which $s(\cdot,\cdot)$ is some semantic similarity measures, $\mathbf{x}_i$ is the $i$-th training sample and $\mathbf{x}_i$ is semantically more similar to $\mathbf{x}_j$ than to $\mathbf{x}_k$. Let $h(\mathbf{x}) \in \{-1,1\}^q$ be the $q$-bit hash codes of image $\mathbf{x}$. We simplify the notation by rewriting $h(\mathbf{x}_i)$, $h(\mathbf{x}_j)$ and $h(\mathbf{x}_k)$ using $\boldsymbol{z}_i$, $\boldsymbol{z}_j$ and $\boldsymbol{z}_k$, respectively. Our goal is to learn embedding hash functions $h(\cdot)$ to preserve the relative similarity ranking order for the images after being mapped into the binary Hamming space. For that purpose, we define a general form of loss functions:

$$\min_{\mathbf{Z}} \sum_{(i,j,k)\in\mathcal{D}} \mathcal{L}(\boldsymbol{z}_i, \boldsymbol{z}_j, \boldsymbol{z}_k), \text{s.t. } \mathbf{Z} \in \{-1,1\}^{q\times n}. \quad (1)$$

Here $\mathbf{Z}$ is the matrix that collects binary codes for all the $n$ data points and $q$ is the bit length. $\mathcal{L}$ is a triplet loss function.

Unlike approaches such as [39], our method shares the advantage of [18] that we are not tied to a specific form of the loss. One typical example of losses that could be used include the *Hinge ranking loss*:

$$\mathcal{L}(\boldsymbol{z}_i, \boldsymbol{z}_j, \boldsymbol{z}_k) = \max(0,\, q/2 - (d_H(\boldsymbol{z}_i, \boldsymbol{z}_j) - d_H(\boldsymbol{z}_i, \boldsymbol{z}_k)). \tag{2}$$

Here $d_H(\cdot, \cdot)$ is the Hamming distance.

We propose an approach to learning binary hash codes that proceeds in two stages. The first stage uses the labelled training data to infer a set of binary codes in which the hamming distance between codes preserves the semantic ranking between triplets of data. The second stage uses deep CNNs to learn the mapping from images to the binary code space (i.e. to learn the hash functions). A similar two-stage approach was advocated in [17], but that work used only pairwise data, and used boosted decision trees rather than deep CNNs to learn the hash functions.

There are various difficulties associated with direct application of triplet losses, and of CNNs to the problem. First, the binary code learning stage requires optimization of Eq. (1) which is in general NP-hard. In Sec. 3, we describe how to infer binary codes with triplet ranking loss by reducing the problem to a binary quadratic program. The use of triplets considerably complicates this process and so this is one of our significant contributions in this paper. Second, while the two-stage approach gains significantly in training time, it has the disadvantage that the learning of the codes and the hash functions do not interact and therefore cannot be mutually beneficial. We propose a method to interleave the code and hash function learning into groups of bits, a process that retains much of the training efficiency, but improves the quality of the codes and hash functions considerably. We explain our use of CNNs and this interleaved and incremental learning in Sec. 4 below.

# 3. Inference for binary codes with triplet ranking loss

Since simultaneously infer multiple bits are intractable in inference task, inspired by the work of [17], we sequentially solve for one bit at a time conditioning on previous bits. When solving for the $r$-th bit, the previous $r-1$ bits are fixed. The binary inference problem becomes minimization of the following objective:

$$\begin{aligned}\sum_{(i,j,k)\in\mathcal{D}} \mathcal{L}(z_{r,i}, z_{r,j}, z_{r,k}; \boldsymbol{z}_i^{(r-1)}, \boldsymbol{z}_j^{(r-1)}, \boldsymbol{z}_k^{(r-1)}),\\ = \sum_{(i,j,k)\in\mathcal{D}} \ell_r(z_{r,i}, z_{r,j}, z_{r,k}),\end{aligned} \tag{3}$$

where $\ell_r$ is the loss function output of the $r$-th bit conditioned on the previous bits. $z_{r,i}$ is the binary code of the $i$-th data point and the $r$-th bit, $\boldsymbol{z}_i^{(r-1)}$ is the binary code vector of the previous $r-1$ bits for the $i$-th data point.

## 3.1. Solving high-order binary inference problem

Directly optimizing the loss function which involves high-order relations (more than pairwise relations) in Eq. (3) is difficult since the optimization involves an extremely large number of triplets, and so can be computationally intractable. To address this problem, we show here how to convert the high-order inference task to a second-order problem which is much more feasible to be optimized. The key "special properties" of the binary space that we rely on are: (i) the possibility of enumerating all possible inputs (there are $2^3 = 8$); (ii) the symmetry of the hamming distance $d(.,.)$. Based on this, the triplet loss can be decomposed into a set of second-order combinations as:

$$\begin{aligned}\ell_r(z_{r,i}, z_{r,j}, z_{r,k}) = \alpha_{ii} z_{r,i} z_{r,i} + \alpha_{ij} z_{r,i} z_{r,j} + \alpha_{ik} z_{r,i} z_{r,k}\\ + \alpha_{ji} z_{r,j} z_{r,i} + \alpha_{jj} z_{r,j} z_{r,j} + \alpha_{jk} z_{r,j} z_{r,k} + \alpha_{ki} z_{r,k} z_{r,i}\\ + \alpha_{kj} z_{r,k} z_{r,j} + \alpha_{kk} z_{r,k} z_{r,k},\end{aligned} \tag{4}$$

where $\alpha_{..}$ are the coefficients of the corresponding second-order combinations. Then we will show that there exists a solution for $\boldsymbol{\alpha}$ to make it a valid decomposition. Here we ignore the redundant terms in Eq. (4), hence it can be rewritten as

$$\begin{aligned}\ell_r(z_{r,i}, z_{r,j}, z_{r,k}) = \alpha_{ii} z_{r,i} z_{r,i} + \alpha_{ij} z_{r,i} z_{r,j}\\ + \alpha_{ik} z_{r,i} z_{r,k} + \alpha_{jk} z_{r,j} z_{r,k} = \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{v},\end{aligned} \tag{5}$$

$$\begin{aligned}\text{where,}\quad \boldsymbol{\alpha} &= [\alpha_{ii}, \alpha_{ij}, \alpha_{ik}, \alpha_{jk}],\\ \mathbf{v} &= [z_{r,i} z_{r,i}, z_{r,i} z_{r,j}, z_{r,i} z_{r,k}, z_{r,j} z_{r,k}].\end{aligned}$$

$\ell_r$ has 8 possible input combinations for $(z_{r,i}, z_{r,j}, z_{r,k})$ (or equivalently $\mathbf{v}$ has 8 possible value combinations), leading to 8 constraints of the form of (5). Because the loss is defined on Hamming distance/affinity, changing the sign of every input leads to identical value of the loss, thus some of these combinations lead to redundant constraints. Eliminating all these redundant combinations leaves only four independent equations (5). Stacking these so that each $\mathbf{v}$ forms a row of a matrix yields the follow set of equations:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \boldsymbol{\alpha} = \begin{bmatrix} \ell_r(1,1,1) \\ \ell_r(1,1,-1) \\ \ell_r(1,-1,1) \\ \ell_r(1,-1,-1) \end{bmatrix}. \tag{6}$$

which can be easily inverted to yield the unique solution of $\boldsymbol{\alpha}$. This shows that for a given triplet loss function, we can decompose it into a set of pairwise terms for each triplet.

We now seek a solution for $\boldsymbol{z}_{(r)}$ – the $r^{\text{th}}$ bit of the code for every data point – that optimizes the triplet relations. Because the triplet relations are now encoded as pairwise relations, we can solve for $\boldsymbol{z}_{(r)}$ as follows. We define $\mathbf{W} \in R^{n \times n}$ as a weight matrix in which $(i,j)$-th element

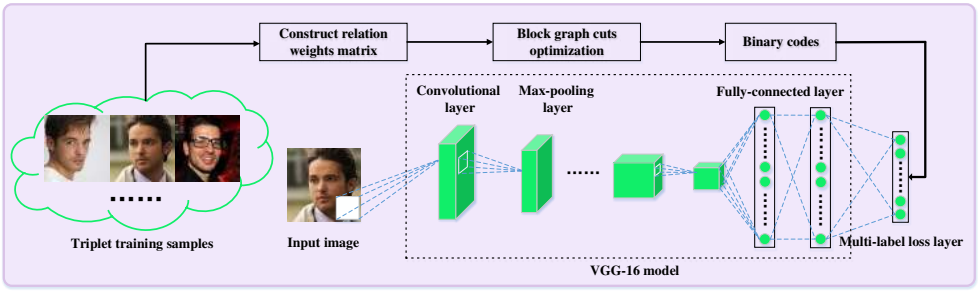



**Figure 2:** Overview of the proposed hashing framework for training one group of binary codes. The framework includes two steps: binary code inference and hash function learning with multi-label CNNs. The inferred binary codes are needed by the multi-label layer of the deep hash functions. The CNN structure of the first a few layers is same as the VGG-16 network.

**Algorithm 1:** Greedy method for constructing blocks

**Input**: Training images: $\{\mathbf{x}_1, ...\mathbf{x}_n\}$; Relation weights matrix: $\mathbf{W}$.
**Output**: Sub-modular blocks: $\{\mathcal{S}_1, \mathcal{S}_2, ...\}$.

$\mathcal{U} \leftarrow \{\mathbf{x}_1, ..., \mathbf{x}_n\}$; $t = 0$;
**while** $\mathcal{U} \neq \emptyset$ **do**
  $t = t + 1$; $\mathcal{S}_t \leftarrow \emptyset$; choose an arbitrary $\mathbf{x}_i$ from $\mathcal{U}$;
  Let $\mathcal{H}$ be $\mathcal{U} \cup \{\mathbf{x}_j | w_{ij} < 0\}$
  **for** *each* $\mathbf{x}_j$ *in* $\mathcal{H}$ **do**
    **if** $w_{jk} \leq 0$ *for* $k = 1, 2, ..., |\mathcal{S}_t|$ **then**
      Add $\mathbf{x}_j$ to $\mathcal{S}_t$; If $\mathbf{x}_j \in \mathcal{U}$, remove it;

of $\mathbf{W}$, $w_{ij}$, represents a relation weight between the $i$-th and $j$-th training points. Specifically, each element of $\mathbf{W}$ is computed as

$$w_{ij} = \sum_{\forall(i,j)} \alpha_{ij}, \tag{7}$$

where $\alpha_{ij}$ are the coefficients corresponding to the pair $(i, j)$. There will be one such $\alpha_{ij}$ for every triplet in which data points $\mathbf{x}_i$ and $\mathbf{x}_j$ appear.

The triplet optimization problem in Eq. (3) can now be equivalently formulated as

$$\min_{\boldsymbol{z}_{(r)} \in \{-1,1\}^n} \boldsymbol{z}_{(r)}^{\mathrm{T}} \mathbf{W} \boldsymbol{z}_{(r)}. \tag{8}$$

Note that the coefficients matrix $\mathbf{W}$ is sparse and symmetric, therefore Eq. (8) is a standard binary quadratic problem. Although we have now shown how to convert the third-order objective in Eq. (3) into a second-order formulation amenable to BQP, a further issue remains: the quadratic objective above contains non-submodular terms, and is therefore difficult to optimize.

To address this, we follow the proposal in [17]. This proceeds by creating a set of sub-problems (or "blocks") each involving a subset of the variables $\boldsymbol{z}_{(r)}$ in which the pairwise relations are all sub-modular. The sub-problems

**Algorithm 2:** Two-step approach for learning deep binary embedding networks

**Input**: Training images: $\{\mathbf{x}_1, ...\mathbf{x}_n\}$; Relation map: $\mathbf{M}$; group length: $a$; number of groups: $b$.
**Output**: The deep hash functions: $h(\cdot)$.

**for** $i = 1, ...b$ **do**
  **for** $j = 1, ...a$ **do**
    Solve linear equations to construct the relation weight matrix $\mathbf{W}$;
    Apply Block Graph-Cut algorithm [17] to solve $((i-1) \times a + j)$-th bit hash codes;
  Learn the deep hash functions $h(\cdot)$ based on $i \times a$ bits hash codes;
  Simultaneously update $i \times a$ bits hash codes by the output of $h(\cdot)$.

are then solved in turn, treating the variables that are not involved in the current block as constants. The inference problem for one block is written as

$$\min_{\boldsymbol{z}_r \in \{-1,1\}^n} \sum_{i \in \mathcal{S}} u_i z_{r,i} + \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} v_{ij} z_{r,i} z_{r,j}, \tag{9}$$

$$\text{where, } u_i = 2 \sum_{j \notin \mathcal{S}} w_{ij} z_{r,j}, \; v_{ij} = w_{ij},$$

and $\mathcal{S}$ is the block to be optimized. Since the above inference problem for one block is sub-modular, we can solve it efficiently using graph cuts.

Algorithm (1) details how the blocks are defined. It is subtly different from [17]; because we are using a triplet loss, the criterion for inclusion in a block is to ensure $w_{ij} < 0$ for each pair $\mathbf{x}_i, \mathbf{x}_j$ in the block, which guarantees sub-modularity for all pairs.

### 3.2. Loss function

The discussion above provides a general framework for learning the binary codes using a triplet loss, but is agnostic

to the exact form of the loss. In the experiments reported in this paper, we use $\ell_r$ as the triplet-based hinge loss function defined in Eq. (2):

$$\ell_r(...) = \max(0,\ r/2 - \Delta d_H^{(r-1)} - \Delta d_H^r), \qquad (10)$$

where,

$$\Delta d_H^{(r-1)} = d_H(\boldsymbol{z}_i^{(r-1)}, \boldsymbol{z}_j^{(r-1)}) - d_H(\boldsymbol{z}_i^{(r-1)}, \boldsymbol{z}_k^{(r-1)}),$$
$$\Delta d_H^r = d_H(z_{r,i}, z_{r,j}) - d_H(z_{r,i}, z_{r,k}).$$

## 4. Deep hash functions learning

Our general scheme now requires that we learn hash functions $h(.)$ that map from data points $\mathbf{x}_i$ to binary codes. We propose to do this using deep CNNs because they have repeatedly been shown to be very effective for similar tasks. The straightforward approach is then to use the training samples, and their known codes as the labelled training set for a standard CNN. As we have noted this two-stage approach yields significant training time gains.

However a major disadvantage is that because the binary codes are determined independently of the hash functions, and the hash functions have no possibility to influence the choice of binary codes. Ideally these stages would interact so that the choice of binary hash codes is influenced not only by the ground-truth relative similarity relations but also by how hard the training points are.

To address this, we propose an interleaved process where we infer a group of bits within a code, followed by learning suitable hash functions for that set of bits and its predecessors, followed in turn by inference of the next group of bits, and so on. This provides a compromise between independently learning the codes and hash functions, and a more end-to-end – but very expensive – approach such as [15].

### 4.1. Incremental optimization

Our key idea here is to optimize the hashing framework in an incremental group-wise manner. More specifically, we assume there are $b$ groups of bits and each group has $a$ bits (e.g., for 64-bit codes we may break this into 8 groups of 8 bits each). For convenience, we shall refer to inference of the $p$-th group binary codes followed by learning the deep hash functions, as the "$p$-th training stage". In the $p$-th training stage, we first infer the $a$ bits of the $p$-th group one bit at a time (as described in Sec. 3) and then train the network parameters $\theta$ so that it minimizes the cross-entropy loss:

$$-\sum_{\rho=1}^{r}\sum_{i=1}^{n}[\delta(z_{\rho,i}=1)\log {z'}_{\rho,i} + \delta(z_{\rho,i}=-1)\log(1-z'_{\rho,i})], \qquad (11)$$

where $\delta(\cdot)$ is the indication function. Here at the $p$-th stage we are targetting the first $r = pa$ bits of the code; $z'_{\rho,i}$ is the $\rho$-th output of the last sigmoid layer for the $i$-th training sample; $z_{\rho,i}$ is the corresponding bit of the binary code obtained from the inference step which serves as the target label of the multi-label classification problem above. Note that in the $p$-th training stage, the bits from all $p$ groups are used to guide the learning of the deep hash functions.

Having completed training the hash functions, we then update the binary codes for all $p$ groups by the output of the learned hash functions. The effect of this is to ensure that the error in the learned hash functions will influence the inference of the next group of hash bits.

This incremental training approach adaptively regulates the binary codes according to both the fitting capability of the deep hash functions and the properties of the training data, steadily improving the quality of hash codes and the final performance. Finally, we summarize our hashing framework in Algorithm 2.

### 4.2. Network architecture

The network of learning deep hash functions consists of multiple convolutional, pooling, and fully connected layers (we follow the VGG-16 model), and a multi-label loss layer for multi-label classification.

We use the pre-trained VGG-16 [28] model for initialization, which is trained on the large-scale ImageNet dataset. The multiple convolution-pooling and fully connected layers are used to capture mid-level image representations. The intermediate output of the last fully connected layer are mapped to a multi-label layer as the feature representation. Then neurons in the multi-label layer are activated by a sigmoid function so that the activations are approximated to $[0, 1]$, followed by the cross-entropy loss of Eq. (11) for multi-label classification.

## 5. Experiments

**Experimental settings** We test the proposed hashing method on two multi-class datasets, one multi-label dataset and one face retrieval dataset. For multi-class datasets, we use the MIT Indoor dataset [23] and CIFAR-10 dataset [12]. The MIT Indoor dataset contains 67 indoor scene categories, and 6,700 images for evaluation. CIFAR-10 contains 60,000 small images in 10 classes. For multilevel similarity measurement, we test our method on the multi-label dataset NUS-WIDE [4]. The NUS-WIDE dataset is a large database containing 269,648 images annotated with 81 concepts. We compare the search accuracies with four recent state-of-the-art state-of-the-art hashing methods, including SFHC [15] (the recent deep CNNs method), FSH [17] (two-step hashing approach using decision trees), KSH [19] and ITQ [7].

For fair comparison, we evaluate the compared hashing methods FSH, KSH and ITQ on the features obtained from the activations of the last hidden layer of the VGG-16 model pre-trained on the ImageNet ILSVRC-2012 dataset [24]. We find that using deep CNN features in general improve

the performance for these three hashing methods, compared with what was originally proposed. We initialize our CNN using the pre-trained model and fine-tune the network on the corresponding training set.

Again for fair comparison, for the deep CNN approach SFHC, we replace its network structure (convolution-pooling, fully-connected layers) with the VGG-16 model and end-to-end train the network based on the triplet hinge loss used in the original paper. We implement SFHC using *Theano* [1] and train the model using two GeForce GTX Titan X. The triplet samples are randomly generated in the course of training, following [15].

For the NUS-WIDE dataset, we construct two comparison settings, setting-1 and setting-2. For setting-1, following the previous work [15, 20], we consider the 21 most frequent tags and the similarity is defined based on whether two images share at least one common tag. For setting-2, we use the similarity precision evaluation metric to evaluate pairwise and triplet performance. As in [32], similarity precision is defined as the % of triplets being correctly ranked.

Given a triplet image set $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$, where $s(\mathbf{x}_i, \mathbf{x}_j) > s(\mathbf{x}_i, \mathbf{x}_k)$. We assume $\mathbf{x}_i$ as the query, if the rank of $\mathbf{x}_j$ is higher than $\mathbf{x}_k$, then we say triplet is correctly ranked. We first randomly sample 1000 probe images from all the data sharing the selected 21 attributes in setting-1. Then we obtain a ranking list for each probe image according to how many attributes it shares with the data and randomly generate 50 triplets per probe image according to the ranking list to form the test set. For the triplet-based methods, the sampled training data is the same as in setting-1. For the compared pairwise-based methods, we directly use the hash functions learned in setting-1 since semantic ranking information cannot be incorporated into the pairwise-based inference pipeline. For CIFAR-10 and NUS-WIDE setting-1, we use the same experimental setting as described in [15].

We use two evaluation metrics: Mean Average Precision (MAP) and the precision of the top-K retrieved examples (Precision), where K is set to 100 in CIFAR-10 and NUS-WIDE setting-1 and set to 80 in MIT Indoor dataset. For NUS-WIDE setting-1, we calculate the MAP values within the top 5000 returned neighbors. The results are represented in Figure 3 and Figure 4.

### 5.1. Implementation details

We implement the network training based on the CNN toolbox *Theano*. Training is done on a standard desktop with a GeForce GTX TITAN X with 12GB memory. In all experiments, we set the mini-batch size for gradient descent to 50, momentum 0.9, weight decay 0.0005 and dropout rate 0.5 on the fully connected layer to avoid over-fitting. The number of binary codes per group is set to 8.

**Figure 5:** The similarity precision curves on NUS-WIDE setting-2.

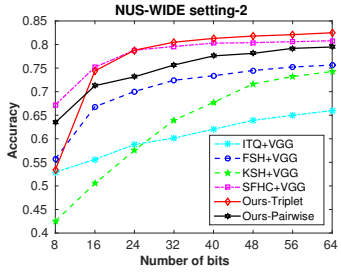
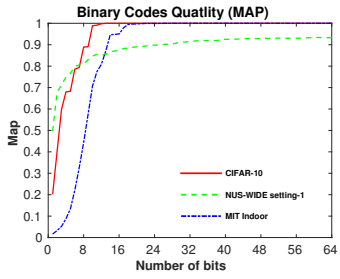



**Figure 6:** Evaluation of the inference performance on three datasets.




### 5.2. Analysis of retrieval results

On all the three datasets, our proposed method shows superior performance in terms of MAP and precision evaluation metrics against the most related work SFHC (deep CNN) and FSH (two-step hashing with boosted trees). As expected, the training speed of our method is much faster than SFHC, and the result is summarized in Table 1. Rather than simply end-to-end learn the hash functions, our method incorporates hash functions learning with a collaborative inference step, where the image representation learning and hash coding can benefit each other through this feedback scheme.

Compared to FSH, the results demonstrate the effectiveness of incorporating relative similarity information as supervision. Note that FSH is based on pairwise information while ours uses triplet based ranking information to learn hash codes. The triplet loss may be better for retrieval tasks because it is directly linked to retrieval measure such as the AUC score. The pairwise loss used by FSH encourages all images in one category to be projected onto a single point in the Hamming space. The triplet loss maximizes a margin between each pair of same-category images and images from different categories. As argued in [25, 33], this may enable images belonging to the same category to reside on a manifold; and at the same time to maintain a distance from other categories.

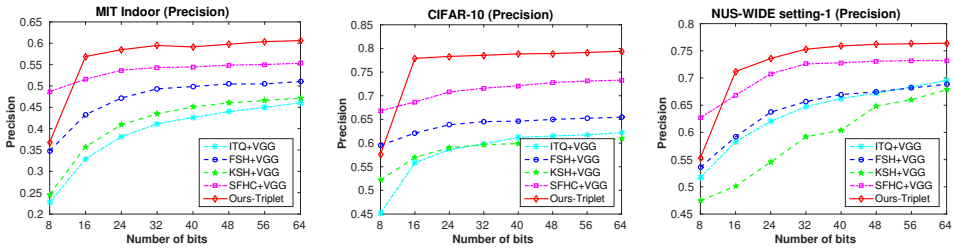



**Figure 3:** The precision curves on three datasets. We compare several state-of-the-art algorithms including ITQ [7], KSH [19], FSH [17] with features extracted from VGG-16 model which is fine-tuned on the corresponding training set and SHFC [15] which is implemented using the VGG-16 network structure.

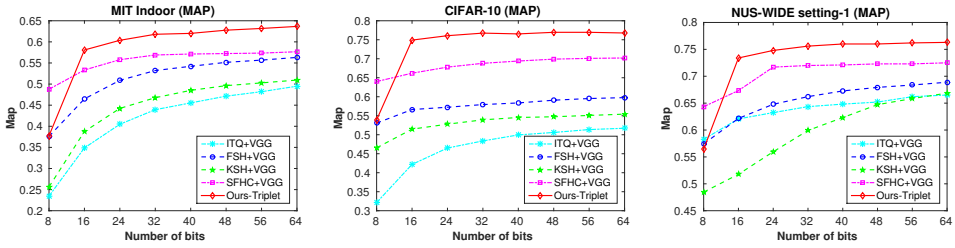



**Figure 4:** The mean average precision curves on three datasets. Settings are the same as in Figure 3.

**Table 1:** Training time of the proposed method and the method SFHC [15] on three datasets. In terms of training time, our method is significantly faster than SFHC.

| Method | Training Time (hours) | | | Number of GPUs |
|---|---|---|---|---|
| | MIT Indoor | CIFAR-10 | NUS-WIDE setting-1 | |
| Ours-Triplet | 18 | 15 | 32 | 1 |
| SFHC | 186 | 174 | 365 | 2 |

### 5.3. Triplet vs. pairwise

From the results shown in Figure 5, we can clearly observe the superiority of triplet-based methods on the ranking based evaluation metric. Thanks to the high quality binary codes and the strong fitting capability of our deep model, our proposed method provides much better performance than pairwise methods by a large margin.

Since the two triplet-based methods (Ours-Triplet and SFHC) simultaneously learn feature representations and hash codes while considering the semantic ranking information, they are more likely to learn hash functions that are tailored for the ranking-based retrieval metric than the pairwise-based methods (Ours-pairwise and FSH).

### 5.4. Evaluation of binary codes quality

We evaluate the binary codes quality on CIFAR-10, MIT Indoor and NUS-WIDE setting-1 datasets (see Figure 6). To evaluate the effectiveness of the binary codes inference pipeline, we infer 64 binary bits without learning the deep hash functions. Then the training database is used as both the probe set and the gallery set for evaluating the inference performance. For the three datasets, we calculate the MAP values within the returned neighbors. We can observe that for CIFAR-10, the binary codes converge very fast at around 10-th bits. MIT Indoor dataset converges slightly slower due to the fact that it has more classes. The binary codes can still perfectly separate all the training samples from different classes. This is because the relations between training points are very simple due to the multi-class similarity relationships. In contrast, due to the complicated relationships between the multi-label training samples, the accuracy of NUS-WIDE setting-1 keeps improving up to 64 bits and is lower than those multi-class datasets. We can see that the code quality is directly proportional to the final retrieval performance. This makes sense since the deep hash functions are learned to fit the binary codes, so the performance of the inference pipeline has a direct impact on the quality of the learned deep hash functions.

### 5.5. Face retrieval

We implement the face search application as follows. *Data preprocessing.* The preprocessing pipeline is: 1) detect the face region using the robust face detector [21] and find 68 face landmarks using the (state-of-the-art) face alignment algorithm [36]; 2) select the middle landmark between two eyes and the middle landmark of the mouth as alignment-anchor points, and align/scale the face image such that distance between the landmarks is 40 pixels; 3) finally we crop a $160 \times 160$ region around the mid-point of the two landmarks in (2).

*Supervised pre-training*. We pre-train the VGG-16 [28] network (using *Caffe* [9]) to classify all the 10575 subjects in

**Table 2:** Face search accuracies under the IJB-A protocol. Results for GOTS and OpenBR are quoted from [11]. Results are reported as the average ± standard deviation over the 10-fold cross validation sets specified in the IJB-A protocol.

| Algorithm | CMC (closed-set search) | | FNIR @ FPIR (open-set search) | |
|---|---|---|---|---|
| | Rank-1 | Rank-5 | 0.1 | 0.01 |
| GORS | $0.443 \pm 0.021$ | $0.595 \pm 0.020$ | $0.765 \pm 0.033$ | $0.953 \pm 0.024$ |
| OpenBR | $0.246 \pm 0.011$ | $0.375 \pm 0.008$ | $0.851 \pm 0.028$ | $0.934 \pm 0.017$ |
| Deep Face Search [31] | $0.820 \pm 0.024$ | $0.929 \pm 0.013$ | $0.387 \pm 0.032$ | $0.617 \pm 0.063$ |
| Proposed Method | $\mathbf{0.831 \pm 0.020}$ | $\mathbf{0.937 \pm 0.015}$ | $\mathbf{0.369 \pm 0.028}$ | $\mathbf{0.598 \pm 0.048}$ |

**Table 3:** Face search accuracies of the proposed method under the IJB-A protocol using different bits per group.

| Group length | CMC (closed-set search) | | FNIR @ FPIR (open-set search) | |
|---|---|---|---|---|
| | Rank-1 | Rank-5 | 0.1 | 0.01 |
| 8 bits | $\mathbf{0.831 \pm 0.020}$ | $\mathbf{0.937 \pm 0.015}$ | $\mathbf{0.369 \pm 0.028}$ | $\mathbf{0.598 \pm 0.048}$ |
| 32 bits | $0.818 \pm 0.023$ | $0.920 \pm 0.016$ | $0.385 \pm 0.030$ | $0.612 \pm 0.052$ |
| 64 bits | $0.793 \pm 0.024$ | $0.908 \pm 0.018$ | $0.398 \pm 0.036$ | $0.627 \pm 0.061$ |
| 128 bits | $0.778 \pm 0.023$ | $0.889 \pm 0.020$ | $0.415 \pm 0.035$ | $0.645 \pm 0.058$ |

the CASIA dataset [37]. This dataset has 494414 images of the 10575 subjects, and we double the number of training examples by horiozontal mirroring, making the feature representation more robust to pose variation.

We test the pre-trained model's discriminative power on the LFW verification data as follows. We use the last 4096-dimensional fully-connected layer as the feature representation and then use PCA to compress it into a 160-dimensional feature vector. Then CNN features are centered and normalized for evaluation. Under the standard LFW [8] face verification protocol, for a single network using only cosine similarity, we achieve an accuracy of $\mathbf{97.03}\% \pm \mathbf{0.98}\%$. Using the joint Bayesian method [3] for face verification, we achieve an accuracy of $\mathbf{98.18}\% \pm \mathbf{0.96}\%$.

Despite using only publicly available training data and one single network, the performance of this model is competitive with state-of-the-art [25, 29, 30, 37].

*Face search.* We then use the above pre-trained CNN model to initialize the deep CNN that models the hash functions of our proposed hashing method. We test the face search performance on the IARPA Janus Benchmark-A (IJB-A) dataset [11] which contains 500 subjects with a total of 25,813 face images. This dataset contains many challenging face images and defines both verification and search protocols. The search task (1:$N$ search) is defined in terms of comparisons between templates consisting of several face images, rather than single face images. For the search protocol, which evaluates both closed-set and open-set search performance, 10-fold cross validation sets are defined based on both the probe and gallery sets consisting of templates. Given an image from the IJB-A dataset, we first detect and align the face following the data preprocessing pipeline. After processing, the final training set consists approximately 1 million faces and 1 billion randomly sampled triplets. Clearly, such a large-scale training dataset may render most existing triplet-based hashing methods computationally intractable. The deep hash functions are learned based on the proposed two-step hashing framework. After the deep hash functions are learned, we generate 128 bits hash codes for each input face image for fast face retrieval. The definitions of CMC, FNIR and FPIR are explained in [11, 31]. The results of the proposed method along with the compared algorithms are reported in Table 2. In [31], a face is represented by the combined features extracted by 6 deep models. However, in our paper, 128 bits binary codes are directed extracted by a single deep model for face representation which enjoys both faster searching speed and less storage space. Also, although using the same training database, the searching accuracy on two protocols both demonstrate the effectiveness of our hashing framework.

### 5.6. Evaluation of the incremental learning

We evaluate different group lengths used in the incremental learning to prove the effectiveness of such an optimization strategy. We implement the experiments on the face retrieval task as described above since there are sufficient training examples and faces are difficult for the deep architecture to fit because of the relatively weak discriminative information they share. The results are reported in Table 3. From the results, we clearly see that smaller group length corresponds to better search accuracies, demonstrating our assertion that incremental optimization helps in terms of code quality and the final performance.

## 6. Conclusion

In this paper, we develop a general supervised hashing method with triplet ranking loss for large-scale image retrieval. Instead of directly training on the extremely large amount of triplet samples, we formulate learning of the deep hash functions as a multi-label classification problem, which allows us to learn deep hash functions orders of magnitude faster than the previous triplet based hashing methods in terms of training speed. The deep hash functions are learned in an incremental scheme, where the inferred binary codes are used to learn image representations and the learned hash functions can give feedback for boosting the quality of binary codes. Experiments demonstrate that the superiority of the proposed method over other state-of-the-art hashing methods.

## References


[1] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012. 6

[2] M. A. Carreira-Perpinan and R. Raziperchikolaei. Hashing with binary autoencoders. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 557–566, 2015. 2

[3] D. Chen, X. Cao, L. Wang, F. Wen, and J. Sun. Bayesian face revisited: A joint formulation. In *Proc. Eur. Conf. Comp. Vis.*, pages 566–579. 2012. 8

[4] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proc. of the ACM Int. Conf. on Image and Video Retrieval.*, 2009. 5

[5] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2475–2483, 2015. 2

[6] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *Proc. Int. Conf. Very Large Datadases*, volume 99, pages 518–529, 1999. 2

[7] Y. Gong, S. Lazebnik, A. Gordo, and 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, 2013. 2, 5, 7

[8] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, University of Massachusetts, Amherst, 2007. 8

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of the ACM Int. Conf. on Multimedia.*, pages 675–678, 2014. 7

[10] K. Jiang, Q. Que, and B. Kulis. Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4933–4941, 2015. 2

[11] B. F. Klare, B. Klein, E. Taborsky, A. Blanton, J. Cheney, K. Allen, P. Grother, A. Mah, M. Burge, and A. K. Jain. Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1931–1939, 2015. 8

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[13] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 1042–1050, 2009. 2

[14] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2130–2137, 2009. 2

[15] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3270–3278, 2015. 1, 2, 5, 6, 7

[16] X. Li, G. Lin, C. Shen, A. Van den Hengel, and A. Dick. Learning hash functions using column generation. In *Proc. Int. Conf. Mach. Learn.*, pages 142–150, 2013. 1, 2

[17] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1971–1978, 2014. 2, 3, 4, 5, 7

[18] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 2552–2559, 2013. 1, 2, 3

[19] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2074–2081, 2012. 2, 5, 7

[20] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proc. Int. Conf. Mach. Learn.*, pages 1–8, 2011. 2, 6

[21] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool. Face detection without bells and whistles. In *Proc. Eur. Conf. Comp. Vis.*, pages 720–735. 2014. 7

[22] M. Norouzi, D. M. Blei, and R. R. Salakhutdinov. Hamming distance metric learning. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 1061–1069, 2012. 1

[23] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 413–420, 2009. 5

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comp. Vis.*, pages 1–42, 2015. 5

[25] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 815–823, 2015. 1, 2, 6, 8

[26] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 37–45, 2015. 2

[27] F. Shen, C. Shen, Q. Shi, A. Van Den Hengel, and Z. Tang. Inductive hashing on manifolds. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1562–1569, 2013. 2

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5, 7

[29] Y. Sun, D. Liang, X. Wang, and X. Tang. Deepid3: Face recognition with very deep neural networks. *arXiv preprint arXiv:1502.00873*, 2015. 8

[30] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1701–1708, 2014. 8

[31] D. Wang, C. Otto, and A. K. Jain. Face search at scale: 80 million gallery. *arXiv preprint arXiv:1507.07242*, 2015. 8

[32] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1386–1393, 2014. 1, 6

[33] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, 2009. 6

[34] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *Proc. Eur. Conf. Comp. Vis.*, pages 340–353. 2012. 2

[35] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 1753–1760, 2009. 2

[36] X. Xiong and F. De la Torre. Supervised descent method and its applications to face alignment. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 532–539, 2013. 7

[37] D. Yi, Z. Lei, S. Liao, and S. Z. Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014. 8

[38] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval. *IEEE Trans. Image Proc.*, (12):4766–4779, 2015. 1

[39] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1556–1564, 2015. 1, 2, 3