This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Simultaneous Feature Learning and Hash Coding with Deep Neural Networks

Hanjiang Lai[†], Yan Pan *[‡], Ye Liu[§], and Shuicheng Yan[†]

[†]Department of Electronic and Computer Engineering, National University of Singapore, Singapore
[‡]School of Software, Sun Yan-Sen University, China
[§] School of Information Science and Technology, Sun Yan-Sen University, China

Abstract

Similarity-preserving hashing is a widely-used method for nearest neighbour search in large-scale image retrieval tasks. For most existing hashing methods, an image is first encoded as a vector of hand-engineering visual features, followed by another separate projection or quantization step that generates binary codes. However, such visual feature vectors may not be optimally compatible with the coding process, thus producing sub-optimal hashing codes. In this paper, we propose a deep architecture for supervised hashing, in which images are mapped into binary codes via carefully designed deep neural networks. The pipeline of the proposed deep architecture consists of three building blocks: 1) a sub-network with a stack of convolution layers to produce the effective intermediate image features; 2) a divide-and-encode module to divide the intermediate image features into multiple branches, each encoded into one hash bit; and 3) a triplet ranking loss designed to characterize that one image is more similar to the second image than to the third one. Extensive evaluations on several benchmark image datasets show that the proposed simultaneous feature learning and hash coding pipeline brings substantial improvements over other state-of-the-art supervised or unsupervised hashing methods.

1. Introduction

With the ever-growing large-scale image data on the Web, much attention has been devoted to nearest neighbor search via hashing methods. In this paper, we focus on learning-based hashing, an emerging stream of hash methods that learn similarity-preserving hash functions to encode input data points (e.g., images) into binary codes.

Many learning-based hashing methods have been proposed, e.g., [8, 9, 4, 12, 16, 27, 14, 25, 3]. The existing learning-based hashing methods can be categorized into unsupervised and supervised methods, based on whether supervised information (e.g., similarities or dissimilarities on data points) is involved. Compact bitwise representations are advantageous for improving the efficiency in both storage and search speed, particularly in big data applications. Compared to unsupervised methods, supervised methods usually embed the input data points into compact hash codes with fewer bits, with the help of supervised information.

In the pipelines of most existing hashing methods for images, each input image is firstly represented by a vector of traditional hand-crafted visual descriptors (e.g., GIST [18], HOG [1]), followed by separate projection and quantization steps to encode this vector into a binary code. However, such fixed hand-crafted visual features may not be optimally compatible with the coding process. In other words, a pair of semantically similar/dissimilar images may not have feature vectors with relatively small/large Euclidean distance. Ideally, it is expected that an image feature representation can sufficiently preserve the image similarities, which can be learned during the hash learning process. Very recently, Xia *et al.* [27] proposed CNNH, a supervised hashing method in which the learning process is decomposed into a stage of learning approximate hash codes from the supervised information, followed by a stage of simultaneously learning hash functions and image representations based on the learned approximate hash codes. However, in this two-stage method, the learned approximate hash codes are used to guide the learning of the image representation, but the learned image representation cannot give feedback for learning better approximate hash codes. This one-way interaction thus still has limitations.

In this paper, we propose a "one-stage" supervised hashing method via a deep architecture that maps input images to binary codes. As shown in Figure 1, the proposed deep architecture has three building blocks: 1) shared stacked

*Corresponding author: Yan Pan, email: panyan5@mail.sysu.edu.cn.

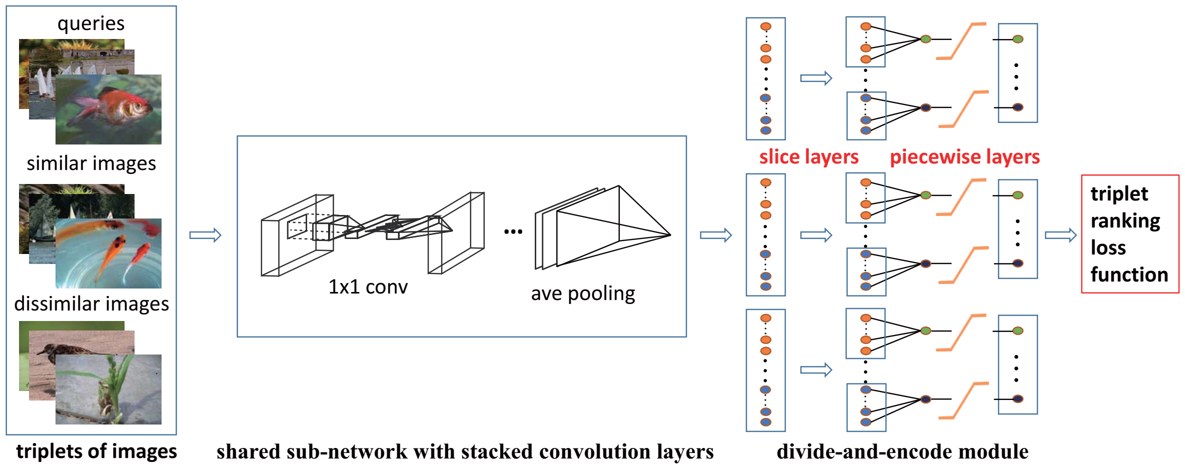


Figure 1. Overview of the proposed deep architecture for hashing. The input to the proposed architecture is in the form of triplets, i.e., (I, I^+, I^-) with a query image I being more similar to an image I^+ than to another image I^-. Through the proposed architecture, the image triplets are first encoded into a triplet of image feature vectors by a shared stack of multiple convolution layers. Then, each image feature vector in the triplet is converted to a hash code by a divide-and-encode module. After that, these hash codes are used in a triplet ranking loss that aims to preserve relative similarities on images.

convolution layers to capture a useful image representation, 2) divide-and-encode modules to divide intermediate image features into multiple branches, with each branch corresponding to one hash bit, (3) a triplet ranking loss [17] designed to preserve relative similarities. Extensive evaluations on several benchmarks show that the proposed deep-networks-based hashing method has substantially superior search accuracies over the state-of-the-art supervised or unsupervised hashing methods.

2. Related Work

Learning-based hashing methods can be divided into two categories: unsupervised methods and supervised methods.

Unsupervised methods only use the training data to learn hash functions that can encode input data points to binary codes. Notable examples in this category include Kernelized Locality-Sensitive Hashing [9], Semantic Hashing [19], graph-based hashing methods [26, 13], and Iterative Quantization [4].

Supervised methods try to leverage supervised information (e.g., class labels, pairwise similarities, or relative similarities of data points) to learn compact bitwise representations. Here are some representative examples in this category. Binary Reconstruction Embedding (BRE) [8] learns hash functions by minimizing the reconstruction errors between the distances of data points and those of the corresponding hash codes. Minimal Loss Hashing (MLH) [16] and its extension [17] learn hash codes by minimizing hinge-like loss functions based on similarities or relative similarities of data points. Supervised Hashing with Kernels (KSH) [12] is a kernel-based method that pursues compact binary codes to minimize the Hamming distances on similar pairs and maximize those on dissimilar pairs.

In most of the existing supervised hashing methods for images, input images are represented by some hand-crafted visual features (e.g. GIST [18]), before the projection and quantization steps to generate hash codes.

On the other hand, we are witnessing dramatic progress in deep convolution networks in the last few years. Approaches based on deep networks have achieved state-of-the-art performance on image classification [7, 21, 23], object detection [7, 23] and other recognition tasks [24]. The recent trend in convolution networks has been to increase the depth of the networks [11, 21, 23] and the layer size [20, 23]. The success of deep-networks-based methods for images is mainly due to their power of automatically learning effective image representations. In this paper, we focus on a deep architecture tailored for learning-based hashing. Some parts of the proposed architecture are designed on the basis of [11] that uses additional 1×1 convolution layers to increase the representational power of the networks.

Without using hand-crafted image features, the recently proposed CNNH [27] decomposes the hash learning process into a stage of learning approximate hash codes, followed by a deep-networks-based stage of simultaneously learning image features and hash functions, with the raw image pixels as input. However, a limitation in CNNH is that the learned image representation (in Stage 2) cannot be used to improve the learning of approximate hash codes, although the learned approximate hash codes can be used to guide the learning of image representation. In the proposed method, we learn the image representation and the hash codes in one stage, such that these two tasks have interaction and help each other forward.

3. The Proposed Approach

We assume $\mathcal{I}$ to be the image space. The goal of hash learning for images is to learn a mapping $\mathcal{F}: \mathcal{I} \rightarrow \{0,1\}^q$[1], such that an input image I can be encoded into a q-bit binary code $\mathcal{F}(I)$, with the similarities of images being preserved.

In this paper, we propose an architecture of deep convolution networks designed for hash learning, as shown in Figure 1. This architecture accepts input images in a triplet form. Given triplets of input images, the pipeline of the proposed architecture contains three parts: 1) a sub-network with multiple convolution-pooling layers to capture a representation of images; 2) a divide-and-encode module designed to generate bitwise hash codes; 3) a triplet ranking loss layer for learning good similarity measures. In the following, we will present the details of these parts, respectively.

3.1. Triplet Ranking Loss and Optimization

In most of the existing supervised hashing methods, the side information is in the form of pairwise labels that indicate the semantical similarites/dissimilarites on image pairs. The loss functions in these methods are thus designed to preserve the pairwise similarities of images. Recently, some efforts [17, 10] have been made to learn hash functions that preserve relative similarities of the form "image I is more similar to image I^+ than to image I^-". Such a form of triplet-based relative similarities can be more easily obtained than pairwise similarities (e.g., the click-through data from image retrieval systems). Furthermore, given the side information of pairwise similarities, one can easily generate a set of triplet constraints[2].

In the proposed deep architecture, we propose to use a variant of the triplet ranking loss in [17] to preserve the relative similarities of images. Specifically, given the training triplets of images in the form of (I, I^+, I^-) in which I is more similar to I^+ than to I^-, the goal is to find a mapping $\mathcal{F}(.)$ such that the binary code $\mathcal{F}(I)$ is closer to $\mathcal{F}(I^+)$ than to $\mathcal{F}(I^-)$. Accordingly, the triplet ranking hinge loss is defined by

$$\begin{aligned}&\hat{\ell}_{triplet}(\mathcal{F}(I), \mathcal{F}(I^+), \mathcal{F}(I^-))\\ =&\max(0, 1-(||\mathcal{F}(I)-\mathcal{F}(I^-)||_{\mathcal{H}} - ||\mathcal{F}(I)-\mathcal{F}(I^+)||_{\mathcal{H}}))\\ &s.t.\ \mathcal{F}(I),\ \mathcal{F}(I^+),\ \mathcal{F}(I^-) \in \{0,1\}^q,\end{aligned} \tag{1}$$

[1]In some of the existing hash methods, e.g., [27, 12], this mapping (or the set of hash functions) is defined as $\mathcal{F}: \mathcal{I} \rightarrow \{-1,1\}^q$, which is essentially the same as the definition used here.

[2]For example, for a pair of similar images (I_1, I_2) and a pair of dissimilar images (I_1, I_3), one can generate a triplet (I_1, I_2, I_3) that represents "image I_1 is more similar to image I_2 than to image I_3".

where $||.||_{\mathcal{H}}$ represents the Hamming distance. For ease of optimization, natural relaxation tricks on (1) are to replace the Hamming norm with the ℓ_2 norm and replace the integer constraints on $\mathcal{F}(.)$ with the range constraints. The modified loss functions is

$$\begin{aligned}&\ell_{triplet}(\mathcal{F}(I), \mathcal{F}(I^+), \mathcal{F}(I^-))\\ =&\max(0, ||\mathcal{F}(I)-\mathcal{F}(I^+)||_2^2 - ||\mathcal{F}(I)-\mathcal{F}(I^-)||_2^2 + 1)\\ &s.t.\ \mathcal{F}(I),\ \mathcal{F}(I^+),\ \mathcal{F}(I^-) \in [0,1]^q.\end{aligned} \tag{2}$$

This variant of triplet ranking loss is convex. Its (sub-)gradients with respect to $\mathcal{F}(I)$, $\mathcal{F}(I^+)$ or $\mathcal{F}(I^-)$ are

$$\begin{aligned}\frac{\partial \ell}{\partial b} &= (2b^- - 2b^+) \times I_{||b-b^+||_2^2 - ||b-b^-||_2^2+1>0}\\ \frac{\partial \ell}{\partial b^+} &= (2b^+ - 2b) \times I_{||b-b^+||_2^2 - ||b-b^-||_2^2+1>0}\\ \frac{\partial \ell}{\partial b^-} &= (2b^- - 2b) \times I_{||b-b^+||_2^2 - ||b-b^-||_2^2+1>0},\end{aligned} \tag{3}$$

where we denote $\mathcal{F}(I)$, $\mathcal{F}(I^+)$, $\mathcal{F}(I^-)$ as b, b^+, b^-. The indicator function $I_{condition} = 1$ if *condition* is true; otherwise $I_{condition} = 0$. Hence, the loss function in (2) can be easily integrated in back propagation in neural networks.

3.2. Shared Sub-Network with Stacked Convolution Layers

With this modified triplet ranking loss function (2), the input to the proposed deep architecture are triplets of images, i.e., $\{(I_i, I_i^+, I_i^-)\}_{i=1}^n$, in which I_i is more similar to I_i^+ than to I_i^- $(i = 1, 2, ...n)$. As shown in Figure 1, we propose to use a shared sub-network with a stack of convolution layers to automatically learn a unified representation of the input images. Through this sub-network, an input triplet (I, I^+, I^-) is encoded to a triplet of intermediate image features (x, x^+, x^-), where x, x^+, x^- are vectors with the same dimension.

In this sub-network, we adopt the architecture of Network in Network [11] as our basic framework, where we insert convolution layers with 1×1 filters after some convolution layers with filters of a larger receptive field. These 1×1 convolution filters can be regarded as a linear transformation of their input channels (followed by rectification non-linearity). As suggested in [11], we use an average-pooling layer as the output layer of this sub-network, to replace the fully-connected layer(s) used in traditional architectures (e.g., [7]). As an example, Table 1 shows the configurations of the sub-network for images of size 256×256. Note that all the convolution layers use rectification activation which are omitted in Table 1.

This sub-network is shared by the three images in each input triplet. Such a way of parameter sharing can significantly reduce the number of parameters in the whole architecture. A possible alternative is that, for (I, I^+, I^-)

Table 1. Configurations of the shared sub-network for input images of size 256 $\times$ 256

type	filter size/stride	output size
convolution	11×11 / 4	$96 \times 54 \times 54$
convolution	1×1 / 1	$96 \times 54 \times 54$
max pool	3×3 / 2	$96 \times 27 \times 27$
convolution	5×5 / 2	$256 \times 27 \times 27$
convolution	1×1 / 1	$256 \times 27 \times 27$
max pool	3×3 / 2	$256 \times 13 \times 13$
convolution	3×3 / 1	$384 \times 13 \times 13$
convolution	1×1 / 1	$384 \times 13 \times 13$
max pool	3×3 / 2	$384 \times 6 \times 6$
convolution	3×3 / 1	$1024 \times 6 \times 6$
convolution	1×1 / 1	$(50 \times$ # bits$) \times 6 \times 6$
ave pool	6×6 / 1	$(50 \times$ # bits$) \times 1 \times 1$

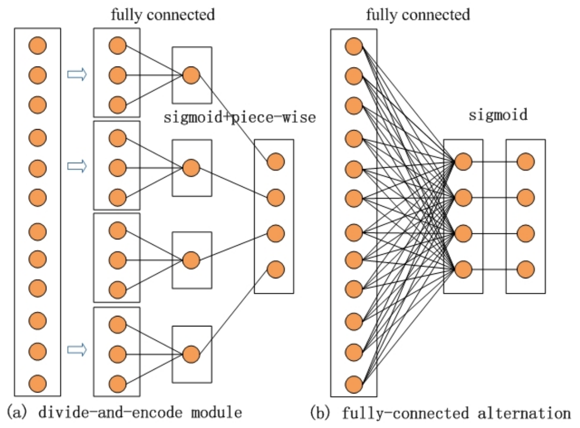


Figure 2. (a) A divide-and-encode module. (b) An alternative that consists of a fully-connected layer, followed by a sigmoid layer.

in a triplet, the query I has an independent sub-network P, while I^+ and I^- have a shared sub-network Q, where P/Q maps $I/(I^+, I^-)$ into the corresponding image feature vector(s) (i.e., x, x^+ and x^-, respectively)[3]. The scheme of such an alternative is similar to the idea of "asymmetric hashing" methods [15], which use two distinct hash coding maps on a pair of images. In our experiments, we empirically show that a shared sub-network of capturing a unified image representation performs better than the alternative with two independent sub-networks.

3.3. Divide-and-Encode Module

After obtaining intermediate image features from the shared sub-network with stacked convolution layers, we propose a divide-and-encode module to map these image features to approximate hash codes. We assume each target hash code has q bits. Then the outputs of the shared sub-network are designed to be $50q$ (see the output size of the average-pooling layer in Table 1). As can be seen in Figure 2(a), the proposed divide-and-encode module firstly divides the input intermediate features into q slices with equal length[4]. Then each slice is mapped to one dimension by a fully-connected layer, followed by a sigmoid activation function that restricts the output value in the range $[0, 1]$, and a piece-wise threshold function to encourage the output of binary hash bits. After that, the q output hash bits are concatenated to be a q-bit (approximate) code.

As shown in Figure 2(b), a possible alternative to the divide-and-encode module is a simple fully-connected layer that maps the input intermediate image features into q-dimensional vectors, followed by sigmoid activation functions to transform these vectors into $[0, 1]^q$. Compared to this alternative, the key idea of the overall divide-and-encode strategy is trying to reduce the redundancy among the hash bits. Specifically, in the fully-connected alternative in Figure 2(b), each hash bit is generated on the basis of the whole (and the same) input image feature vector, which may inevitably result in redundancy among the hash bits. On the other hand, since each hash bit is generated from a separated slice of features, the output hash codes from the proposed divide-and-encode module may be less redundant to each other. Hash codes with fewer redundant bits are advocated by some recent research. For example, the recently proposed Batch-Orthogonal Locality Sensitive Hashing [5] theoretically and empirically shows that hash codes generated by batch-orthogonalized random projections are superior to those generated by simple random projections, where batch-orthogonalized projections generate fewer redundant hash bits than random projections. In the experiments section, we empirically show that the proposed divide-and-encode module leads to superior performance over the fully-connected alternative.

In order to encourage the output of a divide-and-encode module to be binary codes, we use a sigmoid activation function followed by a piece-wise threshold function. Given a 50-dimensional slice $x^{(i)} (i = 1, 2, ..., q)$, the output of the 50-to-1 fully-connected layer is defined by

$$fc_i(x^{(i)}) = W_i x^{(i)}, \tag{4}$$

with W_i being the weight matrix.

[3] Another possible alternative is that each of I, I^+ and I^- in a triplet has an independent sub-networks (i.e., a sub-network $P/Q/R$ corresponds to $I/I^+/I^-$, respectively), which maps it into corresponding intermediate image features. However, such an alternative tends to get bad solutions. An extreme example is, for any input triplets, the sub-network P outputs hash codes with all zeros; the sub-network Q also outputs hash codes with all zeros; the sub-network R outputs hash codes with all ones. Such kind of solutions may have zero loss on training data, but their generalization performances (on test data) can be very bad. Hence, in order to avoid such bad solutions, we consider the alternative that uses a shared sub-network for I^+ and I^- (i.e., let $Q = R$).

[4] For ease of presentation, here we assume the dimension d of the input intermediate image features is a multiple of q. In practice, if $d = q \times s + c$ with $0 < c < q$, we can set the first c slices to be length of $s + 1$ and the rest $q - c$ ones to be length of s.

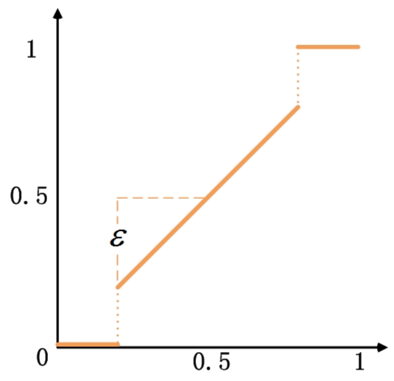


Figure 3. The piece-wise threshold function.

Given $c = fc_i(x^{(i)})$, the sigmoid function is defined by

$$sigmoid(c) = \frac{1}{1+e^{-\beta c}}, \quad (5)$$

where β is a hyper-parameter.

The piece-wise threshold function, as shown in Figure 3, is to encourage binary outputs. Specifically, for an input variable $s = sigmoid(c) \in [0, 1]$, this piece-wise function is defined by

$$g(s) = \begin{cases} 0, & s < 0.5 - \epsilon \\ s, & 0.5 - \epsilon \leq s \leq 0.5 + \epsilon \\ 1, & s > 0.5 + \epsilon, \end{cases} \quad (6)$$

where ϵ is a small positive hyper-parameter.

This piece-wise threshold function approximates the behavior of hard-coding, and it encourages binary outputs in training. Specifically, if the outputs from the sigmoid function are in $[0, 0.5 - \epsilon)$ or $(0.5 + \epsilon, 1]$, they are truncated to be 0 or 1, respectively. Note that in prediction, the proposed deep architecture only generates approximate (real-value) hash codes for input images, where these approximate codes are converted to binary codes by quantization (see Section 3.4 for details). With the proposed piece-wise threshold function, some of the values in the approximate hash codes (that are produced by the deep architecture) are already zeros or ones. Hence, less errors may be introduced by the quantization step.

3.4. Hash Coding for New Images

After the deep architecture is trained, one can use it to generate a q-bit hash code for an input image. As shown in Figure 4, in prediction, an input image I is first encoded into a q-dimensional feature vector $\mathcal{F}(I)$. Then one can obtain a q-bit binary code by simple quantization $b = sign(\mathcal{F}(I) - 0.5)$, where $sign(v)$ is the sign function on vectors that for $i = 1, 2, ..., q$, $sign(v_i) = 1$ if $v_i > 0$, otherwise $sign(v_i) = 0$.

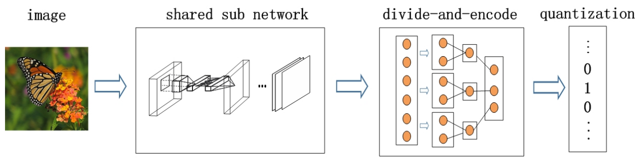


Figure 4. The architecture of prediction.

4. Experiments

4.1. Experimental Settings

In this section, we conduct extensive evaluations of the proposed method on three benchmark datasets:

- The Stree View House Numbers (**SVHN**)[5] dataset is a real-world image dataset for recognizing digits and numbers in natural scene images. SVHN consists of over 600,000 32×32 color images in 10 classes (with digits from 0 to 9).
- The **CIFAR-10**[6] dataset consists of 60,000 color images in 10 classes. Each class has 6,000 images in size 32×32.
- The **NUS-WIDE**[7] dataset contains nearly 270,000 images collected from Flickr. Each of these images is associated with one or multiple labels in 81 semantic concepts. For a fair comparison, we follow the settings in [27, 13] to use the subset of images associated with the 21 most frequent labels, where each label associates with at least 5,000 images. We resize images of this subset into 256×256.

We test and compare the search accuracies of the proposed method with eight state-of-the-art hashing methods, including three unsupervised methods LSH [2], SH [26] and ITQ [4], and five supervised methods CNNH [27], KSH [12], MLH [16], BRE [8] and ITQ-CCA [4].

In SVHN and CIFAR-10, we randomly select 1,000 images (100 images per class) as the test query set. For the unsupervised methods, we use the rest images as training samples. For the supervised methods, we randomly select 5,000 images (500 images per class) from the rest images as the training set. The triplets of images for training are randomly constructed based on the image class labels.

In NUS-WIDE, we randomly select 100 images from each of the selected 21 classes to form a test query set of 2,100 images. For the unsupervised methods, the rest images in the selected 21 classes are used as the training set. For supervised methods, we uniformly sample 500 images from each of the selected 21 classes to form a training set.

[5]http://ufldl.stanford.edu/housenumbers/
[6]http://www.cs.toronto.edu/ kriz/cifar.html
[7]http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

Table 2. MAP of Hamming ranking w.r.t different numbers of bits on three datasets. For NUS-WIDE, we calculate the MAP values within the top 5000 returned neighbors. The results of CNNH is directly cited from [27]. CNNH⋆ is our implementation of the CNNH method in [27] using Caffe, by using a network configuration comparable to that of the proposed method (see the text in Section 4.1 for implementation details).

Method	SVHN(MAP)				CIFAR-10(MAP)				NUS-WIDE(MAP)			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48bits	12 bits	24 bits	32 bits	48 bits
Ours	**0.899**	**0.914**	**0.925**	**0.923**	**0.552**	**0.566**	**0.558**	**0.581**	**0.674**	**0.697**	**0.713**	**0.715**
CNNH⋆	0.897	0.903	0.904	0.896	0.484	0.476	0.472	0.489	0.617	0.663	0.657	0.688
CNNH [27]	N/A				0.439	0.511	0.509	0.522	0.611	0.618	0.625	0.608
KSH [12]	0.469	0.539	0.563	0.581	0.303	0.337	0.346	0.356	0.556	0.572	0.581	0.588
ITQ-CCA [4]	0.428	0.488	0.489	0.509	0.264	0.282	0.288	0.295	0.435	0.435	0.435	0.435
MLH [16]	0.147	0.247	0.261	0.273	0.182	0.195	0.207	0.211	0.500	0.514	0.520	0.522
BRE [8]	0.165	0.206	0.230	0.237	0.159	0.181	0.193	0.196	0.485	0.525	0.530	0.544
SH [26]	0.140	0.138	0.141	0.140	0.131	0.135	0.133	0.130	0.433	0.426	0.426	0.423
ITQ [4]	0.127	0.132	0.135	0.139	0.162	0.169	0.172	0.175	0.452	0.468	0.472	0.477
LSH [2]	0.110	0.122	0.120	0.128	0.121	0.126	0.120	0.120	0.403	0.421	0.426	0.441

The triplets for training are also randomly constructed based on the image class labels.

For the proposed method and CNNH, we directly use the image pixels as input. For the other baseline methods, we follow [27, 12] to represent each image in SVHN and CIFAR-10 by a 512-dimensional GIST vector; we represent each image in NUS-WIDE by a 500-dimensional bag-of-words vector [8].

To evaluate the quality of hashing, we use four evaluation metrics: Mean Average Precision (MAP), Precision-Recall curves, Precision curves within Hamming distance 2, and Precision curves w.r.t. different numbers of top returned samples. For a fair comparison, all of the methods use identical training and test sets.

We implement the proposed method based on the open-source **Caffe** [6] framework. In all experiments, our networks are trained by stochastic gradient descent with 0.9 momentum [22]. We initiate ϵ in the piece-wise threshold function to be 0.5 and decrease it by 20% after every $20,000$ iterations. The mini-batch size of images is 64. The weight decay parameter is 0.0005.

The results of BRE, ITQ, ITQ-CCA, KSH, MLH and SH are obtained by the implementations provided by their authors, respectively. The results of LSH are obtained from our implementation. Since the network configurations of CNNH in [27] are different from those of the proposed method, for a fair comparison, we carefully implement CNNH (referred to as CNNH⋆) based on Caffe, where we use the code provided by the authors of [27] to implement the first stage. In the second stage of CNNH⋆, we use the same stack of convolution-pooling layers as in Table 1, except for modifying the size of the last convolution to $bits \times 1 \times 1$ and using an average pooling layer of size $bits \times 1 \times 1$ as the output layer.

[8]These bag-of-words features are available in the NUS-WIDE dataset.

4.2. Results of Search Accuracies

Table 2 and Figure 2∼4 show the comparison results of search accuracies on all of the three datasets. Two observations can be made from these results:

(1) On all of the three datasets, the proposed method achieves substantially better search accuracies (w.r.t. MAP, precision within Hamming distance 2, precision-recall, and precision with varying size of top returned samples) than those baseline methods using traditional hand-crafted visual features. For example, compared to the best competitor KSH, the MAP results of the proposed method indicate a relative increase of **58.8**% ∼**90.6.**% **/ 61.3**% ∼ **82.2** % **/ 21.2**% ∼ **22.7**% on SVHN / CIFAR-10 / NUS-WIDE, respectively.

(2) In most metrics on all of the three datasets, the proposed method shows superior performance gains against the most related competitors CNNH and CNNH⋆, which are deep-networks-based two-stage methods. For example, with respect to MAP, compared to the corresponding second best competitor, the proposed method shows a relative increase of **9.6** % ∼ **14.0** % **/ 3.9**% ∼ **9.2**% on CIFAR-10 / NUS-WIDE, respectively[9]. These results verify that simultaneously learning useful representation of images and hash codes of preserving similarities can benefit each other.

4.3. Comparison Results of the Divide-and-Encode Module against Its Alternative

A natural alternative to the divide-and-encode module is a simple fully-connected layer followed by a sigmoid layer of restricting the output values' range in $[0, 1]$ (see Figure 2(b)). To investigate the effectiveness of the divide-and-

[9]Note that, on CIFAR-10, some MAP results of CNNH⋆ are inferior to those of CNNH [27]. This is mainly due to different network configurations and optimization frameworks between these two implementations. CNNH⋆ is implemented based on Caffe [6]. But the core of the original implementation in CNNH [27] is based on Cuda-Convnet [7].

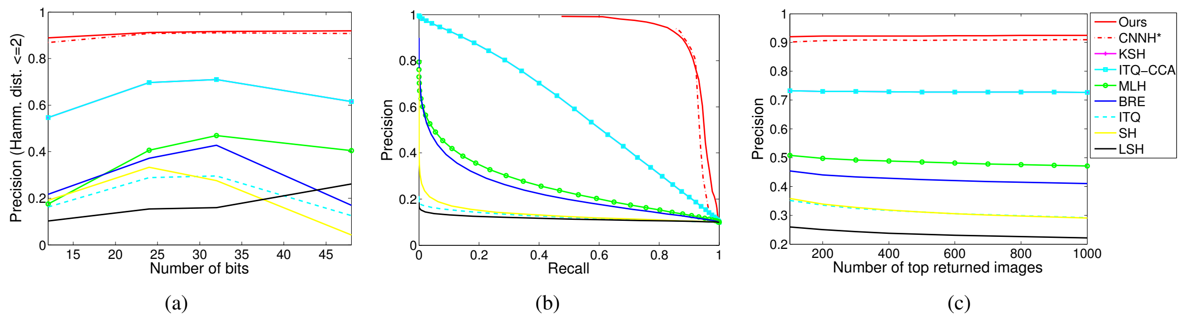


Figure 5. The comparison results on SVNH. (a) Precision curves within Hamming radius 2; (b) precision-recall curves of Hamming ranking with 48 bits; (c) precision curves with 48 bits w.r.t. different numbers of top returned samples.

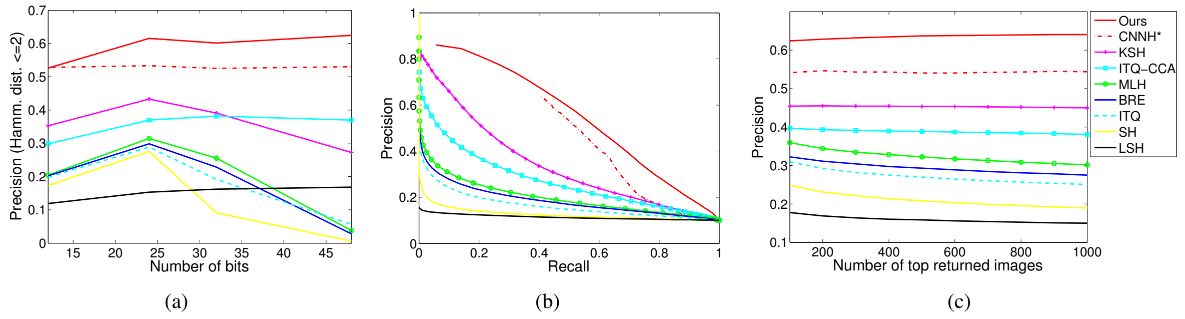


Figure 6. The comparison results on CIFAR10. (a) precision curves within Hamming radius 2; (b) precision-recall curves of Hamming ranking with 48 bits; (c) precision curves with 48 bits w.r.t. different number of top returned samples

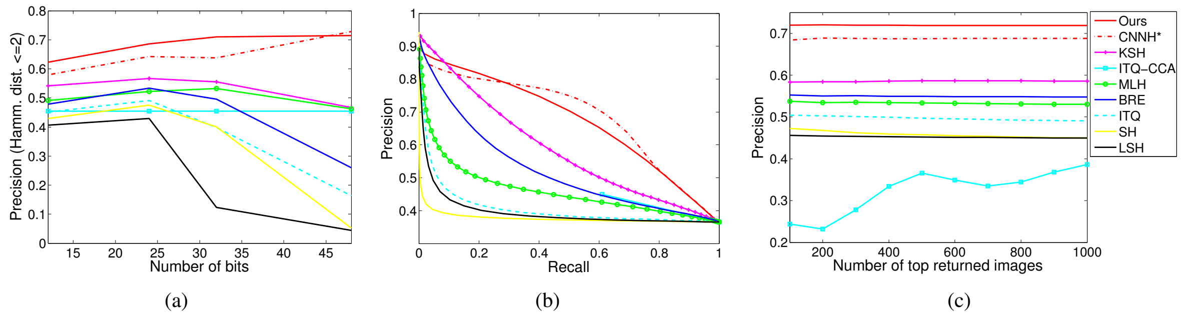


Figure 7. The comparison results on NUS-WIDE. (a) precision curves within Hamming radius 2; (b) precision-recall curves of Hamming ranking with 48 bits; (c) precision curves with 48 bits w.r.t. different number of top returned samples

encode module (DEM), we implement and evaluate a deep architecture derived from the proposed one in Figure 1, by replacing the divide-and-encode module with its alternative in Figure 2(b) and keeping other layers unchanged. We refer to it as "FC".

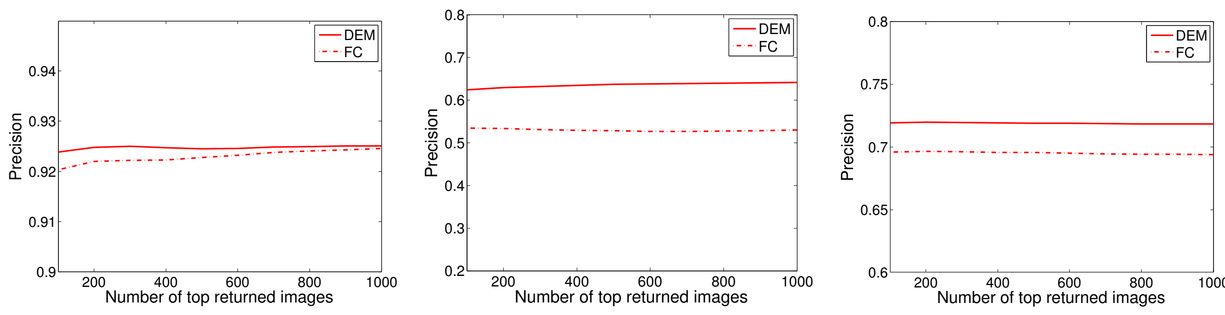
As can be seen from Table 3 and Figure 8, the results of the proposed method outperform the competitor with the alternative of the divide-and-encode module. For example, the architecture with DEM achieves 0.581 accuracy with 48 bits on CIFAR-10, which indicates an improvement of 19.7% over the FC alternative. The underlying reason for the improvement may be that, compared to the FC alternative, the output hash codes from the divide-and-encode modules are less redundant to each other.

4.4. Comparison Results of a Shared Sub-Network against Two Independent Sub-Networks

In the proposed deep architecture, we use a shared sub-network to capture a unified image representation for the three images in an input triplet. A possible alternative to this shared sub-network is that for a triplet (I, I^+, I^-), the query I has an independent sub-network P, while I^+

Table 3. Comparison results of the divide-and-encode module and its fully-connected alternative on three datasets.

Method	SVHN(MAP)				CIFAR-10(MAP)				NUS-WIDE(MAP)			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48bits	12 bits	24 bits	32 bits	48 bits
Ours (DEM)	**0.899**	**0.914**	**0.925**	**0.923**	**0.552**	**0.566**	**0.558**	**0.581**	**0.674**	**0.697**	**0.713**	**0.715**
Ours (FC)	0.887	0.896	0.909	0.912	0.465	0.497	0.489	0.485	0.623	0.673	0.682	0.691



(a) (b) (c)

Figure 8. The precision curves of divide-and-encode module versus its fully-connected alternative with 48 bits w.r.t. different number of top returned samples

and I^- has a shared sub-network Q, where P/Q maps $I/(I^+, I^-)$ into the corresponding image feature vector(s) (i.e., x, x^+ and x^-, respectively).

We implement and compare the search accuracies of the proposed architecture with a shared sub-network to its alternative with two independent sub-networks. As can be seen in Table 4 and 5, the results of the proposed architecture outperform the competitor with the alternative with two independent sub-networks. Generally speaking, although larger networks can capture more information, it also needs more training data. The underlying reason why the architecture with a shared sub-network performs better than the one with two independent sub-networks may be that the training samples are not enough for networks with too much parameters (e.g., 500 training images per class on CIFAR-10 and NUS-WIDE).

Table 4. Comparison results of a shared sub-network against two independent sub-networks on CIFAR-10.

Methods	12 bits	24 bits	32 bits	48 bits
MAP				
1-sub-network	**0.552**	**0.566**	**0.558**	**0.581**
2-sub-networks	0.467	0.494	0.477	0.515
Precision within Hamming radius 2				
1-sub-network	**0.527**	**0.615**	**0.602**	**0.625**
2-sub-networks	0.450	0.564	0.549	0.588

5. Conclusion

In this paper, we developed a "one-stage" supervised hashing method for image retrieval, which generates bitwise hash codes for images via a carefully designed deep architecture. The proposed deep architecture uses a triplet ranking loss designed to preserve relative similarities. Throughout the proposed deep architecture, input images are converted into unified image representations via a shared sub-network of stacked convolution layers. Then, these intermediate image representations are encoded into hash codes by divide-and-encode modules. Empirical evaluations in image retrieval show that the proposed method has superior performance gains over state-of-the-arts.

Table 5. Comparison results of a shared sub-network against two independent sub-networks on NUSWIDE.

Methods	12 bits	24 bits	32 bits	48 bits
MAP				
1-sub-network	**0.674**	**0.697**	**0.713**	**0.715**
2-sub-networks	0.640	0.686	0.688	0.697
Precision within Hamming radius 2				
1-sub-network	**0.623**	**0.686**	**0.710**	**0.714**
2-sub-networks	0.579	0.664	0.696	0.704

Acknowledgment

This work was partially supported by Adobe Gift Funding. It was also supported by the National Natural Science Foundation of China under Grants 61370021, U1401256, 61472453, Natural Science Foundation of Guangdong Province under Grant S2013010011905.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005. 1

[2] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*, pages 518–529, 1999. 5, 6

[3] Y. Gong, S. Kumar, H. A. Rowley, and S. Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 484–491, 2013. 1

[4] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011. 1, 2, 5, 6

[5] J. Ji, S. Yan, J. Li, G. Gao, Q. Tian, and B. Zhang. Batch-orthogonal locality-sensitive hashing for angular similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):1963–1974, 2014. 4

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 6

[7] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1106–1114, 2012. 2, 3, 6

[8] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1042–1050, 2009. 1, 2, 5, 6

[9] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2130–2137, 2009. 1, 2

[10] X. Li, G. Lin, C. Shen, A. v. d. Hengel, and A. Dick. Learning hash functions using column generation. In *Proceedings of the International Conference on Machine Learning*, pages 142–150, 2013. 3

[11] M. Lin, Q. Chen, and S. Yan. Network in network. In *Proceedings of the International Conference on Learning Representations*, 2014. 2, 3

[12] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012. 1, 2, 3, 5, 6

[13] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the International Conference on Machine Learning*, pages 1–8, 2011. 2, 5

[14] X. Liu, J. He, B. Lang, and S.-F. Chang. Hash bit selection: a unified solution for selection problems in hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1570–1577, 2013. 1

[15] B. Neyshabur, N. Srebro, R. Salakhutdinov, Y. Makarychev, and P. Yadollahpour. The power of asymmetry in binary hashing. In *Advances in Neural Information Processing Systems*, pages 2823–2831, 2013. 4

[16] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. In *Proceedings of the International Conference on Machine Learning*, pages 353–360, 2011. 1, 2, 5, 6

[17] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1–9, 2012. 2, 3

[18] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001. 1, 2

[19] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 412–419, 2007. 2

[20] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 2

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[22] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013. 6

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 2

[24] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014. 2

[25] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 2012. 1

[26] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 1753–1760, 2008. 2, 5, 6

[27] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proceedings of the AAAI Conference on Artificial Intellignece*, pages 2156–2162, 2014. 1, 2, 3, 5, 6