Supervised Hashing with Kernels

Wei Liu[†] Jun Wang[‡] Rongrong Ji[†] Yu-Gang Jiang[§] Shih-Fu Chang[†]
[†]Electrical Engineering Department, Columbia University, New York, NY, USA
[‡]IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
[§]School of Computer Science, Fudan University, Shanghai, China
{wliu,rrji,sfchang}@ee.columbia.edu wangjun@us.ibm.com ygj@fudan.edu.cn

Abstract

Recent years have witnessed the growing popularity of hashing in large-scale vision problems. It has been shown that the hashing quality could be boosted by leveraging supervised information into hash function learning. However, the existing supervised methods either lack adequate performance or often incur cumbersome model training. In this paper, we propose a novel kernel-based supervised hashing model which requires a limited amount of supervised information, i.e., similar and dissimilar data pairs, and a feasible training cost in achieving high quality hashing. The idea is to map the data to compact binary codes whose Hamming distances are minimized on similar pairs and simultaneously maximized on dissimilar pairs. Our approach is distinct from prior works by utilizing the equivalence between optimizing the code inner products and the Hamming distances. This enables us to sequentially and efficiently train the hash functions one bit at a time, yielding very short yet discriminative codes. We carry out extensive experiments on two image benchmarks with up to one million samples, demonstrating that our approach significantly outperforms the state-of-the-arts in searching both metric distance neighbors and semantically similar neighbors, with accuracy gains ranging from 13% to 46%.

1. Introduction

Recently, hashing has become a popular method applied to large-scale vision problems including object recognition [15][16], image retrieval [7][19], local descriptor compression [3][14], image matching [4][14], etc. In these problems, hashing is exploited to expedite similarity computation and search. Since the encoding of high-dimensional image feature vectors to short binary codes as proposed in [16], compact hashing has enabled significant efficiency gains in both storage and speed. In many cases, hashing with only several tens of bits per image allows search into a collection of millions of images in a constant time [16][17].

The early exploration of hashing focuses on using random projections to construct randomized hash functions. One of the well-known representatives is Locality-Sensitive Hashing (LSH) which has been continuously expanded to accommodate more distance/similarity metrics including ℓ_p distance for $p \in (0, 2]$ [1], Mahalanobis distance [7], and kernel similarity [6]. Because of theoretical guarantees that original metrics are asymptotically preserved in the Hamming (code) space with increasing code length, LSH-related methods usually require long codes to achieve good precision. Nonetheless, long codes result in low recall since the collision probability that two codes fall into the same hash bucket decreases exponentially as the code length increases.

In contrast to the data-independent hash schemes employed in LSH-related methods, recent endeavors aim at *data-dependent hashing* which poses a compact set of hash bits. Through encoding high-dimensional data points into compact codes, nearest neighbor search can be accomplished with a constant time complexity as long as the neighborhood of a point is well preserved in the code space. In addition, compact codes are particularly useful for saving storage in gigantic databases. To design effective compact hashing, a number of methods such as projection learning for hashing [17][2][14], Spectral Hashing (SH) [18], Anchor Graph Hashing (AGH) [8], Semi-Supervised Hashing (SSH) [17], Restricted Boltzmann Machines (RBMs) (or semantic hashing) [13], Binary Reconstruction Embeddings (BRE) [5], and Minimal Loss Hashing (MLH) [11] have been proposed. We summarize them into three categories, *unsupervised*, *semi-supervised*, and *supervised* methods.

For unsupervised hashing, principled linear projections like PCA Hashing (PCAH) [17] and its rotational variant [2] were suggested for better quantization rather than random projections. Nevertheless, only a few orthogonal projections are good for quantization as the variances of data usually decay rapidly as pointed out by [17]. An alternative solution is to seek nonlinear data representation from the low-energy spectrums of data neighborhood graphs [18][8]. Exactly solving eigenvectors or eigenfunctions of large-scale

graphs is computationally prohibitive. In response, [18][8] proposed several approximate solutions by adopting restrictive assumptions about the data distribution or the neighborhood structure.

While unsupervised hashing is promising to retrieve metric distance neighbors, e.g., ℓ_2 neighbors, a variety of practical applications including image search prefer semantically similar neighbors [16]. Therefore, recent works incorporated supervised information to boost the hashing performance. Such supervised information is customarily expressed as pairwise labels of similar and dissimilar data pairs in availability. One representative work is Semi-Supervised Hashing (SSH) [17] which minimized the empirical error on the labeled data while maximizing entropy of the generated hash bits over the unlabeled data. Another work, namely Weakly-Supervised Hashing [9], handled higher-order supervised information.

Importantly, we argue that supervised hashing could attain higher search accuracy than unsupervised and semi-supervised hashing if the supervised information were thoroughly taken advantage of. Though the simple supervised method Linear Discriminant Analysis Hashing (LDAH) [14] can tackle supervision via easy optimization, it lacks adequate performance because of the use of orthogonal projections in hash functions. There exist more sophisticated supervised methods such as RBM [13], BRE [5] and MLH [11] that have shown higher search accuracy than unsupervised hashing approaches, but they all impose difficult optimization and slow training mechanisms. This inefficiency issue has greatly diminished the applicability of the existing supervised hashing methods to large-scale tasks. We discover that the expensive training costs are caused by the overcomplicated hashing objective functions used in the prior works. To this end, high-quality supervised hashing with a low training cost is fundamentally important, yet still unavailable to the best of our knowledge.

In this paper, we show that the supervised information can be incorporated in a more effective and efficient manner. Specifically, we propose a novel and elegant objective function for learning the hash functions. The prior supervised methods [5][11] both tried to control the Hamming distances in the code space such that they correlate well with the given supervised information. Unfortunately, it is very difficult to directly optimize Hamming distances that are nonconvex and nonsmooth [5]. By utilizing the algebraic equivalence between a Hamming distance and a code inner product, the proposed objective function dexterously manipulates code inner products, leading to implicit yet more effective optimization on Hamming distances. We also exploit the separable property of code inner products to design an efficient greedy algorithm which sequentially solves the target hash functions bit by bit. To accommodate linearly inseparable data, we employ a kernel formulation for the target hash functions, so we name the proposed approach *Kernel-Based Supervised Hashing* (KSH). We evaluate the performance of KSH in searching both metric distance neighbors and semantically similar neighbors on two large image benchmarks up to one million samples, confirming its dramatically higher accuracy compared with the state-of-the-art unsupervised, semi-supervised, and supervised hashing methods.

2. Kernel-Based Supervised Hashing

2.1. Hash Functions with Kernels

Given a data set $\mathcal{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, the purpose of hashing is to look for a group of appropriate hash functions $h : \mathbb{R}^d \mapsto \{1, -1\}$[1], each of which accounts for the generation of a single hash bit. We use a kernel function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ to construct such hash functions because the kernel trick has been theoretically and empirically proven to be able to tackle practical data that is mostly linearly inseparable. Following the Kernelized Locality-Sensitive Hashing (KLSH) [6] algorithm, we first define a prediction function $f : \mathbb{R}^d \mapsto \mathbb{R}$ with the kernel κ plugged in as follows

$$f(\boldsymbol{x}) = \sum_{j=1}^{m} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}) a_j - b, \tag{1}$$

where $\boldsymbol{x}_{(1)}, \cdots, \boldsymbol{x}_{(m)}$ are m samples uniformly selected at random from $\mathcal{X}$, $a_j \in \mathbb{R}$ is the coefficient, and $b \in \mathbb{R}$ is the bias. Based on f, the hash function for a single hash bit is constructed by $h(\boldsymbol{x}) = \operatorname{sgn}(f(\boldsymbol{x}))$ in which the sign function $\operatorname{sgn}(x)$ returns 1 if input variable $x > 0$ and returns -1 otherwise. Note that m is fixed to a constant much smaller than the data set size n in order to maintain fast hashing.

An important criterion guiding the design of hash functions is that the generated hash bit should take as much information as possible, which implies a balanced hash function that meets $\sum_{i=1}^{n} h(\boldsymbol{x}_i) = 0$ [18][17][8]. For our problem, this balancing criterion makes b be the median of $\left\{\sum_{j=1}^{m} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i) a_j\right\}_{i=1}^{n}$. As a fast alternative to the median, we adopt the mean $b = \sum_{i=1}^{n} \sum_{j=1}^{m} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i) a_j / n$ like [17][2]. Through substituting b in eq. (1) with the mean value, we obtain

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_{j=1}^{m} \left(\kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}) - \frac{1}{n} \sum_{i=1}^{n} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i) \right) a_j \\ &= \boldsymbol{a}^\top \bar{\boldsymbol{k}}(\boldsymbol{x}), \end{aligned} \tag{2}$$

where $\boldsymbol{a} = [a_1, \cdots, a_m]^\top$ and $\bar{\boldsymbol{k}} : \mathbb{R}^d \mapsto \mathbb{R}^m$ is a vectorial map defined by

$$\bar{\boldsymbol{k}}(\boldsymbol{x}) = [\kappa(\boldsymbol{x}_{(1)}, \boldsymbol{x}) - \mu_1, \cdots, \kappa(\boldsymbol{x}_{(m)}, \boldsymbol{x}) - \mu_m]^\top, \tag{3}$$

[1] We treat '0' bit as '-1' during training; in the implementation of data coding and hashing, we use '0' back.

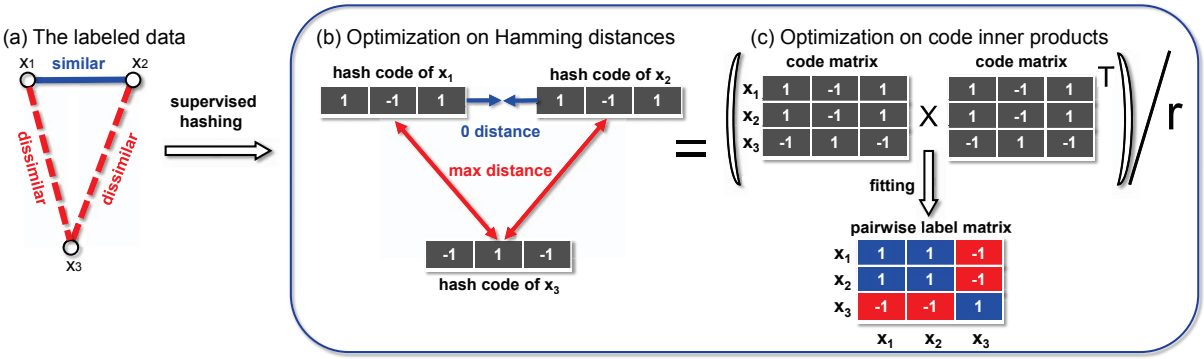

Figure 1. The core idea of our proposed supervised hashing. (a) Three data points with supervised pairwise labels: "1" (similar) and "-1" (dissimilar); (b) optimization on Hamming distances; (c) optimization on code inner products (r is the bit number). The latter two are equivalent due to the one-to-one correspondence between a Hamming distance and a code inner product.

in which $\mu_j = \sum_{i=1}^{n} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i)/n$ can be precomputed. In KLSH, the coefficient vector $\boldsymbol{a}$ came as a random direction drawn from a Gaussian distribution. Since $\boldsymbol{a}$ completely determines a hash function $h(\boldsymbol{x})$, we seek to learn $\boldsymbol{a}$ by leveraging supervised information so that the resulted hash function is discriminative.

2.2. Manipulating Code Inner Products

Suppose that r hash bits are needed. Accordingly, we have to find r coefficient vectors $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_r$ to construct r hash functions $\left\{h_k(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{a}_k^\top \bar{\boldsymbol{k}}(\boldsymbol{x}))\right\}_{k=1}^{r}$. In the customary setting for supervised hashing [5][13][17][11], the supervised information is given in terms of pairwise labels: 1 labels specify *similar* (or neighbor) pairs collected in set $\mathcal{M}$, and -1 labels designate *dissimilar* (or nonneighbor) pairs collected in set $\mathcal{C}$. Such pairs may be acquired from neighborhood structures in a predefined metric space, or from semantic relevancy when semantic-level labels of some samples are available. Without loss of generality, we assume that the first l ($m < l \ll n$) samples $\mathcal{X}_l = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l\}$ are implicated in $\mathcal{M}$ and $\mathcal{C}$. To explicitly record the pairwise relationships among $\mathcal{X}_l$, we define a label matrix $S \in \mathbb{R}^{l \times l}$ as

$$S_{ij} = \begin{cases} 1, & (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{M} \\ -1, & (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{C} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Note that $S_{ii} \equiv 1$ since $(\boldsymbol{x}_i, \boldsymbol{x}_i) \in \mathcal{M}$. The intermediate label 0 implies that the similar/dissimilar relationship about some data pair is unknown or uncertain. The 0 labels mostly appear in the metric-based supervision (see Sec 3).

Our purpose of supervised hashing is to generate discriminative hash codes such that similar pairs can be perfectly distinguished from dissimilar pairs by using Hamming distances in the code space. Specifically, we hope that the Hamming distances between the labeled pairs in $\mathcal{M} \cup \mathcal{C}$ correlate with the labels in S, that is, a pair with $S_{ij} = 1$ will have the minimal Hamming distance 0 while a pair with $S_{ij} = -1$ will take on the maximal Hamming distance, i.e., the number of hash bits r. Fig. 1(b) illustrates our expectation for optimizing the Hamming distances.

However, directly optimizing the Hamming distances is nontrivial because of the complex mathematical formula $\mathcal{D}_h(\boldsymbol{x}_i, \boldsymbol{x}_j) = |\{k | h_k(\boldsymbol{x}_i) \neq h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}|$. In this paper, we advocate *code inner products* that are easier to manipulate and optimize.

Code Inner Products vs. Hamming Distances

Let us write the r-bit hash code of sample $\boldsymbol{x}$ as $code_r(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \cdots, h_r(\boldsymbol{x})] \in \{1, -1\}^{1 \times r}$, and then deduce the code inner product as follows

$$\begin{aligned} & code_r(\boldsymbol{x}_i) \circ code_r(\boldsymbol{x}_j) \\ =& |\{k | h_k(\boldsymbol{x}_i) = h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}| \\ & - |\{k | h_k(\boldsymbol{x}_i) \neq h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}| \\ =& r - 2\,|\{k | h_k(\boldsymbol{x}_i) \neq h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}| \\ =& r - 2\mathcal{D}_h(\boldsymbol{x}_i, \boldsymbol{x}_j), \end{aligned} \tag{5}$$

where the symbol $\circ$ stands for the code inner product. Critically, eq. (5) reveals that the Hamming distance and the code inner product is in one-to-one correspondence, hence enabling equivalent optimization on code inner products.

Given the observation of $code_r(\boldsymbol{x}_i) \circ code_r(\boldsymbol{x}_j) \in [-r, r]$ and $S_{ij} \in [-1, 1]$, we let $code_r(\boldsymbol{x}_i) \circ code_r(\boldsymbol{x}_j)/r$ fit S_{ij} as shown in Fig. 1(c). This makes sense because $code_r(\boldsymbol{x}_i) \circ code_r(\boldsymbol{x}_j)/r = S_{ij} = 1$ leads to $\mathcal{D}_h(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$ and $code_r(\boldsymbol{x}_i) \circ code_r(\boldsymbol{x}_j)/r = S_{ij} = -1$ leads to $\mathcal{D}_h(\boldsymbol{x}_i, \boldsymbol{x}_j) = r$ by eq. (5). In a natural means, we propose a least-squares style objective function $\mathcal{Q}$ to learn the codes of the labeled data $\mathcal{X}_l$:

$$\min_{H_l \in \{1,-1\}^{l \times r}} \mathcal{Q} = \left\| \frac{1}{r} H_l H_l^\top - S \right\|_{\mathrm{F}}^2, \tag{6}$$

where $H_l = \begin{bmatrix} code_r(\boldsymbol{x}_1) \\ \cdots\cdots \\ code_r(\boldsymbol{x}_l) \end{bmatrix}$ denotes the code matrix of $\mathcal{X}_l$, and $\|.\|_{\mathrm{F}}$ represents the Frobenius norm.

We can generalize $\mathrm{sgn}()$ to take the elementwise sign operation for any vector or matrix input, and then express the code matrix H_l as (given $h_k(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{a}_k^\top \bar{\boldsymbol{k}}(\boldsymbol{x})) = \mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x})\boldsymbol{a}_k)$)

$$H_l = \begin{bmatrix} h_1(\boldsymbol{x}_1), \cdots, h_r(\boldsymbol{x}_1) \\ \cdots\cdots \\ h_1(\boldsymbol{x}_l), \cdots, h_r(\boldsymbol{x}_l) \end{bmatrix} = \mathrm{sgn}(\bar{K}_l A), \qquad (7)$$

where $\bar{K}_l = [\bar{\boldsymbol{k}}(\boldsymbol{x}_1), \cdots, \bar{\boldsymbol{k}}(\boldsymbol{x}_l)]^\top \in \mathbb{R}^{l\times m}$ and $A = [\boldsymbol{a}_1, \cdots, \boldsymbol{a}_r] \in \mathbb{R}^{m\times r}$. After substituting H_l in eq. (6) with eq. (7), we obtain an analytical form of the objective function $\mathcal{Q}$:

$$\min_{A\in\mathbb{R}^{m\times r}} \mathcal{Q}(A) = \left\| \frac{1}{r}\mathrm{sgn}(\bar{K}_l A)(\mathrm{sgn}(\bar{K}_l A))^\top - S \right\|_{\mathrm{F}}^2. \qquad (8)$$

The novel objective function $\mathcal{Q}$ is simpler than those of BRE [5] and MLH [11], because it offers a clearer connection and easier access to the model parameter A through manipulating code inner products. In contrast, BRE and MLH optimize Hamming distances by pushing them close to raw metric distances or larger/smaller than appropriately chosen thresholds, either of which formulated a complicated objective function and incurred a tough optimization process, yet cannot guarantee the optimality of its solution.

2.3. Greedy Optimization

The separable property of code inner products allows us to solve the hash functions in an incremental mode. With simple algebra, we rewrite $\mathcal{Q}$ in eq. (8) as

$$\min_{A} \quad \left\| \sum_{k=1}^{r} \mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k)(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top - rS \right\|_{\mathrm{F}}^2, \qquad (9)$$

where the r vectors $\boldsymbol{a}_k$'s, each of which determines a single hash function, are separated in the summation. This inspires a greedy idea for solving $\boldsymbol{a}_k$'s sequentially. At a time, it only involves solving one vector $\boldsymbol{a}_k$ provided with the previously solved vectors $\boldsymbol{a}_1^*, \cdots, \boldsymbol{a}_{k-1}^*$. Let us define a residue matrix $R_{k-1} = rS - \sum_{t=1}^{k-1} \mathrm{sgn}(\bar{K}_l \boldsymbol{a}_t^*)(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_t^*))^\top$ ($R_0 = rS$). Then $\boldsymbol{a}_k$ can be pursued by minimizing the following cost

$$\begin{aligned} &\left\|\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k)(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top - R_{k-1}\right\|_{\mathrm{F}}^2 \\ =& \left((\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top \mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k)\right)^2 \\ &- 2(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top R_{k-1}\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k) + \mathrm{tr}(R_{k-1}^2) \\ =& -2(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top R_{k-1}\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k) + l^2 + \mathrm{tr}(R_{k-1}^2) \\ =& -2(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top R_{k-1}\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k) + const. \end{aligned} \qquad (10)$$

Discarding the constant term, we arrive at a cleaner cost

$$g(\boldsymbol{a}_k) = -(\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k))^\top R_{k-1}\mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k). \qquad (11)$$

A nice feature is that $g(\boldsymbol{a}_k)$ is lower-bounded as eq. (10) is always nonnegative. However, minimizing g is not easy to achieve because it is neither convex nor smooth. In what follows, we propose two optimization schemes to approximately minimize g.

Spectral Relaxation. Motivated by the spectral methods for hashing [18][8], we apply the spectral relaxation trick to drop the sign functions involved in g, resulting in a constrained quadratic problem

$$\begin{aligned} \max_{\boldsymbol{a}_k} \ & (\bar{K}_l \boldsymbol{a}_k)^\top R_{k-1}(\bar{K}_l \boldsymbol{a}_k) \\ \text{s.t.} \ & (\bar{K}_l \boldsymbol{a}_k)^\top (\bar{K}_l \boldsymbol{a}_k) = l \end{aligned} \qquad (12)$$

where the constraint $(\bar{K}_l \boldsymbol{a}_k)^\top (\bar{K}_l \boldsymbol{a}_k) = l$ makes l elements in the vector $\bar{K}_l \boldsymbol{a}_k$ fall into the range of $[-1, 1]$ roughly, so that the solution of the relaxed problem eq. (12) is in the similar range to the original problem eq. (11). Eq. (12) is a standard generalized eigenvalue problem $\bar{K}_l^\top R_{k-1} \bar{K}_l \boldsymbol{a} = \lambda \bar{K}_l^\top \bar{K}_l \boldsymbol{a}$, and $\boldsymbol{a}_k$ is thus sought as the eigenvector associated with the largest eigenvalue. A proper scaling is conducted on the solved eigenvector, saved as $\boldsymbol{a}_k^0$, to satisfy the constraint in eq. (12).

Although spectral relaxation results in fast optimization, it might deviate far away from the optimal solution under larger l (e.g., $\geq$ 5,000) due to the amplified relaxation error (see Sec 4.2). It is therefore used as the initialization to a more principled optimization scheme described below.

Sigmoid Smoothing. Since the hardness of minimizing g lies in the sign function, we replace $\mathrm{sgn}()$ in g with the sigmoid-shaped function $\varphi(x) = 2/(1+\exp(-x)) - 1$ which is sufficiently smooth and well approximates $\mathrm{sgn}(x)$ when $|x| > 6$. Afterward, we propose to optimize the smooth surrogate $\tilde{g}$ of g:

$$\tilde{g}(\boldsymbol{a}_k) = -(\varphi(\bar{K}_l \boldsymbol{a}_k))^\top R_{k-1}\varphi(\bar{K}_l \boldsymbol{a}_k), \qquad (13)$$

where $\varphi()$ operates elementwisely like $\mathrm{sgn}()$. The gradient of $\tilde{g}$ with respect to $\boldsymbol{a}_k$ is derived as

$$\nabla\tilde{g} = -\bar{K}_l^\top \left((R_{k-1}\boldsymbol{b}) \odot (\boldsymbol{1} - \boldsymbol{b} \odot \boldsymbol{b})\right), \qquad (14)$$

where the symbol $\odot$ represents the Hadamard product (i.e., elementwise product), $\boldsymbol{b} = \varphi(\bar{K}_l \boldsymbol{a}_k) \in \mathbb{R}^l$, and $\boldsymbol{1}$ denotes a constant vector with l 1 entries.

Since the original cost g is lower-bounded, its smooth surrogate $\tilde{g}$ is lower-bounded as well. Consequently, we are capable of minimizing $\tilde{g}$ using the regular gradient descent technique. Note that the smooth surrogate $\tilde{g}$ is still nonconvex, so it is unrealistic to look for a global minima of $\tilde{g}$. For fast convergence, we adopt the spectral relaxation solution $\boldsymbol{a}_k^0$ as a warm start and apply Nesterov's gradient method [10] to accelerate the gradient decent procedure. In most cases we can attain a locally optimal $\boldsymbol{a}_k^*$ at which $\tilde{g}(\boldsymbol{a}_k^*)$ is very close to its lower bound, which will be corroborated by the subsequent experiments.

Algorithm 1 Kernel-Based Supervised Hashing (KSH)

Input: a training sample set $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, a pairwise label matrix $S \in \mathbb{R}^{l \times l}$ defined on l samples $\mathcal{X}_l = \{\boldsymbol{x}_i\}_{i=1}^l$, a kernel function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$, the number of support samples m ($< l$), and the number of hash bits r.

Preprocessing: uniformly randomly select m samples from $\mathcal{X}$, and compute zero-centered m-dim kernel vectors $\bar{\boldsymbol{k}}(\boldsymbol{x}_i)$ ($i = 1, \cdots, n$) using the kernel function κ according to eq. (3).

Greedy Optimization:

initialize $R_0 = rS$ and $T_{max} = 500$;

for $k = 1, \cdots, r$ **do**

solve the generalized eigenvalue problem $\bar{K}_l^\top R_{k-1} \bar{K}_l \boldsymbol{a} = \lambda \bar{K}_l^\top \bar{K}_l \boldsymbol{a}$, obtaining the largest eigenvector $\boldsymbol{a}_k^0$ such that $(\boldsymbol{a}_k^0)^\top \bar{K}_l^\top \bar{K}_l \boldsymbol{a}_k^0 = l$;

use the gradient descent method to optimize $\min_{\boldsymbol{a}} -(\varphi(\bar{K}_l \boldsymbol{a}))^\top R_{k-1} \varphi(\bar{K}_l \boldsymbol{a})$ with the initial solution $\boldsymbol{a}_k^0$ and T_{max} budget iterations, achieving $\boldsymbol{a}_k^*$;

$\boldsymbol{h}^0 \longleftarrow \mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k^0)$, $\boldsymbol{h}^* \longleftarrow \mathrm{sgn}(\bar{K}_l \boldsymbol{a}_k^*)$;

if $(\boldsymbol{h}^0)^\top R_{k-1} \boldsymbol{h}^0 > (\boldsymbol{h}^*)^\top R_{k-1} \boldsymbol{h}^*$ **then**

$\boldsymbol{a}_k^* \longleftarrow \boldsymbol{a}_k^0$, $\boldsymbol{h}^* \longleftarrow \boldsymbol{h}^0$;

end if

$R_k \longleftarrow R_{k-1} - \boldsymbol{h}^* (\boldsymbol{h}^*)^\top$;

end for

Coding: for $i = 1, \cdots, n$, do

$code_r(\boldsymbol{x}_i) \longleftarrow \left[\mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_i)\boldsymbol{a}_1^*), \cdots, \mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_i)\boldsymbol{a}_r^*)\right]$.

Output: r hash functions $\{h_k(\boldsymbol{x}) = \mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x})\boldsymbol{a}_k^*)\}_{k=1}^r$ as well as n hash codes $\mathcal{H} = \{code_r(\boldsymbol{x}_i)\}_{i=1}^n$.

Finally, we describe the whole flowchart of the proposed supervised hashing approach that we name *Kernel-Based Supervised Hashing* (KSH) in Algorithm 1. We also name another approach KSH^0 whose hash functions just use the initial spectral relaxation solutions $\{\boldsymbol{a}_k^0\}$.

3. Analysis

Our approaches KSH^0 and KSH can both deal with semantic and metric supervision once the definitions about similar and dissimilar pairs are offered to learning. For example, a similar pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is confirmed if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ share at least one common semantic label or are nearest neighbors to each other under a predefined metric (e.g., ℓ_2); likewise, a dissimilar pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is determined if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ take different labels or are far away in the metric space.

In the semantic case, one can easily achieve the full entries (either 1 or -1) of the label matrix S since the implicated samples are all known to have semantic labels. But in the metric case, one needs to pre-compute two distance thresholds, one for similar pairs and the other for dissimilar pairs, to judge if two samples are metric neighbors or not by comparing their distance with the thresholds [18][5][17]. For the metric supervision, most entries in the label matrix S have 0 labels, which reveals that most distances fall into the middle ground between the two thresholds. To reduce the potential false alarms, our approaches implicitly push the Hamming distances of these 0-labeled pairs to $r/2$ as their code inner products have been pushed to zeros (see eq. (5)), which is reasonable since such pairs are not nearest neighbors in the metric space.

The time complexities for training KSH^0 and KSH are both bounded by $O((nm+l^2m+m^2l+m^3)r)$ which scales linearly with n given $n \gg l > m$. In practice, training KSH^0 is very fast and training KSH is about two times faster than two competing supervised hashing methods BRE and MLH. For each query, the hashing time of both KSH^0 and KSH is constant $O(dm + mr)$.

4. Experiments

We run large-scale image retrieval experiments on two image benchmarks: CIFAR-10[2] and one million subset of the 80 million tiny image collection [15]. CIFAR-10 is a labeled subset of the 80M tiny images, consisting of a total of 60K 32×32 color images from ten object categories each of which contains 6K samples. Every image in this dataset is assigned to a mutually exclusive class label and represented by a 512-dimensional GIST feature vector [12]. The second dataset that we call Tiny-1M was acquired from [17], which does not include any semantic label but has been partially annotated to similar and dissimilar data pairs according to the ℓ_2 distance. Each image in Tiny-1M is represented by a 384-dimensional GIST vector.

We evaluate the proposed KSH^0 and KSH, and compare them against eight state-of-the-art methods including four unsupervised methods LSH [1], PCAH [17], SH [18], and KLSH [6], one semi-supervised method SSH (the nonorthogonal version) [17], and three supervised methods LDAH [14], BRE [5], and MLH [11]. We used the publicly available codes of SH, KLSH, BRE and MLH. All our experiments were run on a workstation with a 2.53 GHz Intel Xeon CPU and 48GB RAM.

Since KLSH, KSH^0 and KSH refer to kernels, we feed them the same Gaussian RBF kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|/2\sigma^2)$ and the same $m = 300$ support samples on each dataset. The kernel parameter σ is tuned to an appropriate value on each dataset. It is noted that we did not assume a specific kernel, and that any kernel satisfying the Mercer's condition can be used in KLSH, KSH^0 and KSH. We follow two search procedures *hash lookup* and *Hamming ranking* using 8 to 48 hash bits. Hash lookup takes constant search time over a single hash table for each compared hashing method. We carry out hash lookup within a Hamming radius 2 and report the search precision. Note that we follow [17] to treat failing to find any hash bucket for a query as zero precision, different from [18][5][11] which ignored the failed queries in computing the mean hash lookup precision over all queries. To quantify the failing cases, we report the

[2] `http://www.cs.toronto.edu/~kriz/cifar.html`

Table 1. Ranking performance on CIFAR-10 (60K). l denotes the number of labeled examples for training (semi-)supervised hashing methods. Four unsupervised methods LSH, PCAH, SH and KLSH do not use any labels. All training/test time is recorded in second.

Method	$l = 1,000$					$l = 2,000$				
	MAP			**Train Time**	**Test Time**	**MAP**			**Train Time**	**Test Time**
	12 bits	24 bits	48 bits	48 bits	48 bits	12 bits	24 bits	48 bits	48 bits	48 bits
LSH [1]	0.1133	0.1245	0.1188	0.5	0.8×10^{-5}	—				
PCAH [17]	0.1368	0.1333	0.1271	1.5	0.9×10^{-5}	—				
SH [18]	0.1330	0.1317	0.1352	3.0	4.0×10^{-5}	—				
KLSH [6]	0.1212	0.1425	0.1602	1.6	4.3×10^{-5}	—				
SSH [17]	0.1514	0.1595	0.1755	2.1	0.9×10^{-5}	0.1609	0.1758	0.1841	2.2	0.9×10^{-5}
LDAH [14]	0.1380	0.1334	0.1267	0.7	0.9×10^{-5}	0.1379	0.1316	0.1257	1.1	0.9×10^{-5}
BRE [5]	0.1817	0.2024	0.2060	494.7	2.9×10^{-5}	0.1847	0.2024	0.2074	1392.3	3.0×10^{-5}
MLH [11]	0.1545	0.1932	0.2074	3666.3	1.8×10^{-5}	0.1695	0.1953	0.2288	3694.4	2.0×10^{-5}
KSH^0	0.1846	0.2047	0.2181	7.0	3.3×10^{-5}	0.2271	0.2461	0.2545	9.4	3.5×10^{-5}
KSH	**0.2325**	**0.2588**	**0.2836**	156.1	4.3×10^{-5}	**0.2700**	**0.2895**	**0.3153**	564.1	4.5×10^{-5}

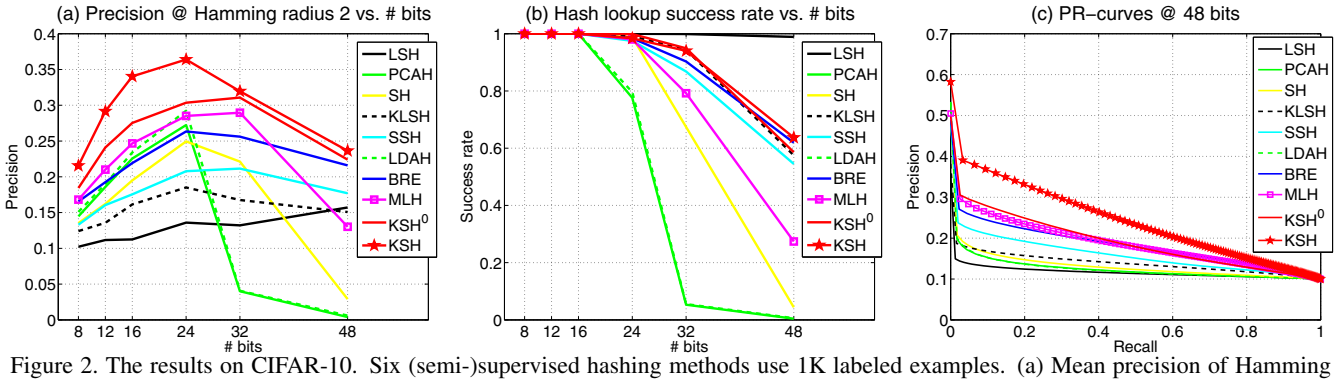

Figure 2. The results on CIFAR-10. Six (semi-)supervised hashing methods use 1K labeled examples. (a) Mean precision of Hamming radius 2 hash lookup; (b) hash lookup success rate; (c) mean precision-recall curves of Hamming ranking with 48 bits.

hash lookup success rate: the proportion of the queries resulting in successful hash lookup in all queries. As well, Hamming ranking, fast enough with short hash codes in practice, measures the search quality through ranking the retrieved samples according to their Hamming distances to a specific query.

4.1. CIFAR-10

This dataset is partitioned into two parts: a training set of 59K images and a test query set of 1K images evenly sampled from ten classes. We uniformly randomly sample 100 and 200 images from each class respectively, constituting 1K and 2K labeled subsets for training (semi-)supervised hashing methods SSH, LDAH, BRE, MLH, KSH^0 and KSH. The pairwise label matrices S are immediately acquired since the exact labels are available. To run BRE and MLH that admit 1,0 labels, we assign the labels of similar pairs to 1 and those of dissimilar pairs to 0. In terms of true semantic neighbors, we report the mean precision of Hamming radius 2 hash lookup, the success rate of hash lookup, the mean average precision (MAP) of Hamming ranking, and the mean precision-recall curves of Hamming ranking over 1K query images. All of the evaluation results are shown in Table 1 and Fig. 2. For every compared method, we also report the training time for compressing all training images into compact codes as well as the test time for coding each query image.

As shown in Table 1 and Fig. 2, KSH achieves the highest search accuracy (hash lookup precision, MAP, and PR-curve) and the second best is KSH^0. The gain in MAP of KSH ranges from 27% to 46% over the best competitor except KSH^0. The prominent superiority of KSH corroborates that the proposed hashing objective $\mathcal{Q}$ and two optimization techniques including spectral relaxation and sigmoid smoothing are so successful that the semantic supervision information is maximally utilized. In the hash lookup success rate, KSH is lower than LSH but still superior to the others. More notably, KSH with only 48 binary bits and a limited amount of supervised information (1.7% and 3.4% labeled samples) significantly outperforms ℓ_2 linear scan (0.1752 MAP) in the GIST feature space, accomplishing up to 1.8 times higher MAP. Compared to BRE and MLH, KSH^0 (several seconds) and KSH (several minutes) are much faster in supervised training. The test time of KSH^0 and KSH is acceptably fast, comparable to the non-linear hashing methods SH, KLSH and BRE.

4.2. Tiny-1M

The one million subset of the 80 million tiny image benchmark [15] was sampled to construct the training set,

Table 2. Ranking performance on Tiny-1M. l denotes the number of pseudo-labeled examples for training (semi-)supervised hashing methods. Four unsupervised methods LSH, PCAH, SH and KLSH do not use any labels. All training/test time is recorded in second.

Method	$l = 5,000$					$l = 10,000$				
	MP/50K			**Train Time**	**Test Time**	**MP/50K**			**Train Time**	**Test Time**
	12 bits	24 bits	48 bits	48 bits	48 bits	12 bits	24 bits	48 bits	48 bits	48 bits
LSH	0.1107	0.1421	0.1856	3.0	0.3×10^{-5}	—				
PCAH	0.2371	0.2159	0.1954	7.1	0.4×10^{-5}	—				
SH	0.2404	0.2466	0.2414	47.1	3.3×10^{-5}	—				
KLSH	0.1834	0.2490	0.3008	9.9	2.6×10^{-5}	—				
SSH	0.1985	0.2923	0.3785	14.8	0.6×10^{-5}	0.1718	0.2767	0.3524	16.9	0.6×10^{-5}
LDAH	0.2365	0.2208	0.2077	5.8	0.6×10^{-5}	0.2373	0.2238	0.2072	13.3	0.6×10^{-5}
BRE	0.2782	0.3400	0.3961	18443.0	3.3×10^{-5}	0.2762	0.3403	0.3889	27580.0	3.3×10^{-5}
MLH	0.2071	0.2592	0.3723	4289.2	1.4×10^{-5}	0.1875	0.2873	0.3489	4820.8	1.8×10^{-5}
KSH^0	0.1889	0.2295	0.2346	56.0	3.1×10^{-5}	0.1886	0.1985	0.2341	84.5	3.2×10^{-5}
KSH	**0.3164**	**0.3896**	**0.4579**	2210.3	3.2×10^{-5}	**0.3216**	**0.3929**	**0.4645**	2963.3	3.3×10^{-5}

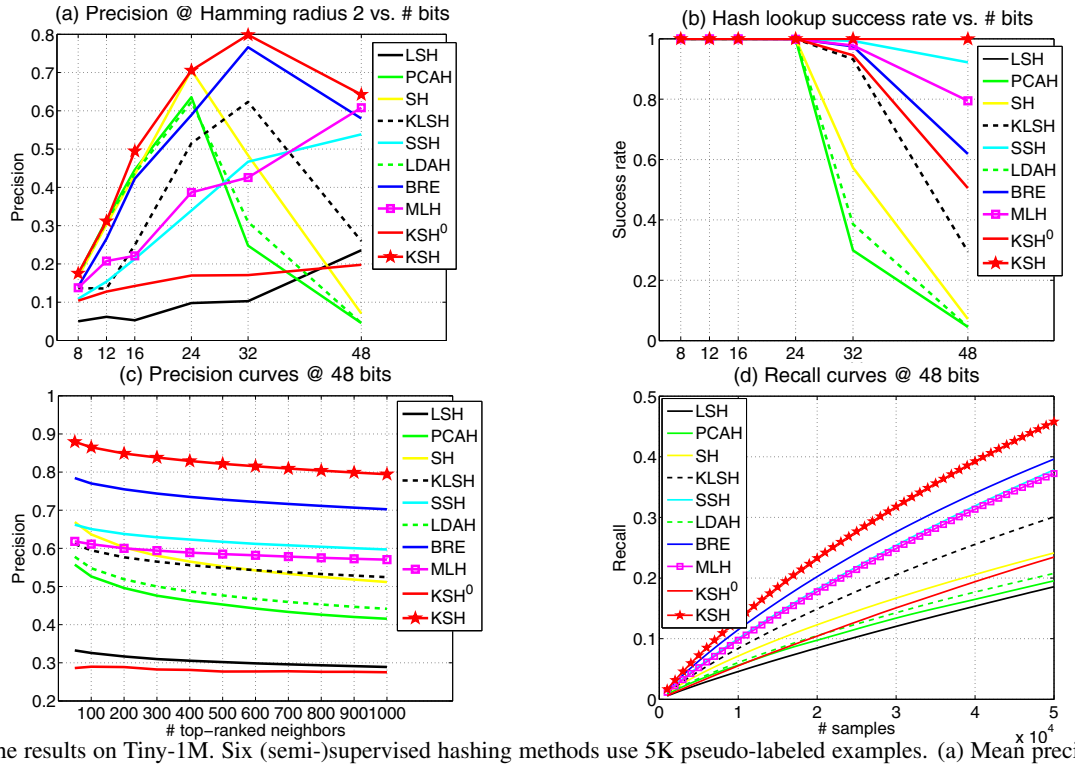

Figure 3. The results on Tiny-1M. Six (semi-)supervised hashing methods use 5K pseudo-labeled examples. (a) Mean precision of Hamming radius 2 hash lookup; (b) hash lookup success rate; (c) mean precision curves of Hamming ranking with 48 bits; (d) mean recall curves of Hamming ranking with 48 bits.

and a separate subset of 2K images was used as the test set [17]. The 10K×10K pairwise pseudo label matrix S was constructed according to the ℓ_2 distance. Concretely, [17] randomly selected 10K images from the training set and computed their Euclidean distance matrix D from which S was obtained by using the rule: $S_{ij} = 1$ if D_{ij} is within 5% of the whole 1M distances and $S_{ij} = -1$ if D_{ij} is more than 95%. The top 5% distances from a query were also used as the groundtruths of nearest metric neighbors. As most entries in S are zeros, to follow [5] each 0 label is replaced by $1 - \hat{D}_{ij}/2$ in which $0 \leq \hat{D}_{ij} \leq 2$ is the normalized ℓ_2 distance. Like above experiments, we treat 1,-1 labels in S as 1,0 labels for running BRE and MLH.

In terms of ℓ_2 metric neighbors (each query has 50K groundtruth neighbors), we evaluate the mean precision of Hamming radius 2 hash lookup, the success rate of hash lookup, the mean top-50K precision (MP/50K) of Hamming ranking, and the mean precision/recall curves of Hamming ranking. The results are shown in Table 2 and Fig. 3. To illustrate the overfitting phenomenon in (semi-)supervised hashing methods, we inspect the half supervision, i.e., the 5K pseudo-labeled images, and the full 10K labeled images, respectively. KSH refrains from overfitting, showing higher MP when absorbing more supervision. On the contrary,

Figure 4. Top six neighbors of four query images, returned by SSH, BRE, MLH and KSH using 5K pseudo-labeled training images and 48 binary bits on the Tiny-1M dataset.

SSH, BRE and MLH all suffer from overfitting to different extents – their MP drops faced with increased supervision.

Again, we can see that KSH consistently attains the highest search accuracy (hash lookup precision, MP, P-curve, and R-curve) and the same highest hash lookup success rate as LSH. The gain in MP of KSH ranges from 13% to 19% over the best competitor. Referring to Sec 2.3, the spectral relaxation solutions employed by KSH^0 might become poor when l is larger, which is verified in these experiments where KSH^0 performs as poorly as LDAH. It is noticeable that KSH with only 48 binary bits and a very limited amount of supervised information (0.5% and 1% pseudo-labeled samples) can retrieve about 46% groundtruth ℓ_2 neighbors (see Table 2) and reach higher precision by using longer bits. Therefore, we can say that KSH well preserves the ℓ_2 metric similarities in the Hamming code space. Besides the quantitative evaluations, Fig. 4 showcases some exemplar query images and their retrieved neighbors with 48 bits, where KSH still exhibits the best search quality in visual relevance. To obtain the exact rank of the retrieved neighbors, we perform ℓ_2 linear scan in a short list of top 0.01% Hamming ranked neighbors like [5]. Such search time is close to real time, costing only 0.06 second per query.

5. Conclusions

We accredit the success of the proposed KSH approach to three primary aspects: 1) kernel-based hash functions were exploited to handle linearly inseparable data; 2) an elegant objective function designed for supervised hashing was skillfully formulated based on code inner products; and 3) a greedy optimization algorithm was deployed to solve the hash functions efficiently. Extensive image retrieval results shown on large image corpora up to one million have demonstrated that KSH surpasses the state-of-the-art hashing methods by a large margin in searching both metric neighbors and semantic neighbors. We thus believe that hashing via optimizing code inner products is a promising direction, generating clearly higher quality than the hashing methods relying on optimizing Hamming distances.

Last but not least, although in this paper we select image retrieval as the testbed of the proposed hashing approach, we want to further highlight that it is a general method and can be applied to a large spectrum of information retrieval problems such as duplicate document detection.

Acknowledgements

This work is supported in part by a Facebook fellowship to the first author.

References

[1] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. SoCG*, 2004.

[2] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. CVPR*, 2011.

[3] H. Jegou, M. Douze, C. Schmid, and P. Perez. Aggregating local descriptors into a compact image representation. In *Proc. CVPR*, 2010.

[4] S. Korman and S. Avidan. Coherency sensitive hashing. In *Proc. ICCV*, 2011.

[5] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS 22*, 2009.

[6] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *TPAMI*, 2012.

[7] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *TPAMI*, 31(12):2143–2157, 2009.

[8] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proc. ICML*, 2011.

[9] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *Proc. CVPR*, 2010.

[10] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2003.

[11] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proc. ICML*, 2011.

[12] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001.

[13] R. Salakhutdinov and G. Hinton. Semantic hashing. *Int. J. Approximate Reasoning*, 50(7):969–978, 2009.

[14] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *TPAMI*, 34(1):66–78, 2012.

[15] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *TPAMI*, 30(11):1958–1970, 2008.

[16] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large databases for recognition. In *Proc. CVPR*, 2008.

[17] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large scale search. *TPAMI*, 2012.

[18] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS 21*, 2008.

[19] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proc. ICCV*, 2011.