kNN Hashing with Factorized Neighborhood Representation

Kun Ding, Chunlei Huo, Bin Fan, Chunhong Pan
National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences, Beijing, P.R. China
Email:{kding,clhuo,bfan,chpan}@nlpr.ia.ac.cn

Abstract

Hashing is very effective for many tasks in reducing the processing time and in compressing massive databases. Although lots of approaches have been developed to learn data-dependent hash functions in recent years, how to learn hash functions to yield good performance with acceptable computational and memory cost is still a challenging problem. Based on the observation that retrieval precision is highly related to the kNN classification accuracy, this paper proposes a novel kNN-based supervised hashing method, which learns hash functions by directly maximizing the kNN accuracy of the Hamming-embedded training data. To make it scalable well to large problem, we propose a factorized neighborhood representation to parsimoniously model the neighborhood relationships inherent in training data. Considering that real-world data are often linearly inseparable, we further kernelize this basic model to improve its performance. As a result, the proposed method is able to learn accurate hashing functions with tolerable computation and storage cost. Experiments on four benchmarks demonstrate that our method outperforms the state-of-the-arts.

1. Introduction

With the rapidly increasing amount of available data, such as image, text and video, fast similarity search and efficient data storage/indexing techniques become increasingly important in many tasks, *e.g*., matching [28], retrieval [29], video segmentation [23] and graph construction [35]. Traditional methods, for example kd-tree [2], are inefficient in dealing with tens of thousands high-dimensional data. Hashing [1, 6, 22, 26, 32], as an emerging and popular technique, enables both fast search and efficient storage by mapping high-dimensional data to low-dimensional binary codes while maintaining some similarity and structural information of the original data.

Many hashing methods are proposed in the literature. Early ones are data-independent [1, 3, 13]. For example, locality-sensitive hashing (LSH) [1] uses random projections as the hash functions. To preserve the geometry of high-dimensional data, lots of hash bits are needed as it does not explore the data distribution when choosing the hashing functions. Recent researches are mainly focused on developing data-dependent or learning-based hashing to automatically learn compact binary codes from data. In general, according to the level of supervision, the learning-based hashing can be categorized into unsupervised [6, 7, 10, 21, 27, 32], semi-supervised [31] and supervised [6, 12, 16, 17, 18, 22, 26] methods. Among them, supervised hashing has attracted extensive attention due to its superiority on preserving the semantic similarity.

Perhaps the most intuitive idea to learn hash functions is to preserve the (dis-)similarity defined in input space [6, 12, 22, 32]. Inspired by this, binary reconstruction embedding (BRE) [12] explicitly preserves the pairwise distances between data, while kernel-based supervised hashing (KSH) [22] learns binary codes to preserve semantic similarity. An alternative objective for hashing is maximizing the precision of a certain classifier applied in Hamming space. Lin *et al*. proposed FastHash [18] to learn hash functions for high-dimensional data with a two-step strategy, where the second step comes down to training binary classifiers, *e.g*., decision trees. Although this framework is simple and flexible, the decomposition may lead to suboptimal performance. More recently, Shen *et al*. improved FastHash by proposing the supervised discrete hashing (SDH) [26] to generate binary codes with the minimal classification loss of a linear classifier. It shows great advantages over the similarity-preserving based hashing paradigm. However, as retrieval is essentially a neighbor search problem, optimizing the accuracy of a kNN classifier might be more reasonable. Therefore, in this paper, we formulate the problem of learning hash functions by directly maximizing the kNN classification accuracy on the binary embeddings of training data. To tackle the problem of discreteness of kNN accuracy, the stochastic neighborhood representation [5, 8, 24] is resorted to define a surrogate objective function. By approximating the involved sign function with a smooth function, we finally obtain a tractable unconstrained problem.

Expensive computational and memory cost in training have long been plaguing many learning-based hashing methods, such as BRE and KSH. To address such potential difficulties in our model, we propose a tractable factorized stochastic neighborhood representation to model the neighborhood structure inherent in data. Compared to the full Gaussian affinity based representations [5, 8, 24], it is more economical for computing and storing. With this representation, an approximate lower bound of the original objective function is constructed favorably. By optimizing on this bound, desirable hash functions can be generated without expensive computational and memory cost.

Real-world data is often linearly inseparable, and it has been shown that the hashing performance for such data could be dramatically boosted by using nonlinear hash functions [13, 22, 26]. A simple yet effective approach for nonlinear hashing learning is utilizing kernel tricks. Here, we also use this technique to extend our basic model to the nonlinear case so as to further improve its performance.

To sum up, our main contributions are: 1) A kNN-based hashing (kNNH) is proposed to improve the existing retrieval performance, and a relaxed objective function is devised to speedup the training procedure and reduce the memory cost; 2) A systematic analysis of approximating the sign function with different kinds of smooth functions, whose influence on hashing learning has been largely ignored in previous methods; 3) A kernelized version of kNNH is proposed to further improve its performance.

2. The Proposed Method

2.1. Formulation

Let our training data be represented by a set of N D-dimensional vectors, given by $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \in \mathbb{R}^{D\times N}$. The associated labels are $\mathbf{y} = [y_1, \ldots, y_N]$ whose entries are in $\mathcal{Y} = \{1, \ldots, C\}$, where C is the number of classes. Our target is to learn K hyperplane hash functions $\{h_k(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}_k^T\mathbf{x})\}_{k=1}^K$, where $\mathbf{w}_k$ is the hyperplane of the k-th hash function and $\mathrm{sgn}(\cdot)$ is the sign function. The binary code for a sample $\mathbf{x}$ is $h(\mathbf{x}) = [h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})]^T = \mathrm{sgn}(\mathbf{W}^T\mathbf{x}) \in \mathbb{H}^K$ with $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D\times K}$ the projection matrix and $\mathbb{H}^K$ the K-dimensional Hamming space.

Different from the existed supervised hashing techniques, we propose to learn hash functions by maximizing the kNN classification accuracy which is closely related to the retrieval precision. The core idea of our method is illustrated in Fig. 1. For a test sample $\mathbf{x}_i$ with the label y_i, its probability of correct classification under the kNN rule applied in $\mathbb{H}^K$ is $\pi_i = \sum_{j=1}^N s_{ij}\delta(j \in \mathcal{N}_{k_t}(\mathbf{x}_i))/k_t$. Here, s_{ij} is 1 if $y_i = y_j$, and 0 otherwise. $\mathcal{N}_{k_t}(\mathbf{x}_i)$ is the set of k_t nearest neighbors of $h(\mathbf{x}_i)$ according to Hamming distance, and so $\delta(j \in \mathcal{N}_{k_t}(\mathbf{x}_i))$ is an indicator function showing whether j is among the k_t-NN of $h(\mathbf{x}_i)$. From the retrieval point of view, when k_t items are returned for the query $\mathbf{x}_i$ by Hamming ranking, its retrieval precision (*i.e.*, the fraction of retrieved relevant samples) is also π_i. Therefore, if the classification accuracy of kNN is optimized, both the retrieval performance (*i.e.*, precision) and the classification performance (w.r.t. the kNN classifier) could be ensured.

However, the direct optimization of π_i is hard because the neighbors of every datum in $\mathbb{H}^K$ constantly change with $\mathbf{W}$. The neighborhood component analysis (NCA) [5] overcame this problem by using the stochastic neighborhood representation. We borrow this idea and define the stochastic neighbors for each sample in $\mathbb{H}^K$. More specifically, the probability that $\mathbf{x}_i$ chooses $\mathbf{x}_j$ as its neighbor is

$$\pi_{ij} = \frac{\exp\big(-2\theta^2\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|_H\big)}{\sum_{j=1}^N \exp\big(-2\theta^2\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|_H\big)}, \quad (1)$$

where $\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|_H$ denotes the Hamming distance between $h(\mathbf{x}_i)$ and $h(\mathbf{x}_j)$, which counts the number of different bits between them. θ is a parameter controlling the number of neighbors that affect $h(\mathbf{x}_i)$: when $\theta \to 0$, all samples are viewed as its neighbor; when $\theta \to \infty$, only $h(\mathbf{x}_i)$ itself is viewed as the neighbor. Considering the relation $\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|_H = 1/2K - 1/2h^T(\mathbf{x}_i)h(\mathbf{x}_j)$, one can convert π_{ij} into

$$\pi_{ij} = \frac{\exp\big(\theta^2 h^T(\mathbf{x}_i)h(\mathbf{x}_j)\big)}{\sum_{j=1}^N \exp\big(\theta^2 h^T(\mathbf{x}_i)h(\mathbf{x}_j)\big)}. \quad (2)$$

Based on the above definition of the stochastic neighbors, π_i can be approximated as $\pi_i \approx \sum_{j=1}^N s_{ij}\pi_{ij}$. As done in NCA, we use the Kullback-Leibler divergence to measure the kNN classification accuracy:

$$J_o(\mathbf{W}) = \sum\nolimits_{i=1}^N \ln(\pi_i). \quad (3)$$

One major limitation of this model is that in order to evaluate $J_o(\mathbf{W})$ w.r.t. $\mathbf{W}$, a $N \times N$ normalized similarity matrix $\mathbf{\Pi} = [\pi_{ij}]$ must be calculated and stored, which is impractical for large N. Note that, NCA [5] and other related algorithms [8, 24, 30] also suffer from this problem.

2.2. Factorized Neighborhood Representation

To make the above model flexible to large problem, we propose to factorize $\mathbf{\Pi}$ as the production of two smaller parametric similarity matrices, *i.e.*, $\mathbf{\Pi} = \mathbf{P}\mathbf{Q}^T$. The entries of $\mathbf{P} = [p_{im}] \in \mathbb{R}^{N\times M}$ and $\mathbf{Q} = [q_{jm}] \in \mathbb{R}^{N\times M}$ are defined as

$$p_{im} = \frac{\exp\big(\theta^2 h^T(\mathbf{x}_i)h(\mathbf{z}_m)\big)}{\sum_{m=1}^M \exp\big(\theta^2 h^T(\mathbf{x}_i)h(\mathbf{z}_m)\big)}, \quad (4)$$

$$q_{jm} = \frac{\exp\big(\theta^2 h^T(\mathbf{x}_j)h(\mathbf{z}_m)\big)}{\sum_{j=1}^N \exp\big(\theta^2 h^T(\mathbf{x}_j)h(\mathbf{z}_m)\big)}, \quad (5)$$

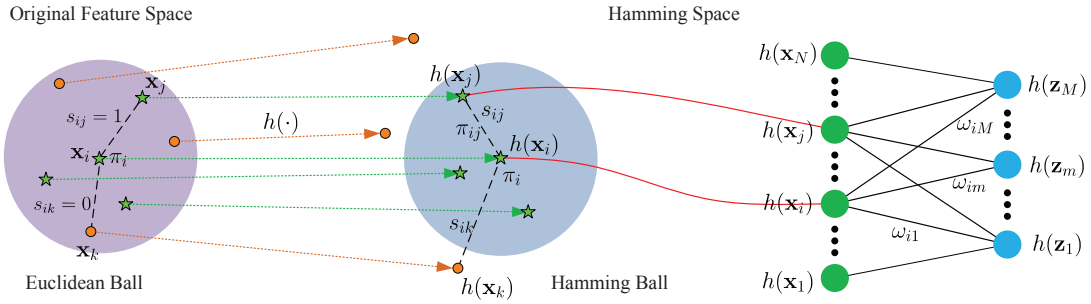


Figure 1. The core idea of kNN hashing. In the leftmost panel, s_{ij} represents the pairwise semantic relation between $\mathbf{x}_i$ and $\mathbf{x}_j$. In the middle panel, the optimal hash functions are learned so that the neighbors of every sample in $\mathbb{H}^K$ are as pure as possible. In the rightmost panel, the normalized similarity π_{ij} is defined as the two-step transition probability from the i-th node to the j-th node.

where $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_M] \in \mathbb{R}^{D\times M}$ ($M \ll N$) can be viewed as the prototypes or anchors that should be learned. $\mathbf{P}$ and $\mathbf{Q}$ reflect the relation between samples and prototypes, with which the similarity between every pair of samples is defined implicitly. To utilize the supervisory information, we pre-assign a label $\hat{y}_m \in \mathcal{Y}$ for each anchor $\mathbf{z}_m$, and keep them fixed throughout. In this paper, we simply adopt the balanced assignment of labels, *i.e.*, every M/C (C is the number of classes) anchors share a same label. More complex strategies, *e.g.* unbalanced assignment, can also be used, which we would leave for future work.

The above factorized neighborhood representation is reasonable and meaningful due to the following reasons: 1) The factorized form of $\mathbf{\Pi}$ meets the nonnegative normalization constraints ($\sum_{j=1}^{N}\pi_{ij} = 1$ and $\pi_{ij} \geq 0$) that are implied in Eqn. (2). These can be checked by applying the nonnegativity and normalization property of $\mathbf{P}$ and $\mathbf{Q}$. 2) $\pi_{ij} = \sum_{m=1}^{M} p_{im}q_{jm}$ can be seen as the inner product of a specific kernel-mapped features of $h(\mathbf{x}_i)$ and $h(\mathbf{x}_j)$, which makes it a reasonable description of the underlying similarity. 3) As shown in Fig. 1, the decomposed $\mathbf{\Pi}$ has a probabilistic explanation: it can be seen as the two-step transition probability matrix of the nodes defined on $\{h(\mathbf{x}_i)\}_{i=1}^{N}$.

To further explain this point, we introduce a bipartite graph $\mathcal{B}(\mathcal{V},\mathcal{U},\mathcal{E})$, where each element of $\mathcal{V} = \{v_i\}_{i=1}^{N}$ and $\mathcal{U} = \{u_m\}_{m=1}^{M}$ represent the Hamming embeddings of a training sample and a prototype, respectively, and $\mathcal{E}$ contains the edges between $\mathcal{V}$ and $\mathcal{U}$. The nodes v_i and u_m are connected by an undirected edge with the weight $\omega_{im} = \exp(\theta^2 h^T(\mathbf{x}_i)h(\mathbf{z}_m))$. The full adjacency matrix of $\mathcal{B}$ is $\mathbf{B} = [\mathbf{0}, \mathbf{\Omega}; \mathbf{\Omega}^T, \mathbf{0}] \in \mathbb{R}^{(M+N)\times(M+N)}$ with $\mathbf{\Omega} = [\omega_{im}] \in \mathbb{R}^{N\times M}$. Over $\mathcal{B}$, we build stationary Markov random walks with the one-step transition probability matrix $\mathbf{P}^{(1)} = \mathbf{D}^{-1}\mathbf{B}$, where $\mathbf{D}$ is a diagonal matrix whose diagonal entries are the row sums of $\mathbf{B}$. By this, the transition probabilities in one step time are $p^{(1)}_{m|i} = p_{im}$, $p^{(1)}_{j|m} = q_{jm}$ and $p^{(1)}_{j|i} = p^{(1)}_{i|j} = 0$. About the two-step transition probability $p^{(2)}_{j|i}$, we have the following proposition.

Proposition 1. *Given the one-step transition probability matrix $\mathbf{P}^{(1)}$ of $\mathcal{B}$, the two-step transition probabilities are*

$$p^{(2)}_{j|i} = \pi_{ij},\ i,j = 1,\ldots,N. \tag{6}$$

The proof of this proposition is supplied in supplementary. It is reasonable to see that the designed neighborhood representation matrix $\mathbf{\Pi}$ describes the probabilities of transition from one sample node to another in two steps. The idea of factorizing a full similarity matrix into two smaller similarity matrices has been applied in Liu *et al*. [20, 21]. However, their approaches do not aim to design the neighborhood representation matrix that should be *singly stochastic*, but to design the adjacency matrix for semi-supervised learning that is *doubly stochastic*. Moreover, the anchors for constructing $\mathbf{\Pi}$ used in the work are learned in a supervised manner instead of being simply set by K-means as they did.

With the factorization form of $\mathbf{\Pi}$, we have the following theorem about $J_o(\mathbf{W})$.

Theorem 1. *Given the non-negative similarity matrices $\mathbf{\Pi}$, $\mathbf{P}$ and $\mathbf{Q}$ that satisfy $\mathbf{\Pi} = \mathbf{P}\mathbf{Q}^T$, the objective function $J_o(\mathbf{W})$ is lower bounded by*

$$J_\alpha(\mathbf{W},\mathbf{Z}) = \sum\nolimits_{i=1}^{N}\ln(p_i) + \sum\nolimits_{m=1}^{M}\alpha_m\ln(q_m), \tag{7}$$

where $p_i = \sum_{m=1}^{M} r_{im}p_{im}$, $q_m = \sum_{j=1}^{N} r_{jm}q_{jm}$, $\alpha_m = \sum_{i=1}^{N} r_{im}p_{im}/p_i$; r_{im} is 1 if $y_i = \hat{y}_m$ and 0 otherwise.

Please refer to the supplementary material for the proof of this theorem. Actually, the first term in Eqn. (7) is to maximize the classification accuracy of the *training samples* by using the learned prototypes; while the second term is to maximize the classification accuracy of the *learned prototypes* by using the training samples. α_m can be seen as trade-off parameters. For simplicity, we fix them as an identical positive constant α, which results in the kNN hashing model (kNNH)

$$J(\mathbf{W},\mathbf{Z}) = \sum\nolimits_{i=1}^{N}\ln(p_i) + \alpha\sum\nolimits_{m=1}^{M}\ln(q_m). \tag{8}$$

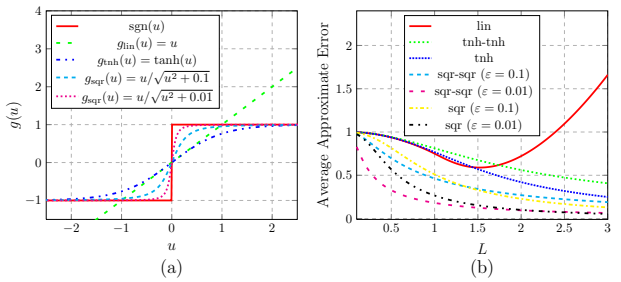


Figure 2. (a) Different approximation functions of $\mathrm{sgn}(\cdot)$; (b) The average approximation error of different functions $\psi(u, v)$. In (b), 'lin' denotes the linear approximation $g_{\text{lin}}(u)g_{\text{lin}}(v) = g_{\text{lin}}(uv)$; 'tnh' and 'sqr' denote the holistic approximations $g_{\text{tnh}}(uv)$ and $g_{\text{sqr}}(uv)$, respectively; 'tnh-tnh' and 'sqr-sqr' denote the separable approximations $g_{\text{tnh}}(u)g_{\text{tnh}}(v)$ and $g_{\text{sqr}}(u)g_{\text{sqr}}(v)$, respectively.

By this new objective function, the lower bound of $J_o(\mathbf{W})$ is sufficiently approximated given the proper α. The time and space complexity of maximizing Eqn. (8) are both $O(N)$, which are much lower than that of maximizing $J_o(\mathbf{W})$ directly, *i.e.*, $O(N^2)$.

Although the first term in Eqn. (8) is similar to the stochastic neighborhood compression (SNC) model [14], it is worthy to note that Eqn. (8) is essentially used to approximate Eqn. (3), which is basically different from SNC. Moreover, kNNH could learn prototypes both in $\mathbb{R}^D$ and $\mathbb{H}^K$. By contrast, SNC and many other prototype learning methods [33, 36] only learn prototypes in $\mathbb{R}^D$.

2.3. Approximation of the Sign Function

Due to the sign function, the objective function $J(\mathbf{W}, \mathbf{Z})$ given in Eqn. (8) is non-smooth, making its optimization a big challenge. In general, there are two strategies to solve this kind of problem: two-step optimization and continuous relaxation. The former one decouples the original problem into two separate tractable subproblems [17, 18]. Although the decoupling allows fast training, it losses the global optimality. The latter replaces the sign function by a smooth function [22, 31], which could lead to good solution if the relaxation is appropriate. In this paper, we adopt the second scheme and obtain a tractable unconstrained optimization problem finally. As different approximation functions have different approximation abilities and computational complexities, it is necessary to compare them in detail.

Since p_{im} and q_{jm} only depend on the product $h(u, v) = h_k(\mathbf{x})h_k(\mathbf{z}) = \mathrm{sgn}(u)\,\mathrm{sgn}(v)$ with $u = \mathbf{w}_k^T\mathbf{x}$ and $v = \mathbf{w}_k^T\mathbf{z}$, we only need to approximate it with some smooth function $\psi(u, v)$. Generally, there are two types of functions to achieve this goal: $\psi(u, v) = g(u)g(v)$ and $\psi(u, v) = g(uv)$. Many functions can be used as $g(\cdot)$: $g_{\text{lin}}(u) = u$, $g_{\text{tnh}}(u) = \tanh(u)$, and $g_{\text{sqr}}(u) = u/\sqrt{u^2 + \varepsilon}$ with ε a positive constant. As shown in Fig. 2(a), all these odd functions are smooth and strictly increasing, which are reasonable approximations of $\mathrm{sgn}(\cdot)$. Different from the other two, $g_{\text{sqr}}(u)$ has a parameter ε that can regulate the quality of approximation.

Since all reasonable approximation functions $\psi(u, v)$ have the same *minimal approximation error* 0 when $u = v = 0$, it is more meaningful to evaluate the *mean approximation error*. Assume that the ℓ_2-norm of samples and prototypes have the upper bound L_1, *i.e.*, $\|\mathbf{x}_i\|_2 \le L_1$, $i = 1, \ldots, N$, $\|\mathbf{z}_m\|_2 \le L_1$, $m = 1, \ldots, M$, and $\mathbf{w}_k$ satisfies $\|\mathbf{w}_k\|_2 \le L_2$, $k = 1, \ldots, K$. Based on the Cauchy inequality, we have $|u| = |\mathbf{w}_k^T\mathbf{x}| \le \|\mathbf{w}_k\|_2\|\mathbf{x}\|_2 \le L_1L_2 \triangleq L$, which also holds for v. Within the domain $[-L, L] \times [-L, L]$ that (u, v) is defined, the average approximation error of $h(u, v)$ by $\psi(u, v)$ is defined as

$$e(L) = \frac{1}{4L^2}\int_{-L}^{L}\int_{-L}^{L} |h(u, v) - \psi(u, v)|\, du\, dv. \qquad (9)$$

The analytical formulations of $e(L)$ for different functions are given in the supplementary file. To intuitively inspect the approximation error, Fig. 2(b) shows how it changes with L for different functions, from which it is clear that the error of linear approximation is unbounded. Even though additional constraints could be imposed on Eqn. (8) to make $L = 1.5$ (the extreme point), the obtained minimal average error is still too large (*i.e.*, 0.589 per bit), which will lead to serious performance degradation. As a result, the linear approximation is actually not a good choice.

All the other approximations have a limited upper bound of error (*i.e.*, 1.0), and their approximation errors decrease with the increasing L but with different rates. Among these approximations, $g_{\text{sqr}}(u)g_{\text{sqr}}(v)$ and $g_{\text{sqr}}(uv)$ with sufficiently small ε can acquire the lowest errors with different L. However, as they may reduce the smoothness of the objective function and make the subsequent optimization problematic, the performance cannot be guaranteed. In other words, when using them we should set a proper ε. From the implementation viewpoint, it is more desirable to use the separable approximation $g(u)g(v)$, as the derivatives of the objective function can be expressed in compact matrix forms conveniently. According to the above analysis, we finally use $g_{\text{tnh}}(u)g_{\text{tnh}}(v)$ to approximate $h(u, v)$ in our model. By this approximation, maximizing the objective function in Eqn. (8) will be a smooth and unconstrained problem, which can be solved by the famous limited-memory BFGS (L-BFGS) algorithm [19]. Please refer to the supplemental material for detailed derivations of the derivatives.

2.4. Kernelized kNN Hashing

Following KSH [22] and kernelized LSH [13], by using kernel tricks, kNNH can better deal with linearly inseparable data. To this end, we define a prediction function $f_k : \mathbb{R}^D \mapsto \mathbb{R}$ for each hash function $h_k(\cdot)$ with the kernel

$\kappa : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$ plugged in as follows

$$f_k(\mathbf{x}) = \sum\nolimits_{m=1}^{M} \kappa(\mathbf{z}_m, \mathbf{x}) a_{mk} - b_k, \tag{10}$$

where $\mathbf{z}_1, \ldots, \mathbf{z}_M$ are the anchors, $\kappa(\cdot, \cdot)$ is a Mercer kernel function, a_{mk} is the coefficient and b_k is the bias. On the basis of $f_k(\mathbf{x})$, the k-th hash function is $h_k(\mathbf{x}) = \mathrm{sgn}(f_k(\mathbf{x}))$. Similar tricks have been used in [12, 13, 22], however, the anchors therein are randomly selected and kept fixed during the hash function learning procedure. By contrast, we learn these anchors from the training data in a supervised manner, which could better capture the data distribution, and thus make the hash functions be more powerful.

For simplicity, we set every bias to be 0. So that, the prediction function for the k-th hash function has a simpler form $f_k(\mathbf{x}) = \mathbf{a}_k^T \mathbf{k}(\mathbf{x})$, where $\mathbf{a}_k = [a_{1k}, \ldots, a_{Mk}]^T$ and $\mathbf{k}(\mathbf{x}) = [\kappa(\mathbf{z}_1, \mathbf{x}), \ldots, \kappa(\mathbf{z}_M, \mathbf{x})]^T$. By this, the Hamming embedding of $\mathbf{x}$ in $\mathbb{H}^K$ is $h(\mathbf{x}) = \mathrm{sgn}(f(\mathbf{x}))$, where $f(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})]^T = \mathbf{A}^T \mathbf{k}(\mathbf{x})$ with $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_K] \in \mathbb{R}^{M \times K}$. By using the separable approximation of the sign function, the inner product in p_{im} can be expressed as

$$h^T(\mathbf{x}_i) h(\mathbf{z}_m) \approx g\big(f^T(\mathbf{x}_i)\big) g\big(f(\mathbf{z}_m)\big). \tag{11}$$

The kernelization enables the learned prototypes to capture the nonlinearity in data. In the kernel version, we learn $\mathbf{A}$ instead of $\mathbf{W}$. Thus, we denote the objective function in Eqn. (8) as $J(\mathbf{A}, \mathbf{Z})$, and call this method as kernelized kNN hashing (k^2NNH). The optimization of k^2NNH is similar to the linear case except for more complicated derivations of derivatives caused by wrapping $\mathbf{Z}$ in the kernel functions, which are detailed in the supplemental material.

3. Experiments

3.1. Methods and Evaluation Protocols

We evaluate kNNH and k^2NNH on three small benchmarks (Mnist [15], CIFAR10 [11] and SUN397 [34]) and a large benchmark (ILSVRC14 [4]). They are compared against other popular unsupervised (ITQ [6] and AGH [21]), semi-supervised (SSH [31]), and supervised (BRE, ITQCCA [6], KSH, CGH [16] and SDH) methods covering both linear (ITQ, ITQCCA, SSH and linCGH (linear CGH)) and nonlinear (AGH, BRE, KSH, kerCGH (CGH with Gaussian kernel) and SDH) ones. For our methods, M is set to $30C$, $30C$, $3C$ and $3C$ (C is the number of classes) on Mnist, CIFAR10, SUN397 and ILSVRC14, respectively; α is set to $10^{-4}N/M$; θ is set as $(s_\theta \mathbb{E}[\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|_H])^{-1}$ where $\mathbb{E}[\cdot]$ denotes the expectation and s_θ is set between 0 and 1. In k^2NNH, the Gaussian kernel $\kappa(\mathbf{z}, \mathbf{x}) = \exp(-\frac{1}{2}\gamma^2 \|\mathbf{z} - \mathbf{x}\|_2^2)$ is used, where γ is estimated by $(s_\gamma \mathbb{E}[\|\mathbf{z}^0 - \mathbf{x}\|_2])^{-1}$ with $\mathbf{z}^0$ the initial prototype obtained by K-means and s_γ set between 0 and 1. For the compared methods, the number of anchors is set to M if any, and other parameters are set according to their authors for a fair comparison. We conduct experiments on both retrieval and recognition.

To evaluate the retrieval quality, we report the following results: mean average precision (MAP), precision curves within Hamming radius 2 using hash lookup, precision curves w.r.t. different number of top returned images, precision-recall curves.

To evaluate the classification performance of different hashing methods, the kNN classification precisions are reported, which are obtained by classifying the test data in Hamming space. Except for the above hashing methods, other related data compression techniques are also compared, which include: 1) Using all training data (Full) as baseline; 2) Learning M/C prototypes per class by optimizing the 1-nearest prototype classifier (ONPC) [33]; 3) Learning M/C prototypes per class by SNC [14]. With the compressed training sets produced by these methods, the kNN classification precisions are computed. The k of the kNN classifier is obtained by 5-fold cross validation.

3.2. Datasets

We use the 784-dimensional image gray value and the 512-dimensional GIST [25] as the feature for Mnist and CIFAR10, respectively. On SUN397 and ILSVRC14, we first extract the convolutional features (ConvFeat) on 'fc7' layer by the pre-trained CaffeNet [9] and then reduce the features to 512 dimensions by PCA.

On Mnist and CIFAR10, we randomly select 100 images per class as the queries and the remaining images are taken as the database. For ITQ and AGH we use the database for training. For the supervised methods, we randomly select 1K images per class from the database as labeled images for training. For SSH, the 10K labeled images combined with the remaining unlabeled images in the database are used for training. On SUN397 and ILSVRC14, the data splitting and the usage are similar, but the size of the query set and the labeled training subset are 10, 30 images per class for SUN397 and 10, 10 images per class for ILSVRC14.

3.3. Results

Retrieval. The retrieval results on three small datasets are reported in Table 2 and Fig. 3(a-c). From Table 2, we can see that in most cases k^2NNH acquires the highest MAP. With 64-bit binary code, the improvements of MAP over KSH, kerCGH and SDH are at least 0.04, 0.04 and 0.05 on Mnist, CIFAR10 and SUN397, respectively. For the linear case, kNNH has a comparable MAP to that of ITQCCA, both of which outperform other linear methods.

The hash lookup precisions shown in Fig. 3(a) reveal that k^2NNH outperforms most methods when search speed is the first concern in which case the used bits are as few as possible. Even with more bits, k^2NNH still returns lots of

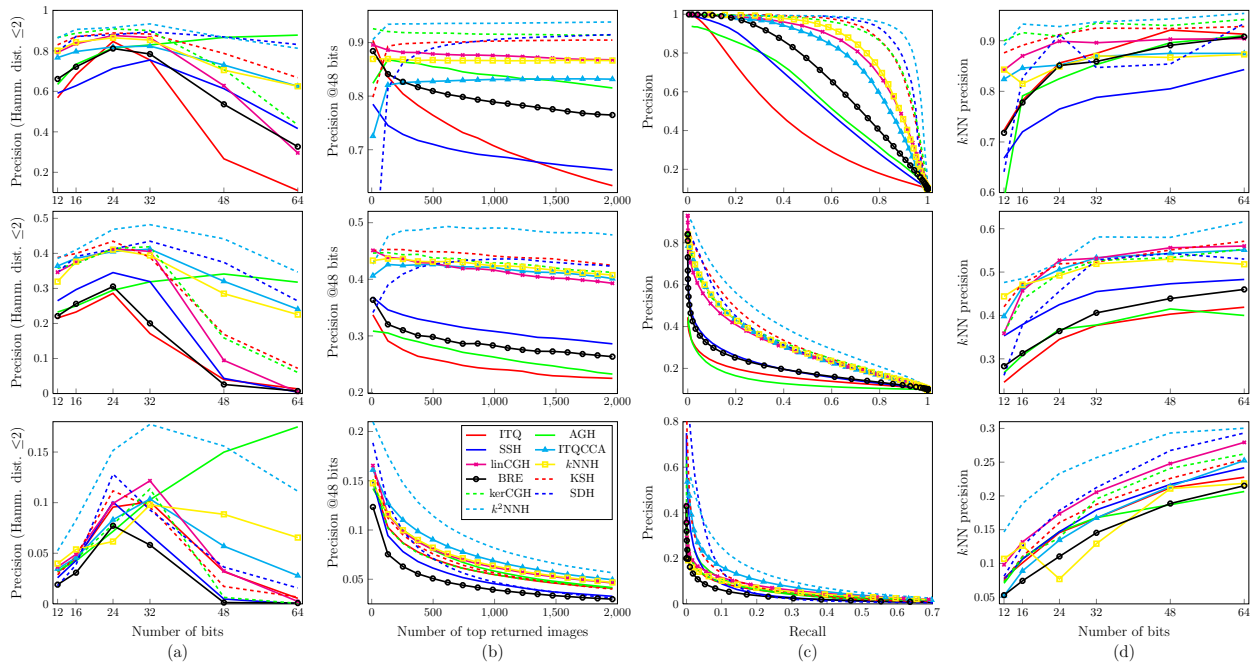


Figure 3. Retrieval and classification performance of different hashing methods on Mnist, CIFAR10 and SUN397 (from top to bottom). (a) Precision within Hamming radius 2 using hash lookup with different number of hash bits; (b) Precision curves with 48 bits w.r.t. different number of top returned images; (c) precision-recall curves with 48 bits; (d) kNN classification precision with various hash bits.

Method	12 bits	16 bits	24 bits	32 bits	48 bits	64 bits
ITQCCA	0.0338	0.0434	0.0629	0.0803	0.1076	0.1322
kNNH	0.0515	0.0620	0.0655	0.0804	0.1077	0.1265
KSH	0.0333	0.0434	0.0615	0.0834	0.1115	0.1316
kerCGH	0.0400	0.0471	0.0540	0.0584	0.0797	0.0949
SDH	0.0333	0.0501	0.0805	0.1083	0.1486	0.1757
k^2NNH	**0.1010**	**0.1289**	**0.1529**	**0.1655**	**0.1786**	**0.1825**

Table 1. MAP of different hashing methods on ILSVRC14.

correct neighbors by hash lookup. This demonstrates that k^2NNH can map similar samples to the closer buckets as many as possible. From Fig. 3(b), k^2NNH is much better than the other methods on Mnist and CIFAR10 when the top returned images are relatively more, *e.g.*, 50+, and it acquires the highest Hamming ranking precisions with different number of returned images on SUN397. From Fig. 3(c), k^2NNH acquires the highest precisions for most recalls.

To further demonstrate the effectiveness of our methods, we also conduct the experiments on ILSVRC14. The results are shown in Table 1. From this table, kNNH is as good as ITQCCA, if not better. For nonlinear case, k^2NNH obtains the highest MAP on all bits, which is significantly better than other methods. Thus, by maximizing the kNN accuracy and learning anchors, our method does learn good hash functions.

Exemplar-based recognition. The need of storing all training exemplars and the costly brute-force neighbor search are two bottlenecks of the kNN-based object/scene recognition. The classical prototype learning [14, 33] is a common technique in reducing the storage and computation of the kNN classifier. Hashing, as a new data reduction method, has seldom been explored in this field except for a few scattered works [14, 29]. This part will systematacially compare the ability of these two groups of methods in the context of kNN classification. The comparison is carried out in terms of kNN precision, storage cost and test time.

The kNN precision of different hashing methods with different hash bits are shown in Fig. 3(d), and the comprehensive evaluation of hashing and prototype leaning is given in Table 3. In Table 3, the storage includes the cost for data pre-processing (*e.g.*, saving the mean vector), storing hash functions and the compressed training samples (in $\mathbb{R}^N$ or $\mathbb{H}^K$) as well as their labels. Here, we use 16-bit integer, 8-bit integer and double to store the labels, the binary features and the other data, respectively. The test time consists of the time for pre-processing the query feature (*e.g.*, subtracting the mean), Hamming encoding and 1NN classification.

From Fig. 3(d), the kNN classification precision of k^2NNH outperforms all other hashing methods. From Table 3, we have the following observations: 1) In general, prototype leaning (ONPC and SNC) is indeed a good data compression technique for kNN—high precision, relatively low storage and fast test speed. 2) Although the current hashing methods have shown effectiveness in retrieval, most of them are not proficient in the exemplar-based recognition. In other words, their precision, storage and test speed

Method	Mnist (70K, 784-Pixel)				CIFAR10 (60K, 512-GIST)				SUN397 (108K, 512-ConvFeat)			
	MAP			Time(s)	MAP			Time(s)	MAP			Time(s)
	24 bits	48 bits	64 bits	48 bits	24 bits	48 bits	64 bits	48 bits	24 bits	48 bits	64 bits	48 bits
ITQ	0.4286	0.4399	0.4473	9.0	0.1696	0.1731	0.1742	5.9	0.0414	0.0577	0.0641	**10.6**
AGH	0.6508	0.6256	0.6185	393.9	0.1638	0.1542	0.1525	220.0	0.0578	0.0732	0.0779	1377.1
SSH	0.5368	0.5659	0.5878	189.3	0.2114	0.2172	0.2128	199.1	0.0335	0.0457	0.0482	336.1
ITQCCA	0.7477	0.7682	0.7723	18.9	0.3039	0.3232	0.3278	8.2	0.0546	0.0861	0.1009	34.9
linCGH	0.7869	0.8042	0.7994	2526.9	0.2989	0.3129	0.3122	2217.4	0.0496	0.0697	0.0788	1942.3
kNNH	0.7944	0.8165	0.8149	549.0	0.2968	0.3300	0.3222	332.3	0.0412	0.0772	0.0808	816.1
BRE	0.6099	0.6571	0.6696	48345.9	0.1869	0.2058	0.2147	29599.8	0.0211	0.0345	0.0414	44784.6
KSH	0.8555	0.8771	0.8780	6637.0	0.3222	0.3485	0.3613	6419.2	0.0408	0.0628	0.0774	14090.1
kerCGH	**0.8638**	0.8802	0.8899	2224.1	0.3104	0.3366	0.3402	2260.3	0.0439	0.0562	0.0653	2701.6
SDH	0.8597	0.8924	0.8974	**5.9**	0.3148	0.3385	0.3431	**5.1**	0.0399	0.0703	0.0815	15.4
k^2NNH	0.8615	**0.9288**	**0.9395**	359.1	**0.3487**	**0.4005**	**0.4108**	368.7	**0.0944**	**0.1243**	**0.1327**	1689.1

Table 2. MAP and training time of different hashing methods on Mnist, CIFAR10 and SUN397.

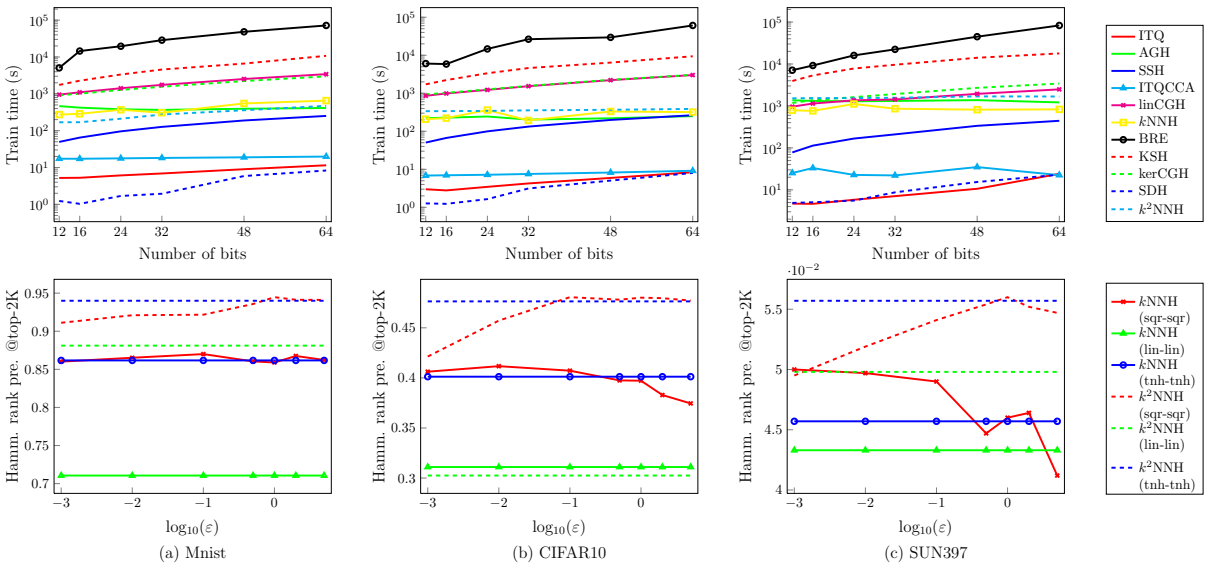
Method	Mnist (70K, 784-Pixel)				CIFAR10 (60K, 512-GIST)				SUN397 (108K, 512-ConvFeat)			
	kNN Pre.(%)		Str.(KB)	Time(μs)	kNN Pre.(%)		Str.(KB)	Time(μs)	kNN Pre.(%)		Str.(KB)	Time(μs)
	48 bits	64 bits	48 bits	48 bits	48 bits	64 bits	48 bits	48 bits	48 bits	64 bits	48 bits	48 bits
Full	94.60		61269.5	255.0	52.40		40019.5	198.3	33.58		47663.3	167.7
ONPC	95.00		1838.1	16.0	59.00		1200.6	18.9	38.21		4766.3	21.3
SNC	94.80		1838.1	18.7	58.40		1200.6	18.3	**40.15**		4766.3	20.5
ITQ	92.10	91.30	378.3	103.6	40.30	41.90	274.1	104.2	21.21	22.77	289.1	112.7
AGH	89.60	91.00	1972.3	141.8	41.50	40.00	1334.8	127.9	18.69	20.65	5080.7	177.8
SSH	80.50	84.30	734.3	107.3	47.30	48.30	519.6	100.7	21.69	24.16	534.6	112.5
ITQCCA	87.50	87.50	378.3	101.4	54.30	55.10	274.1	99.7	21.41	25.24	289.1	111.6
linCGH	90.30	90.40	385.1	112.8	55.60	56.00	278.9	109.1	24.79	27.93	293.8	121.8
kNNH	86.70	87.30	**302.5**	**10.5**	53.00	51.80	**198.4**	**7.2**	21.08	21.86	**205.3**	**16.0**
BRE	89.10	90.80	13165.3	238.5	43.90	46.00	8630.3	191.3	18.87	21.51	8785.3	180.0
KSH	92.40	92.80	2030.5	104.9	55.10	57.10	1393.0	106.0	22.59	25.42	5313.0	135.1
kerCGH	93.00	94.20	2043.5	132.3	53.30	55.30	1401.7	118.6	24.08	26.22	5321.7	154.5
SDH	85.40	93.40	2028.5	113.8	54.30	53.00	1391.0	115.9	26.73	29.32	5304.1	138.4
k^2NNH	94.30	**95.40**	1958.5	23.9	58.00	**61.60**	1318.9	15.0	29.32	30.03	5224.0	38.6

Table 3. kNN classification performance, *i.e.*, precision (kNN Pre.), storage (Str.), and test time per sample, of different methods.

are less satisfactory than prototype learning. 3) By directly modeling on the kNN classification, our proposed kNNH acquires the fastest test speed, the lowest storage and moderate precision on all datasets. Thus, it is very suitable for large-scale data or the applications on low-memory devices. These advantages are owing to the fact that kNNH combined prototype learning and hashing learning in an effective way. 4) Compared to the prototype learning, k^2NNH acquires higher classification precisions on Mnist and CIFAR10 with comparable storage and test time, demonstrating the effectiveness of kernel strategy. On SUN397, although all the hashing methods perform worse than prototype learning, the classification precision gaps of k^2NNH to prototype learning are remarkably lower than others.

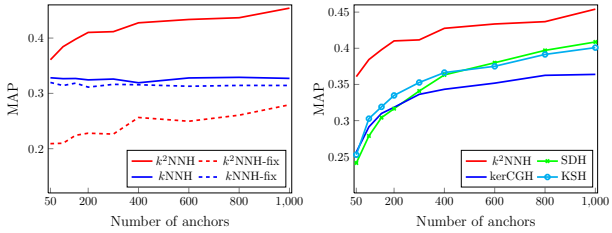
Training time. Many prior supervised hashing methods are slow in training due to the operations on the $N \times N$ similarity matrix. For example, for about 10K training samples, BRE needs 20K+ seconds and KSH needs 6K+ seconds to learn 48-bit hash functions (*c.f*. Table 2), limiting their ability to process larger dataset effectively. By contrast, with the help of neighborhood factorization, our methods can easily deal with the large-size dataset. For example, we only need less than 400s for training on CIFAR10 for both kNNH and k^2NNH, which is 16 times faster than KSH and 78 times faster than BRE. Furthermore, from the first row of Fig. 4, compared to many other methods, the training times of kNNH and k^2NNH grow slowly with K. Although kNNH and k^2NNH are slower than ITQCCA and SDH in training, k^2NNH achieves much better retrieval and recognition performance (*c.f*. Table 2 and Table 3) and they spend much less time for classification (*c.f*. Table 3).

Different approximations to sign function. The Hamming ranking precisions with different approximation functions are shown in the second row of Fig. 4. These curves validate that a proper approximation of the sign function is important. Even though the 'lin-lin' approximation is simple and commonly-used, its large approximation error leads to poor performance. The 'sqr-sqr' approximation can afford different approximation accuracies, however, it needs selecting the best ε. The 'tnh-tnh' approximation acquires the comparable performance to the best 'sqr-sqr' approximation, and it does not have additional parameter to set.



(a) Mnist (b) CIFAR10 (c) SUN397

Figure 4. Training time and the effect of different approximation functions on three methods. First row: training time; Second row: Hamming ranking precision of top-2K ranked neighbors with different approximation functions $\psi(u, v)$. Note that ε is only used in 'sqr-sqr', and so other methods have constant results.



(a) (b)

Figure 5. Anchor type and anchor number. (a) Learned anchors vs fixed anchors; (b) MAP with varying anchor number.

Performance	Method	16 bits	24 bits	32 bits	48 bits
Train. Time(s)	linNCA	1100.9	1114.6	1113.1	1124.3
	kNNH	222.3	362.4	193.5	332.3
MAP	linNCA	0.2485	0.2824	0.3055	0.3259
	kNNH	0.2645	0.2968	0.3161	0.3300

Table 4. The performance of linNCA and kNNH on CIFAR10.

Effectiveness of neighborhood factorization. The training time and MAP of directly solving the original objective function in Eqn. (3) (called linNCA) and solving the relaxed objective function in Eqn. (8) (*i.e.*, kNNH) are compared in Table 4. We can see that kNNH achieves comparable MAP to linNCA with much shorter training time for most hash bits. We found that the training time of linNCA decreases with the increased hash bits as it practically needs fewer iterations to converge when the number of hash bits increases. It is worth to note that when the number of training samples is further increased, the computation time of linNCA in each iteration will largely increase while that of kNNH is less influenced due to the factorized neighborhood representation. Moreover, linNCA is also impractical for large training set considering its high memory cost.

Different anchor types and anchor numbers. The effect of anchor type and number is studied on CIFAR10 and the results are shown in Fig. 5. From Fig. 5(a), one can note that, in the nonlinear case, picking anchors by K-means (k^2NNH-fix) only achieves a half MAP of that obtained by learning them in k^2NNH. For the linear case, optimizing anchors also helps, even not very much. The results of k^2NNH-fix are worse than those of linear models might because the quality of feature mapping relies heavily on the anchor quality. From Fig. 5(b), it is obvious that k^2NNH could achieve equal MAP to other methods with much fewer anchors. For instance, with only 200 anchors, k^2NNH reaches the MAP of SDH using 1000 anchors. Thus, its query time and storage cost reduce to 1/5 of SDH.

4. Conclusions

This paper proposes a novel hashing method that learns hash functions by optimizing the kNN accuracy of the binary embeddings of the training data. By introducing a factorized neighborhood system, it scales well to large problems. By extending the linear model to the nonlinear case using kernel tricks, the performance is significantly improved. Experiments on retrieval and recognition demonstrate that the proposed method outperforms prior hashing methods.

5. Acknowledgments

This work was supported in part by the Projects of the National Natural Science Foundation of China (Grant No. 61375024, 91438105, 61203277, 91338202) and the Beijing Natural Science Foundation (Grant No. 4142057).

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 2008.

[2] J. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 1975.

[3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Ann. Symp. Comput. Geometry*, 2004.

[4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[5] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2004.

[6] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 2013.

[7] K. He, F. Wen, and J. Sun. K-Means hashing: An affinity-preserving quantization method for learning binary compact codes. In *CVPR*, 2013.

[8] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *NIPS*, 2002.

[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. ACM Int. Conf. Multimedia*, 2014.

[10] Z. Jin, Y. Hu, Y. Lin, D. Zhang, S. Lin, D. Cai, and X. Li. Complementary projection hashing. In *ICCV*, 2013.

[11] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

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

[13] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[14] M. Kusner, S. Tyree, K. Weinberger, and K. Agrawal. Stochastic neighbor compression. In *ICML*, 2014.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[16] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. Dick. Learning hash functions using column generation. In *ICML*, 2013.

[17] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *ICCV*, 2013.

[18] G. Lin, C. Shen, and A. van den Hengel. Supervised hashing using graph cuts and boosted decision trees. *TPAMI*, 2015.

[19] D. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 1989.

[20] W. Liu, J. He, and S. Chang. Large graph construction for scalable semi-supervised learning. In *ICML*, 2010.

[21] W. Liu, J. Wang, and S. Chang. Hashing with graphs. In *ICML*, 2011.

[22] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[23] X. Liu, D. Tao, M. Song, Y. Ruan, C. Chen, and J. Bu. Weakly supervised multiclass video segmentation. In *CVPR*, 2014.

[24] L. Maaten and G. Hinton. Visualizing high-dimensional data using t-SNE. *JMLR*, 2008.

[25] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001.

[26] F. Shen, C. Shen, W. Liu, and H. Shen. Supervised discrete hashing. In *CVPR*, 2015.

[27] F. Shen, C. Shen, Q. Shi, A. van den Hengel, and Z. Tang. Inductive hashing on manifolds. In *CVPR*, 2013.

[28] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *TPAMI*, 2012.

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

[30] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *JMLR*, 2010.

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

[32] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[33] P. Wohlhart, M. Kostinger, M. Donoser, P. Roth, and H. Bischof. Optimizing 1-nearest prototype classifiers. In *CVPR*, 2013.

[34] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[35] Y. Zhang, K. Huang, G. Geng, and C. Liu. Fast $k$nn graph construction with locality sensitive hashing. In *ECML PKDD*, 2013.

[36] Z. Zhang, P. Sturgess, S. Sengupta, N. Crook, and P. Torr. Efficient discriminative learning of parametric nearest neighbor classifiers. In *CVPR*, 2012.