Order Preserving Hashing for Approximate Nearest Neighbor Search

Jianfeng Wang[†][*] Jingdong Wang[‡] Nenghai Yu[†] Shipeng Li[‡]
[†]University of Science and Technology of China, P. R. China
[‡]Microsoft Research Asia, P. R. China
wjf2006@mail.ustc.edu.cn, {jingdw, spli}@microsoft.com, ynh@ustc.edu.cn

ABSTRACT

In this paper, we propose a novel method to learn similarity-preserving hash functions for approximate nearest neighbor (NN) search. The key idea is to learn hash functions by maximizing the alignment between the similarity orders computed from the original space and the ones in the hamming space. The problem of mapping the NN points into different hash codes is taken as a classification problem in which the points are categorized into several groups according to the hamming distances to the query. The hash functions are optimized from the classifiers pooled over the training points. Experimental results demonstrate the superiority of our approach over existing state-of-the-art hashing techniques.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Order preserving hashing; approximate nearest neighbor search; learning to hash

1. INTRODUCTION

Nearest neighbor search (or similarity search) in large data sets has wide applications in machine learning and computer vision. However, exact nearest neighbor search is often intractable because of the large scale of the database and the curse of the high dimensionality. Instead, approximate nearest neighbor (ANN) search is more practical and has been attracting a lot of attention [32].

As one of the commonly-used approaches, hashing uses multiple integer-valued hash functions to map similar inputs to nearby hash buckets. The pioneering work, locality sensitive hashing (LSH) [8], adopts random projections to compute hash codes. It is probabilistically guaranteed that nearer points have a larger probability to share the same binary codes and farther points have a larger probability to be mapped to the farther binary codes. However, LSH is data independent and has been shown in practice to be weaker than data-dependent hashing methods. To improve the performance, LSH usually needs long code length or multiple hash tables resulting in a large space and time cost.

Recently many learning-to-hashing algorithms have been developed, such as [3, 4, 6, 7, 10, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25, 27, 28, 29, 31, 34, 37, 38, 39, 40, 41]. Most algorithms aim to learn hash functions so that the similarities in the input space are kept in the hamming space, in various manners. Binary reconstructive embedding [16] aims to minimize the differences between the distances in the input (kernel) space and those in the hamming space. In real data sets that are usually not uniformly distributed, preserving absolute similarities/distances does not guarantee to produce good nearest neighbor search that concerns more the relative similarity than the absolute similarity. Instead, some other algorithms, e.g., [27, 34, 38], divide the point pairs into similar and dissimilar pairs, i.e., binary similarities, and find hash functions by preserving the binary similarities. Such approaches are more suitable to preserve semantic similarity and have little capability of differentiating multiple level similarities that are essential for nearest neighbor search.

In this paper, we propose a novel ANN search approach, order preserving hashing (OPH). The key idea is to learn hash functions based on a so-called order alignment criterion, which is some notion of similarity preserving. Order alignment means that the similarity order computed in the input space should be well preserved in the hamming space. and is essentially the goal of ANN search. Hashing with preserving orders is more general than binary similarity preserving, and advantageous over direct distance preserving in dealing with real skewed data.

Our formulation is also inspired by a widely-used bucket balance criterion, i.e., points are uniformly distributed in hash buckets. The joint consideration of such two criteria, if viewing each point as a query, results in a list of categories, where each category is composed of the points having the same hamming distance to the query. Then we formulate the category order alignment as a multi-class classification problem with each category viewed as a class. We present an easily-implemented optimization algorithm that incorporates Sigmoid relaxation, mini-batch stochastic gradient descent, and active set. Experimental results over standard

[*]The work was done during Jianfeng Wang was an intern at Microsoft Research Asia.

MM'13, October 21–25, 2013, Barcelona, Spain.
Copyright 2013 ACM 978-1-4503-2404-5/13/10 ...$15.00.
http://dx.doi.org/10.1145/2502081.2502100 .

sets demonstrate the superiority of our approach over existing state-of-the-art ANN search algorithm.

2. RELATED WORK

Hashing is a category of algorithms that map high-dimensional vectors into binary codes. Recently many hashing algorithms have been developed for approximate nearest neighbor search and have achieved a lot of success in multimedia applications, e.g., image search [5, 18, 36, 43], video retrieval [1, 33], duplicate discovery in video sequence [42] and document retrieval [31]. The common principle of different hashing algorithms is to align the similarities between the hamming space and the input space. Existing algorithms can be roughly divided into two categories, probabilistic similarity preserving and determinative similarity preserving.

Probabilistic similarity preserving. Locality sensitive hashing (LSH) [8], one of the typical representatives, is a method of performing ANN search in high dimensions, dependent on probabilistic dimension reduction of high-dimensional data (random projection is a widely-used way to dimension reduction). The key idea is to hash the points using several hash functions to ensure a probabilistic guarantee of similarities, i.e., for each function the probability of collision is much higher for objects (points) that are close to each other than those far apart.

There are various algorithms extending LSH, such as [11, 17, 25, 29]. Kernelized LSH [17] studies the hash functions by extending the idea of random projection to the kernel space. Shift-invariant kernel hashing [29] adopts a random cosine function to generate the hash codes and shows that the hamming distance is closely related to the shift-invariant kernel value. Super-bit LSH [11] orthogonalizes the random projections in batches and an unbiased estimate of the angle similarity is theoretically supported with a smaller variance under certain conditions. These approaches do not explore the data distribution for hash function construction and usually need longer hash codes or more hash tables to achieve a satisfactory performance, thus resulting in a large space and time cost.

Deterministic similarity preserving. In contrast to probabilistic similarity preserving, deterministic similarity preserving hashing algorithms formulate a problem that minimizes the misalignment rate between the similarities/distaces computed from the hamming and input space.

Binary reconstructive embedding [16] aims to align the hamming distance over the hash codes and the distance in the input Euclidean (and kernel) spaces. Iterative quantization hashing [3] constructs hash functions by rotating the axes so that the difference between the binary codes and the projected data is minimized [3], while the goal of isotropic hashing [14] is to make the variances along the rotated linear projections are as same as possible. Both iterative quantization hashing and isotropic hashing are related to multidimensional scaling and in some sense aim to keep the Euclidean distance.

Entropy-based coding [20] regards the similarities as a distribution and learns hash functions by preserving the normalized similarity distribution and minimizes the Kullback-Leibler divergence between the distributions computed from the input space and the hamming space. Spectral hashing (SH) [40] exploits the similarities in a different way and aims to minimize the summation of the hamming distance weighted by the similarities computed in the input space. The following works [6, 23] extend spectral hashing by generalizing to kernel hash functions.

Rather than preserving absolute similarities that are shown to have limited capabilities in dealing with real skewed data, algorithms on preserving binary similarities (point pairs are divided into similar and dissimilar pairs.) have been studied. The sequential projection learning scheme [38] aims to minimize (maximize) the hamming distance of similar (dissimilar) pairs. LDAHash [34] learns hash functions by minimizing (maximizing) the expectation of the hamming distances over positive(negative) pairs. The weakly-supervised approach [24] exploits the extra property, large margin to help learn hash functions. Minimal loss hashing [27] adopts a hinge-like loss function to penalize the case that the hamming distances of the similar points are larger than a threshold and the hamming distances of the dissimilar pairs are smaller than another threshold. The triplet similarity preserving hashing approach [28] is a framework of learning a broad class of hash functions, using a triplet ranking loss, is shown to be more flexible and very well suited to the preservation of semantic similarity.

Besides, there are hashing algorithms that exploit other information to improve the performance. Complementary hashing [41] learns multiple complementary hash tables instead of randomized hash tables. Multi-modality hashing [13, 23, 33] aims to learn hash functions by exploring multiple features. Various distance measures in the binary-code space, such as spherical hamming distance [7], Manhattan distance [15], weighted hamming distance [12, 39] and non-parametric distance [10], are considered to learn hash functions. Query-adaptive hashing [9] performs an selection of the most appropriate hash functions from a pool of functions.

3. FORMULATION

The reference data set is denoted by $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ with $\mathbf{x} \in \mathbb{R}^d$. The hashing scheme for ANN search maps each point $\mathbf{x}$ to a m-dimensional binary vector $\mathbf{c}$ through a hash function $\mathbf{h} : \mathbb{R}^d \mapsto \mathbb{H}^m$, where $\mathbb{H}^m$ denotes the Hamming space of dimension m. We use $d_H(\mathbf{c}, \tilde{\mathbf{c}})$ to denote the hamming distance, i.e., the number of bits on which $\mathbf{c}$ and $\tilde{\mathbf{c}}$ differ.

An ideal hash function expects that the ordered lists computed from the hamming space and the original space are well matched, which we call *order alignment*. For a query point $\mathbf{x}$, denote the similarity order computed in the hamming space as $L^h = (i_1, \cdots, i_N)$ and the order computed from the original space, as $L^e = (j_1, \cdots, j_N)$. Order alignment between the two lists of points, called *point order alignment*, means $i_n = j_n, \ \forall n$.

In the hamming space, there are a limited number of $(m+1)$ distance values. Therefore, searching in the hamming distance results in a set of ordered $m+1$ categories, $(\mathcal{C}^h_0, \ \mathcal{C}^h_1, \ \cdots, \ \mathcal{C}^h_m)$, where $\mathcal{C}^h_k = \{i^k_1, \cdots, i^k_{M_k}\}$. Accordingly, the ordered list L^e is divided into $m+1$ categories, $(\mathcal{C}^e_0, \mathcal{C}^e_1, \cdots, \mathcal{C}^e_m)$, which we call target categories, such that $|\mathcal{C}^h_k| = |\mathcal{C}^e_k|$. Order alignment between the two lists of categories, which we call *category order alignment*, means $\mathcal{C}^h_k = \mathcal{C}^e_k$.

However, category order alignment does not guarantee a good hashing. Considering the extreme case that all points are mapped to a same hash code, we can see category order alignment is satisfied, $|\mathcal{C}^h_0| = |\mathcal{C}^e_0|$ and $|\mathcal{C}^h_1| = \cdots = |\mathcal{C}^h_m| = 0$.

But this case does not lead to a good hashing. To handle such an issue, we propose to exploit another condition, *bucket balance*, which means that the points are uniformly distributed in all hash buckets and is helpful to guarantee search efficiency.

We aim to learn hash functions based on the above two properties, category order alignment and bucket balance. Given the target categories that are constructed according to the bucket balance property, category order alignment is regarded as a multi-class classification problem. That's for a query all the reference data points are assigned to one of $m+1$ target categories. To penalize the misalignment rate, we transform the multi-class classification problem to m binary-class classification problems. Specifically, we partition the target categories into m sets of binary super-categories, $\{(\mathcal{N}_0^e, \overline{\mathcal{N}_0^e}), (\mathcal{N}_1^e, \overline{\mathcal{N}_1^e}), \cdots, (\mathcal{N}_{m-1}^e, \overline{\mathcal{N}_{m-1}^e})\}$, where each pair of super-categories $(\mathcal{N}_i^e, \mathcal{N}_i^h)$ forms a binary-class classification problem. Here, $\mathcal{N}_i^e = \cup_{j=0}^{i} \mathcal{C}_j^e$ and $\overline{\mathcal{N}_i^e} = \mathcal{X} - \mathcal{N}_i^e$, and the super-categories in the hamming space, $\{(\mathcal{N}_i^h, \overline{\mathcal{N}_i^h})\}_{i=0}^{m-1}$, is similarly built. The objective function of penalizing misalignment rate is formulated as follows,

$$
\begin{aligned}
& L(\mathbf{h}(\cdot); \mathcal{X}) && (1)\\
= & \sum\nolimits_{n=1}^{N} L(\mathbf{h}(\cdot); \mathbf{x}_n) && (2)\\
= & \sum\nolimits_{n=1}^{N} \sum\nolimits_{i=0}^{m-1} L(\mathbf{h}(\cdot); \mathbf{x}_n, i) && (3)\\
= & \sum\nolimits_{n=1}^{N} \sum\nolimits_{i=0}^{m-1} (|\mathcal{N}_{ni}^e - \mathcal{N}_{ni}^h| + \lambda |\mathcal{N}_{ni}^h - \mathcal{N}_{ni}^e|), && (4)
\end{aligned}
$$

where $|\mathcal{N}_{ni}^e - \mathcal{N}_{ni}^h|$ and $|\mathcal{N}_{ni}^h - \mathcal{N}_{ni}^e|$ are the errors for the target binary super-categories, $(\mathcal{N}_{ni}^e, \overline{\mathcal{N}_{ni}^e})$, and $|\mathcal{N}_{ni}^h - \mathcal{N}_{ni}^e| = |\overline{\mathcal{N}_{ni}^e} - \overline{\mathcal{N}_{ni}^h}|$. λ is a trade-off parameter to balance the penalties for the two categories.

4. OPTIMIZATION

In this paper, we use the widely-used hash function, the linear projection, as an example implementation of order preserving hashing. The linear hash functions are formulated as follows,

$$
\begin{aligned}
\mathbf{h}(\mathbf{x}) = & \operatorname{sign}(\mathbf{W}^T \mathbf{x} + \mathbf{b}) \qquad (5)\\
= & [\operatorname{sign}(\mathbf{w}_1^T \mathbf{x} + b_1) \ \cdots \ \operatorname{sign}(\mathbf{w}_m^T \mathbf{x} + b_m)]^T,
\end{aligned}
$$

where $\mathbf{W}$ is a projection matrix of size $d \times m$, each column $\mathbf{w}_k \in \mathbb{R}^d$ is a projection vector and $\mathbf{w}_k^T \mathbf{w}_k = 1$, $\mathbf{b} \in \mathbb{R}^m$ being the threshold for the hash function. $\operatorname{sign}(\cdot)$ is a sign function, where $\operatorname{sign}(z) = 1$ when $z > 0$ and sign(z) = 0 otherwise. For convenience, we formulate an augmented matrix $\overline{\mathbf{W}} \in \mathbb{R}^{(d+1)\times m}$, $\overline{\mathbf{W}} = [\mathbf{W}^T | \mathbf{b}]^T$, and then the normalization constraint can be equivalently written as $\overline{\mathbf{w}}_k^T \overline{\mathbf{w}}_k = 1$. Accordingly, a point $\mathbf{x}$ is augmented as $\overline{\mathbf{x}} = [\mathbf{x}^T | 1]^T$. As a result, $\mathbf{h}(\overline{\mathbf{x}}) = \operatorname{sign}(\overline{\mathbf{W}}^T \overline{\mathbf{x}})$. The following description might drop the upper bar if not affecting the understanding.

Recalling that $\mathcal{N}_{ni}^h$ is formed by the points whose hamming distances to the point $\mathbf{x}_n$ are not larger than i, we have the following equations,

$$
\begin{aligned}
|\mathcal{N}_{ni}^e - \mathcal{N}_{ni}^h| &= \sum_{\mathbf{x}' \in \mathcal{N}_{ni}^e} \operatorname{sign}(d_H(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}')) - i),\\
|\mathcal{N}_{ni}^h - \mathcal{N}_{ni}^e| &= \sum_{\mathbf{x}' \notin \mathcal{N}_{ni}^e} \operatorname{sign}(i + 1 - d_H(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}'))).
\end{aligned}
$$

Here, the function $\operatorname{sign}(\cdot)$ is used to check if the point is correctly classified. Replacing $d_H(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}'))$ by the equivalent form in the hamming space, $\|\mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}')\|_2^2$, we have

$$
\begin{aligned}
& L(\mathbf{W}; \mathbf{x}_n, i)\\
= & \sum\nolimits_{\mathbf{x}' \in \mathcal{N}_{ni}^e} \operatorname{sign}(\|\mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}')\|_2^2 - i)\\
& + \lambda \sum\nolimits_{\mathbf{x}' \notin \mathcal{N}_{ni}^e} \operatorname{sign}(i + 1 - \|\mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}')\|_2^2). \qquad (6)
\end{aligned}
$$

4.1 Relaxation

It can be observed that the function $\operatorname{sign}(\cdot)$ in Eqn. (5) and (6) is non-smooth, which makes the optimization uneasy. To address this issue, we propose to relax the $\operatorname{sign}(\cdot)$ to a Sigmoid function $\phi_\alpha(z) = \frac{1}{1+e^{-\alpha z}}$. Its first-order derivative is $\phi'_\alpha(z) = \alpha \phi_\alpha(z)(1 - \phi_\alpha(z))$. Here α is used to balance the quality approximating $\operatorname{sign}(\cdot)$ and the smoothness degree. A larger α leads to a better approximation, but a more difficult optimization.

The hash function $\mathbf{h}(\mathbf{x}) = \operatorname{sign}(\mathbf{W}^T \mathbf{x})$ is relaxed as

$$
\mathbf{h}(\mathbf{x}) = \boldsymbol{\phi}_\beta(\mathbf{W}^T \mathbf{x}) = [\phi_\beta(\mathbf{w}_1^T \mathbf{x}) \ \cdots \ \phi_\beta(\mathbf{w}_m^T \mathbf{x})]^T. \qquad (7)
$$

Besides, the function in Eqn. (6) is written by furthermore relaxing other two sign functions as follows,

$$
\begin{aligned}
L_{ni}(\mathbf{W}) = & L(\mathbf{W}; \mathbf{x}_n, i)\\
\triangleq & \sum\nolimits_{\mathbf{x}' \in \mathcal{N}_{ni}^e} \phi_{\gamma_{i_1}}(d_{RH}(\mathbf{x}_n, \mathbf{x}') - i)\\
& + \lambda \sum\nolimits_{\mathbf{x}' \notin \mathcal{N}_{ni}^e} \phi_{\gamma_{i_2}}(i + 1 - d_{RH}(\mathbf{x}_n, \mathbf{x}')), \qquad (8)
\end{aligned}
$$

where $d_{RH}(\mathbf{x}, \mathbf{x}') = \|\boldsymbol{\phi}_\beta(\mathbf{W}^T \mathbf{x}) - \boldsymbol{\phi}_\beta(\mathbf{W}^T \mathbf{x}')\|_2^2$ and γ_{i_1} and γ_{i_2} are the parameters.

The overall objective function is then written as,

$$
\begin{aligned}
\min_{\mathbf{W}} \quad & L(\mathbf{W}) = \frac{1}{N} \sum\nolimits_{n=1}^{N} \sum\nolimits_{i=0}^{m-1} L_{ni}(\mathbf{W})\\
\text{s.t.} \quad & \mathbf{w}_k^T \mathbf{w}_k = 1, \forall k = 1, \cdots, m.
\end{aligned} \qquad (9)
$$

4.2 Quadratic penalty

We propose to use the quadratic penalty method [26] to solve this nonlinear constrained optimization problem in Eqn. (9).

$$
Q(\mathbf{W}; \mu) \triangleq L(\mathbf{W}) + \frac{\mu}{4} \sum\nolimits_{k=1}^{m} (\mathbf{w}_k^T \mathbf{w}_k - 1)^2, \qquad (10)
$$

where $\mu > 0$ is the penalty parameter. By driving μ to ∞, we penalize the constraint violations with increasing severity. We consider a sequence of values μ_t with $\mu_t \to \infty$ as $t \to \infty$, and seek the approximate minimizer $\mathbf{W}_t$ of $Q(\mathbf{W}; \mu_t)$. Practically, $\mu_{t+1} = \kappa \mu_t$ with $\kappa > 1$. In searching for $\mathbf{W}_t$, we use the minimizer $\mathbf{W}_{t-1}$ of $Q(\mathbf{W}; \mu_{t-1})$ to construct an initial guess.

A gradient descent algorithm is adopted to minimize $Q(\mathbf{W}; \mu_t)$. The gradient of $Q(\mathbf{W}; \mu_t)$ is computed as follows,

$$
\begin{aligned}
\nabla_k Q_t \triangleq & \frac{\partial Q(\mathbf{W}; \mu_t)}{\partial \mathbf{w}_k}\\
= & \frac{1}{N} \sum\nolimits_{n=1}^{N} \sum\nolimits_{i=0}^{m-1} \frac{\partial L_{ni}(\mathbf{W})}{\partial \mathbf{w}_k} + \mu_t (\mathbf{w}_k^T \mathbf{w}_k - 1) \mathbf{w}_k\\
\triangleq & \frac{1}{N} \sum\nolimits_{n=1}^{N} \nabla_k L_n + \mu_t \nabla_k G.
\end{aligned}
$$

Here, $\partial L_{ni}(\mathbf{W})/\partial \mathbf{w}_k = \sum_{\mathbf{x}' \in \mathcal{N}^e_{ni}} \partial \phi_{\gamma_{i_1}}(d_{RH}(\mathbf{x}_n, \mathbf{x}') - i)/\partial \mathbf{w}_k$ $+ \lambda \sum_{\mathbf{x}' \notin \mathcal{N}^e_{ni}} \partial \phi_{\gamma_{i_2}}(i + 1 - d_{RH}(\mathbf{x}_n, \mathbf{x}'))/\partial \mathbf{w}_k$.

Each iteration of the gradient descent algorithm of minimizing $Q(\mathbf{W}; \mu_t)$ requires to (1) compute $\nabla_k L_n$ for each point $\mathbf{x}_n$ and (2) go through all the N points to compute a single $\nabla_k L_n$. The cost is $O(N^2)$ (neglecting m as $m \ll N$). Given the scale N of the reference dataset $\mathcal{X}$ is very large, the direct solution might not be feasible. We propose to use two speedup schemes: stochastic gradient descent and active set, to address the above two issues, respectively.

4.2.1 Stochastic gradient descent

The standard (batch) gradient descent algorithm would perform the iterations,

$$\begin{aligned}\mathbf{w}_k \leftarrow & \mathbf{w}_k - \eta \nabla_k Q_t \\ = & \mathbf{w}_k - \eta(\frac{1}{N} \sum_{n=1}^{N} \nabla_k L_n + \mu_t \nabla_k G).\end{aligned}$$

We propose to take mini-batches to update each iteration, where the true gradient is approximated by a sum over a small number of randomly sampled training examples. The iteration is updated as the following,

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta(\frac{1}{|\mathcal{I}'|} \sum_{n \in \mathcal{I}'} \nabla_k L_n + \mu_t \nabla_k G), \tag{11}$$

where $\mathcal{I}'$ is a subset, randomly sampled from $\mathcal{I} = \{1, 2, \cdots, n\}$ at this iteration, and $|\mathcal{I}'| \ll |\mathcal{I}|$. This mini-batches manner reduces the complexity of each iteration to $O(N|\mathcal{I}'|) = O(N)$.

4.2.2 Active set

Considering the derivative of the Sigmoid function, $f(\alpha, z) \triangleq \phi'_\alpha(z) = \alpha \phi_\alpha(z)(1 - \phi_\alpha(z))$, we can see that $\phi'_\alpha(z) \approx 0$ if $\phi_\alpha(z) \approx 0$. We can also easily show that the absolute value of each entry in the derivative of $d_{RH}(\mathbf{x}, \mathbf{x}')$, $\partial d_{RH}(\mathbf{x}, \mathbf{x}')/\partial \mathbf{w}_k = 2(\phi_\beta(\mathbf{w}_k^T \mathbf{x}) - \phi_\beta(\mathbf{w}_k^T \mathbf{x}'))(f(\beta, \mathbf{w}_k^T \mathbf{x})\mathbf{x} - f(\beta, \mathbf{w}_k^T \mathbf{x}')\mathbf{x}')$, is up-bounded. Consequently,

$$\nabla_k \phi_{\gamma_{i_1}} = \frac{\partial \phi_{\gamma_{i_1}}(d_{RH}(\mathbf{x}_n, \mathbf{x}') - i)}{\partial \mathbf{w}_k} \approx 0$$

if $\phi_{\gamma_{i_1}}(d_{RH}(\mathbf{x}_n, \mathbf{x}') - i) \approx 0$, and

$$\nabla_k \phi_{\gamma_{i_2}} = \frac{\partial \phi_{\gamma_{i_2}}(i + 1 - d_{RH}(\mathbf{x}_n, \mathbf{x}'))}{\partial \mathbf{w}_k} \approx 0$$

if $\phi_{\gamma_{i_2}}(i + 1 - d_{RH}(\mathbf{x}_n, \mathbf{x}')) \approx 0$. This suggests that it is unnecessary to go through all the N points when evaluating $\frac{\partial L_{ni}(\mathbf{W})}{\partial \mathbf{w}_k}$.

The approximation is obtained as follows,

$$\frac{\partial L_{ni}(\mathbf{W})}{\partial \mathbf{w}_k} \approx \sum_{\mathbf{x}' \in \mathcal{A}} \nabla_k \phi_{\gamma_{i_1}} + \lambda \sum_{\mathbf{x}' \in \mathcal{B}} \nabla_k \phi_{\gamma_{i_2}}. \tag{12}$$

Here, $\mathcal{A} = \mathcal{N}^e_{ni} - \mathcal{N}^h_{n(i-\Delta_{i_1})}$ and $\mathcal{B} = \mathcal{N}^h_{n(i+\Delta_{i_2})} - \mathcal{N}^e_{ni}$. $\mathcal{N}^h_{n(i-\Delta_{i_1})}$ consists of the points whose hamming distances to $\mathbf{x}_n$ are not larger than $i - \Delta_{i_1}$ and $\mathcal{N}^h_{n(i+\Delta_{i_2})}$ consists of the points whose hamming distances to $\mathbf{x}_n$ are not larger than $i + \Delta_{i_2}$. It means that $\frac{\partial L_{ni}(\mathbf{W})}{\partial \mathbf{w}_k}$ can be evaluated over the points in $(\mathcal{N}^e_{ni} - \mathcal{N}^h_{n(i-\Delta_{i_1})})$ and $(\mathcal{N}^h_{n(i+\Delta_{i_2})} - \mathcal{N}^e_{ni})$, which are related to false negative and false positive. In our implementation, $\Delta_{i_1} = \Delta_{i_2} = 1$. The set, $\mathcal{A}_{ni} = (\mathcal{N}^e_{ni} - \mathcal{N}^h_{n(i-\Delta_{i_1})}) \cup (\mathcal{N}^h_{n(i+\Delta_{i_2})} - \mathcal{N}^e_{ni})$ are called active set, and hence its size is much smaller than the number of the reference data points, $|\mathcal{A}_{ni}| \ll N$. Thus, the cost of the gradient computation is greatly reduced, from $O(N)$ to $O(|\mathcal{A}_{ni}|)$.

$\mathcal{N}^e_{ni}$ is already available from the ground truth over the reference data points. To identify the active set $\mathcal{A}_{ni}$, we need additionally to obtain the points in $\mathcal{N}^h_{n(i-\Delta_{i_1})}$ and $\mathcal{N}^h_{n(i+\Delta_{i_2})}$, which is equivalent to retrieving the points whose hamming distances to $\mathbf{h}(\mathbf{x}_n)$ lie in $[0, i-\Delta_{i_1}]$ and $[0, i+\Delta_{i_2}]$. To make it, it needs to go through all the points to compute their hash codes. This step needs one pass of all the points, taking $O(N)$ time, However, in practice it is quite fast and only takes 5% of the time of the gradient computation over the active set because the hamming distance evaluation is very efficient. Thus, the active set scheme reduces a lot of time cost.

The overall algorithm is described in Algorithm 1.

Algorithm 1 Order Preserving Hashing

```
Input: Code length $m$ and reference data $\mathcal{X}$
Output: Hash functions $\mathbf{w}_k, k = 1, ..., m$
 1: initialize $\mathbf{w}_k$, $\mu$, $max\_outer\_iter$, $max\_inner\_iter$
 2: Construct $\mathcal{N}^e_{ni}, n = 1, ..., N; i = 1, ..., m$ according to the
    bucket balance property
 3: for $outer\_iter = 1 : max\_outer\_iter$ do
 4:   for $inner\_iter = 1 : max\_inner\_iter$ do
 5:     Generate randomly sampled examples $\mathcal{I}'$
 6:     Calculate $\mathcal{N}^h_{ni}$ on $\mathcal{I}'$
 7:     Construct active sets $\mathcal{A}_{ni}$ on $\mathcal{I}'$ according to
        Sec. 4.2.2
 8:     Calculate $\frac{\partial L_{ni}(\mathbf{W})}{\partial \mathbf{w}_k}$ according to Eqn. (12)
 9:     $\nabla_k L_n \leftarrow \sum_i \frac{\partial L_{ni}(\mathbf{W})}{\partial \mathbf{w}_k}$
10:     Update $\mathbf{w}_k$ according to Eqn. (11)
11:   end for
      $\mu \leftarrow \kappa\mu$
12: end for
```

5. CONNECTIONS

The similarity order preserved in the proposed order preserving hashing approach is essentially some notion of similarity. This section discusses the connections with several representative similarity-preserving hashing approaches.

Relation to absolute similarity/distance preserving. Binary reconstructive embedding (BRE) [16] computes hash functions by preserving absolute distances, and entropy-based coding [20] aims to preserve the normalized similarity distribution. In contrast, our approach aims to preserve similarity orders (relative similarities), which is essentially the goal of nearest neighbor search. Experiments show that all the approaches achieve satisfactory performance when the data points are uniformly distributed and almost not skewed. However, in most real data sets that are usually not uniformly distributed and skewed, preserving relative similarities is more flexible than preserving absolute similarities (distances).

Relation to binary similarity preserving. Rather than preserving absolute similarities, hashing algorithms have been developed with the goal of preserving binary similarities, such as [19, 27, 38]. The former one [19] is different from our approach in that it is for learning a boosted search forest. The latter two [27, 38] are more related to our approach, and the idea is to learn hash functions, expecting that positive pairs in the hamming space are nearer or not larger than a threshold and negative pairs are farther or not smaller than another threshold. Positive pairs are defined as neighboring points for ANN search or points with the same semantic label for classification, and negative pairs are farther points or points with different labels. These approaches are more suitable for classification, but do not discriminate multiple similarities and hence cannot guarantee good ANN search performance. Instead, our approach takes into consideration order alignment, and is advantageous in discriminating multiple similarities.

Relation to ranking loss. The very recent work [28] presents a triplet ranking loss based hashing approach and demonstrates its application to classification. Such a loss is formed from a triplet, where one is more similar to a second one than the third one, or two with the same label and another sample with a different label. This ranking loss, if used for ANN search, is close to the order alignment property of our approach. The joint consideration of all the possible $O(N^3)$ triplets is equivalent to order alignment that is formed with $O(N^2)$ constraints. [28] samples a subset of triplets for the efficiency issue. Differently, our approach exploits it in an example-based manner, constructing the data satisfying the order alignment as the training data. Besides, we transform the problem to an equivalent problem, a set of multi-class classification problems, optimized with an easily-implemented algorithm.

6. EXPERIMENT

6.1 Setup

Approaches. We present the performance comparison of our approach, order preserving hashing (OPH), against several state-of-the-art hashing methods, including minimal loss hashing (MLH) [27], binary reconstructive embedding (BRE) [16], iterative quantization hashing (ITQ) [3], isotropic hashing (IsoHash) [14], kernelized supervised hashing (KSH) [21], anchor graph hashing (AGH) [22], spectral hashing (SH) [40], and unsupervised sequential projection locality hashing (USPLH) [38]. All the results were obtained with the implementations generously provided by their respective authors and by following their instructions to tune the algorithm parameters.

Data set. We report the search performance over five widely-used data sets: LabelMe [30], Peekaboom [35], GIST $1M$ [10], SIFT100K, and SIFT1M [10]. The first two data sets include $22,019$ and $57,637$ images, respectively. Each image is represented by a 512-dimensional GIST descriptor. $2K$ images are randomly sampled as the query set and the remaining as the reference set. The GIST1M data set contains $1M$ 960-dimensional GIST features as the reference set $1K$ features as the query set, and extra $500K$ features as the training set that is used to learn the hash functions. The SIFT100K data set contains $100K$ 128-dimensional SIFT features as the reference data set and extra $10K$ features as the query set, which are collected from the Caltech 101 data set [2]. The SIFT1M data set contains $1M$ SIFT features as the reference set, $10K$ features as the query set, and extra $100K$ as the training set. The ground-truth neighbors, exact K-nearest neighbors, for each query, are computed by linear scan, i.e., comparing each query with all the points in the reference data set using the raw feature.

6.2 Implementation

In the implementation of our approach as illustrated in Algorithm. 1, we initialize linear projections $\mathbf{W}$ of the hash functions by assigning each entry with a random variable sampled from a standard Gaussian distribution $\mathcal{N}(0, 1)$ and normalizing the augmented projection vector to have unit length. Both *max_outer_iter* and *max_inner_iter* are set 10 with μ initialized by 1 and $\kappa = 1.5$. The variable α, which is used to relax $\text{sign}(z)$ to the Sigmoid function $\phi_\alpha(z)$, is selected according to the range of z. In particular, if $\max|z| = \zeta$, $\alpha = 2\log(\epsilon^{-1})/\zeta$ to guarantee that the approximation error is not larger than a small constant ϵ ($\epsilon = 10^{-6}$ in our implementation) if $\zeta/2 \leq |z| \leq \zeta$. This empirical rule is applied to determining β in Eqn. (7) and γ_{i_1}, γ_{i_2} in Eqn. (8). Specifically, $\beta = 2\log(10^6)/\sqrt{\max\|\mathbf{x}\|_2^2 + 1}$; $\gamma_{i_1} = 2\log(10^6)/(m - i)$ as $\max(d_{RH}(\mathbf{x}, \mathbf{x}') - i) = m - i$; $\gamma_{i_2} = 2\log(10^6)/(1 + i)$ as $\max(i + 1 - d_{RH}(\mathbf{x}, \mathbf{x}')) = i + 1$. In each iteration of the mini-batch stochastic gradient algorithm, we randomly sample 200 training samples.

Rather than partitioning target categories into $m + 1$ binary super-categories, which is impractical for larger m and takes much time cost in optimization, we sample a small number (S, set as 10 in our implementation) of binary super-categories, $\{(\mathcal{N}^e_{k_1}, \overline{\mathcal{N}^e_{k_1}}), (\mathcal{N}^e_{k_2}, \overline{\mathcal{N}^e_{k_2}}), \cdots, (\mathcal{N}^e_{k_S}, \overline{\mathcal{N}^e_{k_S}})\}$ with $|\mathcal{N}^e_{k_{s+1}} - \mathcal{N}^e_{k_s}|$ being almost the same. Besides, it is not guaranteed that strict bucket balance always leads to the best performance. Thus, we relax the bucket balance property by fixing the point membership in the super-category but perturbing k_s to $\{k_s - 1, k_s, k_s + 1\}$ (in other words, the hamming distance of the points in $\mathcal{N}^e_{k_s}$ from the query can be constrained to be no larger than $k_s - 1$ or $k_s + 1$). We determine those choices and the selection of λ, by validation with a randomly-sampled set, 5% of the training set, as the validation set. There are many possible combinations of the choices, which leads to huge time cost during validation. To eliminate the time cost, we validate the choice for each variable by fixing other variables and perform the validation several times such that on average each variable is checked at least 3 times.

All experiments are conducted on a workstation with an Intel Xeon CPU of 3.33GHz and 24GB memory. The training time of our Matlab implementation is no more than 6 hours through all the datasets, except that the training time for the 960-dimensional GIST1M data set for code length 256 is about 8 hours.

6.3 Results

We report the performance comparison in terms of widely-used quantitative schemes, recall vs. #(retrieved items), recall vs. code length, mean average precision, and the qualitative results.

Recall vs. #(retrieved items). This scheme is often used to evaluate the performance of the ANN search strategy based on hamming ranking: a certain number of top ranked points according to hamming distances are returned and followed by a possible reranking step, where a more costly ranking function over more accurate features is evaluated. The number of returned images clearly reflects the computational cost of the reranking stage, while recall, the fraction of retrieved images that are among the K-NNs of the query, gives the upper-bound performance.

Fig. 1 shows the performance comparison under various code lengths (from 32 bits to 256 bits) over the five data

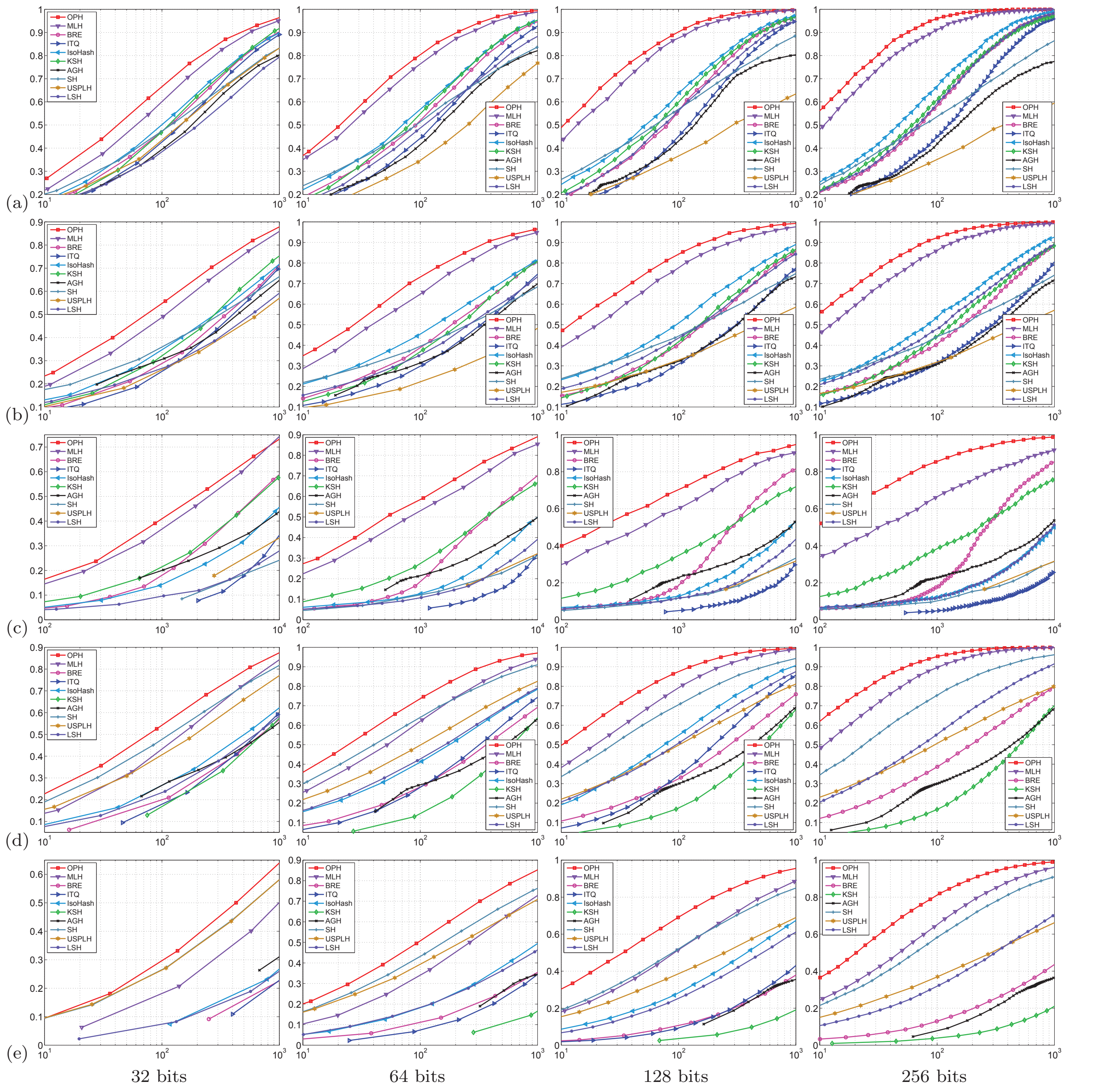


Figure 1: Performance comparison in terms of recall vs. #retrieved items under various code lengths. x-axis: #retrieved items. y-axis: recall. (a)-(e) correspond to Labelme, Peekaboom, GIST$1M$, SIFT$100K$, and SIFT$1M$, respectively.

sets for finding the nearest neighbor. The results for ITQ and IsoHash with code length 256 for 128-dimensional SIFT features are not available since the two algorithms can not generate hash codes with the length larger than 128 without using kernel hash functions. From these figures, we can see several observations. First, our approach (OPH) outperforms all the other approaches over all the five data sets and under all the code lengths. Second, the superiority of our approach gets more signification with longer binary codes. This is a desired property as longer hash codes are often used for real problems. Last but not least, by comparing (a)(22K GIST), (b) (57K GIST) with (c) (1M GIST), and (d) (100K SIFT) with (e) (1M SIFT), we find that the improvement of our approach over the second best approach (MLH) tend to be more significant in the larger reference set.

We present the performance in terms of recall vs. #(retrieved items) when finding different numbers (from 5 to 200) of nearest neighbors, which is shown in Fig. 2. The results of $K = 1$ have been shown in Fig. 1. Results with 128 bits are reported, and comparisons with other code lengths are similar. Overall speaking, our approach gets the better

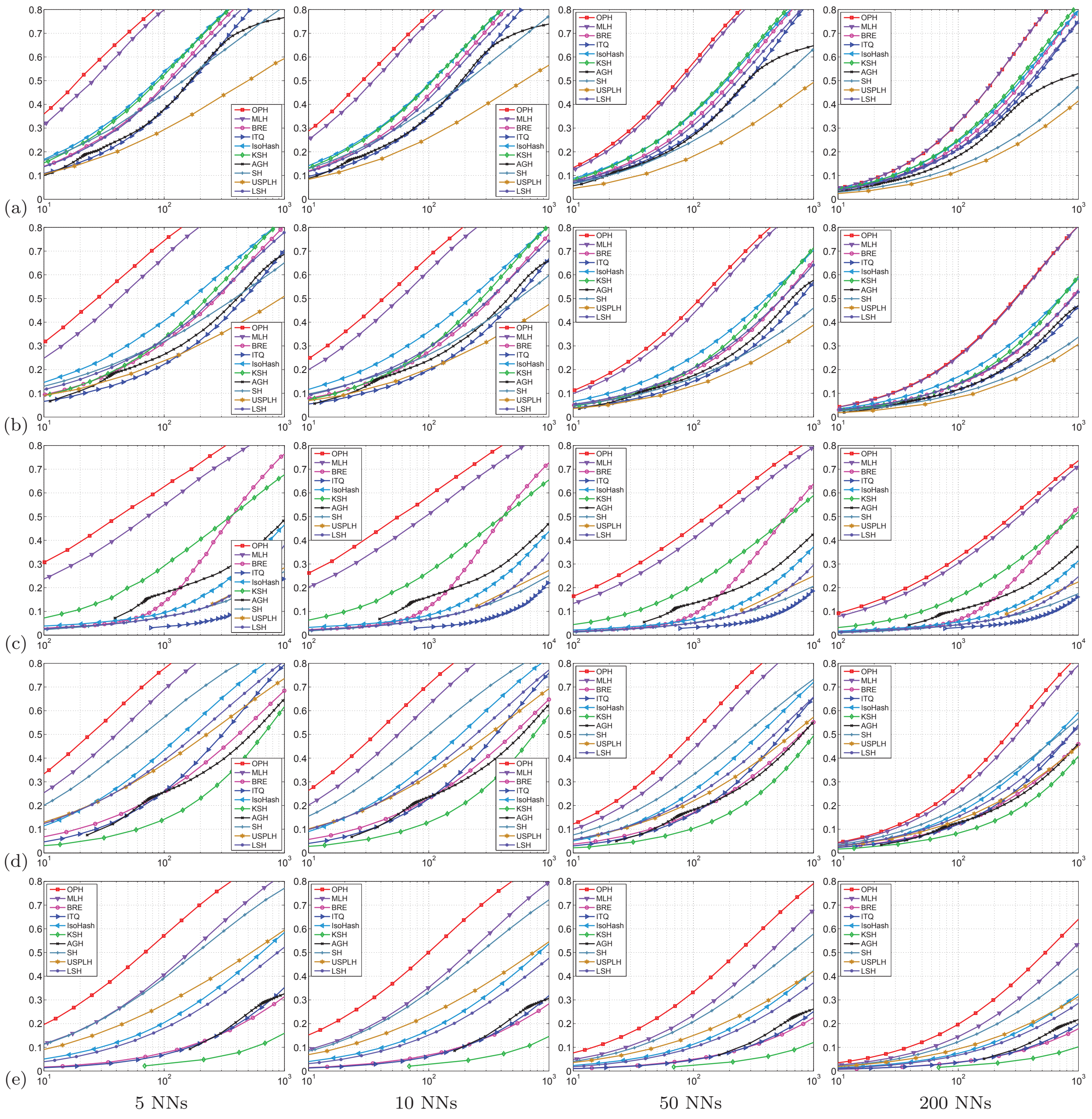
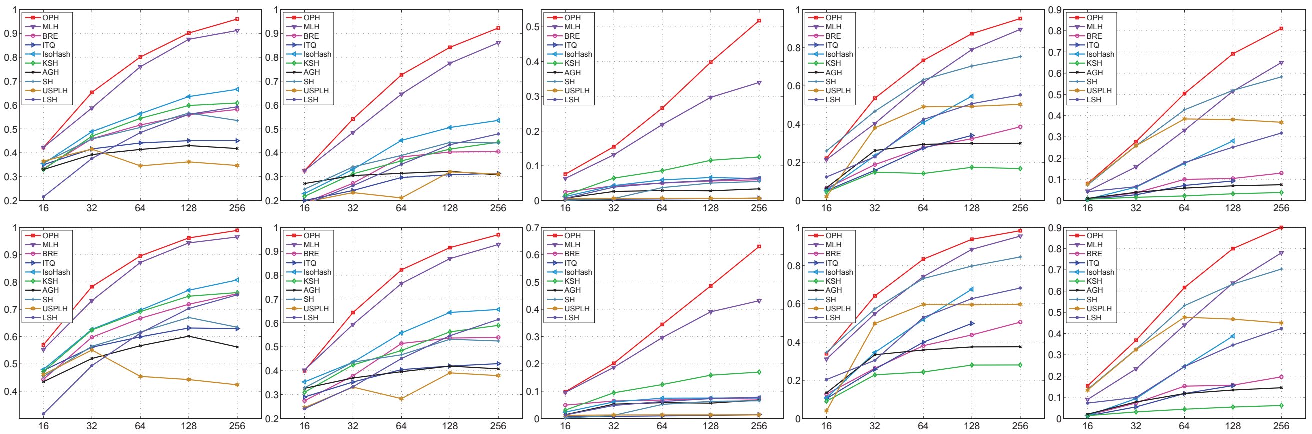


Figure 2: Performance comparison in terms of recall vs. #retrieved items when finding different numbers of nearest neighbors. x-axis: #retrieved items. y-axis: recall. (a)-(e) correspond to Labelme, Peekaboom, GIST$1M$, SIFT$100K$, and SIFT$1M$, respectively.

performance than all the other approaches. For the small data sets, (a) Labelme and (b) Peekaboom, our approach gets much improvement for 5, 10 and 50 NNs, and similar performance with the second best for 200 NNs. This is understandable as the points in small data sets are relatively easily discriminated. For the large data sets, (c) GIST$1M$, (d) SIFT$100K$ and (e) SIFT$1M$, the improvement is much more significant, which indicates that our approach is more powerful in dealing with larger data sets. Furthermore, we find that the superiority of our approach is more significant for smaller K. This implies that our approach is much more useful in many applications, for examples, feature correspondence in the structure-from-motion problem and bag-of-visual-words for object retrieval only require to retrieve a few number of nearest neighbors.

Recall vs. code length. Fig. 3 shows the comparison. The results are obtained for finding the nearest neighbor when retrieving top-100 and top-200 ranked points corresponding to the first and second row of Fig. 3, respectively. We can see that our approach benefits a lot with a larger code length. It can also be observed the improvement of OPH is more significant on larger data sets. More impor-



(a) Labelme (b) Peekaboom (c) GIST1M (d) SIFT100K (e) SIFT1M

Figure 3: Performance comparison in terms of recall vs. code length. x-axis: code length. y-axis: recall. The first row: recall@top-100. The second row: recall@top-200.

tantly, we find that our approach with 128 bits achieves similar and even better performance than other approaches with 256 bits, which indicates that our approach reduces the storage cost of saving binary codes when meeting the same recall target.

Mean average precision. Average precision (AP) of each query is the area of the precision-recall curve. Mean average precision is the mean of AP over all the queries and is useful to get an overview of search performance. The precision and recall scores are recorded by considering all possible numbers of candidates ranked by the hamming distance. The comparisons are shown in Table 1 with the ground truth being 50 nearest neighbors. On smaller data sets Labelme and Peekaboom, OPH is comparable, if not superior, to the state-of-the-art methods. One reason is data sets are small and relatively less challenged to differentiate. On larger data sets GIST1M, SIFT100K, and SIFT1M, the improvement is significant and impressive, which infers on average the rank list obtained from our approach is much better.

Qualitative results. Fig. 4 and 5 shows some qualitative results on Labelme (left of the figures) and Peekaboom (right of the figures) with code length 128. From Fig. 4, OPH offers a better performance than the others, and good qualitative results can also be observed in Fig. 5.

7. CONCLUSIONS

In this paper, we propose a new hashing approach, order preserving hashing, to approximate nearest neighbor search. The key idea is to maximize the alignment of category orders computed between the original and hamming space. Together with the bucket balance criteria, the problem of learning the hash function is cast to a multi-class classification problem. To penalize the misalignment rate, we transform the multi-class classification problem to multiple binary-class classification problems, for which Sigmoid relaxation, stochastic gradient descent and active set are employed to efficiently solve the problem. Extensive experimental results demonstrate the superiority of our approach.

Possible extensions. Our approach uses linear hash functions as an example, but is not limited and can be generalized to other hash functions, such as kernel hash functions or multi-layers neural nets. On the other hand, the target order can be defined as any metric in the input space and even a non-metric measure without being limited in the Euclidean distance.

8. REFERENCES

[1] L. Cao, Z. Li, Y. Mu, and S.-F. Chang. Submodular video hashing: a unified framework towards video pooling and indexing. In *ACM Multimedia*, pages 299–308, 2012.

[2] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR 2004 Workshop on Generative-Model Based Vision*, 2004.

[3] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011.

[4] J. He, S. Chang, R. Radhakrishnan, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, pages 753–760, 2011.

[5] J. He, J. Feng, X. Liu, T. Cheng, T.-H. Lin, H. Chung, and S.-F. Chang. Mobile product search with bag of hash bits and boundary reranking. In *CVPR*, pages 3005–3012, 2012.

[6] J. He, W. Liu, and S. Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010.

[7] J. Heo, Y. Lee, J. He, S. Chang, and S. Yoon. Spherical hashing. In *CVPR*, pages 2957–2964, 2012.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[9] H. Jégou, L. Amsaleg, C. Schmid, and P. Gros. Query-adaptative locality sensitive hashing. In *ICASSP*. IEEE, apr 2008.

[10] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

[11] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *NIPS*, pages 108–116, 2012.

Table 1: Comparison in terms of mean average precision (%). The best scores are in bold font. Our approach OPH obtains a significant gain on larger data sets (SIFT$100K$, SIFT$1M$, GIST$1M$) and comparable performance or superior on smaller data sets (Labelme, Peekaboom).

Data Set	Code Length	Approaches									
		OPH	MLH	BRE	ITQ	IsoHash	KSH	AGH	SH	USPLH	LSH
Labelme	32	**21.11**	19.91	16.45	20.36	18.51	16.72	10.90	9.28	9.62	8.87
	64	**33.94**	32.48	27.59	32.09	28.35	24.57	12.04	11.18	17.66	17.57
	128	44.36	**45.22**	40.59	44.66	42.34	31.45	12.35	13.73	9.67	32.52
	256	56.85	55.89	53.31	**58.40**	57.35	36.32	13.10	14.54	9.61	49.64
Peekaboom	32	**13.00**	12.59	9.42	12.10	10.63	9.40	6.01	5.25	6.22	4.47
	64	**23.00**	21.65	18.70	21.20	19.67	14.64	6.74	6.25	8.73	10.62
	128	**34.14**	33.08	30.42	32.75	32.31	21.33	7.05	9.28	5.94	23.74
	256	45.42	44.42	42.08	46.60	**47.41**	24.86	7.45	10.21	5.99	39.54
GIST$1M$	32	**2.00**	1.74	1.14	1.68	1.39	1.15	0.94	0.68	0.26	0.56
	64	**4.12**	3.51	2.67	3.27	3.25	2.21	1.05	1.08	0.32	1.50
	128	**6.97**	5.96	5.24	5.14	5.21	3.83	1.09	1.45	0.31	3.12
	256	**10.43**	7.20	8.18	6.95	7.70	5.26	1.08	2.19	0.33	6.04
SIFT$100K$	32	**12.40**	9.97	5.14	8.14	7.06	4.75	4.97	10.09	8.01	5.02
	64	**25.17**	18.63	11.31	18.37	15.47	8.87	5.98	17.36	12.67	12.52
	128	**37.41**	32.13	20.76	32.44	28.27	12.69	6.46	22.51	16.19	24.50
	256	**51.57**	46.42	32.78	-	-	15.97	6.67	32.42	18.56	40.69
SIFT$1M$	32	**5.07**	3.07	1.51	2.69	2.31	1.26	1.11	4.23	3.92	1.49
	64	**13.58**	8.11	4.46	8.16	7.24	2.94	1.50	9.81	8.19	5.68
	128	**26.00**	18.01	10.58	17.87	16.53	5.04	1.68	15.56	10.52	14.05
	256	**39.88**	31.69	20.16	-	-	7.38	1.77	26.10	12.05	28.59

[12] Y.-G. Jiang, J. Wang, X. Xue, and S.-F. Chang. Query-adaptive image search with hash codes. *IEEE Transactions on Multimedia*, 15(2):442–453, 2013.

[13] S. Kim, Y. Kang, and S. Choi. Sequential spectral learning to hash with multiple representations. In *ECCV (5)*, pages 538–551, 2012.

[14] W. Kong and W.-J. Li. Isotropic hashing. In *NIPS*, pages 1655–1663, 2012.

[15] W. Kong, W.-J. Li, and M. Guo. Manhattan hashing for large-scale image retrieval. In *SIGIR*, pages 45–54, 2012.

[16] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009.

[17] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(6):1092–1104, 2012.

[18] Y.-H. Kuo, K.-T. Chen, C.-H. Chiang, and W. H. Hsu. Query expansion for hash-based image object retrieval. In *ACM Multimedia*, pages 65–74, 2009.

[19] Z. Li, H. Ning, L. Cao, T. Zhang, Y. Gong, and T. Huang. Learning to search efficiently in high dimensions. In *NIPS*, pages 1710–1718, 2011.

[20] R. Lin, D. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010.

[21] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[22] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011.

[23] X. Liu, J. He, D. Liu, and B. Lang. Compact kernel hashing with multiple features. In *ACM Multimedia*, pages 881–884, 2012.

[24] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344–3351, 2010.

[25] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *AAAI*, 2010.

[26] J. Nocedal and S. Wright. *Numerical Optimization*, volume 104 of *Springer Serials in Operations Research*. Springer-Verlag, London, 2006.

[27] M. Norouzi and D. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011.

[28] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, pages 1070–1078, 2012.

[29] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009.

[30] B. Russell, A. Torralba, K. Murphy, and W. Freeman. Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008.

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

[32] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT press, 2006.

[33] J. Song, Y. Yang, Z. Huang, H. Shen, and R. Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *ACM Multimedia*, pages 423–432, 2011.

[34] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1):66–78, 2012.

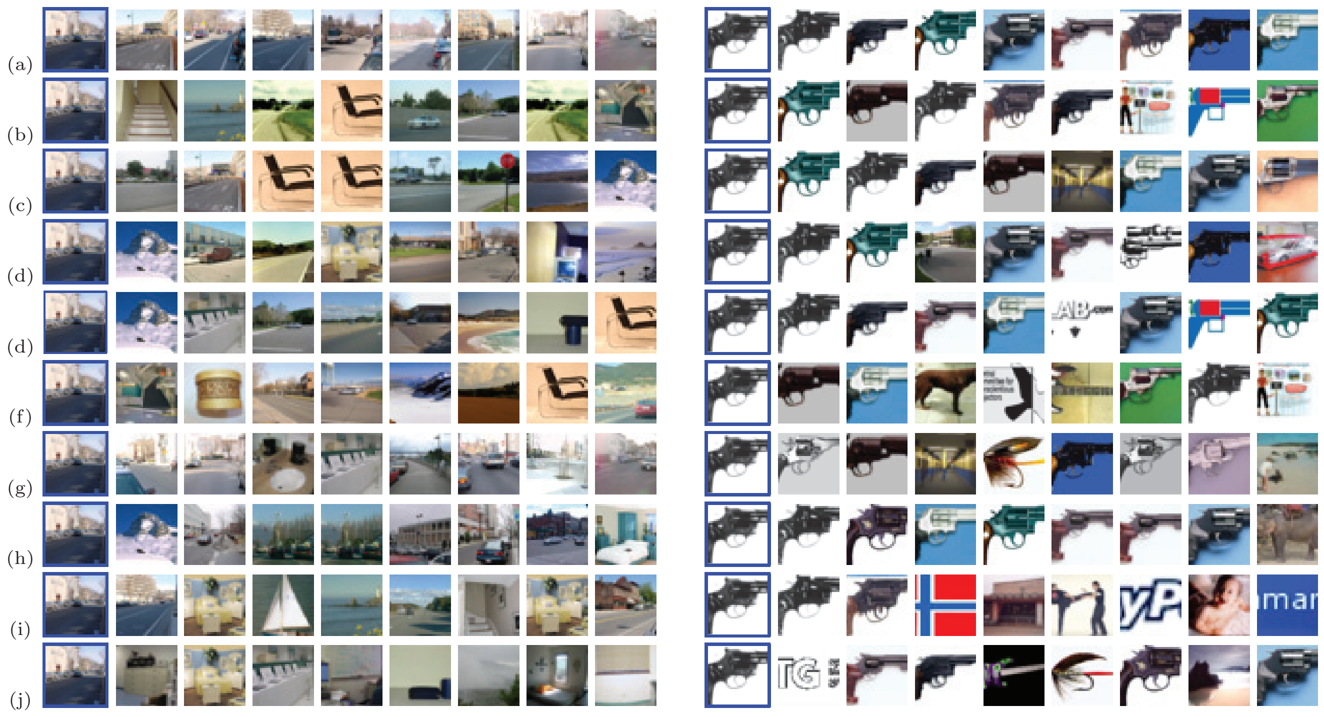


Figure 4: Qualitative results of various approaches. For each row, the first image is a query image and the remaining are the ranking results by the learned hash codes. (a)-(j): OPH, MLH [27], BRE [16], ITQ [3], IsoHash [14], KSH [21], AGH [22], SH [40], USPLH [13], LSH [8].

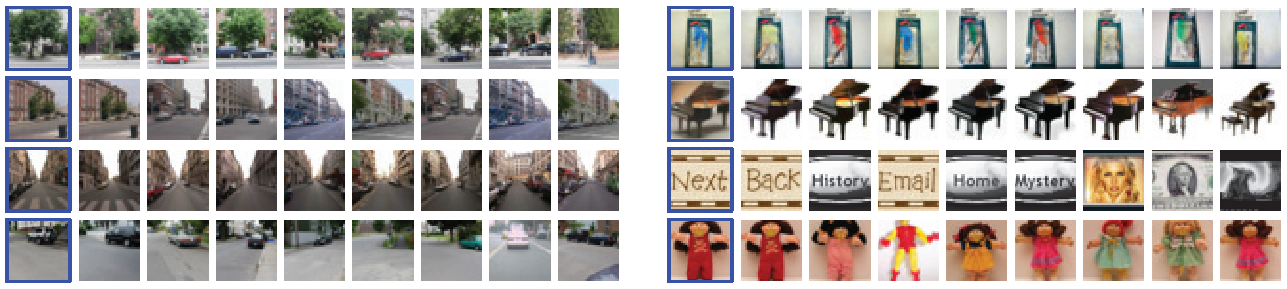


Figure 5: Qualitative results of OPH. For each row, the first image is a query image and the remaining are the ranking results by the learned hash codes.

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

[36] K.-Y. Tseng, Y.-L. Lin, Y.-H. Chen, and W. H. Hsu. Sketch-based image retrieval on mobile devices using compact hash bits. In *ACM Multimedia*, pages 913–916, 2012.

[37] J. Wang, O. Kumar, and S. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010.

[38] J. Wang, S. Kumar, and S. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, pages 1127–1134, 2010.

[39] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *ECCV (5)*, pages 340–353, 2012.

[40] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[41] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *ICCV*, pages 1631–1638, 2011.

[42] J. Yuan, G. Gravier, S. Campion, X. Liu, and H. Jégou. Efficient mining of repetitions in large-scale tv streams with product quantization hashing. In *ECCV Workshops (1)*, pages 271–280, 2012.

[43] Y. Zhuang, Y. Liu, F. Wu, Y. Zhang, and J. Shao. Hypergraph spectral hashing for similarity search of social image. In *ACM Multimedia*, pages 1457–1460, 2011.