A Survey on Learning to Hash

Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen

Abstract—Nearest neighbor search is a problem of finding the data points from the database such that the distances from them to the query point are the smallest. Learning to hash is one of the major solutions to this problem and has been widely studied recently. In this paper, we present a comprehensive survey of the learning to hash algorithms, categorize them according to the manners of preserving the similarities into: pairwise similarity preserving, multiwise similarity preserving, implicit similarity preserving, as well as quantization, and discuss their relations. We separate quantization from pairwise similarity preserving as the objective function is very different though quantization, as we show, can be derived from preserving the pairwise similarities. In addition, we present the evaluation protocols, and the general performance analysis, and point out that the quantization algorithms perform superiorly in terms of search accuracy, search time cost, and space cost. Finally, we introduce a few emerging topics.

Index Terms—Similarity search, approximate nearest neighbor search, hashing, learning to hash, quantization, pairwise similarity preserving, multiwise similarity preserving, implicit similarity preserving.

1 Introduction

The problem of nearest neighbor search, also known as similarity search, proximity search, or close item search, is aimed at finding an item, called nearest neighbor, which is the nearest to a query item under a certain distance measure from a search (reference) database. The cost of finding the exact nearest neighbor is prohibitively high in the case that the reference database is very large or that computing the distance between the query item and the database item is costly. The alternative approach, approximate nearest neighbor search, is more efficient and is shown to be enough and useful for many practical problems, thus attracting an enormous number of research efforts.

Hashing, a widely-studied solution to the approximate nearest neighbor search, aims to transform a data item to a low-dimensional representation, or equivalently a short code consisting of a sequence of bits, called hash code. There are two main categories of hashing algorithms: locality sensitive hashing [14], [44] and learning to hash. Locality sensitive hashing (LSH) is data-independent. Following the pioneering works [14], [44], there are a lot of efforts, such as proposing random hash functions satisfying the locality sensitivity property for various distance measures [10], [11], [14], [19], [20], [100], [111], proving better search efficiency and accuracy [19], [113], developing better search schemes [29], [29], [95], providing a similarity estimator with smaller variance [70], [55], [74], [54], smaller storage [72], [73], or faster computation of hash functions [71], [74], [54], [130]. LSH has been adopted in many applications, e.g., fast object detection [21], image matching [17], [99] The detailed review on LSH can be found in [147].

Learning to hash, the interest of this survey, is a data-dependent hashing approach which aims to learn hash functions from a specific dataset so that the nearest neighbor search result in the hash coding space is as close as possible to the search result in the original space, and the search cost as well as the space cost are also small. The development of learning to hash has been inspired by the connection between the Hamming distance and the distance provided from the original space, e.g., the cosine distance shown in SimHash [14]. Since the two early algorithms, semantic hashing [120], [121] and spectral hashing [156] that learns projection vectors instead of the random projections as done in [14], learning to hash has been attracting a large amount of research interest in computer vision and machine learning and has been applied to a wide-range of applications such as large scale object retrieval [51], image classification [122] and detection [139], and so on.

The main methodology of learning to hash is similarity preserving, i.e., minimizing the gap between the similarities computed/given in the original space and the similarities in the hash coding space in various forms. The similarity in the original space might be from the semantic (class) information, or from the distance (e.g., Euclidean distance) computed in the original space, which is of broad interest and widely studied in real applications, e.g., large scale image search and image classification. Hence the later is the main focus in this paper.

This survey categorizes the algorithms according to the similarity preserving manner into: pairwise similarity preserving, multiwise similarity preserving, implicit similarity preserving, quantization which we will show is also a form of pairwise similarity preserving, as well as an end-to-end hash learning strategy learning the hash codes directly from the object, e.g., image, under the deep learning framework instead of first learning the representations and then learning the hash codes from the representations. In addition, we discuss other problems including evaluation datasets and evaluation schemes, and so on. Meanwhile, we present the empirical observation that the quantization approach

- *J. Wang is with Microsoft Research, Beijing, P.R. China. E-mail: jingdw@microsoft.com*
- *T. Zhang is with University of Science and Technology of China. Email: zting@mail.ustc.edu.cn*
- *J. Song and N. Sebe is with Department of Information Engineering and Computer Science, University of Trento, Italy. Email: {jingkuan.song, niculae.sebe}@unitn.it*
- *H.T. Shen is with School of Computer Science and Engineering, University of Electronic Science and Technology of China. Email: shenhengtao@hotmail.com Corresponding author: Heng Tao Shen.*

outperforms other approaches and give some analysis about this observation.

In comparison to other surveys on learning to hash [145], [147], this survey focuses more on learning to hash, discusses more on quantization-based solutions. Our categorization methodology is helpful for readers to understand connections and differences between existing algorithms. In particular, we point out the empirical observation that quantization is superior in terms of search accuracy, search efficiency and space cost

2 Background

2.1 Nearest Neighbor Search

Exact nearest neighbor search is defined as searching an item $\mathrm{NN}(\mathbf{q})$ (called nearest neighbor) for a query item $\mathbf{q}$ from a set of N items $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ so that $\mathrm{NN}(\mathbf{q}) = \arg\min_{\mathbf{x}\in\mathcal{X}} \mathrm{dist}(\mathbf{q}, \mathbf{x})$, where $\mathrm{dist}(\mathbf{q}, \mathbf{x})$ is a distance computed between $\mathbf{q}$ and $\mathbf{x}$. A straightforward generalization is K-nearest neighbor search, where K nearest neighbors are needed to be found.

The distance between a pair of items $\mathbf{x}$ and $\mathbf{q}$ depends on the specific nearest search problem. A typical example is that the search (reference) database $\mathcal{X}$ lies in a d-dimensional space $\mathbb{R}^d$ and the distance is introduced by an ℓ_s norm, $\|\mathbf{x} - \mathbf{q}\|_s = (\sum_{i=1}^{d} |x_i - q_i|^s)^{1/s}$. The search problem under the Euclidean distance, i.e., the ℓ_2 norm, is widely studied. Other forms of the data item, for example, the data item is formed by a set, and other forms of distance measures, such as ℓ_1 distance, cosine similarity and so on, are also possible.

There exist efficient algorithms (e.g., k-d trees) for exact nearest neighbor search in low-dimensional cases. In large scale high-dimensional cases, it turns out that the problem becomes hard and most algorithms even take higher computational cost than the naive solution, i.e., the linear scan. Therefore, a lot of recent efforts moved to searching approximate nearest neighbors: error-constrained nearest (near) neighbor search, and time-constrained approximate nearest neighbor search [103], [105]. The error-constrained search includes (randomized) $(1+\epsilon)$-approximate nearest neighbor search [1], [14], [44], (approximate) fixed-radius near neighbor (R-near neighbor) search [6], and so on.

Time-constrained approximate nearest neighbor search limits the time spent during the search and is studied mostly for real applications, though it usually lacks an elegant theory behind. The goal is to make the search as accurate as possible by comparing the returned K approximate nearest neighbors and the K exact nearest neighbors, and to make the query cost as small as possible. For example, when comparing the learning to hash approaches that use linear scan based on the Hamming distance for search, it is typically assumed that the search time is the same for the same code length by ignoring other small cost. When comparing the indexing structure algorithms, e.g., tree-based [103], [105], [152] or neighborhood graph-based [151], the time-constrained search is usually transformed to another approximate way: terminate the search after examining a fixed number of data points.

2.2 Search with Hashing

The hashing approach aims to map the reference (and query) items to the target items so that approximate nearest neighbor search is efficiently and accurately performed by resorting to the target items and possibly a small subset of the raw reference items. The target items are called hash codes (a.k.a., hash values, or simply hashes). In this paper, we may also call it short/compact codes interchangeably.

The hash function is formally defined as: $y = h(\mathbf{x})$, where y is the hash code, may be an integer, or a binary value: 1 and 0 (or -1), and $h(\cdot)$ is the hash function. In the application to approximate nearest neighbor search, usually several hash functions are used together to compute the compound hash code: $\mathbf{y} = \mathbf{h}(\mathbf{x})$, where $\mathbf{y} = [y_1\ y_2\ \cdots\ y_M]^\top$ and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x})\ h_2(\mathbf{x})\ \cdots\ h_M(\mathbf{x})]^\top$. Here we use a vector $\mathbf{y}$ to represent the compound hash code for convenience.

There are two basic strategies for using hash codes to perform nearest (near) neighbor search: *hash table lookup* and *hash code ranking*. The search strategies are illustrated in Figure 1.

The main idea of *hash table lookup* for accelerating the search is reducing the number of the distance computations. The data structure, called hash table (a form of inverted index), is composed of buckets with each bucket indexed by a hash code. Each reference item $\mathbf{x}$ is placed into a bucket $\mathbf{h}(\mathbf{x})$. Different from the conventional hashing algorithm in computer science that avoids collisions (i.e., avoids mapping two items into some same bucket), the hashing approach using a hash table essentially aims to maximize the probability of collision of near items and at the same time minimize the probability of collision of the items that are far away. Given the query $\mathbf{q}$, the items lying in the bucket $\mathbf{h}(\mathbf{q})$ are retrieved as the candidates of the nearest items of $\mathbf{q}$. Usually this is followed by a reranking step: rerank the retrieved nearest neighbor candidates according to the true distances computed using the original features and attain the nearest neighbors.

To improve the recall, two ways are often adopted. The first way is to visit a few more buckets (but with a single hash table), whose corresponding hash codes are the nearest to (the hash code $\mathbf{h}(\mathbf{q})$ of) the query according to the distances in the coding space. The second way is to construct several (e.g., L) hash tables. The items lying in the L hash buckets $\mathbf{h}_1(\mathbf{q}), \cdots, \mathbf{h}_L(\mathbf{q})$ are retrieved as the candidates of near items of $\mathbf{q}$ which are possibly ordered according to the number of hits of each item in the L buckets. To guarantee the high precision, each of the L hash codes, $\mathbf{y}_l$, needs to be a long code. This means that the total number of the buckets is too large to index directly, and thus the buckets that are non-empty are retained by using the conventional hashing over the hash codes $\mathbf{h}_l(\mathbf{x})$.

The second way essentially stores multiple copies of the id for each reference item. Consequently, the space cost is larger. In contrast, the space cost for the first way is smaller as it only uses a single table and stores one copy of the id for each reference item, but it needs to access more buckets to guarantee the same recall with the second way. The multiple assignment scheme is also studied: construct a single table, but assign a reference item to multiple hash buckets. In essence, it is shown that the second way, multiple hash

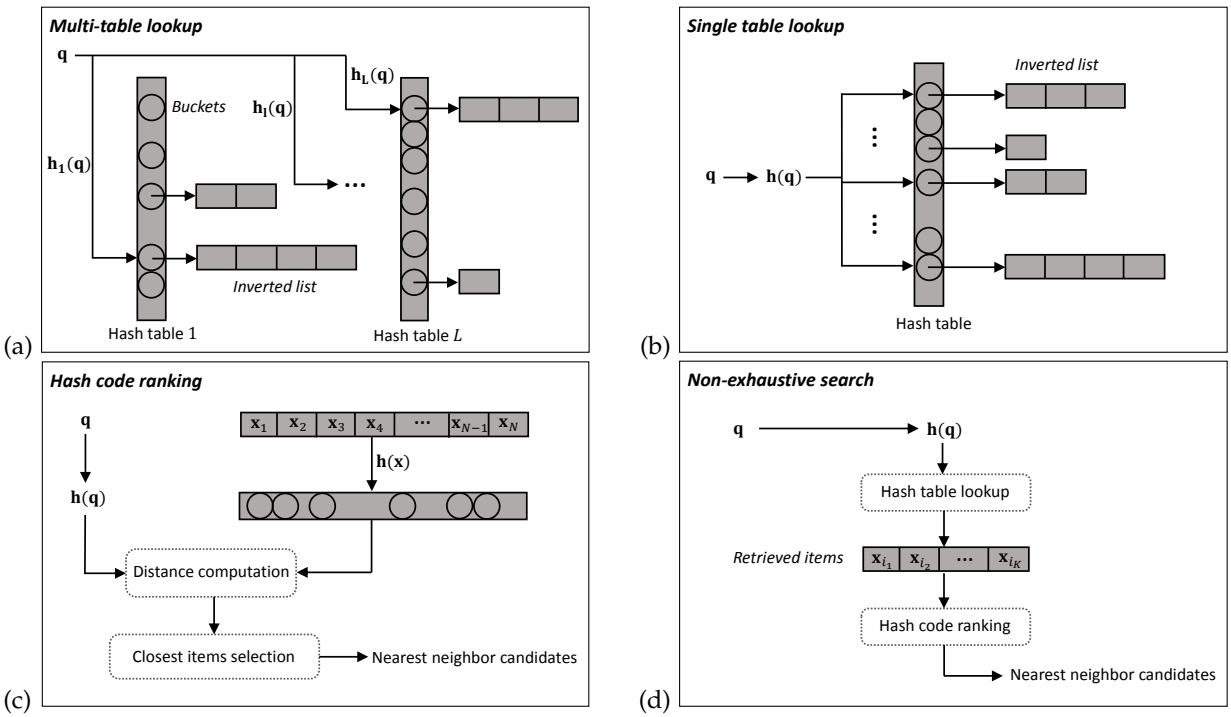


Fig. 1. Illustrating the search strategies. (a) Multi table lookup: the list corresponding to the hash code of the query in each table is retrieved. (b) Single table lookup: the lists corresponding to and near to the hash code of the query are retrieved. (c) Hash code ranking: compare the query with each reference item in the coding space. (d) Non-exhaustive search: hash table lookup (or other inverted index structure) retrieves the candidates, followed by hash code ranking over the candidates.

tables, can be regarded as a form of multiple assignment.

Hash code ranking performs an exhaustive search: compare the query with each reference item by fast evaluating their distance (e.g., using distance table lookup or using the CPU instruction __popcnt for Hamming distance) according to (the hash code of) the query and the hash code of the reference item, and retrieve the reference items with the smallest distances as the candidates of nearest neighbors. Usually this is followed by a reranking step: rerank the retrieved nearest neighbor candidates according to the true distances computed using the original features and attain the nearest neighbors.

This strategy exploits one main advantage of hash codes: the distance using hash codes is efficiently computed and the cost is much smaller than that of the distance computation in the original input space.

Comments: Hash table lookup is mainly used in locality sensitive hashing, and has been used for evaluating learning to hash in a few publications. It has been pointed in [156] and also observed from empirical results that LSH-based hash table lookup, except min-hash, is rarely adopted in reality, while hash table lookup with quantization-based hash codes is widely used in the non-exhaustive strategy to retrieve coarse candidates [50]. Hash code ranking goes through all the candidates and thus is inferior in search efficiency compared with hash table lookup which only checks a small subset of candidates, which are determined by a lookup radius.

A practical way is to do a non-exhaustive search which is suggested in [4], [50]: first retrieve a small set of candidates using the inverted index that can be viewed as a hash table, and then compute the distances of the query to the candidates using the hash codes which are longer, providing the top candidates subsequently reranked using the original features. Other research efforts include organizing the hash codes to avoid the exhaustive search with a data structure, such as a tree or a graph structure [104].

3 Learning to Hash

Learning to hash is the task of learning a (compound) hash function, $\mathbf{y} = \mathbf{h}(\mathbf{x})$, mapping an input item $\mathbf{x}$ to a compact code $\mathbf{y}$, aiming that the nearest neighbor search result for a query $\mathbf{q}$ is as close as possible to the true nearest search result and the search in the coding space is also efficient. A learning-to-hash approach needs to consider five problems: what hash function $\mathbf{h}(\mathbf{x})$ is adopted, what similarity in the coding space is used, what similarity is provided in the input space, what loss function is chosen for the optimization objective, and what optimization technique is adopted.

3.1 Hash Function

The hash function can be based on linear projection, kernels, spherical function, (deep) neural networks, a non-parametric function, and so on. One popular hash function is the linear hash function, e.g., [136], [141]:

$$y = h(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b), \tag{1}$$

where $\mathrm{sgn}(z) = 1$ if $z \geqslant 0$ and $\mathrm{sgn}(z) = 0$ (or equivalently -1) otherwise, $\mathbf{w}$ is the projection vector, and b is the bias variable. The kernel function,

$$y = h(\mathbf{x}) = \mathrm{sgn}\left(\sum\nolimits_{t=1}^{T} w_t K(\mathbf{s}_t, \mathbf{x}) + b\right), \tag{2}$$

is also adopted in some approaches, e.g., [40], [66], where $\{\mathbf{s}_t\}$ is a set of representative samples that are randomly

drawn from the dataset or cluster centers of the dataset and $\{w_t\}$ are the weights. The non-parametric function based on nearest vector assignment is widely used for quantization-based solutions:

$$y = \arg\min_{k\in\{1,\cdots,K\}} \|\mathbf{x} - \mathbf{c}_k\|_2, \tag{3}$$

where $\{\mathbf{c}_1, \cdots, \mathbf{c}_K\}$ is a set of centers computed by some algorithms, e.g., K-means, and $y \in \mathbb{Z}^+$ is an integer. In contrast to other hashing algorithms in which the distance, e.g., Hamming distance, is often directly computed from hash codes, the hash codes generated from the nearest vector assignment-based hash function are the indices of the nearest vectors, and the distance is computed using the centers corresponding to the hash codes.

The form of hash function is an important factor influencing the search accuracy using the hash codes, as well as the time cost of computing hash codes. A linear function is efficiently evaluated, while the kernel function and the nearest vector assignment based function lead to better search accuracy as they are more flexible. Almost all the methods using a linear hash function can be extended to nonlinear hash functions, such as kernelized hash functions, or neural networks. Thus we do not use the hash function to categorize the hash algorithms.

3.2 Similarity

In the input space the distance d^o_{ij} between any pair of items $(\mathbf{x}_i, \mathbf{x}_j)$ could be the Euclidean distance, $\|\mathbf{x}_i - \mathbf{x}_j\|_2$ or others. The similarity s^o_{ij} is often defined as a function about the distance d^o_{ij}, and a typical function is the Gaussian function: $s^o_{ij} = g(d^o_{ij}) = \exp(-\frac{(d^o_{ij})^2}{2\sigma^2})$. There exist other similarity forms, such as cosine similarity $\frac{\mathbf{x}_i^\top \mathbf{x}_j}{\|\mathbf{x}_i\|_2\|\mathbf{x}_j\|_2}$ and so on. Besides, the semantic similarity is often used for semantic similarity search. In this case, the similarity s^o_{ij} is usually binary, valued 1 if the two items $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same semantic class, 0 (or -1) otherwise. The hashing algorithms for semantic similarity usually can be applied to other distances, such as Euclidean distance, by defining a pseudo-semantic similarity: $s^o_{ij} = 1$ for nearby points (i, j) and $s^o_{ij} = 0$ (or -1) for farther points (i, j).

In the hash coding space, the typical distance d^h_{ij} between $\mathbf{y}_i$ and $\mathbf{y}_j$ is the Hamming distance. It is defined as the number of bits where the values are different and is mathematically formulated as

$$d^h_{ij} = \sum\nolimits_{m=1}^{M} \delta[y_{im} \neq y_{jm}],$$

which is equivalent to $d^h_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|_1$ if the code is valued by 1 and 0. The distance for the codes valued by 1 and -1 is similarly defined. The similarity based on the Hamming distance is defined as $s^h_{ij} = M - d^h_{ij}$ for the codes valued by 1 and 0, computing the number of bits where the values are the same. The inner product $s^h_{ij} = \mathbf{y}_i^\top \mathbf{y}_j$ is used as the similarity for the codes valued by 1 and -1. These measures are also extended to the weighted case: e.g., $d^h_{ij} = \sum_{m=1}^{M} \lambda_m \delta[y_{im} \neq y_{jm}]$ and $s^h_{ij} = \mathbf{y}_i^\top \mathbf{\Lambda} \mathbf{y}_j$, where $\mathbf{\Lambda} = \mathrm{Diag}(\lambda_1, \lambda_2, \cdots, \lambda_M)$ is a diagonal matrix and each diagonal entry is the weight of the corresponding hash code.

Besides the Hamming distance/similarity and its variants, the Euclidean distance is typically used in quantization approaches, and is evaluated between the vectors corresponding to the hash codes, $d^h_{ij} = \|\mathbf{c}_{y_i} - \mathbf{c}_{y_j}\|_2$ (symmetric distance) or between the query $\mathbf{q}$ and the center that is the approximation to $\mathbf{x}_j$, $d^h_{qj} = \|\mathbf{q} - \mathbf{c}_{y_j}\|_2$ (asymmetric distance, which is preferred because the accuracy is higher and the time cost is almost the same). The distance is usually evaluated in the search stage efficiently by using a distance lookup table. There are also some works learning/optimizing the distances between hash codes [37], [148] after the hash codes are already computed.

3.3 Loss Function

The basic rule of designing the loss function is to preserve the similarity order, i.e., minimize the gap between the approximate nearest neighbor search result computed from the hash codes and the true search result obtained from the input space.

The widely-used solution is pairwise similarity preserving, making the distances or similarities between a pair of items from the input and coding spaces as consistent as possible. The multiwise similarity preserving solution, making the order among multiple items computed from the input and coding spaces as consistent as possible, is also studied. One class of solutions, e.g., spatial partitioning, implicitly preserve the similarities. The quantization-based solution and other reconstruction-based solutions aim to find the optimal approximation of the item in terms of the reconstruction error through a reconstruction function (e.g., in the form of a lookup table in quantization or an auto-encoder in [120]). Besides similarity preserving items, some approaches introduce bucket balance or its approximate variants as extra constraints, which is also important for obtaining better results or avoiding trivial solutions.

3.4 Optimization

The challenges for optimizing the hash function parameters lie in two main factors. One is that the problem contains the sgn function, which leads to a challenging mixed-binary-integer optimization problem. The other is that the time complexity is high when processing a large number of data points, which is usually handled by sampling a subset of points or a subset of constraints (or equivalent basic terms in the objective functions).

The ways to handle the sgn function are summarized below. The first way is the most widely-adopted continuous relaxation, including sigmoid relaxation, tanh relaxation, and directly dropping the sign function $\mathrm{sgn}(z) \approx z$. The relaxed problem is then solved using various standard optimization techniques. The second one is a two-step scheme [76], [77] with its extension to alternative optimization [32]: optimizing the binary codes without considering the hash function, followed by estimating the function parameters from the optimized hash codes. The third one is discretization: drop the sign function $\mathrm{sgn}(z) \approx z$ and regard the hash code as an approximation of the hash function, which is formulated as a loss $(y-z)^2$. There also exist other ways only adopted in a few algorithms, e.g., transforming the problem into a latent structure-SVM formulation in [107], [109] , the coordinate-descent approach in [66] (fixing all but one weight, optimize the original objective with respect to a single weight in

each iteration), both of which do not conduct continuous relaxation.

3.5 Categorization

Our survey categorizes the existing algorithms to various classes: the pairwise similarity preserving class, the multi-wise similarity preserving class, the implicit similarity preserving class, as well as the quantization class, according to what similarity preserving manner is adopted to formulate the objective function. We separate the quantization class from the pairwise similarity preserving class as they are very different in formulation though the quantization class can be explained from the perspective of pairwise similarity preserving. In the following description, we may call quantization as quantization-based hashing and other algorithms in which a hash function generates a binary value as binary code hashing. In addition, we will also discuss other studies on learning to hash. The summary of the representative algorithms is given in Table 1.

The main reason we choose the similarity preserving manner to do the categorization is that similarity preservation is the essential goal of hashing. It should be noted that as pointed in [145], [147], other factors, such as the hash function, or the optimization algorithm, is also important for the search performance.

4 Pairwise Similarity Preserving

The algorithms aligning the distances or similarities of a pair of items computed from the input space and the Hamming coding space are roughly divided in the following groups:

- Similarity-distance product minimization (SDPM): $\min \sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij}$. The distance in the coding space is expected to be smaller if the similarity in the original space is larger. Here $\mathcal{E}$ is a set of pairs of items that are considered.
- Similarity-similarity product maximization (SSPM): $\max \sum_{(i,j)\in\mathcal{E}} s^o_{ij} s^h_{ij}$. The similarity in the coding space is expected to be larger if the similarity in the original space is larger.
- Distance-distance product maximization (DDPM): $\max \sum_{(i,j)\in\mathcal{E}} d^o_{ij} d^h_{ij}$. The distance in the coding space is expected to be larger if the distance in the original space is larger.
- Distance-similarity product minimization (DSPM): $\min \sum_{(i,j)\in\mathcal{E}} d^o_{ij} s^h_{ij}$. The similarity in the coding space is expected to be smaller if the distance in the original space is larger.
- Similarity-similarity difference minimization (SSDM): $\min \sum_{(i,j)\in\mathcal{E}} (s^o_{ij} - s^h_{ij})^2$. The difference between the similarities is expected to be as small as possible.
- Distance-distance difference minimization (DDDM): $\min \sum_{(i,j)\in\mathcal{E}} (d^o_{ij} - d^h_{ij})^2$. The difference between the distances is expected to be as small as possible.
- Normalized similarity-similarity divergence minimization (NSSDM): $\min \mathrm{KL}(\{\bar{s}^o_{ij}\}, \{\bar{s}^h_{ij}\}) = \min(-\sum_{(i,j)\in\mathcal{E}} \bar{s}^o_{ij} \log \bar{s}^h_{ij})$. Here $\bar{s}^o_{ij}$ and $\bar{s}^h_{ij}$ are normalized similarities in the input space and the coding space: $\sum_{ij} \bar{s}^o_{ij} = 1$ and $\sum_{ij} \bar{s}^h_{ij} = 1$.

The following reviews these groups of algorithms except the distance-similarity product minimization group for which we are not aware of any algorithm belonging to. It should be noted that merely optimizing the above similarity preserving function, e.g., SDPM and SSPM, is not enough and may lead to trivial solutions, and it is necessary to combine other constraints, which is detailed in the following discussion. We also point out the relation between similarity-distance product minimization and similarity-similarity product maximization, the relation between similarity-similarity product maximization and similarity-similarity difference minimization, as well as the relation between distance-distance product maximization and distance-distance difference minimization.

4.1 Similarity-Distance Product Minimization

We first introduce spectral hashing and its extensions, and then review other forms.

4.1.1 Spectral Hashing

The goal of spectral hashing [156] is to minimize $\sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij}$, where the Euclidean distance in the hashing space, $d^h_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|_2^2$, is used for formulation simplicity and optimization convenience, and the similarity in the input space is defined as: $s^o_{ij} = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2})$. Note that the Hamming distance in the search stage can be still used for higher efficiency as the Euclidean distance and the Hamming distance in the coding space are consistent: the larger the Euclidean distance, the larger the Hamming distance. The objective function can be written in a matrix form,

$$\min \sum\nolimits_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij} = \mathrm{trace}(\mathbf{Y}(\mathbf{D} - \mathbf{S})\mathbf{Y}^\top), \qquad (4)$$

where $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2 \cdots \mathbf{y}_N]$ is a matrix of $M \times N$, $\mathbf{S} = [s^o_{ij}]_{N\times N}$ is the similarity matrix, and $\mathbf{D} = \mathrm{diag}(d_{11}, \cdots, d_{NN})$ is a diagonal matrix, $d_{nn} = \sum_{i=1}^N s^o_{ni}$.

There is a trivial solution to the problem (4): $\mathbf{y}_1 = \mathbf{y}_2 = \cdots = \mathbf{y}_N$. To avoid it, the code balance condition is introduced: the number of data items mapped to each hash code is the same. Bit balance and bit uncorrelation are used to approximate the code balance condition. Bit balance means that each bit has about 50% chance of being 1 or -1. Bit uncorrelation means that different bits are uncorrelated. The two conditions are formulated as,

$$\mathbf{Y}\mathbf{1} = 0, \quad \mathbf{Y}\mathbf{Y}^\top = \mathbf{I}, \qquad (5)$$

where $\mathbf{1}$ is an N-dimensional all-1 vector, and $\mathbf{I}$ is an identity matrix of size N.

Under the assumption of separate multi-dimensional uniform data distribution, the hashing algorithm is given as follows,

1) Find the principal components of the N d-dimensional reference data items using principal component analysis (PCA).
2) Compute the M one-dimensional Laplacian eigenfunctions with the M smallest eigenvalues along each PCA direction (d directions in total).

TABLE 1
A summary of representative hashing algorithms with respect to similarity preserving functions, code balance, hash function similarity in the coding space, and the manner to handle the sgn function. pres. = preserving, sim. = similarity. BB = bit balance, BU = bit uncorrelation, BMIM = bit mutual information minimization, BKB = bucket balance. H = Hamming distance, WH = weighted Hamming distance, SH = spherical Hamming distance, C = Cosine, E = Euclidean distance, DNN = deep neural networks; Drop = drop the sgn operator in the hash function, Sigmoid = Sigmoid relaxation, $[a, b]$ = $[a, b]$ bounded relaxation, Tanh = Tanh relaxation, Discretize = drop the sgn operator in the hash function and regard the hash code as a discrete approximation of the hash value, Keep = optimization without relaxation for sgn, Two-step = two-step optimization.

Approach		Similarity pres.	Code balance	Hash function	Code sim.	sgn
Pairwise	Spectral hashing [156] (2008)	$s_{ij}^o d_{ij}^h$	BB + BU	Eigenfunction	H	Drop
	ICA hashing [39] (2011)		BB + BMIM	Linear		Drop
	Kernelized spectral hashing [40] (2010)		BB + BU	Kernel		Drop
	Hashing with graphs [86] (2011)		BB + BU	Eigenfunction		Drop
	Discrete graph hashing [84] (2014)		BB + BU	Kernel		Discretize
	Kernelized Discrete graph hashing [128] (2016)		BB + BU	Kernel		Discretize & Two-step
	Self-taught hashing [166] (2010)		BB + BU	Linear		Two-step
	LDA hashing [136] (2012)		BU	Linear		Drop
	Minimal loss hashing [107] (2011)		-	Linear		Keep
	Deep supervised hashing [82] (2016)		-	DNN		Drop
	Semi-supervised hashing [141], [142], [143] (2010)	$s_{ij}^o s_{ij}^h$	BB+BU	Linear	H	Drop
	Topology preserving hashing [169] (2013)	$d_{ij}^o d_{ij}^h + s_{ij}^o d_{ij}^h$	BB+BU	Linear	H	Drop
	Binary reconstructive embedding [66] (2009)	$(d_{ij}^o - d_{ij}^h)^2$	-	Kernel	H	Keep
	ALM or NMF based hashing [106] (2015)		-	Linear		$[0, 1]$ + two-step
	Compressed hashing [79](2013)		BB	Kernel		Drop
	Supervised hashing with kernels [85] (2012)	$(s_{ij}^o - s_{ij}^h)^2$	-	Kernel	H	Sigmoid
	Bilinear hyperplane hashing [87] (2012)		-	BiLinear		Sigmoid
	Label-regularized maximum margin hashing [102] (2010)		BB	Kernel		Drop
	Scalable graph hashing [56] (2015)		BU	Kernel		Drop
	Binary hashing [25] (2016)		-	Kernel		Two-step
	CNN hashing [158] (2014)		-	DNN		$[-1, 1]$
	Multi-dimensional spectral hashing [155] (2012)		BI + BU	Eigenfunction	WH	Drop
	Spec hashing [78] (2010)	$\mathrm{KL}(\{\bar{s}_{ij}^o\}, \{\bar{s}_{ij}^h\})$	-	Decision stump	H	Two-step
Multiwise	Order preserving hashing [150] (2013)	Rank order	BKB	Linear	H	Sigmoid
	Top rank supervised binary coding [132] (2015)	Triplet loss	-	Linear	H	Tanh
	Triplet loss hashing [109] (2012)		-	Linear + NN		Keep
	Deep semantic ranking based hashing [174] (2015)		-	DNN		Sigmoid
	Simultaneous Feature Learning and Hash Coding [67] (2015)		-	DNN		Drop
	Listwise supervision hashing [146] (2013)		BU	Linear		Drop
Implicit	Picodes [7] (2011)	-	-	Linear	H	Keep
	Random maximum margin hashing [61] (2011)		BB	Kernel	H	Keep
	Complementary projection hashing [60] (2013)		BB+BU	Kernel	H	Drop
	Spherical hashing [41] (2012)		BB	Spherical	SH	Keep
Quantization	Isotropic hashing [63] (2012)	$\approx \|\|\mathbf{x} - \mathbf{y}\|\|_2$	BU	Linear	H	Drop
	Iterative quantization [35], [36] (2011)	$\|\|\mathbf{x} - \mathbf{y}\|\|_2$	-	Linear	H	Keep
	Harmonious hashing [159] (2013)		BB+BU			Drop
	Matrix hashing [33] (2013)		-			Keep
	Angular quantization [34] (2012)		-		C	Keep
	Deep hashing [80] (2015)		BB+BU	DNN	H	Discretize
	Hashing with binary deep neural network [24] (2016)		BB+BU	DNN	H	Discretize
	Product quantization (PQ) [50] (2011)	$\|\|\mathbf{x} - \mathbf{y}\|\|_2$	-	Nearest vector	E	-
	Cartesian k-means [108] (Optimized PQ [31]) (2013)		-			-
	Composite quantization [171] (2014)		-			-
	Additive quantization [2] (2014)		-			-
	Revisiting additive quantization [96] (2016)		-			-
	Quantized sparse representations [47] (2016)		-			-
	Supervised discrete hashing [125] (2015)	$\|\|\mathbf{x} - \mathbf{y}\|\|_2$	-	Kernel	H	Discretize
	Supervised quantization [154] (2016)		-	Nearest vector	E	-

3) Pick the M eigenfunctions with the smallest eigenvalues among Md eigenfunctions.
4) Threshold the eigenfunction at zero, obtaining the binary codes.

The one-dimensional Laplacian eigenfunction for the case of uniform distribution on $[r_l, r_r]$ is $\phi_m(x) = \sin(\frac{\pi}{2} + \frac{m\pi}{r_r - r_l}x)$, and the corresponding eigenvalue is $\lambda_m = 1 - \exp(-\frac{\epsilon^2}{2}|\frac{m\pi}{r_r - r_l}|^2)$, where m $(= 1, 2, \cdots)$ is the frequency and ϵ is a fixed small value. The hash function is formally written as $h(\mathbf{x}) = \mathrm{sgn}(\sin(\frac{\pi}{2} + \gamma \mathbf{w}^\top \mathbf{x}))$, where γ depends on the frequency m and the range of the projection along the direction $\mathbf{w}$.

Analysis: In the case the spreads along the top M PCA directions are the same, the hashing algorithm partitions each direction into two parts using the median (due to the bit balance requirement) as the threshold, which is equivalent to thresholding at the mean value under the assumption of uniform data distributions. In the case that the true data distribution is a multi-dimensional isotropic Gaussian distribution, the algorithm is equivalent to two quantization algorithms: iterative quantization [36], [35] and isotropic hashing [63].

Regarding the performance, this method performs well for a short hash code but poor for a long hash code. The reason includes three aspects. First, the assumption that the data follow a uniform distribution does not hold in real cases. Second, the eigenvalue monotonously decreases with respect to $|\frac{m}{r_r - r_l}|^2$, which means that the PCA direction with a large spread ($|r_r - r_l|$) and a lower frequency (m) is preferred. Hence there might be more than one eigenfunction picked along a single PCA direction, which breaks the uncorrelation requirement. Last, thresholding the eigenfunction $\phi_m(x) = \sin(\frac{\pi}{2} + \frac{m\pi}{r_r - r_l}x)$ at zero leads to that near points may be mapped to different hash values and farther points may be mapped to the same hash value. As a result, the Hamming distance is not well consistent to the

distance in the input space.

Extensions: There are some extensions using PCA. (1) *Principal component hashing* [98] uses the principal direction to formulate the hash function; (2) *Searching with expectations* [123] and *transform coding* [9] that transforms the data using PCA and then adopts the rate distortion optimization (bits allocation) approach to determine which principal direction is used and how many bits are assigned to such a direction; (3) *Double-bit quantization* that handles the third drawback in spectral hashing by distributing two bits into each projection direction, conducting only 3-cluster quantization, and assigning 01, 00, and 11 to each cluster. Instead of PCA, *ICA hashing* [39] adopts independent component analysis for hashing and uses bit balance and bit mutual information minimization for code balance.

There are many other extensions in a wide range, including similarity graph extensions [75], [179], [92], [86], [84], [79], [128], [170], hash function extensions [40], [124], weighted Hamming distance [153], self-taught hashing [166], sparse hash codes [177], discrete hashing [164], and so on.

4.1.2 Variants

Linear discriminant analysis (LDA) hashing [136] minimizes a form of the loss function: $\min \sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij}$, where $d^h_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|_2^2$. Different from spectral hashing, (1) $s^o_{ij} = 1$ if data items $\mathbf{x}_i$ and $\mathbf{x}_j$ are a similar pair, $(i,j) \in \mathcal{E}^+$, and $s^o_{ij} = -1$ if data items $\mathbf{x}_i$ and $\mathbf{x}_j$ are a dissimilar pair, $(i,j) \in \mathcal{E}^-$ (2) a linear hash function is used: $\mathbf{y} = \mathrm{sgn}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$, and (3) a weight α is imposed to $s^o_{ij} d^h_{ij}$ for the similar pair. As a result, the objective function is written as:

$$\alpha \sum\nolimits_{(i,j)\in\mathcal{E}^+} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 - \sum\nolimits_{(i,j)\in\mathcal{E}^-} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2. \quad (6)$$

The projection matrix $\mathbf{W}$ and the threshold $\mathbf{b}$ are separately optimized: (1) to estimate the orthogonal matrix $\mathbf{W}$, drop the sgn function in Equation (6), leading to an eigenvalue decomposition problem; (2) estimate $\mathbf{b}$ by minimizing Equation (6) with fixed $\mathbf{W}$ through a simple 1D search scheme. A similar loss function, contrastive loss, is adopted in [18] with a different optimization technique.

The loss function in *minimal loss hashing* [107] is in the form of $\min \sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij}$. Similar to LDA hashing, $s^o_{ij} = 1$ if $(i,j) \in \mathcal{E}^+$ and $s^o_{ij} = -1$ if $(i,j) \in \mathcal{E}^-$. Differently, the distance is hinge-like: $d^h_{ij} = \max(\|\mathbf{y}_i - \mathbf{y}_j\|_1 + 1, \rho)$ for $(i,j) \in \mathcal{E}^+$ and $d^h_{ij} = \min(\|\mathbf{y}_i - \mathbf{y}_j\|_1 - 1, \rho)$ for $(i,j) \in \mathcal{E}^-$. The intuition is that there is no penalty if the Hamming distance for similar pairs is small enough and if the Hamming distance for dissimilar pairs is large enough. The formulation, if ρ is fixed, is equivalent to,

$$\begin{aligned}\min\ & \sum\nolimits_{(i,j)\in\mathcal{E}^+} \max(\|\mathbf{y}_i - \mathbf{y}_j\|_1 - \rho + 1, 0) \\ & + \sum\nolimits_{(i,j)\in\mathcal{E}^-} \lambda \max(\rho - \|\mathbf{y}_i - \mathbf{y}_j\|_1 + 1, 0), \end{aligned} \quad (7)$$

where ρ is a hyper-parameter used as a threshold in the Hamming space to differentiate similar pairs from dissimilar pairs, λ is another hyper-parameter that controls the ratio of the slopes for the penalties incurred for similar (or dissimilar) points. The hash function is in the linear form: $\mathbf{y} = \mathrm{sgn}(\mathbf{W}^\top \mathbf{x})$. The projection matrix $\mathbf{W}$ is estimated by transforming $\mathbf{y} = \mathrm{sgn}(\mathbf{W}^\top \mathbf{x}) = \arg\max_{\mathbf{y}'\in\mathcal{H}} \mathbf{h}'^\top \mathbf{W}^\top \mathbf{x}$ and optimizing using structured prediction with latent variables. The hyper-parameters ρ and λ are chosen via cross-validation.

Comments: Besides the optimization techniques, the main differences of the three representative algorithms, i.e., spectral hashing, LDA hashing, and minimal loss hashing, are twofold. First, the similarity in the input space in spectral hashing is defined as a continuous positive number computed from the Euclidean distance, while in LDA hashing and minimal loss hashing the similarity is set to 1 for a similar pair and -1 for a dissimilar pair. Second, the distance in the hashing space for minimal loss hashing is different from spectral hashing and LDA hashing.

4.2 Similarity-Similarity Product Maximization

Semi-supervised hashing [141], [142], [143] is the representative algorithm in this group. The objective function is $\max \sum_{(i,j)\in\mathcal{E}} s^o_{ij} s^h_{ij}$. The similarity s^o_{ij} in the input space is 1 if the pair of items $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same class or are nearby points, and -1 otherwise. The similarity in the coding space is defined as $s^h_{ij} = \mathbf{y}_i^\top \mathbf{y}_j$. Thus, the objective function is rewritten as maximizing:

$$\sum\nolimits_{(i,j)\in\mathcal{E}} s^o_{ij} \mathbf{y}_i^\top \mathbf{y}_j. \quad (8)$$

The hash function is in a linear form $\mathbf{y} = \mathbf{h}(\mathbf{x}) = \mathrm{sgn}(\mathbf{W}^\top \mathbf{x})$. Besides, the bit balance is also considered, and is formulated as maximizing the variance, $\mathrm{trace}(\mathbf{Y}\mathbf{Y}^\top)$, rather than letting the mean be 0, $\mathbf{Y}\mathbf{1} = 0$. The overall objective is to maximize

$$\mathrm{trace}(\mathbf{Y}\mathbf{S}\mathbf{Y}^\top) + \eta\, \mathrm{trace}(\mathbf{Y}\mathbf{Y}^\top), \quad (9)$$

subject to $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$, which is a relaxation of the bit uncorrelation condition. The estimation of $\mathbf{W}$ is done by directly dropping the sgn operator.

An unsupervised extension is given in [143]: sequentially compute the projection vector $\{\mathbf{w}_m\}_{m=1}^M$ from $\mathbf{w}_1$ to $\mathbf{w}_M$ by optimizing the problem 9. In particular, the first iteration computes the PCA direction as the first $\mathbf{w}$, and at each of the later iterations, $s^o_{ij} = 1$ if nearby points are mapped to different hash values in the previous iterations, and $s^o_{ij} = -1$ if far points are mapped to same hash values in the previous iterations. An extension of the semi-supervised hashing to nonlinear hash functions is presented in [157] using the kernel hash function. An iterative two-step optimization using graph cuts is given in [32].

Comments: It is interesting to note that $\sum_{(i,j)\in\mathcal{E}} s^o_{ij} \mathbf{y}_i^\top \mathbf{y}_j = \mathtt{const} - \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} s^o_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = \mathtt{const} - \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij}$ if $\mathbf{y} \in \{1,-1\}^M$, where const is a constant variable (and thus $\mathrm{trace}(\mathbf{Y}\mathbf{S}\mathbf{Y}^\top) = \mathtt{const} - \mathrm{trace}(\mathbf{Y}(\mathbf{D} - \mathbf{S})\mathbf{Y}^\top)$). In this case, similarity-similarity product maximization is equivalent to similarity-distance product minimization.

4.3 Distance-Distance Product Maximization

The mathematical formulation of distance-distance product maximization is $\max \sum_{(i,j)\in\mathcal{E}} d^o_{ij} d^h_{ij}$. Topology preserving

hashing [169] formulates the objective function by starting with this rule:

$$\sum\nolimits_{i,j} d^o_{ij} d^h_{ij} = \sum\nolimits_{i,j} d^o_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = \mathrm{trace}(\mathbf{Y}\mathbf{L}_d\mathbf{Y}^\top), \tag{10}$$

where $\mathbf{L}_d = \mathrm{Diag}\{\mathbf{D}^o\mathbf{1}\} - \mathbf{D}^o$ and $\mathbf{D}^o = [d^o_{ij}]_{N\times N}$.

In addition, similarity-distance product minimization is also considered:

$$\sum\nolimits_{(i,j)\in\mathcal{E}} s_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = \mathrm{trace}(\mathbf{Y}\mathbf{L}\mathbf{Y}^\top). \tag{11}$$

The overall formulation is given as follows,

$$\max \frac{\mathrm{trace}(\mathbf{Y}(\mathbf{L}_d + \alpha\mathbf{I})\mathbf{Y}^\top)}{\mathrm{trace}(\mathbf{Y}\mathbf{L}\mathbf{Y}^\top)}, \tag{12}$$

where $\alpha\mathbf{I}$ is introduced as a regularization term, $\mathrm{trace}(\mathbf{Y}\mathbf{Y}^\top)$, maximizing the variances, which is the same for semi-supervised hashing [141] for bit balance. The problem is optimized by dropping the sgn operator in the hash function $\mathbf{y} = \mathrm{sgn}(\mathbf{W}^\top\mathbf{x})$ and letting $\mathbf{W}^\top\mathbf{X}\mathbf{L}\mathbf{X}^\top\mathbf{W}$ be an identity matrix.

4.4 Distance-Distance Difference Minimization

Binary reconstructive embedding [66] belongs to this group: $\min \sum_{(i,j)\in\mathcal{E}} (d^o_{ij} - d^h_{ij})^2$. The Euclidean distance is used in both the input and coding spaces. The objective function is formulated as follows,

$$\min \sum\nolimits_{(i,j)\in\mathcal{E}} (\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 - \frac{1}{M}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2)^2. \tag{13}$$

The kernel hash function is used:

$$y_{nm} = h_m(\mathbf{x}) = \mathrm{sgn}(\sum\nolimits_{t=1}^{T_m} w_{mt} K(\mathbf{s}_{mt}, \mathbf{x})), \tag{14}$$

where $\{\mathbf{s}_{mt}\}_{t=1}^{T_m}$ are sampled data items, $K(\cdot,\cdot)$ is a kernel function, and $\{w_{mt}\}$ are the weights to be learnt.

Instead of relaxing or dropping the sgn function, a coordinate descent optimization scheme is presented in [66]: fix all but one weight w_{mt} and optimize the problem 13 with respect to w_{mt}. There is an exact, optimal update to this weight w_{mt} (fixing all the other weights), which is achieved with the time complexity $O(N\log N + |\mathcal{E}|)$. Alternatively, a two-step solution is presented in [106]: first relax the binary variables to $(0, 1)$ and optimize the problem via an augmented Lagrangian formulation and a nonnegative matrix factorization formulation.

Comments: We have the following equation,

$$\min \sum\nolimits_{(i,j)\in\mathcal{E}} (d^o_{ij} - d^h_{ij})^2 \tag{15}$$

$$= \min \sum\nolimits_{(i,j)\in\mathcal{E}} ((d^o_{ij})^2 + (d^h_{ij})^2 - 2d^o_{ij}d^h_{ij}) \tag{16}$$

$$= \min \sum\nolimits_{(i,j)\in\mathcal{E}} ((d^h_{ij})^2 - 2d^o_{ij}d^h_{ij}). \tag{17}$$

This shows that the difference between distance-distance difference minimization and distance-distance product maximization lies on $\min \sum_{(i,j)\in\mathcal{E}} (d^h_{ij})^2$, minimizing the distances between the data items in the hash space. This could be regarded as a regularizer, complementary to distance-distance product maximization $\max \sum_{(i,j)\in\mathcal{E}} d^o_{ij}d^h_{ij}$ which tends to maximize the distances between the data items in the hash space.

4.5 Similarity-Similarity Difference Minimization

Similarity-similarity difference minimization is mathematically formulated as $\min \sum_{(i,j)\in\mathcal{E}} (s^o_{ij} - s^h_{ij})^2$. *Supervised hashing with kernels* [85], one representative approach in this group, aims to minimize an objective function,

$$\min \sum\nolimits_{(i,j)\in\mathcal{E}} (s^o_{ij} - \frac{1}{M}\mathbf{y}_i^\top\mathbf{y}_j)^2, \tag{18}$$

where $s^o_{ij} = 1$ if (i, j) is similar, and $s^o_{ij} = -1$ if it is dissimilar. $\mathbf{y} = \mathbf{h}(\mathbf{x})$ is a kernel hash function. Kernel reconstructive hashing [162] extends this technique using a normalized Gaussian kernel similarity. Scalable graph hashing [56] uses the feature transformation to approximate the similarity matrix (graph) without explicitly computing the similarity matrix. Binary hashing [25] solves the problem using a two-step approach, in which the first step adopts semi-definite relaxation and augmented lagrangian to estimate the discrete labels.

Comments: We have the following equation,

$$\min \sum\nolimits_{(i,j)\in\mathcal{E}} (s^o_{ij} - s^h_{ij})^2 \tag{19}$$

$$= \min \sum\nolimits_{(i,j)\in\mathcal{E}} ((s^o_{ij})^2 + (s^h_{ij})^2 - 2s^o_{ij}s^h_{ij}) \tag{20}$$

$$= \min \sum\nolimits_{(i,j)\in\mathcal{E}} ((s^h_{ij})^2 - 2s^o_{ij}s^h_{ij}). \tag{21}$$

This shows that the difference between similarity-similarity difference minimization and similarity-similarity product maximization lies in $\min \sum_{(i,j)\in\mathcal{E}} (s^h_{ij})^2$, minimizing the similarities between the data items in the hash space, intuitively letting the hash codes be as different as possible. This could be regarded as a regularizer complementary to similarity-similarity product maximization $\max \sum_{(i,j)\in\mathcal{E}} s^o_{ij}s^h_{ij}$, which has a trivial solution: the hash codes are the same for all data points.

Extensions and variants: *Multi-dimensional spectral hashing* [155] uses a similar objective function, but with a weighted Hamming distance,

$$\min \sum\nolimits_{(i,j)\in\mathcal{E}} (s^o_{ij} - \mathbf{y}_i^\top\mathbf{\Lambda}\mathbf{y}_j)^2, \tag{22}$$

where $\mathbf{\Lambda}$ is a diagonal matrix. Both $\mathbf{\Lambda}$ and hash codes $\{\mathbf{y}_i\}$ are needed to be optimized. The algorithm for solving the problem 22 to compute the hash codes is similar to that given in spectral hashing [156]. *Bilinear hyperplane hashing* [87] extends the formulation of supervised hashing with kernels by introducing a bilinear hyperplane hashing function. *Label-regularized maximum margin hashing* [102] formulates the objective function from three components: the similarity-similarity difference, a hinge loss from the hash function, and the maximum margin part.

4.6 Normalized Similarity-Similarity Divergence Minimization

Spec hashing [78], belonging to this group, views each pair of data items as a sample and their (normalized) similarity as the probability, and finds the hash functions so that the probability distributions from the input space and the coding space are well aligned. The objective function is written as follows,

$$\mathrm{KL}(\{\bar{s}^o_{ij}\}, \{\bar{s}^h_{ij}\}) = \mathtt{const} - \sum\nolimits_{(i,j)\in\mathcal{E}} \bar{s}^o_{ij} \log \bar{s}^h_{ij}. \tag{23}$$

Here, $\bar{s}^o_{ij}$ is the normalized similarity in the input space, $\sum_{ij}\bar{s}^o_{ij}=1$. $\bar{s}^h_{ij}$ is the normalized similarity in the Hamming space, $\bar{s}^h_{ij}=\frac{1}{Z}\exp\left(-\lambda d^h_{ij}\right)$, where Z is a normalization variable $Z=\sum_{ij}\exp\left(-\lambda d^h_{ij}\right)$.

Supervised binary hash code learning [27] presents a supervised learning algorithm based on the Jensen-Shannon divergence which is derived from minimizing an upper bound of the probability of Bayes decision errors.

5 Multiwise Similarity Preserving

This section reviews the category of hashing algorithms that formulate the loss function by maximizing the agreement of the similarity orders over more than two items computed from the input space and the coding space.

Order preserving hashing [150] aims to learn hash functions through aligning the orders computed from the original space and the ones in the coding space. Given a data point $\mathbf{x}_n$, the database points $\mathcal{X}$ are divided into $(M+1)$ categories, $(\mathcal{C}^h_{n0},\mathcal{C}^h_{n1},\cdots,\mathcal{C}^h_{nM})$, where $\mathcal{C}^h_{nm}$ corresponds to the items whose distance to the given point is m, and $(\mathcal{C}^o_{n0},\mathcal{C}^o_{n1},\cdots,\mathcal{C}^o_{nM})$, using the distances in the hashing space and the distances in the input (original) space, respectively. $(\mathcal{C}^o_{n0},\mathcal{C}^o_{n1},\cdots,\mathcal{C}^o_{nM})$ is constructed such that in the ideal case the probability of assigning an item to any hash code is the same. The basic objective function maximizing the alignment between the two categories is given as follows,

$$L(\mathbf{h}(\cdot);\mathcal{X})=\sum\nolimits_{n\in\{1,\cdots,N\}}\sum_{m=0}^{M}(|\mathcal{C}^o_{nm}-\mathcal{C}^h_{nm}|+|\mathcal{C}^h_{nm}-\mathcal{C}^o_{nm}|),$$

where $|\mathcal{C}^o_{nm}-\mathcal{C}^h_{nm}|$ is the cardinality of the difference of the two sets. The linear hash function $\mathbf{h}(\mathbf{x})$ is used and dropping the sgn function is adopted for optimization.

Instead of preserving the order, KNN hashing [23] directly maximizes the kNN accuracy of the search result, which is solved by using the factorized neighborhood representation to parsimoniously model the neighborhood relationships inherent in the training data.

Triplet loss hashing [109] formulates the hashing problem by maximizing the similarity order agreement defined over triplets of items, $\{(\mathbf{x},\mathbf{x}^+,\mathbf{x}^-)\}$, where the pair $(\mathbf{x},\mathbf{x}^-)$ is less similar than the pair $(\mathbf{x},\mathbf{x}^+)$. The triplet loss is defined as

$$\ell_{\text{triplet}}(\mathbf{y},\mathbf{y}^+,\mathbf{y}^-)=\max(1-\|\mathbf{y}-\mathbf{y}^-\|_1+\|\mathbf{y}-\mathbf{y}^+\|_1,0). \tag{24}$$

The objective function is given as follows,

$$\sum_{(\mathbf{x},\mathbf{x}^+,\mathbf{x}^-)\in\mathcal{D}}\ell_{\text{triplet}}(\mathbf{h}(\mathbf{x}),\mathbf{h}(\mathbf{x}^+),\mathbf{h}(\mathbf{x}^-))+\frac{\lambda}{2}\operatorname{trace}(\mathbf{W}^\top\mathbf{W}),$$

where $\mathbf{h}(\mathbf{x})=\mathbf{h}(\mathbf{x};\mathbf{W})$ is the compound hash function. The problem is optimized using the algorithm similar to minimal loss hashing [107]. The extension to asymmetric Hamming distance is also discussed in [109]. Binary optimized hashing [18] also uses a triplet loss function, with a slight different distance measure in the Hamming space and a different optimization technique.

Top rank supervised binary coding [132] presents another form of triplet losses in order to penalize the samples that are incorrectly ranked at the top of a Hamming-distance ranking list more than those at the bottom.

Listwise supervision hashing [146] also uses triplets of items. The formulation is based on a triplet tensor $\mathbf{S}^o$ defined as follows,

$$s^o_{ijk}=s(\mathbf{q}_i;\mathbf{x}_j,\mathbf{x}_k)=\begin{cases}1 & \text{if } \mathrm{s}^o(\mathbf{q}_i,\mathbf{x}_j)<\mathrm{s}^o(\mathbf{q}_i,\mathbf{x}_k)\\ -1 & \text{if } \mathrm{s}^o(\mathbf{q}_i,\mathbf{x}_j)>\mathrm{s}^o(\mathbf{q}_i,\mathbf{x}_k)\\ 0 & \text{if } \mathrm{s}^o(\mathbf{q}_i,\mathbf{x}_j)=\mathrm{s}^o(\mathbf{q}_i,\mathbf{x}_k)\end{cases}.$$

The objective is to maximize triple-similarity-triple-similarity product:

$$\sum\nolimits_{i,j,k}s^h_{ijk}s^o_{ijk}, \tag{25}$$

where s^h_{ijk} is a ranking triplet computed by the binary code using the cosine similarity, $s^h_{ijk}=\mathrm{sgn}(\mathbf{h}(\mathbf{q}_i)^\top\mathbf{h}(\mathbf{x}_j)-\mathbf{h}(\mathbf{q}_i)^\top\mathbf{h}(\mathbf{x}_k))$. Through dropping the sgn function, the objective function is transformed to

$$-\sum\nolimits_{i,j,k}\mathbf{h}(\mathbf{q}_i)^\top(\mathbf{h}(\mathbf{x}_j)-\mathbf{h}(\mathbf{x}_k))s^o_{ijk}, \tag{26}$$

which is solved by dropping the sgn operator in the hash function $\mathbf{h}(\mathbf{x})=\mathrm{sgn}(\mathbf{W}^\top\mathbf{x})$.

Comments: Order preserving hashing considers the relation between the search lists while triplet loss hashing and listwise supervision hashing consider triplewise relation. The central ideas of triplet loss hashing and listwise supervision hashing are very similar, and their difference lies in how to formulate the loss function besides the different optimization techniques they adopted.

6 Implicit Similarity Preserving

We review the category of hashing algorithms that focus on pursuing effective space partitioning without explicitly evaluating the relation between the distances/similarities in the input and coding spaces. The common idea is to partition the space, formulated as a classification problem, with the maximum margin criterion or the code balance condition.

Random maximum margin hashing [61] learns a hash function with the maximum margin criterion. The point is that the positive and negative labels are randomly generated by randomly sampling N data items and randomly labeling half of the items with -1 and the other half with 1. The formulation is a standard SVM formulation that is equivalent to the following form,

$$\max\frac{1}{\|\mathbf{w}\|_2}\min\{\min_{i=1,\cdots,\frac{N}{2}}(\mathbf{w}^\top\mathbf{x}^+_i+b),\min_{i=1,\cdots,\frac{N}{2}}(-\mathbf{w}^\top\mathbf{x}^-_i-b)\},$$

where $\{\mathbf{x}^+_i\}$ are the positive samples and $\{\mathbf{x}^-_i\}$ are the negative samples. Note that this is different from PICODES [7] as random maximum margin hashing adopts the hyperplanes learnt from SVM to form the hash functions while PICODES [7] exploits the hyperplanes to check whether the hash codes are semantically separable rather than forming hash functions.

Complementary projection hashing [60], similar to complementary hashing [160], finds the hash function such that the items are as far away as possible from the partition plane corresponding to the hash function. It is formulated as $\mathcal{H}(\epsilon-|\mathbf{w}^\top\mathbf{x}+b|)$, where $\mathcal{H}(\cdot)=\frac{1}{2}(1+\mathrm{sgn}(\cdot))$ is the unit step

function. Moreover, the bit balance condition, $\mathbf{Y1} = 0$, and the bit uncorrelation condition, the non-diagonal entries in $\mathbf{YY}^\top$ are 0, are considered. An extension is also given by using the kernel hash function. In addition, when learning the mth hash function, the data item is weighted by a variable, which is computed according to the previously computed $(m-1)$ hash functions.

Spherical hashing [41] uses a hypersphere to partition the space. The spherical hash function is defined as $h(\mathbf{x}) = 1$ if $d(\mathbf{p}, \mathbf{x}) \leqslant t$ and $h(\mathbf{x}) = 0$ otherwise. The compound hash function consists of M spherical functions, depending on M pivots $\{\mathbf{p}_1, \cdots, \mathbf{p}_M\}$ and M thresholds $\{t_1, \cdots, t_M\}$. The distance in the coding space is defined based on the distance: $\frac{\|\mathbf{y}_1 - \mathbf{y}_2\|_1}{\mathbf{y}_1^T \mathbf{y}_2}$. Unlike the pairwise and multiwise similarity preserving algorithms, there is no explicit function penalizing the disagreement of the similarities computed in the input and coding spaces. The M pivots and thresholds are learnt such that it satisfies a pairwise bit balance condition: $|\{\mathbf{x} \mid h_m(\mathbf{x}) = 1\}| = |\{\mathbf{x} \mid h_m(\mathbf{x}) = 0\}|$, and $|\{\mathbf{x} \mid h_i(\mathbf{x}) = b_1, h_j(\mathbf{x}) = b_2\}| = \frac{1}{4}|\mathcal{X}|, b_1, b_2 \in \{0, 1\}, i \neq j$.

7 Quantization

The following provides a simple derivation showing that the quantization approach can be derived from the distance-distance difference minimization criterion. There is a similar statement in [50] obtained from the statistical perspective: the distance reconstruction error is statistically bounded by the quantization error. Considering two points $\mathbf{x}_i$ and $\mathbf{x}_j$ and their approximations $\mathbf{z}_i$ and $\mathbf{z}_j$, we have

$$
\begin{aligned}
& |d_{ij}^o - d_{ij}^h| && (27)\\
= & ||\mathbf{x}_i - \mathbf{x}_j|_2 - |\mathbf{z}_i - \mathbf{z}_j|_2| && (28)\\
= & ||\mathbf{x}_i - \mathbf{x}_j|_2 - |\mathbf{x}_i - \mathbf{z}_j|_2 + |\mathbf{x}_i - \mathbf{z}_j|_2 - |\mathbf{z}_i - \mathbf{z}_j|_2| && (29)\\
\leqslant & ||\mathbf{x}_i - \mathbf{x}_j|_2 - |\mathbf{x}_i - \mathbf{z}_j|_2| + ||\mathbf{x}_i - \mathbf{z}_j|_2 - |\mathbf{z}_i - \mathbf{z}_j|_2| && (30)\\
\leqslant & |\mathbf{x}_j - \mathbf{z}_j|_2 + |\mathbf{x}_i - \mathbf{z}_i|_2. && (31)
\end{aligned}
$$

Thus, $|d_{ij}^o - d_{ij}^h|^2 \leqslant 2(|\mathbf{x}_j - \mathbf{z}_j|_2^2 + |\mathbf{x}_i - \mathbf{z}_i|_2^2)$, and

$$
\begin{aligned}
& \min \sum\nolimits_{i,j \in \{1,2,\cdots,N\}} |d_{ij}^o - d_{ij}^h|^2 && (32)\\
\leqslant & \min 2 \sum\nolimits_{i,j \in \{1,2,\cdots,N\}} (|\mathbf{x}_j - \mathbf{z}_j|_2^2 + |\mathbf{x}_i - \mathbf{z}_i|_2^2) && (33)\\
= & \min 4 \sum\nolimits_{i \in \{1,2,\cdots,N\}} |\mathbf{x}_i - \mathbf{z}_i|_2^2. && (34)
\end{aligned}
$$

This means that the distance-distance difference minimization rule is transformed to minimizing its upper-bound, the quantization error, which is described as a theorem below.

Theorem 1. The distortion error in the quantization approach is an upper bound (with a scale) of the differences between the pairwise distances computed from the input features and from the approximate representation.

The quantization approach for hashing is roughly divided into two main groups: hypercubic quantization, in which the approximation $\mathbf{z}$ is equal to the hash code $\mathbf{y}$, and Cartesian quantization, in which the approximation $\mathbf{z}$ corresponds to a vector formed by the hash code $\mathbf{y}$, e.g., $\mathbf{y}$ represents the index of a set of candidate approximations. In addition, we will review the related reconstruction-based hashing algorithms.

7.1 Hypercubic Quantization

Hypercubic quantization refers to a category of algorithms that quantize a data item to a vertex in a hypercubic, i.e., a vector belonging to a set $\{[y_1\ y_2\ \cdots\ y_M]^\top \mid y_m \in \{-1, 1\}\}$ or the rotated hypercubic vertices. It is in some sense related to 1-bit compressive sensing [8]: Its goal is to design a measurement matrix $\mathbf{A}$ and a recovery algorithm such that a k-sparse unit vector $\mathbf{x}$ can be efficiently recovered from the sign of its linear measurements, i.e., $\mathbf{b} = \mathrm{sgn}(\mathbf{Ax})$, while hypercubic quantization aims to find the matrix $\mathbf{A}$ which is usually a rotation matrix, and the codes $\mathbf{b}$, from the input $\mathbf{x}$.

The widely-used scalar quantization approach with only one bit assigned to each dimension can be viewed as a hypercubic quantization approach, and can be derived by minimizing

$$||\mathbf{x}_i - \mathbf{y}_i||_2^2 \tag{35}$$

subject to $\mathbf{y}_i \in \{1, -1\}$. The local digit coding approach [64] also belongs to this category.

7.1.1 Iterative quantization

Iterative quantization [35], [36] preprocesses the data, by reducing the dimension using PCA to M dimensions, $\mathbf{v} = \mathbf{P}^\top \mathbf{x}$, where $\mathbf{P}$ is a matrix of size $d \times M$ ($M \leqslant d$) computed using PCA, and then finds an optimal rotation $\mathbf{R}$ followed by a scalar quantization. The formulation is given as,

$$\min \|\mathbf{Y} - \mathbf{R}^\top \mathbf{V}\|_F^2, \tag{36}$$

where $\mathbf{R}$ is a matrix of $M \times M$, $\mathbf{V} = [\mathbf{v}_1 \mathbf{v}_2 \cdots \mathbf{v}_N]$ and $\mathbf{Y} = [\mathbf{y}_1 \mathbf{y}_2 \cdots \mathbf{y}_N]$.

The problem is solved via alternative optimization. There are two alternative steps. Fixing $\mathbf{R}$, $\mathbf{Y} = \mathrm{sgn}(\mathbf{R}^\top \mathbf{V})$. Fixing $\mathbf{B}$, the problem becomes the classic orthogonal Procrustes problem, and the solution is $\mathbf{R} = \hat{\mathbf{S}}\mathbf{S}^\top$, where $\mathbf{S}$ and $\hat{\mathbf{S}}$ is obtained from the SVD of $\mathbf{YV}^\top$, $\mathbf{YV}^\top = \mathbf{S}\mathbf{\Lambda}\hat{\mathbf{S}}^\top$.

Comments: We present an integrated objective function that is able to explain the necessity of PCA dimension reduction. Let $\bar{\mathbf{y}}$ be a d-dimensional vector, which is a concatenated vector from $\mathbf{y}$ and an all-zero subvector: $\bar{\mathbf{y}} = [\mathbf{y}^\top 0 \ldots 0]^\top$. The integrated objective function is written as follows:

$$\min \|\bar{\mathbf{Y}} - \bar{\mathbf{R}}^\top \mathbf{X}\|_F^2, \tag{37}$$

where $\bar{\mathbf{Y}} = [\bar{\mathbf{y}}_1 \bar{\mathbf{y}}_2 \cdots \bar{\mathbf{y}}_N]$, $\mathbf{X} = [\mathbf{x}_1 \mathbf{x}_2 \cdots \mathbf{x}_N]$, and $\bar{\mathbf{R}}$ is a rotation matrix of $d \times d$. Let $\bar{\mathbf{P}}$ be the projection matrix of $d \times d$, computed using PCA, $\bar{\mathbf{P}} = [\mathbf{P}\mathbf{P}_\perp]$, and $\mathbf{P}_\perp$ is a matrix of $d \times (d - M)$. It can be derived that, the solutions for $\mathbf{y}$ of the two problems in 37 and 36 are the same, and $\bar{\mathbf{R}} = \bar{\mathbf{P}}\,\mathrm{diag}(\mathbf{R}, \mathbf{I}_{(d-M)\times(d-M)})$.

7.1.2 Extensions and Variants

Harmonious hashing [159] modifies iterative quantization by adding an extra constraint: $\mathbf{YY}^\top = \sigma\mathbf{I}$. The problem is solved by relaxing $\mathbf{Y}$ to continuous values: fixing $\mathbf{R}$, let $\mathbf{R}^\top \mathbf{V} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top$, then $\mathbf{Y} = \sigma^{1/2}\mathbf{U}\mathbf{V}^\top$; fixing $\mathbf{Y}$, $\mathbf{R} = \hat{\mathbf{S}}\mathbf{S}^\top$, where $\mathbf{S}$ and $\hat{\mathbf{S}}$ is obtained from the SVD of $\mathbf{YV}^\top$, $\mathbf{YV}^\top = \mathbf{S}\mathbf{\Lambda}\hat{\mathbf{S}}^\top$. The hash function is finally computed as $\mathbf{y} = \mathrm{sgn}(\mathbf{R}^\top \mathbf{v})$.

Isotropic hashing [63] finds a rotation following PCA preprocessing such that $\mathbf{R}^\top\mathbf{V}\mathbf{V}^\top\mathbf{R} = \mathbf{\Sigma}$ becomes a matrix with equal diagonal values, i.e., $[\mathbf{\Sigma}]_{11} = [\mathbf{\Sigma}]_{22} = \cdots = [\mathbf{\Sigma}]_{MM}$. The objective function is written as $\|\mathbf{R}^\top\mathbf{V}\mathbf{V}^\top\mathbf{R} - \mathbf{Z}\|_F = 0$, where $\mathbf{Z}$ is a matrix with all the diagonal entries equal to an unknown variable σ. The problem can be solved by two algorithms: lift and projection, and gradient flow.

Comments: The goal of making the variances along the M directions being the same is to make the bits in the hash codes equally contributing to the distance evaluation. In the case that the data items satisfy the isotropic Gaussian distribution, the solution from isotropic hashing is equivalent to iterative quantization.

Similar to iterative quantization, the PCA preprocessing in isotropic hashing is also interpretable: finding a global rotation matrix $\bar{\mathbf{R}}$ such that the first M diagonal entries of $\bar{\mathbf{\Sigma}} = \bar{\mathbf{R}}^\top\mathbf{X}\mathbf{X}^\top\bar{\mathbf{R}}$ are equal, and their sum is as large as possible, which is formally written as follows,

$$\max \quad \sum_{m=1}^{M}[\bar{\mathbf{\Sigma}}]_{mm} \tag{38}$$

$$\text{s.t.} \quad [\bar{\mathbf{\Sigma}}]_{mm} = \sigma, m = 1, \cdots, M, \ \bar{\mathbf{R}}^\top\bar{\mathbf{R}} = \mathbf{I}. \tag{39}$$

Other extensions include cosine similarity preserving quantization (*Angular quantization* [34]), nonlinear embedding replacing PCA embedding [46] [175], matrix hashing [33], and so on. Quantization is also applied to supervised problems: Supervised discrete hashing [125], [127], [168], [170], present an SVM-like formulation to minimize the quantization loss and the classification loss in the hash coding space, and jointly optimize the hash function parameters and the SVM weights. Intuitively, the goal of these methods is that the hash codes are semantically separable, which is guaranteed through maximizing the classification performance.

7.2 Cartesian Quantization

Cartesian quantization refers to a class of quantization algorithms in which the composed dictionary $\mathcal{C}$ is formed from a Cartesian product of a set of small source dictionaries $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_P\}$: $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_P = \{(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})\}$, where $\mathcal{C}_p = \{\mathbf{c}_{p0}, \mathbf{c}_{p2}, \cdots, \mathbf{c}_{p(K_p-1)}\}$, $i_p \in \{0, 1, \cdots, K_p - 1\}$.

The benefits include that (1) P small dictionaries, with totally $\sum_{p=1}^{P} K_p$ dictionary items, generate a larger dictionary with $\prod_{p=1}^{P} K_p$ dictionary items; (2) the (asymmetric) distance from a query $\mathbf{q}$ to the composed dictionary item $(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})$ (an approximation of a data item) is computed from the distances $\{\text{dist}(\mathbf{q}, \mathbf{c}_{1i_1}), \cdots, \text{dist}(\mathbf{q}, \mathbf{c}_{Pi_P})\}$ through a sum operation, thus the cost of the distance computation between a query and a data item is $O(P)$, if the distances between the query and the source dictionary items are precomputed; and (3) the query cost with a set of N database items is reduced from Nd to NP through looking up a distance table which is efficiently computed between the query and the P source dictionaries.

7.2.1 Product Quantization

Product quantization [50], which initiates the quantization-based compact coding solution to similarity search, forms the P source dictionaries by dividing the feature space into P disjoint subspaces, accordingly dividing the database into P sets, each set consisting of N subvectors $\{\mathbf{x}_{p1}, \cdots, \mathbf{x}_{pN}\}$, and then quantizing each subspace separately into (usually $K_1 = K_2 = \cdots = K_P = K$) clusters. Let $\{\mathbf{c}_{p1}, \mathbf{c}_{p2}, \cdots, \mathbf{c}_{pK}\}$ be the cluster centers of the pth subspace. The operation forming an item in the dictionary from a P-tuple $(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})$ is the concatenation $[\mathbf{c}_{1i_1}^\top \mathbf{c}_{2i_2}^\top \cdots \mathbf{c}_{Pi_P}^\top]^\top$. A data point assigned to the nearest dictionary item $(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})$ is represented by a compact code $(i_1, i_2, \cdots, i_P)$, whose length is $P \log_2 K$. The distance $\text{dist}(\mathbf{q}, \mathbf{c}_{pi_p})$ between a query $\mathbf{q}$ and the dictionary element in the pth dictionary is computed as $\|\mathbf{q}_p - \mathbf{c}_{pi_p}\|_2^2$, where $\mathbf{q}_p$ is the subvector of $\mathbf{q}$ in the pth subspace.

Mathematically, product quantization can be viewed as minimizing the following objective function,

$$\min_{\mathbf{C}, \{\mathbf{b}_n\}} \quad \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2. \tag{40}$$

Here $\mathbf{C}$ is a matrix of $d \times PK$ in the form of

$$\mathbf{C} = \text{diag}(\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_P) = \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{C}_P \end{bmatrix},$$

where $\mathbf{C}_p = [\mathbf{c}_{p1}\mathbf{c}_{p2}\cdots\mathbf{c}_{pK}]$. $\mathbf{b}_n = [\mathbf{b}_{n1}^\top \mathbf{b}_{n2}^\top \cdots \mathbf{b}_{nP}^\top]^\top$ is the composition vector, and its subvector $\mathbf{b}_{np}$ of length K is an indicator vector with only one entry being 1 and all others being 0, showing which element is selected from the pth source dictionary for quantization.

Extensions: *Distance-encoded product quantization* [42] extends product quantization by encoding both the cluster index and the distance between the cluster center and the point. The cluster index is encoded in a way similar to that in product quantization. The way of encoding the distance between a point and its cluster center is as follows: the points belonging to one cluster are partitioned (quantized) according to the distances to the cluster center, the points in each partition are represented by the corresponding partition index, and accordingly the distance of each partition to the cluster center is also recorded with the partition index.

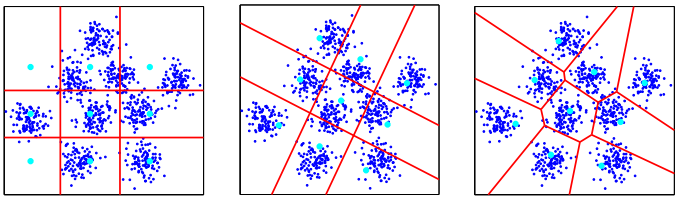
Cartesian k-means [108] and *optimized production quantization* [31] extend product quantization and introduce a rotation $\mathbf{R}$ into the objective function,

$$\min_{\mathbf{R}, \mathbf{C}, \{\mathbf{b}_n\}} \quad \sum_{n=1}^{N} \|\mathbf{R}^\top\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2. \tag{41}$$

The introduced rotation does not affect the Euclidean distance as the Euclidean distance is invariant to the rotation, and helps to find an optimized subspace partition for quantization. *Locally optimized product quantization* [62] applies optimized production quantization to the search algorithm with the inverted index, where there is a quantizer for each inverted list.

7.2.2 Composite Quantization

In composite quantization [171], the operation forming an item in the dictionary from a P-tuple $(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})$ is the summation $\sum_{p=1}^{P} \mathbf{c}_{pi_p}$. In order to compute the distance from a query $\mathbf{q}$ to the composed dictionary



(a) (b) (c)

Fig. 2. 2D toy examples illustrating the quantization algorithms. The space partitioning results are generated by (a) product quantization, (b) Cartesian k-means, and (c) composite quantization. The space partition from composition quantization is more flexible.

item formed by $(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})$ from the distances $\{\mathrm{dist}(\mathbf{q}, \mathbf{c}_{1i_1}), \cdots, \mathrm{dist}(\mathbf{q}, \mathbf{c}_{1i_1})\}$, a constraint is introduced: the summation of the inner products of all pairs of elements that are used to approximate the vector $\mathbf{x}_n$ but from different dictionaries, $\sum_{i=1}^{P}\sum_{j=1,\neq i}^{P}\mathbf{c}_{ik_{in}}\mathbf{c}_{jk_{jn}}$, is constant.

The problem is formulated as

$$\begin{aligned} \min_{\{\mathbf{C}_p\},\{\mathbf{b}_n\},\epsilon} \quad & \sum_{n=1}^{N}\|\mathbf{x}_n - [\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_P]\mathbf{b}_n\|_2^2 \\ \text{s. t.} \quad & \sum_{j=1}^{P}\sum_{i=1,i\neq j}^{P}\mathbf{b}_{ni}^\top\mathbf{C}_i^\top\mathbf{C}_j\mathbf{b}_{nj} = \epsilon, \\ & \mathbf{b}_n = [\mathbf{b}_{n1}^\top\mathbf{b}_{n2}^\top\cdots\mathbf{b}_{nP}^\top]^\top, \\ & \mathbf{b}_{np}\in\{0,1\}^K,\ \|\mathbf{b}_{np}\|_1 = 1, \\ & n = 1,2,\cdots,N; p = 1,2,\cdots P. \end{aligned} \tag{42}$$

Here, $\mathbf{C}_p$ is a matrix of size $d\times K$, and each column corresponds to an element of the pth dictionary $\mathcal{C}_p$.

Sparse composite quantization [172] improves composite quantization by constructing a sparse dictionary, $\sum_{p=1}^{P}\sum_{k=1}^{K}\|\mathbf{c}_{pk}\|_0 \leqslant S$, with S being a parameter controlling the sparsity degree, resulting in a great reduction of the distance table computation cost which takes almost the same as the most efficient approach: product quantization.

Connection with product quantization: It is shown in [171] that both product quantization and Cartesian k-means can be regarded as constrained versions of composite quantization. Composite quantization attains smaller quantization errors, yielding better search accuracy with similar search efficiency. A 2D illustration of the three algorithms is given in Figure 2, where 2D points are grouped into 9 groups. It is observed that composition quantization is more flexible in partitioning the space and thus the quantization error is possibly smaller.

Composite quantization, product quantization, and Cartesian k-means (optimized product quantization) can be explained from the view of sparse coding, as pointed in [2], [138], [171]: the dictionary ($\{\mathbf{C}_p\}$) in composite quantization (product quantization and Cartesian k-means) satisfies the constant (orthogonality) constraint, and the sparse codes ($\{\mathbf{b}_n\}$) are 0 and 1 vectors where there is only one 1 for each subvector corresponding to a source dictionary.

Comments: As discussed in product quantization [50], the idea of using the summation of several dictionary items as an approximation of a data item has already been studied in the signal processing research area, known as multi-stage vector quantization, residual quantization, or more generally structured vector quantization [38], and recently re-developed for similarity search under the Euclidean distance (additive quantization [2], [149], and tree quantization [3] modifying additive quantization by introducing a tree-structure sparsity) and inner product [26].

7.2.3 Variants

The work in [37] presents an approach to compute the source dictionaries given the M hash functions $\{h_m(\mathbf{x}) = b_m(g_m(\mathbf{x}))\}$, where $g_m()$ is a real-valued embedding function and $b_m()$ is a binarization function, for a better distance measure, quantization-like distance, instead of Hamming or weighted Hamming distance. It computes M dictionaries, each corresponding to a hash bit and computed as

$$\bar{g}_{kb} = \mathrm{E}(g_k(\mathbf{x}) \mid b_k(g_k(\mathbf{x})) = b), \tag{43}$$

where $b = 0$ and $b = 1$. The distance computation cost is $O(M)$ through looking up a distance table, which can be accelerated by dividing the hash functions into groups (e.g., each group contains 8 functions, and thus the cost is reduced to $O(\frac{M}{8})$), building a table (e.g., consisting of 256 entries) per group instead of per hash function, and forming a larger distance lookup table. In contrast, *optimized code ranking* [148] directly estimates the distance table rather than computing it from the estimated dictionary.

Composite quantization [171] points to relation between Cartesian quantization and sparse coding. This indicates the application of sparse coding to similarity search. *Compact sparse coding* [15], the extension of robust sparse coding [16], adopts sparse codes to represent the database items: the atom indices corresponding to nonzero codes, which is equivalent to letting the hash bits associated with nonzero codes be 1 and 0 for zero codes, are used to build the inverted index, and the nonzero coefficients are used to reconstruct the database items and calculate the distances between the database items and the query. *Anti-sparse coding* [52] aims to learn a hash code so that non-zero elements in the hash code are as many as possible.

7.3 Reconstruction

We review a few reconstruction-based hashing approaches. Essentially, quantization can be viewed as a reconstruction approach for a data item. *Semantic hashing* [120], [121] generates the hash codes using the deep generative model, a restricted Boltzmann machine (RBM), for reconstructing the data item. As a result, the binary codes are used for finding similar data. A variant method proposed in [13] reconstructs the input vector from the binary codes, which is effectively solved using the auxiliary coordinates algorithm. A simplified algorithm [5] finds a binary hash code that can be used to effectively reconstruct the vector through a linear transformation.

8 Other Topics

Most hashing learning algorithms assume that the similarity information in the input space, especially the semantic similarity information, and the database items have already been given. There are some approaches to learn hash functions without such assumptions: *active hashing* [176] that actively selects the labeled pairs which are most informative for hash function learning, *online hashing* [43], *smart hashing* [163],

online sketching hashing [69], and *online adaptive hashing* [12], which learn the hash functions when the similar/dissimilar pairs come sequentially.

The manifold structure in the database is exploited for hashing, which is helpful for semantic similarity search, such as *locally linear hashing* [46], *spline regression hashing* [93], and *inductive manifold hashing* [126]. Multi-table hashing, aimed at improving locality sensitive hashing, is also studied, such as *complementary hashing* [160] and its multi-view extension [91], *reciprocal hash tables* [90] and its query-adaptive extension [88], and so on.

There are some works extending the Hamming distance. In contrast to multi-dimensional spectral hashing [155] in which the weights for the weighted Hamming distance are the same for arbitrary queries, the query-dependent distance approaches learn a distance measure whose weights or parameters depend on a specific query. *Query adaptive hashing* [81], a learning-to-hash version extended from query adaptive locality sensitive hashing [48], aims to select the hash bits (thus hash functions forming the hash bits) according to the query vector. *Query-adaptive class-specific bit weighting* [57], [58] presents a weighted Hamming distance measure by learning the class-specific bit weights from the class information of the query. *Bits reconfiguration* [101] is to learn a good distance measure over the hash codes precomputed from a pool of hash functions.

The following reviews three research topics: joint feature and hash learning with deep learning, fast search in the Hamming space replacing the exhaustive search, and the important application of the Cartesian quantization to inverted index.

8.1 Joint Feature and Hash Learning via Deep Learning

The great success in deep neural network for representation learning has inspired a lot of deep compact coding algorithms [30], [67], [158], [174]. Typically, these approaches except [67] simultaneously learn the representation using a deep neural network and the hashing function under some loss functions, rather than separately learn the features and then learn the hash functions.

The methodology is similar to other learning to hash algorithms that do not adopt deep learning, and the hash function is more general and could be a deep neural network. We provide here a separate discussion because this area is relatively new. However, we will not discuss semantic hashing [120] which is usually not thought as a feature learning approach but just a hash function learning approach. In general, almost all non-deep-learning hashing algorithms if the similarity order (e.g., semantic similarity) is given, can be extended to deep learning based hashing algorithms. In the following, we discuss the deep learning based algorithms and also categorize them according to their similarity preserving manners.

- Pairwise similarity preserving. The similarity-similarity difference minimization criterion is adopted in [158]. It uses a two-step scheme: the hash codes are computed by minimizing the similarity-similarity difference without considering the visual information, and then the image representation and hash function are jointly learnt through deep learning.
- Multiwise similarity preserving. The triplet loss is used in [67], [174], which adopt the loss function defined in Equation (24) (1 is dropped in [67])
- Quantization. Following the scalar quantization approach, deep hashing [80] defines a loss to penalize the difference between the binary hash codes (see Equation (35)) and the real values from which a linear projection is used to generate the binary codes, and introduces the bit balance and bit uncorrelation conditions.

8.2 Fast Search in the Hamming Space

The computation of the Hamming distance is shown much faster than the computation of the distance in the input space. It is still expensive, however, to handle a large scale data set using linear scan. Thus, some indexing algorithms already shown effective and efficient for general vectors are borrowed for the search in the Hamming space. For example, min-hash, a kind of LSH, is exploited to search over high-dimensional binary data [129]. In the following, we discuss other representative algorithms.

Multi-index hashing [110] and its extension [133] aim to partition the binary codes into M disjoint substrings and build M hash tables each corresponding to a substring, indexing all the binary codes M times. Given a query, the method outputs the NN candidates which are near to the query at least in one hash table. *FLANN* [104] extends the FLANN algorithm [103] that was initially designed for ANN search over real-value vectors to search over binary vectors. The key idea is to build multiple hierarchical cluster trees to organize the binary vectors and to search for the nearest neighbors simultaneously over the multiple trees by traversing each tree in a best-first manner.

PQTable [97] extends multi-index hashing from the Hamming space to the product-quantization coding space, for fast exact search. Unlike multi-index hashing flipping the bits in the binary codes to find candidate tables, PQTable adopts the multi-sequence algorithm [4] for efficiently finding candidate tables. The neighborhood graph-based search algorithm [144] for real-value vectors is extended to the Hamming space [59].

8.3 Inverted Multi-Index

Hash table lookup with binary hash codes is a form of inverted index. Retrieving multiple hash buckets for multiple hash tables is computationally cheaper compared with the subsequent reranking step using the true distance computed in the input space. It is also cheap to visit more buckets in a single table if the standard Hamming distance is used, as the nearby hash codes of the hash code of the query which can be obtained by flipping the bits of the hash code of the query. If there are a lot of empty buckets which increases the retrieval cost, the double-hash scheme or the fast search algorithm in the Hamming space, e.g., [104], [110] can be used to fast retrieve the hash buckets.

Thanks to the multi-sequence algorithm, the Cartesian quantization algorithms are also applied to the inverted index [4], [172], [31] (called inverted multi-index), in which

each composed quantization center corresponds to an inverted list. Instead of comparing the query with all the composed quantization centers, which is computationally expensive, the multi-sequence algorithm [4] is able to efficiently produce a sequence of (T) inverted lists ordered by the increasing distances between the query and the composed quantization centers, whose cost is $O(T \log T)$. The study (Figure 5 in [151]) shows that the time cost of the multi-sequence algorithm when retrieving $10K$ candidates over the two datasets: SIFT1M and GIST1M is the smallest compared with other non-hashing inverted index algorithms.

Though the cost of the multi-sequence algorithm is greater than that with binary hash codes, both are relatively small and negligible compared with the subsequent reranking step that is often conducted in real applications. Thus the quantization-based inverted index (hash table) is more widely used compared with the conventional hash tables with binary hash codes.

9 Evaluation Protocols

9.1 Evaluation Metrics

There are three main concerns for an approximate nearest neighbor search algorithm: space cost, search efficiency, and search quality. The space cost for hashing algorithms depends on the code length for hash code ranking, and the code length and the table number for hash table lookup. The search performance is usually measured under the same space cost, i.e., the code length (and the table number) is chosen the same for different algorithms.

The search efficiency is measured as the time taken to return the search result for a query, which is usually computed as the average time over a number of queries. The time cost often does not include the cost of the reranking step (using the original feature representations) as it is assumed that such a cost given the same number of candidates does not depend on the hashing algorithms and can be viewed as a constant. When comparing the performance in the case the Hamming distance in hash code ranking is used in the coding space, it is not necessary to report the search time costs because they are the same. It is necessary to report the search time cost when a non-Hamming distance or the hash table lookup scheme is used.

The search quality is measured using recall@R (i.e., a recall-R curve). For each query, we retrieve its R nearest items and compute the ratio of the true nearest items in the retrieved R items to T , i.e., the fraction of T ground-truth nearest neighbors are found in the retrieved R items. The average recall score over all the queries is used as the measure. The ground-truth nearest neighbors are computed over the original features using linear scan. Note that the recall@R is equivalent to the accuracy computed after reordering the R retrieved nearest items using the original features and returning the top T items. In the case where the linear scan cost in the hash coding space is not the same (e.g., binary code hashing, and quantization-based hashing), the curve in terms of search recall and search time cost is usually reported.

The semantic similarity search, a variant of nearest neighbor search, sometimes uses the precision, the recall, the precision-recall curve, and mean average precision (mAP). The precision is computed at the retrieved position R, i.e., R items are retrieved, as the ratio of the number of retrieved true positive items to R. The recall is computed, also at position R, as the ratio of the number of retrieved true positive items to the number of all true positive items in the database. The pairs of recall and precision in the precision-recall curve are computed by varying the retrieved position R. The mAP score is computed as follows: the average precision for a query, the area under the precision-recall curve is computed as $\sum_{t=1}^{N} P(t)\Delta(t)$, where $P(t)$ is the precision at cut-off t in the ranked list and $\Delta(t)$ is the change in recall from items $t-1$ to t; the mean of average precisions over all the queries is computed as the final score.

TABLE 2
A summary of evaluation datasets

	Dim	Reference set	Learning set	Query set
MNIST	784	$60K$	-	$10K$
SIFT10K	128	$10K$	$25K$	100
SIFT1M	128	$1M$	$100K$	$10K$
GIST1M	960	$1M$	$50K$	$1K$
Tiny1M	384	$1M$	-	$100K$
SIFT1B	128	$1B$	$100M/1M$	$10K$
GloVe1.2M	200	$\approx 1.2M$	-	$10K$

9.2 Evaluation Datasets

The widely-used evaluation datasets have different scales from small, large, to very large. Various features have been used, such as SIFT features [94] extracted from Photo-tourism [131] and Caltech 101 [28], GIST features [112] from LabelMe [119] and Peekaboom [140], as well as some features used in object retrieval: Fisher vectors [116] and VLAD vectors [51]. The following presents a brief introduction to several representative datasets, which is summarized in Table 2.

MNIST [68] includes $60K$ 784-dimensional raw pixel features describing grayscale images of handwritten digits as a reference set, and $10K$ features as the queries.

SIFT10K [50] consists of $10K$ 128-dimensional SIFT vectors as the reference set, $25K$ vectors as the learning set, and 100 vectors as the query set. SIFT1M [50] is composed of $1M$ 128-dimensional SIFT vectors as the reference set, $100K$ vectors as the learning set, and $10K$ as the query set. The learning sets in SIFT10K and SIFT1M are extracted from Flicker images and the reference sets and the query sets are from the INRIA holidays images [49].

GIST1M [50] consists of $1M$ 960-dimensional GIST vectors as the reference set, $50K$ vectors as the learning set, $1K$ vectors as the query set. The learning set is extracted from the first $100K$ images from the tiny images [137]. The reference set is from the Holiday images combined with Flickr1M [49]. The query set is from the Holiday image queries. Tiny1M [152][1] consists of $1M$ 384-dimensional GIST vectors as the reference set and $100K$ vectors as the query set. The two sets are extracted from the $1100K$ tiny images.

SIFT1B [53] includes $1B$ 128-dimensional BYTE-valued SIFT vectors as the reference set, $100M$ vectors as the

1. http://research.microsoft.com/~jingdw/SimilarImageSearch/NNData/NNdatasets.html

TABLE 3
A summary of query performance comparison for approximate nearest neighbor search under Euclidean distance.

	Accuracy	Efficiency	Overall
pairwise	low	high	low
multiwise	fair	high	fair
quantization	high	fair	high

learning set and $10K$ vectors as the query set. The three sets are extracted from around $1M$ images. This dataset, and SIFT10K, SIFT1M and GIST1M are publicly available[2].

GloVe1.2M [115][3] contains $1,193,514$ 200-dimensional word feature vectors extracted from Tweets. We randomly sample $10K$ vectors as the query set and use the remaining as the training set.

9.3 Training Sets and Hyper-Parameters Selection

There are three main choices of the training set over which the hash functions are learnt for learning-to-hash algorithms. The first choice is a separate set used for learning hash functions, which is not contained in the reference set. The second choice is to sample a small subset from the reference set. The third choice is to use all the reference set to train hash functions. The query set and the reference set are then used to evaluate the learnt hash functions.

In the case where the query is transformed to a hash code, e.g., when adopting the Hamming distance for most binary hash algorithms, learning over the whole reference set might lead to over-fitting and the performance might be worse than learning with a subset of the reference set or a separate set. In the case where the raw query is used without any processing, e.g., when adopting the asymmetric distance in Cartesian quantization, learning over the whole reference set is better as it results in better approximation of the reference set.

There are some hyper-parameters in the objective functions, e.g, the objective functions in minimal loss hashing [107] and composite quantization [171]. It is unfair and not suggested to select the hyper-parameters corresponding to the best performance over the query set. It is suggested instead to select the hyper-parameters by validation, e.g., sampling a subset from the reference set as the validation set which is reasonable because the validation criterion is not the objective function value but the search performance.

10 Performance Analysis

10.1 Query Performance

We summarize empirical observations and the analysis of the nearest neighbor search performance using the compact coding approach, most of which have already been mentioned or discussed in the existing works. We discuss about both hash table lookup and hash code ranking, with more focus on hash code ranking because the major usage of the learning to hash algorithms lies in hash code ranking for retrieving top candidates from a set of candidates obtained from the inverted index or other hash table lookup algorithms. The analysis is mainly focusing on the major

2. http://corpus-texmex.irisa.fr/
3. http://nlp.stanford.edu/projects/glove/

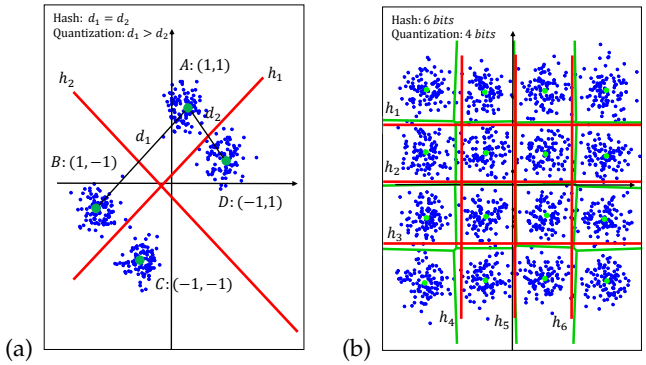


Fig. 3. 2D toy examples illustrating the comparison between binary code hashing and quantization. (a) shows the Hamming distances from clusters B and D to cluster A, usually adopted in the binary code hashing algorithms, are the same while the Euclidean distances, used in the quantization algorithms, are different. (b) the binary code hashing algorithms need 6 hash bits (red lines show the corresponding hash functions) to differentiate the 16 uniformly-distributed clusters while the quantization algorithms only require 4 ($= \log 16$) bits (green lines show the partition line).

application of hashing: nearest neighbor search with the Euclidean distance. The conclusion for semantic similarity search is similar in principle and the performance also depends on the ability of representing the semantic meaning of the input features. We also present empirical results of the quantization algorithms and the representative binary coding algorithms for hash code ranking.

10.1.1 Query Performance with Hash Table Lookup

We give a performance summary of the query scheme using hash table lookup for the two main hash algorithms: the binary hash codes and the quantization-based hash codes.

In terms of space cost, hash table lookup with binary hash codes has a little but negligible advantage over that with quantization-based hash codes because the main space cost comes from the indices of the reference items and the extra cost from the centers corresponding to the buckets using quantization is relatively small. Multi-assignment and multiple hash tables increase space cost as they require to store multiple copies of reference vector indices. As an alternative choice, single-assignment with a single table can be used but more buckets are retrieved for high recall.

When retrieving the same number of candidates, hash table lookup using binary hash codes is better in terms of the query time cost, but inferior to the quantization approach in terms of the recall, which has probably been firstly discussed in [114]. In terms of recall vs. time cost the quantization approach is overall superior as the cost from the multi-sequence algorithm is relatively small and negligible compared with the subsequent reranking step, which is observed from our experience, and can be derived from [103] and [4]. In general, the performance for other algorithms based on the weighted Hamming distance and the learnt distance is in between. The observation holds for a single table with single assignment and multiple assignment, or multiple tables.

10.1.2 Query Performance with Hash Code Ranking

The following provides a short summary of the overall performance for three main categories: pairwise similarity

preserving, multiwise similarity preserving, and quantization in terms of search cost and search accuracy under the same space cost, guaranteed by coding the items using the same number of bits, ignoring the small space cost of the dictionary in Cartesian quantization and the distance lookup tables.

Search accuracy: Multiwise similarity preserving is better than pairwise similarity preserving as it considers more information for hash function learning. There is no observation/conclusion on which algorithm, pairwise or multiwise similarity preserving algorithm, performs consistently the best. Nevertheless, there is a large amount of pairwise and multiwise similarity preserving algorithms because different algorithms may be suitable to different data distributions and optimization also affects the performance.

It has been shown in Section 7.1 that the cost function of hypercubic quantization is an approximation of the distance-distance difference. But it outperforms pairwise and multiwise similarity preserving algorithms. This is because it is infeasible to consider all pairs (triples) of items for the distance-distance difference in pairwise (multiwise) similarity preserving algorithms, and thus only a small subset of the pairs (triples), by sampling a subset of items or pairs (triples), is considered for almost all the pairwise (multiwise) similarity preserving hashing algorithms, while the cost function for quantization is an approximation for all pairs of items. This point is also discussed in [154].

Compared with binary code hashing including hypercubic quantization, another reason for the superiority of Cartesian quantization, as discussed in [171], is that there are only a small number ($L+1$) of distinct Hamming distances in the coding space for binary code hashing with the code length being L, while the number of distinct distances for Cartesian quantization is much larger. It is shown that the performance from learning a distance measure using a way like the quantization approach [37] or directly learning a distance lookup table [148][4] from precomputed hash codes is comparable to the performance of the Cartesian quantization approach if the codes from the quantization approach are given as the input.

Search cost: The evaluation of the Hamming distance using the CPU instruction __popcnt is faster than the distance-table lookup. For example, it is around twice faster for the same code length L than distance table lookup if a subtable corresponds to a byte and there are totally $\frac{L}{8}$ subtables. It is worth pointing (also observed in [171]) that the Cartesian quantization approaches relying on the distance table lookup still achieve better search accuracy even with a code of the half length, which indicates that the overall performance of the quantization approaches in terms of space cost, query time cost, and search accuracy is superior.

In summary, if the online performance in terms of space cost, query time cost, and search accuracy is cared about, the quantization algorithms are suggested for hash code ranking, hash table lookup, as well as the scheme of combining inverted index (hash table lookup) and hash code ranking. The comparison of the query performances of pairwise and multiwise similarity preserving algorithms, as well as quantization is summarized in Table 3.

Figure 3 presents 2D toy examples. Figure 3 (a) shows that the quantization algorithm is able to discriminate the non-uniformly distributed clusters with different between-cluster distances while the binary code hashing algorithm is lacking such a capability due to the Hamming distance. Figure 3 (b) shows that the binary hash coding algorithms require more (6) hash bits to differentiate the 16 uniformly-distributed clusters while the quantization algorithms only require 4 ($=\log 16$) bits.

10.1.3 Empirical Results

We present the empirical results of the several representative hashing and quantization algorithms over SIFT1M [50]. We show the results for searching the nearest neighbor ($T=1$) with 128 bits and the conclusion holds for searching more nearest neighbors ($T>1$) and with other numbers of bits. More results, such as the search time cost, and results using inverted multi-index with different quantization algorithms can be found in [172]. We also conduct experiments over word feature vectors GloVe1.2M. We present the results using recall@R for searching the nearest neighbor ($T=1$) with 128 bits.

We also report the results over deep learning features extracted from the ILSVRC 2012 dataset. The ILSVRC 2012 dataset is a subset of ImageNet [22] and contains over 1.2 million images. We use the provided training set, $1,281,167$ images, as the retrieval database and use the provided validation set, $50,000$ images, as the test set. Similar to [125], the 4096-dimensional feature extracted from the convolution neural networks (CNN) in [65] is used to represent each image. We evaluate the search performance under the Euclidean distance in terms of recall@R, where R is the number of the returned top candidates, and under the semantic similarity in terms of MAP vs. #bits.

Figure 4 shows the recall@R curves and the MAP results. We have several observations. (1) The performance of the quantization method is better than the hashing method in most cases for both Euclidean distance-based and semantic search. (2) LSH, a data-independent algorithm is generally worse than other learning to hash approaches. (3) For Euclidean distance-based search the performance of CQ is the best among quantization methods, which is consistent with the analysis and the $2D$ illustration shown in Figure 2.

10.2 Training Time Cost

We present the analysis of the training time cost for the case of using the linear hash function. The pairwise similarity preserving category considers the similarities of all pairs of items, and thus in general the training process takes quadratic time with respect to the number N of the training samples ($O(N^2M+N^2d)$). To reduce the computational cost, sampling schemes are adopted: sample a small number (e.g., $O(N)$) of pairs, whose time complexity becomes linear with respect to N, resulting in ($O(NM+Nd)$), or sample a subset of the training items (e.g., containing $\bar{N}$ items), whose time complexity becomes smaller ($O(\bar{N}^2M+\bar{N}^2d)$). The multiwise similarity preserving category considers the similarities of all triples of items, and in general the training

4. A similar idea is concurrently proposed in [117], [118] to learn a better similarity for a bag-of-words representation and quantized kernels.

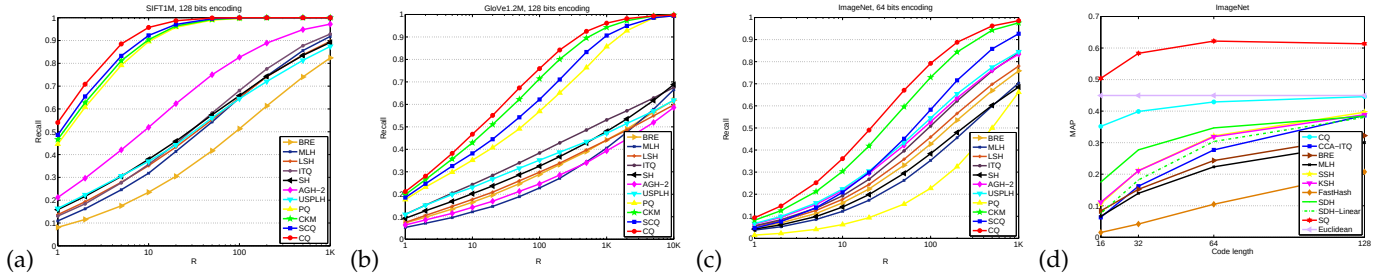


Fig. 4. (a) and (b) show the performance in terms of recall@R over SIFT$1M$ and GloVe$1.2M$ for the representative quantization algorithms. (c) and (d) show the performance over the ILSVRC 2012 ImageNet dataset under the Euclidean distance in terms of recall@R and under the semantic similarity in terms of mAP vs. # bits. BRE = binary reconstructive embedding [66], MLH = minimal loss hashing [107], LSH = locality sensitive hashing [14], ITQ = iterative quantization [35], [36], SH = spectral hashing [156], AGH-2 = two-layer hashing with graphs [86], USPLH = unsupervised sequential projection learning hashing [143], PQ = product quantization [50], CKM = Cartesian k-means [108], CQ = composite quantization [171], SCQ = sparse composite quantization [172] whose dictionary is the same sparse with PQ. CCA-ITQ = iterative quantization with canonical correlation analysis [36], SSH = semi-supervised hashing [143], KSH = supervised hashing with kernels [85], FastHash = fash supervised hashing [76], SDH = supervised discrete hashing with kernels [125], SDH-linear = supervised discrete hashing without using kernel representations [125], SQ = supervised quantization [154], Euclidean = linear scan with the Euclidean distance.

cost is greater and the sampling scheme is also used for acceleration. The analysis for kernel hash functions and other complex functions is similar, and the time complexity for both training hash functions and encoding database items is higher.

Iterative quantization consists of a PCA preprocessing step whose time complexity is $O(Nd^2)$, and the hash code and hash function optimization step, whose time complexity is $O(NM^2 + M^3)$ (M is the number of hash bits). The whole complexity is $O(Nd^2 + NM^2 + M^3)$. Product quantization includes the k-means process for each partition, and the complexity is $TNKP$, where K is usually 256, $P = \frac{M}{8}$, and T is the number of iterations for the k-means algorithm. The complexity of Cartesian k-means is $O(Nd^2 + d^3)$. The time complexity of composite quantization is $O(NKPd + NP^2 + P^2K^2d)$. In summary, the time complexity of iterative quantization is the lowest and that of composite quantization is the highest. This indicates that it takes larger offline computation cost to get a higher (online) search performance.

11 Emerging Topics

The main goal of the hashing algorithm is to accelerate the online search as the distance can be efficiently computed through fast Hamming distance computation or fast distance table lookup. The offline hash function learning and hash code computation are shown to be still expensive, and have become attractive in research. The computation cost of the distance table used for looking up is thought ignorable and in reality could be higher when handling high-dimensional databases. There is also increasing interest in topics such as multi-modality and cross-modality hashing [45] and semantic quantization.

11.1 Speed up the Learning and Query Processes

Scalable Hash Function Learning. The algorithms depending on the pairwise similarity, such as binary reconstructive embedding, usually sample a small subset of pairs to reduce the cost of learning hash functions. It has been shown that the search accuracy is increased with a high sampling rate, but the training cost is greatly increased. The algorithms even without relying on the pairwise similarity, e.g., quantization, were also shown to be slow and even infeasible when handling very large data, e.g., $1B$ data items, and usually have to learn hash functions over a small subset, e.g., $1M$ data items. This poses a challenging request to learn the hash function over larger datasets.

Hash Code Computation Speedup. Existing hashing algorithms rarely take into consideration the cost of encoding a data item. Such a cost during the query stage becomes significant in the case that only a small number of database items or a small database are compared to the query. The search combined with the inverted index and compact codes is such a case. When kernel hash functions are used, encoding the database items to binary codes is also much more expensive than that with linear hash functions. The composite quantization-like approach also takes much time to compute the hash codes.

A recent work, circulant binary embedding [165], accelerates the encoding process for the linear hash functions, and tree-quantization [3] sparsifies the dictionary items into a tree structure, to speeding up the assignment process. However, more research is needed to speed up the hash code computation for other hashing algorithms, such as composite quantization.

Distance Table Computation Speedup. Product quantization and its variants need to precompute the distance table between the query and the elements of the dictionaries. Most existing algorithms claim that the cost of distance table computation is negligible. However in practice, the cost becomes bigger when using the codes computed from quantization to rank the candidates retrieved from the inverted index. This is a research direction that will attract research interest in the near future, such as a recent study, sparse composite quantization [172].

11.2 Promising Extensions

Semantic Quantization. Existing quantization algorithms focus on the search under the Euclidean distances. Like binary code hashing algorithms where many studies on semantic similarity have been conducted, learning quantization-based hash codes with semantic similarity is attracting interest. There are already a few studies. For example, we

have proposed an supervised quantization approach [154] and some comparisons are provided in Figure 4.

Multiple and Cross Modality Hashing. One important characteristic of big data is the variety of data types and data sources. This is particularly true for multimedia data, where various media types (e.g., video, image, audio and hypertext) can be described by many different low- and high-level features, and relevant multimedia objects may come from different data sources contributed by different users and organizations. This raises a research direction, performing joint-modality hashing learning by exploiting the relation among multiple modalities, for supporting some special applications, such as cross-modal search. This topic is attracting a lot of research efforts nowadays, such as collaborative hashing [89], [167], collaborative quantization [173], and cross-media hashing [134], [135], [178], [161], [83].

12 Conclusion

In this paper, we categorize the learning-to-hash algorithms into four main groups: pairwise similarity preserving, multiwise similarity preserving, implicit similarity preserving, and quantization, present a comprehensive survey with a discussion about their relations. We point out the empirical observation that quantization is superior in terms of search accuracy, search efficiency and space cost. In addition, we introduce a few emerging topics and the promising extensions.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006. 2

[2] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *CVPR*, pages 931–939, 2014. 6, 12

[3] A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *CVPR*, 2015. 12, 17

[4] A. Babenko and V. S. Lempitsky. The inverted multi-index. In *CVPR*, pages 3069–3076, 2012. 3, 13, 14, 15

[5] R. Balu, T. Furon, and H. Jegou. Beyond "project and sign" for cosine estimation with binary codes. In *ICASSP*, pages 6884–6888, 2014. 12

[6] J. L. Bentley, D. F. Stanat, and E. H. W. Jr. The complexity of finding fixed-radius near neighbors. *Inf. Process. Lett.*, 6(6):209–212, 1977. 2

[7] A. Bergamo, L. Torresani, and A. W. Fitzgibbon. Picodes: Learning a compact code for novel-category recognition. In *NIPS.*, pages 2088–2096, 2011. 6, 9

[8] P. Boufounos and R. G. Baraniuk. 1-bit compressive sensing. In *CISS*, pages 16–21, 2008. 10

[9] J. Brandt. Transform coding for fast approximate nearest neighbor search in high dimensions. In *CVPR*, pages 1815–1822, 2010. 7

[10] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–29, Washington, DC, USA, 1997. IEEE Computer Society. 1

[11] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997. 1

[12] F. Çakir and S. Sclaroff. Adaptive hashing for fast similarity search. In *ICCV*, pages 1044–1052, 2015. 13

[13] M. Á. Carreira-Perpiñán and R. Raziperchikolaei. Hashing with binary autoencoders. In *CVPR*, pages 557–566, 2015. 12

[14] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. 1, 2, 17

[15] A. Cherian. Nearest neighbors using compact sparse codes. In *ICML (2)*, pages 1053–1061, 2014. 12

[16] A. Cherian, V. Morellas, and N. Papanikolopoulos. Robust sparse hashing. In *ICIP*, pages 2417–2420, 2012. 12

[17] O. Chum and J. Matas. Large-scale discovery of spatially related images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(2):371–377, 2010. 1

[18] Q. Dai, J. Li, J. Wang, and Y. Jiang. Binary optimized hashing. In *ACM Multimedia*, pages 1247–1256, 2016. 7, 9

[19] A. Dasgupta, R. Kumar, and T. Sarlós. Fast locality-sensitive hashing. In *KDD*, pages 1073–1081, 2011. 1

[20] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004. 1

[21] T. L. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100, 000 object classes on a single machine. In *CVPR*, pages 1814–1821, 2013. 1

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 16

[23] K. Ding, C. Huo, B. Fan, and C. Pan. knn hashing with factorized neighborhood representation. In *ICCV*, pages 1098–1106, 2015. 9

[24] T. Do, A. Doan, and N. Cheung. Learning to hash with binary deep neural network. In *ECCV*, pages 219–234, 2016. 6

[25] T. Do, A. Doan, D. T. Nguyen, and N. Cheung. Binary hashing with semidefinite relaxation and augmented lagrangian. In *ECCV*, pages 802–817, 2016. 6, 8

[26] C. Du and J. Wang. Inner product similarity search using compositional codes. *CoRR*, abs/1406.4966, 2014. 12

[27] L. Fan. Supervised binary hash code learning with jensen shannon divergence. In *ICCV*, pages 2616–2623, 2013. 9

[28] 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. 14

[29] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD Conference*, pages 541–552, 2012. 1

[30] L. Gao, J. Song, F. Zou, D. Zhang, and J. Shao. Scalable multimedia retrieval by deep learning hashing with relative similarity learning. In *ACM Multimedia*, pages 903–906, 2015. 13

[31] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2946–2953, 2013. 6, 11, 13

[32] T. Ge, K. He, and J. Sun. Graph cuts for supervised binary coding. In *ECCV*, pages 250–264, 2014. 4, 7

[33] Y. Gong, S. Kumar, H. A. Rowley, and S. Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *CVPR*, pages 484–491, 2013. 6, 11

[34] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik. Angular quantization-based binary codes for fast similarity search. In *NIPS*, pages 1205–1213, 2012. 6, 11

[35] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011. 6, 10, 17

[36] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(12):2916–2929, 2013. 6, 10, 17

[37] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik. Asymmetric distances for binary embeddings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(1):33–47, 2014. 4, 12, 16

[38] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998. 12

[39] J. He, S.-F. Chang, R. Radhakrishnan, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, pages 753–760, 2011. 6, 7

[40] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010. 3, 6, 7

[41] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *CVPR*, pages 2957–2964, 2012. 6, 10

[42] J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization. In *CVPR*, pages 2139–2146, 2014. 11

[43] L.-K. Huang, Q. Yang, and W.-S. Zheng. Online hashing. In *IJCAI*, 2013. 12

[44] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998. 1, 2

[45] G. Irie, H. Arai, and Y. Taniguchi. Alternating co-quantization for cross-modal hashing. In *ICCV*, pages 1886–1894, 2015. 17

[46] G. Irie, Z. Li, X.-M. Wu, and S.-F. Chang. Locally linear hashing for extracting non-linear manifolds. In *CVPR*, pages 2123–2130, 2014. 11, 13

[47] H. Jain, P. Pérez, R. Gribonval, J. Zepeda, and H. Jégou. Approx-

imate search with quantized sparse representations. In *ECCV*, pages 681–696, 2016. 6
[48] H. Jegou, L. Amsaleg, C. Schmid, and P. Gros. Query adaptative locality sensitive hashing. In *ICASSP*, pages 825–828, 2008. 13
[49] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, pages 304–317, 2008. 14
[50] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011. 3, 6, 10, 11, 12, 14, 16, 17
[51] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311, 2010. 1, 14
[52] H. Jégou, T. Furon, and J.-J. Fuchs. Anti-sparse coding for approximate nearest neighbor search. In *ICASSP*, pages 2029–2032, 2012. 12
[53] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *ICASSP*, pages 861–864, 2011. 14
[54] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-max hash for jaccard similarity. In *ICDM*, pages 301–309, 2013. 1
[55] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *NIPS*, pages 108–116, 2012. 1
[56] Q. Jiang and W. Li. Scalable graph hashing with feature transformation. In *IJCAI*, pages 2248–2254, 2015. 6, 8
[57] Y.-G. Jiang, J. Wang, and S.-F. Chang. Lost in binarization: query-adaptive ranking for similar image search with compact codes. In *ICMR*, page 16, 2011. 13
[58] 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
[59] Z. Jiang, L. Xie, X. Deng, W. Xu, and J. Wang. Fast nearest neighbor search in the hamming space. In *MMM*, pages 325–336, 2016. 13
[60] Z. Jin, Y. Hu, Y. Lin, D. Zhang, S. Lin, D. Cai, and X. Li. Complementary projection hashing. In *ICCV*, pages 257–264, 2013. 6, 9
[61] A. Joly and O. Buisson. Random maximum margin hashing. In *CVPR*, pages 873–880, 2011. 6, 9
[62] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2329–2336, 2014. 11
[63] W. Kong and W.-J. Li. Isotropic hashing. In *NIPS*, pages 1655–1663, 2012. 6, 11
[64] N. Koudas, B. C. Ooi, H. T. Shen, and A. K. H. Tung. Ldc: Enabling search by partial distance in a hyper-dimensional space. In *ICDE*, pages 6–17, 2004. 10
[65] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 16
[66] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009. 3, 4, 6, 8, 17
[67] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015. 6, 13
[68] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001. 14
[69] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu. Online sketching hashing. In *CVPR*, pages 2503–2511, 2015. 13
[70] P. Li, K. W. Church, and T. Hastie. Conditional random sampling: A sketch-based sampling technique for sparse data. In *NIPS*, pages 873–880, 2006. 1
[71] P. Li, T. Hastie, and K. W. Church. Very sparse random projections. In *KDD*, pages 287–296, 2006. 1
[72] P. Li and A. C. König. b-bit minwise hashing. In *WWW*, pages 671–680, 2010. 1
[73] P. Li, A. C. König, and W. Gui. b-bit minwise hashing for estimating three-way similarities. In *NIPS*, pages 1387–1395, 2010. 1
[74] P. Li, A. B. Owen, and C.-H. Zhang. One permutation hashing. In *NIPS*, pages 3122–3130, 2012. 1
[75] P. Li, M. Wang, J. Cheng, C. Xu, and H. Lu. Spectral hashing with semantically consistent graph for image indexing. *IEEE Transactions on Multimedia*, 15(1):141–152, 2013. 7
[76] G. Lin, C. Shen, Q. Shi, A. van_den_Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, pages 1971–1978, 2014. 4, 17
[77] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *ICCV*, pages 2552–2559, 2013. 4
[78] R.-S. Lin, D. A. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010. 6, 8
[79] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li. Compressed hashing. In *CVPR*, pages 446–451, 2013. 6, 7
[80] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *CVPR*, pages 2475–2483, 2015. 6, 13
[81] D. Liu, S. Yan, R.-R. Ji, X.-S. Hua, and H.-J. Zhang. Image retrieval with query-adaptive hashing. *TOMCCAP*, 9(1):2, 2013. 13
[82] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *CVPR*, pages 2064–2072, 2016. 6
[83] L. Liu, Z. Lin, L. Shao, F. Shen, G. Ding, and J. Han. Sequential discrete hashing for scalable cross-modality similarity retrieval. *IEEE Trans. Image Processing*, 26(1):107–118, 2017. 18
[84] W. Liu, C. Mu, S. Kumar, and S. Chang. Discrete graph hashing. In *NIPS*, pages 3419–3427, 2014. 6, 7
[85] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012. 6, 8, 17
[86] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011. 6, 7, 17
[87] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. In *ICML*, 2012. 6, 8
[88] X. Liu, C. Deng, B. Lang, D. Tao, and X. Li. Query-adaptive reciprocal hash tables for nearest neighbor search. *IEEE Transactions on Image Processing*, 25(2):907–919, 2016. 13
[89] X. Liu, J. He, C. Deng, and B. Lang. Collaborative hashing. In *CVPR*, pages 2147–2154, 2014. 18
[90] X. Liu, J. He, and B. Lang. Reciprocal hash tables for nearest neighbor search. In *AAAI*, 2013. 13
[91] X. Liu, L. Huang, C. Deng, J. Lu, and B. Lang. Multi-view complementary hash tables for nearest neighbor search. In *ICCV*, pages 1107–1115, 2015. 13
[92] Y. Liu, J. Shao, J. Xiao, F. Wu, and Y. Zhuang. Hypergraph spectral hashing for image retrieval with heterogeneous social contexts. *Neurocomputing*, 119:49–58, 2013. 7
[93] Y. Liu, F. Wu, Y. Yang, Y. Zhuang, and A. G. Hauptmann. Spline regression hashing for fast image search. *IEEE Transactions on Image Processing*, 21(10):4480–4491, 2012. 13
[94] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 14
[95] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007. 1
[96] J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *ECCV*, pages 137–153, 2016. 6
[97] Y. Matsui, T. Yamasaki, and K. Aizawa. Pqtable: Fast exact asymmetric distance neighbor search for product quantization using hash tables. In *ICCV*, pages 1940–1948, 2015. 13
[98] Y. Matsushita and T. Wada. Principal component hashing: An accelerated approximate nearest neighbor search. In *PSIVT*, pages 374–385, 2009. 7
[99] Y. Moon, S. Noh, D. Park, C. Luo, A. Shrivastava, S. Hong, and K. Palem. Capsule: A camera-based positioning system using learning. In *SOCC*, 2015. 1
[100] R. Motwani, A. Naor, and R. Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007. 1
[101] Y. Mu, X. Chen, X. Liu, T.-S. Chua, and S. Yan. Multimedia semantics-aware query-adaptive hashing with bits reconfigurability. *IJMIR*, 1(1):59–70, 2012. 13
[102] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344–3351, 2010. 6, 8
[103] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP (1)*, pages 331–340, 2009. 2, 13, 15
[104] M. Muja and D. G. Lowe. Fast matching of binary features. In *CRV*, pages 404–410, 2012. 3, 13
[105] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240, 2014. 2
[106] L. Mukherjee, S. N. Ravi, V. K. Ithapu, T. Holmes, and V. Singh. An NMF perspective on binary hashing. In *ICCV*, pages 4184–4192, 2015. 6, 8
[107] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011. 4, 6, 7, 9, 15, 17
[108] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, pages 3017–3024, 2013. 6, 11, 17
[109] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, pages 1070–1078, 2012. 4, 6, 9
[110] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming

space with multi-index hashing. In *CVPR*, pages 3108–3115, 2012. 13
[111] R. O'Donnell, Y. Wu, and Y. Zhou. Optimal lower bounds for locality sensitive hashing (except when q is tiny). In *ICS*, pages 275–283, 2011. 1
[112] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001. 14
[113] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006. 1
[114] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010. 15
[115] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. 15
[116] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *CVPR*, pages 3384–3391, 2010. 14
[117] D. Qin, X. Chen, M. Guillaumin, and L. J. V. Gool. Quantized kernel learning for feature matching. In *NIPS*, pages 172–180, 2014. 16
[118] D. Qin, Y. Chen, M. Guillaumin, and L. J. V. Gool. Learning to rank histograms for object retrieval. In *BMVC*, 2014. 16
[119] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008. 14
[120] R. Salakhutdinov and G. E. Hinton. Semantic hashing. In *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007. 1, 4, 12, 13
[121] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009. 1, 12
[122] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, pages 1665–1672, 2011. 1
[123] H. Sandhawalia and H. Jegou. Searching with expectations. In *ICASSP*, pages 1242–1245, 2010. 7
[124] J. Shao, F. Wu, C. Ouyang, and X. Zhang. Sparse spectral hashing. *Pattern Recognition Letters*, 33(3):271–277, 2012. 7
[125] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *CVPR*, pages 37–45, 2015. 6, 11, 16, 17
[126] F. Shen, C. Shen, Q. Shi, A. van den Hengel, and Z. Tang. Inductive hashing on manifolds. In *CVPR*, pages 1562–1569, 2013. 13
[127] F. Shen, X. Zhou, Y. Yang, J. Song, H. T. Shen, and D. Tao. A fast optimization method for general binary code learning. *IEEE Trans. Image Processing*, 25(12):5610–5621, 2016. 11
[128] X. Shi, F. Xing, J. Cai, Z. Zhang, Y. Xie, and L. Yang. Kernel-based supervised discrete hashing for image retrieval. In *ECCV*, pages 419–433, 2016. 6, 7
[129] A. Shrivastava and P. Li. Fast near neighbor search in high-dimensional binary data. In *ECML PKDD*, pages 474–489, 2012. 13
[130] A. Shrivastava and P. Li. Densifying one permutation hashing via rotation for fast near neighbor. In *ICML (1)*, pages 557–65, 2014. 1
[131] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, 2006. 14
[132] D. Song, W. Liu, R. Ji, D. A. Meyer, and J. R. Smith. Top rank supervised binary coding for visual search. In *ICCV*, pages 1922–1930, 2015. 6, 9
[133] J. Song, H. T. Shen, J. Wang, Z. Huang, N. Sebe, and J. Wang. A distance-computation-free search scheme for binary code databases. *IEEE Trans. Multimedia*, 18(3):484–495, 2016. 13
[134] J. Song, Y. Yang, Z. Huang, H. T. Shen, and J. Luo. Effective multiple feature hashing for large-scale near-duplicate video retrieval. *IEEE Transactions on Multimedia*, 15(8):1997–2008, 2013. 18
[135] J. Song, Y. Yang, Y. Yang, Z. Huang, and H. T. Shen. Inter-media hashing for large-scale retrieval from heterogeneous data sources. In *SIGMOD Conference*, pages 785–796, 2013. 18
[136] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1):66–78, 2012. 3, 6, 7
[137] A. B. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008. 14
[138] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):480–492, 2012. 12
[139] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *CVPR*, pages 2320–2327, 2012. 1
[140] L. von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *CHI*, pages 55–64, 2006. 14
[141] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010. 3, 6, 7, 8
[142] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, pages 1127–1134, 2010. 6, 7
[143] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(12):2393–2406, 2012. 6, 7, 17
[144] J. Wang and S. Li. Query-driven iterated neighborhood graph search for large scale indexing. In *ACM Multimedia*, pages 179–188, 2012. 13
[145] J. Wang, W. Liu, S. Kumar, and S. Chang. Learning to hash for indexing big data - A survey. *Proceedings of the IEEE*, 104(1):34–57, 2016. 2, 5
[146] J. Wang, W. Liu, A. X. Sun, and Y.-G. Jiang. Learning hash codes with listwise supervision. In *ICCV*, pages 3032–3039, 2013. 6, 9
[147] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014. 1, 2, 5
[148] J. Wang, H. T. Shen, S. Yan, N. Yu, S. Li, and J. Wang. Optimized distances for binary code ranking. In *ACM Multimedia*, pages 517–526, 2014. 4, 12, 16
[149] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li. Optimized cartesian k-means. *CoRR*, abs/1405.4054, 2014. 12
[150] J. Wang, J. Wang, N. Yu, and S. Li. Order preserving hashing for approximate nearest neighbor search. In *ACM Multimedia*, pages 133–142, 2013. 6, 9
[151] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo. Fast neighborhood graph search using cartesian concatenation. In *ICCV*, pages 2128–2135, 2013. 2, 14
[152] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, and X.-S. Hua. Trinary-projection trees for approximate nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2013. 2, 14
[153] Q. Wang, D. Zhang, and L. Si. Weighted hashing for fast large scale similarity search. In *CIKM*, pages 1185–1188, 2013. 7
[154] X. Wang, T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Supervised quantization for similarity search. In *CVPR*, 2016. 6, 16, 17, 18
[155] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *ECCV (5)*, pages 340–353, 2012. 6, 8, 13
[156] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008. 1, 3, 5, 6, 8, 17
[157] C. Wu, J. Zhu, D. Cai, C. Chen, and J. Bu. Semi-supervised nonlinear hashing using bootstrap sequential projection learning. *IEEE Trans. Knowl. Data Eng.*, 25(6):1380–1393, 2013. 7
[158] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, pages 2156–2162, 2014. 6, 13
[159] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, and D. Cai. Harmonious hashing. In *IJCAI*, 2013. 6, 10
[160] 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. 9, 13
[161] X. Xu, F. Shen, Y. Yang, H. T. Shen, and X. Li. Learning discriminative binary codes for large-scale cross-modal retrieval. *IEEE Transactions on Image Processing*, 26(5):2494–2507, May 2017. 18
[162] H. Yang, X. Bai, J. Zhou, P. Ren, Z. Zhang, and J. Cheng. Adaptive object retrieval with kernel reconstructive hashing. In *CVPR*, pages 1955–1962, 2014. 8
[163] Q. Yang, L.-K. Huang, W.-S. Zheng, and Y. Ling. Smart hashing update for fast response. In *IJCAI*, 2013. 12
[164] Y. Yang, F. Shen, H. T. Shen, H. Li, and X. Li. Robust discrete spectral hashing for large-scale image semantic indexing. *IEEE Trans. Big Data*, 1(4):162–171, 2015. 7
[165] F. Yu, S. Kumar, Y. Gong, and S.-F. Chang. Circulant binary embedding. In *ICML (2)*, pages 946–954, 2014. 17
[166] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *SIGIR*, pages 18–25, 2010. 6, 7
[167] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T. Chua. Discrete collaborative filtering. In *SIGIR*, pages 325–334, 2016. 18
[168] H. Zhang, N. Zhao, X. Shang, H. Luan, and T. Chua. Discrete image hashing using large weakly annotated photo collections. In *AAAI*, pages 3669–3675, 2016. 11
[169] L. Zhang, Y. Zhang, J. Tang, X. Gu, J. Li, and Q. Tian. Topology preserving hashing for similarity search. In *ACM Multimedia*, pages 123–132, 2013. 6, 8

[170] S. Zhang, J. Li, J. Guo, and B. Zhang. Scalable discrete supervised hash learning with asymmetric matrix factorization. *CoRR*, abs/1609.08740, 2016. 7, 11
[171] T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *ICML (2)*, pages 838–846, 2014. 6, 11, 12, 15, 16, 17
[172] T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *CVPR*, 2015. 12, 13, 16, 17
[173] T. Zhang and J. Wang. Collaborative quantization for cross-modal similarity search. In *CVPR*, pages 2036–2045, 2016. 18
[174] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015. 6, 13
[175] K. Zhao, H. Lu, and J. Mei. Locality preserving hashing. In *AAAI*, pages 2874–2881, 2014. 11
[176] Y. Zhen and D.-Y. Yeung. Active hashing and its application to image and text retrieval. *Data Min. Knowl. Discov.*, 26(2):255–274, 2013. 12
[177] X. Zhu, Z. Huang, H. Cheng, J. Cui, and H. T. Shen. Sparse hashing for fast multimedia search. *ACM Trans. Inf. Syst.*, 31(2):9, 2013. 7
[178] X. Zhu, Z. Huang, H. T. Shen, and X. Zhao. Linear cross-modal hashing for efficient multimedia search. In *ACM Multimedia*, pages 143–152, 2013. 18
[179] 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. 7