A Survey on Learning to Hash

Jingdong Wang, Heng Tao Shen, and Ting Zhang

Abstract—Nearest neighbor search is a problem of finding the data points from a 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 future directions.

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 approximate nearest neighbor search, aims to transforming the 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 [29] and learning to hash. Locality sensitive hashing (LSH) is data-independent. The research efforts mainly come from the theory community, such as proposing random hash functions satisfying the local sensitive property for various distance measures [5], [6], [7], [10], [11], [69], [78], proving better search efficiency and accuracy [10], [80], and developing better search schemes [15], [15], [67], and the machine learning community, such as developing hash functions providing a similarity estimator with smaller variance [47], [37], [51], [36], or smaller storage [49], [50], or faster computation of hash functions [48], [51], [36], [88].

Learning to hash, the interest in 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 to the search result in the original space as possible, and the search cost and the space cost are also small. Since the pioneering algorithm, spectral hashing [107], learning to hash has been attracting a large amount of research interests in computer vision and machine learning and has been applied to a wide-range of applications such as large scale object retrieval [33], image classification and detection [85] [94], and so on.

The main methodology of learning to hash is similarity preserving, i.e., minimizing the gap between the similarities or similarity orders computed/given in the original space and 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 more widely interested and studied in most real applications, e.g., large scale search by image and image classification, and thus the main focus in this paper.

This survey categorizes the algorithms according to the similarity preserving manners into: pairwise similarity preserving, multiwise similarity preserving, implicit similarity preserving, and quantization that, we show, is a form of pairwise similarity preserving, and discusses other problems, including evaluation datasets and schemes, online search given the hash codes, and so on. In addition, we present the empirical observation that the quantization approach outperforms other approaches and give some analysis about this observation. Finally, we point out the future directions, such as an end-to-end learning strategy for real applications, directly learning the hash codes from the object, e.g., image, instead of first learning the representations and then learning the hash codes from the representations.

1.1 Organization of This Paper

The organization of the remaining part is given as the following. Section 2 introduces the exact and approximate nearest neighbor search problems, and the search algorithms with hashing. Section 3 provides the basic concepts in the learning-to-hashing approach and categorizes the existing algorithms from the perspective of loss function into four main classes: pairwise alignment, multiple-wise alignment,

- *J. Wang is with Microsoft Research, Beijing, P.R. China. E-mail: jingdw@microsoft.com*
- *H.T. Shen are with School of Information Technology and Electrical Engineering, The University of Queensland, Australia. Email:shenht@itee.uq.edu.au*
- *T. Zhang are with University of Science and Technology, China. Email: zting@mail.ustc.edu.cn*

implicit alignment and quantization, which are discussed in Sections 4, 5, 6, and 7, respectively. Section 8 presents other works in learning to hash. Sections 9 and 10 gives some evaluation protocols and performance analysis. Finally, Sections 11 and 12 point out the future research trends and conclude this survey, respectively.

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 N item $\mathbf{q}$ from a set of 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-NN search, where K nearest neighbors ($\mathrm{KNN}(\mathbf{q})$) are needed to be found.

The distance between an arbitrary 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 induced 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 notions of the search database, for example, the data item is formed by a set, and distance measures, such as ℓ_1 distance, cosine similarity and so on, are also possible.

There exist efficient algorithms (e.g., k-d trees and its variants) for exact nearest neighbor search in low-dimensional cases. In large scale high-dimensional cases, it turns out that the problems become hard and most algorithms even take higher computational cost than the naive solution, linear scan. Therefore, a lot of recent efforts are moved to search approximate nearest neighbors: $(1 + \epsilon)$-approximate nearest neighbor search [29], which is studied mainly in the theory community, and time-fixed approximate nearest neighbor search. Other nearest neighbor search problems include (approximate) fixed-radius near neighbor (R-near neighbor) problem, and randomized nearest neighbor search which the locality sensitive hashing research community is typical interested in.

Time-fixed approximate nearest neighbor search is studied mainly in machine learning and computer vision for real applications, such as the learning to hash approach, though there is usually lack of elegant theory. The goal is to make the average search as accurate as possible by comparing the returned K approximate nearest neighbors and the K exact nearest neighbors, and the query cost as small as possible.

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 (also known as hash values, 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, and may be a binary value, 1 and 0 (or -1) or an integer, 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 $\mathbf{y} = [y_1\, y_2\, \cdots\, y_M]^T$ and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x})\, h_2(\mathbf{x})\, \cdots\, h_M(\mathbf{x})]^T$. 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 to reduce the number of the distance computations from N to N' ($N \gg N'$), and thus the time complexity is reduced from $O(Nd)$ to $O(N'd)$. The data structure, called hash table (a form of inverted index), is composed of buckets with each 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 aims to maximize the probability of collision of near items. 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 followed by a reranking step: rerank the retrieved nearest neighbor candidates according to the true distances computed using the original features and attain the K nearest neighbors or R-near 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 of) the query $\mathbf{h}(\mathbf{q})$ in terms of the distances in the coding space. The second way is to construct more 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 $\mathbf{q}$. To guarantee the high precision, each of the L hash codes, $\mathbf{y}_i$, 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 nonempty are retained by using conventional hashing of 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 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 function $\_\_\mathrm{popcnt}$ 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 K nearest neighbors or R-near neighbors.

This strategy exploits one main advantage of hash codes: the distance using hash codes is efficiently computed and

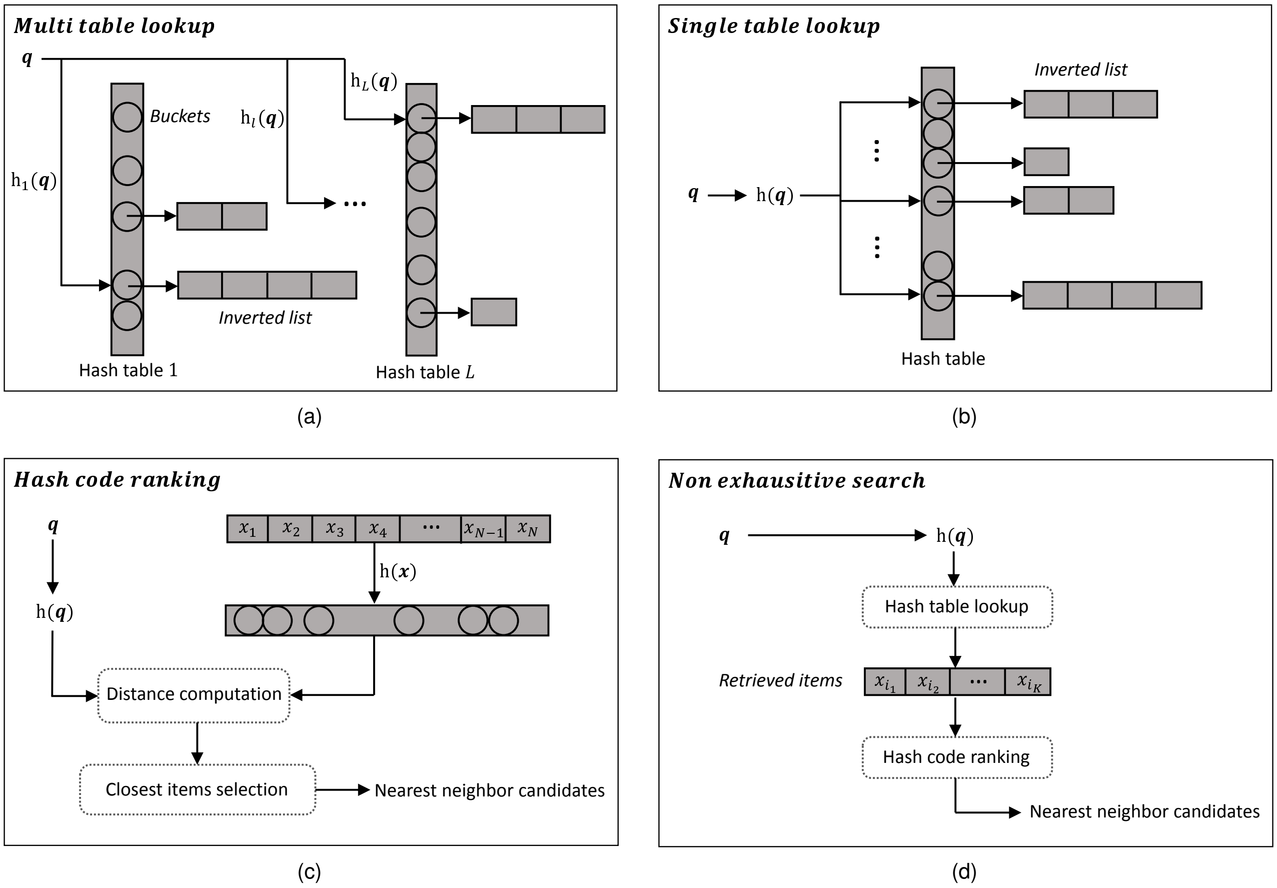


Fig. 1. Illustrating the search strategies. (a) Multi table lookup: the list corresponding to the hash code of the query in each table is retrieved. (b) Single table lookup: the lists corresponding to and near to the hash code of the query are retrieved. (c) Hash code ranking: compare the query with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

the cost is much smaller than that of the computation in the original input space, reduced from d to d' where $d \gg d'$ and the whole cost is reduced from Nd to Nd'.

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 is observed that hash table lookup with binary hash codes shows inferior performance and hence 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. In comparison to hash table lookup, hash code ranking is superior in search accuracy while inferior in search efficiency, and overall performs better, and thus more widely used in real applications and in experimental evaluations.

A practical way is to do a non-exhaustive search: first retrieve a small set of candidates using inverted index, and then compute the distances of the query with the candidates using the hash codes, providing the top candidates subsequently reranked using the original features. Other research efforts include organizing the hash codes to avoid exhaustive search with a data structure, such as a tree or a graph structure [73].

3 Learning to Hash

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

3.1 Hash Function

The hash function can be a form based on linear projection, kernels, spherical function, neural network, a non-parametric function, and so on. One popular hash function is the linear hash function:

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

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

$$y = h(\mathbf{x}) = \operatorname{sgn}(\sum_{t=1}^{T} w_t K(\mathbf{s}_t, \mathbf{x}) + b), \tag{2}$$

is also adopted in some approaches, 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 algorithm, 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.

Hash functions are 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 leads to better search accuracy as they are more flexible. Almost all the methods using a linear hash function can be extended to kernelized hash functions. Thus we do not use hash functions to categorize the hash algorithms.

There are various algorithms developed and exploited to optimize the hash function parameters. We summarize the common ways to handle the sgn function which is a main factor leading to the difficulty of estimating the parameters (e.g., the projection vector $\mathbf{w}$ in the linear hash function). There are roughly three approximation estimation schemes. The first one is a continuous relaxation, e.g., sigmoid relaxation $\operatorname{sgn}(z) \approx \phi_\alpha(z) = \frac{1}{1+e^{-\alpha z}}$. The second one is directly dropping the sign function $\operatorname{sgn}(z) \approx z$. The third one is a two-step scheme [53], [54] with its extension to iterative two step optimization [17]: optimizing the binary codes without considering the hash function, followed by estimating the function parameters from the optimized hash codes.

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 a Euclidean distance, $\|\mathbf{x}_i - \mathbf{x}_j\|_2$ or other metric distances. The similarity is often defined as a function about the distance d^o_{ij}: $s^o_{ij} = g(d^o_{ij})$, 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 may be other similarity forms, such as the cosine similarity $\frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$ and so on. Besides, the semantic similarity is also 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 not the same and mathematically formed as

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

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, meaning the number of bits where the values are not the same. The inner product $s^h_{ij} = \mathbf{y}_i^T \mathbf{y}_j$ 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 $s^h_{ij} = \mathbf{y}_i^T \mathbf{\Lambda} \mathbf{y}_j$ where $\mathbf{\Lambda} = \operatorname{Diag}(\lambda_1, \lambda_2, \cdots, \lambda_M)$ is a diagonal matrix and each diagonal entry is the weight of the corresponding hash code.

Besides the Hamming distance/simialrity and its variants, the Euclidean distance is used, typically in quantization approaches, and 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 approximate to $\mathbf{x}_j$ $d^h_{qj} = \|\mathbf{q} - \mathbf{c}_{y_j}\|_2$ (asymmetric distance, which is preferred as the accuracy is higher and the time cost is almost the same), which is efficiently evaluated in the search stage by using a distance lookup table. There are also some works learning/optimizing the distances between hash codes 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 aims to find the optimal approximation of the item in terms of the distortion error, and we will show that it is a way of preserving the pairwise similarities. Besides similarity preserving items, some approaches introduce bucket balance or its approximate variants as extra constraints.

3.4 Categorization

Our survey categorizes the existing algorithms to various classes: the pairwise similarity preserving class, the multiwise similarity preserving class, the implicit similarity preserving class, as well as the quantization class, according to how the loss function is formulated. We separate the quantization class from the pairwise similarity preserving class as they are very different in formulations though the quantization class can be explained from the perspective of pairwise similarity preserving. In the following descriptions, 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.

TABLE 1
A summary of representative hashing algorithms with respect to similarity preserving functions, code balance, hash function and similarity in the coding space. 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, C = Cosine, E = Euclidean distance.

Approach		Similarity pres.	Code balance	Hash function	Code sim.
pairwise	Spectral hashing [107]	$s^o_{ij} d^h_{ij}$	BB + BU	Eigenfunction	H
	ICA hashing [24]		BB + BMIM	Linear	
	Kernelized spectral hashing [25]		BB + BU	Kernel	
	Hashing with graphs [60]		BB + BU	Eigenfunction	
	Discrete graph hashing [58]		BB + BU	kernel	
	Compressed hashing [56]		BB	Kernel	
	Self-taught hashing [115]		BB + BU	Linear	
	LDA hashing [92]		BU	Linear	
	Minimal loss hashing [74]		-	Linear	
	Semi-supervised hashing [96], [97], [98]	$s^o_{ij} s^h_{ij}$	BU	Linear	H
	Topology preserving hashing [116]	$d^o_{ij} d^h_{ij} + s^o_{ij} d^h_{ij}$	BU	Linear	H
	Binary reconstructive embedding [45]	$(d^o_{ij} - d^h_{ij})^2$	-	Kernel	H
	Supervised hashing with kernels [59]	$(s^o_{ij} - s^h_{ij})^2$	-	Kernel	H
	Bilinear hyperplane hashing [61]		-	BiLinear	
	Label-regularized maximum margin hashing [71]		BB	Kernel	
	Multi-dimensional spectral hashing [106]		BI + BU	Eigenfunction	WH
	Spec hashing [55]	$\mathrm{KL}(\{\bar{s}^o_{ij}\}, \{\bar{s}^h_{ij}\})$	-		H
multiwise	Order preserving hashing [102]	rank order	BKB	Linear	H
	Triplet loss hashing [76]	triplet loss	BU	Linear + NN	H
	Listwise supervision hashing [99]	triplet loss	BU	Linear	H
quantization	Isotropic hashing [43]	$\approx \lvert\mathbf{x} - \mathbf{y}\rvert_2$	BU	Linear	H
	Iterative quantization [20], [21]	$\lvert\mathbf{x} - \mathbf{y}\rvert_2$	-	Linear	H
	Harmonious hashing [110]		BU		
	Matrix hashing [18]		-		
	Angular quantization [19]		-		C
	Product quantization (PQ) [32]	$\lvert\mathbf{x} - \mathbf{y}\rvert_2$	-	Nearest vector	E
	Cartesian k-means [75] (Optimized PQ [16])		-		
	Composite quantization [117]		-		

4 Pairwise Similarity Preserving

The algorithms preserving the distances or similarities of a pair of items computed from the input space and the Hamming coding space are roughly divided as 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 $\bar{s}^o_{ij} = \frac{s^o_{ij}}{\sum_{(i',j')\in\mathcal{E}} s^o_{i'j'}}$.

The following reviews these groups of algorithms except the distance-similarity product minimization group to which we are not aware of any algorithms belonging. We also point out the relation between similarity-distance product minimization and similarity-similarity product minimization, the relation between similarity-similarity product minimization and similarity-similarity difference minimization, as well as the relation between distance-distance product minimization and distance-distance difference minimization

4.1 Similarity-Distance Product Minimization

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

4.1.1 *Spectral Hashing*

The goal of spectral hashing [107] is to minimize $\min \sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij}$, 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: $s^o_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$. 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 and the larger Hamming distance. The objective function can be written in a matrix form,

$$\min \sum_{(i,j)\in\mathcal{E}} s^o_{ij} d^h_{ij} = \operatorname{trace}(\mathbf{Y}(\mathbf{D}-\mathbf{S})\mathbf{Y}^T), \tag{4}$$

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 $\mathbf{D} = \operatorname{Diag}(d_{11}, \cdots, d_{NN})$ 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 a around 50% chance of being 1 or -1. Bit uncorrelation means that different bits are uncorrelated. The two conditions are formulated as,

$$\mathbf{Y}\mathbf{1} = 0, \quad \mathbf{Y}\mathbf{Y}^T = \mathbf{I}, \tag{5}$$

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).
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 $h(\mathbf{x}) = \operatorname{sgn}(\sin(\frac{\pi}{2} + \gamma\mathbf{w}^T\mathbf{x}))$, where γ depends on the frequency m and the range of the projection along the direction $\mathbf{w}$.

Analysis: In the case that 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 [21], [20], and isotropic hashing [43].

Regarding the performance, it is good for a short hash code but poor for a long hash code. The reason includes three aspects. First, the assumption that the data follows 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 eigenfunctions 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 even far 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: *ICA hashing* [24] studies code balance in spectral hashing and formulates it as maximizing the entropy $\operatorname{Entropy}(y_1, y_2, \cdots, y_M) = \sum_{m=1}^M \operatorname{Entropy}(y_m) - I(y_1, y_2, \cdots, y_M)$. It is subsequently formulated as bit balance: $\operatorname{E}(y_m) = 0$ (where $\operatorname{Entropy}(y_m)$ is maximized as 1) and mutual information minimization, $I(y_1, y_2, \cdots, y_M)$, which is related to $I(y_1, y_2, \cdots, y_M) = I(\mathbf{w}_1^T\mathbf{x}, \mathbf{w}_2^T\mathbf{x}, \cdots, \mathbf{w}_M^T\mathbf{x})$ for the linear hash function $\mathbf{y} = \operatorname{sgn}(\mathbf{W}^T\mathbf{x} - \mathbf{b})$ when using the scheme used in independent component analysis. The overall formulation is to minimize the approximate mutual information subject to $\operatorname{trace}(\mathbf{Y}(\mathbf{D}-\mathbf{S})\mathbf{Y}^T) \leqslant \eta$ and the bit balance constraint.

Kernelized spectral hashing [25] extends spectral hashing using the kernel hash function. *Hypergraph spectral hashing* [124], [64] extends spectral hashing from an ordinary (pairwise) graph to a hypergraph (multiwise graph), and formulates the problem using the hypergraph Laplacian. *Sparse spectral hashing* [86] combines boosting similarity sensitive hashing and sparse principal component analysis under the original spectral hashing framework. *Hashing with graphs* [60] uses the anchor graph to approximate the neighborhood graph and accordingly uses the graph Laplacian over the anchor graph to approximate the graph Laplacian of the original graph for fast computing the eigenvectors with a side contribution of exploiting a hierarchical hashing to address the boundary issue. Its extension, *discrete Graph Hashing* [58], provides a new optimization algorithm. *Compressed hashing* [56] borrows the idea about anchor graph in [60] and uses the anchors to generate a sparse representation.

Weighted hashing [105] extends spectral hashing by adopting the weighted Hamming distance between hash codes, $\|\boldsymbol{\alpha}^T(\mathbf{y}_i - \mathbf{y}_j)\|_2^2$ and introducing an extra constraint: $\frac{\alpha_1}{\operatorname{var}(y_1)} = \frac{\alpha_2}{\operatorname{var}(y_2)} = \cdots = \frac{\alpha_M}{\operatorname{var}(y_M)}$, where $\boldsymbol{\alpha} = [\alpha_1\alpha_2\cdots\alpha_M]^T$ and $\operatorname{var}(\cdot)$ is the variance operator. *Self-taught hashing* [115] changes the constraints in the spectral hashing to $\mathbf{YDY}^T = \mathbf{I}$ and $\mathbf{YD1} = \mathbf{0}$ and uses the linear hash function, $h(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}^T\mathbf{x} + b)$. *Sparse hashing* [122] adds two parts into the objective function: the sparsity constraint of the hash codes $\|\mathbf{y}\|_1$ and the reconstruction constraint from the hash codes $\|\mathbf{x} - \mathbf{P}^T\mathbf{y}\|_2^2$ and uses the linear hash function, $h(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}^T\mathbf{x})$.

Other extensions include *principal component hashing* [68] that also uses the principal direction to formulate the hash function, *spectral hashing with semantically consistent graph* [52] that constructs a semantically consistent graph by learning a linear transformation matrix such that the similarity computed over the transformed space is consistent to the semantic similarity as well as the Euclidean distance-based similarity, *transform coding* [4] that transforms the data using PCA and then assigns several bits to each principal direction using bit allocation to determine which principal direction is used and how many bits are assigned to such a direction, and *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.

4.1.2 Variants

Linear discriminant analysis (LDA) hashing [92] 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 $\mathbf{x}_i$ are a similar pair, $(i,j) \in \mathcal{E}^+$, and $s^o_{ij} = -1$ if data items $\mathbf{x}_i$ and $\mathbf{x}_i$ are a dissimilar pair, $(i,j) \in \mathcal{E}^-$ (2) a linear hash function is used: $\mathbf{y} = \operatorname{sgn}(\mathbf{W}^T\mathbf{x} + \mathbf{b})$, 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:

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

The projection matrix $\mathbf{W}$ and the threshold $\mathbf{b}$ are separately optimized: (1) drop the sgn function in Equation (6), leading an eigenvalue decomposition problem, to estimate an orthogonal matrix $\mathbf{W}$; (2) estimate $\mathbf{b}$ by minimizing Equation (6) with fixing $\mathbf{W}$ through a simple 1D search scheme.

The loss function in *minimal loss hashing* [74] 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,

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

where ρ is a hyper-parameter used as a threshold in the Hamming space to differentiate similar pairs from dissimilar pairs, λ is also a 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: $\mathbf{y} = \operatorname{sgn}(\mathbf{W}^T\mathbf{x})$. The projection matrix $\mathbf{W}$ is estimated by transforming $\mathbf{y} = \operatorname{sgn}(\mathbf{W}^T\mathbf{x}) = \arg\max_{\mathbf{y}'\in\mathcal{H}} \mathbf{h}'^T\mathbf{W}^T\mathbf{x}$ and optimizing using structured prediction with latent variables. The hyperparameters ρ and λ are chosen via cross-validation.

Comments: The main differences of the three representative algorithms, 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 they in LDA hashing and minimal loss hashing are adopted 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 [96], [97], [98] 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 a same class or are nearby points, and -1 otherwise. The similarity in the coding space is defined as $s^h_{ij} = \mathbf{y}_i^T\mathbf{y}_j$. Thus, the objective function is rewritten as maximizing:

$$\sum_{(i,j)\in\mathcal{E}} s^o_{ij}\mathbf{y}_i^T\mathbf{y}_j. \tag{8}$$

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

$$\operatorname{trace}(\mathbf{Y}\mathbf{S}\mathbf{Y}^T) + \eta \operatorname{trace}(\mathbf{Y}\mathbf{Y}^T), \tag{9}$$

subject to $\mathbf{W}^T\mathbf{W} = \mathbf{I}$, 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 [98]: 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 semi-supervised hashing to nonlinear hash functions is presented in [108] using the kernel hash function. An iterative two-step optimization using graph cuts is given in [17].

Comments: It is interesting that $\sum_{(i,j)\in\mathcal{E}} s^o_{ij}\mathbf{y}_i^T\mathbf{y}_j = \texttt{const} - \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} s^o_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = \texttt{const} - \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} s^o_{ij}d^h_{ij}$ if $\mathbf{y} \in \{1,-1\}^M$, where $\texttt{const}$ is a constant variable (and thus $\operatorname{trace}(\mathbf{Y}\mathbf{S}\mathbf{Y}^T) = \texttt{const} - \operatorname{trace}(\mathbf{Y}(\mathbf{D} - \mathbf{S})\mathbf{Y}^T)$). 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 [116] formulates the problem by starting with this rule:

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

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

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

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

The overall formulation is given as follows,

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

where $\alpha\mathbf{I}$ introduces a regularization term, $\operatorname{trace}(\mathbf{Y}\mathbf{Y}^T)$, maximizing the variances, which is the same to semi-supervised hashing [96] for bit balance. The problem is optimized by dropping the sgn operator in the hash function $\mathbf{y} = \operatorname{sgn}(\mathbf{W}^T\mathbf{x})$ and letting $\mathbf{W}^T\mathbf{X}\mathbf{L}\mathbf{X}\mathbf{W}$ be an identity matrix.

4.4 Distance-Distance Difference Minimization

Binary reconstructive embedding [45] 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,

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

The kernel hash function is used:

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

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 and using a two-step scheme, an alternative optimization scheme is presented in [45]: fixing all but one weight w_{mt} and optimizing 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 $O(N \log N + |\mathcal{E}|)$.

Comments: We have the following equation,

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

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

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

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 maximizing 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* [59], one representative approach in this group, aims to minimize an objective function,

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

where $s^o_{ij} = 1$ if (i,j) is similar, and $s^o_{ij} = -1$ if dissimilar. $\mathbf{y} = \mathbf{h}(\mathbf{x})$ is a kernel hash function. Kernel reconstructive hashing [112] extends this technique using a normalized Gaussian kernel similarity.

Comments: We have the following equation,

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

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

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

This shows that the difference between similarity-similarity difference minimization and similarity-similarity product maximization lies on $\min \sum_{(i,j)\in\mathcal{E}} (s^h_{ij})^2$, minimizing the similarities between the data items in the hash space, intuitively letting the hash codes 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: *Bilinear hyperplane hashing* [61] extends the formulation of supervised hashing with kernels. It uses the same objective function, and introduces two differences: (1) a bilinear hyperplane hashing function,

$$h(\mathbf{z}) = \begin{cases} \operatorname{sgn}(\mathbf{u}^T\mathbf{z}\mathbf{z}^T\mathbf{v}) & \text{if } \mathbf{z} \text{ is a database vector,} \\ \operatorname{sgn}(-\mathbf{u}^T\mathbf{z}\mathbf{z}^T\mathbf{v}) & \text{if } \mathbf{z} \text{ is a hyperplane normal,} \end{cases} \tag{22}$$

where the bilinear projection vectors $\mathbf{u}$ and $\mathbf{v}$ are the parameters of the hash functions; (2) a new definition for the similarity in the input space,

$$s^o_{ij} = \begin{cases} 1 & \text{if } \cos(\theta_{\mathbf{x}_i,\mathbf{x}_j}) \geqslant t_1 \\ -1 & \text{if } \cos(\theta_{\mathbf{x}_i,\mathbf{x}_j}) \leqslant t_2 \\ 2|\cos(\theta_{\mathbf{x}_i,\mathbf{x}_j})| - 1 & \text{otherwise} \end{cases}, \tag{23}$$

where t_1 and t_2 are two thresholds. The problem is solved by relaxing sgn with the sigmoid-shaped function and finding the solution with the gradient descent algorithm.

Multi-dimensional spectral hashing [106] uses a similar objective function, but with a weighted Hamming distance,

$$\min \sum_{(i,j)\in\mathcal{E}} (s^o_{ij} - \mathbf{y}_i^T \boldsymbol{\Lambda} \mathbf{y}_j)^2, \tag{24}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix. Both $\boldsymbol{\Lambda}$ and hash codes $\{\mathbf{y}_i\}$ are needed to be optimized. The algorithm for solving the problem 24 to compute hash codes is similar to that given in spectral hashing [107].

Label-regularized maximum margin hashing [71] computes the hash function one by one, with the objective function consisting of three components: the similarity-similarity difference, $\sum_{(i,j)\in\mathcal{E}} (s^o_{ij} - y_i y_j)^2$ ($s^o_{ij} \in \{-1, 1\}, y_i \in \{-1, 1\}$), a hinge loss from the hash function, $\max(0, 1 - y_j(\mathbf{w}^T\mathbf{x} + b))$, as well as the maximum margin part,

$$\min_{\{y_i\},\mathbf{w},b,\{\xi_i\},\{\zeta_{ij}\}} \|\mathbf{w}\|_2^2 + \lambda_1 \sum_{(i,j)\in\mathcal{E}} (s^o_{ij} - y_i y_j)^2 + \tag{25}$$

$$\lambda_2 \sum_{j=1}^{N} \max(0, 1 - y_j(\mathbf{w}^T\mathbf{x} + b)). \tag{26}$$

A bit balance constraint is introduced, $-l \leqslant \mathbf{w}^T\mathbf{x}_i + b \leqslant l$ to encourage that half of data items are mapped to -1 or 1.

4.6 Normalized Similarity-Similarity Divergence Minimization

Spec hashing [55], 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,

$$\operatorname{KL}(\{\bar{s}^o_{ij}\}, \{\bar{s}^h_{ij}\}) = \text{const} - \sum_{(i,j)\in\mathcal{E}} \bar{s}^o_{ij} \log \bar{s}^h_{ij}. \tag{27}$$

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, $\bar{s}^h_{ij} = \frac{1}{Z}\exp(-\lambda d^h_{ij})$, where Z is a normalization variable $Z = \sum_{ij}\exp(-\lambda d^h_{ij})$.

Supervised binary hash code learning [13] presents a supervised binary hash code 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 [102] 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 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 query 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 deal case the probability 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,

$$L(\mathbf{h}(\cdot);\mathcal{X}) = \sum_{n\in\{1,\cdots,N\}}\sum_{m=0}^{M}(|\mathcal{C}^o_{nm} - \mathcal{C}^h_{nm}| + |\mathcal{C}^h_{nm} - \mathcal{C}^o_{nm}|), \tag{28}$$

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.

Triplet loss hashing [76] 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

$$\ell_{\text{triplet}}(\mathbf{y}, \mathbf{y}^+, \mathbf{y}^-) = \max(1 - \|\mathbf{y} - \mathbf{y}^-\|_1 + \|\mathbf{y} - \mathbf{y}^+\|_1, 0). \tag{29}$$

The objective function is given as follows,

$$\sum_{(\mathbf{x},\mathbf{x}^+,\mathbf{x}^-)\in\mathcal{D}} \ell_{\text{triplet}}(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}^+), \mathbf{h}(\mathbf{x}^-)) + \frac{\lambda}{2}\operatorname{trace}(\mathbf{W}^T\mathbf{W}), \tag{30}$$

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 [74]. The extension to asymmetric Hamming distance is also discussed in [76].

Listwise supervision hashing [99] also uses triplets of items. The formulation is based on a triplet tensor $\mathbf{S}$ defined as follows,

$$s^o_{ijk} = s(\mathbf{q}_i; \mathbf{x}_j, \mathbf{x}_k) = \begin{cases} 1 & \text{if } \mathrm{s}^o(\mathbf{q}_i, \mathbf{x}_j) < \mathrm{s}^o(\mathbf{q}_i, \mathbf{x}_k) \\ -1 & \text{if } \mathrm{s}^o(\mathbf{q}_i, \mathbf{x}_j) > \mathrm{s}^o(\mathbf{q}_i, \mathbf{x}_k) \\ 0 & \text{if } \mathrm{s}^o(\mathbf{q}_i, \mathbf{x}_j) = \mathrm{s}^o(\mathbf{q}_i, \mathbf{x}_k) \end{cases}. \tag{31}$$

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

$$\sum_{i,j,k} s^h_{ijk} s^o_{ijk}, \tag{32}$$

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

$$-\sum_{i,j,k}\mathbf{h}(\mathbf{q}_i)^T(\mathbf{h}(\mathbf{x}_j) - \mathbf{h}(\mathbf{x}_k))s^o_{ijk}, \tag{33}$$

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

Comments: Order preserving hashing aims to consider 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 lie in how to formulate the loss function.

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 [41] 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,

$$\max\frac{1}{\|\mathbf{w}\|_2}\min\{\min_{i=1,\cdots,\frac{N}{2}}(\mathbf{w}^T\mathbf{x}^+_i + b), \min_{i=1,\cdots,\frac{N}{2}}(-\mathbf{w}^T\mathbf{x}^-_i - b)\}, \tag{34}$$

where $\{\mathbf{x}^+_i\}$ are the positive samples and $\{\mathbf{x}^-_i\}$ are the negative samples.

Complementary projection hashing [40], similar to complementary hashing [111], 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 $\mathcal{H}(\epsilon - |\mathbf{w}^T\mathbf{x} + b|)$, where $\mathcal{H}(\cdot) = \frac{1}{2}(1 + \operatorname{sgn}(\cdot))$ is the unit step function. Moreover, the bit balance condition, $\mathbf{Y1} = 0$, and the bit uncorrelation condition, the non-diagonal entries in $\mathbf{YY}$ are 0, are considered. An extension is also given by using the kernel hash function. In addition, in learning the mth hash function, the data item is weighted by a variable, computed from the previously-learnt hash function, according to the previously computed $(m-1)$ hash functions: $u^m = 1 + \sum_{j=1}^{m-1}\mathcal{H}(\epsilon - |\mathbf{w}_j^T\mathbf{x} + b_j|)$.

Spherical hashing [26] 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 similarity 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

We show that the quantization approach can be derived from the perspective of distance-distance difference minimization. Considering two points $\mathbf{x}_i$ and $\mathbf{x}_j$ and its approximation $\mathbf{z}_i$ and $\mathbf{z}_j$, we have

$$\begin{aligned}
& |d^o_{ij} - d^h_{ij}| && (35)\\
=\, & ||\mathbf{x}_i - \mathbf{x}_j|_2 - |\mathbf{z}_i - \mathbf{z}_j|_2| && (36)\\
=\, & ||\mathbf{x}_i - \mathbf{x}_j|_2 - |\mathbf{x}_i - \mathbf{z}_j|_2 + |\mathbf{x}_i - \mathbf{z}_j|_2 - |\mathbf{z}_i - \mathbf{z}_j|_2| && (37)\\
\leqslant\, & ||\mathbf{x}_i - \mathbf{x}_j|_2 - |\mathbf{x}_i - \mathbf{z}_j|_2| + ||\mathbf{x}_i - \mathbf{z}_j|_2 - |\mathbf{z}_i - \mathbf{z}_j|_2| && (38)\\
\leqslant\, & |\mathbf{x}_j - \mathbf{z}_j|_2 + |\mathbf{x}_i - \mathbf{z}_i|_2. && (39)
\end{aligned}$$

Thus, $|d^o_{ij} - d^h_{ij}|^2 \leqslant (|\mathbf{x}_j - \mathbf{z}_j|_2^2 + |\mathbf{x}_i - \mathbf{z}_i|_2^2)$, and

$$\begin{aligned}
& \min \sum_{i,j \in \{1,2,\cdots,N\}} |d^o_{ij} - d^h_{ij}|^2 && (40)\\
\leqslant\, & \min \sum_{i,j \in \{1,2,\cdots,N\}} (|\mathbf{x}_j - \mathbf{z}_j|_2^2 + |\mathbf{x}_i - \mathbf{z}_i|_2^2) && (41)\\
=\, & \min 2 \sum_{i \in \{1,2,\cdots,N\}} |\mathbf{x}_i - \mathbf{z}_i|_2^2. && (42)
\end{aligned}$$

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.

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 $\{[y_1\ y_2\ \cdots\ y_M]^T \mid y_m \in \{-1, 1\}\}$ or the rotated hypercubic vertices. The well-known scalar quantization, the simplest hypercubic quantization, can be derived by minimizing $|\mathbf{x}_i - \mathbf{y}_i|_2^2$ subject to $\mathbf{y}_i \in \{1, -1\}$. The local digit coding approach [44] also belongs to this category.

7.1.1 Iterative quantization

Iterative quantization [20], [21] preprocesses the data, by reducing the dimension using PCA to M dimensions, $\mathbf{v} = \mathbf{P}^T \mathbf{x}$, 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,

$$\min \|\mathbf{Y} - \mathbf{R}^T \mathbf{V}\|_F^2, \tag{43}$$

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 $\mathbf{R}$, $\mathbf{Y} = \operatorname{sign}(\mathbf{R}^T \mathbf{V})$. Fixing $\mathbf{B}$, the problem becomes the classic orthogonal Procrustes problem, and the solution is $\mathbf{R} = \hat{\mathbf{S}} \mathbf{S}^T$, where $\mathbf{S}$ and $\hat{\mathbf{S}}$ is obtained from the SVD of $\mathbf{Y}\mathbf{V}^T$, $\mathbf{Y}\mathbf{V}^T = \mathbf{S} \boldsymbol{\Lambda} \hat{\mathbf{S}}^T$.

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: $\bar{\mathbf{y}} = [\mathbf{y}^T 0 ... 0]^T$. The integrated objective function is written as follows:

$$\min \|\bar{\mathbf{Y}} - \bar{\mathbf{R}}^T \mathbf{X}\|_F^2, \tag{44}$$

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 seen that, the solutions for $\mathbf{y}$ of the two problems in 44 and 43 are the same, and $\bar{\mathbf{R}} = \bar{\mathbf{P}} \operatorname{Diag}(\mathbf{R}, \mathbf{I}_{(d-M)\times(d-M)})$.

7.1.2 Extensions and Variants

Harmonious hashing [110] modifies iterative quantization by adding an extra constraint: $\mathbf{Y}\mathbf{Y}^T = \sigma \mathbf{I}$. The problem is solved by relaxing $\mathbf{Y}$ to continuous values: fixing $\mathbf{R}$, let $\mathbf{R}^T \mathbf{V} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{V}^{\mathbf{T}}$, then $\mathbf{Y} = \sigma^{1/2} \mathbf{U} \mathbf{V}^T$; fixing $\mathbf{Y}$, $\mathbf{R} = \hat{\mathbf{S}} \mathbf{S}^T$, where $\mathbf{S}$ and $\hat{\mathbf{S}}$ is obtained from the SVD of $\mathbf{Y}\mathbf{V}^T$, $\mathbf{Y}\mathbf{V}^T = \mathbf{S} \boldsymbol{\Lambda} \hat{\mathbf{S}}^T$. The hash function is finally computed as $\mathbf{y} = \operatorname{sgn}(\mathbf{R}^T \mathbf{v})$.

Isotropic hashing [43] finds a rotation following a PCA preprocessing such that $\mathbf{R}^T \mathbf{V} \mathbf{V}^T \mathbf{R} = \boldsymbol{\Sigma}$ becomes a matrix with equal diagonal values, i.e., $[\boldsymbol{\Sigma}]_{11} = [\boldsymbol{\Sigma}]_{22} = \cdots = [\boldsymbol{\Sigma}]_{MM}$. The objective function is written as $\|\mathbf{R}^T \mathbf{V} \mathbf{V}^T \mathbf{R} - \mathbf{Z}\|_F = 0$, 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 same is to make the bits in the hash codes equally contributed 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 $\bar{\mathbf{\Sigma}} = \bar{\mathbf{R}}^T\mathbf{X}\mathbf{X}^T\bar{\mathbf{R}}$ are equal, and their sum is as large as possible, which is formally written as follows,

$$\max \quad \sum_{m=1}^{M} [\bar{\mathbf{\Sigma}}]_{mm} \tag{45}$$

$$\text{s.t.} \quad [\bar{\mathbf{\Sigma}}]_{mm} = \sigma, m = 1, \cdots, M, \tag{46}$$

$$\bar{\mathbf{R}}^T\bar{\mathbf{R}} = \mathbf{I}. \tag{47}$$

Locally linear hashing [30] replaces the PCA preprocess in iterative quantization using locally linear embedding. In particular, the locally linear embedding and the rotation matrix are jointly optimized. The objective function is given as $\min_{\mathbf{Z},\mathbf{R},\mathbf{Y}} \operatorname{trace}(\mathbf{Z}^T\mathbf{M}\mathbf{Z}) + \eta\|\mathbf{Y} - \mathbf{Z}\mathbf{R}\|_F^2$ subject to $\mathbf{Y} \in \{1, -1\}^{N\times M}, \mathbf{R}^T\mathbf{R} = \mathbf{I}$. Here $\mathbf{Z}$ is a nonlinear embedding similar to locally linear embedding and $\mathbf{M}$ is a sparse matrix, $\mathbf{M} = (\mathbf{I}-\mathbf{W})^T(\mathbf{I}-\mathbf{W})$. $\mathbf{W}$ is the locally linear reconstruction weight matrix. The hash function for an out-of-sample $\mathbf{q}$ is $\mathbf{y}_q = \operatorname{sign}(\bar{\mathbf{Y}}^T\mathbf{w}_q)$, where $\mathbf{w}_q$ is a locally linear reconstruction weight, and $\bar{\mathbf{Y}}$ corresponds to the hash codes of the cluster centers, computed using k-means, of the database $\mathbf{X}$. *Locality preserving hashing* [120] jointly optimizes the locality preserving projection and the quantization in the projection space: $\operatorname{trace}\{\mathbf{W}^T\mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{W}\} + \rho\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2$ with $\mathbf{L}$ being the graph Laplacian matrix for matrix.

Angular quantization [19], a variant iterative quantization, addresses the ANN search problem under the cosine similarity. The objective function of finding the binary codes, similar to iterative quantization, is $\max_{\mathbf{R},\{\mathbf{y}_n\}} \sum_{n=1}^{N} \frac{\mathbf{y}_n^T}{\|\mathbf{y}_n\|_2} \frac{\mathbf{R}^T\mathbf{x}_n}{\|\mathbf{R}^T\mathbf{x}_n\|_2}$ subject to $\mathbf{y}_n \in \{0, 1\}^M$ and $\mathbf{R}^T\mathbf{R} = \mathbf{I}$. The hash code is computed by using the nearest vertex from the vertices of the binary hypercube $\{0, 1\}^d$ to approximate the data vector $\mathbf{x}$, $\arg\max_{\mathbf{y}} \frac{\mathbf{y}^T\mathbf{x}}{\|\mathbf{y}\|_2}$, subject to $\mathbf{y} \in \{0, 1\}^M$, which is shown to be solved in $O(M \log M)$ time. The similarity in the Hamming space is computed by $\frac{\mathbf{y}_x^T\mathbf{y}_q}{\|\mathbf{y}_q\|_2\|\mathbf{y}_x\|_2}$.

Matrix Hashing [18] aims to hash a matrix $\mathbf{X}$ to short codes using a bilinear projection algorithm. The (compound) hash function is defined as $\operatorname{vec}(\operatorname{sign}(\mathbf{R}_l^T\mathbf{X}\mathbf{R}_r))$, where $\mathbf{X}$ is a matrix of $d_l \times d_r$, $\mathbf{R}_l$ of size $d_l \times d_l$ and $\mathbf{R}_r$ of size $d_r \times d_r$ are two orthogonal matrices. The objective is to minimize the angle between the rotated feature $\operatorname{vec}(\mathbf{R}_l^T\mathbf{X}\mathbf{R}_r)$ and its binary encoding $\mathbf{B} \in \{-1, +1\}^{d_l\times d_r}$: $\max_{\mathbf{R}_l,\mathbf{R}_r,\{\mathbf{B}_n\}} \sum_{n=1}^{N} \operatorname{trace}(\mathbf{B}_n\mathbf{R}_r^T\mathbf{X}_n^T\mathbf{R}_l)$. where $\mathbf{B}_n = \operatorname{sign}(\mathbf{R}_l^T\mathbf{X}_n\mathbf{R}_r)$, The problem is optimized by alternating between $\{\mathbf{B}_n\}$, $\mathbf{R}_l$ and $\mathbf{R}_r$. One comment is that $\max_{\mathbf{R}_l,\mathbf{R}_r,\{\mathbf{B}_n\}} \sum_{n=1}^{N} \operatorname{trace}(\mathbf{B}_n\mathbf{R}_r^T\mathbf{X}_n^T\mathbf{R}_l)$ is equivalent to $\min_{\mathbf{R}_l,\mathbf{R}_r,\{\mathbf{B}_n\}} \sum_{n=1}^{N} \|\mathbf{B}_n - \mathbf{R}_r^T\mathbf{X}_n^T\mathbf{R}_l\|_F$, which is also a quantization form like iterative quantization.

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 (1) that 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) that 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 $\{\operatorname{dist}(\mathbf{q}, \mathbf{c}_{1i_1}), \cdots, \operatorname{dist}(\mathbf{q}, \mathbf{c}_{Pi_P})\}$ 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) that 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 [32] 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 $[\mathbf{c}_{1i_1}^T\mathbf{c}_{2i_2}^T \cdots \mathbf{c}_{Pi_P}^T]^T$. 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 $\log_2 K$. The distance $\operatorname{dist}(\mathbf{q}, \mathbf{c}_{pi_p})$ 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 $\mathbf{q}$ corresponding to the pth subspace.

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

$$\min_{\mathbf{C},\{\mathbf{b}_n\}} \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2. \tag{48}$$

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

$$\mathbf{C} = \operatorname{diag}(\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_P) = \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{C}_P \end{bmatrix}, \tag{49}$$

where $\mathbf{C}_p = [\mathbf{c}_{p1}\mathbf{c}_{p2} \cdots \mathbf{c}_{pK}]$. $\mathbf{b}_n = [\mathbf{b}_{n1}^T\mathbf{b}_{n2}^T \cdots \mathbf{b}_{nP}^T]^T$ 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* [27] 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 given as: 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 distances of each partition to the cluster center are also recorded associated with the partition index.

Cartesian k-means [75] and *optimized production quantization* [16] extends product quantization and introduces a rotation $\mathbf{R}$ into the objective function,

$$\min_{\mathbf{R},\mathbf{C},\{\mathbf{b}_n\}} \sum_{n=1}^{N} \|\mathbf{R}^T\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2. \tag{50}$$

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* [42] 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 [117] 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 item formed from $(\mathbf{c}_{1i_1}, \mathbf{c}_{2i_2}, \cdots, \mathbf{c}_{Pi_P})$ from the distances $\{\operatorname{dist}(\mathbf{q}, \mathbf{c}_{1i_1}), \cdots, \operatorname{dist}(\mathbf{q}, \mathbf{c}_{1i_1})\}$, 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

$$\begin{aligned} \min_{\{\mathbf{C}_p\},\{\mathbf{b}_n\},\epsilon} \quad & \textstyle\sum_{n=1}^{N} \|\mathbf{x}_n - [\mathbf{C}_1\mathbf{C}_2\cdots\mathbf{C}_P]\mathbf{b}_n\|_2^2 \\ \text{s. t.} \quad & \textstyle\sum_{j=1}^{P}\sum_{i=1,i\neq j}^{P} \mathbf{b}_{ni}^T\mathbf{C}_i^T\mathbf{C}_j\mathbf{b}_{nj} = \epsilon, \\ & \mathbf{b}_n = [\mathbf{b}_{n1}^T\mathbf{b}_{n2}^T\cdots\mathbf{b}_{nP}^T]^T, \\ & \mathbf{b}_{np} \in \{0,1\}^K,\ \|\mathbf{b}_{np}\|_1 = 1, \\ & n = 1, 2, \cdots, N; p = 1, 2, \cdots P. \end{aligned} \tag{51}$$

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 [118] improves composite quantization by constructing a sparse dictionaries, $\sum_{p=1}^{P}\sum_{k=1}^{K} \|\mathbf{c}_{pk}\|_0 \leqslant S$, with S being a parameter controlling the sparsity degree, with a great reduction of the distance table computation cost and can take almost the same to the most efficient approach: product quantization.

Connection with product quantization: It is shown in [117] that both product quantization and Cartesian k-means can be regarded as constrained versions of composite quantization. Composite quantization attains smaller quantization errors, yielding a 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 small.

Composite quantization, product quantization, Cartesian k-means (optimized product quantization) can be explained from the perspective of sparse coding, as pointed in [117]: 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: 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 [23], and recently re-developed for similarity search under the Euclidean distance (additive quantization [1], [101], and tree quantization [2] modifying additive quantization, by introducing a tree-structure sparsity) and inner product [12].

7.2.3 Variants

The work in [22] 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

$$\bar{g}_{kb} = \mathrm{E}(g_k(\mathbf{x}) \mid b_k(g_k(\mathbf{x})) = b), \tag{52}$$

where $b = 0$ and $b = 1$. The 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 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* [100] directly estimates the distance table rather than computing it from the estimated dictionary.

Composite quantization [117] points the relation between Cartesian quantization and sparse coding. This indicates the application of sparse coding to similarity search. *Compact sparse coding* [8], the extension of the early work robust sparse coding [9], 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 approximate distances between the database items and the query. *Anti-sparse coding* [34] aims to learn a hash code so that *non-zero elements in the hash code are as many as possible*.

Quantization can be viewed as a reconstruction approach for a data item. *Semantic hashing* [83], [84] generates the hash codes using the deep generative model, for reconstructing the data item. As a result, the binary codes are used for finding similar data.

8 Other Topics

In this section, we review the works that handle other issues besides designing the loss function for hash function optimization.

8.1 Active and Online Hashing

Most hashing learning algorithms assume the similarity information in the input space, especially the semantic similarity information, and the database items have already been given. There are some approaches learning hash functions without such assumptions.

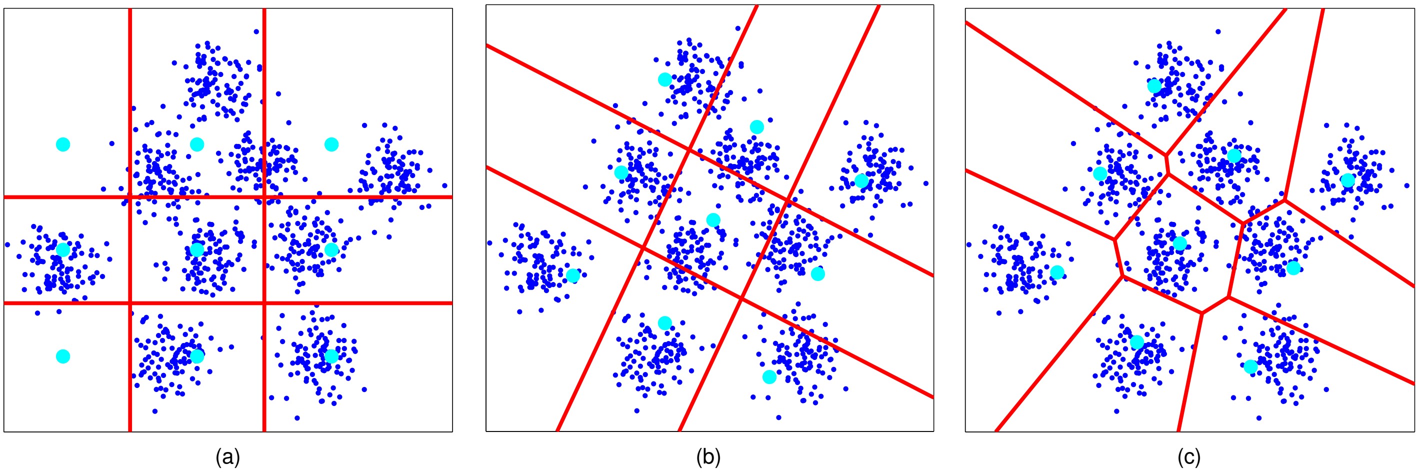


Fig. 2. 2D toy examples illustrating the quantization algorithms. The space partitioning results are generated by (a) product quantization, (b) Cartesian k-means, and (c) composite quantization. The space partition from composition quantization is more flexible.

Active hashing [121] starts with a small set of pairs of points with labeling information and learns hash functions by actively selecting the labeled pairs that are most informative. *Online hashing* [28] presents an algorithm to learn the hash functions when the similar/dissimilar pairs come sequentially rather than at the beginning all the similar/dissimilar pairs come together. *Smart hashing* [113] also addresses the problem when the similar/dissimilar pairs come sequentially. Unlike the online hash algorithm that updates all hash functions, smart hashing only selects a small subset of hash functions for relearning for a fast response to newly-coming labeled pairs.

8.2 Manifold Hashing

The manifold structure in the database is exploited for hashing, which is helpful for semantic similarity search. *Locally linear hashing* [30] combines the manifold learning approach, linearly linear embedding and iterative quantization. *Spline regression hashing* [65] is aimed at discovering a global hash function in a kernel form, such that the hash value from the global hash function is consistent to those from the local hash functions that correspond to its neighborhood points. *Inductive manifold hashing* [87] clusters the data items into K clusters, whose centers are $\{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K\}$, embeds the cluster centers into a low-dimensional space, $\{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_K\}$, using existing manifold embedding technologies, and finally the hash function is computed $\mathbf{h}(\mathbf{x}) = \operatorname{sign}(\frac{\sum_{k=1}^{K} w(\mathbf{x},\mathbf{c}_k)\mathbf{y}_k}{\sum_{k=1}^{K} w(\mathbf{x},\mathbf{c}_k)})$ with $w(\mathbf{x}, \mathbf{c}_k)$ being the similarity between $\mathbf{x}$ and $\mathbf{c}_k$.

8.3 Multi-Table Hashing

Complementary hashing [111] aims to learn multiple hash tables such that nearest neighbors have a large probability to appear in the same bucket at least in one hash table. The algorithm learns the hashing functions in a sequential manner for the multiple hash tables. The compound hash function for the first table is learnt by solving a similar problem in [96]

Reciprocal hash tables [63] extend complementary hashing by building a graph over a pool of B hash functions (with the output being a binary value) and searching the best hash functions over such a graph for building a hashing table, with updating the graph weight using a boosting-style algorithm and finding the subsequent hash tables.

8.4 Online Search

Most hashing algorithms focus on the problems in the offline training stage. In this section, we briefly introduce a few works on studying the online search stage: query-dependent distance and index structure for hash codes.

8.4.1 Query-Dependent Distance

In contrast to multi-dimensional spectral hashing in which the weights for the weighted Hamming distance are the same for arbitrary quires, the query-dependent distance approaches learn a distance measure whose weight or parameters depend on a specific query.

Query adaptive hashing [57] aims to select the hash bits (thus hash functions forming the hash bits) according to the query vector. The approach in the online stage selects a few hash functions from the offline computed hash functions $\mathbf{h}(\mathbf{x}) = \operatorname{sgn}(\mathbf{W}^T\mathbf{x})$ by solving the following,

$$\min_{\boldsymbol{\alpha}} \|\mathbf{q} - \mathbf{W}\boldsymbol{\alpha}\|_2^2 + \rho\|\boldsymbol{\alpha}\|_1. \tag{53}$$

Given the optimal solution $\boldsymbol{\alpha}^*$, $\alpha_i^* = 0$ means the ith hash function is not selected, and the hash functions corresponding to the nonzero entries in $\boldsymbol{\alpha}^*$ are selected.

Query-adaptive class-specific bit weights [38], [39] presents a weighted Hamming distance measure by learning the weights from the query information. Specifically, the approach learns class-specific bit weights so that the weighted Hamming distance between the hash code belonging to a class and the hash code belonging to that class's center (the mean of those hash codes) is minimized.

Bits reconfiguration [70] is to learn a good distance measure over the hash codes precomputed from a pool of hash functions: $\|\mathbf{W}^T(\mathbf{y}_i - \mathbf{y}_j)\|_2^2$ with $\mathbf{W}$ being a transformation matrix.

8.4.2 Fast Search in the Hamming Space

The computation of the Hamming distance is shown much faster than that of the distance in the input space. It is still expensive 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.

Multi-index hashing [77] aims 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 [73] extends the FLANN algorithm [72] that is 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.

8.4.3 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 following reranking step using the true distance computed in the coding 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 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 fast search algorithm in the Hamming space, e.g., [73], [77] can be used to fast retrieve the hash buckets.

Thanks to the multi-sequence algorithm, the Cartesian quantization algorithms are also applied to inverted index [3], [118], [16] (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 [3] 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 [103]) 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 often is 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.

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	$1B$	$100M/1M$	$10K$

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 depends on the hashing algorithms and can be viewed as a constant. When comparing the performance in the case the Hamming distances 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 after reordering the R retrieved nearest items using the original features and return the top T items. In the case 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.

9.2 Evaluation Datasets

The widely-used evaluation datasets are with different scales from small, large, and very large. Various features have been used, such as SIFT features [66] extracted from Photo-tourism [89] and Caltech 101 [14], GIST features [79] from LabelMe [82] and Peekaboom [95], as well as some features used in object retrieval: Fisher vectors [81] and VLAD vectors [33]. The following presents a brief introduction to several representative datasets, which is summarized in Table 2.

MNIST [46] includes $60K$ $784D$ raw pixel features describing grayscale images of handwritten digits as a reference set, and $10K$ features as the queries.

SIFT$10K$ [32] consists of $10K$ 128-dimensional SIFT vectors as the reference set, $25K$ vectors as the learning set, and 100 vectors as the query set. SIFT$1M$ [32] 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 SIFT$10K$ and SIFT$1M$ are extracted from Flicker images and the reference sets and the query sets are from the INRIA holidays images [31].

GIST$1M$ [32] 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 [93]. The reference set is from the Holiday images combined with Flickr$1M$ [31]. The query set is from the Holiday image queries. Tiny$1M$ [104][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.

SIFT$1B$ [35] includes $1B$ 128-dimensional BYTE-valued SIFT vectors as the reference set, $100M$ vectors as the learning set and $10K$ vectors as the query set. The three sets are extracted from around $1M$ images. This dataset, and SIFT$10K$, SIFT$1M$ and GIST$1M$ are publicly available[2].

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 that the query is transformed to a hash code, e.g., adopting the Hamming distance for most binary hash algorithms, learning over the whole reference set might be over-fitting and the performance might be worse than learning with a subset of the reference set or a separate set. In the case that the raw query is used without any processing, e.g., 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, minimal loss hashing [74] and composite quantization [117]. It is unfair and not suggested to select the hyper-parameters corresponding to the best performance over the query set. It is suggested 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.

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

10 Performance Analysis

10.1 Query Performance

In this section, we present the empirical observation and some analysis validating the observations. We discuss about both hash table lookup and hash code ranking, with more focus on hash code ranking as it is more widely studied and practically adopted. The analysis is mainly interested in the major application of hashing: nearest neighbor search with the Euclidean distance. The conclusion for semantic similarity search in principle is similar and the performance also depends on the ability of representing the semantic meaning of the input features. In addition, we present empirical results of the quantization algorithms, which are shown better than binary code hashing, for hash code ranking and its application to the very large scale dataset SIFT$1B$.

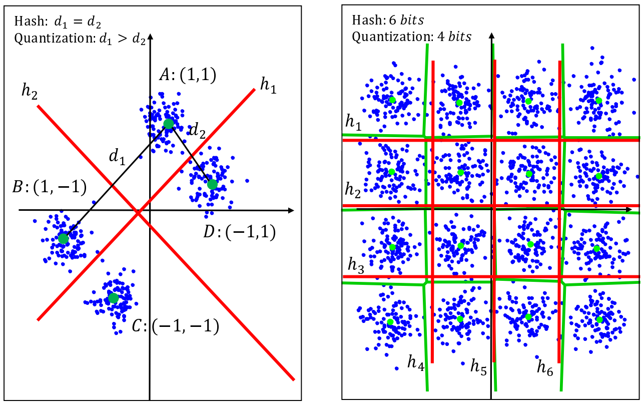
10.1.1 Query Performance with Hash Table Lookup

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

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 it needs to store multiple copies of reference vector indices. As an alternative choice, single-assignment with a single table can be used and 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. 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. In general, the performance for other algorithms based on weighted Hamming distance and learnt distance is in between. The

[1] http://research.microsoft.com/~jingdw/SimilarImageSearch/NNData/NNdatasets.html
[2] http://corpus-texmex.irisa.fr/



(a) (b)

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 requires 4 $(= \log 16)$ bits (green lines show the partition line).

observations holds for a single table with single assignment or multiple assignment, and multiple tables.

10.1.2 Query Performance with Hash Code Ranking

In contrast to hash table lookup, hash code ranking is more widely adopted in the research area of learning-to-hash and the real search systems. 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.

In terms of 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 pairwise (multiwise) similarity preserving algorithm performs consistently the best though there are a large amount of pairwise (multiwise) similarity preserving algorithms. It is shown that the cost function of hypercubic quantization is an approximation of the distance-distance difference. But it outperforms pairwise and multiwise similarity preserving. This is because it is infeasible to consider all pairs (all triples) of items for the distance-distance difference in pairwise (multiwise) similarity preserving, and thus only a small subset of the pairs (triples), by sampling a subset of items or pairs, are considered, while the cost function for quantization is an approximation for all pairs of items. In general, most binary code hashing algorithms can benefit from the kernel hash function, and weighted Hamming distances as well as learnt distances for binary codes, which increase the search accuracy and also the search cost.

Compared with binary code hashing including hypercubic quantization, another reason for the superiority of Cartesian quantization is that there are only a small number of $(\theta(L))$ distinct Hamming distances in the coding space for hypercubic quantization 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 [22] or directly learning a distance lookup table [100] 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.

TABLE 4
A summary of performance comparison with quantization algorithms.

	Query			Training
	Accuracy	Efficiency	Overall	Efficiency
ITQ	low	high	low	high
PQ	fair	fair	low	high
CKM (OPQ)	fair	fair	fair	fair
CQ	high	fair	high	low

In terms of search cost, the evaluation of the Hamming distance using the function $\_\_\text{popcnt}$ 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 sub-table corresponds to a byte and there are totally $\frac{L}{8}$ sub-tables. It is worthy pointing that the Cartesian quantization approaches relying on the distance table lookup achieve still 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 a 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, 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 lack of such a capability due to the Hamming distance. Figure 3 (b) shows that the binary hash coding algorithms requires more (6) hash bits to differentiate the 16 uniformly-distributed clusters while the quantization algorithms only requires 4 $(= \log 16)$ bits.

10.1.3 Empirical Results

We present the empirical results of the six representative quantization algorithms: iterative quantization [20] with the Hamming distance (ITQH) and with the asymmetric distance [22] (ITQA), product quantization [32] (PQ), Cartesian k-means [75] (optimized product quantization [16]) (CKM), composite quantization [117] (CQ), and sparse composite quantization [118] (SQ), over two representative datasets: MNIST [46] and SIFT1M [32]. The results are collected from the two recent papers [117], [118]. All the algorithms learn the hash functions and the codes over the reference sets. We only show the results for search the nearest neighbor

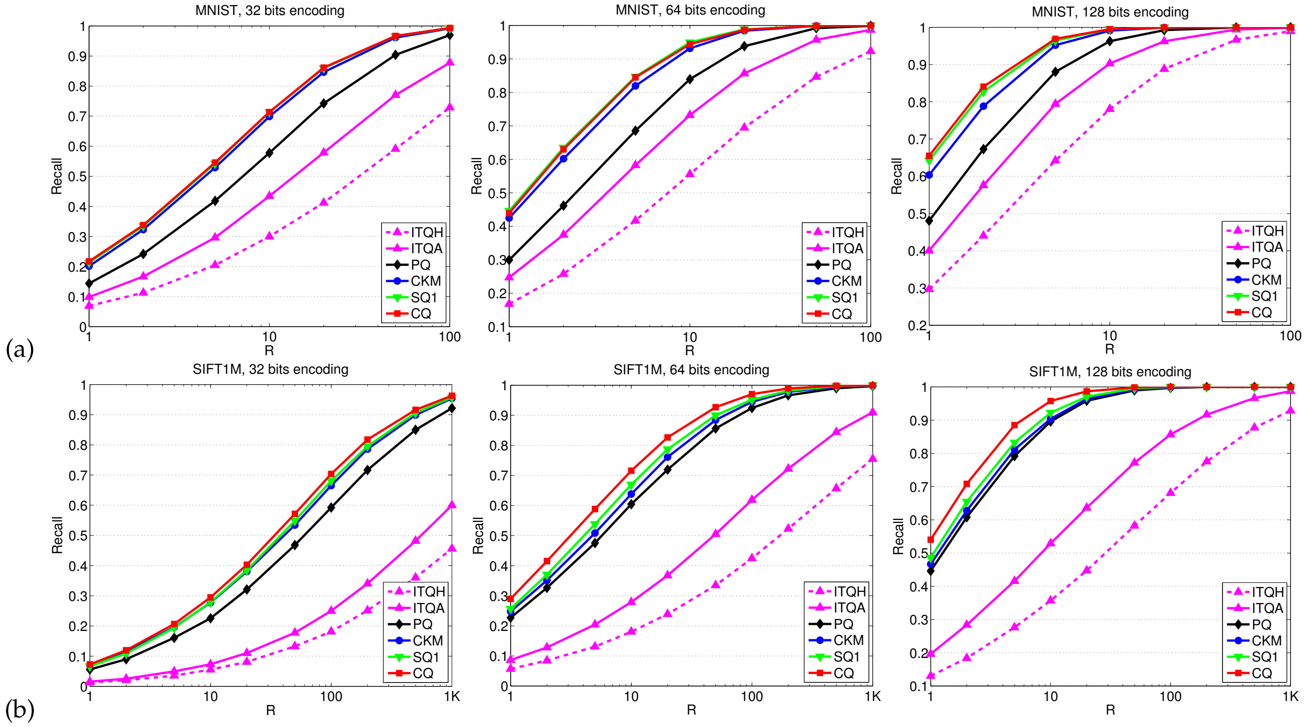


Fig. 4. The performance in terms of recall@R over MNIST and SIFT$1M$ for the representative quantization algorithms. ITQH = iterative quantization with the Hamming distance [20], ITQA = iterative quantization with asymmetric distance [22], PQ = product quantization [32], CKM = Cartesian k-means [75], CQ = composite quantization [117], SQ1 = sparse composite quantization [118] whose dictionary is the same sparse with PQ.

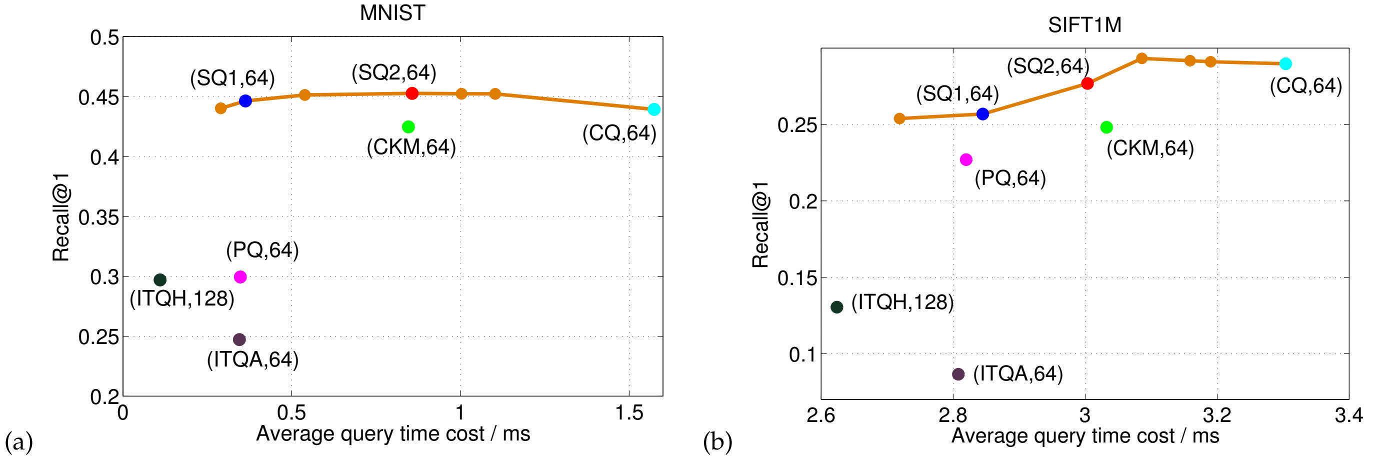


Fig. 5. The performance in terms of recall and search time. (a,b) = (method, #bits). The brown curve is plotted by varying the sparsity of the dictionary in sparse composite quantization (SQ). (a) shows the performance over a small dataset, MNIST. The time cost of ITQH with 128 bits is the smallest, but the recall performance is not comparable to other algorithms with 64 bits. (b) shows the performance over a large dataset, SIFT$1M$. The time cost with 128 bits for ITQH is similar to PQ and SQ with higher sparsity, but the recall is much lower.

($K = 1$) and the conclusion holds for searching more nearest neighbors ($K > 1$).

The recall@R curves are shown in Figure 4. We have several observations. (1) The performance of ITQH (ITQ with the Hamming distance) is the worst. This validates the above performance analysis for binary code hashing and quantization-based hashing: quantization is superior over binary code hashing. (2) The performance of ITQA (ITQ with the asymmetric distance) is better than ITQH, but the second worst. This provides the evidence for the analysis: the performance of binary code hashing is improved with the learnt Euclidean-like asymmetric distance, and separating hash code computation and the distance learning in ITQA is inferior compared with the joint manner as done in other quantization algorithms. (3) The performances of CQ and SQ1 (in which the dictionary of CQ is the same sparse with PQ) are the best, which indicates the empirical results are consistent to the analysis and the $2D$ illustration shown in Figure 2.

We show the results in terms of recall and search time in Figure 5. The computation of distance lookup table might be expensive when the size of the reference set is very small. Figure 5 (a) shows the performance over the MNIST that contains only $60K$ reference items. There are a few observations: (1) SQ, PQ, CKM, and CQ outperform ITQH and ITQA in terms of recall. (2) The time cost with 64 bits for other algorithms is higher than ITQH with 128 bits. This is because the computation of distance lookup table might be expensive when the size of the reference set is very small though the time cost of linear scan using distance lookup table for 64 bits is very close to that using fast hamming distance evaluation for 128 bits. In contrast, the recalls of SQ, PQ, CKM, and CQ with 64 bits are much better than ITQH with 128 bits, and the time costs for SQ1 and PQ are very similar to ITQH. The reason is that the extra cost of distance table computation is negligible when handling $1M$ reference items.

In addition, we report the results over SIFT1B (BIGANN) [35]. We follow the inverted multi-index algorithm [3]: use a coarse quantizer to build the inverted index for fast retrieving candidates, and the Multi-D-ADC search strategy: a fine quantizer to generate the hash codes for the residual displacement between each vector and its closest cell centroids obtained through the indexing stage for efficiently reranking. More details can be found from [3], [118]. The results are collected from [118] and do not include the results of ITQH and ITQA as their performances are very poor. There are two results for sparse composite quantization: SQ1 in which the dictionary is the same sparse with PQ and SQ2 in which the sparsity of the dictionary is equivalent to CKM. All the algorithms are trained over the first 1M items of the reference set. The results are shown in Table 5. The observations include: (1) The recall performance for CQ is the best, and the search time cost is the largest due to the expensive distance table computation, but the time cost becomes similar to other algorithms when retrieving more (L) candidates; (2) SQ1 (SQ2) performs better than PQ (CKM), higher recall and almost the same search time.

TABLE 5
Comparison of the Multi-D-ADC system with different quantization algorithms in terms of recall@R with R being $1, 10, 100$, time cost (in millisecond) with database vector reconstruction ($T1$), time cost (in millisecond) without database vector reconstruction but through distance lookup tables ($T2$). L is the length of the candidate list reranked by the system. The results are collected from [118].

Alg.	L	R@1	R@10	R@100	$T1$	$T2$
BIGANN, 1 billion SIFTs, 64 bits per vector						
PQ	10000	0.158	0.479	0.713	6.2	4.1
CKM		0.181	0.525	0.751	11.9	4.6
CQ		0.195	0.558	0.765	15.7	7.1
SQ1		0.184	0.530	0.736	7.3	4.3
SQ2		0.191	0.546	0.754	8.6	4.5
PQ	30000	0.172	0.507	0.814	13.2	9.8
CKM		0.193	0.556	0.851	30.3	10.1
CQ		0.200	0.597	0.869	42.6	12.9
SQ1		0.192	0.571	0. 849	15.8	9.9
SQ2		0.198	0.586	0.860	19.9	10.0
PQ	100000	0.173	0.517	0.862	37.4	30.5
CKM		0.195	0.568	0.892	95.8	31.6
CQ		0.204	0.612	0.920	125.9	33.4
SQ1		0.194	0.584	0.903	43.7	30.9
SQ2		0.199	0.597	0.907	58.6	31.2
BIGANN, 1 billion SIFTs, 128 bits per vector						
PQ	10000	0.312	0.673	0.739	7.0	5.5
CKM		0.357	0.718	0.772	12.4	5.8
CQ		0.379	0.738	0.781	29.0	7.9
SQ1		0.347	0.702	0.755	8.2	5.6
SQ2		0.368	0.725	0.773	9.5	5.7
PQ	30000	0.337	0.765	0.883	15.8	14.1
CKM		0.380	0.811	0.903	32.7	14.4
CQ		0.404	0.833	0.906	76.4	16.8
SQ1		0. 372	0.802	0.890	18.9	14.3
SQ2		0.392	0.821	0.904	25.8	14.4
PQ	100000	0.345	0.809	0.964	48.7	43.3
CKM		0.389	0.848	0.970	107.6	44.9
CQ		0.413	0.877	0.975	242.3	47.3
SQ1		0.381	0.852	0.969	59.3	43.6
SQ2		0.401	0.858	0.971	77.4	43.9

10.2 Training Time Cost

We present the analysis of 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 ($O(NM + Nd)$), 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 triple of items, and in general the training 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. It indicates that it takes larger offline computation cost to get a higher (online) search performance. The comparison of the query performances and the training cost of various quantization algorithms is summarized in Table 4. In comparison to binary code hashing, the quantization category is in theory cheaper and both the categories can benefit from sampling a subset of items.

11 Future Trends

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 handing high-dimensional databases. There are also increasing interests in other topics, such as multi-modality and cross-modality hashing and semantic quantization. Recent deep learning developments also indicate an emerging topic, learning an end-to-end hashing system without a separate intermediary feature extraction step.

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 is shown that the search accuracy is increased with a high sampling rate, but the training cost is greatly increased. The algorithms even without relying pairwise similarity, e.g., quantization, are 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 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 [114], accelerates the encoding process for the linear hash functions, and tree-quantization [2] sparsifies the dictionary items into a tree structure, to speeding up the assignment process. It expects more research study 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 inverted index. This is a research direction that will attract research interests, such as a recent study, sparse composite quantization [118].

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 interests.

Multiple and Cross Modality Hashing. One important characteristic of big data is the variety of data types and data sources. This is particularly true to 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-model search. This topic is attracting a lot of research efforts, such as collaborative hashing [62], and cross-media hashing [90], [91], [123].

Joint Feature and Hash Learning. Almost all existing algorithms assume the features are already given for learning hash functions. It would be an interesting trend that the features and the hash functions are jointly learnt. There are a few works [109] [119] very recently. We believe that an end-to-end hashing learning system will become hot.

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 and present a comprehensive survey, and present their relations. We point out the empirical observation that quantization is superior in terms of search accuracy, search efficiency and space cost, and the future research directions.

References

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

[2] A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *CVPR*, 2015. 12, 19

[3] A. Babenko and V. S. Lempitsky. The inverted multi-index. In *CVPR*, pages 3069–3076, 2012. 14, 18

[4] J. Brandt. Transform coding for fast approximate nearest neighbor search in high dimensions. In *CVPR*, pages 1815–1822, 2010. 6

[5] 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

[6] 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

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

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

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

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

[11] 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

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

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

[14] 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. 15

[15] 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

[16] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2946–2953, 2013. 5, 12, 14, 16

[17] T. Ge, K. He, and J. Sun. Graph cuts for supervised binary coding. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII*, pages 250–264, 2014. 4, 7

[18] 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. 5, 11

[19] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik. Angular quantization-based binary codes for fast similarity search. In *NIPS*, pages 1205–1213, 2012. 5, 11

[20] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011. 5, 6, 10, 16, 17

[21] 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. 5, 6, 10

[22] 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. 12, 16, 17

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

[24] 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. 5, 6

[25] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010. 5, 6

[26] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *CVPR*, pages 2957–2964, 2012. 9

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

[28] L.-K. Huang, Q. Yang, and W.-S. Zheng. Online hashing. In *IJCAI*, 2013. 13

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

[30] 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

[31] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, pages 304–317, 2008. 15

[32] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011. 5, 11, 15, 16, 17

[33] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311, 2010. 1, 15

[34] H. Jégou, T. Furon, and J.-J. Fuchs. Anti-sparse coding for approximate nearest neighbor search. In *ICASSP*, pages 2029–2032, 2012. 12

[35] 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. 15, 18

[36] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-max hash for jaccard similarity. In *ICDM*, pages 301–309, 2013. 1

[37] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *NIPS*, pages 108–116, 2012. 1

[38] 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

[39] 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

[40] Z. Jin, Y. Hu, Y. Lin, D. Zhang, S. Lin, D. Cai, and X. Li. Complementary projection hashing. In *ICCV*, pages 257–264, 2013. 9

[41] A. Joly and O. Buisson. Random maximum margin hashing. In *CVPR*, pages 873–880, 2011. 9

[42] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2329–2336, 2014. 12

[43] W. Kong and W.-J. Li. Isotropic hashing. In *NIPS*, pages 1655–1663, 2012. 5, 6, 10

[44] 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

[45] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009. 5, 8

[46] 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. 15, 16

[47] 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

[48] P. Li, T. Hastie, and K. W. Church. Very sparse random projections. In *KDD*, pages 287–296, 2006. 1

[49] P. Li and A. C. König. b-bit minwise hashing. In *WWW*, pages 671–680, 2010. 1

[50] P. Li, A. C. König, and W. Gui. b-bit minwise hashing for estimating three-way similarities. In *NIPS*, pages 1387–1395, 2010. 1

[51] P. Li, A. B. Owen, and C.-H. Zhang. One permutation hashing. In *NIPS*, pages 3122–3130, 2012. 1

[52] 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. 6

[53] 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

[54] 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

[55] R.-S. Lin, D. A. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010. 5, 8

[56] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li. Compressed hashing. In *CVPR*, pages 446–451, 2013. 5, 6

[57] 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

[58] W. Liu, C. Mu, S. Kumar, and S. Chang. Discrete graph hashing. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3419–3427, 2014. 5, 6

[59] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012. 5, 8

[60] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011. 5, 6

[61] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. In *ICML*, 2012. 5, 8

[62] X. Liu, J. He, C. Deng, and B. Lang. Collaborative hashing. In *CVPR*, pages 2147–2154, 2014. 19

[63] X. Liu, J. He, and B. Lang. Reciprocal hash tables for nearest neighbor search. In *AAAI*, 2013. 13

[64] 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. 6

[65] 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

[66] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 15

[67] 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

[68] Y. Matsushita and T. Wada. Principal component hashing: An accelerated approximate nearest neighbor search. In *PSIVT*, pages 374–385, 2009. 6

[69] R. Motwani, A. Naor, and R. Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007. 1

[70] 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

[71] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344–3351, 2010. 5, 8

[72] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP (1)*, pages 331–340, 2009. 14

[73] M. Muja and D. G. Lowe. Fast matching of binary features. In *CRV*, pages 404–410, 2012. 3, 14

[74] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011. 5, 7, 9, 15

[75] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, pages 3017–3024, 2013. 5, 12, 16, 17

[76] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, pages 1070–1078, 2012. 5, 9

[77] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, pages 3108–3115, 2012. 14

[78] 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

[79] 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. 15

[80] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006. 1

[81] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pages 3384–3391, 2010. 15

[82] 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. 15

[83] R. Salakhutdinov and G. E. Hinton. Semantic hashing. In *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007. 12

[84] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009. 12

[85] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, pages 1665–1672, 2011. 1

[86] J. Shao, F. Wu, C. Ouyang, and X. Zhang. Sparse spectral hashing. *Pattern Recognition Letters*, 33(3):271–277, 2012. 6

[87] F. Shen, C. Shen, Q. Shi, A. van den Hengel, and Z. Tang. Inductive hashing on manifolds. In *CVPR*, pages 1562–1569, 2013. 13

[88] A. Shrivastava and P. Li. Densifying one permutation hashing via rotation for fast near neighbor. In *ICML (1)*, page 557565, 2014. 1

[89] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, 2006. 15

[90] 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. 19

[91] 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. 19

[92] 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. 5, 7

[93] 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. 15

[94] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *CVPR*, pages 2320–2327, 2012. 1

[95] L. von Ahn, R. Liu, and M. Blum. Peekaboom: a game for locating objects in images. In *CHI*, pages 55–64, 2006. 15

[96] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010. 5, 7, 13

[97] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, pages 1127–1134, 2010. 5, 7

[98] 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. 5, 7

[99] J. Wang, W. Liu, A. X. Sun, and Y.-G. Jiang. Learning hash codes with listwise supervision. In *ICCV*, pages 3032–3039, 2013. 5, 9

[100] J. Wang, H. T. Shen, S. Yan, N. Yu, S. Li, and J. Wang. Optimized distances for binary code ranking. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, pages 517–526, 2014. 12, 16

[101] 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

[102] J. Wang, J. Wang, N. Yu, and S. Li. Order preserving hashing for approximate nearest neighbor search. In *ACM Multimedia*, pages 133–142, 2013. 5, 9

[103] 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. 14

[104] 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. 15

[105] Q. Wang, D. Zhang, and L. Si. Weighted hashing for fast large scale similarity search. In *CIKM*, pages 1185–1188, 2013. 6

[106] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *ECCV (5)*, pages 340–353, 2012. 5, 8

[107] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008. 1, 5, 8

[108] 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

[109] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2156–2162, 2014. 19

[110] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, and D. Cai. Harmonious hashing. In *IJCAI*, 2013. 5, 10

[111] 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

[112] 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

[113] Q. Yang, L.-K. Huang, W.-S. Zheng, and Y. Ling. Smart hashing update for fast response. In *IJCAI*, 2013. 13

[114] F. Yu, S. Kumar, Y. Gong, and S.-F. Chang. Circulant binary embedding. In *ICML (2)*, pages 946–954, 2014. 19

[115] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *SIGIR*, pages 18–25, 2010. 5, 6

[116] 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. 5, 7

[117] T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *ICML (2)*, pages 838–846, 2014. 5, 12, 15, 16, 17

[118] T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *CVPR*, 2015. 12, 14, 16, 17, 18, 19

[119] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. *CoRR*, abs/1501.06272, 2015. 19

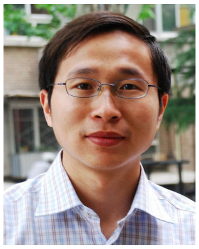
[120] K. Zhao, H. Lu, and J. Mei. Locality preserving hashing. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2874–2881, 2014. 11

[121] 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. 13

[122] 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. 6

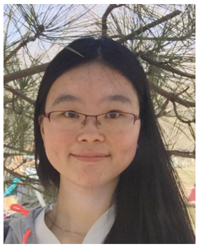
[123] X. Zhu, Z. Huang, H. T. Shen, and X. Zhao. Linear cross-modal hashing for efficient multimedia search. In *ACM Multimedia*, pages 143–152, 2013. 19

[124] 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. 6



Jingdong Wang is a Lead Researcher at the Visual Computing Group, Microsoft Research Asia. He received the M.Eng. and B.Eng. degrees in Automation from the Department of Automation, Tsinghua University, Beijing, China, in 2001 and 2004, respectively, and the PhD degree in Computer Science from the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, Hong Kong, in 2007. His areas of interest include computer vision, machine learning, pattern recognition, and multimedia computing. In particular, he has worked on kernel methods, semi-supervised learning, data clustering, image segmentation, and image and video presentation, management and search. At present, he is mainly working on the Big Media project, including large-scale indexing and clustering, Web image search and mining, and visual understanding such as salient object detection, image recognition, face alignment and recognition.

Heng Tao Shen is a Professor of Computer Science in School of Information Technology and Electrical Engineering, The University of Queensland. He obtained his B.Sc. (with 1st class Honours) and Ph.D. from Department of Computer Science, National University of Singapore in 2000 and 2004 respectively. He then joined the University of Queensland as a Lecturer and became a Professor in 2011. His research interests include Multimedia/Mobile/Web Search and Big Data Management. He is the winner of Chris Wallace Award for outstanding Research Contribution in 2010 from CORE Australasia. He is an Associate Editor of IEEE TKDE, and is a PC Co-Chair for ACM Multimedia 2015.



Ting Zhang is a PhD candidate in the department of Automation at University of Science and Technology of China. She received the Bachelar degree in mathematical science from the school of the gifted young in 2012. Her main research interests include machine learning, computer vision and pattern recognition.She is currently a research intern at Microsoft Research, Beijing.