This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

AMVH: Asymmetric Multi-Valued Hashing

Cheng Da[1, 2], Shibiao Xu[1], Kun Ding[1, 2], Gaofeng Meng[1], Shiming Xiang[1, 2], Chunhong Pan[1]
[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2]School of Computer and Control Engineering, University of Chinese Academy of Sciences
Email:{cheng.da,shibiao.xu,kding,gfmeng,smxiang,chpan}@nlpr.ia.ac.cn

Abstract

Most existing hashing methods resort to binary codes for similarity search, owing to the high efficiency of computation and storage. However, binary codes lack enough capability in similarity preservation, resulting in less desirable performance. To address this issue, we propose an asymmetric multi-valued hashing method supported by two different non-binary embeddings. (1) A real-valued embedding is used for representing the newly-coming query. (2) A multi-integer-embedding is employed for compressing the whole database, which is modeled by binary sparse representation with fixed sparsity. With these two non-binary embeddings, the similarities between data points can be preserved precisely. To perform meaningful asymmetric similarity computation for efficient semantic search, these embeddings are jointly learnt by preserving the label-based similarity. Technically, this results in a mixed integer programming problem, which is efficiently solved by alternative optimization. Extensive experiments on three multi-label datasets demonstrate that our approach not only outperforms the existing binary hashing methods in search accuracy, but also retains their query and storage efficiency.

1. Introduction

Recently, most of the massive data is represented in high-dimensional space for use [16, 32]. Measuring the similarity of these data points with high computational efficiency is a fundamental task in real-world applications. Technically, several recent works [3, 17, 29, 39] have shown that hashing methods can compress high-dimensional data points into compact binary codes and can simultaneously preserve the similarity and structural information of the original data. Thus, hashing yields a mechanism for high computational efficiency and acceptable accuracy.

In the literature, many hashing methods have been proposed [2, 7, 8, 13, 19, 22, 33, 34, 36], which can be roughly divided into two categories: data-independent and data-dependent hashing methods. Locality Sensitive Hashing (LSH) [7] is a representative of the first class that uses random projection as hash function. However, the major drawback of LSH and its variants [18, 31] is that many hash bits are required to guarantee good performance. In parallel, recent efforts mainly focus on designing data-dependent hashing methods that learn compact binary codes by exploiting the data distribution [9, 23, 24]. These methods are categorized into unsupervised [9, 15, 22, 40], semi-supervised [38, 41] and supervised methods [14, 21, 24, 27, 34].

According to the different encoding strategies of query and database, hashing methods can also be grouped into symmetric and asymmetric ones. Most of the existing hashing methods are symmetric, in which the binary codes are derived from the same hash function, and the Hamming distances between them are computed for retrieval. In comparison to symmetric hashing, some pioneering works [6, 11, 26] have theoretically proven that asymmetric hashing can attain superior accuracy with shorter codes by using two different hash functions for query and database. For example, Gordo *et al.* [11] claimed that compressing the query into the binary codes is not a strict requirement and presented two general asymmetric distances, which are applicable to many hashing methods. The key insight is that query is real-valued and database is still binary, so that the more precise information of the query is capable of facilitating better similarity search. Since the hash functions for both query and database are derived from the same real-valued embedding function, this method does not fully take the advantage of the asymmetry. Neyshabur *et al.* [26] went a step further by proposing an asymmetric binary hashing method using two distinct binary hash functions, but the binary constraints still limit the effectiveness.

In addition, two-step hashing [20, 21] has attracted broad research interests due to its simplicity, flexibility and effectiveness. It decomposes the hashing learning problem into two steps: a binary code inference step and a hash function learning step based on the learned codes. As mentioned in [5], different encoding strategies can be adopted for database and query in two-step hashing, thereby it is essentially an asymmetric hashing method. Subsequently, discrete hashing methods, such as SDH [34], FastHash [21], COSDISH [14], generalize the idea of two-step hashing by introducing some coupling between the two steps mentioned above. Clearly, they also belong to the class of asymmetric hashing. Although great success has been acquired by two-step hashing and discrete hashing, they are still suboptimal due to the binary limitation.

In this work, we aim to alleviate the binary limitation, and therefore propose a novel asymmetric hashing method, named Asymmetric Multi-Valued Hashing (AMVH). The idea of our approach is quite intuitive — using real values and multiple integer values instead of binary ones should permit the better preservation of the similarity between data points. Based on this idea, we present two different non-binary embeddings: the real-valued embedding and the multi-integer-valued embedding. The former is used to map the query into a real-valued low-dimensional space. The latter is used to compress the database points. However, multiple integer values are not conducive for building the lookup table that is commonly used for efficient search, due to a large number of combinations of these integer values. Hence, binary sparse representation is proposed to circumvent this problem, namely, a multi-integer-valued vector is represented by a product of a binary dictionary and an indicator vector. In this way, the lookup table about dictionary can be built efficiently. Meanwhile, the efficiency of computation and storage can be guaranteed.

To make the two different embeddings to be useful representations for asymmetric similarity computation in the query stage, these embeddings should be optimized to allow the meaningful comparison between them. For this reason, we propose to minimize the difference between the predictive similarity and the ground-truth one provided by the semantic labels. However, the discrete constraints of the base atoms and the sparse coefficients bring us a mixed integer programming problem, which is NP-hard [14, 33, 34]. In order to solve this problem, a well-designed alternative optimization algorithm is exploited, where each subproblem can be solved efficiently, yielding satisfactory solutions.

To sum up, the main contributions of this work are:

- A novel asymmetric hashing method supported by two different non-binary embeddings is proposed, which can alleviate the binary limitation and can remarkably improve the capability of similarity preservation.
- We introduce binary sparse representation to model the multi-integer-valued embedding, which permits the construction of the lookup table for efficient similarity search. To the best of our knowledge, the multi-integer-valued embedding is first proposed in hashing.
- A well-designed alternative optimization algorithm is proposed to efficiently solve our problem, which is scalable to deal with large-scale datasets.

2. Asymmetric Multi-Valued Hashing

2.1. Basic Formulation

This work mainly focuses on supervised hashing method which enables the attractive performance in semantic similarity search [14, 34]. Suppose that database points are represented by a set of N D-dimensional vectors, denoted by $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D\times N}$. Their associated labels are $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N] \in \{0,1\}^{C\times N}$, where C is the number of classes. When the c-th entry y_{ci} equals to 1, it means $\mathbf{x}_i$ belongs to the c-th class. Traditional hashing methods learn a binary embedding vector $\mathbf{b}_i \in \{-1,1\}^K$ for each data point, which results in a binary matrix $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_N] \in \{-1,1\}^{K\times N}$. A commonly-used objective to learn these binary codes is to preserve the Hamming affinity [14, 21, 24]. Concretely, it minimizes the difference between the predictive and ground-truth affinities, that is,

$$\begin{aligned} \min_{\mathbf{B}} \quad & \|\frac{1}{K}\mathbf{B}^T\mathbf{B} - \mathbf{S}\|_F^2 \\ \text{s.t.} \quad & \mathbf{B} \in \{-1,1\}^{K\times N}, \end{aligned} \tag{1}$$

where $\mathbf{S}$ is a pairwise similarity matrix derived by the labels, and $\|\cdot\|_F$ is the Frobenius norm. This model is first introduced in kernel-based supervised hashing (KSH) [24], and becomes a standard optimization problem for hashing learning [14, 21]. However, one major deficiency of this formulation is the symmetric binary inner product form, which is limited in approximating the real-valued similarity.

Therefore, we propose Asymmetric Multi-Valued Hashing method (AMVH) to alleviate the binary limitation, in which the traditional binary values are extended to real values and multiple integer values. In the following sections, we firstly introduce the real-valued embedding, by which the real-valued query is utilized to search. Then, the multi-integer-valued embedding is elaborated, by which the database points are represented by multiple integer values, improving the capability of similarity preservation.

2.2. Real-valued Embedding for Query

Linear binary hash function, denoted by $h(\mathbf{x}) = \text{sign}(\mathbf{W}^T\mathbf{X})$, is commonly used in traditional binary hashing methods [4, 24]. Here $\mathbf{W} \in \mathbb{R}^{D\times K}$ is a transformation matrix that projects data $\mathbf{X}$ onto a K-dimensional real-valued space ($K < D$), and $\text{sign}(\cdot)$ is the element-wise sign function. As mentioned above, directly using the real-valued information should permit more accurate approximation of similarity [11]. For this reason, the real-valued embedding is directly imposed on AMVH. Thus, replacing one of the $\mathbf{B}$s in (1) by $\mathbf{W}^T\mathbf{X}$, we obtain the following model with an asymmetric objective function:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{B}} \quad & \|(\mathbf{W}^T\mathbf{X})^T\mathbf{B} - \mathbf{S}\|_F^2 \\ \text{s.t.} \quad & \mathbf{B} \in \{-1,1\}^{K\times N}. \end{aligned} \tag{2}$$

Here the similarity between the real-valued embeddings $\mathbf{W}^T\mathbf{X}$ and the binary codes $\mathbf{B}$ can be measured by inner product. Meanwhile, the two kinds of embeddings are jointly learnt by preserving the pairwise semantic similarities. Once problem (2) is solved, the binary codes $\mathbf{B}$ of database points can be obtained, and the unseen query $\mathbf{x}$ can be encoded by the learned linear hash function $h(\mathbf{x}) = \mathbf{W}^T\mathbf{x}$.

2.3. Multi-integer-valued Embedding for Database

Although problem (2) is modeled in the asymmetric viewpoint, the database points are still represented by binary codes, which have limited capability in approximating diverse similarities. To this end, we propose the multi-integer-valued embedding, by which database points are compressed into multiple integer vectors. Consequently, the capability of similarity preservation is further improved.

However, multiple integer values are not conducive for fast search, because the lookup table can not be build efficiently for so many integer values. Therefore, binary sparse representation is proposed to model the multi-integer-valued embeddings with fixed sparsity. Technically, a multi-integer-valued vector $\hat{\mathbf{b}}_i$ is denoted by a product of a binary dictionary $\mathbf{C}$ and an indicator vector $\mathbf{a}_i$, *i.e.*, $\hat{\mathbf{b}}_i = \mathbf{C}\mathbf{a}_i$. Here $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_M] \in \{-1, 1\}^{K\times M}$ is the binary dictionary with M $(M \ll N)$ atoms and $\mathbf{a}_i \in \{0, 1\}^M$ is the sparse indicator vector, which strictly contains l 1s and indicates that only l atoms can be selected from the dictionary. $l\,(0 < l < M)$ is an important hyper-parameter, reflecting the diversity of multi-integer-valued embeddings indirectly. Based on this representation, $\hat{\mathbf{B}}$ includes at most $2\,l + 1$ kinds of integer values, denoted by $\hat{\mathbf{B}} = [\hat{\mathbf{b}}_1, \ldots, \hat{\mathbf{b}}_N] \in \{-l, -l+1, \ldots, l-1, l\}^{K\times N}$. Consequently, we can build the lookup table about the dictionary $\mathbf{C}$ and apply $\mathbf{a}_i$ for table lookup operation, as will be detailed in Section 2.5. Accordingly, $\mathbf{B}$ in (2) can be replaced by $\hat{\mathbf{B}} = \mathbf{CA}$, and the problem (2) is reformulated as follows:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{C},\mathbf{A}} & \|(\mathbf{W}^T\mathbf{X})^T\mathbf{CA} - \mathbf{S}\|_F^2 \\ \text{s.t.} \quad & \mathbf{1}^T\mathbf{A} = l\,\mathbf{1}^T,\ \mathbf{A} \in \{0,1\}^{M\times N}, \\ & \mathbf{C} \in \{-1,1\}^{K\times M}, \end{aligned} \tag{3}$$

where these two embeddings can also be jointly learnt as in (2). Note that when $\mathbf{C}$ is a complete dictionary and $l = 1$, problem (3) reduces to problem (2). Theoretically, problem (2) can be regarded as a special case of problem (3).

Notably, the compositional form of $\mathbf{CA}$ is analogous to Cartesian k-means [28]. However, there exist intrinsically distinguishable differences. Cartesian k-means employs $\mathbf{CA}$ to construct more real-valued cluster centers so as to reconstruct the original features with smaller quantization errors and storage, and it only pays attention to the number of cluster centers. By contrast, AMVH intents to alleviate the binary limitation so that it utilizes this form to construct integer values based on binary atoms in a concise format, which lays emphasis on the value of compact codes.

Problem (3) mainly focuses on the pairwise relations. As demonstrated in [34], pointwise supervised information can be formulated as a classification term for guiding the hash function learning. Therefore, we additionally introduce a classification error term to the objective function in (3) as a regularization term and obtain the final formulation:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{V},\mathbf{C},\mathbf{A}} & \|(\mathbf{W}^T\mathbf{X})^T\mathbf{CA} - \mathbf{S}\|_F^2 \\ & + \lambda\|\mathbf{V}^T\mathbf{CA} - \mathbf{Y}\|_F^2 \\ \text{s.t.} \quad & \mathbf{1}^T\mathbf{A} = l\,\mathbf{1}^T,\ \mathbf{A} \in \{0,1\}^{M\times N}, \\ & \mathbf{C} \in \{-1,1\}^{K\times M}, \end{aligned} \tag{4}$$

where λ is a penalty parameter that determines the strength of the classification term and $\mathbf{V} \in \mathbb{R}^{K\times C}$ is the weight of a linear classifier, which guarantees that the good multi-integer-valued embeddings are beneficial for classification. Once problem (4) is optimized, the multi-integer-valued embeddings $\mathbf{CA}$ can be employed for compressing the database points. In addition, the linear embedding function can be utilized for encoding the newly-coming queries.

2.4. Optimization

Generally, problem (4) is a mixed integer programming problem, which is non-convex with $\mathbf{W}$, $\mathbf{V}$, $\mathbf{C}$ and $\mathbf{A}$ together. To address this problem, a well-designed alternative optimization algorithm is presented. In other words, only one variable is optimized with the others fixed at each step. In this section, the details of the optimization algorithm are elaborated as follows.

Initialization. Several initialization methods have been tried on $\mathbf{C}$ and $\mathbf{A}$. However, we empirically find that the principled initialization methods (*e.g.*, k-means clustering [10]) do not have distinct advantages of retrieval precision over random initialization. Thus, we finally initialize $\mathbf{C}$ randomly to -1 or $+1$ with the same probability, and randomly set l entries in each column $\mathbf{a}$ of $\mathbf{A}$ to 1.

W-Step. By fixing $\mathbf{V}$, $\mathbf{C}$ and $\mathbf{A}$, problem (4) is then simplified as

$$\min_{\mathbf{W}} \quad \|(\mathbf{W}^T\mathbf{X})^T\mathbf{CA} - \mathbf{S}\|_F^2, \tag{5}$$

which can be easily solved by using matrix manipulations, resulting in a closed-form solution:

$$\mathbf{W} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{S}(\mathbf{CA})^T(\mathbf{CA}(\mathbf{CA})^T)^{-1}. \tag{6}$$

C-Step. Ignoring the irrelevant variables of $\mathbf{C}$ in (4), we require to solve the following problem:

$$\begin{aligned} \min_{\mathbf{C}} \quad & \|\tilde{\mathbf{X}}^T\mathbf{CA}\|_F^2 + \lambda\|\mathbf{V}^T\mathbf{CA}\|_F^2 - 2\mathrm{tr}(\mathbf{R}^T\mathbf{C}) \\ \text{s.t.} \quad & \mathbf{C} \in \{-1,1\}^{K\times M}, \end{aligned} \tag{7}$$

where $\tilde{\mathbf{X}} = \mathbf{W}^T\mathbf{X}$ are the real-valued embeddings of $\mathbf{X}$, $\mathbf{R} = \tilde{\mathbf{X}}\mathbf{S}\mathbf{A}^T + \lambda\mathbf{V}\mathbf{Y}\mathbf{A}^T$, and $\mathrm{tr}(\cdot)$ is the trace norm. Here, $\mathbf{C}$ is circularly updated row by row via the discrete cyclic coordinate descent method [34]. Suppose that $\mathbf{C} = [\mathbf{c}^T; \mathbf{C}']$, $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}^T; \tilde{\mathbf{X}}']$, $\mathbf{V} = [\mathbf{v}^T; \mathbf{V}']$, $\mathbf{R} = [\mathbf{r}^T; \mathbf{R}']$, and $\mathbf{U} = \mathbf{A}\mathbf{A}^T$, where $\mathbf{c}^T$ is one row of $\mathbf{C}$ and $\mathbf{C}'$ is the matrix of $\mathbf{C}$ excluding $\mathbf{c}^T$. In addition, $\tilde{\mathbf{x}}^T$, $\tilde{\mathbf{X}}'$, $\mathbf{v}^T$, $\mathbf{V}'$, $\mathbf{r}^T$ and $\mathbf{R}'$ are denoted in the similar way.

For different l, there are different complexities for solving $\mathbf{C}$. Thus, we discuss the solving procedure in two cases, *i.e.*, $l = 1$ and $l > 1$. In case of $l = 1$, $\mathbf{U}$ is a diagonal matrix, which makes the quadratic term about $\mathbf{c}$ be a constant and results in a simple binary linear programming problem:

$$\begin{aligned} \min_{\mathbf{c}} \quad & \mathrm{tr}(\mathbf{c}^T\mathbf{U}(\mathbf{C}')^T\tilde{\mathbf{X}}'\tilde{\mathbf{x}}) + \mathrm{tr}(\mathbf{c}^T\mathbf{U}(\mathbf{C}')^T\mathbf{V}'\mathbf{v}) \\ & - \mathrm{tr}(\mathbf{c}^T\mathbf{r}) \\ \text{s.t.} \quad & \mathbf{c} \in \{-1, 1\}^M. \end{aligned} \tag{8}$$

Due to the simplicity of this problem, we can clearly obtain a closed-form solution:

$$\mathbf{c} = \mathrm{sign}(\mathbf{z}), \tag{9}$$

where $\mathbf{z} = \mathbf{r} - \mathbf{U}(\mathbf{C}')^T\tilde{\mathbf{X}}'\tilde{\mathbf{x}} - \lambda\mathbf{U}(\mathbf{C}')^T\mathbf{V}'\mathbf{v}$. In the case of $l > 1$, $\mathbf{U}$ is not a diagonal matrix. By some simplifications, a binary quadratic programming problem is obtained

$$\begin{aligned} \min_{\mathbf{c}} \quad & \mathbf{c}^T\mathbf{U}\mathbf{c} - 2\mathbf{c}^T\mathbf{f} \\ \text{s.t.} \quad & \mathbf{c} \in \{-1, 1\}^M, \end{aligned} \tag{10}$$

where $\mathbf{f} = \mathbf{z}/\|\tilde{\mathbf{x}}\|_2^2$. This problem can be solved by general integer programming solvers, such as Gurobi[1], but it could be time-consuming. Meanwhile, we empirically find that an approximate solution can already yield quite satisfactory results. Thus, we solve a relaxed objective of (10) by dropping the binary constraints, and obtain the following approximate solution:

$$\mathbf{c} = \mathrm{sign}(\mathbf{U}^{-1}\mathbf{f}). \tag{11}$$

A-Step. To update $\mathbf{A}$ with $\mathbf{W}, \mathbf{V}$ and $\mathbf{C}$ fixed, we let $\mathbf{G}_1 = \mathbf{X}^T\mathbf{W}\mathbf{C}$ and $\mathbf{G}_2 = \mathbf{V}^T\mathbf{C}$, and the problem (4) is rewritten as follows:

$$\begin{aligned} \min_{\mathbf{A}} \quad & \|\mathbf{G}_1\mathbf{A} - \mathbf{S}\|_F^2 + \lambda\|\mathbf{G}_2\mathbf{A} - \mathbf{Y}\|_F^2 \\ \text{s.t.} \quad & \mathbf{A} \in \{0, 1\}^{M \times N}. \end{aligned} \tag{12}$$

Clearly, the sub-problems about the columns of $\mathbf{A}$ are separable. Thus, $\mathbf{A}$ can be optimized column by column. When solving one column $\mathbf{a}$ of $\mathbf{A}$, the corresponding problem is formulated as follows:

$$\begin{aligned} \min_{\mathbf{a}} \quad & \mathbf{a}^T\mathbf{M}\mathbf{a} - 2\mathbf{a}^T\mathbf{h} \\ \text{s.t.} \quad & \|\mathbf{a}\|_0 = l, \ \mathbf{a} \in \{0, 1\}^M, \end{aligned} \tag{13}$$

[1] http://www.gurobi.com/

Algorithm 1 Asymmetric Multi-Valued Hashing

```
Input: Training data $\mathbf{X} \in \mathbb{R}^{D\times N}, \mathbf{Y} \in \{0,1\}^{C\times N}$, and
    $\mathbf{S} \in \mathbb{R}^{N\times N}$; code length $K$; dictionary size $M$; maximum iteration number $t$; hyper-parameters $l, \lambda$.
1: Initialize $\mathbf{C}$ and $\mathbf{A}$ by random initialization.
2: for iter $i = 1 \to t$ do
3:     W-Step: Update $\mathbf{W}$ using Eqn. (6)
4:     C-Step: Circularly update $\mathbf{C}$ row by row using
       Eqn. (9) for $l = 1$ and using Eqn. (11) for $l > 1$.
5:     A-Step: Update $\mathbf{A}$ column by column using
       Eqn. (14) for $l = 1$ and using the proposed forward
       greedy algorithm, i.e., Eqn. (15) for $l > 1$.
6:     V-Step: Update $\mathbf{V}$ using Eqn. (16).
7: end for
Output: Transformation matrix $\mathbf{W}$; dictionary $\mathbf{C}$; indicator matrix $\mathbf{A}$; classification matrix $\mathbf{V}$.
```

where $\|\cdot\|_0$ is the ℓ_0 norm, $\mathbf{M} = \mathbf{G}_1^T\mathbf{G}_1 + \lambda\mathbf{G}_2^T\mathbf{G}_2$, and $\mathbf{h} = \mathbf{G}_1^T\mathbf{s} + \lambda\mathbf{G}_2^T\mathbf{y}$. Besides, $\mathbf{s}$ and $\mathbf{y}$ are the corresponding columns to $\mathbf{a}$ in $\mathbf{S}$ and $\mathbf{Y}$, respectively.

Moreover, problem (13) is a constrained binary quadratic programming problem, which is generally difficult to solve. Most of the existing hashing methods solve the approximate objectives by dropping the discrete constraints [25, 35]. Alternatively, some recent works [14, 34] handle the discrete optimization problems directly, demonstrating the preferable performance. Here we also investigate how to discretely solve $\mathbf{a}$ in two cases, *i.e.*, $l = 1$ and $l > 1$. In the case of $l = 1$, the optimal solution of $\mathbf{a}$ can be easily obtained

$$a_i = \begin{cases} 1, & \text{if } i = j; \\ 0, & \text{otherwise}, \end{cases} \tag{14}$$

where $j = \arg\min_{j=1,\ldots,M} m_{jj} - 2h_j$. In the case of $l > 1$, a forward greedy algorithm is proposed to address this problem. The main procedure of this algorithm is detailed as follows. First, we define two index sets of $\mathbf{a}$, that is, $\mathcal{S} = \varnothing$ and $\bar{\mathcal{S}} = \{1, \ldots, M\}$. Second, an index p in $\bar{\mathcal{S}}$ is identified if the objective function in (13) is minimized with $a_p = 1$. Then, the index p is moved from $\bar{\mathcal{S}}$ to $\mathcal{S}$. Third, we successively find one index in $\bar{\mathcal{S}}$ at a time that minimizes (13) combined with the indices in $\mathcal{S}$, and move this index to $\mathcal{S}$. Once the size of $\mathcal{S}$ equals to l, the algorithm stops. Finally, the approximate solution of (13) is obtained

$$a_i = \begin{cases} 1, & \text{if } i \in \mathcal{S}; \\ 0, & \text{otherwise}. \end{cases} \tag{15}$$

V-Step. With the other variables fixed, $\mathbf{V}$ is updated by solving a least squares regression problem. The closed-form solution is

$$\mathbf{V} = (\mathbf{C}\mathbf{A}(\mathbf{C}\mathbf{A})^T)^{-1}(\mathbf{C}\mathbf{A})\mathbf{Y}^T. \tag{16}$$

For clarity, the whole alternative optimization algorithm of AMVH is summarized in Algorithm 1.

2.5. Query

In the query stage, the goal is to find some items from the database that are similar to the query. By the above training procedure, the transformation matrix $\mathbf{W}$, the binary dictionary $\mathbf{C}$, and the indicator matrix $\mathbf{A}$ have been obtained. Given a D-dimensional query vector $\mathbf{q}$, we first project it onto K-dimensional subspace by $\hat{\mathbf{q}} = \mathbf{W}^T\mathbf{q}$. Then, the similarities between $\hat{\mathbf{q}}$ and the multi-integer-valued embeddings $\mathbf{CA}$ of the database points are calculated, that is, $\hat{\mathbf{S}} = (\hat{\mathbf{q}})^T\mathbf{CA}$. Once $\hat{\mathbf{S}}$ is obtained, the nearest neighbors are returned by sorting these similarities.

To improve the computational efficiency, a lookup table is built for the query, denoted by $\mathbf{T_c} = (\hat{\mathbf{q}})^T\mathbf{C}$. As for computing $\hat{\mathbf{S}} = \mathbf{T_c}\mathbf{A}$, only the table lookup and addition operations are required, according to the binary indicator matrix $\mathbf{A} \in \{0,1\}^{M\times N}$. Specifically, we still discuss how to calculate $\hat{\mathbf{S}}$ in two cases, *i.e.*, $l = 1$ and $l > 1$. In the case of $l = 1$, the multi-integer-valued embeddings degrade into binary codes so that only the table lookup operations on $\mathbf{T_c}$ are required. This query strategy is denoted by AMVH$_{real}$. In the case of $l > 1$, besides the table lookup operations, l selected atoms have to be summed, thus leading to the multi-integer-embeddings, which is denoted by AMVH$_{mul}$. Notably, if we construct the binary codes of the query by sign$(\hat{\mathbf{q}})$ for $l = 1$, AMVH can also perform similarity search based on Hamming distance. This query strategy is denoted by AMVH$_{bin}$. The differences of these query strategies are clearly shown in Table 1.

Table 1. Three different query strategies of AMVH.

Method	$l=1$		$l>1$
	AMVH$_{bin}$	AMVH$_{real}$	AMVH$_{mul}$
Query	Binary	Real-valued	Real-valued
Database	Binary	Binary	Multi-integer-valued

2.6. Analysis

Query complexity. Here we analyze the query complexity of AMVH$_{real}$ and AMVH$_{mul}$. As mentioned before, for a single query, we only need the computation of $\mathbf{T_c}$ that scales in $O(KM)$ and some table lookup and addition operations to calculate $\mathbf{T_c}\mathbf{A}$, which scale in $O(l\,N)$. Due to $KM \ll N$, the query complexity mainly relies on the size N of search database and the hyper-parameter l, rather than the code length K. Therefore, the query time of AMVH$_{real}$ and AMVH$_{mul}$ is constant for all code lengths, differing from the traditional binary hashing methods, whose query complexity depends on the code length K, which prevents the use of long codes.

Storage complexity. Compared with conventional binary hashing methods, the storage complexity of AMVH$_{mul}$ is acceptable. Due to the fact that the memory cost of the dictionary $\mathbf{C} \in \{-1,+1\}^{K\times M}$ is negligible, the database storage is mainly from $\mathbf{A} \in \{0,1\}^{M\times N}$. In view of the sparsity of $\mathbf{a}$, only l indices of 1s in each $\mathbf{a}$ are required to store for each database point. Specifically, the storage of each sample (*i.e.*, $\mathbf{a}$) is $l\log M$ bits. Therefore, an interesting observation is that the storage of AMVH$_{mul}$ only relies on the hyper-parameter l and the number of atoms M, rather than the code length K. For this reason, if appropriate values of l and M are set, the storage of AMVH$_{mul}$ might be less than that of conventional binary hashing methods, even if the multi-integer-valued embeddings are employed.

3. Experiments

In this section, we compare AMVH with the binary hashing methods on search tasks in terms of search accuracy and efficiency. All the experiments are conducted on a 64-bit windows PC with 32 GB RAM and 3.50 GHz CPU.

3.1. Experimental Settings

Datasets. Three multi-label datasets are adopted to evaluate the performance of AMVH: ESP-GAME [37], MIR-FLICKR [12] and NUS-WIDE [1]. The ESP-GAME dataset consists of $20,768$ images, with each image labeled with multiple semantic labels from 268 categories. The MIR-FLICKR dataset includes $25,000$ images crawled from Flickr, with each image associated with multiple semantic labels from 38 categories. Each image of both ESP-GAME dataset and MIR-FLICKR dataset is represented by a 512-dimensional GIST [30] feature vector. The NUS-WIDE dataset contains $269,648$ images collected from Flickr, each of which belongs to multiple categories taken from 81 concept tags. And the provided 500-dimensional bag-of-words (BoW) feature vectors are utilized in our experiments. As in [14], $209,347$ images are collected by removing the images without any labels. For all datasets, two images are semantically similar if they share at least one label; otherwise, they are dissimilar. Each dataset is randomly split into a query set with 1000 samples and a training set with the remaining samples for evaluation.

Compared methods. Owing to the superiority of supervised hashing, AMVH is mainly compared against several state-of-the-art supervised hashing methods: ITQ-CCA [9], KSH [24], FastHash [21], SDH [34], COSDISH [14], and one unsupervised method SGH [13]. For the pairwise hashing methods with high computational complexity (*i.e.*, KSH and FastHash), $10K$ samples are randomly selected from training set for learning. While all the training samples are utilized for the remaining methods, due to their favorable scalability. All experiments are repeated 10 times with random data partitions, and the averaged results are reported.

3.2. Experimental Details

Kernel feature mapping. RBF kernel mapping is a commonly-used and powerful method for nonlinear hashing [17, 24, 34]. For each data point $\mathbf{x} \in \mathbb{R}^D$, we random-

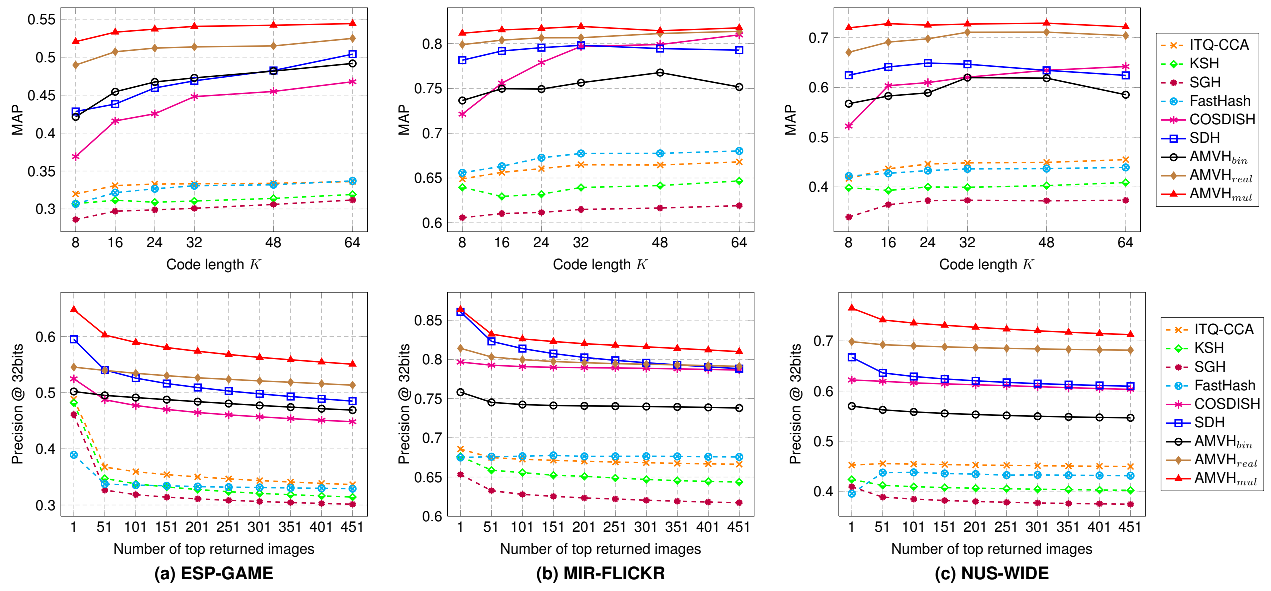

Figure 1. Retrieval performance of different hashing methods on three datasets. Top row: MAP with different code lengths; bottom row: top-K precision with 32-bit codes w.r.t. different numbers of top returned images.

ly select m anchor points $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ from the database and map $\mathbf{x}$ into an m-dimensional representation, using $\phi(\mathbf{x}) = [\exp(\|\mathbf{x} - \mathbf{u}_1\|^2/\sigma), \ldots, \exp(\|\mathbf{x} - \mathbf{u}_m\|^2/\sigma)]$. Here, σ is the kernel width estimated according to the average Euclidean distances between the training samples.

Label transformation. Given the binary label vectors, we generate the similarity matrix by $\mathbf{Y}^T\mathbf{Y}$. Compared to the binary similarity matrix used in [21, 24], $\mathbf{Y}^T\mathbf{Y}$ can make better use of the supervised information implied in the semantic labels. In addition, as will be detailed later, it also facilitates the training of large-scale dataset efficiently. However, due to the label imbalance problem demonstrated by [5], $\mathbf{Y}^T\mathbf{Y}$ is improper for direct use. For this reason, the label transformation is applied to construct a new semantic similarity matrix: $\hat{\mathbf{S}} = (1+\alpha)\mathbf{Y}^T\mathbf{Y} - \alpha$. Here α is set to $\mu(n_s/n_d)$, where μ is a hyper-parameter; n_s and n_d are the number of similar and dissimilar pairs, respectively, which satisfy $n_s + n_d = N^2$. Explicitly using $\hat{\mathbf{S}}$ in Algorithm 1 is both time and storage consuming. Inspired by SGH [13], $\hat{\mathbf{S}}$ is decomposed into a product of two smaller matrices, that is, $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{Q}$, where $\mathbf{P} = [\sqrt{1+\alpha}\mathbf{Y}; \sqrt{\alpha}\mathbf{1}^T], \mathbf{Q} = [\sqrt{1+\alpha}\mathbf{Y}; -\sqrt{\alpha}\mathbf{1}^T] \in \mathbb{R}^{(C+1)\times N}$. Thus, we employ the factorization instead of the original $\hat{\mathbf{S}}$ for computation, which reduces the complexity from $O(N^2)$ to $O(NC)$.

Implementation details. For the compared hashing methods, the public codes and the suggested parameters from the corresponding authors are utilized. For the kernel-based hashing methods (KSH, SDH, and AMVH), $m = 1,000$ anchors are randomly chosen for RBF kernel mapping. For a fair comparison, RBF-kernel SVM and kernel ridge regression are adopted as hash functions for FastHash and COSDISH, respectively. For AMVH, we set parameter λ to 1 N/C and μ to 0.5 via cross-validation; dictionary size to 256, maximum iteration number t to 15, parameter l to 1 or 10. And $\mathbf{C}$ is circularly updated 10 times.

Evaluation criteria. The widely used criteria: mean average precision (MAP) and top-K precision are adopted to evaluate retrieval performance. Additionally, we report the query time under different code lengths to evaluate the search efficiency and the training time to evaluate the scalability. For all methods, the query time consists of three parts: one query sample encoding, distance (similarity) matrix construction and RBF kernel mapping if any.

3.3. Results

Retrieval performance. The retrieval performance of different hashing methods on ESP-GAME, MIR-FLICKR and NUS-WIDE is illustrated in Fig. 1. It is obvious that the performance of the unsupervised hashing method (SGH) is worse than that of the supervised methods on all datasets. Moreover, the asymmetric hashing methods, especially COSDISH, SDH and AMVH, achieve preferable performance over the symmetric methods (*e.g.*, KSH and ITQ-CCA). Therefore, we mainly compare AMVH against COSDISH and SDH in the remaining sections.

From the top row in Fig. 1, it broadly shows that both AMVH$_{real}$ and AMVH$_{mul}$ outperform the other six methods remarkably on three datasets, especially on ESP-GAME and NUS-WIDE. First, we compare AMVH$_{real}$ with SDH and COSDISH, as the database points are represented by binary codes in these methods. Obviously, the performance of AMVH$_{real}$ is better than the others. For example, AMVH$_{real}$ outperforms SDH by 14.53% with 8 bits on ESP-GAME. Clearly, this shows that the real-valued query

Table 2. Training and testing time of different hashing methods on MIR-FLICKR and NUS-WIDE.

Method	MIR-FLICKR (25K, 512-GIST)						NUS-WIDE (209K, 500-BoW)					
	Training Time (s)			Testing Time (ms)			Training Time (s)			Testing Time (ms)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
ITQ-CCA	11.28	11.84	13.20	0.0865	0.1292	0.1255	119.16	123.39	126.22	0.6467	0.8631	0.8959
KSH	630.91	1285.59	2628.70	0.1197	0.1613	0.1535	624.58	1160.41	2201.88	0.7345	0.8956	0.8533
SGH	6.61	11.73	22.64	0.1280	0.1708	0.1655	31.19	54.37	77.89	0.7193	0.9120	0.8698
FastHash	346.15	632.88	1170.55	0.1232	0.1625	0.1566	402.15	747.89	1469.37	0.7462	0.9108	0.8756
COSDISH	5.49	17.50	67.85	0.1210	0.1619	0.1530	49.59	129.72	425.68	0.7125	0.9135	0.8797
SDH	10.02	20.08	48.45	0.1248	0.1681	0.1606	110.61	235.20	598.67	0.7150	0.9224	0.8662
AMVH$_{real}$	7.03	11.93	36.65	0.1642	0.1618	0.1615	61.06	105.38	316.41	1.1274	1.1265	1.1363
AMVH$_{mul}$	14.21	18.96	59.38	0.3814	0.3888	0.3852	126.76	166.89	496.78	3.1443	3.0689	3.1317

can improve the search accuracy. Second, compared with AMVH$_{real}$, AMVH$_{mul}$ further enhances the capability of similarity preservation by the multi-integer-valued embedding, achieving the highest MAP on all datasets. For example, AMVH$_{mul}$ outperforms COSDISH by 12.41% with 64 bits on NUS-WIDE. Thus, these results verify the superiority of the multi-integer-valued embedding strategy of our approach. Third, AMVH$_{bin}$ obtains a little worse performance, which is because that transforming real values to binary codes brings some quantization error. However, AMVH$_{bin}$ is comparable to SDH and COSDISH on ESP-GMAE and NUS-WIDE. This indirectly demonstrates the power of our approach. Finally, it is worthy to note that the performance of AMVH$_{real}$ and AMVH$_{mul}$ with shorter codes is superior to that of SDH and COSDISH with 64 bits in most cases. This indicates that AMVH can achieve satisfying MAP with short code length by leveraging the real-valued and the multi-integer-valued embeddings.

From the bottom row in Fig. 1, the top-K precision of AMVH$_{mul}$ is much higher than other methods, especially on ESP-GAME and NUS-WIDE. Concretely, AMVH$_{mul}$ gains 11.53% improvement over SDH on ESPGAME at top-51 and 18.50% improvement over COSDISH on NUS-WIDE at top-251. Morever, although the top-1 precision of SDH is almost equal to that of AMVH$_{mul}$, the performance degradation of SDH is more serious than AMVH$_{mul}$ and COSDISH, on MIR-FLICKR. In a word, AMVH$_{mul}$ retains stable performance and preserves the highest precision with different numbers of top returned images.

Training time. In order to evaluate the efficiency of our approach, we report the training time of AMVH$_{real}$ and AMVH$_{mul}$ on two large datasets MIR-FLICKR and NUS-WIDE in Table 2. It can be seen that some prior hashing methods are time-consuming due to direct manipulations on the $N \times N$ similarity matrix $\mathbf{S}$ (*e.g.*, FastHash and KSH). In comparison with COSDISH and SDH, AMVH$_{real}$ takes a little longer time to learn 16-bit hash function and less time for 32-bit and 64-bit. Moreover, the training time of AMVH$_{mul}$ is longer than that of AMVH$_{real}$ but comparable to that of SDH and COSDISH with longer codes.

To further evaluate the scalability of our approach, we select five training subsets of different sizes from the NUS-WIDE. Table 3 reports the training time of COSDISH, SDH and AMVH on different subsets of NUS-WIDE with 16-bit codes. Clearly, AMVH$_{real}$ is more scalable than SDH and is slightly worse than COSDISH, but AMVH$_{mul}$ is comparable with SDH and is worse than COSDISH. However, refering to Table 2, our approach is scalable with different code lengths. In summary, AMVH is scalable to deal with large-scale datasets and long code lengths.

Table 3. Training time (in seconds) on different sizes of subsets from NUS-WIDE with 16-bit codes.

Method	NUS-WIDE (209K, 500-BoW)				
	5 K	20 K	50 K	100 K	200 K
COSDISH	1.46	4.10	10.64	22.18	41.14
SDH	2.50	11.83	28.99	61.95	130.89
AMVH$_{real}$	1.79	6.33	16.26	34.98	69.55
AMVH$_{mul}$	3.24	12.10	29.98	63.01	137.43

Testing time. Referring to Table 2, three conclusions can be drawn as follows: (1) The testing time of the compared methods becomes longer as the code length increases, while that of AMVH$_{real}$ and AMVH$_{mul}$ is almost the same for all code lengths. The reason is that, as analyzed in Section 2.6, the testing time of AMVH$_{real}$ and AMVH$_{mul}$ only depends on l and the database size N, rather than the code length K; (2) AMVH$_{real}$ takes slightly longer testing time than other methods except AMVH$_{mul}$, since the floating point operations are more complex than bitwise operations; (3) The testing time of AMVH$_{mul}$ is the longest, but it can obtain such desirable performance as shown in Fig 1. Since both the testing time and retrieval performance rely on l in our approach, it is flexible to make a trade-off between them.

3.4. Empirical Analysis

Convergence. We validate the convergence of the alternative optimization algorithm by experiments. The convergence curves on three datasets with 16-bit codes are illustrated in the left of Fig. 3. For convenience, the objective values are normalized by dividing the maximum on each

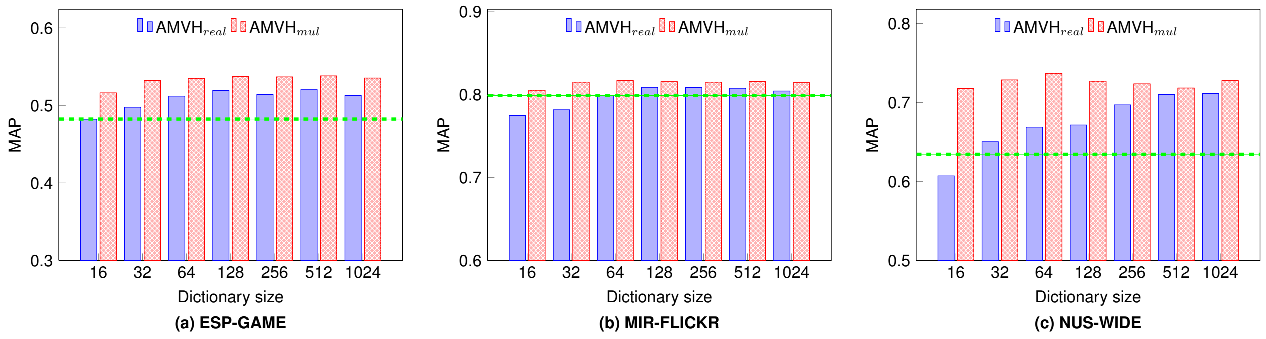

Figure 2. The search performance of AMVH$_{real}$ and AMVH$_{mul}$ on the different dictionary sizes on three datasets with 48-bit codes. Green dashed lines represent the highest MAP of the compared binary hashing methods with 48-bit codes.

dataset. Clearly, the proposed algorithm converges in less than 15 iterations, demonstrating the high convergence rate.

The effect of l. To empirically verify the effect of l, the MAP on all datasets with 16-bit codes is reported in the right of Fig. 3. For convenience, we normalize the MAP by dividing the maximum on each dataset. It can be seen that the search performance has a strong relation to l. The MAP increases rapidly between $l = 1$ and $l = 10$, and tends to be saturated when l is larger than 15. Clearly, AMVH achieves high MAP with smaller l on NUS-WIDE. It means that choosing a larger l is not a strict requirement, and thus l is set to 10 for AMVH$_{mul}$ on all datasets just to demonstrate the superiority of the multi-integer-valued embeddings.

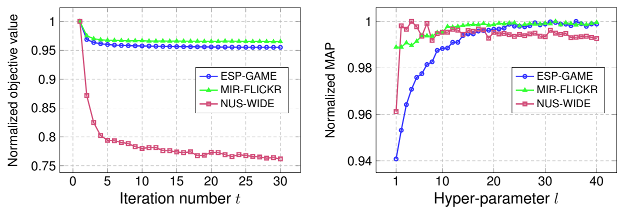

Figure 3. Left: convergence curves of AMVH$_{mul}$ on three datasets with 16-bit codes. Right: the effect of l on MAP on three datasets with 16-bit codes.

The effect of M. To investigate the effect of dictionary size M on the search performance, we show the MAP results on three datastes with 48-bit codes in Fig. 2. As can be seen from this figure, the MAP of AMVH$_{real}$ gradually becomes higher with more atoms, especially on NUS-WIDE. While AMVH$_{mul}$ always keeps stably satisfying performance even in the extreme case of 16 atoms. In Fig. 2, green dashed lines represent the highest MAP of the compared binary hashing methods with 48-bit codes. Compared with this baseline, AMVH$_{mul}$ with 16 atoms outperforms the binary hashing with 48-bit codes on all datasets. While AMVH$_{real}$ requires more atoms to surpass them. The underlying reason is that AMVH$_{real}$ chooses only one atom to represent the database points. Thus, there are only M kinds of hash codes. While AMVH$_{mul}$ use the sum of 10 atoms selected from M atoms for encoding so that there are far more than M combinations. Consequently, it reveals that the multi-integer-valued embeddings can remarkably improve the search accuracy even with a small dictionary.

As previously mentioned, the storage of AMVH$_{mul}$ with 32 atoms is $10 \times \log_2 32 = 50$ bits per sample, which approximately equals to that of the binary hashing methods with 48 bits. As illustrated in Fig. 2, AMVH$_{mul}$ with 32 atoms is greatly superior to the compared binary hashing methods with approximate storage cost (*i.e.*, 48 bits) on three datasets. In addition, AMVH$_{real}$ is also better than the binary hashing methods in most cases. Specifically, the storage of AMVH$_{real}$ with 64 atoms (*i.e.*, $1 \times \log_2 64 = 6$ bits) is far less than 48 bits of binary codes. To sum up, AMVH in a way of the multi-valued embeddings achieves favorable performance with the same or even less storage cost compared to the traditional binary hashing methods.

4. Conclusion

In this paper, we proposed the Asymmetric Multi-Valued Hashing method (AMVH). By leveraging both the real-valued embeddings and the multi-integer-valued embeddings, the proposed approach alleviates the binary limitation and achieves appealing search performance. Due to the well-designed binary sparse representation and the efficient alternative optimization algorithm, AMVH remains high efficiency of query and storage, even if the non-binary embeddings are employed. Experiments conducted on three multi-label datasets demonstrate the superiority of our AMVH to the existing conventional binary hashing methods.

5. Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 91646207, 91338202, 61620106003, and 61671451. This work was also supported in part by the Strategic Priority Research Program of the CAS (Grant XDB02060009) and the Beijing Natural Science Foundation under Grant (4162064).

References

[1] T. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. NUS-WIDE: a real-world web image database from national university of singapore. In *CIVR*, pages 1–9, 2009. 5

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

[3] K. Ding, B. Fan, C. Huo, S. Xiang, and C. Pan. Cross-modal hashing via rank-order preserving. *TMM*, 19(3):571–585, 2017. 1

[4] K. Ding, C. Huo, B. Fan, and C. Pan. knn hashing with factorized neighborhood representation. In *ICCV*, pages 1098–1106, 2015. 2

[5] K. Ding, C. Huo, B. Fan, S. Xiang, and C. Pan. In defense of locality-sensitive hashing. *TNNLS*, preprint:1–17, 2016. 1, 6

[6] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *SIGIR*, pages 123–130, 2008. 1

[7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999. 1

[8] 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. 1

[9] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 35(12):2916–2929, 2013. 1, 5

[10] Y. Gong, M. Pawlowski, F. Yang, L. Brandy, L. Boundev, and R. Fergus. Web scale photo hash clustering on a single machine. In *CVPR*, pages 19–27, 2015. 3

[11] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik. Asymmetric distances for binary embeddings. *TPAMI*, 36(1):33–47, 2014. 1, 2

[12] M. J. Huiskes and M. S. Lew. The MIR flickr retrieval evaluation. In *MIR*, pages 39–43, 2008. 5

[13] Q. Jiang and W. Li. Scalable graph hashing with feature transformation. In *IJCAI*, pages 2248–2254, 2015. 1, 5, 6

[14] W. Kang, W. Li, and Z. Zhou. Column sampling based discrete supervised hashing. In *AAAI*, pages 1230–1236, 2016. 1, 2, 4, 5

[15] W. Kong and W. Li. Isotropic hashing. In *NIPS*, pages 1655–1663, 2012. 1

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 1

[17] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009. 1, 5

[18] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *TPAMI*, 34(6):1092–1104, 2012. 1

[19] W. Li, S. Wang, and W. Kang. Feature learning based deep supervised hashing with pairwise labels. In *IJCAI*, pages 1711–1717, 2016. 1

[20] 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. 1

[21] G. Lin, C. Shen, and A. van den Hengel. Supervised hashing using graph cuts and boosted decision trees. *TPAMI*, 37(11):2317–2331, 2015. 1, 2, 5, 6

[22] L. Liu and L. Shao. Sequential compact code learning for unsupervised image hashing. *TNNLS*, preprint:1–11, 2016. 1

[23] W. Liu, C. Mu, S. Kumar, and S. Chang. Discrete graph hashing. In *NIPS*, pages 3419–3427, 2014. 1

[24] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012. 1, 2, 5, 6

[25] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011. 4

[26] B. Neyshabur, N. Srebro, R. Salakhutdinov, Y. Makarychev, and P. Yadollahpour. The power of asymmetry in binary hashing. In *NIPS*, pages 2823–2831, 2013. 1

[27] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011. 1

[28] M. Norouzi and D. J. Fleet. Cartesian k-means. In *CVPR*, pages 3017–3024, 2013. 3

[29] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, pages 1070–1078, 2012. 1

[30] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001. 5

[31] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009. 1

[32] J. Sánchez, F. Perronnin, T. Mensink, and J. J. Verbeek. Image classification with the fisher vector: Theory and practice. *IJCV*, 105(3):222–245, 2013. 1

[33] F. Shen, W. Liu, S. Zhang, Y. Yang, and H. T. Shen. Learning binary codes for maximum inner product search. In *ICCV*, pages 4148–4156, 2015. 1, 2

[34] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *CVPR*, pages 37–45, 2015. 1, 2, 3, 4, 5

[35] F. Shen, C. Shen, Q. Shi, A. van den Hengel, Z. Tang, and H. T. Shen. Hashing on nonlinear manifolds. *TIP*, 24(6):1839–1851, 2015. 4

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

[37] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI*, pages 319–326, 2004. 5

[38] J. Wang, O. Kumar, and S. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010. 1

[39] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927:1–29, 2014. 1

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

[41] J. Zhang, Y. Peng, and J. Zhang. SSDH: semi-supervised deep hashing for large scale image retrieval. *CoRR*, abs/1607.08477:1–13, 2016. 1