Post Tuned Hashing: A New Approach to Indexing High-dimensional Data

Zhendong Mao[1], Quan Wang[1], Yongdong Zhang[2], Bin Wang[1]
[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]Institute of Computing Technology, Chinese Academy of Sciences
maozhendong@iie.ac.cn,wangquan@iie.ac.cn,zhyd@ict.ac.cn,wangbin@iie.ac.cn

ABSTRACT

Learning to hash has proven to be an effective solution for indexing high-dimensional data by projecting them to similarity-preserving binary codes. However, most existing methods end up the learning scheme with a binarization stage, *i.e.*, binary quantization, which inevitably destroys the neighborhood structure of original data. As a result, those methods still suffer from great similarity loss and result in unsatisfactory indexing performance. In this paper we propose a novel hashing model, namely Post Tuned Hashing (PTH), which includes a new post-tuning stage to refine the binary codes after binarization. The post-tuning seeks to rebuild the destroyed neighborhood structure, and hence significantly improves the indexing performance. We cast the post-tuning into a binary quadratic optimization framework and, despite its NP-hardness, give a practical algorithm to efficiently obtain a high-quality solution. Experimental results on five noted image benchmarks show that our PTH improves previous state-of-the-art methods by 13-58% in mean average precision [1].

KEYWORDS

Learning to hash; Unsupervised binary hashing; ANN search

ACM Reference Format:
Zhendong Mao[1], Quan Wang[1], Yongdong Zhang[2], Bin Wang[1] . 2018. Post Tuned Hashing: A New Approach to Indexing High-dimensional Data. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3240508.3240529

1 INTRODUCTION

As opposed to traditional tree-based indexing methods which suffer from the curse of dimensionality [7], binary hashing has recently attracted much attention due to its computational and storage efficiency [1, 5, 16, 20, 21, 27, 30, 31, 33]. The goal of binary hashing is to learn low-dimensional and similarity-preserving binary codes to represent original high-dimensional data. With binary codes, the task of nearest neighbor search can be efficiently conducted through bitwise operations. Binary hashing techniques can be roughly categorized into supervised (semi-supervised) methods and unsupervised methods. Supervised methods incorporate label information to learn binary codes, making it particularly suitable for semantic search [19, 21, 24, 27–29, 34, 37]. Unsupervised hashing methods explore intrinsic properties of data to learn binary codes, which can be used for indexing unlabeled data [2, 5, 6, 9, 13, 22, 33, 36]. In this paper we focus on unsupervised hashing.

Traditional unsupervised hashing follows a two-stage paradigm [2, 35, 36]: 1) a projection stage which maps original data to a low-dimensional continuous space, and 2) a binarization stage which binarizes the projection results into the final codes. Previous studies focus their attention either on the projection stage or on the binarization stage, with an aim to preserve as much as possible the neighborhood relationships. Most of the currently available methods try to preserve similarity in the projection stage by exploring different properties of original data, such as distributions [6, 32], pairwise relationships [29, 33], manifold structures [9, 22, 36] and so on. Notable examples include Spectral Hashing (SH) [33], Anchor Graph Hashing (AGH) [22] and Spherical Hashing (SpH) [6]. Some other methods seek to reduce the quantization loss in the binarization stage. For example, Iterative Quantization (ITQ) [5] applies an orthogonal rotation to the projected data so as to minimize the difference between projection results and binarization results. Double Bit Quantization (DBQ) [12] proposes to quantize each projection dimension into multiple bits.

Although trying to preserve the neighborhood relationships, those two-stage approaches end up with a binarization stage which will inevitably destroy the neighborhood structure of original data, and hence result in a great ***neighborhood error*** , *i.e.*, neighboring points in the original data space get dissimiar codes while non-neighboring points similar codes. Figure 1 provides an illustration of the neighborhood error in two state-of-the-art hashing methods, *i.e.*, SpH and ITQ. It clearly shows that both SpH and ITQ binarize a large number of neighboring points into different codes, but meanwhile a large number of non-neighboring points into the same

[1]We provide a toy demo of PTH at https://github.com/maozhendong

MM '18, October 22–26, 2018, Seoul, Republic of Korea

ACM ISBN 978-1-4503-5665-7/18/10…$15.00
https://doi.org/10.1145/3240508.3240529

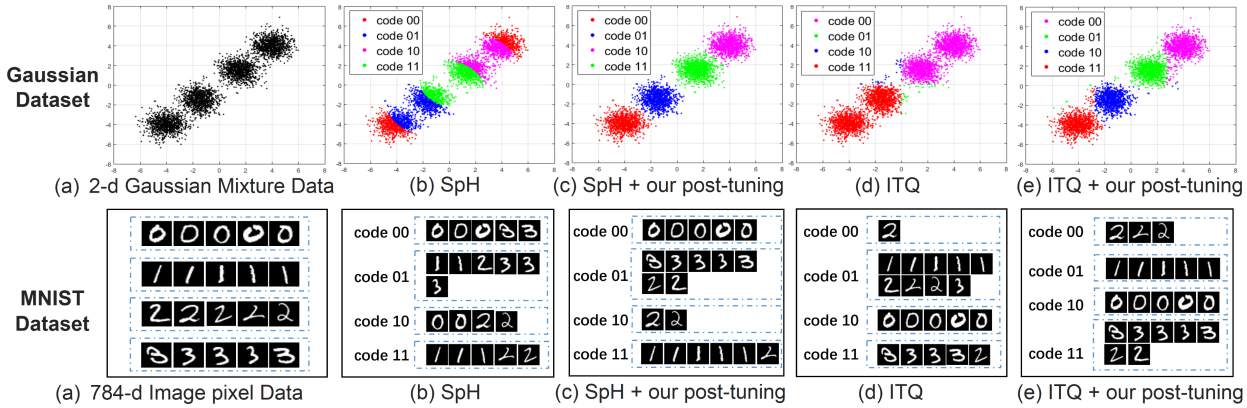


Figure 1: Toy illustration of the superiority of post-tuning. Two datasets are used, *i.e.*, 4,000 2-d points from 4 different Gaussian distributions (top) and image pixels of digits 0,1,2,3 from MNIST (bottom, the first 5 images of each digit are shown). (a) shows the data points. (b) and (d) show the 2-bits binary codes of these points learned by SpH[6](non-linear method) and ITQ[5](linear method), respectively. Clearly, SpH and ITQ significantly destroy the neighborhood structure of original data and result in a large *neighborhood error*. On the Gaussian mixture data (top), SpH (b) binarizes neighboring points (points from the same distribution) into different codes, and ITQ (d) binarizes non-neighboring points (points from different distributions) into the same code. Similar phenomenon is observed on the image pixel data (bottom, (b) and (d)). (c) and (e) show the results after post-tuning SpH and ITQ, respectively. We see that post-tuning helps to rebuild the neighborhood structure and hence greatly reduces the neighborhood error.

code. The neighborhood error will significantly degrade the performance of binary codes in ANN search.

In this paper we propose a novel three-stage hashing model, namely Post Tuned Hashing (PTH), to minimize the neighborhood error. PTH is distinct from prior works by adopting a new independent post-tuning stage to rebuild the destroyed neighborhood structure after binarization. In this new stage, a post-tuning matrix is learned to refine the raw binarization outputs, *e.g,* binary codes of most existing methods. We cast the problem of learning the post-tuning matrix into a binary quadratic optimization framework where the objective is to minimize the neighborhood error. To tackle the binary quadratic optimization in a computationally tractable manner, we transform it into a combinatorial optimization problem with the properties of matroids and give an optimized greedy algorithm to efficiently obtain a local optimum. Our key contributions can be summarized as follows:

1. We propose a novel hashing model, namely PTH, which includes three stages: projection, binarization and post-tuning. A post-tuning algorithm tailored for this model is presented. To the best of our knowledge, this is the first work that proposes to independently post-tune the binary codes after binarization for rebuilding neighborhood structure.

2. We propose an out-of-sample extension of the post-tuning algorithm, through which PTH can be easily generalized to handle new data points outside the original training set. Moreover, PTH is a quite general framework. The post-tuning algorithm can be applied to a wide range of existing methods to further refine their binary codes and improve their performance.

3. Extensive experiments on five benchmark datasets demonstrate the superiority of our method. PTH improves previous state-of-the-art methods by 13-58% in mAP.

2 RELATED WORK

Indexing high-dimensional data is a fundamental task in retrieval and recognition applications [30, 31][17, 18]. In recent years, binary hashing, which maps high-dimensional data into compact similarity-preserving binary codes, has attracted much attention due to the significant efficiency gains in both storage and speed [1, 5, 15, 16, 20, 21, 27, 33]. The early binary hashing methods are data-independent, notable example is Locality-Sensitive Hashing (LSH) [1]. However, this kind of methods need relatively long codes to ensure good search performance. To solve this problem, many data-dependent methods have been proposed. Supervised data-dependent methods incorporate the label information to learn compact binary codes [15, 19, 21, 24, 27–29, 34, 37] while unsupervised data-dependent methods only explore data intrinsic properties [2, 5, 6, 9, 13, 22, 32, 33, 36]. We focus here on data-dependent, unsupervised approaches.

Most of the unsupervised methods address the problem of learning binary code as an optimization problem where the objective function is designed to preserve similarity. Unfortunately, the objective function is typically very challenging (NP-hard in general) due to the discrete constraints. A practical and widely used solution is to relax the discrete constraints to divide the hashing into two independent stages [4, 21, 35, 36]: continuous projection for dimensionality reduction and binarization for generating binary codes. Most

of works focus on preserving similarity in the projection stage, *e.g.*, Spectral Hashing (SH) [33], Spherical Hashing (SpH) [6], PCA Hashing (PCAH) [32] and Anchor Graph Hashing (AGH) [22]. Some approaches seek to reduce the quantization loss in the binarization stage, *e.g.*, Iterative Quantization(ITQ) [5], Discrete Collaborative Filtering (DCF) [35], Hierarchical Quantization (HQ) [22] and Double Bit Quantization (DBQ) [12]. However, the two-stage approaches still suffer from a great similarity loss owing to the last binarization stage.

3 POST TUNED HASHING

Given a set $X \in \mathbb{R}^{d\times n}$ of n data points $\{x_i\}_{i=1}^n \in \mathbb{R}^d$, the paradigm of unsupervised binary hashing is to first project the data points to a low-dimensional continuous manifold by using linear or non-linear functions $P : \mathbb{R}^d \to \mathbb{R}^m$, and then binarize the projection results to map them to a binary valued space, namely Hamming space[2]:

$$H(X) = sgn(P(X)) \tag{1}$$

where $sgn : \mathbb{R}^m \to \{-1, 1\}^m$ is an element-wise function which quantizes each dimension of the projected data into a binary value by thresholding at 0.

However, this two-stage paradigm is inadequate in preserving the neighborhood relationships of data. Even if the neighborhood relationships are well preserved in the first projection stage, the later binarization stage will inevitably destroy the neighborhood structure of original data. Therefore, existing two-stage methods suffer from a great ***neighborhood error***, *i.e.*, neighboring points in the original data space get dissimiar codes while non-neighboring points similar codes (refer to Fig. 1). Without loss of generality, the neighborhood error can be defined as:

$$\mathcal{L} = ||S - V||_F^2 \tag{2}$$

where $||\cdot||_F$ is the Frobenius norm, S and $V \in \mathbb{R}^{n\times n}$ are the pairwise neighborhood relationship matrixes of the original data points X and their binary codes B, respectively. Specifically, the ij-th entry of S denotes whether x_i and x_j are neighboring points, and the ij-th entry of V denotes whether the codes of x_i and x_j are similar. Obviously, a large $\mathcal{L}$ will significantly degrade the performance of the binary codes in ANN search.

To address this problem, we propose a three-stage model, namely Post Tuned Hashing (PTH), where a novel post-tuning procedure $R : \{-1, 1\}^m \to \{-1, 1\}^m$ is incorporated to further refine the binary codes so as to minimize the neighborhood error:

$$PTH(X) = R(H(X)) \tag{3}$$

Note that a wide range of existing methods can be used to generate $H(X)$ in this model. That means, the proposed post-tuning procedure can be applied to a wide range of existing methods to further refine their binary codes and improve their performance.

[2]we use the term binary to refer to either $\{0, 1\}$ or $\{-1, 1\}$, we use -1 in the code learning and lastly, convert it to 0 for storage and query.

3.1 Overall Framework

To minimize the neighborhood error $\mathcal{L}$, we need to rewrite it in a computationally tractable manner. In Eq. (2), the ij-th entry of S denotes the neighborhood relationship between x_i and x_j, which is determined by their Euclidean distance $d(x_i, x_j)$ in the original data space[3]:

$$S_{ij} = \begin{cases} 1 & if \;\; d(x_i, x_j) < \epsilon \\ -1 & otherwise \end{cases} \tag{4}$$

where ϵ is a threshold for identifying neighbors in Euclidean space[4]. The ij-th entry of V denotes the neighborhood relationship between the binary codes b_i and b_j (codes of x_i and x_j), which is defined by a scaled inner product:

$$V_{ij} = (b_i \cdot b_j)/m \tag{5}$$

V_{ij} is a discrete variable ranging from -1 to 1. $V_{ij} = 1$ indicates that b_i and b_j are exactly the same and $V_{ij} = -1$ indicates that b_i and b_j are completely different. Now we can rewrite the neighborhood error as:

$$\mathcal{L} = ||S - \frac{1}{m}B^T B||_F^2 \tag{6}$$

where $B \in \{-1, 1\}^{m\times n}$ denotes the binary codes of X. Eq. 6 is similar to a model of supervised hashing proposed by kernel-based supervised hashing (KSH) [21]. However, KSH is a two-stage hashing method which tries to minimize the model directly over the binary codes B by relaxing B to real ones, and then uses the *sgn* function for binarization. This will inevitably destroy the neighborhood structure, and is exactly the problem we aim to address.

PTH minimizes $\mathcal{L}$ over a binary matrix $\boldsymbol{U}$, namely ***post-tuning matrix***, so as to refine the binary code $Z = H(X)$. It is a post-process after hashing. Z could be the binary codes generated by any hashing method, *e.g.*, KSH. The post-process rebuilds the neighborhood structure destroyed by the binarization stage. By adopting the binary post-tuning matrix, the post-process results will be binary by nature.

$$\begin{aligned} &min \;\; \mathcal{Q}(U) = ||S - \frac{1}{m}(U \circ Z)^T(U \circ Z)||_F^2 \\ &subject \;\; to \qquad u_{ij} \in \{-1, 1\} \end{aligned} \tag{7}$$

where $\circ$ represents the Hadamard product (*i.e.*, element-wise product), u_{ij} denotes the ij-th entry of U. Or rather, each entry u_{ij} of the post-tuning matrix U indicates whether the corresponding bit (*i.e.*, the ij-th entry of Z) should be flipped to minimize the neighborhood error. The final binary codes of PTH are the refined results $B = U \circ Z$.

3.2 Optimization Algorithm

Even if each code in Z is a single bit, *i.e.*, the code length is 1 ($m = 1$), finding an optimal U for Eq. (7) is an NP-hard problem since it is equivalent to the unconstrained boolean quadratic program which is long-known to be computationally

[3]Using -1 alone to represent non-neighboring relationships may be too coarse. Typically, we use $(e^{-d(x_i,x_j)} - e^{-\epsilon})/e^{-\epsilon}$ instead of -1 in Eq.4

[4]We find an optimal ϵ that leads to the highest mAP on the training set by using half-interval search.

difficult [23]. Therefore, the problem of finding an optimal U can be solved only by enumeration at cost $O(2^{m\times n})$. However, a practical algorithm to efficiently obtain a local optimum can be given based on the following observation:

Observation *Any quadratic term in the objective function is a constant, minimizing $\mathcal{Q}(U)$ is to minimize the sum of its linear terms.*

The observation tells us that we need to determine each entry of U to minimize the sum of the linear terms w.r.t those entries. To this end, we first calculate the linear terms w.r.t. each entry of U. Let $\gamma = 1/m$, we rewrite Eq. (7) as:

$$\min_{U} \sum_{ij} [S_{ij} - \gamma(\sum_{k=1}^{m} u_{ki} z_{ki} u_{kj} z_{kj})]^2 \tag{8}$$

In Eq. (8), the linear terms w.r.t. the p-th row of U are:

$$-2\sum_{ij} u_{pi} u_{pj} z_{pi} z_{pj} [\gamma S_{ij} - \gamma^2 (\sum_{\substack{k=1 \\ k\neq p}}^{m} u_{ki} z_{ki} u_{kj} z_{kj})] \tag{9}$$

Let $\bar{z}_p$ be the column vectors whose entries are those of the p-th row of Z, Q_{ij} the ij-th entry of the matrix $Q = \bar{z}_p \cdot \bar{z}_p^T$. Let $(U \circ Z)_{\backslash p}$ be the matrix of $U \circ Z$ with the p-th row removed, and O_{ij} the ij-th entry of the matrix $O = [(U \circ Z)_{\backslash p}]^T \cdot [U \circ Z)_{\backslash p}]$. we can rewrite (9) as :

$$-2\gamma \sum_{ij} u_{pi} Q_{ij} (S_{ij} - \gamma O_{ij}) u_{pj} \tag{10}$$

Let c_{ij} be the ij-th entry of the symmetric matrix $C = Q \circ (S - \gamma O)$, the linear terms with respect to the pq-th entry of U is given:

$$-4\gamma(\sum_{\substack{k=1 \\ k\neq q}}^{n} u_{pk} c_{qk}) u_{pq} \tag{11}$$

For the entry u_{pq}, minimizing $\mathcal{Q}(U)$ is to minimize (11). The linear coefficient $-4\gamma(\sum_{k\neq q} u_{pk} c_{qk})$ can be considered as the weight of u_{pq}. Therefore, we need to have $u_{pq} = 1$ whenever its linear coefficient ≤ 0 and -1 otherwise.

Eq. (11) shows that there is interference in determining different entries of U, and the weights of those entries are different. This is a typical combinatorial optimization, which can be solved by a greedy algorithm: initialize U to a m-by-n matrix of ones and at each iteration update an entry with the largest linear coefficient (in absolute value) until the gain is small. This is the basic post-tuning algorithm. We further improve it by the following optimizations:

Updating strategy. We change the updating strategy to "update the current entry if its coefficient is larger than a fixed threshold η". So we can determine the entries in sequential order, which significantly reduces the complexity of the algorithm. In our experiment, η is set to the mean of all coefficients. Furthermore, we use the same matrix C for updating a row of U, since C is determined by all the row of U and thus not sensitive to the change of one row. It achieves nearly the same performance as element-by-element updating but is much faster.

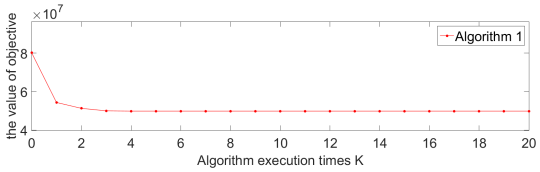


Figure 2: Behavior of the objective Eq.(7) with Algorithm 1.

Pruning strategy. We only tune the elements whose projection results (value before binarization) are close to 0, or smaller than a threshold δ. The reason is that only these elements have high probability of being binarized into incorrect values. The pruning reduces the complexity of the post-tuning, and also improves its effectiveness by the reduction of over-fitting. δ is set to the mean absolute value of projection results in our experiment.

The final post-tuning algorithm is summarized in Algorithm 1. Note that it should be performed K times to iteratively refine the code until the objective achieves a stable optimum. The typical behavior of the objective (7) with respect to K is shown in Figure 2. We set $K = 5$.

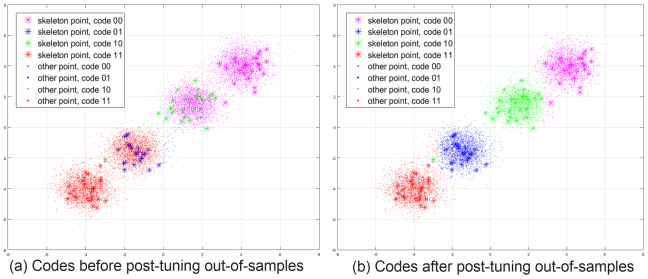
Algorithm 1 Post-tuning Algorithm

Require: Data $\mathbf{X}$, Projection result $\mathbf{Y} = P(X)$, Code $\mathbf{Z}$
Calculate S, refer to Eq. (4).
Initial post-tuning matrix U: a m-by-n matrix of ones.
for p =1:m **do**
  $Q \leftarrow \bar{z}_p \cdot \bar{z}_p^T$
  $O \leftarrow [(U \circ Z)_{\backslash p}]^T \cdot [U \circ Z)_{\backslash p}]$
  $C \leftarrow Q \circ (S - \gamma O)$
  the principal diagonal of C is set to 0: $diag(C) = 0$
  for q =1:n **do**
    if $|Y_{ij}| < \delta$ and $|-4\gamma \sum_k u_{pk} c_{qk}| > \eta$ **then**
      $u_{pq} \leftarrow sgn(\sum_k u_{pk} c_{qk})$
    end if
  end for
end for
return Code $B = U \circ Z$

3.3 Out-of-Sample Post-Tuning

We have presented a post-tuning algorithm for refining the binary codes of known data X, a key problem then is to generalize the post-tuning algorithm so that it can handle new data, or query, *i.e.*, $q \notin X$. Clearly, the post-tuning of any out-of-sample $q \notin X$ must be consistent with that of X. Therefore, we denote X as ***skeleton points***, since it determines the re-coding rule of post-tuning. The complete post-tuning consists of two steps: 1) post-tune skeleton points X and 2) post-tune out-of-samples in accordance with X.

Recall that the post-tuning of any skeleton point $x_i \in X$ is to reduce the incorrect relationships between the code of x_i and the codes of all other points in X. Considering consistency, the post-tuning of out-of-sample $q \notin X$ should be to reduce the incorrect relationships between the code of



(a) Codes before post-tuning out-of-samples (b) Codes after post-tuning out-of-samples

Figure 3: Toy illustration of the out-of-sample post-tuning. A dataset consists of 10,000 2-d points from 4 different Gaussian distributions is used, and their ITQ codes are pre-computed. We randomly sample 100 points as skeleton points (denoted by $*$) and the rest 9,900 points are used as the out-of-samples (denoted by $\cdot$). The complete post-tuning consists of two steps: (i) post-tune the skeleton points using Eq.(7) (ii) post-tune the rest points using Eq.(12). (a) and (b) show the codes before and after step (ii), respectively. Obliviously, the post-tuning of out-of-samples is consistent with the post-tuning of skeleton points.

q and the codes of X. Denote by z^q the raw binary code of q, the post-tuning of out-of-sample q can be formulated as:

$$\begin{aligned} &min\ \mathcal{R}(u^q) = ||S^q - \frac{1}{m}(u^q \circ z^q)^T B||_F^2 \\ &subject \quad to \qquad u^q \in \{-1, 1\}^m \end{aligned} \tag{12}$$

where S^q denotes the true neighborhood relationships between q and X (refer to Eq.(4)), B denotes the post-tuned codes of X, u^q the post-tuning vector to be learned and $u^q \circ z^q$ the post-tuned code of q. Algorithm 1 with some trivial modifications could be used to solve this problem. Figure 3 illustrates the out-of-sample post-tuning. It shows that the skeleton points act as bridges for rebuilding the neighborhood structure of out-of-samples, and the post-tuning of out-of-samples is consistent with that of skeleton points. Note that the post-tuning is an independent learning procedure performed after hashing, therefore, X could be different from the training set of hashing. In our experiment, we select a small amount of points in the hashing training set to form X considering efficiency. Detailed analysis on skeleton points selection is presented in Sec 4.6.

3.4 Complexity

Space Complexity. The space complexity of PTH is $O(n \times m) + O(k \times l)$, where n is the size of the whole dataset, m the dimension of binary codes, k the number of the skeleton points, l the space of one original feature. The first term is to store binary codes, which is required in all hashing methods; and the second the skeleton points in original features, which is an extra cost in PTH. Since the number of the skeleton points is small and will not increase with the dataset size (refer to Sec 4.6), the extra space cost of PTH is small.

Time Complexity. Given a query, it takes $O(k \times d + k \times m \times m)$ for PTH to generate the binary code. d is the dimension of original feature. It is worth mentioning that $O(k \times m \times m)$ is to compute dot product of binary codes (compute matrix O), which can be accelerated by bit-wise operations. In addition, PTH needs an off-line training for post-tuning skeleton points, which takes $O(k \times k \times d)$. The querying/training time of PTH is evaluated in Sec 4.6, which shows that PTH is very efficient.

4 EXPERIMENT

4.1 Experimental Setup

Datasets All methods are carried out on five noted datasets: CIFAR-10[5], NUS-WIDE[6], USPS[7], MNIST[8] and ANN-GIST1M[9]. We split each dataset into a training set with 10,000 random samples and a test set with all remaining samples. For our post-tuning, we randomly select 1,000 skeleton points from the training set. We repeat the test by 10 times and report the average result. In each test we randomly chose 1, 000 points from the test set as queries.

Baseline Methods Our post-tuning can be combined with most of the unsupervised methods to get a new version. We select some representative methods for evaluation, including Binary Autoencoder Hashing (BAH) [2], Spherical Hashing (SpH) [6], Iterative Quantization (ITQ) [5], Density Sensitive Hashing (DSH) [11], Spectral Hashing (SH) [33], Shift-Invariant Kernel based Locality-Sensitive Hashing (SKLSH) [26] and PCA Hashing(PCAH) [32]. The used codes are provided by corresponding authors and we run them with suggested parameters.

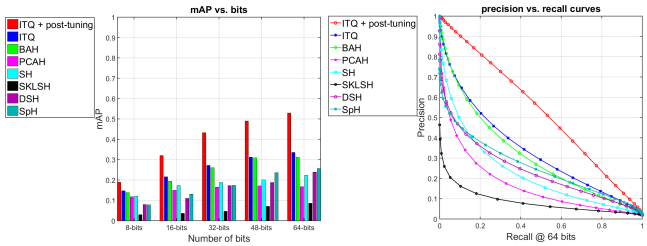
Evaluation Protocols A standard evaluation protocol is to evaluate performance of nearest neighbor search [2, 36]. For each query, the ground truth is its Euclidean neighbors in the original data space, which are determined by a threshold of the average distance to the top 500 nearest neighbor [5, 36]. Based on the ground truth, the precision-recall curve and the mean average precision (mAP) are computed.

4.2 Results on Natural Images

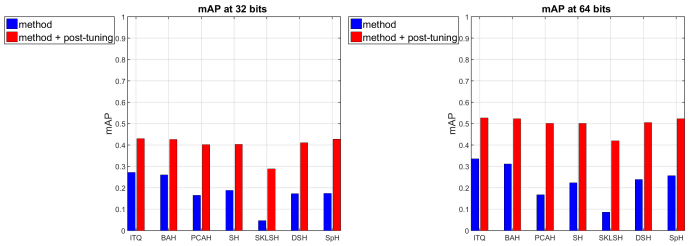
We first report the evaluation results on two natural image datasets, CIFAR-10 [14] and NUS-WIDE [3]. CIFAR-10 contains $60K$ natural images in 10 classes. We ignore the labels and extract a 1024-dimensional feature from each image by using the GoogleNet with batch normalization [8]. NUS-WIDE is a larger dataset which has about $270k$ high-resolution images. We use the provided 225-dimensional block-wise color moments features to represent images.

Figure 4 presents the evaluation results on CIFAR. Specifically, Fig. 4(a) shows the performance comparison between a PTH method, *i.e.*, ITQ + post-tuning, and state-of-the-art hashing methods. We see that the PTH method consistently outperforms these two-stage methods by a large margin. Fig. 4(b) shows the comparison results between methods and their new version with post-tuning. We can see that, after being

[5]http://www.cs.toronto.edu/~kriz/cifar.html
[6]http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
[7]http://www.cs.nyu.edu/~roweis/data.html
[8]http://yann.lecun.com/exdb/mnist/
[9]http://corpus-texmex.irisa.fr/

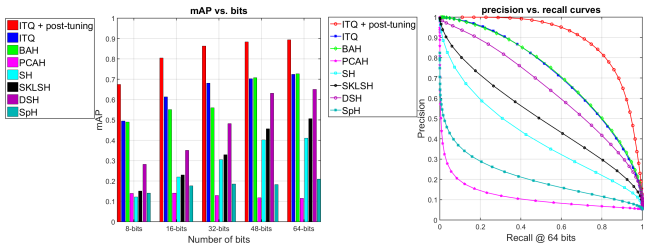


(a) The performance comparison between a PTH method (ITQ + post-tuning, generally performs best) and state-of-the-art hashing methods.

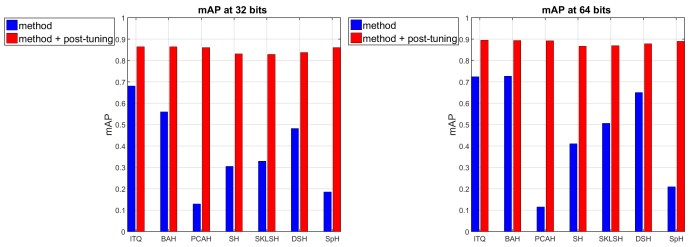


(b) The performance comparison between state-of-the-art hashing methods and their post-tuning version.

Figure 4: Evaluation on CIFAR dataset (1024-dimensional data). In (a), the left and right figure show mAP at different code length and PR curves at 64 bits, respectively. In (b), left and right figure show mAP at 32 bits and at 64 bits, respectively.



(a) The performance comparison between a PTH method (ITQ + post-tuning, generally performs best) and state-of-the-art hashing methods.



(b) The performance comparison between state-of-the-art hashing methods and their post-tuning version.

Figure 5: Evaluation on NUSWIDE dataset (225-dimensional data). In (a), the left and right figure show mAP at different code length and PR curves at 64 bits, respectively. In (b), left and right figure show mAP at 32 bits and at 64 bits, respectively.

post-tuned, the mAP of these method can increase by at least 50% on CIFAR. Figure 5 gives the evaluation results on NUS-WIDE. The post-tuned methods show great performance and outperform the others. Clearly, the post-tuning significantly improves the performance of all methods. With post-tuning, any of these method outperforms ITQ, which is one of the top-performing methods.

4.3 Results on Handwritten Digits

We next present the evaluation results on two handwritten digits datasets, USPS and MNIST. USPS has $11K$ tiny images of 16×16 pixels, each of them is represented by a 256-dimensional vector of pixel values. MNIST consists of $70K$ images of 28×28 pixels, each image is represented by a 784-dimensional pixel vector.

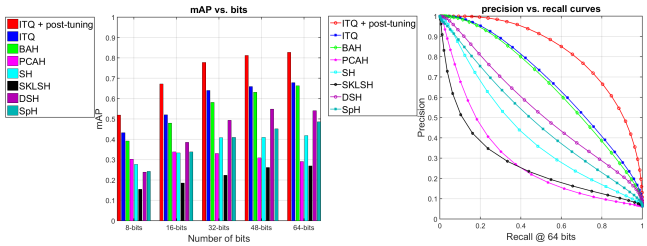
The evaluation results on USPS and MNIST are shown in Figure 6 and 7, respectively. Again, the post-tuned methods outperform all the un-tuned methods in all cases. However, the performance improvement from post-tuning on these two datasets is generally less than that on CIFAR and NUS-WIDE. This behavior is due to the data space difference. The data space of USPS and MNIST is much smaller than the data space of CIFAR and NUS-WIDE, since the data of the former are pixel vectors while the data of the latter are real number features. Therefore, the neighborhood error resulted by mapping data from original data space to hamming space is smaller, and hence the benefit of post-tuning is smaller. This behavior also demonstrates the effectiveness of our post-tuning in minimizing the neighborhood error.

4.4 Results on Large Scale Dataset

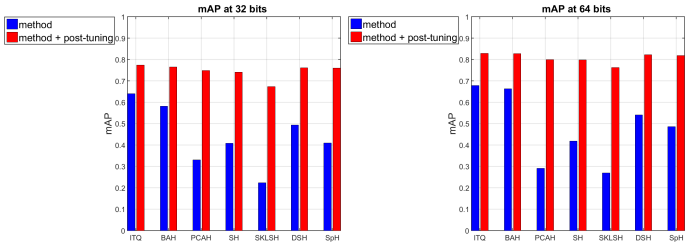
Lastly, we report the evaluation results on a large scale dataset, ANN-GIST1M. ANN-GIST1M consists of 1 million GIST features [25], each of them is a 960-dimensional floating point vector. Figure 8 shows the evaluation results on ANN-GIST1M. As it can be seen, any of the post-tuned methods outperforms the top-performing un-tuned method by a large margin. Compared with the second largest dataset, *i.e.* NUS-WIDE, the performance improvement from post-tuning is even more remarkable on this largest dataset. This phenomenon demonstrates the effectiveness of PTH on large scale dataset. It is worth noting that most of the methods gain more benefit from post-tuning on ANN-GIST1M and CIFAR, which are the two datasets with highest data dimensions, than on other datasets. It demonstrates that PTH is very effective in handing high-dimensional data.

4.5 Comparison with ANN Technology Besides Binary Hashing

Besides binary hashing methods, product quantization (PQ) [10] is a popular solution for efficient nearest (near) neighbor search. In PQ, each data point is represented by several cluster

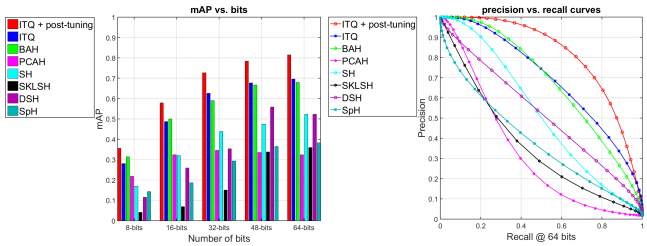


(a) The performance comparison between a PTH method (ITQ + post-tuning, generally performs best) and state-of-the-art hashing methods.

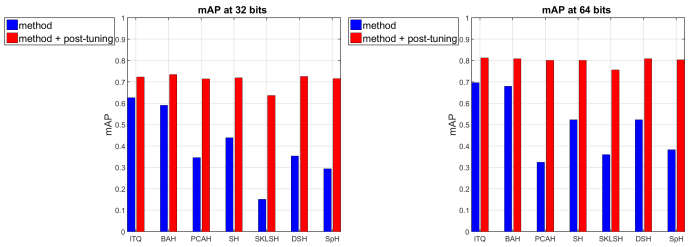


(b) The performance comparison between state-of-the-art hashing methods and their post-tuning version.

Figure 6: Evaluation on USPS dataset (256-dimensional data). In (a), the left and right figure show mAP at different code length and PR curves at 64 bits, respectively. In (b), left and right figure show mAP at 32 bits and at 64 bits, respectively.

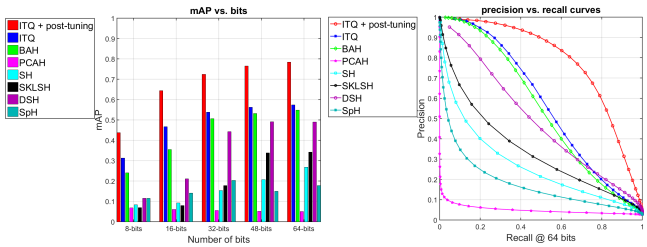


(a) The performance comparison between a PTH method (ITQ + post-tuning, generally performs best) and state-of-the-art hashing methods.

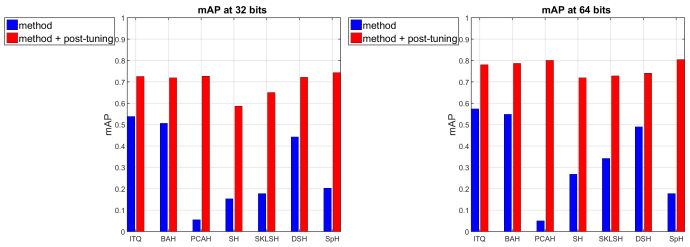


(b) The performance comparison between state-of-the-art hashing methods and their post-tuning version.

Figure 7: Evaluation on MNIST dataset (784-dimensional data). In (a), the left and right figure show mAP at different code length and PR curves at 64 bits, respectively. In (b), left and right figure show mAP at 32 bits and at 64 bits, respectively.



(a) The performance comparison between a PTH method (ITQ + post-tuning, generally performs best) and state-of-the-art hashing methods.



(b) The performance comparison between state-of-the-art hashing methods and their post-tuning version.

Figure 8: Evaluation on ANN-GIST1M dataset (960-dimensional data). In (a), the left and right figure show mAP at different code length and PR curves at 64 bits, respectively. In (b), left and right figure show mAP at 32 bits and at 64 bits, respectively.

centers and the search process is based on Euclidean distances between cluster centers. As the number of distinct distances for PQ is much larger than that for binary hashing methods, the performance of PQ is often better than binary hashing methods [36] [31]. However, even with the help of lookup tables, the search process of PQ is slower [36]. Specifically, the distance computation of PQ (using distance-table lookup for Euclidean distance) is slower than binary hashing methods (using CPU instruction _popcnt for Hamming distance) [31].

We compare PTH (ITQ + post-tuning) with PQ to better validate our method. For PQ, we set the number of cluster centers for each subspace to 256 and use symmetric distance [10]. Table 1 shows the comparison results of top-500 nearest neighbor search. We see that PTH outperforms PQ on all the datasets. The advantage of PTH is even more remarkable on the largest dataset, i.e, GIST1M. As a conclusion, we think PTH is capable of achieving higher performance than PQ.

4.6 Parameter Study

In this subsection, we present the experimental analysis on skeleton points. Skeleton points could be randomly selected from the hashing training set. The question is how many skeleton points are needed? Fig. 9 shows the performance of PTH (32-bits code) with increasing number of skeleton

32-bits code length

method \ dataset	USPS	MNIST	CIFAR	NUSWIDE	GIST1M
• PTH	**0.778**	**0.726**	**0.432**	**0.862**	**0.723**
• PQ	0.774	0.588	0.192	0.724	0.441

48-bits code length

method \ dataset	USPS	MNIST	CIFAR	NUSWIDE	GIST1M
• PTH	**0.812**	**0.784**	**0.489**	**0.883**	**0.765**
• PQ	0.804	0.767	0.395	0.824	0.639

64-bits code length

method \ dataset	USPS	MNIST	CIFAR	NUSWIDE	GIST1M
• PTH	**0.827**	**0.814**	**0.529**	**0.893**	**0.784**
• PQ	0.827	0.811	0.442	0.862	0.677

Table 1: mAP of PTH (ITQ + post-tuning) and PQ

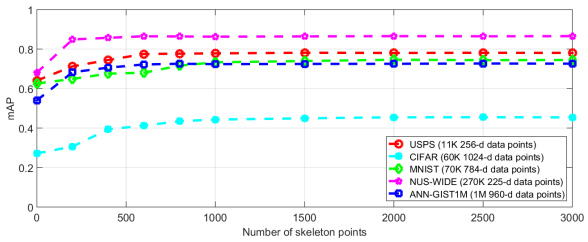


Figure 9: Performance of PTH (ITQ + post-tuning) with different number of skeleton points. Number of skeleton points = 0 means no post-tuning, in this case, PTH = ITQ + no post-tuning = ITQ.

points. We see that the performance gain from the post-tuning increases rapidly and then reach the ceiling. It means the benefit from the neighboring/non-neighboring information offered by skeleton points has an upper bound. The least number of skeleton points leading to the performance ceiling is optimal. Extra skeleton points will result in unnecessary space/time costs.

A following question is that, given a kind of data, will this optimal number (of skeleton points) increase as the dataset size increase? Fig. 9 shows that the optimal numbers for the 5 datasets are close, though their sizes are very different. Therefore, we think the optimal number will not increase with the dataset size. To further validate this conclusion, we construct several datasets of different sizes, and then investigate the optimal numbers for these datasets. These datasets consist of 1M,5M,10M,50M and 80M 384-d GIST features respectively, which are randomly selected from the Tiny Images Dataset[10] (has 80M features in total). The optimal numbers for these datasets are presented in Table 2 (a). We also give the mAP of PTH and ITQ on these datasets, see Table 2 (b). ITQ is selected here since it performs the best in all the baseline methods on these datasets. Table 2 (a) shows that a tiny set of skeleton points, *i.e.* typically 1,000 points, is enough for datasets of different sizes. Table 2 (b) shows that with this tiny set of skeleton points, PTH is capable of achieving high performance.

[10]http://horatio.cs.nyu.edu/mit/tiny/data/

(a)

data set size	1M	5M	10M	50M	80M
	800	1100	700	900	900

(b)

method \ data set size	1M	5M	10M	50M	80M
• PTH	0.57	0.58	0.64	0.72	0.72
• ITQ	0.48	0.51	0.55	0.58	0.59

Table 2: (a) Optimal skeleton point number for PTH with increasing dataset sizes, and (b) mAP of PTH (based on optimal number of skeleton points) and ITQ with increasing dataset sizes, at 32-bits.

Method	Training (offline)	Single query (online)
ITQ	1.1272 (s)	0.05103 (s)
PTH(ITQ + post-tuning)	5.1036 (s)	0.05384 (s)

Table 3: Training time and query time (query time = code generation time + Hamming ranking time) of ITQ and PTH, on ANN-GIST1M at 32 bits.

4.7 Computation Time

PTH needs an extra time for post-tuning skeleton points in the training step, and an extra time for post-tuning the query in the querying step. Fortunately, it is still very fast owing to the tiny number of skeleton points. Table 3 shows the training time and query time of ITQ and PTH (1000 skeleton points). ITQ is selected since it is one of the fastest method in both training and code generation. The results are obtained from MATLAB on a desktop PC with 64GB RAM and an Intel quad-core of 3.3 GHZ using a single core. The post-tuning increases the training time from 1.12 seconds to 5.10 seconds, since it needs about 4 seconds for computing the refined codes of skeleton points. The training is still very fast in practice, considering that it is performed off-line. The post-tuning also increases the query time from 0.051 seconds to 0.053 seconds. The time increment of query is very small, and most importantly, it will not increase with dataset size.

5 CONCLUSION

In this paper, we have proposed a novel "three-stage" hashing model, *i.e.,* projection, binarization and post-tuning, to learn effective binary codes for indexing high-dimensional data. With the proposed model, namely PTH, we show that a neighborhood relationship based post-tuning algorithm can significantly improve the quality of binary codes. Experimental results on five noted datasets demonstrate the superiority of PTH. Moreover, the proposed post-tuning algorithm is a valuable complement to the state-of-the-art methods since it can be applied to them to further refine their binary codes and improve their indexing performance.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (grant No. 2016QY03D0505), and the National Natural Science Foundation of China (grants No. 61525206 and 61502477).

REFERENCES

[1] Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 459–468.

[2] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. 2015. Hashing with binary autoencoders. In *CVPR*. 557–566.

[3] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. 2009. NUS-WIDE: a real-world web image database from National University of Singapore. In *Proceedings of the ACM international conference on image and video retrieval*. ACM, 48.

[4] Cheng Da, Shibiao Xu, Kun Ding, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2017. AMVH: Asymmetric Multi-Valued Hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 736–744.

[5] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2013), 2916–2929.

[6] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. 2015. Spherical hashing: binary code embedding with hyperspheres. *IEEE transactions on pattern analysis and machine intelligence* 37, 11 (2015), 2304–2316.

[7] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[8] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[9] Go Irie, Zhenguo Li, Xiao-Ming Wu, and Shih-Fu Chang. 2014. Locally linear hashing for extracting non-linear manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2115–2122.

[10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117.

[11] Zhongming Jin, Cheng Li, Yue Lin, and Deng Cai. 2014. Density sensitive hashing. *IEEE transactions on cybernetics* 44 (2014), 1362–1371.

[12] Weihao Kong and Wu-Jun Li. 2012. Double-Bit Quantization for Hashing.. In *AAAI*, Vol. 1. 5.

[13] Weihao Kong and Wu-Jun Li. 2012. Isotropic hashing. In *Advances in Neural Information Processing Systems*. 1646–1654.

[14] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[15] Brian Kulis and Trevor Darrell. 2009. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*. 1042–1050.

[16] Brian Kulis and Kristen Grauman. 2012. Kernelized Locality-Sensitive Hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 6 (2012), 1092–1104.

[17] Anan Liu, Yuting Su, Weizhi Nie, and Mohan S Kankanhalli. 2017. Hierarchical Clustering Multi-Task Learning for Joint Human Action Grouping and Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1 (2017), 102–114.

[18] An-An Liu, Wei-Zhi Nie, Yue Gao, and Yu-Ting Su. 2016. Multi-modal clique-graph matching for view-based 3D model retrieval. *IEEE Transactions on Image Processing* 25, 5 (2016), 2103–2116.

[19] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. 2016. Deep Supervised Hashing for Fast Image Retrieval. In *CVPR*.

[20] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *Advances in Neural Information Processing Systems*. 3419–3427.

[21] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2074–2081.

[22] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2011. Hashing with graphs. In *Proceedings of the 28th international conference on machine learning*. 1–8.

[23] Zhi-Quan Luo, Wing-Kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang. 2010. Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine* 27, 3 (2010), 20–34.

[24] Mohammad Norouzi and David M Blei. 2011. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th international conference on machine learning*. 353–360.

[25] Aude Oliva and Antonio Torralba. 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision* 42, 3 (2001), 145–175.

[26] Maxim Raginsky and Svetlana Lazebnik. 2009. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in neural information processing systems*. 1509–1517.

[27] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic hashing. *International Journal of Approximate Reasoning* 50, 7 (2009), 969–978.

[28] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised discrete hashing. In *CVPR*. 37–45.

[29] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2012. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 12 (2012), 2393–2406.

[30] Jun Wang, Wei Liu, Sanjiv Kumar, and Shihfu Chang. 2016. Learning to Hash for Indexing Big Data A Survey. *Proc. IEEE* 104, 1 (2016), 34–57.

[31] J. Wang, T. Zhang, J. Song, N Sebe, and H. T. Shen. 2017. A Survey on Learning to Hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP, 99 (2017), 1–1.

[32] Xin-Jing Wang, Lei Zhang, Feng Jing, and Wei-Ying Ma. 2006. Annosearch: Image auto-annotation by search. In *CVPR*, Vol. 2. IEEE, 1483–1490.

[33] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Advances in neural information processing systems*. 1753–1760.

[34] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. 2014. Supervised Hashing for Image Retrieval via Image Representation Learning.. In *AAAI*, Vol. 1. 2.

[35] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *SIGIR*, Vol. 16.

[36] Lei Zhang, Yongdong Zhang, Jinhui Tang, Xiaoguang Gu, Jintao Li, and Qi Tian. 2013. Topology preserving hashing for similarity search. In *Proceedings of the 21st ACM international conference on Multimedia*. ACM, 123–132.

[37] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. 2015. Deep semantic ranking based hashing for multi-label image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1556–1564.