Hashing with Binary Matrix Pursuit

Fatih Cakir[1], Kun He[2], and Stan Sclaroff[2]

[1] FirstFuel Software, Lexington, MA
`fcakirs@gmail.com`
[2] Department of Computer Science
Boston University, Boston, MA
{`hekun,sclaroff`}`@cs.bu.edu`

Abstract. We propose theoretical and empirical improvements for two-stage hashing methods. We first provide a theoretical analysis on the quality of the binary codes and show that, under mild assumptions, a residual learning scheme can construct binary codes that fit any neighborhood structure with arbitrary accuracy. Secondly, we show that with high-capacity hash functions such as CNNs, binary code inference can be greatly simplified for many standard neighborhood definitions, yielding smaller optimization problems and more robust codes. Incorporating our findings, we propose a novel two-stage hashing method that significantly outperforms previous hashing studies on widely used image retrieval benchmarks.

1 Introduction

A main challenge for "learning to hash" methods lies in the discrete nature of the problem. Most approaches are formulated as non-linear mixed integer programming problems which are computationally intractable. Common optimization remedies include discarding the binary constraints and solving for continuous embeddings [1–5]. At test time the embeddings are typically thresholded to obtain the desired binary codes. However, even the relaxed problem is highly nonconvex requiring nontrivial optimization procedures (*e.g.*, [6]), and the thresholded embeddings are prone to large quantization errors, necessitating additional measures (*e.g.*, [7]).

One prominent alternative to the relaxation approach is *two-stage hashing*, which decomposes the optimization problem into two stages: binary code inference (i) and hash function learning (ii). For a training set, binary codes are inferred in the inference stage, which are then used as target vectors in the hash function learning stage. Such methods closely abide to the discrete nature of the problem as the binary codes are directly incorporated into the optimization procedure. In two-stage hashing, most of the attention is drawn to the more challenging binary code inference step. Typically, this task is itself decomposed into a stage-wise problem where binary codes are learned in an iterative fashion. While theoretical guarantees for the underlying iterative scheme are usually provided, the overall quality of the binary codes is often overlooked. It is desirable to also determine the quality of the constructed binary codes.

In this paper, our first contribution is to provide an analysis on the quality of learned binary codes in two-step hashing. We focus on the frequently considered matrix fitting formulation (*e.g.*, [6, 8–11]), in which a "neighborhood structure" is defined through an affinity matrix and the task is to generate binary codes so as to preserve the affinity values. We first demonstrate that ordinary Hamming distances are unable to fully preserve the neighborhood. Then, with a weighted Hamming metric, we prove that a residual learning scheme can construct binary codes that can preserve any neighborhood with arbitrary accuracy under mild assumptions. Our analysis reveals that distance scaling, as well as fixing the dimensionality of the Hamming space, which are often employed in many hashing studies [6, 12–14], are both unnecessary.

On the other hand, one common inconvenience in two-stage hashing methods is that, steps (i) and (ii) are often interleaved, so as to enable bit correction during training [11, 15, 16]. Bit correction has shown to improve retrieval performance, especially when the hash mapping constitutes simple functions such as linear hyperplanes and decision stumps [9]. In contrast, we show that such an interleaved process is unnecessary with high capacity hash functions such as Convolutional Neural Networks (CNNs).

A further benefit of removing interleaving is that the affinity matrix can be constructed directly according to the definition of the neighborhood structure, instead of the pairwise similarities between training instances. For example, when preserving semantic similarity, the neighborhood is generally defined through class label agreement. Defining the affinity with respect to labels rather than instances yields a much smaller optimization problem for the inference task (i), and provides robustness for the subsequent hash function learning (ii). In contrast, instance-based inference schemes result in larger optimization problems, often necessitating subsampling to reduce the scale.

With these insights in mind, we implement our novel two-stage hashing method with standard CNN architectures, and conduct experiments on multiple image retrieval datasets. The affinity matrix in our formulation may or may not be derived from class labels, and can constitute binary or multi-level affinities. In fact, we consider a variety of experiments that include multi-class (CIFAR-10 [17], ImageNet100 [18]), multi-label (NUSWIDE [19]) and unlabeled (22K LabelMe [20]) datasets. We achieve new state-of-the-art performance for all of these datasets. In summary, our contributions are:

1. We provide a technical analysis on the quality of the inferred binary codes demonstrating that under mild assumptions we can fit any neighborhood with arbitrary accuracy. Our analysis is relevant to the formulations used in many two-stage hashing methods (*e.g.*, [8, 9, 11, 21, 22]).
2. We demonstrate that with high-capacity hash functions such as CNNs, the bit correction task is expendable. As a result, binary code inference can be performed on items that directly define the neighborhood, yielding more robust target vectors and improving the retrieval performance. We achieve state-of-the-art performance in four standard image retrieval benchmarks.

2 Related Work

We only review hashing studies most relevant to our problem. For a general survey, please refer to [23].

The two-stage strategy for hashing was pioneered by Lin *et al.* [21] in which the authors reduced the binary code inference task into a series of binary quadratic programming (BQP) problems. The target codes are optimized in an iterative fashion and traditional machine learning classifiers such as Support Vector Machines (SVMs) and linear hyperplanes that fit the target vectors are employed as the hash functions. In [9], the authors proposed a graph-cut algorithm to solve the BQP problem and employed boosted decision trees as the hash functions. The graph-cut algorithm has shown to yield a solution well bounded with respect to the optimal value [24]. The authors also demonstrated that, with shallow models an interleaved process of binary code inference and hash function learning allowed bit correction and improved the retrieval performance. Differently, Xia *et al.* [8] proposed using a coordinate descent algorithm with Newton's method to solve the BQP problem and utilized CNNs as the hash mapping. Do *et al.* [11] solved the the BQP problem using semidefinite relaxation and Lagrangian approaches. They also investigate the quality of the relaxed solution and prove that it is within a factor of the global minimum. Zhuang *et al.* [22] demonstrated that the same BQP approach can be extended to solve a triplet-based loss function. Other work reminiscent of these two-stage methods include hashing techniques that employ alternating optimization to minimize the original optimization problem [10,15,16,25].

While error-bounds and convergences properties of the underlying iterative scheme is usually provided, none of the aforementioned studies provide a technical guarantee on the overall quality of the constructed binary codes. In this study we provide such an analysis. Our technical analysis has connections to low-rank matrix learning [26–29] in which we construct binary codes in a gradient descent or *matrix pursuit* methodology. Differently, we constrain ourselves with binary rank-one matrices, which are required for Hamming distance computations. Also, while not all two-stage hashing studies follow an interleaved process (*e.g.*, [21,30,31]), to the best of our knowledge, all construct the affinity matrix using training instances. This warrants an in-depth look to the necessity of such a process when high-capacity hash functions are employed.

Our hashing formulation follows the matrix fitting formulation which is almost exclusively used in two-stage methods. This formulation was originally proposed in [6] and has been widely adopted in subsequent hashing studies (*e.g.*, [8,9,11,21,32]). Whereas the major contribution in this paper lies in establishing convergence properties of the binary code inference task, our formulation also has subtle and key differences to [6] and other two-stage methods. Specifically, we allow weighted hamming distances with optimally learned weights given the inferred binary codes. We perform inference directly on items that define the neighborhood, enabling more robust target vector construction as will be shown. In retrieval experiments, we compare against recent hashing studies, including [4,14,16,33–39], and achieve state-of-the-art performances.

3 Formulation

In this section, we first discuss the two stages of our hashing formulation: binary code inference and hash mapping learning. An analysis on affinity matrix construction comes next. All proofs are provided in the supplementary material.

3.1 Binary Code Inference

In this section, we explain our inference step (i). We are given a metric space $(\mathcal{X}, d)$ where $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ denotes a set of items and $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a metric. Note that $\mathbf{x}$ can correspond to instances, labels, multi-labels or any item that is involved in defining the neighborhood. Given the assumption that the neighborhood is defined through metric d, we learn the hash mapping $\Phi : \mathcal{X} \to \mathbb{H}^b$ by optimizing the *neighborhood preservation fit*:

$$\min_{\Phi} \sum_{i,j} [\gamma d(\mathbf{x}_i, \mathbf{x}_j) - d_h(\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j))]^2, \qquad (1)$$

where d_h is the Hamming distance and γ is a suitably selected scaling parameter. In order to scale distances to the range of d_h, we set $\gamma = b/d_{\max}$ where $d_{\max} = \max_{\mathbf{x},\mathbf{y} \in \mathcal{X}} d(\mathbf{x}, \mathbf{y})$ is known.[3] Solving Eq. 1 entails discrete loss minimization, which in general is a non-linear mixed-integer programming problem. Instead, two-stage methods decompose the solution into two steps, the first involving a binary integer program to find a set of binary codes, or auxilliary variables $\{\mathbf{u}_i \in \mathbb{H}^b\}_{i=1}^n$ that minimize Eq. 1. This program can be formulated as:

$$\min_{\mathbf{u}} \sum_{i,j} [\gamma d(\mathbf{x}_i, \mathbf{x}_j) - d_h(\mathbf{u}_i, \mathbf{u}_j)]^2 = \min_{\mathbf{u}} \frac{1}{4} \sum_{i,j} [\mathbf{u}_i^\top \mathbf{u}_j - s(\mathbf{x}_i, \mathbf{x}_i)]^2, \qquad (2)$$

where $s(\mathbf{x}_i, \mathbf{x}_j) = b - 2\gamma d(\mathbf{x}_i, \mathbf{x}_j), \forall i, j \in \mathcal{X}$. While the LHS of Eq. 2 is a distance equivalence problem, the RHS is an affinity matching task. Such affinity based preservation objectives have also been considered previously [1, 6, 14, 33].

In our formulation, we consider weighted Hamming distances by weighting each bit in $\mathbf{u}$. The weighted Hamming distance has been used in past studies to provide more granular similarities compared to its unweighted counterpart (*e.g.*, [40–44]). While this hashing scheme still enjoys low memory footprint and fast distance computations, weighting the individual bits enables us to construct binary codes that better preserve affinity values, as will be shown later.

We reformulate Eq. 2 by defining weight vector $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_b]^\top$:

$$\frac{1}{4} \sum_{i,j} [(\boldsymbol{\alpha} \odot \mathbf{u}_i)^\top \mathbf{u}_j - s(\mathbf{x}_i, \mathbf{x}_i)]^2 \propto \frac{1}{2} \|\mathcal{U} - \mathcal{R}\|_F^2 = f(\mathcal{U}), \qquad (3)$$

where $\odot$ denotes the Hadamard product, $\mathcal{U}_{ij} = (\boldsymbol{\alpha} \odot \mathbf{u}_i)^\top \mathbf{u}_j, \mathcal{R}_{ij} = s(\mathbf{x}_i, \mathbf{x}_j), \forall i, j \in \mathcal{X}$ and $\|\cdot\|_F$ denotes the Frobenius norm. We note that the *affinity matrix* $\mathcal{R}$ is real and symmetric as per its construction from metric d.

[3] Our later analysis in this section only requires $d_{\max}$ to be bounded.

Let $\mathbf{V} = [\mathbf{u}_1, \cdots, \mathbf{u}_n]^\top \in \mathbb{H}^{n\times b}$ denote the *binary code matrix*, then $\mathcal{U}$ can be written as the weighted sum of b rank-one matrices $\sum_{k=1}^{b} \alpha_k \mathbf{v}_k \mathbf{v}_k^\top$ where $\mathbf{v}_k \in \{-1, 1\}^n$ is the k-th column in $\mathbf{V}$. Given this fact, our binary inference problem can be reformulated as:

$$\min \ f(\mathcal{U}), \quad \text{s.t. } \mathcal{U} = \sum_{k=1}^{b} \alpha_k \mathbf{v}_k \mathbf{v}_k^\top, \ \mathbf{v} \in \{-1, +1\}^n. \tag{4}$$

The additive property of $\mathcal{U}$ is attractive, since it suggests that the problem could be solved by a *stepwise* algorithm that adds the $\mathbf{v}_k$'s one by one. In particular, we will apply the projected gradient descent algorithm to solve Eq. 4. Starting with an initial value, $\mathcal{U}_0 = \mathbf{0}$, an update step can be formulated as:

$$\mathcal{U}_t \leftarrow \mathcal{U}_{t-1} + \alpha_t \mathbf{v}_t \mathbf{v}_t^\top, \tag{5}$$

where

$$\mathbf{v}_t = \underset{\mathbf{v}\in\{-1,+1\}^n}{\arg\max} \ \langle \mathbf{v}\mathbf{v}^\top, -\nabla f(\mathcal{U}_{t-1})\rangle \tag{6}$$

finds the projection of the negative gradient direction $-\nabla f(\mathcal{U}_{t-1})$ in the subspace spanned by rank-one binary matrices, and α_t is a step size. This projection is important for maintaining the additive property in Eq. 4.

Since $\langle \mathbf{v}\mathbf{v}^\top, \nabla f\rangle = \mathbf{v}^\top \nabla f \mathbf{v}$, Eq. 6 is a BQP problem which in general is NP-hard. Here, we take a spectral relaxation approach which is also used in past methods (*e.g.*, [6, 21, 33]). A closed-form solution to Eq. 6 exists if the binary vector $\mathbf{v}$ is relaxed to continuous values. Specifically, if $Q = -\nabla f(\mathcal{U})$, the following relaxation yields the Rayleigh Quotient [45]:

$$\max_{\mathbf{v}^\top \mathbf{v} = n} \ \mathbf{v}^\top Q \mathbf{v} = n\lambda_{\max}(Q), \tag{7}$$

where $\lambda_{\max}$ denotes the largest eigenvalue, and the optimal solution, $\mathbf{v}^*$, is the corresponding eigenvector. The binarized value of $\mathbf{v}^*$, $\text{sgn}(\mathbf{v}^*)$, is an approximate solution for Eq. 6. This solution can optionally be used as an initial point for BQP solvers in further maximizing Eq. 6, (*e.g.*, [46–48]). Note that the main technical results to be given are independent of the particular BQP solver.

The negative gradient $-\nabla f(\mathcal{U}_{t-1}) = \mathcal{R} - \sum_{k=1}^{t-1} \mathbf{v}_k \mathbf{v}_k^\top$, also a symmetric matrix, can be considered as the *residual* at iteration $t-1$. At each iteration, we find the most correlated rank-one matrix with this residual and move our solution in that direction. If the step size α_t is set to 1 for all t, then $\mathcal{U}$ can be decomposed as the product of the binary code matrices $\mathbf{V}\mathbf{V}^\top$, yielding ordinary Hamming distances. However, with constant step sizes, the below property states that there exist certain affinity matrices $\mathcal{R}$ such that no $\mathcal{U}$ exists that *fits* $\mathcal{R}$.

Property 1. *Let Q_t be the residual $-\nabla f(\mathcal{U}_t)$ at iteration t. There exists a $\mathcal{R}$ such that $\forall t, \|Q_t\|_F > 0$.*

Such a result motivates us to relax the constraint on the step size parameter α_t. If α is relaxed to any real value, then what we have essentially is weighted Hamming distances and we demonstrate that one can monotonically decrease the residual $\mathcal{R}$ in this case. We now provide our main theorem:

Theorem 2. *If $\alpha_t \in \mathbb{R}$, then the gradient descent algorithm Eq. 5 - 6 satisfies*

$$\|Q_t\|_F \leq \eta^{t-1}\|Q_{t-1}\|_F, \quad \forall t \tag{8}$$

where $\eta \in [0,1]$.

Theorem 2 states that the norm of the residual is only monotonically non-increasing. However, it may not strictly decrease, since the solution $\mathbf{v}_t$ of Eq. 6 can actually be orthogonal to the gradient, *i.e.*, $\mathbf{v}_t^\top Q_{t-1}\mathbf{v}_t$ might be zero. If we ensure non-orthogonal directions are selected at each iteration, then the residual strictly decreases, as the following corollary states.

Corollary 3. *If $\mathbf{v}_t^\top Q_{t-1}\mathbf{v}_t \neq 0, \ \forall t$ then the residual norm $\|Q_t\|_F$ strictly decreases.*

Although the directions $\mathbf{v}_t\mathbf{v}_t^\top$ are greedily selected with step sizes α_t, one can refine step sizes of all past directions at each iteration. This generally leads to much faster convergence. More formally, we can refine the step size parameters by solving the following regression problem:

$$\boldsymbol{\alpha}^* = \underset{\alpha_1,\cdots,\alpha_t}{\arg\min} \frac{1}{2}\|\sum_{k=1}^{t}\alpha_k\mathbf{v}_k\mathbf{v}_k^\top - \mathcal{R}\|_F^2. \tag{9}$$

Fortunately, Eq. 9 is an ordinary least squares problem admitting a closed-form solution. Let $\widehat{\mathbf{v}}_k = \text{vec}(\mathbf{v}_k\mathbf{v}_k^\top)$ and $\widehat{\mathbf{r}} = \text{vec}(\mathcal{R})$ where $\text{vec}(\cdot)$ denotes the vectorization operator. Given $\widehat{\mathbf{V}}_t = [\widehat{\mathbf{v}}_1, \cdots, \widehat{\mathbf{v}}_t]$, the minimizer of Eq. 9 is

$$\boldsymbol{\alpha}_t^* = (\widehat{\mathbf{V}}_t^\top\widehat{\mathbf{V}}_t)^{-1}\widehat{\mathbf{V}}_t^\top\widehat{\mathbf{r}}, \tag{10}$$

where $\boldsymbol{\alpha}_t^* = [\alpha_1^*, \cdots, \alpha_t^*]^\top$. The solution requires $\mathcal{O}(t^3)+\mathcal{O}(t^2n^2)+\mathcal{O}(tn^2)$ operations with $n = |\mathcal{X}|$. If $n > \sqrt{t}$, the time complexity is dominated by the $\mathcal{O}(t^2n^2)$ term. Note that in practice, typical values for t, the number of bits, are small (< 100) and can be considered a constant factor.

We now provide a property indicating that this refinement of the step-sizes does not break the monotonicity as defined in Theorem 2 and Corollary 3.

Property 4. *Let Q_t be the residual matrix at iteration t and α_t set according to Theorem 2. Let $\widehat{Q}_t$ be the residual after refining the step-sizes $\boldsymbol{\alpha_t} = [\alpha_1, \cdots, \alpha_t]^\top$ using Eq. 10. Then $\|\widehat{Q}_t\|_F \leq \|Q_t\|_F$.*

After learning $\mathcal{U} = \sum_{k=1}^{\top}\alpha_k\mathbf{v}\mathbf{v}^t = \mathbf{A}\odot\mathbf{V}\mathbf{V}^\top$ where $A_{k,\cdot} = [\alpha_1, \cdots, \alpha_t], \forall k$ we obtain our binary code matrix $\mathbf{V} = [\mathbf{u}_1, \cdots, \mathbf{u}_n]^\top$ that contains the target codes for each element $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \in \mathcal{X}$. This ends our inference step (i). We summarize our inference scheme in Alg. Binary code inference.

Remarks. We consider two different binary inference schemes: `constant` where the binary codes are constructed with constant step sizes yielding ordinary Hamming distances; and, `regress` where each bit is weighted yielding the weighted Hamming distance. For `regress`, since $(\boldsymbol{\alpha}\odot\mathbf{u}_i)^\top\mathbf{u}_j = b(1-2d(\mathbf{x}_i,\mathbf{x}_j)/d_{\max})$ in Eq. 3, we can embed the constant b into the weight vector variable $\boldsymbol{\alpha}$. As a result, in contrast to hashing methods where the Hamming space dimensionality b must be specified (*e.g.*, to set margin and scaling parameters [6, 10, 14]), our

Algorithm: Binary code inference

```
input  : $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$. Boolean variable regress.
         (Optional) Procedure Improve($Q, \mathbf{v}_0$) to improve the solution $\mathbf{v}^\top Q \mathbf{v}$
         s.t. $\mathbf{v} = \{-1, 1\}^n$ where $\mathbf{v}_0$ is an initial solution. $\mathcal{U} = \mathbf{0}$.
output: Code matrix $\mathbf{V} = [\mathbf{u}_1, \cdots, \mathbf{u}_n]^\top$, weight vector $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_T]^\top$
1  $\gamma = \frac{1}{d_{\max}}$, $\mathcal{R}_{ij} = 1 - 2\gamma d(\mathbf{x}_i, \mathbf{x}_j)$ (if regress), $\mathcal{R}_{ij} = s(\mathbf{x}_i, \mathbf{x}_j) \times b$ (if $\neg$regress)
2  for $t \leftarrow 1, ..., T$ do
3      $\mathbf{v}_t \leftarrow$ eigenvector corresponding to the largest eigenvalue of $-\nabla f(\mathcal{U}_{t-1})$
4      $\mathbf{v}_t \leftarrow \mathrm{sgn}(\mathbf{v}_t)$, $\alpha_t \leftarrow 1$
5      $\mathbf{v}_t \leftarrow$ Improve($-\nabla f(\mathcal{U}_{t-1}), \mathbf{v}_t$) (optional)
6      if regress then // $\alpha_t \in \mathbb{R}$
7          Set $[\alpha_1, \cdots, \alpha_t]^\top$ using Eq. 10
8      end
9      $\mathcal{U}_t \leftarrow \sum_t \alpha_t \mathbf{v}_t \mathbf{v}_t^\top$, $V_{\cdot,t} \leftarrow \mathbf{v}_t$ // Append $\mathbf{v}_t$ to $\mathbf{V}$
10 end
```

method only requires $d_{\max}$ to be bounded. On the other hand, regular Hamming distance, or `constant`, requires scaling with b beforehand. The approximate solution of Eq. 7 can be improved by using off-the-shelf BQP solvers. In Alg. Binary code inference, we refer to such solvers as the subroutine `Improve`$(\cdot)$. In this paper, we consider using a simple heuristic [46], which merely requires a positive objective value for Eq. 6.

We now proceed with step (ii): hash mapping learning.

3.2 Hash Mapping Learning

Recall that we inferred target codes $\mathbf{u} \in \mathbb{H}^b$ for each item $\mathbf{x} \in \mathcal{X}$, where $\mathbf{x}$ may correspond to data instances, classes, multi-labels *etc.*, depending on the neighborhood definition. For example, when the dataset is unsupervised and the neighborhood is defined merely through data instances, then $\mathcal{X}$ may correspond to the feature space with $d(\mathbf{x}_i, \mathbf{x}_j)$ being the Euclidean distance. For multi-class datasets, $\mathcal{X}$ and $d(\mathbf{x}_i, \mathbf{x}_j)$ may represent the set of classes and the distance values between pairs of classes, respectively. For multi-label datasets, $\mathcal{X}$ may correspond to the set of possible label combinations. Our binary inference scheme constructs target codes to items that *directly* define the neighborhood. In the experiments section, we cover various scenarios.

If $\mathcal{X}$ does not represent the feature space, then after the binary inference step (i), the target codes get assigned to data instances in a one-to-many fashion, depending on the relationship between the target code and data instance. For sake of clarity, we assume $\mathcal{X}$ is the feature space in this section.

We employ a collection of hash functions to learn the mapping, where a function $f : \mathcal{X} \to \{-1, 1\}$ accounts for the generation of a bit in the binary code. Many types of hash functions are considered in the literature. For simplicity, we consider the thresholded scoring function:

$$f(\mathbf{x}) \triangleq \mathrm{sgn}(\psi(\mathbf{x})), \tag{11}$$

where ψ can be either a shallow model such as a linear function, or a deep neural network. In experiments, we consider both types of embeddings. $\Phi(\mathbf{x}) = [f_1(\mathbf{x}), \cdots, f_b(\mathbf{x})]^\top$ then becomes a vector-valued function to be learned.

Recall that we inferred target codes $\mathbf{u} \in \mathbb{H}^b$ for each element $\mathbf{x} \in \mathcal{X}$. Having the target codes at our disposal, we now would like to find Φ such that the Hamming distances between $\Phi(\mathbf{x})$ and the corresponding target codes $\mathbf{u}$ are minimized. Hence, the objective can be formulated as:

$$\sum_{i=1}^{n} d_h(\Phi(\mathbf{x}_i), \mathbf{u}_i). \tag{12}$$

The Hamming distance is defined as $d_h(\Phi(\mathbf{x}_i), \mathbf{u}_i) = \sum_t [\![f_t(\mathbf{x}_i) \neq u_{it}]\!]$ where both d_h and the functions f_t are non-differentiable. Fortunately, we can relax f_t by dropping the *sgn* function in Eq. 11 and derive an upper bound on the Hamming loss. Note that $d_h(\Phi(\mathbf{x}_i), \mathbf{u}_i) = \sum_t [\![f_t(\mathbf{x}_i) \neq u_{it}]\!] \leq \sum_t l(-u_{it}\psi_t(\mathbf{x}_i))$ with a suitably selected convex margin-based function l. Thus, by substituting this surrogate function into Eq. 12, we can directly minimize this upper bound using stochastic gradient descent. We use the hinge loss as the upper bound l.

As similar to other two-stage hashing methods, at the heart of our formulation are the target vectors which are inferred as to fit the affinity matrix $\mathcal{R}$. Next, we take a closer look on how to construct this affinity matrix.

3.3 Affinity Matrix Construction

The affinity matrix can be defined through pairwise similarities of items that directly define the neighborhood, which may not correspond to training instances. Despite this flexibility of the formulation, previous related hashing studies generally consider using training instances.

For certain neighborhoods, constructing the affinity matrix with training instances might yield suboptimal binary codes. To illustrate this case, consider Fig. 1 where we compare two sets of binary codes inferred from two different affinity matrices in a series of experiments. The neighborhood definition in these experiments is a standard one, typically found in nearly all hashing work. Specifically, we assume 10 classes and define the class affinity matrix as shown in Fig. 1 (a). We also consider a hypothetical set of 1000 instances, each assigned to one of these 10 classes, and construct the affinity matrix as shown in Fig. 1 (b) which we simply refer as the instance affinity matrix. Similarity of the instances are based on their class id's and deduced from the class affinity matrix. We infer binary codes under varying lengths as to reconstruct the class and instance affinity matrices. As explained in Section 3.2, instances are assigned the binary code of their respective classes for the class based inference. The experiments are repeated 5 times and average results are reported.

We first highlight the residual matrix Q_t norm in Fig. 1 (c). Note that the residual norm of the class based inference converges to zero with fewer iterations: 40 bit codes are able to reconstruct the class affinity matrix with minimal discrepancy. On the other hand, lengthier codes are required to fully reconstruct

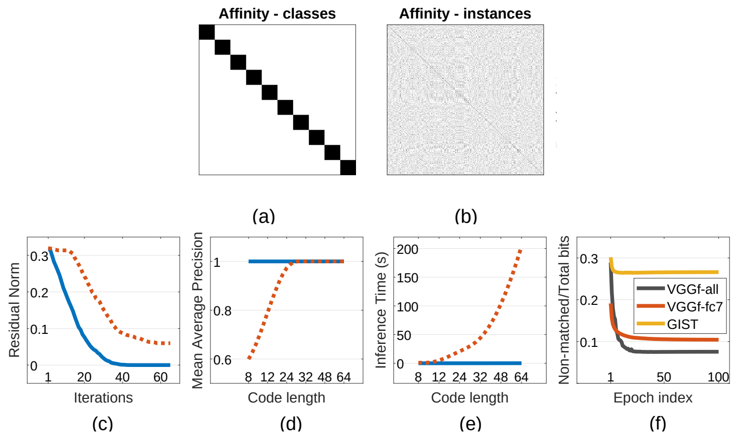

Fig. 1: In a series of experiments, we compare two sets of binary codes constructed with two different affinity matrices: class (a) and instance based (b). (c)-(e) contrasts the binary codes with respect to residual norm, mAP and inference time. Results for binary codes inferred from the class and instance affinity matrices are denoted with (━) and (⋯), respectively. We also learn hash functions with varying complexities to fit the inferred binary codes and plot the fraction of non-matched bits to the total number bits (f).

the instance affinity matrix. We also provide the retrieval performance for the two sets of binary codes. Mean Average Precision (mAP) is the evaluation criterion. For this experiment, 100 instances are sampled from the instance set as queries, while the rest constitute the retrieval set. As demonstrated in Fig. 1 (d), their is a dramatic difference in mAP values especially with compact codes. The difference can be as large as 0.40. This type of sub-optimality for the binary codes inferred through the instance affinity matrix have also been observed previously (*e.g.*, [22]). Lastly, Fig. 1 (e) gives the training time for the two inference schemes. While the inference time depends on the particular BQP solver, the number of decision variables nevertheless scales quadratically with the number of items in $\mathcal{X}$, as seen by the dramatic difference in the training time between the two inference schemes, especially with lengthier codes. Depending on the instance matrix size, the difference can easily scale up requiring subsampling to reduce the scale of the optimization task.

Given the evident disadvantages, why is the affinity matrix constructed from instances? The primary reason is because in most two-stage hashing methods the inference and hash function learning steps are interleaved for *on-the-fly* bit correction purposes. This requires the affinity matrix to correspond to pairwise instance similarities as the inferred bits will *immediately* be used for training the hash functions. However, given recent advances in deep learning, high-capacity predictors are becoming available, nullifying the need for bit correction. Consequently, one can opt to solve a smaller and more robust optimization problem defined on items that directly define the neighborhood.

To illustrate this point we learn hash functions of varying complexities to fit the set of binary codes,[4] and plot the fraction of non-matched bits to total number bits during hash function learning. We use the training set of CIFAR-10 and train the hash functions to fit the inferred 32-bit binary codes (total: $32 \times 50{,}000$ bits). We consider single layer neural networks on GIST [49] and fc7 features of a VGG-F network [50] pretrained on ImageNet [19], in addition to fine-tuning all the VGG-F layers. Fig. 1 (f) gives the results. Notice that as the capacity of the hash function increases the ratio of non-matched bits decrease significantly. While this ratio is above 0.25 with a single layer neural net on GIST, the single layer neural net trained on fc7 features yields just above 10% unmatched bits. When we fine-tune all layers of a VGG-F network this percentage reduces well below 10%. We can induce that with more complex architectures the ratio will diminish even more so.

We incorporate these insights into our formulation and conduct retrieval experiments against competing methods in the next section, where we achieve new state-of-the-art performances.

4 Experiments

We conduct experiments on widely used image retrieval benchmarks: CIFAR-10 [17], NUSWIDE [18], 22K LabelMe [20] and ImageNet100 [19].

CIFAR-10 is a dataset for image classification and retrieval, containing 60K images from 10 different categories. We follow the setup of [2, 14, 22, 38]. This setup corresponds to two distinct partitions of the dataset. In the first case (*cifar-1*), we sample 500 images per category, resulting in 5,000 training examples to learn the hash mapping. The test set contains 100 images per category (1000 in total). The remaining images are then used to populate the hash table. In the second case (*cifar-2*), we sample 1000 images per category to construct the test set (10,000 in total). The remaining items are both used to learn the hash mapping and populate the hash table. Two images are considered neighbors if they belong to the same class.

NUSWIDE is a dataset containing 269K images. Each image can be associated with multiple labels, corresponding with 81 ground truth concepts. Following the setup in [2, 14, 22, 38], we only consider images annotated with the 21 most frequent labels. In total, this corresponds to 195,834 images. The experimental setup also has two distinct partitionings: *nus-1* and *nus-2*. For both cases, a test set is constructed by randomly sampling 100 images per label (2,100 images in total). To learn the hash mapping, 500 images per label are randomly sampled in *nus-1* (10,500 in total). The remaining images are then used to populate the hash table. In the second case, *nus-2*, all the images excluding the test set are used in learning the hash mapping and populating the hash table. Two images are considered neighbors if they share a single label. We also specify a richer

[4] These binary codes are obtained from the class based inference scheme, though similar behavior is exhibited with codes obtained with instance based inference.

neighborhood by allowing multi-level affinities. In this scenario, two images have an affinity value equal to the number of common labels they share.

22K LabelMe consists of 22K images, each represented with a 512-dimensionality GIST descriptor. Following [3, 12], we randomly partition the dataset into two: a training and test set consisting of 20K and 2K instances, respectively. A 5K subset of the training set is used in learning the hash mapping. As this dataset is unsupervised, we use the l_2 norm in determining the neighborhood. Similar to NUSWIDE, we allow multi-level affinities for this dataset. We consider four distance percentiles deduced from the training set and assign multi-level affinity values between the instances.

ImageNet100 is a subset of ImageNet [19] containing 130K images from 100 classes. We follow [4] and sample 100 images per class for training. All images in the selected classes from the ILSVRC 2012 validation set are used as the test set. Two images are considered neighbors if they belong to the same class.

Experiments without using multi-level affinities in defining the neighborhood are evaluated using a variant of Mean Average Precision (mAP), depending on the protocol we follow. We collectively group these as *binary affinity* experiments. *Multi-level affinity* experiments are evaluated using Normalized Discounted Cumulative Gain (NDCG), a metric standard in information retrieval for measuring ranking quality with multi-level similarities. In both experiments, Hamming distances are used to retrieve and rank data instances.

We term our method HBMP (**H**ashing with **B**inary **M**atrix **P**ursuit), and compare it against state-of-the-art hashing methods. These methods include: Spectral Hashing (**SH**) [33], Iterative Quantization (**ITQ**) [34], Supervised Hashing with Kernels (**SHK**) [6], Fast Hashing with Decision Trees (**FastHash**) [9], Structured Hashing (**StructHash**) [37], Supervised Discrete Hashing (**SDH**) [16], Efficient Training of Very Deep Neural Networks (**VDSH**) [36], Deep Supervised Hashing with Pairwise Labels (**DPSH**) [38], Deep Supervised Hashing with Triplet Labels (**DTSH**) [14] and Mutual Information Hashing (**MIHash**) [39, 51]. These competing methods have been shown to outperform earlier and other works such as [1, 2, 8, 12, 13, 41, 52].

For CIFAR-10 and NUSWIDE experiments, we fine tune the VGG-F architecture. For ImageNet100 experiments, we fine-tune the AlexNet architecture. Both deep learning models are pretrained using the ImageNet dataset. For *non-deep* methods, we use the output of the penultimate layer of both architectures. For the 22K LabelMe benchmark, we learn shallow models on top of the GIST descriptor. For deep learning based hashing methods, this corresponds to using a single fully connected neural network layer.

4.1 Results

We provide results for experiments with *binary similarities* with mAP as the evaluation criterion, and then for *multi-level similarities* with NDCG. In CIFAR-10, set $\mathcal{X}$, in which the binary inference is performed upon, represents the 10 classes. For NUSWIDE, as the neighborhood is defined using the multi-labels, it is

VGG – F	CIFAR – 10 (mAP)				NUSWIDE (mAP@5K)			
Method	12 Bits	24 Bits	32 Bits	48 Bits	12 Bits	24 Bits	32 Bits	48 Bits
SH [33]	0.183	0.164	0.161	0.161	0.621	0.616	0.615	0.612
ITQ [34]	0.237	0.246	0.255	0.261	0.719	0.739	0.747	0.756
SHK [6]	0.488	0.539	0.548	0.563	0.768	0.804	0.815	0.824
SDH [16]	0.478	0.557	0.584	0.592	**0.780**	0.804	0.816	0.824
FastHash [9]	0.553	0.607	0.619	0.636	0.779	0.807	0.816	0.825
StructHash [37]	0.664	0.693	0.691	0.700	0.748	0.772	0.790	0.801
VDSH [36]	0.538	0.541	0.545	0.548	0.769	0.796	0.803	0.807
DPSH [38]	0.713	0.727	0.744	0.757	0.758	0.793	0.818	0.830
DTSH [14]	0.710	0.750	0.765	0.774	0.773	0.813	0.820	0.838
MIHash [39]	0.738	0.775	0.791	0.816	0.773	**0.820**	**0.831**	**0.843**
HBMP	**0.799**	**0.804**	**0.830**	**0.831**	0.757	0.805	0.822	0.840

Table 1: Binary affinity experiments on CIFAR-10 and NUSWIDE datasets with *cifar-1* and *nus-1* partitionings. The underlying deep learning architecture is VGG-F. HBMP outperforms competing methods on CIFAR-10, and shows improvements, especially with lengthier codes on NUSWIDE.

VGG – F	CIFAR – 10 (mAP)				NUSWIDE (mAP@50K)			
Method	16 Bits	24 Bits	32 Bits	48 Bits	16 Bits	24 Bits	32 Bits	48 Bits
DRSH [52]	0.608	0.611	0.617	0.618	0.609	0.618	0.621	0.631
DRSCH [53]	0.615	0.622	0.629	0.631	0.715	0.722	0.736	0.741
DPSH [38]	0.903	0.885	0.915	0.911	0.715	0.722	0.736	0.741
DTSH [14]	0.915	0.923	0.925	0.926	0.756	0.776	0.785	0.799
MIHash [39]	0.927	0.938	0.942	0.943	0.798	0.814	0.819	0.820
HBMP	**0.942**	**0.944**	**0.945**	**0.945**	**0.804**	**0.829**	**0.841**	**0.855**

Table 2: Binary affinity experiments on CIFAR-10 and NUSWIDE datasets with *cifar-2* and *nus-2* partitionings (with VGG-F architecture). HBMP achieves new state-of-the-art performances, significantly improving over competing methods.

then intuitive for set $\mathcal{X}$ to represent label combinations. In our case, we consider unique label combinations in the training set resulting in $\mathcal{X} = 4850$ items for binary inference. For the 22K LabelMe dataset, the items directly correspond to training instances. We provide results for the `regress` binary inference scheme, denoted simply as HBMP. A comparison between `constant` and `regress` is given in the supplementary material.

Binary Affinity Experiments. Table 1 gives results for the *cifar-1* and *nus-1* experimental settings in which mAP and mAP@5K values are reported for the CIFAR-10 and NUSWIDE datasets, respectively. Deep-learning based hashing methods such as DPSH, DTSH and MIHash outperform most non-deep hashing solutions. This is not surprising as feature representations are simultaneously learned along the hash mapping in these methods. Certain two-stage methods, *e.g.*, FastHash, remain competitive and top deep learning methods including DTSH and MIHash for various hash code lengths, especially for NUSWIDE. Our two-stage method, HBMP, outperforms all competing methods in majority of the cases, including MIHash, DTSH and DPSH with very large improvement mar-

AlexNet	ImageNet100 (mAP@1K)			
Method	16 Bits	32 Bits	48 Bits	64 Bits
HashNet [4]	0.506	0.630	0.663	0.683
MIHash [39]	0.569	0.661	0.685	0.694
HBMP	**0.574**	**0.692**	**0.712**	**0.742**

Table 3: mAP@1K values on ImageNet100 using AlexNet. HBMP outperforms the two state-of-the-art formulations using mutual information [39] and continuation methods [4].

gins. Specifically for CIFAR-10, the best competing method is MIHash, a recent study that learns the hash mapping using a mutual information formulation. The improvement over MIHash is over **6**% for certain hash code lengths, *e.g.*, for 12 bits **0.799** *vs.* 0.738 mAP. Our method significantly improves over SHK as well, which also proposes a matrix fitting formulation but learns its hash mapping in an interleaved manner. This validates defining the binary code inference over items that directly define the neighborhood, *i.e.* classes for CIFAR-10.

For the NUSWIDE dataset, the binary inference is done over the set of label combinations in the training data. HBMP demonstrates either comparable results or outperforms the state-of-the-art hashing methods. A relevant recent two-stage hashing method is [22] in which the same settings (*cifar-1* and *nus-1*) are used but with fine-tuning a $\mathsf{VGG}-16$ architecture. Their CIFAR-10 and NUSWIDE results have at most 0.80 mAP and 0.75 mAP@5K values, respectively, for all hash code lengths. HBMP, on the other hand, achieves these performance values with the inferior VGG-F architecture.

To further emphasize the merits of HBMP, we consider the experimental settings *cifar-2* and *nus-2* and compare against recent deep-learning hashing methods. In this setting, we again fine-tune the VGG-F architecture pretrained on ImageNet. Table 2 gives the results. Notice that our method significantly outperforms all techniques, and yields new state-of-the-art results for CIFAR-10 and NUSWIDE.

Retrieval results for ImageNet100 are given in Table 3. In these experiments, we only compare against MIHash, the overall best competing method in past experiments and HashNet [4], another very recent deep learning based hashing study. As demonstrated, HBMP establishes the new state-of-the-art in image retrieval for this benchmark. HBMP outperforms both methods significantly, *e.g.*, with 64-bits, we demonstrate $4-6\%$ improvement. This further validates the quality of the binary codes produced with HBMP.

Multilevel Affinity Experiments. In these experiments, we allow multi-level similarities between items of set $\mathcal{X}$ and use NDCG as the evaluation criterion. For NUSWIDE, we consider the number of shared labels as affinity values. For 22K LabelMe dataset, we consider using distance percentiles $\{2\%, 5\%, 10\%, 20\%\}$ deduced from the training set to assign inversely proportional affinity values between the training instances. This emphasizes multi-level rankings among neigh-

	NUSWIDE (VGG − F, NDCG)				22K LabelMe (GIST, NDCG)			
Method	16 Bits	32 Bits	48 Bits	64 Bits	16 Bits	24 Bits	32 Bits	48 Bits
FastHash [9]	0.885	0.896	0.899	0.902	0.672	0.716	0.740	0.757
StructHash [37]	0.889	0.893	0.894	0.898	0.704	0.768	0.802	0.824
DPSH [38]	0.895	0.905	0.909	0.909	0.677	0.740	0.755	0.765
DTSH [14]	0.896	0.905	0.911	0.913	0.620	0.685	0.694	0.702
MIHash [39]	0.886	0.903	0.909	0.912	0.713	0.822	**0.855**	**0.873**
HBMP	**0.914**	**0.924**	**0.927**	**0.930**	**0.823**	**0.829**	0.849	0.866

Table 4: Multi-level affinity experiments on NUSWIDE and 22K LabelMe using VGG-F and GIST, respectively. The partitioning used for NUSWIDE is *nus-1*. The evaluation criterion is Normalized Discounted Cumulative Gain (NDCG). HBMP improves over the state-of-the-art in majority of the cases.

bors in the original feature space. In 22K LabelMe, we use a single fully connected layer as the hash mapping for the deep-learning based methods.

Table 4 gives results. For NUSWIDE, HBMP outperforms all state-of-the-art methods including MIHash. In 22K LabelMe, HBMP either achieves state-of-the-art performance, or is a close second. An interesting observation is that, when the feature learning aspect is removed due to the use of precomputed GIST features, non-deep methods such as FastHash and StructHash outperform deep-learning hashing methods DPSH and DTSH. While FastHash and StruchHash enjoy non-linear hash functions such as boosted decision trees, this also indicates that the prowess of DPSH and DTSH might come primarily through feature learning. On the other hand, both HBMP and MIHash show top performances with a single fully connected layer as the hash mapping, indicating that they produce binary codes that more accurately reflect the neighborhood. Regarding 22K LabelMe, for HBMP, set $\mathcal{X}$ corresponds to training instances, as similarly in other methods. This suggests that the performance improvement of HBMP is not merely due to the fact that the binary inference is performed upon items that directly define the neighborhood, but also due to our formulation that learns a Hamming metric with optimally selected bit weights.

5 Conclusion

We have proposed improvements to a commonly used formulation in two-stage hashing methods. We first provided a theoretical result on the quality of the binary codes showing that, under mild assumptions, we can construct binary codes that fit the neighborhood with arbitrary accuracy. Secondly, we analyzed the sub-optimality of binary codes constructed as to fit an affinity matrix that is not defined on items directly related to the neighborhood. Incorporating our findings, we proposed a novel two-stage hashing method that significantly outperforms previous hashing studies on multiple benchmarks.

Acknowledgments. The authors thank Sarah Adel Bargal for helpful discussions. This work is primarily conducted at Boston University, supported in part by a BU IGNITION award, and equipment donated by NVIDIA.

References

1. Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. In *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2012.
2. Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
3. Fatih Cakir and Stan Sclaroff. Adaptive hashing for fast similarity search. In *IEEE International Conf. on Computer Vision (ICCV)*. IEEE, 2015.
4. Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. HashNet: Deep learning to hash by continuation. In *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2017.
5. Kun He, Fatih Cakir, Sarah Adel Bargal, and Stan Sclaroff. Hashing as tie-aware learning to rank. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.
6. Wei Liu, Jun Wang, Rongrong Ji, Yu Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.
7. Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Deep supervised hashing for fast image retrieval. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
8. Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *Conf. on Artificial Intelligence (AAAI)*, 2014.
9. Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.
10. Xiaoshuang Shi, Fuyong Xing, Jinzheng Cai, Zizhao Zhang, Yuanpu Xie, and Lin Yang. Kernel-based supervised discrete hashing for image retrieval. In *Proc. European Conf. on Computer Vision (ECCV)*, 2016.
11. Thanh Toan Do, Anh Dzung Doan, Duc Thanh Nguyen, and Ngai Man Cheung. Binary hashing with semidefinite relaxation and augmented lagrangian. In *Proc. European Conf. on Computer Vision (ECCV)*, 2016.
12. Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2009.
13. Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *Proc. International Conf. on Machine Learning (ICML)*, 2011.
14. Xiaofang Wang, Yi Shi, and Kris Makoto Kitani. Deep supervised hashing with triplet labels. In *Proc. Asian Conf. on Computer Vision (ACCV)*, 2016.
15. Tiezheng Ge, Kaiming He, and Jian Sun. Graph cuts for supervised binary coding. In *Proc. European Conf. on Computer Vision (ECCV)*, 2014.
16. Fumin Shen, Chunhua Shen Wei, Liu Heng, and Tao Shen. Supervised discrete hashing. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
17. Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. In *University of Toronto Technical Report*, 2009.
18. Tat Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan Tao. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *ACM Conf. on Image and Video Retrieval (CIVR)*, 2009.

19. Jia Deng, Wei Dong, Richard Socher, Li Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009.
20. Bryan Russell, Antonio Torralba, Kevin Murphy, and William T. Freeman. Labelme: a database and web-based tool for image annotation. In *International Journal of Computer Vision (IJCV)*, 2008.
21. David Suter Guosheng Lin, Chunhua Shen and Anton van den Hengel. A general two-step approach to learning-based hashing. In *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2013.
22. Bohan Zhuang, Guosheng Lin, Chunhua Shen, and Ian Reid. Fast training of triplet-based deep binary embedding networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
23. Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. In *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2018.
24. Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2001.
25. Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2014.
26. Shai Shalev-Shwartz, Alon Gonen, and Ohad Shamir. Large-scale convex minimization with a low-rank constraint. In *Proc. International Conf. on Machine Learning (ICML)*, 2011.
27. Xinhua Zhang, Dale Schuurmans, and Yao Liang Yu. Accelerated training for matrix-norm regularization: A boosting approach. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2012.
28. Zheng Wang, Ming Jun Lai, Zhaosong Lu, Wei Fan, Hasan Davulcu, and Jieping Ye. Rank-one matrix pursuit for matrix completion. In *Proc. International Conf. on Machine Learning (ICML)*, 2014.
29. Quanming Yao and James T. Kwok. Learning of generalized low-rank models: A greedy approach. In *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2016.
30. Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *Proc. ACM SIGIR Conf. on Research & Development in Information Retrieval (SIGIR)*, 2010.
31. Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image resentation learning. In *Conf. on Artificial Intelligence (AAAI)*, 2014.
32. Qiang Yang, Long-Kai Huang, Wei-Shi Zheng, and Yingbiao Ling. Smart hashing update for fast response. In *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2013.
33. Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2008.
34. Yunchao Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.
35. Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *Proc. International Conf. on Machine Learning (ICML)*, 2010.

36. Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient training of very deep neural networks for supervised hashing. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
37. Guosheng Lin, Fayao Liu, Chunhua Shen, Jianxin Wu, and Heng Tao Shen. Structured learning of binary codes with column generation for optimizing ranking measures. In *International Journal of Computer Vision (IJCV)*, 2016.
38. Wu Jun Li, Sheng Wang, and Wang Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2016.
39. Fatih Cakir, Kun He, Sarah A. Bargal, and Stan Sclaroff. Mihash: Online hashing with mutual information. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
40. Xin Jing Wang, Lei Zhang, Feng Jing, and Wei Ying Ma. Annosearch: Image auto-annotation by search. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2006.
41. Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *Proc. European Conf. on Computer Vision (ECCV)*, 2012.
42. Xi Li, Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Anthony Dick. Learning hash functions using column generation. In *Proc. International Conf. on Machine Learning (ICML)*, 2013.
43. Lei Zhang, Yongdong Zhang, Jinhu Tang, Ke Lu, and Qi Tian. Binary code ranking with weighted hamming distance. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.
44. Yu Gang Jiang, Jung Wang, Xiangyang Xue, and Shih-Fu Chang. Query-adaptive image search with hash codes. In *IEEE Trans. on Multimedia*, 2013.
45. Roger A. Horn and Charles R. Johnson. Matrix analysis. In *Cambrigde University Press*, 1983.
46. Peter Merz and Bernd Freisleben. Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics*, 2002.
47. Christoph Buchheim, Marianna De Santis, Laura Palagi, and Mauro Piacentini. An exact algorithm for nonconvex quadratic integer minimization using ellipsoidal relaxations. *SIAM Journal on Optimization*, 2013.
48. Peng Wang, Chunhua Shen, Anton van den Hengel, and Phillip Torr. Large-scale binary quadratic optimization using semidefinite relaxation and applications. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2017.
49. Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. In *International Journal of Computer Vision (IJCV)*, 2001.
50. Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. British Machine Vision Conference (BMVC)*, 2014.
51. Fatih Cakir, Kun He, Sarah Adel Bargal, and Stan Sclaroff. Hashing with mutual information. *arXiv preprint arXiv:1803.00974*, 2018.
52. Fang Zhao, Y. Huang, L. Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
53. Ruimao Zhang, Liang Lin, Rui Zhang, Wangmeng Zuo, and Lei Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person reidentification. In *IEEE Trans. on Image Processing*, 2015.