Lost in Binarization: Query-Adaptive Ranking for Similar Image Search with Compact Codes

Yu-Gang Jiang[§], Jun Wang[†], Shih-Fu Chang[§]
[§]Columbia University, New York, NY 10027, USA
[†]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
{yjiang, sfchang}@ee.columbia.edu wangjun@us.ibm.com

ABSTRACT

With the proliferation of images on the Web, fast search of visually similar images has attracted significant attention. State-of-the-art techniques often embed high-dimensional visual features into low-dimensional Hamming space, where search can be performed in real-time based on Hamming distance of compact binary codes. Unlike traditional metrics (e.g., Euclidean) of raw image features that produce continuous distance, the Hamming distances are discrete integer values. In practice, there are often a large number of images sharing equal Hamming distances to a query, resulting in a critical issue for image search where ranking is very important. In this paper, we propose a novel approach that facilitates *query-adaptive ranking* for the images with equal Hamming distance. We achieve this goal by firstly offline learning bit weights of the binary codes for a diverse set of predefined semantic concept classes. The weight learning process is formulated as a quadratic programming problem that minimizes intra-class distance while preserving inter-class relationship in the original raw image feature space. Query-adaptive weights are then rapidly computed by evaluating the proximity between a query and the concept categories. With the adaptive bit weights, the returned images can be ordered by weighted Hamming distance at a finer-grained binary code level rather than at the original integer Hamming distance level. Experimental results on a Flickr image dataset show clear improvements from our query-adaptive ranking approach.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms

Algorithm, Experimentation, Performance.

Keywords

Query-adaptive ranking, weighted Hamming distance, binary code, image search.

1. INTRODUCTION

The capability of searching similar images in large databases has great potential in many real-world applications. While traditional image search engines heavily rely on textual words associated to the images, scalable content-based search techniques are receiving increasing attention and have recently appeared in some search engines such as *Google* and *Bing*. Apart from providing better image search experience for ordinary users on the Web, scalable similar image search has also been shown to be helpful for solving traditionally very hard problems in computer vision (e.g., image segmentation and categorization) [28].

Typically a scalable similar image search system should have two major components: i) effective image feature representation, and ii) efficient data structure. It is well known that the quality of image search results largely relies on the representation power of image features. The latter, an efficient data structure, is necessary since existing image features are mostly of high dimensions, on top of which exhaustively comparing a query with many database samples is computationally very slow.

In this work, we represent images using the popular bag-of-visual-words (BoW) framework [25], in which local invariant image descriptors (e.g., SIFT [18]) are extracted and quantized into a set of visual words. The BoW features are then embedded into compact binary codes for efficient search. For this, we consider state-of-the-art binary embedding techniques including hashing [30] and deep learning [8]. Binary embedding is preferable over the tree-based indexing structures (e.g., the kd-tree [1]) as it generally requires greatly reduced memory and also works better in high dimensions. With the binary codes, image similarity can be measured in Hamming space by Hamming distance – an integer value obtained by counting the number of bits with different values. For large scale applications, the dimension of the Hamming space is usually set as a small number (e.g., less than a hundred) limited by the memory budget [29, 34].

Despite the success of using Hamming distance of binary codes in scalable similar image search, it is important to realize that it lacks in providing good ranking that is crucial for image search – there can be C_d^i different binary codes sharing equal distance i ($i > 0$) to a query in a d-dimensional Hamming space. For example, there are 1,128 different binary codes with Hamming distance 2 to a query using 48-d compact binary codes. As a result, hundreds or even thousands of images may share an equal Hamming distance in practice, but are very unlikely to be equivalently relevant to the query.

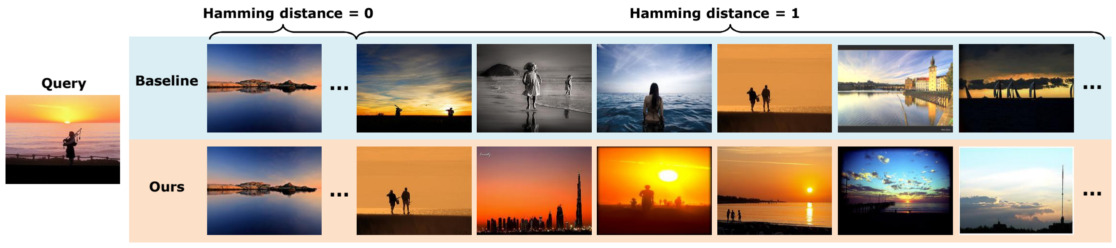


Figure 1: Search results of a *sunset scene* query from a Flickr image dataset, using 48-bit binary codes generated by deep learning. The top and bottom rows respectively show the most similar images based on query-independent Hamming distance and the proposed query-adaptive Hamming distance. It can be clearly seen that our method returns more relevant images. Note that our method does not permute images with exactly the same code to the query (three in total for this query), i.e., Hamming distance $= 0$; see texts for more explanations. This figure is best viewed in color.

This paper proposes a novel approach to compute *query-adaptive weights* for each bit of the binary codes, which has two main advantages. First, with the bit-level weights, we are able to rank the returned images at a finer-grained binary code level[1], rather than at the traditional Hamming distance level. In other words, we can push the resolution of ranking from d (Hamming distance level) up to 2^d (binary code level). Second, contrary to using a single set of weights for all the queries, our approach tailors a different and more suitable set of weights for each query.

To compute the query-adaptive weights in real-time, we harness a set of semantic concept classes that cover most semantic elements of image content (e.g., scenes and objects). Bit-level weights for each of the classes are learned offline using a novel formulation that not only maximizes intra-class sample similarity but also preserves inter-class relationships. We show that the optimal weights can be computed by iteratively solving quadratic programming problems. These precomputed class-specific weights are then utilized for online computation of the query-adaptive weights, through rapidly evaluating the proximity of a query image to the samples from the semantic classes. With the query-adaptive weights, weighted Hamming distance is finally applied to evaluate similarities between the query and images in a target database. We name our proposed approach as *query-adaptive Hamming distance*, as opposed to the query-independent Hamming distance widely used in existing works. Notice that during online search it is unnecessary to compute the weighted Hamming distance based on real-valued vectors (weights imposed on the binary codes), which would bury one of the most important advantages of binary embedding. Instead the weights can be utilized as indicators to efficiently order the returned images at binary code level.

Figure 1 displays search results using a query from a dataset of Flickr images. As can be seen, our approach obtains clearly better result (bottom row) by reordering images with Hamming distance larger than 0. It is worth noting that in most cases there is typically a very small number, if not zero, of images having Hamming distance 0 to the queries (see Figure 3), as there is only one binary code satisfying this condition (in contrast to $C_d^i, i > 0$).

Our main contributions are summarized as follows:

1. We propose a complete framework to learn adaptive Hamming distance for each query in real-time, in order to provide better ranking for similar image search with compact binary codes. This is achieved by novelly exploring a set of predefined semantic classes. To the best of our knowledge, it is the first work on query-adaptive ranking for image search in Hamming space.
2. To learn the suitable weights for each of the semantic classes, we propose a formulation that not only minimizes intra-class sample similarity but also maintains inter-class proximity, which can be efficiently solved by quadratic programming.

[1]Some existing approaches *rerank* top results at the finest-grained image level based on visual similarity of original raw features, which are not scalable since the raw features cannot fit in memory and disk-read operations for loading the features are slow.

The remaining sections are organized as follows. We review related works in Section 2 and give an overview of our approach in Section 3. Section 4 introduces two binary embedding methods, and Section 5 elaborates our formulation for learning the query-adaptive Hamming distance. We conduct experimental validations in Section 6 and conclude in Section 7.

2. RELATED WORK

Searching visually similar images has been a longstanding research issue, dating back at least to the early 1990s [26]. See good surveys by Smeulders et al. [26] and Datta et al. [4] for related works in the past decade. Many works adopted simple features such as color and texture [27, 22], while more effective features such as GIST [21] and SIFT [18] have been popular in recent years [25, 20]. In this paper, we choose the popular bag-of-visual-words (BoW) framework grounded on the local SIFT features, whose effectiveness has been validated in numerous applications. Since our work in this paper is more related to fast search, in the rest of this section we mainly review existing works on efficient data structures, which are roughly divided into three categories: 1) inverted index, 2) tree-based index, and 3) binary embedding.

Indexing data with inverted file was initially proposed and is still very popular for fast textual document retrieval in the IR community [37]. It has been introduced to the field of image retrieval as recent image feature representations such as BoW are very analogous to the bag-of-words representation of textual documents. In this structure, a list of references to each document (image) for each text (visual) word are created so that relevant documents (images) can be quickly located given a query with several words. A key difference of document retrieval from visual search, however, is that the textual queries usually contain very few words. For instance, on average there are 4 words per query in Google web search. While in the BoW representation, a single image may contain hundreds of words, resulting in a large number of candidate images (from the inverted lists) that need further verification – usually based on similarities of the original BoW features. This largely limits the application of inverted files for large scale image search. While increasing visual vocabulary size in BoW can reduce the number of candidates, it will also significantly increase memory usage [12]. For example, indexing 1 million BoW features of 10,000 dimensions will need 1GB memory with a compressed version of the inverted file. In contrast, for the binary embedding as will be discussed later, the memory consumption is much lower (48MB for 1 million 48-bit binary codes).

Tree-based indexing techniques [1, 17, 19, 20] have been frequently applied to fast visual search. Nister and Stewenius [20] used a visual vocabulary tree to perform real-time object retrieval in 40,000 images. Muja and Lowe [19] used multiple randomized kd-trees for SIFT feature matching. One drawback of the tree-based methods is that they are not suitable for high-dimensional feature. For example, let the dimensionality be d and the number of samples be n, one general rule is $n \gg 2^d$ in order to have kd-tree working more efficiently than exhaustive search [10].

In view of the limitations of both the inverted file and the tree-based methods, embedding high-dimensional image features into compact binary codes has attracted much more attention. Binary embedding satisfies both query-speed (via hash table or efficient bitwise operation) and memory requirements. Generally there are two ways for computing the binary code: hashing and deep learning. The former uses a group of projections to hash an input space into multiple buckets such that similar images are likely to be mapped into the same bucket. Most of the existing hash techniques are *unsupervised*. Among them, one of the most well-known hashing methods is Locality Sensitive Hashing (LSH) [11]. Recently, Kulis and Grauman [16] extended LSH from input feature space to arbitrary kernel space, and Chum et al. [3] proposed min-Hashing to extend LSH for sets of features. Since these LSH-based methods use random projections, when the dimension of the input space is high, many more bits (random projections) are needed for satisfactory performance. In light of this, Weiss et al. [34] proposed a spectral hashing (SH) method that hashes the input space based on data distribution, ensuring that projections are orthogonal and sample number is balanced across different buckets. To further utilize image label information, several *(semi-)supervised* methods have been proposed to learn good hash functions [15, 30].

Binary embedding by learning deep belief networks was proposed by Hinton and Salakhutdinov [8]. While it was occasionally also viewed as a type of hashing methods [23], we discuss it separately since its mechanism for generating the binary codes is very different from the traditional hashing techniques. Similar to the supervised hashing methods [15, 30], the deep network also requires image labels in order to learn a good mapping. In the network, multiple Restricted Boltzmann Machines (RBMs) are stacked and trained to gradually map image features at the bottom layer to binary codes at the top (deepest) layer. Several recent works have successfully applied deep learning for scalable image search [29, 9].

All these binary embedding methods, either unsupervised or supervised, have one limitation when applied to image search. As discussed in the introduction, the Hamming distance of binary codes cannot provide good ranking, which is very important in practice. This paper introduces a means to learn *query-adaptive* weights for each bit of the binary codes, so that images can be efficiently ranked at a finer resolution based on weighted Hamming distance. Our work in this paper is not on the proposal of new data structures or embedding techniques. Rather, our objective is to alleviate one weakness that all binary embedding methods share particularly in the context of image search.

There have been a few works using weighted Hamming distance for image retrieval, including parameter-sensitive hashing [24], Hamming distance weighting [13], and the AnnoSearch [32]. Each bit of the binary code is assigned with a weight in [24, 32], while in [13], the aim is to weigh the overall Hamming distance of local features for image matching. These methods are fundamentally different from this work. They all used *a single set* of weights to measure either the importance of each bit in Hamming space [24, 32], or to rescale the final Hamming distance for better matching of sets of features [13], while ours is designed to learn different weights efficiently and adaptively for each query.

3. SYSTEM ARCHITECTURE

Figure 2 gives an overview of our search system, where all the image features in both target database and the semantic classes are embedded into binary codes. The flowchart on the right illustrates the process of online search. In this figure, we assume that the class-specific bit-level weights of the binary codes (left; under the names of the semantic classes) were already computed offline. This is done by an algorithm that lies in the very heart of our approach, which will be discussed later in Section 5. During online search, binary code of the query image will be used to search against labeled images in the predefined semantic classes – we pool a large set of images that are close to the query in the Hamming space to predict its potential semantic content, and then compute query-adaptive weights using the precomputed weights of the potentially relevant classes. Finally, images in the database are rapidly ranked using weighted (query-adaptive) Hamming distance.

In the next we first briefly introduce two popular binary embedding methods. We then elaborate our algorithm for computing the bit-level weights for each of the predefined semantic classes. After that, we introduce a method for predicting the query adaptive weights.

4. BINARY EMBEDDING

We consider two state-of-the-art binary embedding techniques: *semi-supervised hashing* and *deep learning*.

4.1 Semi-Supervised Hashing

Recently, Semi-Supervised Hashing (SSH) was proposed for binary embedding. SSH leverages semantic similarities among labeled data while remains robust to overfitting [30]. The objective function of SSH consists of two main components, supervised empirical fitness and unsupervised information theoretic regularization. More specifically, SSH tries to minimize the empirical error on a small amount of labeled data while an unsupervised term provides effective regularization by maximizing desirable properties like variance and independence of individual bits. In the setting of SSH, one is given a set of n points, $\mathcal{P} = \{\mathbf{p}_i\},\ i = 1 \ldots n,\ \mathbf{p}_i \in \mathbb{R}^D$, in which a small fraction of pairs are associated with two categories of label information, $\mathcal{M}$ and $\mathcal{C}$. Specifically, a pair

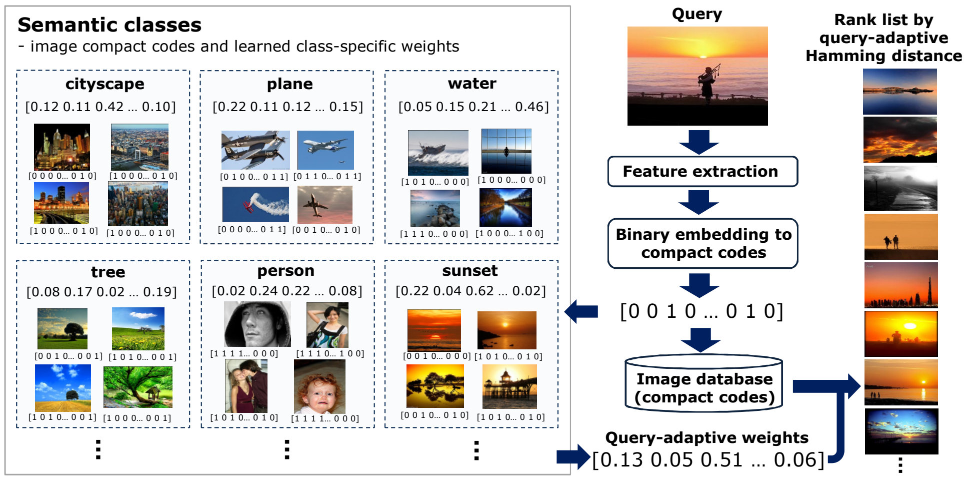


Figure 2: Overview of our image search framework using query-adaptive Hamming distance. Given a query image, we first compute bag-of-visual-words feature and embed it into compact binary code (Section 4). The binary code is then used to predict query-adaptive weights by harnessing a set of semantic classes with pre-computed class-specific weights (Section 5). Finally, the query-adaptive weights are applied to generate a rank list of search results using weighted (query-adaptive) Hamming distance.

$(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{M}$ is denoted as a neighbor-pair when $(\mathbf{p}_i, \mathbf{p}_j)$ are close in a metric space or share common class labels. Similarly, $(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{C}$ is called a nonneighbor-pair if two samples are far away in metric space or have no common class labels. The goal of SSH is to learn d hash functions $(\mathbf{H})$ that maximize the following objective function:

$$J(\mathbf{H}) = \sum_{k=1}^{d}\{ \sum_{(\mathbf{p}_i,\mathbf{p}_j)\in\mathcal{M}} h_k(\mathbf{p}_i)h_k(\mathbf{p}_j) - \sum_{(\mathbf{p}_i,\mathbf{p}_j)\in\mathcal{C}} h_k(\mathbf{p}_i)h_k(\mathbf{p}_j)\}$$
$$+ \sum_{k=1}^{d} \mathrm{var}[h_k(\mathbf{p})],$$

where the first term measures the empirical accuracy over the labeled sample pair sets $\mathcal{M}$ and $\mathcal{C}$, and the second part, i.e., the summation of the variance of hash bits, realizes the maximum entropy principle [31]. The above optimization is nontrivial. However, after relaxation, the optimal solution can be approximated using eigen-decomposition. Furthermore, a sequential learning algorithm called S3PLH was designed to implicitly learn bit-dependent hash codes with the capability of progressively correcting errors made by previous hash bits [31], where in each iteration, the weighted pairwise label information is updated by imposing higher weights on point pairs violated by the previous hash function. In this work, the S3PLH algorithm is applied to generate compact hash codes. We select 5000 labeled samples for learning $\mathbf{H}$. Both training and testing (binary embedding) of SSH are very efficient [30, 31].

4.2 Deep Learning

Learning with a deep belief network (DBN) was initially proposed for dimensionality reduction [8]. It was recently adopted for efficient search applications with binary embedding [23, 29, 9]. Similar to SSH, DBN also requires image labels during training phase, such that images with the same label are more likely to be mapped to the same bucket. Since the DBN structure gradually reduces the number of units in each layer[2], the high-dimensional input of the raw descriptors can be projected into a compact Hamming space.

Broadly speaking, a general DBN is a directed acyclic graph, where each node represents a stochastic variable. There are two critical steps in using DBN for binary embedding, i.e., learning interactions between variables and inferencing observations from inputs. The learning of a DBN with multiple layers is very hard since it usually requires to estimate millions of parameters. Fortunately, it has been shown in [7, 8] that the training process can be much more efficient if a DBN is specifically structured based on the RBMs – each single RBM has two layers containing respectively output visible units and hidden units, and multiple RBMs can be *stacked* to form a *deep* belief net. Starting from the input layer with D dimension, the network can be specifically designed to reduce the number of units, and finally output compact d-dimensional binary codes.

To obtain the optimal weights in the entire network, the training process of a DBN has two critical stages: unsupervised pre-training and supervised fine-tuning. The greedy pre-training phase is progressively executed layer by layer from input to output, aiming to place the network weights (and the biases) to suitable neighborhoods in the parameter space. After achieving convergence of the parameters of one layer via Contrastive Divergence, the outputs of this layer are fixed and treated as inputs to drive the training of the next layer. During the fine-tuning stage, labeled data is adopted to help refine the network parameters through back-propagation. Specifically, a cost function is defined to ensure that points (binary codes) within a certain neighborhood share the same label [6]. The network parameters are then refined to maximize this objective function using conjugate gradient descent.

In our experiments, the dimension of the image feature is fixed to 500. Similar to the network architecture used in [29], we use DBNs with five layers of sizes 500-500-500-256-d, where d is the dimension of the final binary code, ranging from 32 to 48. The training process requires to learn $500^2 + 500^2 + 500 \cdot 256 + 256 \cdot d$ weights in total – a minimum number of $636,192$ weights ($d = 32$) and a maximum number of $640,288$ weights ($d = 48$). For the number of training samples, we use a total number of $160,000$ samples in the pre-training stage, and 50 batches of neighborhood regions with size 1000 in the fine-tuning stage. Based on the efficient algorithms described earlier [7, 8], the entire training process can be finished within 1 day using a fast computer with a 2G Core-2 Quad CPU (15-24 hours depending on the

[2]In some cases the number of units may increase or remain the same for a few layers, and then decrease.

output code size). Since parameter training is an offline process, this computational cost is fairly acceptable. Compared to training, generating binary codes with the learned parameters is way faster – using the same computer it requires just 0.7 milliseconds to compute a 48-bit code for each image.

5. IMAGE SEARCH BY QUERY-ADAPTIVE HAMMING DISTANCE

With binary embedding, scalable image search can be executed in Hamming space using Hamming distance. The computation of the Hamming distance is highly efficient via bitwise operations. For example, comparing a query with a database containing a few millions of samples can finish within a fraction of a second. By definition, the Hamming distance between two binary codes is the total number of bits with different values, where the specific indexes of the bits are not considered, i.e., all the bits are treated equivalently. For example, given three binary codes $\mathbf{x} = 1100$, $\mathbf{y} = 1111$, and $\mathbf{z} = 0000$, the Hamming distance of $\mathbf{x}$ and $\mathbf{y}$ is equal to that of $\mathbf{x}$ and $\mathbf{z}$, regardless of the fact that $\mathbf{z}$ differs from $\mathbf{x}$ in the first two bits while $\mathbf{y}$ differs in the last two bits. Due to this nature of the Hamming distance, practically there can be hundreds or even thousands of samples having the same distance to a query. Back to the example, suppose we know that the first two bits are more important (discriminative) for $\mathbf{x}$, then $\mathbf{y}$ should be ranked higher than $\mathbf{z}$ if $\mathbf{x}$ is a query. In this section, we propose to learn query-adaptive weights for each bit of the binary code, so that images sharing the same Hamming distance can be ordered at a finer resolution.

5.1 Learning Class-Specific Weights

To quickly compute the query-adaptive weights, we propose to firstly learn class-specific weights for a number of semantic concept classes (e.g., scenes and objects). Assume that we have a dataset of k semantic classes, each with a set of labeled images. We learn bit-level weights separately for each class, with an objective of maximizing intra-class similarity as well as maintaining inter-class relationship. Formally, for two binary codes $\mathbf{x}$ and $\mathbf{y}$ in classes i and j respectively, their proximity is measured by weighted Hamming distance $\|\mathbf{a}_i \circ \mathbf{x} - \mathbf{a}_j \circ \mathbf{y}\|^2$, where $\circ$ denotes element-wise (Hadamard) product, and $\mathbf{a}_i$ ($\mathbf{a}_j$) is the bit-level weight vector for class i (j).

Let $\mathcal{X}$ be a set of n binary codes in a d-dimensional Hamming space, $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, $\mathbf{x}_j \in \mathbb{R}^d$, $j = 1, ..., n$. Denote $\mathcal{X}_i \subset \mathcal{X}$ as the subset of codes from class i, $i = 1, ..., k$. Our goal is to learn k weight vectors $\mathbf{a}_1$,...,$\mathbf{a}_k$, where $\mathbf{a}_i \in \mathbb{R}^d$ corresponds to class i. The learned weights should satisfy some constraints. First, $\mathbf{a}_i$ should be nonnegative (i.e., each entry of $\mathbf{a}_i$ is nonnegative), denoted as $\mathbf{a}_i \geq \mathbf{0}$. To fix the scale, we enforce the summation of the entries of $\mathbf{a}_i$ to 1, i.e., $\mathbf{a}_i^\top \mathbf{1} = 1$, where $\mathbf{1}$ denotes a vector of ones of d dimension. Ideally, a good weight vector should ensure the weighted codes from the same class to be close to each other. This can be quantified as follows:

$$f(\mathbf{a}_1, ..., \mathbf{a}_k) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathcal{X}_i} \|\mathbf{a}_i \circ \mathbf{x} - \mathbf{a}_i \circ \mathbf{c}^{(i)}\|^2, \quad (1)$$

where $\mathbf{c}^{(i)}$ is the center of the binary codes in class i, i.e., $\mathbf{c}^{(i)} = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} \mathbf{x}$, with n_i being the total number of binary codes in $\mathcal{X}_i$. Notice that although the sample proximity, to some extent, was considered in the binary embedding process of SSH and DBN, Eq. (1) is still helpful since weighting the binary codes is able to further condense samples from the same class. More importantly, the optimal bit-level weights to be learned here will serve as the key for computing the query-adaptive Hamming distance.

In addition to intra-class similarity, we also want the inter-class relationship in the original image feature (BoW) space to be preserved in the weighted Hamming space. Let $s_{ij} \geq 0$ denote the proximity between classes i and j ($s_{ij} = s_{ji}$). s_{ij} can be quantified using average BoW feature similarity of samples from classes i and j. Then it is expected that the weighted codes in classes i in j should be relatively more similar if s_{ij} is large. This maintains the class relationship, which is important since the semantic classes under our consideration are not exclusive – in fact some of them are highly correlated (e.g., *tree* and *grass*). We formalize this idea as a minimization problem of the following term:

$$g(\mathbf{a}_1, ..., \mathbf{a}_k) = \sum_{i,j=1}^{k} s_{ij} \|\mathbf{a}_i \circ \mathbf{c}^{(i)} - \mathbf{a}_j \circ \mathbf{c}^{(j)}\|^2. \quad (2)$$

Based on above analysis, we propose the following optimization problem to learn the weights for each class:

$$\min_{\mathbf{a}_1, ..., \mathbf{a}_k} \; f(\mathbf{a}_1, ..., \mathbf{a}_k) + \lambda g(\mathbf{a}_1, ..., \mathbf{a}_k) \quad (3)$$

$$\text{s.t. } \mathbf{a}_i^\top \mathbf{1} = 1, \; i = 1, ..., k, \quad (4)$$

$$\mathbf{a}_i \geq \mathbf{0}, \; i = 1, ..., k, \quad (5)$$

where $\lambda \geq 0$ is a parameter that controls the balance of the two terms. Next we show that the above optimization problem can be efficiently solved using an iterative quadratic programming (QP) scheme.

First, we can rewrite f in matrix form as

$$f(\mathbf{a}_1, ..., \mathbf{a}_k) = \sum_{i=1}^{k} \mathbf{a}_i^\top A_i \mathbf{a}_i \quad (6)$$

where A_i is a symmetric positive semidefinite matrix:

$$A_i = \text{diag}(\sum_{\mathbf{x} \in \mathcal{X}_i} (\mathbf{x} - \mathbf{c}^{(i)}) \circ (\mathbf{x} - \mathbf{c}^{(i)})). \quad (7)$$

Also, to simplify g, we have

$$\begin{aligned} &\|\mathbf{a}_i \circ \mathbf{c}^{(i)} - \mathbf{a}_j \circ \mathbf{c}^{(j)}\|^2 \\ &= \mathbf{a}_i^\top C_{ii} \mathbf{a}_i - 2\mathbf{a}_i^\top C_{ij} \mathbf{a}_j + \mathbf{a}_j^\top C_{jj} \mathbf{a}_j \end{aligned} \quad (8)$$

where $C_{ij} = \text{diag}(\mathbf{c}^{(i)} \circ \mathbf{c}^{(j)})$.

With these preparations, we can expand Eq. (3) as

$$\begin{aligned} &f(\mathbf{a}_1, ..., \mathbf{a}_k) + \lambda g(\mathbf{a}_1, ..., \mathbf{a}_k) \\ &= \sum_{i=1}^{k} \mathbf{a}_i^\top A_i \mathbf{a}_i + \lambda \sum_{j,l=1}^{k} s_{jl} (\mathbf{a}_j^\top C_{jj} \mathbf{a}_j - 2\mathbf{a}_j^\top C_{jl} \mathbf{a}_l + \mathbf{a}_l^\top C_{ll} \mathbf{a}_l) \\ &= \mathbf{a}_i^\top (A_i + 2\lambda (\sum_l s_{il} - s_{ii}) C_{ii}) \mathbf{a}_i - (4\lambda \sum_{j \neq i} s_{ji} C_{ji} \mathbf{a}_j)^\top \mathbf{a}_i + \\ &(\sum_{j \neq i}^{k} \mathbf{a}_j^\top A_j \mathbf{a}_j + 2\lambda \sum_{j \neq i,l} s_{jl} \mathbf{a}_j^\top C_{jj} \mathbf{a}_j - 2\lambda \sum_{j \neq i, l \neq i} s_{jl} \mathbf{a}_j^\top C_{jl} \mathbf{a}_l) \\ &= \frac{1}{2} \mathbf{a}_i^\top Q_i \mathbf{a}_i + \mathbf{p}_i^\top \mathbf{a}_i + t_i, \end{aligned} \quad (9)$$

Algorithm 1 Learning class-specific weights.

```
1: INPUT:
       Binary codes $\mathcal{X}_i$ and class similarity $s_{ij}$, $i, j = 1, ..., k$.
2: OUTPUT:
       Class-specific weights $\mathbf{a}_j$, $j = 1, ..., k$.
3: Compute $\mathbf{c}^{(i)}$, and initialize $\mathbf{a}_j = \mathbf{1}/d$, $j = 1, ..., k$;
4: Repeat
5:   For $i = 1, ..., k$
6:     Compute $Q_i$, $\mathbf{p}_i$, and $t_i$ using Eq. (10) – Eq. (12);
7:     Solve the following QP problem:
         $\mathbf{a}_i^* = \arg\min_{\mathbf{a}_i} \frac{1}{2}\mathbf{a}_i^\top Q_i \mathbf{a}_i + \mathbf{p}_i^\top \mathbf{a}_i + t_i$
         s.t. $\mathbf{a}_i^\top \mathbf{1} = 1$ and $\mathbf{a}_i \geq \mathbf{0}$;
8:     Set $\mathbf{a}_i = \mathbf{a}_i^*$;
9:   End for
10: Until convergence
```

where

$$Q_i = 2A_i + 4\lambda(\sum_l s_{il} - s_{ii})C_{ii}, \tag{10}$$

$$\mathbf{p}_i = -4\lambda \sum_{j\neq i} s_{ji} C_{ji} \mathbf{a}_j, \tag{11}$$

$$t_i = \sum_{j\neq i}^{k} \mathbf{a}_j^\top A_j \mathbf{a}_j + 2\lambda \sum_{j\neq i,l} s_{jl} \mathbf{a}_j^\top C_{jj} \mathbf{a}_j - 2\lambda \sum_{j\neq i, l\neq i} s_{jl} \mathbf{a}_j^\top C_{jl} \mathbf{a}_l. \tag{12}$$

Up to this point we have derived the quadratic form of Eq. (3) w.r.t. $\mathbf{a}_i$. Notice that Q_i is symmetric and positive semidefinite as A_i and C_{ii} are. Given all $\mathbf{a}_j$, $j \neq i$, the quadratic program in Eq. (9) is convex, and thus $\mathbf{a}_i$ can be solved optimally. This analysis suggests an iterative procedure summarized in Algorithm 1 for solving the optimization problem stated in Eq. (3) – Eq. (5). The stop condition (convergence) is defined as the energy difference of two states $|\mathcal{E}_{current} - \mathcal{E}_{previous}| < \xi$, where $\mathcal{E} = f(\mathbf{a}_1, ..., \mathbf{a}_k) + \lambda g(\mathbf{a}_1, ..., \mathbf{a}_k)$ and ξ is set as a small value (10^{-6} in our experiments).

Algorithm Discussion. It is interesting to briefly discuss the difference and connection of our algorithm to some general machine learning methods such as LDA [5] and distance metric learning, although ours is particularly designed for this specific application. LDA is a well-known method that linearly projects data into a low-dimensional space where the sample proximity is reshaped to maximize class separability. While LDA also tries to condense samples from the same class, it learns a single uniform transformation matrix $G \in \mathbb{R}^{d\times s}$ to map all original d-dimensional features to a lower s-dimensional space. In contrast, we learn a d-dimensional weight vector separately for each class.

Distance metric learning tries to find a metric so that samples of different classes in the learned metric space can be better separated. Typically a single Mahalanobis distance metric is learned for the entire input space [35]. There are also a few exceptions that consider multiple metrics, e.g., a metric is trained per category in [33]. Besides the different formulations, our method aims to learn bit-level weights of binary codes for each class, which was not considered in these relevant works.

5.2 Computing Query-Adaptive Weights

As described in Figure 2, labeled images and the learned weights of the predefined semantic concept classes form a *semantic database* for rapid computation of the query-adaptive weights. Given a query image q and its binary code $\mathbf{x}_q$, the objective is to adaptively determine a suitable weight vector $\mathbf{a}_q$, so that we can apply weighted Hamming distance to compare $\mathbf{x}_q$ to each binary code $\mathbf{x}_t$ in the target database:

$$d(\mathbf{x}_q, \mathbf{x}_t) = \|\mathbf{a}_q \circ \mathbf{x}_q - \mathbf{a}_q \circ \mathbf{x}_t\|^2. \tag{13}$$

To compute $\mathbf{a}_q$, we query $\mathbf{x}_q$ against the semantic database based on Hamming distance. Semantic labels of the top k most similar images are collected to predict the labels of the query, which are then utilized to compute the query-adaptive weights. Specifically, denote $\mathcal{T}$ as the set of the most relevant semantic classes to the query q, and m_i as the number of images (within the top k list) from the i^{th} class in $\mathcal{T}$. The query adaptive weights are computed via linear combination as $\mathbf{a}_q = \sum_{i\in\mathcal{T}}(m_i \mathbf{a}_i)/\sum_{i\in\mathcal{T}} m_i$, where $\mathbf{a}_i$ is the precomputed weight vector of the corresponding class. We empirically choose 500 for k (random selection for images with equal Hamming distances) and 3 for $|\mathcal{T}|$.

It is important to note that there is no need to compute the weighted Hamming distance based on the form shown in Eq. (13), which can be rewritten as $(\mathbf{a}_q \circ \mathbf{a}_q)^\top(\mathbf{x}_q \oplus \mathbf{x}_t)$, where $\oplus$ means the `xor` of the two binary codes. While the weighting can now be achieved by an inner-product operation, it is actually possible to avoid this computational cost – by computing traditional Hamming distance and then ranking the images at binary code level rather than Hamming distance level. Since we have computed the weight of each bit, the orders of the binary codes (relevance to the query) could be obtained with very minor additional cost. Similar observations on this were also given in [24].

6. EXPERIMENTS

Dataset and Setup: We conduct image search experiments using the NUS-WIDE dataset [2]. NUS-WIDE contains 269,648 Flickr images, which are divided into a training set (161,789 images) and a test set (107,859 images). It is fully labeled with 81 semantic concepts/classes, covering a wide range of semantic topics from objects (e.g., *bird* and *car*) to scenes (e.g., *mountain* and *harbor*). Notice that NUS-WIDE is to our knowledge the largest available dataset that is fully labeled with a large number of classes. Other well-known and larger datasets such as ImageNet and MIT TinyImages are either not fully labeled[3] or unlabeled, creating difficulties in performance evaluation.

The concepts in NUS-WIDE are very suitable for constructing the *semantic database* in our approach – therefore we use the training set to learn class-specific weights based on the algorithm introduced in Section 5. We compute a set of weights for each type of the binary codes, where the balance parameter λ is uniformly set as 1. In our experiments, we tried a range of λ and found the performance remains fairly stable.

Local features are extracted from all the images based on Lowe's DoG detector and SIFT descriptor [18]. We then compute soft-weighted BoW features [14] using a visual vo-

[3]Each image in ImageNet has only one label. E.g., a *car* image may be labeled to a vehicle-related category, but not to *road* or *building* categories, although both may co-occur with *car*.

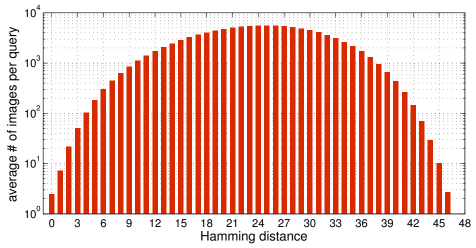
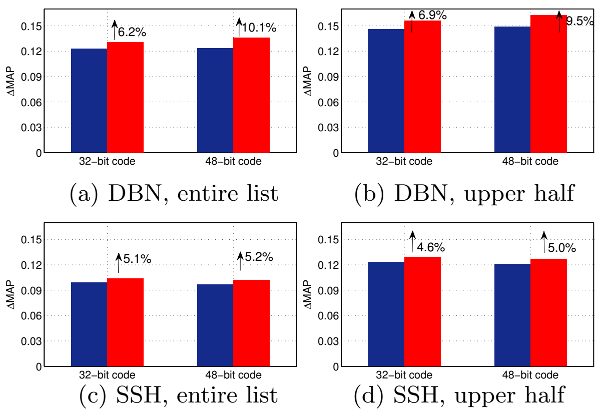


Figure 3: Average number of returned images at each Hamming distance, computed using 20k queries and 48-bit binary codes generated by DBN. Note that the y-axis is plotted in log-scale.



(a) DBN, entire list (b) DBN, upper half

(c) SSH, entire list (d) SSH, upper half

Figure 4: Comparison of our query-adaptive Hamming distance (red bars) and traditional Hamming distance (blue bars). Results are measured by ΔMAP over 8,000 randomly selected queries. We show ΔMAP over *entire* and *upper half* of the result rank lists using binary codes from both DBN and SSH. The performance gains are marked on top of the red bars.

cabulary of 500 visual words. For evaluation, we rank all images in the test dataset according to their (weighted) Hamming distances to each query. An image will be treated as a correct hit if it shares at least one common class label to the query. The performance is then evaluated using ΔAP, an extended version of the average precision where prior probability of each query result is taken into account [36]. To aggregate performance of multiple queries, mean ΔAP (ΔMAP) is used.

Characteristics of Binary Codes for Search: We first analyze the number of images returned at each Hamming distance. The 48-bit binary codes from DBN are used in this experiment. Note that we will not specifically investigate the effect of code-length in this paper, since several previous works on binary embedding have already shown that codes of 32-50 bits work well for search [29, 15]. In general, using more bits may lead to higher precision, but at the price of larger memory usage and longer search time.

Figure 3 visualizes the results, averaged over 20,000 randomly selected queries. As shown in the figure, the number of returned images rapidly increases when the Hamming radius (distance) goes up. This shows one nature of the Hamming distance – as mentioned in the introduction, there can be C_d^i different binary codes sharing the same integer distance i ($i > 0$) to a query in a d-dimensional Hamming space. Consequently, the number of binary codes (and correspondingly the number of images) at each specific distance will increase dramatically as i grows until $i = d/2$. For some queries, there can be as many as 10^3-10^4 images sharing equal Hamming distance. This clearly motivates the need of our proposed approach, which provides ranking at a finer granularity. On the other hand, although our approach does not permute images with Hamming distance 0 to the queries, from the analysis we see that this is not a critical issue since most queries have none or just a few such images (2.4 on average in the evaluated queries).

Results and Analysis: We now evaluate how much performance gain can be achieved from the proposed query-adaptive Hamming distance. Figure 4 displays the results using 32-bit and 48-bit binary codes from both DBN and SSH. In this experiment, we randomly pick 8,000 queries (each contains at least one semantic label) and compute averaged performance over all the queries. As shown in Figure 4(a,c), our approach significantly outperforms traditional Hamming distance. For the DBN codes, it improves the 32-bit baseline by 6.2% and the 48-bit baseline by 10.1% over the entire rank lists. A little lower but very consistent improvements (about 5%) are obtained for the SSH codes. These results clearly validate the effectiveness of learning query-adaptive weights for image search with compact codes.

Figure 4(b,d) further gives the performance over upper half of the ranked search results, using the same set of queries. The aim of this evaluation is to verify whether our approach is able to improve the ranking of top images, i.e., those with relatively smaller Hamming distances to the queries. As expected, we see similar performance gain to that over the entire list.

As for the baseline performance, DBN codes are better because more labeled training samples are used (50k for DBN vs. 5k for SSH). Note that comparing DBN to SSH is beyond the focus of this work, as the latter is a semi-supervised method that prefers and is more suitable for cases with limited training samples. Direct comparison of the two with equal training set size can be found in [30].

To see whether the improvement is consistent over the evaluated queries, we group the queries into 81 categories based on their associated labels. Results from the 48-bit DBN codes are displayed in Figure 5. Steady performance gain is obtained for almost all the query categories, and none of them suffers from performance degradation, which again validates the effectiveness of our approach. Figure 6 shows top-5 images for two example queries, where our approach shows better results by replacing the clearly irrelevant images with more suitable ones.

7. CONCLUSION

Searching images with binary codes and Hamming distance has great potential in practice, as its efficiency allows us to rapidly browse thousands of images on a regular laptop computer (e.g., for personal photo organization), and millions or even billions of images on a decent server (e.g., for Web scale applications). This paper proposed a novel framework for scalable image search using *query-adaptive Hamming distance*, by efficiently harnessing a large set of predefined semantic concept classes. Compared with traditional Hamming distance of binary codes, our approach is able to adaptively produce finer-grained ranking of search results. Experiments on a Flickr image dataset have confirmed the effectiveness of our approach. One important

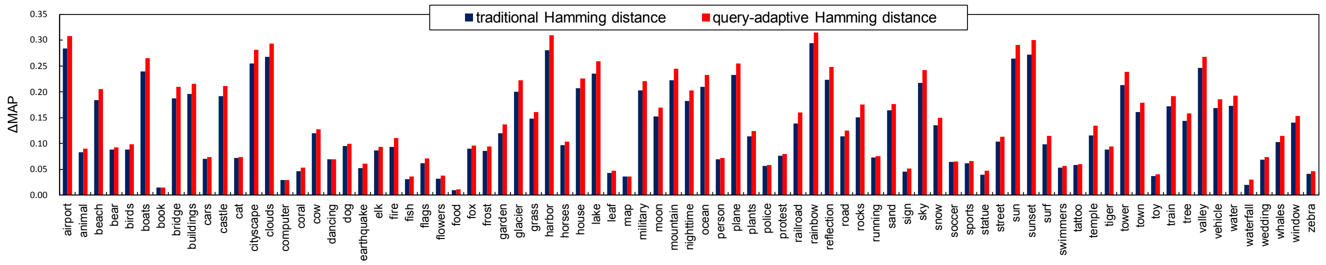


Figure 5: Per-category ΔMAP comparison using 48-bit DBN codes. The queries are grouped into 81 categories based on their associated labels.

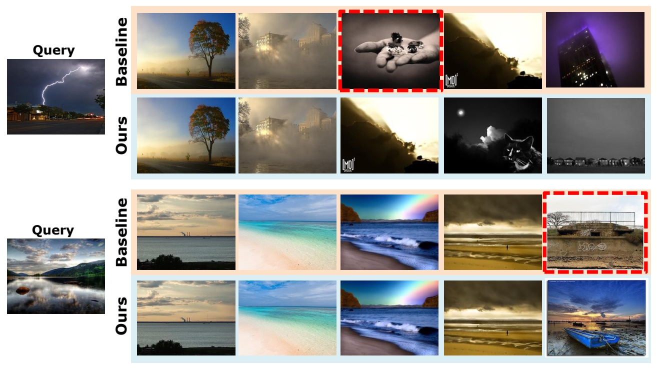


Figure 6: Top 5 returned images of two example queries, using the 48-bit DBN codes. Our approach shows better results than the baseline traditional Hamming distance, by removing irrelevant images (red dotted boxes) from the top-5 lists. Others are all visually relevant (*night-view/outdoor scene* for the first query, and *water-view* for the second one).

future work is to further enlarge the number of semantic classes for more accurate prediction of the query-adaptive bit weights.

Acknowledgement

This work is supported in part by NSF Award No CNS-07-51078. We thank Dr. Zhenguo Li for his help on this work.

8. REFERENCES

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. NUSWIDE: A real-world web image database from national university of singapore. In *CIVR*, 2009.

[3] O. Chum, M. Perdoch, and J. Matas. Geometric min-hashing: Finding a (thick) needle in a haystack. In *CVPR*, 2009.

[4] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2), 2008.

[5] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[6] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*. 2004.

[7] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[8] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[9] E. Horster and R. Lienhart. Deep networks for image retrieval on large-scale databases. In *ACM Multimedia*, 2008.

[10] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 39. CRC press, 2004.

[11] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Symposium on Theory of Computing*, 1998.

[12] H. Jegou, M. Douze, and C. Schmid. Packing bag-of-features. In *CVPR*, 2009.

[13] H. Jegou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *IJCV*, 87:191–212, 2010.

[14] Y. G. Jiang, C. W. Ngo, and J. Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *CIVR*, 2007.

[15] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[16] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[17] N. Kumar, L. Zhang, and S. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *ECCV*, 2008.

[18] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[19] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, 2009.

[20] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[21] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 42:145–175, 2001.

[22] T. Quack, U. Monich, L. Thiele, and B. Manjunath. Cortina: A system for large-scale, content-based web image retrieval. In *ACM Multimedia*, 2004.

[23] R. Salakhutdinov and G. Hinton. Semantic hashing. *SIGIR Workshop*, 2007.

[24] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, 2003.

[25] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[26] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *PAMI*, 22(12):1349–1380, 2000.

[27] J. R. Smith and S.-F. Chang. VisualSEEk: a fully automated content-based image query system. In *ACM Multimedia*, 1996.

[28] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *PAMI*, 30(11):1958–1970, 2008.

[29] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008.

[30] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, 2010.

[31] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, 2010.

[32] X.-J. Wang, L. Zhang, F. Jing, and W.-Y. Ma. Annosearch: image auto-annotation by search. In *CVPR*, 2006.

[33] K. Weinberger and L. Saul. Fast solvers and efficient implementations for distance metric learning. In *ICML*, 2008.

[34] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008.

[35] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. *NIPS*, pages 521–528, 2003.

[36] J. Yang and A. G. Hauptmann. (Un)reliability of video concept detection. In *CIVR*, 2008.

[37] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006.