Query-Adaptive Image Search With Hash Codes

Yu-Gang Jiang, Jun Wang, *Member, IEEE*, Xiangyang Xue, *Member, IEEE*, and Shih-Fu Chang, *Fellow, IEEE*

***Abstract*—Scalable image search based on visual similarity has been an active topic of research in recent years. State-of-the-art solutions often use hashing methods to embed high-dimensional image features into Hamming space, where search can be performed in real-time based on Hamming distance of compact hash codes. Unlike traditional metrics (e.g., Euclidean) that offer continuous distances, the Hamming distances are discrete integer values. As a consequence, there are often a large number of images sharing equal Hamming distances to a query, which largely hurts search results where fine-grained ranking is very important. This paper introduces an approach that enables query-adaptive ranking of the returned images with equal Hamming distances to the queries. This is achieved by firstly offline learning bitwise weights of the hash codes for a diverse set of predefined semantic concept classes. We formulate the weight learning process as a quadratic programming problem that minimizes intra-class distance while preserving inter-class relationship captured by original raw image features. Query-adaptive weights are then computed online by evaluating the proximity between a query and the semantic concept classes. With the query-adaptive bitwise weights, returned images can be easily ordered by weighted Hamming distance at a finer-grained hash code level rather than the original Hamming distance level. Experiments on a Flickr image dataset show clear improvements from our proposed approach.**

***Index Terms*—Query-adaptive image search, scalability, hash codes, weighted Hamming distance.**

I. Introduction

With the explosion of images on the Internet, there is a strong need to develop techniques for efficient and scalable image search. While traditional image search engines heavily rely on textual words associated to the images, scalable content-based search is receiving increasing attention. Apart from providing better image search experience for ordinary Web users, large-scale similar image search has also been demonstrated to be very helpful for solving a number of very hard problems in computer vision and multimedia such as image categorization [1].

Generally a large-scale image search system consists of two key components—an effective image feature representation and an efficient search mechanism. It is well known that the quality of search results relies heavily on the representation power of image features. The latter, an efficient search mechanism, is critical since existing image features are mostly of high dimensions and current image databases are huge, on top of which exhaustively comparing a query with every database sample is computationally prohibitive.

In this work we represent images using the popular bag-of-visual-words (BoW) framework [2], where local invariant image descriptors (e.g., SIFT [3]) are extracted and quantized based on a set of visual words. The BoW features are then embedded into compact hash codes for efficient search. For this, we consider state-of-the-art techniques including semi-supervised hashing [4] and semantic hashing with deep belief networks [5], [6]. Hashing is preferable over tree-based indexing structures (e.g., kd-tree [7]) as it generally requires greatly reduced memory and also works better for high-dimensional samples. With the hash codes, image similarity can be efficiently measured (using logical `XOR` operations) in Hamming space by Hamming distance, an integer value obtained by counting the number of bits at which the binary values are different. In large scale applications, the dimension of Hamming space is usually set as a small number (e.g., less than a hundred) to reduce memory cost and avoid low recall [8], [9].

Although hashing has been shown to be effective for visual search in several existing works [8]–[10], it is important to realize that it lacks in providing a good ranking that is crucial for image search. There can be C_d^i different hash codes sharing an equal distance i $(i > 0)$ to a query in a d-dimensional Hamming space. Taking 48-d hash codes as an example, there are as many as 1128 different codes having Hamming distance 2 to a query. As a result, hundreds or even thousands of images may share the same ranking in search result list, but are very unlikely to be equivalently relevant to the query. Although one can exhaustively compute the similarity for such candidate images to obtain exact ranking [4], it will significantly increase both computational cost and memory needs.

The main contribution of this paper is the proposal of a novel approach that computes *query-adaptive weights* for each bit of the hash codes, which has two main advantages. First, images can be ranked on a finer-grained hash code level since—with the bitwise weights—each hash code is expected to have a unique similarity to the queries. In other words, we can push the resolution of ranking from d (traditional Hamming distance level) up to 2^d (hash code level[1]). Second, contrary to using a single set of

Manuscript received November 22, 2011; revised April 23, 2012 and July 16, 2012; accepted July 22, 2012. Date of publication November 30, 2012; date of current version January 15, 2013. This work was supported in part by a national 973 program (No. 2010CB327900), two programs from the National Natural Science Foundation of China (No. 61201387 and No. 61228205), and two STCSM's programs (No. 10511500703 and No. 12XD1400900), China. The associate editor coordinating the review of this manuscript and approving it for publication was Shin'ichi Satoh.

Y.-G. Jiang and X. Xue are with the School of Computer Science, Fudan University, Shanghai, China (e-mail: ygj@fudan.edu.cn; xyxue@fudan.edu.cn).

J. Wang is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: wangjun@us.ibm.com).

S.-F. Chang is with the Department of Electrical Engineering and the Department of Computer Science, Columbia University, New York, NY 10027 USA (e-mail: sfchang@ee.columbia.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2012.2231061

[1]Given a code length d, there can be 2^d different binary hash codes.

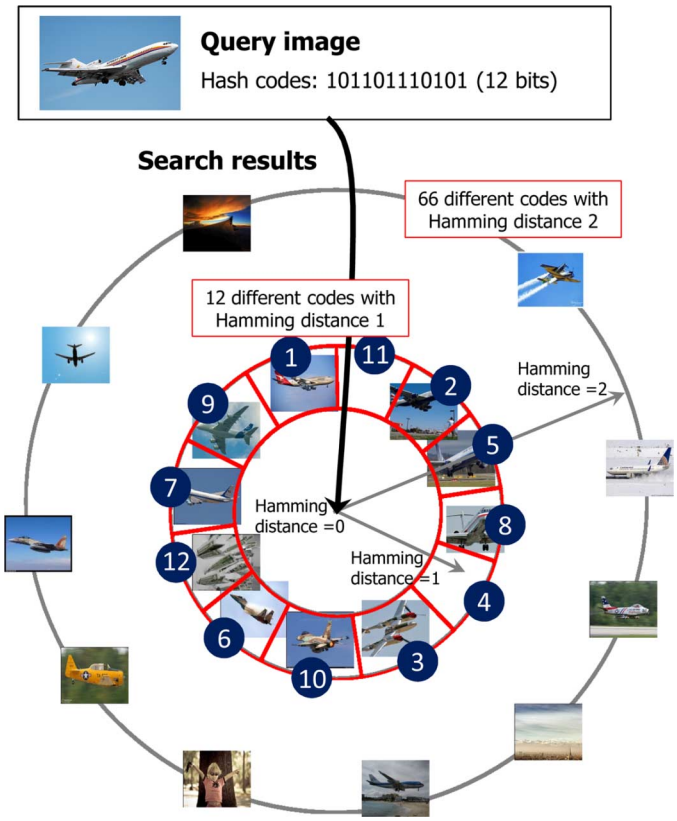


Fig. 1. An illustration of the proposed query-adaptive image search approach, using an example query represented by a 12-bit hash code. Traditionally hashing-based search results are ordered by integer Hamming distance, which is not ideal since many different hash codes share the same distance to the query. For instance, in this example there are 12 hash codes having Hamming distance 1 (each differs from the query in one bit). The proposed approach is able to order results at finer-grained hash code level. As exemplified over the images with Hamming distance 1 to the query, we propose a way to differentiate the ranking of the 12 different hash codes, where the order is online determined adaptively for each query. See texts for more explanations.

weights for all the queries, our approach tailors a different and more suitable set of weights for each query. Fig. 1 illustrates the proposed approach.

The query-adaptive bitwise weights need to be computed in real-time. To this end, we harness a set of semantic concept classes that cover many semantic aspects of image content (e.g., scenes and objects). Bitwise weights for each of the semantic classes are learned offline using a novel formulation that not only maximizes intra-class sample similarities but also preserves inter-class relationships. We show that the optimal weights can be computed by iteratively solving quadratic programming problems. These pre-computed class-specific bitwise weights are then utilized for online computation of the query-adaptive weights, through rapidly evaluating the proximity of a query image to the image samples of the semantic classes. Finally, weighted Hamming distance is applied to evaluate similarities between the query and images in a target database. We name this weighted distance 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 hash codes), which would bury one of the most important advantages of hashing. Instead the weights can be utilized as indicators to efficiently order the returned images (found by logical XOR operations) at hash code level.

Fig. 2 shows search results of a query from a Flickr image dataset. We see that the proposed approach produces clearly better result (bottom row) by ordering images with Hamming distance 1 to the query. Notice that in most cases there is typically a very small number, if not zero, of images with Hamming distance 0 to search queries (see Fig. 6), as there is only one hash code satisfying this condition (in contrast to $C_d^i, i > 0$).

The rest of this paper is organized as follows. We briefly review existing efficient search methods and discuss related works in Section II. Section III gives an overview of the proposed query-adaptive image search system. Section IV briefly introduces two hashing methods that will be used in this work and Section V elaborates our approach. Experimental setup is described in Section VI and results are discussed in Section VII. Finally, Section VIII concludes this paper.

II. Related Works

There are very good surveys on general image retrieval task. See Smeulders *et al.* [11] for works from the 1990s and Datta *et al.* [12] for those from the past decade. Many people adopted simple features such as color and texture in systems developed in the early years [13], while more effective features such as GIST [14] and SIFT [3] have been popular recently [2], [15]. In this work, we choose the popular bag-of-visual-words (BoW) representation grounded on the local invariant SIFT features. The effectiveness of this feature representation has been verified in numerous applications. Since the work in this paper is more related to efficient search, this section mainly reviews existing works on efficient search mechanisms, which are roughly divided into three categories: inverted file, tree-based indexing, and hashing.

Inverted index was initially proposed and is still very popular for document retrieval in the informational retrieval community [16]. It was 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 is 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 merely 4 words per query in Google web search.[2] While in the BoW representation, a single image may contain hundreds of visual words, resulting in a large number of candidate images (from the inverted lists) that need further verification—a process that is 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

[2]http://www.beussery.com/blog/index.php/2008/02/google-average-number-of-words-per-query-have-increased/

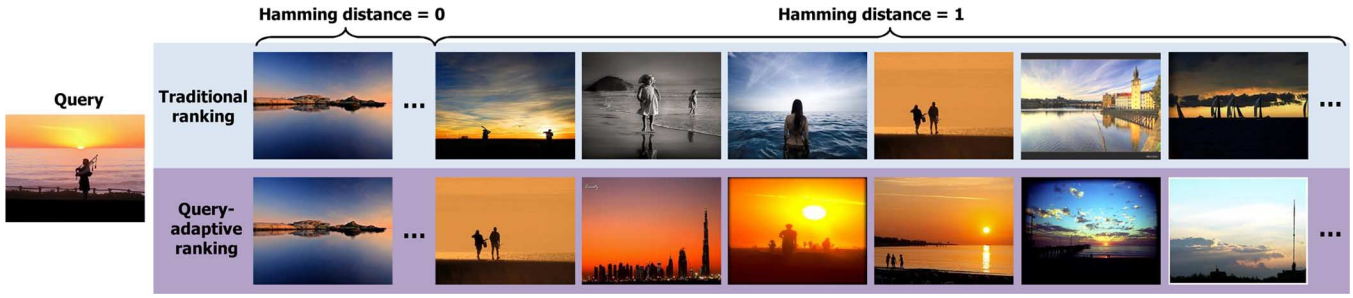


Fig. 2. Search result lists in a Flickr image dataset, using a "sunset scene" query image (left). Top and bottom rows respectively show the most similar images based on traditional Hamming distance and our proposed query-adaptive weighted Hamming distance. It can be clearly seen that our method produces more relevant result by ranking images at a finer resolution. Note that the proposed method does not permute images with exactly the same code to the query (three images in total for this query), i.e., $\text{Hamming distance} = 0$. This figure is best viewed in color.

[17]. For example, indexing 1 million BoW features of 10 000 dimensions will need 1 GB memory with a compressed version of the inverted file. In contrast, for the binary representation in hashing methods, as will be discussed later, the memory consumption is much lower (e.g., 48 MB for 1 million 48-bit hash codes).

Indexing with tree-like structures [7], [18], [15] has been frequently applied to fast visual search. Nister and Stewenius [15] used a visual vocabulary tree to achieve real-time object retrieval in 40 000 images. Muja and Lowe [18] adopted multiple randomized kd-trees [7] for SIFT feature matching in image applications. One drawback of the classical tree-based methods is that they normally do not work well with 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 [19]. There are also several works focusing on improving tree-based approaches for large-scale search [20], [21], where promising image search performance has been reported. Compared with these methods, hashing has a major advantage in speed since it allows constant-time search.

In view of the limitations of both inverted file and tree-based indexing, embedding high-dimensional image features into hash codes has become very popular recently. Hashing satisfies both query time and memory requirements as the binary hash codes are compact in memory and efficient in search via hash table lookup or bitwise operations. Hashing methods normally use a group of projections to divide an input space into multiple buckets such that similar images are likely to be mapped into the same bucket.

Most of the existing hashing techniques are *unsupervised*. Among them, one of the most well-known hashing methods is Locality Sensitive Hashing (LSH) [22]. Recently, Kulis and Grauman [23] extended LSH to work in arbitrary kernel space, and Chum *et al.* [24] 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 to achieve satisfactory performance. Motivated by this mainly, Weiss *et al.* [9] proposed a spectral hashing (SH) method that hashes the input space based on data distribution. SH also ensures that the projections are orthogonal and sample number is balanced across different buckets.

Although SH can achieve similar or even better performance than LSH with a fewer number of bits, it is important to underline that these unsupervised hashing techniques are not robust enough for similar image search. This is due to the fact that similarity in image search is not simply equivalent to the proximity of low-level visual features, but is more related to high-level image semantics (e.g., objects and scenes). Under this circumstance, it is helpful to use some machine learning techniques to partition the low level feature space according to training labels on semantic level. Several *supervised* methods have been proposed recently to learn good hash functions [25], [26], [4], [27]–[30], [6]. In [25], Kulis and Darrell proposed a method to learn hash functions by minimizing reconstruction error between original feature distances and Hamming distances of hash codes. By replacing the original feature distances with semantic similarities, this method can be applied for supervised learning of hash functions. In [26], Lin *et al.* proposed to learn hash functions based on semantic similarities of objects in images. In [4], Wang *et al.* proposed a semi-supervised hashing algorithm that learns hash functions based on image labels. The advantage of this algorithm is that it not only utilizes given labels, but also exploits unlabeled data when learning the hash functions. It is especially suitable for the cases where only a limited number of labels are available. The idea of using a few pairwise data labels for hash function learning was also investigated in [27], using an algorithm called label-regularized max-margin partition. In [28], a scalable graph-based method was proposed to exploit unlabeled data in large datasets for learning hash codes. In [29], Bronstein *et al.* used boosting algorithms for supervised hashing in shape retrieval. Boosting was also used by Xu *et al.* [30] who proposed to learn multiple hash tables. In [6], Salakhutdinov and Hinton proposed a method called semantic hashing, which uses deep belief networks [5] to learn hash codes. Similar to the other supervised hashing methods, the deep belief 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 this method for scalable image search [8], [31].

All these hashing methods, either unsupervised or supervised, share one limitation when applied to image search. As discussed in the introduction, the Hamming distance of hash codes cannot

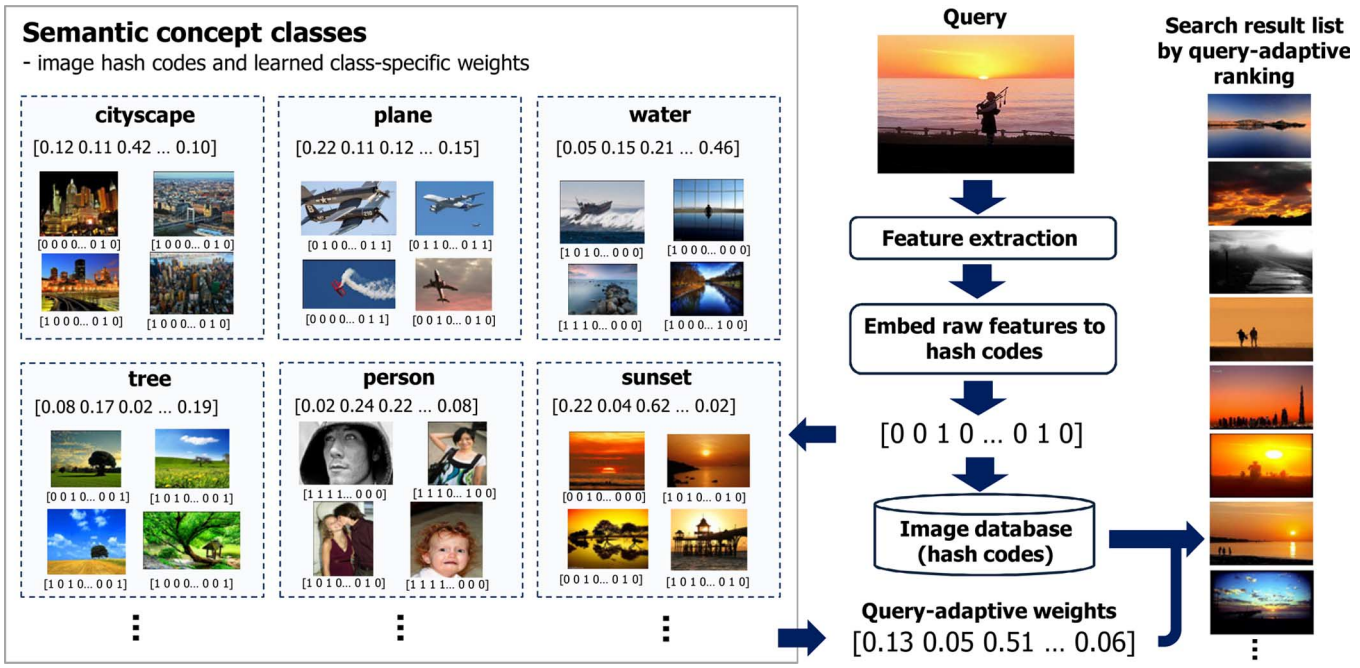


Fig. 3. Framework of query-adaptive image search with hash codes. Given a query image, we first extract bag-of-visual-words feature and embed it into a short hash code (Section IV). The hash code is then used to predict query-adaptive bitwise weights by harnessing a set of semantic concept classes with pre-computed class-specific bitwise weights (Section V). Finally, the query-adaptive weights are applied to rank search results using weighted (query-adaptive) Hamming distance.

offer fine-grained ranking of search results, which is very important in practice. This paper proposes a means to rank images at a finer resolution. Note that we will not propose new hashing methods—our objective is to alleviate one weakness that all the hashing methods share particularly in the context of image search.

There have been a few works using weighted Hamming distance for image search, including parameter-sensitive hashing [32], Hamming distance weighting [33], and the AnnoSearch [34]. Each bit of the hash codes was assigned with a weight in [32], [34], while in [33], the main purpose was 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 [32], [34], or to rescale the final Hamming distance for better matching of sets of features [33], while ours is designed to learn different category-specific bitwise weights offline and adapt to each query online. Another relevant work is by Herve *et al.* [35], who proposed query-adaptive LSH, which is essentially a feature selection process that picks a subset of bits from LSH adaptively for each query. Since their aim was to further reduce search complexity by using less bits, the problem of coarse ranking remains unsolved. This paper extends upon a previous conference publication [36] with additional exploration on query-adaptive hash code selection, more detailed analysis of the weight learning algorithm, and many amplified discussions.

III. System Framework

The proposed query-adaptive image search system is depicted in Fig. 3. To reach the goal of query-adaptive search, we harness a set of semantic concept classes, each with a set of representative images as shown on the left of the figure. Low-level features (bag-of-visual-words) of all the images are embedded into hash codes, on top of which we compute bitwise weights for each of the semantic concepts separately. The weight computation process is done by an algorithm that lies in the very heart of our approach, which will be discussed later in Section V-A.

The flowchart on the right of Fig. 3 illustrates the process of online search. We first compute hash code of the query image, which is used to search against the images in the predefined semantic classes. From there we pool a large set of images which are close to the query in Hamming space, and use them to predict bitwise weights for the query (cf. Section V-B). One assumption made here is that images around the query in Hamming space, collectively, should be able to infer query semantics, and therefore the pre-computed class-specific weights of these images can be used to compute bitwise weights for the query. Finally, with the query-adaptive weights, images from the target database can be rapidly ranked by weighted (query-adaptive) Hamming distance to the query.

IV. Hashing

In this work two state-of-the-art hashing techniques are adopted, *semi-supervised hashing* [4] and *semantic hashing with deep belief networks* [6].

A. Semi-Supervised Hashing

Semi-Supervised Hashing (SSH) is a newly proposed algorithm that leverages semantic similarities among labeled data

while remains robust to overfitting [4], [10]. The objective function of SSH consists of two major components, supervised empirical fitness and unsupervised information theoretic regularization. More specifically, on one hand, the supervised part tries to minimize an empirical error on a small amount of labeled data. The unsupervised term, on the other hand, provides effective regularization by maximizing desirable properties like variance and independence of individual bits. Mathematically, one is given a set of n points, $\mathcal{V} = \{\mathbf{v}_i\}, i = 1 \ldots n$, $\mathbf{v}_i \in \mathbb{R}^D$, in which a small fraction of pairs are associated with two categories of label information, $\mathcal{M}$ and $\mathcal{C}$. Specifically, a pair $(\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{M}$ is denoted as a neighbor-pair when $(\mathbf{v}_i, \mathbf{v}_j)$ share common class labels. Similarly, $(\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{C}$ is called a non-neighbor-pair if the two samples have no common class label. The goal of SSH is to learn d hash functions $(\mathbf{H})$ that maximize the following objective function:

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

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 [10]. This optimization problem is non-trivial. However, after relaxation, the optimal solution can be approximated using eigen-decomposition. Furthermore, an algorithm called semi-supervised sequential projection learning based hashing (S3PLH) was designed to implicitly learn bit-dependent hash codes with the capability of progressively correcting errors made by previous hash bits [10], 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 hash codes. Throughout this paper, we use 5,000 labeled samples for learning hash functions $\mathbf{H}$. Both training (hash function learning) and testing of SSH are very efficient [4], [10].

B. Semantic Hashing With Deep Belief Networks

Learning with deep belief networks (DBN) was initially proposed for dimensionality reduction [5]. It was recently adopted for semantic hashing in large-scale search applications [6], [8], [31]. Like SSH, to produce good hash codes DBN also requires image labels during training phase, such that images with the same label are more likely to be hashed into the same bucket. Since the DBN structure gradually reduces the number of units in each layer,[3] the high-dimensional input of original image features 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 hash code generation, the learning of interactions between variables and the inference of 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 by Hinton *et al.* in [37], [5] 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 hash codes. To obtain 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 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 (hash codes) within a certain neighborhood share the same label [38]. 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 [8], we use a five-layer DBN of sizes $500 - 500 - 500 - 256 - d$, where d is the dimension of the final hash codes. The training process requires to learn $500^2 + 500^2 + 500 \cdot 256 + 256 \cdot d$ weights in total. 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 1000×1000 in the fine-tuning stage. Based on the efficient algorithms described earlier [37], [5], training a set of DBN codes can be finished within 1 day using a fast computer with an Intel Core2 Quad 2 GHz CPU (15–24 hours depending on the output code size). Since parameter training is an offline process, this computational cost is acceptable. Compared to training, generating hash codes with the learned DBN parameters is a lot faster. Using the same computer it requires just 0.7 milliseconds to compute a 48-bit hash code of an image.

V. Query-Adaptive Search

With hash codes, scalable image search can be performed in Hamming space using Hamming distance. By definition, the Hamming distance between two hash codes is the total number of bits at which the binary values are different. Specific indices/locations of the bits with different values are not considered. For example, given three hash 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 sharing the same distance to a query. Going back to the example, suppose we knew that the first two bits are more important (discriminative) for $\mathbf{x}$, then $\mathbf{y}$ should be ranked higher than $\mathbf{z}$ if $\mathbf{x}$ was the query. In this section, we

[3] In practice, the number of units may increase or remain stable for a few layers, and then decrease.

propose to learn query-adaptive weights for each bit of the hash codes, so that images with the same Hamming distance to the query can be ordered in a finer resolution.

A. Learning Class-Specific Bitwise Weights

To quickly compute the query-adaptive weights, we propose to firstly learn class-specific bitwise 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 representative images (training data). We learn bitwise weights separately for each class, with an objective of maximizing intra-class similarity as well as maintaining inter-class relationship. Formally, for two hash 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 bitwise weight vector for class i (j).

Let $\mathcal{X}$ be a set of n hash codes in a d-dimensional Hamming space, $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $\mathbf{x}_j \in \mathbb{R}^d$, $j = 1, \ldots, n$. Denote $\mathcal{X}_i \subset \mathcal{X}$ as the subset of codes from class i, $i = 1, \ldots, k$. Our goal is to learn k weight vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k$, where $\mathbf{a}_i \in \mathbb{R}^d$ corresponds to class i. The learned weights should satisfy a few 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 of the weight values, 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 images (represented by weighted hash codes) from the same class to be close to each other. This can be quantified as follows:

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

where $\mathbf{c}^{(i)}$ is the center of the hash codes in class i:

$$\mathbf{c}^{(i)} = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} \mathbf{x}; \quad (2)$$

n_i is the total number of hash codes in $\mathcal{X}_i$. Notice that although the sample proximity, to some extent, was considered in the hashing methods, (1) is still helpful since weighting the hash codes is able to further condense samples from the same class. More importantly, the optimal bitwise weights to be learned here will be the key for computing the query-adaptive weights.

In addition to intra-class similarity, we also want the inter-class relationship in the original image feature (BoW) space to be preserved by the weighted Hamming distance. 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 hash codes in the two classes 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 fully exclusive. Some of them are actually highly correlated (e.g., *tree* and *grass*). We formalize this idea as a minimization problem of the following term:

$$g(\mathbf{a}_1, \ldots, \mathbf{a}_k) = \sum_{i,j=1}^{k} s_{ij} \left\| \mathbf{a}_i \circ \mathbf{c}^{(i)} - \mathbf{a}_j \circ \mathbf{c}^{(j)} \right\|^2. \quad (3)$$

Integrating both intra-class similarity and inter-class relationship, we propose the following optimization problem to learn the weights for each semantic class:

$$\min_{\mathbf{a}_1, \ldots, \mathbf{a}_k} \quad f(\mathbf{a}_1, \ldots, \mathbf{a}_k) + \lambda g(\mathbf{a}_1, \ldots, \mathbf{a}_k) \quad (4)$$

$$\text{s.t.} \quad \mathbf{a}_i^\top \mathbf{1} = 1, \quad i = 1, \ldots, k, \quad (5)$$

$$\mathbf{a}_i \geq \mathbf{0}, \quad i = 1, \ldots, k, \quad (6)$$

where $\lambda \geq 0$ is a parameter controlling the balance of the two terms.

Let us now show that the above optimization problem can be efficiently solved using an iterative quadratic programming (QP) scheme. First, the term f can be rewritten in matrix form as

$$f(\mathbf{a}_1, \ldots, \mathbf{a}_k) = \sum_{i=1}^{k} \mathbf{a}_i^\top A_i \mathbf{a}_i \quad (7)$$

where A_i is a symmetric positive semidefinite matrix:

$$A_i = \operatorname{diag}\left(\sum_{\mathbf{x} \in \mathcal{X}_i} \left(\mathbf{x} - \mathbf{c}^{(i)} \right) \circ \left(\mathbf{x} - \mathbf{c}^{(i)} \right) \right). \quad (8)$$

In addition, we can expand g as

$$\left\| \mathbf{a}_i \circ \mathbf{c}^{(i)} - \mathbf{a}_j \circ \mathbf{c}^{(j)} \right\|^2 = \mathbf{a}_i^\top C_{ii} \mathbf{a}_i - 2\mathbf{a}_i^\top C_{ij} \mathbf{a}_j + \mathbf{a}_j^\top C_{jj} \mathbf{a}_j \quad (9)$$

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

With these preparations, we can expand and rewrite (4) as

$$\begin{aligned} & f(\mathbf{a}_1, \ldots, \mathbf{a}_k) + \lambda g(\mathbf{a}_1, \ldots, \mathbf{a}_k) \\ &= \sum_{i=1}^{k} \mathbf{a}_i^\top A_i \mathbf{a}_i + \lambda \sum_{j,l=1}^{k} s_{jl} \\ &\quad \times \left(\mathbf{a}_j^\top C_{jj} \mathbf{a}_j - 2\mathbf{a}_j^\top C_{jl} \mathbf{a}_l + \mathbf{a}_l^\top C_{ll} \mathbf{a}_l \right) \\ &= \mathbf{a}_i^\top \left(A_i + 2\lambda \left(\sum_l s_{il} - s_{ii} \right) C_{ii} \right) \mathbf{a}_i \\ &\quad - \left(4\lambda \sum_{j \neq i} s_{ji} C_{ji} \mathbf{a}_j \right)^\top \mathbf{a}_i \\ &\quad + \left(\sum_{j \neq i}^{k} \mathbf{a}_j^\top A_j \mathbf{a}_j + 2\lambda \sum_{j \neq i, l} s_{jl} \mathbf{a}_j^\top C_{jj} \mathbf{a}_j \right. \\ &\quad \left. -2\lambda \sum_{j \neq i, l \neq i} s_{jl} \mathbf{a}_j^\top C_{jl} \mathbf{a}_l \right) \\ &= \frac{1}{2} \mathbf{a}_i^\top Q_i \mathbf{a}_i + \mathbf{p}_i^\top \mathbf{a}_i + t_i, \end{aligned} \quad (10)$$

where

$$Q_i = 2A_i + 4\lambda \left(\sum_l s_{il} - s_{ii} \right) C_{ii}, \quad (11)$$

$$\mathbf{p}_i = -4\lambda \sum_{j \neq i} s_{ji} C_{ji} \mathbf{a}_j, \quad (12)$$

$$t_i = \sum_{j\neq i}^{k} \mathbf{a}_j^\top A_j \mathbf{a}_j + 2\lambda \left(\sum_{j\neq i,l} s_{jl} \mathbf{a}_j^\top C_{jj} \mathbf{a}_j - \sum_{j\neq i,l\neq i} s_{jl} \mathbf{a}_j^\top C_{jl} \mathbf{a}_l \right). \quad (13)$$

We have now derived the quadratic form of (4) 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 (10) is convex, and thus $\mathbf{a}_i$ can be solved optimally. These analysis suggests an iterative procedure summarized in Algorithm 1 for solving the optimization problem stated in (4)–(6).

Algorithm 1 Learning class-specific bitwise weights.

1: INPUT:
Hash codes $\mathcal{X}$;
Class similarity s_{ij} in original image feature space, $i, j = 1, \ldots, k$.
2: Compute $\mathbf{c}^{(i)}$ using (2);
3: Initialize $\mathbf{a}_j = \mathbf{1}/d, j = 1, \ldots, k$;
4: **Repeat**
5: **For** $i = 1, \ldots, k$
6: Compute Q_i, $\mathbf{p}_i$, and t_i using (11)–(13);
7: Solve the following QP problem:

$$\mathbf{a}_i^* = \arg\min_{\mathbf{a}_i} \frac{1}{2}\mathbf{a}_i^\top Q_i \mathbf{a}_i + \mathbf{p}_i^\top \mathbf{a}_i + t_i \quad \text{s.t.} \quad \mathbf{a}_i^\top \mathbf{1} = 1 \text{ and } \mathbf{a}_i \geq \mathbf{0};$$

8: Set $\mathbf{a}_i = \mathbf{a}_i^*$;
9: **End for**
10: **Until** convergence
11: OUTPUT:
Class-specific bitwise weights $\mathbf{a}_j, j = 1, \ldots, k$.

1) Convergence Analysis and Implementation: As stated earlier, given all $\mathbf{a}_j, j \neq i$, the quadratic program in (10) is convex. Solving it with global minimization w.r.t. $\mathbf{a}_i$ will always reduce the value of the energy function in (4), which will definitely not be greater than the energy value derived from a previous iteration. In addition, it is obvious that the energy function is lower-bounded since both terms of the function are non-negative (recall the definitions in (1) and (3)). Therefore, the iterative optimization process given in Algorithm 1 is a Block Coordinate Descent approach, which gradually reduces the energy and leads to convergence at a non-negative value.

In practice, to avoid long time convergence procedure, we define an empirical stop condition (convergence) when the energy difference of two successive states $|\mathcal{E}_{\text{current}} - \mathcal{E}_{\text{previous}}| < \xi$, where $\mathcal{E} = f(\mathbf{a}_1, \ldots, \mathbf{a}_k) + \lambda g(\mathbf{a}_1, \ldots, \mathbf{a}_k)$ and ξ is set as a small value (10^{-6} in our experiments). Having such a stop condition can help avoid unneeded deep optimization that leads to almost invisible changes to the bitwise weights.

2) Connections to Existing Algorithms: We briefly discuss the differences and connections of our proposed algorithm to some well-known machine learning methods such as LDA [39] 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 such 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 [40]. There are also a few exceptions where multiple metrics were considered, e.g., a metric was trained for each category in [41]. These methods are relevant in the sense that they also deal with multiple categories separately. Nevertheless, they cannot be directly applied to our problem, because the class-specific bitwise weights are in a very different form from that of distance metric matrices.

B. Computing Query-Adaptive Bitwise Weights

As described in Fig. 3, images and the learned weights of the predefined semantic concept classes form a *semantic database* for rapid computation of the query-adaptive bitwise weights. Given a query image q and its hash codes $\mathbf{x}_q$, the objective here is to adaptively determine a suitable weight vector $\mathbf{a}_q$, such that weighted Hamming distance (WHD) can be applied to compare $\mathbf{x}_q$ with each hash code $\mathbf{x}_t$ in the target database:

$$\text{WHD}(\mathbf{x}_q, \mathbf{x}_t) = \|\mathbf{a}_q \circ \mathbf{x}_q - \mathbf{a}_q \circ \mathbf{x}_t\|^2. \quad (14)$$

To compute $\mathbf{a}_q$, we query $\mathbf{x}_q$ against the semantic database based on typical Hamming distance. Semantic labels of the top-K most similar images are collected to predict query semantics, which is 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 ith class in $\mathcal{T}$. The query adaptive weights are computed as

$$\mathbf{a}_q = \sum_{i\in\mathcal{T}} (m_i \mathbf{a}_i) \Big/ \sum_{i\in\mathcal{T}} m_i, \quad (15)$$

where $\mathbf{a}_i$ is the precomputed weight vector of the corresponding class. We expect that, although top results from typical Hamming distance lack in good ranking, query semantics may be inferred by collectively using a large pool of samples. We empirically set $|\mathcal{T}|$ as 3 throughout the experiments.

The right side of (14) 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 hash codes. While the weighting can now be achieved by an inner-product operation, we can avoid a significant portion of the computation in practice. The idea is very simple—since a common way of using hashing in a real system is to retrieve images locating only within a certain Hamming radius to a query, we can first find a subset of the hash codes with non-weighted Hamming distance smaller than a certain value to the query (e.g., $\text{Hamming distance} \leq 3$). Then, each of the retrieved hash code in the thresholded subset is re-ranked by computing the weighted Hamming Distance to

the query. In other words, we do not need to order all the hash codes in practice; only the small subset of hash codes with a few bits different from the query is sufficient for pooling the top search results of a query image. In this way, it's possible that some of the hash codes are incorrectly excluded from the initial subset. But the true top matched hash codes with the shortest weighted distances are preserved.

C. Extension to Query-Adaptive Hash Code Selection

The above approach is designed to use a single set of general hash codes for image search. In this subsection, we further extend our approach to the case where multiple sets of hash codes are available. Since there are multiple semantic concept classes, it is very easy to train a set of hash codes for each class by extensively using images containing the corresponding concept. The class-specific hash codes are expected to be more discriminative for a particular class of images. The weight learning algorithm introduced in Section I can be applied to each set of class-specific hash codes to produce a separate group of bitwise weights. During online search, the semantic database is used not only for computing query-adaptive weights, but also to select a suitable set of class-specific hash codes for each query.

Fig. 4 depicts the extended framework. Given a query image, we first compute its *general* (globally trained) hash code and query against the semantic database. Like the query-adaptive weight computation process introduced earlier, we use the same set of images to predict query semantics. Hash codes trained for the dominant concept class in this selected set will be chosen. The query is then embedded into the selected class-specific hash code using hash functions trained for the corresponding concept class. Finally, the new hash code is concatenated with the general code for search, and results are sorted at binary code level based on bitwise weights predicted from a similar procedure as described in Section V.B. This new framework provides a means to adaptively select both hash codes and bitwise weights. Additional computation is required in this extended framework as the query needs to be hashed twice and more bits are used in search. Fortunately, since both testing process (hash code computation) of the hashing algorithms and logical XOR operations are very efficient, this added computational cost is acceptable in general.

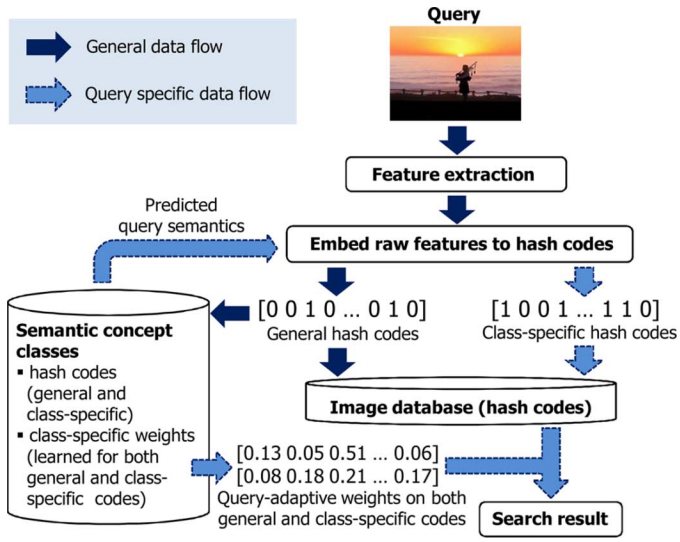


Fig. 4. Extended framework for query-adaptive hash code selection. The semantic concept classes are used to infer query semantics, which guides the selection of a good set of hash codes for the query, and the computation of corresponding query-adaptive weights on the chosen hash codes. The selected class-specific codes are used together with the general codes for image search.

Fig. 5. Example images from several semantic concept classes in NUS-WIDE dataset.

VI. Experimental Setup

A. Dataset, Training, and Performance Measurement

We conduct image search experiments using the widely adopted NUS-WIDE dataset [42]. NUS-WIDE contains 269,648 Flickr images, divided into a training set (161,789 images) and a test set (107,859 images). It is fully labeled with 81 semantic concept classes, covering many topics from objects (e.g., *bird* and *car*) to scenes (e.g., *mountain* and *harbor*). Each image in NUS-WIDE may contain multiple semantic classes. Fig. 5 shows several example images from this dataset. Notice that NUS-WIDE is one of the largest publicly available datasets with complete labels of a wide range of classes. Other well-known and larger datasets such as ImageNet and MIT TinyImages are either not fully labeled[4] or unlabeled, which are therefore not suitable for quantitative performance evaluation.

Local invariant features are extracted from all the images based on Lowe's DoG detector and SIFT descriptor [3]. Soft-weighted bag-of-visual-words features are then computed using a visual vocabulary generated by clustering a subset of SIFT features [43]. The concepts in NUS-WIDE are very suitable for constructing the *semantic database* in our approach. Therefore we use the training images to learn a general set of hash functions by applying labels from every class. On the same training set, we also learn the class-specific hash functions by extensively applying training labels from each class. Class-specific bitwise weights are learned for the general hash functions, as

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

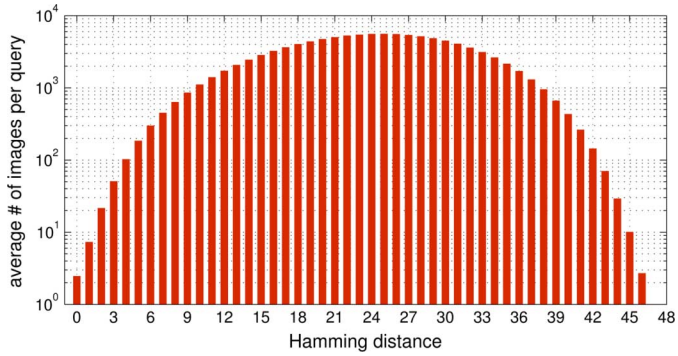


Fig. 6. Average number of returned images at each Hamming distance, computed using 20,000 queries and 48-bit hash codes generated by DBN. Note that the y-axis is plotted in log-scale.

well as for each set of the class-specific hash functions, using the algorithm introduced in Section V. The weight learning algorithm is very efficient; training bitwise weights for a set of hash codes only needs 5 minutes on a regular PC (Intel Core2 Duo 2.4 GHz CPU and 2GB RAM).

For performance measurement, we rank all images in the NUS-WIDE 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. Performance is then evaluated by ΔAP, an extended version of average precision where prior probability, i.e., the proportion of relevant images in the test set to the query, is taken into account [44]. To aggregate performance of multiple queries, mean ΔAP (ΔMAP) is used.

B. Evaluation Plan

Part 1: Characteristics of hash codes based image search. At the beginning of the evaluation, we experimentally verify one weakness of hash codes based image search, i.e., the coarse ranking problem, which is also the main motivation of this work.

Part 2: Query-adaptive ranking. This set of experiments evaluates the proposed framework (Fig. 3) for query-adaptive ranking of search results. Hash codes from both SSH and DBN will be adopted. Specifically, we compare performance of the refined ranking (by the weighted query-adaptive Hamming distance) with original results by traditional Hamming distance, using the two different types of hash codes separately.

Part 3: Query-adaptive hash code selection. The last experiment tests the extension of query-adaptive search to dynamic hash code selection (cf. the extended framework in Fig. 4). Here we only use SSH codes since training DBN codes for 81 concept classes is extremely slow. The purpose of this experiment is to learn *how much performance gain class-specific hash codes could offer*. To the best of our knowledge, similar empirical analysis has never been conducted in previous studies.

VII. Results and Discussions

A. Characteristics of Hash Codes Based Search

Let us first check the number of test images with each Hamming distance value to a query. The 48-bit hash 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 hashing have already shown that codes of 32–50 bits work well in practice [8], [25]. In general, using more bits may lead to higher precision, but at the price of low recall and longer search time.

Fig. 6 visualizes the results, averaged over 20 000 randomly selected queries. As shown in the figure, the number of returned images at each Hamming distance rapidly grows with the distance values until $d/2$ (24). This verifies one nature of Hamming distance, as mentioned in the introduction, there can be C_d^i different hash codes sharing the same integer distance $i(i > 0)$ to a query in a d-dimensional Hamming space. Consequently, the number of hash codes (and correspondingly the number of images) at each specific distance increases dramatically as i grows until $i = d/2$. For some queries, there can be as many as 10^3-10^4 images sharing equal distances. This motivates the need of our proposed approach that provides ranking at a finer granularity. Although our approach does not permute/re-rank images with Hamming distance 0 to the queries, this analysis reveals that this is not a critical issue since most queries have none or just a few such images (2.4 on average in this evaluation).

B. Query-Adaptive Ranking

Next we move on to evaluate how much performance gain can be achieved from the proposed query-adaptive Hamming distance, using 32-bit and 48-bit hash codes from both SSH and DBN (the general sets trained with labels from every class). Fig. 7 displays the results. The parameters λ and K are set as 1 and 500 respectively. We randomly pick 8,000 queries (each contains at least one semantic label) and compute averaged performance over all the queries. As shown in Fig. 7(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 with the SSH codes. The steady improvements clearly validate the usefulness of learning query-adaptive bitwise weights for hash codes based image search.

Fig. 7(b), (d) further shows performance over the upper half part of 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 observe similar performance gain to that over the entire list.

Looking at the two hashing methods, DBN codes are better because more labeled training samples are used (50 k for DBN vs. 5 k for SSH). Note that the comparison of DBN and SSH is beyond the focus of this work, as the latter is a semi-supervised method, which 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 [4].

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 Fig. 8. Significant performance gain is observed for almost all the categories, and none of them suffers from

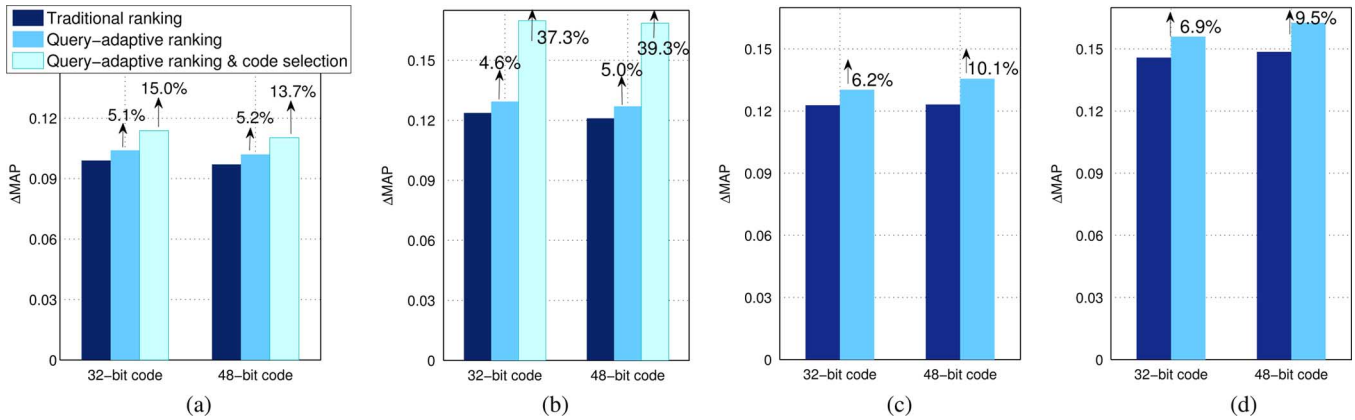


Fig. 7. Search performance comparison of traditional Hamming distance based ranking, query-adaptive ranking, and query-adaptive ranking with code selection. Results are measured over 8,000 queries. We show performance over *entire* and *upper half* of the result rank lists using hash codes computed by SSH and DBN. Performance gains (over the baseline traditional ranking) are marked on top of the query-adaptive search results. (a) SSH, entire list, (b) SSH, upper half, (c) DBN, entire list, (d) DBN, upper half.

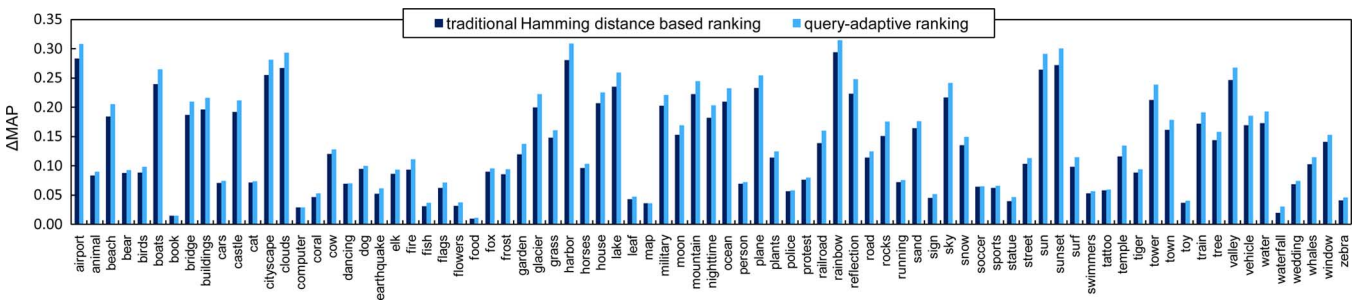


Fig. 8. Per-category performance comparison using 48-bit DBN codes. The queries are grouped into 81 categories based on their associated labels.

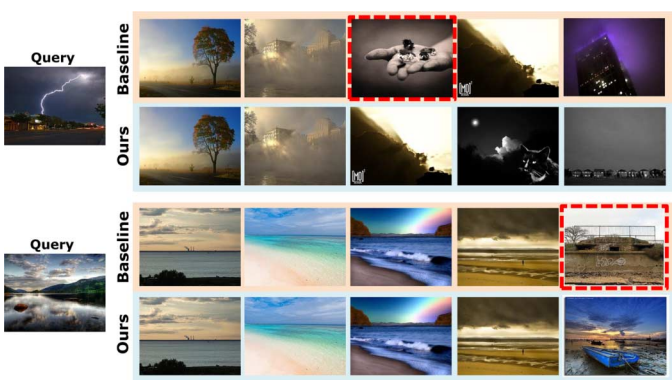


Fig. 9. Top 5 returned images of two example queries, using the 48-bit DBN codes. Our approach returns better results than the baseline traditional Hamming distance, by eliminating irrelevant images (red dotted boxes) from the top-5 lists. Others are all visually relevant (*night-view/outdoor scene* for the first query, and *water-view* for the second one).

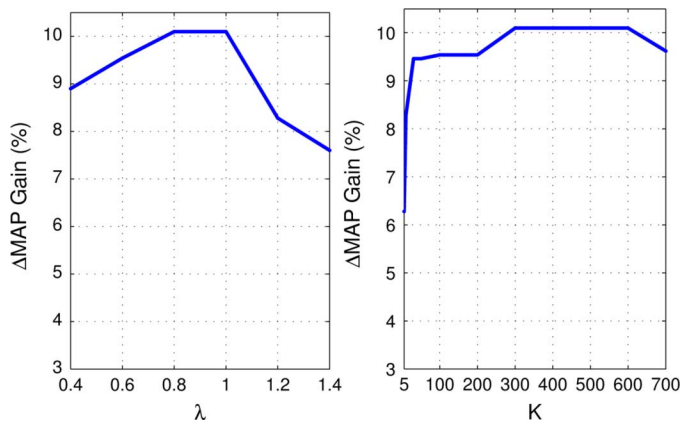


Fig. 10. Performance gain of query-adaptive ranking versus parameters λ (left) and K (right), using DBN codes.

performance degradation. This shows another advantage of our approach—it offers consistently improved results for most queries. Fig. 9 gives top-5 images of two example queries, where the query-adaptive ranking approach produces better results by replacing less relevant images to the queries with more suitable ones.

We also evaluate the effect of parameter λ in class-specific weight learning and K in query-adaptive weight computation, using the DBN codes. Results are visualized in Fig. 10. We see that the performance gain increases with λ first and then decreases, indicating that the inter-class relationship is helpful. For K, we observe a fairly stable performance gain when it is set to a value larger than 50.

C. Query-Adaptive Hash Code Selection

Finally we evaluate query-adaptive hash code selection, following the extended framework described in Section V.C. Fig. 7(a), (b) shows the overall search performance on the same set of queries using SSH codes (the light blue bars on the right). We see obvious performance improvement from

this extension, with overall gains 9.4% (32-bit) and 8.1% (48-bit) over the results obtained by query-adaptive ranking with general hash codes (15.0% and 13.7% respectively over the baseline traditional ranking). Over the upper half of search result lists, the improvement over query-adaptive ranking is very significant: 31.2% and 32.8% for 32-bit and 48-bit codes respectively. These results confirm the effectiveness of the class-specific hash codes and our query-adaptive code selection framework. In addition, we also find that the query-adaptive weights imposed on the selected class-specific hash codes do not contribute as much as that on the general hash codes (around 2%, versus 5–10% on the general codes). This is probably because the hash code selection process already takes query semantics into account by choosing a more suitable set of hash codes for each query. Although the bitwise weights can still improve result ranking, from query-adaptive perspective very limited additional information can be further attained.

While promising, it is worth noting that the query-adaptive hash code selection framework incurs additional computation and memory cost. First, query images need to be hashed twice and search needs to be performed with more bits, which—as mentioned in Section V.C—are generally acceptable since hashing algorithms and bitwise operations are efficient. Second and more importantly, in order not to affect real-time search, we also need to pre-load the class-specific codes of all database samples, which would require 81 times of the memory needed by the general codes. Although this is still much less than that needed by original raw features, the requirement is nontrivial when very large database is in use. Therefore we conclude that class-specific hash codes are useful for improved performance, but this introduces a trade-off between search performance and memory usage that needs to be carefully considered in real-world applications, e.g., per application needs and hardware configuration.

VIII. Conclusion

We have presented a novel framework for query-adaptive image search with hash codes. By harnessing a large set of predefined semantic concept classes, our approach is able to predict query-adaptive bitwise weights of hash codes in real-time, with which search results can be rapidly ranked by weighted Hamming distance at finer-grained hash code level. This capability largely alleviates the effect of a coarse ranking problem that is common in hashing-based image search. Experimental results on a widely adopted Flickr image dataset confirmed the effectiveness of our proposal.

To answer the question of "how much performance gain can class-specific hash codes offer?", we further extended our framework for query-adaptive hash code selection. Our findings indicate that the class-specific codes can further improve search performance significantly. One drawback, nevertheless, is that nontrivial extra memory is required by the use of additional class-specific codes, and therefore we recommend careful examination of the actual application needs and hardware environment in order to decide whether this extension could be adopted.

References

[1] A. Torralba, R. Fergus, and W. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.

[2] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *Proc. IEEE Int. Conf. Computer Vision*, 2003.

[3] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010.

[5] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[6] R. Salakhutdinov and G. Hinton, "Semantic hashing," in *Proc. Workshop of ACM SIGIR Conf. Research and Development in Information Retrieval*, 2007.

[7] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[8] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2008.

[9] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Adv. Neural Inf. Process. Syst.*, 2008.

[10] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *Proc. Int. Conf. Machine Learning*, 2010.

[11] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 12, pp. 1349–1380, Dec. 2000.

[12] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Comput. Surveys*, vol. 40, no. 2, 2008.

[13] J. R. Smith and S.-F. Chang, "VisualSEEk: A fully automated content-based image query system," in *Proc. ACM Int. Conf. Multimedia*, 1996.

[14] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vision*, vol. 42, pp. 145–175, 2001.

[15] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2006.

[16] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Comput. Surveys*, vol. 38, no. 2, 2006.

[17] H. Jegou, M. Douze, and C. Schmid, "Packing bag-of-features," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2009.

[18] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proc. Int. Conf. Computer Vision Theory and Applications*, 2009, pp. 331–340.

[19] P. Indyk, , J. E. Goodman and J. O'Rourke, Eds., "Nearest neighbors in high-dimensional spaces," in *Handbook of Discrete and Computational Geometry*. Boca Raton, FL: CRC, 2004, ch. 39.

[20] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2008.

[21] Y. Jia, J. Wang, G. Zeng, H. Zha, and X.-S. Hua, "Optimizing KD-trees for scalable visual descriptor indexing," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010, pp. 3392–3399.

[22] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. Symp. Theory of Computing*, 1998.

[23] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proc. IEEE Int. Conf. Computer Vision*, 2009.

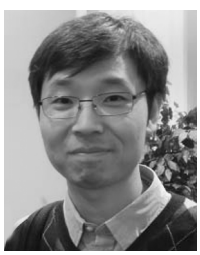
[24] O. Chum, M. Perdoch, and J. Matas, "Geometric min-hashing: Finding a (thick) needle in a haystack," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2009.

[25] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Adv. Neural Inf. Process. Syst.*, 2009.

[26] R.-S. Lin, D. A. Ross, and J. Yagnik, "Spec hashing: Similarity preserving algorithm for entropy-based coding," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010.

[27] Y. Mu, J. Shen, and S. Yan, "Weakly-supervised hashing in kernel space," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010.

[28] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Machine Learning*, 2011.
[29] M. M. Bronstein, A. M. Bronstein, F. Michel, and N. Paragios, "Data fusion through cross-modality metric learning using similarity-sensitive hashing," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010.
[30] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu, "Complementary hashing for approximate nearest neighbor search," in *Proc. IEEE Int. Conf. Computer Vision*, 2011.
[31] E. Horster and R. Lienhart, "Deep networks for image retrieval on large-scale databases," in *Proc. ACM Int. Conf. Multimedia*, 2008.
[32] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. IEEE Int. Conf. Computer Vision*, 2003.
[33] H. Jegou, M. Douze, and C. Schmid, "Improving bag-of-features for large scale image search," *Int. J. Comput. Vision*, vol. 87, pp. 191–212, 2010.
[34] X.-J. Wang, L. Zhang, F. Jing, and W.-Y. Ma, "Annosearch: Image auto-annotation by search," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2006.
[35] H. Jegou, L. Amsaleg, C. Schmid, and P. Gros, "Query adaptative locality sensitive hashing," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, 2008.
[36] Y.-G. Jiang, J. Wang, and S.-F. Chang, "Lost in binarization: Query-adaptive ranking for similar image search with compact codes," in *Proc. ACM Int. Conf. Multimedia Retrieval*, 2011.
[37] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computat.*, vol. 14, no. 8, pp. 1771–1800, 2002.
[38] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov, "Neighbourhood components analysis," in *Adv. Neural Inf. Process. Syst.*, 2004.
[39] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, pp. 179–188, 1936.
[40] E. Xing, A. Ng, M. Jordan, and S. Russell, "Distance metric learning with application to clustering with side-information," *Adv. Neural Inf. Process. Syst.*, pp. 521–528, 2003.
[41] K. Weinberger and L. Saul, "Fast solvers and efficient implementations for distance metric learning," in *Proc. Int. Conf. Machine Learning*, 2008.
[42] 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 *Proc. ACM Int. Conf. Image and Video Retrieval*, 2009.
[43] Y. G. Jiang, C. W. Ngo, and J. Yang, "Towards optimal bag-of-features for object categorization and semantic video retrieval," in *Proc. ACM Int. Conf. Image and Video Retrieval*, 2007.
[44] J. Yang and A. G. Hauptmann, "(Un)reliability of video concept detection," in *Proc. ACM Int. Conf. Image and Video Retrieval*, 2008.



Yu-Gang Jiang received the Ph.D. degree in computer science from the City University of Hong Kong, Kowloon, Hong Kong, in 2009.

During 2008–2011, he was with the Department of Electrical Engineering, Columbia University, New York. He is currently an Associate Professor of computer science with Fudan University, Shanghai, China. His research interests include multimedia retrieval and computer vision.

Dr. Jiang is an active participant of the Annual U.S. NIST TRECVID Evaluation and has designed a few top-performing video analytic systems over the years. His work has led to a best demo award from ACM Hong Kong, the second prize of ACM Multimedia Grand Challenge 2011, and a recognition by IBM T. J. Watson Research Center as one of ten "emerging leaders in multimedia" in 2009. He has served on the program committees of many international conferences and is a Guest Editor of a forthcoming special issue on Socio-Mobile Media Analysis and Retrieval, IEEE TRANSACTIONS ON MULTIMEDIA.

Jun Wang (M'12) received the M.Phil. and Ph.D. degrees from Columbia University, New York, in 2010 and 2011, respectively.

Currently, he is a Research Staff Member in the business analytics and mathematical sciences department at IBM T. J. Watson Research Center, Yorktown Heights, NY. He also worked as an intern at Google Research in 2009, and as a research assistant at Harvard Medical School, Harvard University in 2006. He is the recipient of several awards and scholarships, including the Jury thesis award from the Department of Electrical Engineering at Columbia University in 2011, the Google global intern scholarship in 2009, and a Chinese government scholarship for outstanding self-financed students abroad in 2009. His research interests include machine learning, business analytics, information retrieval and hybrid neural-computer vision systems.

Xiangyang Xue (M'05) received the B.S., M.S., and Ph.D. degrees in communication engineering from Xidian University, Xi'an, China, in 1989, 1992, and 1995, respectively.

He joined the Department of Computer Science, Fudan University, Shanghai, China, in 1995. Since 2000, he has been a Full Professor. His current research interests include multimedia information processing and retrieval, pattern recognition, and machine learning. He has authored more than 100 research papers in these fields.

Dr. Xue is an Associate Editor of the IEEE TRANSACTIONS ON AUTONOMOUS MENTAL DEVELOPMENT. He is also an editorial board member of the *Journal of Computer Research and Development*, and the *Journal of Frontiers of Computer Science and Technology*.

Shih-Fu Chang (S'89–M'90–SM'01–F'04) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1985 and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, in 1991 and 1993, respectively.

He is the Richard Dicker Professor in the Departments of Electrical Engineering and Computer Science, Senior Vice Dean of School of Engineering and Applied Science, and Director of Digital Video and Multimedia Lab at Columbia University. He has made significant contributions to multimedia search, computer vision, media forensics, and machine learning. He has been recognized with ACM SIGMM Technical Achievement Award, IEEE Kiyo Tomiyasu Award, Navy ONR Young Investigator Award, IBM Faculty Award, Recognition of Service Awards from ACM and IEEE Signal Processing Society, and NSF CAREER Award. He and his students have received several Best Paper Awards, including the Most Cited Paper of the Decade Award from Journal of Visual Communication and Image Representation. He is a Fellow of the American Association for the Advancement of Science. He served as Editor-in-Chief for IEEE Signal Processing Magazine (2006–2008), and Chair of Columbia's Electrical Engineering Department (2007–2010).