UNIVERSIDAD POLITÉCNICA DE VALENCIA

MASTER'S FINAL PROJECT

Similarity-Preserving Binary Hashing for Image Retrieval in large databases

Author:
Guillermo GARCÍA FRANCO

Supervisor:
Dr. Roberto PAREDES PALACIOS

June 29, 2012

Abstract

Hashing techniques have become very popular to solve the content-based image retrieval problem in gigantic image databases because they allow to represent feature vectors using compact binary codes. Binary codes provide speed and are memory-efficient. Different approaches have been taken by researchers, some of them based on the Spectral Hashing objective function, among these the recently proposed Anchor Graph Hashing. In this paper an extension to the Anchor Graph Hashing technique which deals with supervised/label information is proposed. This extension is based on representing the samples in an intermediate semantic space that comes from the definition of an equivalence relation in a intermediate geometric hashing. The results show that our approach is a very effective way to incorporate such supervised information to the Anchor Graph Hashing method. On the other hand, the results show that our approach is very effective to deal with clean supervised information but still some further efforts are required in those scenarios where the label information has important presence of noise.

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 Content Based Image Retrieval . . . 3
  - 1.2.1 Content versus Concept . . . 4
  - 1.2.2 Application. Use . . . 4
  - 1.2.3 Content-Based Image Retrieval System Structure . . . 5
  - 1.2.4 Different system models based on query . . . 6
- 1.3 Similarity Search in Information Retrieval . . . 8
- 1.4 Similarity-Preserving Binary Hashing . . . 10

2 Hashing Methods **12**
- 2.1 Unsupervised and Supervised Learning Methods . . . 12
- 2.2 Notation . . . 12

3 Unsupervised Hashing Methods **13**
- 3.1 Locality Sensitive Hashing (LSH) . . . 13
- 3.2 Spectral Hashing (SH) . . . 14
- 3.3 Iterative Quantization (ITQ) . . . 15
- 3.4 Binary Reconstructive Embeddings (BRE) . . . 17
- 3.5 Minimal Loss hashing for Compact Binary Codes (MLH) . . . 18
- 3.6 Anchor Graph Hashing (AGH) . . . 19

4 Supervised Hashing Methods **20**
- 4.1 Supervised Anchor Graph Hashing (SAGH) . . . 20
  - 4.1.1 The proposed SAGH approach . . . 21
  - 4.1.2 Hashing query images . . . 22
- 4.2 Supervised Multiple Random Anchor Graph Hashing (SM-RAGH) . . . 23

5 Literature Experimentation **26**
- 5.1 Measures . . . 26
- 5.2 Datasets used in literature . . . 26

6 Experiments **28**
- 6.1 CIFAR-11 . . . 28
- 6.2 NUS-WIDE . . . 29

7 Conclusions **35**

1 Introduction

1.1 Motivation

With the advance of multimedia technology and the Internet, we have at our disposal billions of images available online. As the amount of the data available continues to grow, methods to perform efficient searches on these huge databases becomes vital for many applications. One important application for this technology is the visual search or content based image retrieval (CBIR) on large collections of images, such as those on the Internet or personal collections. Another important application is image annotation using data-driven methods. These methods produce an image annotation by means of searching very similar images in a large-scale database following a search-to-annotation strategy. In general, these methods derive the content of an image by propagating the labels of its similar images.

The main objective is thus to be able to retrieve the most similar images from a large, and possibly distributed database of images. Consequently, search methods should be memory efficient, allowing to store millions of images, and also should perform a fast similarity search.

In order to find similar images in a large-scale database, several approaches from the approximate nearest neighbors search literature are designed to solve this problem. The most successful methods can be mainly split into two different categories: Tree-decomposition methods and Hashing methods. Tree-decomposition methods store the reference samples in a tree structure providing in average, an approximate nearest neighbor search in logarithmic time with respect to the number of samples. The main drawback of these methods when working with images is that images are generally represented using a very high dimensional vector. In such a situation, tree indexing structures become inefficient due to an increasing need of backtracking to explore all the nodes. In order to alleviate this problem and focused on the high dimensional representation of images, the Hashing techniques emerged as a solution to the approximate nearest neighbor search in high dimensional spaces, [11, 2].

Hashing methods map the high-dimensional representation into a binary representation with a fixed number of bits. Binary codes are very storage-efficient, since relatively few bits are required, millions of images can be stored into computer memory. Moreover, computing the hamming distance for binary codes is very fast, as it can be performed efficiently by using bit XOR operation and counting the number of set bits [13, 27].

The hash function design is crucial, this being mainly the unique difference among all these methods. Generally the different methods learn a hash function that preserves the topology of the samples in the original space, i.e. images that are near in the original high dimensional space share the same (or similar) binary code, while images far in the original space have very different binary codes. These methods work with unsupervised information, thus the preservation of the geometric topology is the unique goal to pursue. However, when there is additional information available, which could be supervised (i.e. labels annotated by a human) or it could also be what currently is being called privileged information (i.e. information available only for the training samples, such as text related to an image) [25], better performance can be obtained by methods which try to preserve the semantic topology. Since images visually different could contain similar semantic concepts, in these cases the hash code should be designed to (also) preserve the semantic topology.

The document is organized in the following way. This section will introduce the problem of concept based image retrieval, its application and different system configurations. It will aso define the problems when handling large databases introducing the concepts of similarity search and binary hashing. Section 2 will briefly introduce the concepts of unsupervised and supervised methods for image retrieval and the notation used along the document. The following sections 3 and 4 will be focused in different unsupervised and supervised methods in the literature. Section 4 will be completely focused in the proposed supervised methods. Section 5 contains the different measures and databases used in the experimentation found in the literature, and in the following section 6 presents the results obtained with the proposed methods in two databases. Finally section 7 has the conclusions and future work.

1.2 Content Based Image Retrieval

Content-based image retrieval (CBIR) is the application of computer vision techniques to an image retrieval problem, which is the problem of searching for digital images in large databases.

Content based image retrieval is opposed to a *concept based* approach. While the second one relies in metadata such as keywords, tags or descriptions associated with the image, the first one analyses the actual contents of the images. The content of the image is information that can be derived from the image itself such as colors, shapes or textures.

Figure 1: Image Retrieval: the user makes a query and the system retrieves the most relevant images to the given query.

1.2.1 Content versus Concept

Nowadays most web based image search engines use concept-based search engines that rely on metadata. The problem with this approach is that we find a lot of garbage in the results.

- Language: ambiguities, such as polysemy or synonymy (with synonyms we can miss images. Some systems use supergroups -categorizing images in semantic classes-, but still scaling issue).
- Textual information: Even though easy to search with existing technology, impractical for large databases or automatically generated images (surveillance cameras) because it requires humans to personally *describe with words* every image in the database.
- Human Tagging: apart from being expensive and inefficient for large databases also leads to errors tagging, miswritten or incorrect tags, images labelled differently by different users and they may not tag every concept in the image.

1.2.2 Application. Use

Potential uses for CBIR include:

- Art collections

- Photograph archives
- Retail catalogs
- Medical diagnosis
- Crime prevention
- The military
- Intellectual property
- Architectural and engineering design
- Geographical information and remote sensing systems

1.2.3 Content-Based Image Retrieval System Structure

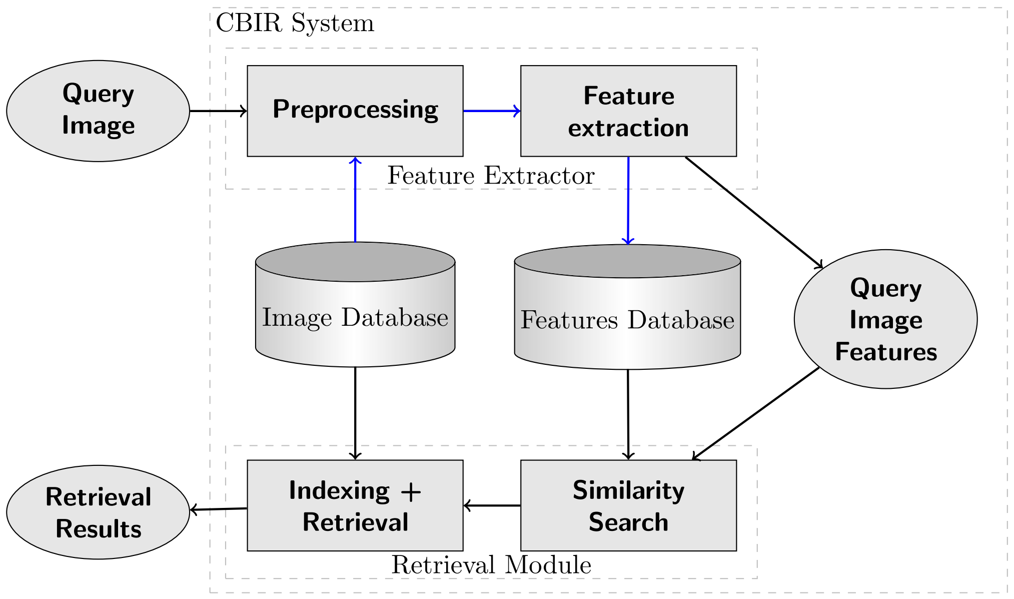


Figure 2: Diagram of a CBIR system. The training process is done off-line and is represented with blue arrows. The process of an user query is performed on-line and is represented with black arrows.

Query Image This component describes the input to the system, it is the image which is taken as the input query for the search operation. User is suppose to select an image in order to find similar images for that particular image.

Feature Extractor This component is an important component of the system. It mainly extracts the information from the image. Input images are processed to extract features in order to represent the image contents in numerical form. These feature vectors are generally used as the image signature and they can be thought of as points in a high-dimensional space. Every image is assigned with one of this set of identifying descriptors which will be used in the retrieval/matching phase to retrieve relevant images.

Image Database This is the set of all the images that the system is going to provide as a result to the user queries.

Features Database This is where all the information extracted from the training images is stored. Feature vectors from all the training images are extracted and stored. The system will use these to perform feature comparison during the similarity search process. Features might be stored as a simple list or set of feature vectors or in more complexes structures such as k-d-trees, metric trees, etc.

Similarity Search This component basically does the comparison part and returns its own output according to the technique it uses to do the comparison. The system provides similarity scores for each one of the images in its image database. This process is based on some similarity measure to compute/measure the distance between the query image descriptors and the feature database. From the system's viewpoint the similarity of two images depends on the distance in feature space between the feature points defined by the descriptors. Therefore, the shorter the distance between the two points, the more similar the corresponding images are.

Indexing and Retrieval The system selects the number of images to present to the user as a result to the query.

1.2.4 Different system models based on query

There are different ways of providing the user query in a CBIR system.

Query by example

System is provided with an example image to base its search upon. The result images should share common elements/content with the provided image.

The user can provide images in very different ways:

- Image provided by the user (uploaded, camera, etc)
- Preexisting image selected from random set (directly or browsing customized/hierarchical categories).
- Visual sketch: the user draws a rough approximation of the image they are looking for (p.e. with blobs of colors or general shapes)

Instead of giving as a query a whole example image, the user can

- Query by image region (rather than the entire image),
- Query by multiple example images,
- Query by direct specification of image features

This query technique removes the difficulties that can arise when trying to describe images with words or labels.

Semantic retrieval

The user makes a semantic retrieval. Users can directly request certain images like "find pictures of white cats". This type of open-ended task is very difficult for computers to perform.

In order to provide semantic-based retrieval in a CBIR system images need to be indexed by some textual information. Users usually prefer querying images by keywords. As manual indexing is a tedious task for indexers and leads to some problematic as we discussed before in 1.2.1, annotating images automatically would be very useful for semantic-based image retrieval [16, 21].

Relevance Feedback

CBIR systems can make use of relevance feedback, where users progressively refine the search results by marking the result images as "relevant", "not relevant", or "neutral" to the search query. Then the search is repeated using this new information. 3 shows a diagram of this process. After the first retrieval, the user is asked to provide positive (relevant) and negative (irrelevant) examples as feedback among the initial retrieved image sets. Then this information is used as positive and negative feedback for query refinement.

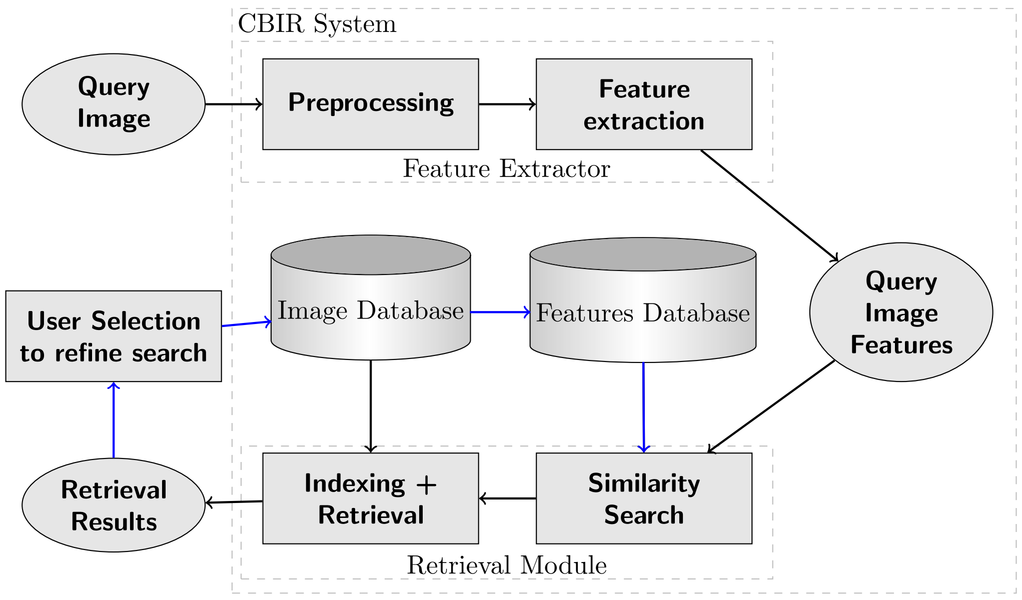


Figure 3: Diagram of a CBIR system with relevance feedback. The feedback process is marked with red arrows.

1.3 Similarity Search in Information Retrieval

In the field of Information Retrieval (IR), the problem of **Similarity Search** (nearest neighbour search) is given a query document find its most similar documents from a large document collection (corpus). In the specific case of CBIR we want to find the most similar images in our corpus to a given image.

Similarity Search Given a set $\mathcal{S}$ of points in a metric space $\mathcal{M}$ and a query point $\boldsymbol{x_z} \in \mathcal{M}$, similarity search consist on finding the closest point in $\mathcal{S}$ to $\boldsymbol{x_z}$.

Nowadays image databases might contain millions of images so exhaustive search is not applicable, as it will be too slow. Nearest neighbour search should meet this requirements in order to be applicable to large databases problems:

- Memory efficient: Store millions of images in memory efficiently.
- Fast similarity search: Quickly find similar images to a target image.

Approximated Techniques

Recent work is centered in **approximate techniques**. Computing exact nearest neighbors in dimensions seems to be a very difficult task, but for image search applications sometimes it may be sufficient to retrieve a "good guess" of the nearest neighbor, that is, an **Approximate Nearest Neighbor (ANN)** [3]. We can use an algorithm which doesn't guarantee to return the actual nearest neighbor in every case, although with significantly faster running times and relatively small errors.

Definition Given an $\epsilon > 0$, a $(1+\epsilon)$-ANN, to a query point $\boldsymbol{x_z}$, is a point $\boldsymbol{y} \in \mathcal{S}$, such that

$$d(\boldsymbol{x_z}, \boldsymbol{y}) \leq (1+\epsilon) d(\boldsymbol{q}, n_{\boldsymbol{x_z}})$$

where $n_{\boldsymbol{x_z}} \in S$ is the nearest neighbor to $\boldsymbol{x_z}$ in $\mathcal{S}$.

Approaches for fast NNS

In order to solve the problem of efficiently finding similar images in a large-scale database, several approaches have been designed. The most successful methods can be mainly split into two different categories: Tree-decomposition methods and Hashing methods.

Tree-decomposition methods partitions the data recursively, storing the reference samples in a tree structure providing in average, an approximate nearest neighbor search in logarithmic time with respect to the number of samples and degenerating to a linear search in the worst case scenario.

This category includes algorithms such as k-d-trees [4], M-trees [6], cover trees [7], metric trees [24], and other related techniques.

The main drawback when working with images is that these are generally represented using very high dimensional vectors. In such situation, tree indexing structures become inefficient due to an increasing need of backtracking to explore all the nodes. In order to alleviate this problem and focused on the high dimensional representation of images, the Hashing techniques emerged as a solution to the approximate nearest neighbor search in high dimensional spaces, [11, 2].

Hashing methods map the high-dimensional feature vector representation into a binary representation with a fixed number of bits.

Locality-sensitive hashing [11] has been the most popular method, and extensions have been explored in order to accommodate different distances like l_p norms [8], learned metrics [12], and image kernels [1]. Algorithms based on LSH come with the guarantee that the approximate nearest neighbors (neighbors within $(1+\epsilon)$ times the true nearest neighbor distance) may be found in time that is sublinear in the total number of database objects (but as a function of ϵ). Several recent methods have explored ways to improve upon the random projection techniques used in LSH. Some of them will be discussed later in sections 3 and 4.

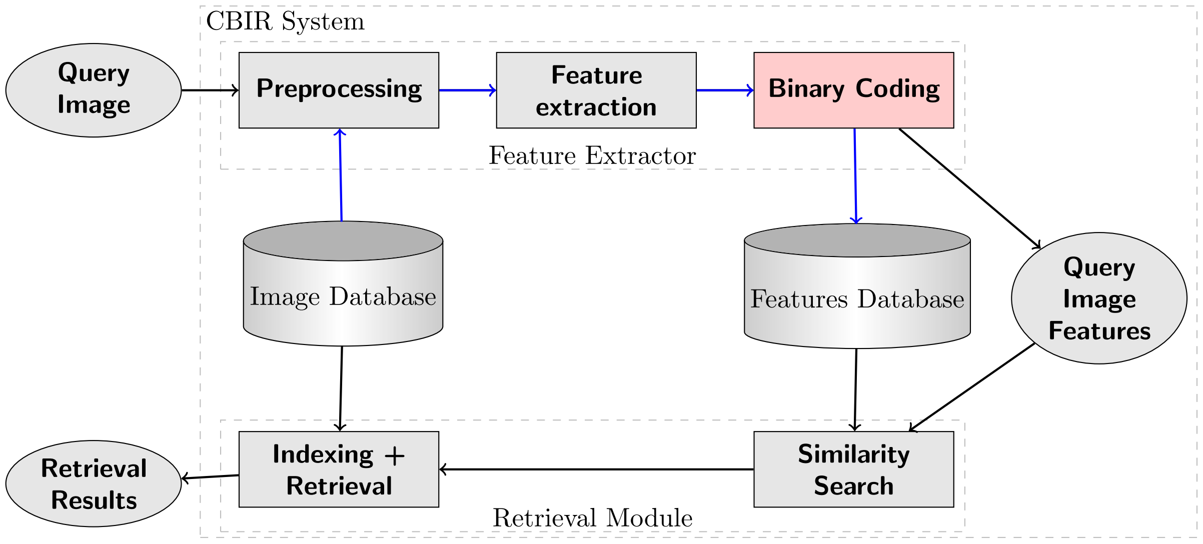


Figure 4: Diagram of a CBIR system using hashing methods for feature representation.

1.4 Similarity-Preserving Binary Hashing

Binary codes are very storage-efficient, since relatively few bits are required, millions of images can be stored into computer memory. Moreover, binary embeddings allow extremely fast similarity search operations, as computing the hamming distance for binary codes can be performed efficiently by using bit XOR operation and counting the number of set bits [13, 27].

This thesis is going to be centered in the hashing approach to Approximate Nearest Neighbors, also referred to as **Similarity-Preserving Binary Hashing**:

- Given a hash function, each item $\boldsymbol{x_i}$ is mapped into a binary code $\boldsymbol{y_i}$.
- Similar items are mapped to similar binary codes.

- Given a query point $\boldsymbol{x_z}$ one applies the hash functions to the query $\{h_1(\boldsymbol{x_z}), ..., h_q(\boldsymbol{x_z})\}$, obtains a query code $b(\boldsymbol{x_z}) = thr(\boldsymbol{y_z})$ and finds all the database entries that are within a small hamming distance from the query code.

Using **binary codes** as hash key indexes, search becomes sublinear. Calculating hamming distance in binary codes is very fast, and binary codes are also very storage-efficient.

- Easily computed for a novel input. Hamming distance between two binary codes can be computed efficiently by using bit XOR operation and counting the number of set bits [13, 27]
- Small number of bits required to code the full dataset. As the encoded data is highly compressed it can be loaded into the main memory.

Therefore, using binary hash codes we can reduce storage requirements and accelerate search and retrieval in large collections of high-dimensional data.

Standard simple binary hashing function is a thresholded linear projection followed by binary quantization, where x is the input vector, $\boldsymbol{W}$ the parameter matrix, thr represents binary quantization to produce a binary vector.

$$b(\boldsymbol{x}) = thr(\boldsymbol{W}\boldsymbol{x})$$

- Each row of $\boldsymbol{W}$ defines an hyperplane in the input space.
- Two datapoints share the same k^{th} bit if they fall on the same side of the hyperplane defined by $\boldsymbol{w_k}$ (the k^{th} row of $\boldsymbol{W}$).
- There's one hyperplane for each bit, and the hamming distance between the binary codes of two points is the number of hyperplanes that separate them.

2 Hashing Methods

The most common methods for solving large scale image retrieval problems are hashing methods. This methods approximate nearest neighbor search, converting each vectorized image on the database in a compact binary code, and providing sublineal search cost.

2.1 Unsupervised and Supervised Learning Methods

We can approach the problem of image retrieval in two different scenarios, depending if the training data is labelled (*supervised*) or not. In machine learning, unsupervised learning refers to the problem of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning. In supervised learning the training data consist of a set of training examples where each example is a pair consisting of an input object (typically a vector; in image retrieval would be a vectorized image) and a desired output value (also called the supervisory signal) which in the case of image retrieval is the set of labels associated to each image.

2.2 Notation

Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ be the set of n feature vectors extracted from training images and represented in a d-dimensional space. The goal is to learn a binary embedding function of q bits:

$$f : \mathbb{R}^d \longrightarrow \{-1, 1\}^q \tag{1}$$

where for convenience the binary symbols have been defined as -1 and 1. The training set $\mathcal{X}$ produces the set of binary codes $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \{-1, 1\}^q$, and for an arbitrary input test sample, the same mapping would be used so that the hamming distance can be employed to find its nearest neighbors from the training set.

Additionally, in the supervised scenario we assume that each training sample $\mathbf{x}_i$ has an associated label vector $\mathbf{t}_i \in \mathbb{R}^l$ which provides semantic information about the sample. Usually the label vector $\mathbf{t}_i$ is a binary vector indicating the presence or absence of each one of the l terms, $\mathbf{t}_i \in \{-1, 1\}^l$.

3 Unsupervised Hashing Methods

In this section we are going to describe unsupervised methods found in the literature. In the unsupervised scenario, feature vectors extracted from the images are the only input provided to the hashing method.

3.1 Locality Sensitive Hashing (LSH)

LSH [9] solves the approximate or exact Near Neighbor Search in high dimensional spaces. The basic idea of this method is to hash the input items so that similar items are mapped to the same buckets with high probability. That means that the probability of collision is much higher for closer objects than for those that are further apart. This kind of hash functions are called locality-sensitive hash functions.

Definition Consider a family $\mathcal{H}$ of hash functions mapping $\mathbb{R}^d$ to some universe U. A family $\mathcal{H}$ is called (R, cR, p_1, p_2)-sensitive if for any $\boldsymbol{x_i}, \boldsymbol{x_j} \in \mathbb{R}^d$

- if $||\boldsymbol{x_i} - \boldsymbol{x_j}|| \leq R$ then $Pr_{\mathcal{H}}[h(\boldsymbol{x_i}) = h(\boldsymbol{x_j})] \geq p_1$,
- if $||\boldsymbol{x_i} - \boldsymbol{x_j}|| \geq cR$ then $Pr_{\mathcal{H}}[h(\boldsymbol{x_i}) = h(\boldsymbol{x_j})] \leq p_2$.

A standard random hyperplane locality-sensitive hashing generates embeddings via random projections. Each hash function h_k is generated independently by selecting a random vector $\boldsymbol{w}_k$ from a multivariate Gaussian with zero-mean and identity covariance. Then the hash function is given as $h_k(\boldsymbol{x}) = sign(\boldsymbol{x}\boldsymbol{w}_k)$.

$$\boldsymbol{Y} = sign(\boldsymbol{X}\boldsymbol{W})$$

If every bit in the code is calculated by a random linear projection followed by a random threshold, then as the number of bits increases the Hamming distance between codewords converges to the cosine angle between them. This means that the Hamming distance between codewords asymptotically approaches the Euclidean distance between the items, so therefore this method maintains the topology of the input data in the limit as the number of bits increases.

Random projections are used in LSH [11] and related methods. These methods are dataset independent, make no prior assumption about the data distribution, and come with theoretical guarantees that specific metrics (e.g., cosine similarity) are increasingly well preserved in Hamming space as the code length increases. But they require large code lengths for good retrieval accuracy, and they are not applicable to general similarity measures, like human ratings. This is the main reason why there's been a growing interest in **learning the hash functions** instead.

Basically, learning hash binary functions provides those two advantages:

- codes can become more compact
- more general classes of similarity measures can be preserved (p.e. based in human labels, which might not correspond to any measure distance).

Some key learning-based hash approaches:

- Parameter Sensitive Hashing [22]: boosting (Shakhnarovich et al 2003)
- Semantic Hashing [20]: Neural Networks (Salakhutdinov and Hilton 2007)
- Spectral hashing [28]: Spectral Methods (Weiss et al 2008), assumes uniform distribution over the data.
- Loss-based methods [15, 18]

Binary codes

Most of the methods that learn hash functions use some restrictions in order to make sure that the resulting binary codes are short and make a good use of each one of their bits.

- maximize variance of each bit
  $\max(\sum_k var(h_k(\boldsymbol{x})))$
- bits independent/balanced
  $\sum_i(\boldsymbol{y_i})) = 0$ for $i = \{1, ..., n\}$
- bits pairwise uncorrelated minimizes the redundancy among bits
  $\frac{1}{n}\sum_i(\boldsymbol{y_i}\boldsymbol{y_i}^T) = \boldsymbol{I}$

3.2 Spectral Hashing (SH)

Let $A_{n\times n}$ be the affinity matrix for the n datapoints which elements a_{ij} indicate the similarity between training samples i and j. The average Hamming distance between similar neighbors can be written as a sum of similarity-weighted squares of differences between codes.

$$\sum_{ij} A_{ij}||y_i - y_j||^2$$

Relaxing the bits independence assumption and requiring the bits to be uncorrelated we obtain the following problem proposed in []:

$$\begin{aligned} &\min_{i,j} \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 \\ &s.t. \quad \boldsymbol{y}_i \in \{1,-1\}^q, \ \ \sum_i \boldsymbol{y}_i = 0, \ \ \frac{1}{n}\sum_i \boldsymbol{y}_i\boldsymbol{y}_i^T = \boldsymbol{I} \end{aligned} \tag{2}$$

The solution to this objective function is going to provide similar codes $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ for similar samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. This previous objective function is commonly expressed in terms of the graph Laplacian matrix $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ as:

$$\begin{aligned} &\min_{\boldsymbol{Y}} \frac{1}{2}\sum_{i,j=1}^{n} \|\mathbf{y}_i - \mathbf{y}_j\|^2 a_{ij} = \mathsf{Tr}(\boldsymbol{Y}^T\boldsymbol{L}\boldsymbol{Y}) \\ &s.t. \quad \boldsymbol{Y} \in \{1,-1\}^{n\times q}, \ \ \boldsymbol{1}^T\boldsymbol{Y} = \boldsymbol{0}, \ \ \boldsymbol{Y}^T\boldsymbol{Y} = n\boldsymbol{I}_{q\times q} \end{aligned} \tag{3}$$

where $\boldsymbol{L} = \text{diag}(\boldsymbol{A1}) - \boldsymbol{A}$.

The first restriction assures to generate codes with balanced bits, and the second one minimizes the redundancy among bits forcing the codes to be orthogonal. This restrictions avoid having a closed solution where all codes are the same.

Spectral relaxation could be applied to make this NP-hard problem tractable, dropping the integer constraint and allowing $\boldsymbol{Y} \in \mathbb{R}^{n\times q}$. With this, the solution $\boldsymbol{Y}$ would be the r eigenvectors of length $\sqrt{n}$ corresponding to the q smallest eigenvalues (ignoring eigenvalue 0) of the graph Laplacian L.

3.3 Iterative Quantization (ITQ)

This is a simple and efficient alternating minimization scheme for finding a rotation of zero-centered data so as to minimize the quantization error of mapping this data to the vertices of a zero-centered binary hypercube.

After centering the input feature vectors, in order to find codes with maximum variance and pairwise uncorrelated, input data $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is projected using an unsupervised data embedding such as PCA, even though it can be used with other unsupervised or supervised embeddings such as canonical correlation analysis (CCA).

If $\boldsymbol{W} \in \mathbb{R}^{d \times q}$ is the matrix having as column vectors the hyperplane coefficients $\boldsymbol{w}_k$ obtained using PCA, each bit $k = 1, ..., q$ can be encoded with the function $h_k(\boldsymbol{x}) = sgn(\boldsymbol{x}\boldsymbol{w}_k)$.

$$h_k(\boldsymbol{x}) = sgn(\boldsymbol{x}\boldsymbol{w_k}) = sgn(\boldsymbol{v})$$

The entire encoding process would be:

$$\boldsymbol{Y} = sgn(\boldsymbol{X}\boldsymbol{W}) = sgn(\boldsymbol{V})$$

If $\boldsymbol{W}$ is an optimal solution, so is $\boldsymbol{W}\boldsymbol{R}$, being $\boldsymbol{R}$ an orthogonal $q \times q$ matrix. Therefore, the projected data $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{R}$ can be orthogonally transformed. ITQ will orthogonally transform the projected data in order to minimize the quantization loss.

$$\boldsymbol{Y} = sgn(\boldsymbol{X}\boldsymbol{W}\boldsymbol{R}) = sgn(\boldsymbol{V}\boldsymbol{R})$$

Definition Let $\boldsymbol{v} \in \mathbb{R}^q$ be a vector in the projected space. It is easy to show that $sgn(\boldsymbol{v})$ is the vertex of the hypercube $\{-1, 1\}^q$ closest to $\boldsymbol{v}$ in terms of Euclidean distance. *Quantization loss* is the difference obtained when adjusting the real projected data $\boldsymbol{v}$ into the binary code hypercube $\{-1, 1\}^q$.

$$||sgn(\boldsymbol{v}) - \boldsymbol{v}||_2$$

The smaller the quantization loss, the better the resulting binary code will preserve the original locality structure of the data. They are going to find an orthogonal rotation, such that the projected points are as closest as possible to their binary quantization (minimizing distances to their corresponding point in the zero-centered binary hypercube in hamming space).

$$min(\mathcal{Q}(\boldsymbol{Y}, \boldsymbol{R}))$$

$$\mathcal{Q}(\boldsymbol{Y}, \boldsymbol{R}) = ||\boldsymbol{Y} - \boldsymbol{V}\boldsymbol{R}||_F$$

We can see in the quantization loss function the connection of ITQ to the *orthogonal Procrustes problem.*

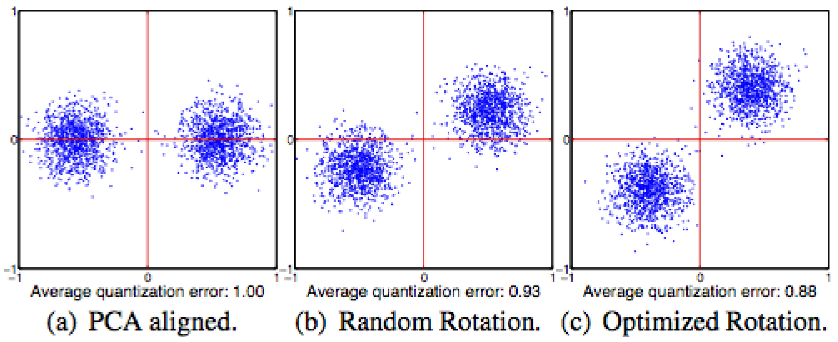


Figure 5: ITQ Method. Left: PCA aligned rotation. Center: Random rotation. Right: Optimized Rotation

3.4 Binary Reconstructive Embeddings (BRE)

BRE [15] is a learning-based binary hashing method based on optimizing a loss function: minimize the reconstruction error between the original distances and the Hamming distances of the corresponding binary embeddings via an scalable coordinate-descent algorithm.

In order to compute the q-dimensional binary embedding, the data is projected using a set of q hash functions $h_1, ..., h_b$. Each hash function h_i is a binary-valued function. An input feature vector $\boldsymbol{x}_i$ would produce a low-dimensional binary reconstruction $\boldsymbol{y}_i = [h_1(\boldsymbol{x}_i); h_2(\boldsymbol{x}_i); ...; h_q(\boldsymbol{x}_i)]$.

In article [15] the hash functions are dependent on one another.

$$h_i(\boldsymbol{x}) = sign(\sum_{j=1}^{k} \boldsymbol{W}_{ij}\kappa(\boldsymbol{x}_{ij}, \boldsymbol{x}))$$

Loss function that penalizes the difference between euclidean distance in the input space and the hamming distance between binary codes.

$$\mathcal{O}(x_{i=1}^{n}, W) = \sum_{(i,j)\in\mathcal{N}} (d(x_i, x_j) - \tilde{d}(x_i, x_j))^2 = \sum_{(i,j)\in N} (\frac{1}{2}||x_i - x_j||^2 - \frac{1}{q}||\tilde{x_i} - \tilde{x_j}||^2)$$

This objective is not continuous nor differentiable, their first approach was using a sigmoid function, but minimizing $\mathcal{O}$ with that approach and using a quasi-Newton L-BGFS resulted in poor local optima. They consider fixing all but one weight W_{pq}, and optimizing $\mathcal{O}$ with respect to W_{pq}. An optimal update of this weight can be achieved in $O(nlogn + nk)$. Such approach will update a single hash function h_p. We can update all functions in $O(nq(k + logn))$

3.5 Minimal Loss hashing for Compact Binary Codes (MLH)

- less training time than BRE
- based on structured prediction with latent variables
- Optimizes empirical loss function
- Applicable to general similarity measures (p.e. Human ratings)

In MLH [18] the loss function assigns a cost given two binary codes $\boldsymbol{y}_i, \boldsymbol{y}_j$ and the similarity label s between them.

$$L : \{-1,1\}^q \times \{-1,1\}^q \times \{0,1\} \rightarrow \mathbb{R} \tag{4}$$

This loss function L has to measure how compatible are the codes with the similarity label, and should assign a small cost when $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are nearby and a large cost when they are not.

They want to minimize empirical loss function $\mathcal{L}$, defined over a subset of training pairs with similarity labels.

$$\mathcal{L}(W) = \sum_{(i,j)\in\mathcal{S}} L(b(x_i;w), b(x_j;w), s_{ij})$$

$$s_{ij} = \begin{cases} 1 & \text{if } x_i \text{ and } s_j \text{ are similar} \\ 0 & \text{otherwise} \end{cases}$$

They define a hinge-like loss function to capture the similarity principles said before, defines notions of far and near in hamming space using a parameter ρ and the hamming distance between two codes h and g.

$$L(h,g,s) = l_\rho(||h-g||_H, s)$$

$$l_\rho(m,s) = \begin{cases} max(m-\rho+1,0) & \text{for } s=1 \\ \lambda max((\rho-m+1,0) & \text{for } s=0 \end{cases}$$

As long as codes of similar items are within hamming distance ρ bits they have cost zero and beyond that the cost linearly increases. For dissimilar items codes further than rho bits have cost zero and then as they come closer their cost linearly increases. Their objective depends on the differences between codes instead of the actual codes.

They maximize an upper bound on the loss function (motivated by structural SVM formulations) because it is continuous and non-convex.

Coordinate descent algorithm. Initialize W randomly and then iterate over pairs, computing the codes, solving the loss-adjusted inference and updating W en each step.

3.6 Anchor Graph Hashing (AGH)

In Spectral Hashing (3.2) relaxation solved the problem for similarity search using binary codes computing the q eigenvectors corresponding to the q smallest eigenvalues of the graph Laplacian $\boldsymbol{L}$. Anyway, the problem is still computationally expensive because of the computation of the underlying graph and the Laplacian when n is too large.

Based on the Spectral Hashing approach, the authors in [17] introduced a very effective approach for image hashing: Anchor Graph Hashing (AGH). This unsupervised technique aims at capturing and preserving the semantic topology assuming that close-by points usually share labels. The solution proposed to deal with the problem in Spectral Hashing is to avoid computing the whole similarity matrix $\boldsymbol{A}$ for all the n samples. To this end, a small set of m points being $m \ll n$ called anchors, are selected (e.g. using k-means clusters). With these anchors, the matrix $\boldsymbol{A}$ is approximated as $\hat{\boldsymbol{A}} = \boldsymbol{Z}\boldsymbol{\Lambda}^{-1}\boldsymbol{Z}^T$, where $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{Z}^T\boldsymbol{1})$, and the matrix $\boldsymbol{Z} \in \mathbb{R}^{n \times m}$ is highly sparse, each column only having s values different from zero, which correspond to similarity values of the s nearest anchors. Because of this sparsity, the solution can be obtained by an eigenvalue decomposition of a much smaller $m \times m$ matrix, instead of $n \times n$ of matrix $\boldsymbol{A}$.

For further details, the reader should refer to [17].

4 Supervised Hashing Methods

In a supervised scenario, feature vectors are provided with a set of labels that describe the content of a training image. Usually the set of labels provided with the image is represented as a vector of the size of the labels dictionary, having 1 in the position of the labels that contains and 0 in the ones that it does not. There are not as much supervised methods in the literature as unsupervised. Among them, ITQ method using CCA stands out providing accurate results. As ITQ method is already discussed in 3.3, this section will focus completely in the proposed extension to AGH: Supervised Anchor Graph Hashing method.

4.1 Supervised Anchor Graph Hashing (SAGH)

The aim of Anchor Graph Hashing is to preserve the original topology by embedding near images to near hashing codes. The results showed in [17] and the reduced computational complexity make this technique a very interesting hashing method for large-scale scenarios. As mentioned above, the main assumption of AGH is that close-by images share labels. However, we can assume that images far in the original space could also share labels and thus being very close in the semantic space. Taking into account that nowadays the images are represented using a very low-level representation, mainly based on bag-of-visual-words, this second assumption is reasonable and motivates the supervised scenario. We propose an extension to AGH that considers side-information provided by the label vectors $\mathbf{t}$, when such information is available.

Note that the hashing function of AGH depends on the similarity between the input sample and the m anchor vectors, and since the label information is not available for the test samples, the label information cannot be introduced into the similarity matrix $\boldsymbol{A}$ as can be done for other methods. Therefore the label information has to be included in an indirect way.

Our extension, that we call *Supervised AGH (SAGH)*, is based on an two-step repeated application of AGH that uses the label information in an indirect way and the definition of an equivalence relation $\sim$. The resulting procedure can be summarized as follows, first the training samples are embedded into a geometric binary code, then a *semantic* representation is derived from this geometric code and finally a new embedding is performed into the desired binary hash code.

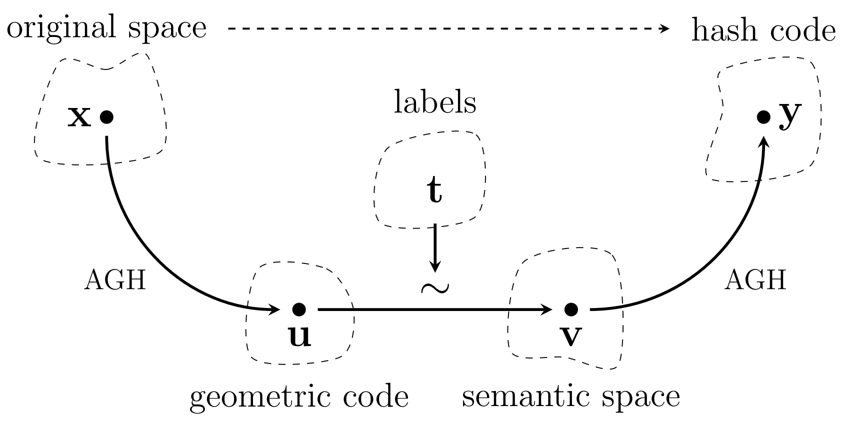


Figure 6: SAGH can be seen as a repeated application of AGH. A first embedding is obtained from the original space to a geometric code. Then, using an equivalence relation and the set of labels, a semantic representation is produced. Finally, a second embedding is obtained from this intermediate semantic space to the final binary hash code.

4.1.1 The proposed SAGH approach

In the proposed Supervised AGH, the intermediate semantic representation of the samples allows that semantically similar samples are coded with similar binary codes despite of being far in the original representation space. From this perspective, SAGH not only can have a better performance because of using the label information, it can also potentially encode the data with shorter binary codes.

The SAGH is performed by first applying the standard AGH to the training set providing an initial hash code of p bits $\mathcal{U} = \{\mathbf{u}_1, \dots, \mathbf{u}_n\} \subset \{-1, 1\}^p$, usually $p > q$ in order to produce a sparser distribution of the data. We will refer to this first hash code as a the *geometric* code. As mentioned above, the goal of this first hashing is to produce a semantic embedding of the training data. To this end, we define the equivalence relation $\sim$ in the set $\mathcal{X}$. Two samples are equivalent under this relation if these samples have the same geometric code:

$$\mathbf{x}_i \sim \mathbf{x}_j \Longleftrightarrow \mathbf{u}_i = \mathbf{u}_j \tag{5}$$

The equivalence class of a particular sample $\mathbf{x} \in \mathcal{X}$ is then defined as:

$$[\mathbf{x}] = \{\mathbf{x}' \in \mathcal{X} \mid \mathbf{u}_\mathbf{x} = \mathbf{u}_{\mathbf{x}'}\}$$

With this definition we propose a semantic representation of a particular training sample $\mathbf{x}_i$ as:

$$\mathbf{v}_i = \frac{1}{\mid [\mathbf{x}_i] \mid} \sum_{\mathbf{x} \in [\mathbf{x}_i]} \mathbf{t}_{\mathbf{x}} \tag{6}$$

where $\mid [\mathbf{x}_i] \mid$ is the number of elements in the equivalence class and $\mathbf{t}_{\mathbf{x}}$ is the label vector associated to the sample $\mathbf{x}$. In fact, with this definition all the samples inside an equivalence class share the same semantic representation. Thus, each equivalence class has an associated semantic representation that we denote by $\mathbf{v}_{[\mathbf{x}]}$.

Alternative equivalence relations could be defined in order to group samples depending on different strategies. For instance, several geometric codes could be obtained from the application of AGH with different parameters, e.g. number of nearest anchors s, anchor selection, etc., that yield different geometric codes for the same sample and allowing to define better equivalence relations.

Independently of the equivalence relation definition, equation (6) maps geometric codes $\mathbf{u} \in \{-1, 1\}^p$ into the semantic representations $\mathbf{v} \in \mathbb{R}^l$. As a result we have a representation in a semantic space $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathbb{R}^l$. This set of semantic representations for the n training samples is considered as a new input to a second AGH that produces an embedding into the final desired binary representation $\mathbf{y} \in \{-1, 1\}^q$ with q bits. This second AGH uses as input the *different* intermediate semantic codes $\mathbf{v}$ generated by the proposed approach. In principle the number of possible different semantic codes can be $\min(n, 2^p)$, but it is much more less in the practical situation. Figure **??** illustrates the SAGH mechanism.

Using this two-step hashing, points that were far in the original space but with similar semantic information should have similar intermediate semantic codes and thus will be mapped to nearby codes in the definitive binary space.

4.1.2 Hashing query images

The process to obtain a hash code for a query image follows a similar procedure. For a query image $\hat{\mathbf{x}}$ a geometric code $\hat{\mathbf{u}}$ is produced using the first AGH mapping. This geometric code could be the same (i.e. having hamming distance zero) than some geometric code $\mathbf{u}_i$ seen in the training step, thus $\hat{\mathbf{x}} \sim \mathbf{x}_i$ and then we will assign the same intermediate semantic code, $\hat{\mathbf{v}} = \mathbf{v}_i$. But the geometric code $\hat{\mathbf{u}}$ could also be *empty* in the training step, and then there is no semantic code to assign to it. In this case we propose the following procedure. First we have to find the radius R of the minimum hamming ball with non-empty geometric code $\mathbf{u}$ around $\hat{\mathbf{u}}$:

$$R = \min_r \{1, \ldots, p\} \quad s.t. \; \exists \; \mathbf{x} \in \mathcal{X} : \; d(\mathbf{u}_{\mathbf{x}}, \hat{\mathbf{u}}) = r \tag{7}$$

where $d(\cdot,\cdot)$ is the hamming distance. This radius defines the set $\mathcal{B}_R$ of the different equivalence classes inside this hamming distance. We propose to obtain the semantic embedding of the query point as an average of all the semantic representations associated to the equivalence classes inside $\mathcal{B}_R$:

$$\hat{\mathbf{v}} = \frac{1}{\mid \mathcal{B}_R \mid} \sum_{[\mathbf{x}] \in \mathcal{B}_R} \mathbf{v}_{[\mathbf{x}]} \tag{8}$$

Finally, the semantic code $\hat{\mathbf{v}}$ will be embedded using again AGH to the definitive hash code $\hat{\mathbf{y}}$.

It is important to note that the hashing obtained by SAGH will be affected mainly by the equivalence relation definition (5), the procedure to obtain a semantic representation associated to each equivalence class (6) and the semantic embedding for those query images that fall into empty geometric codes (8).

In fact, experimentation with SAGH 6 showed that the performance of the method decreased as more samples would fall into empty geometric codes. In order to decrease the number of samples obtaining unknown binary codes, we modified SAGH approach (4.2) defining a different equivalence relation.

4.2 Supervised Multiple Random Anchor Graph Hashing (SMRAGH)

When AGH is first run during training, a representation in geometric space is obtained for all the different samples in the training set. Training samples are represented as points in this new geometric space. Taking into account that close samples in appearance should generate similar hashing codes, we notice that if we represent all training samples in this geometric space $\{0,1\}^q$ we could find that the training samples occupy the space in such way that samples fall closer if they have similar contents, and there are also some areas in geometric space that remain deserted, because training samples haven't generated geometric codes laying in these areas.

Later on, during test, a geometric representation for the test feature vectors is obtained. If this point in geometric space lays in the same place as one of the training samples did, then the test sample is assigned the same label as the training sample. On the other hand, if the test sample lays in a deserted area, as explained in 4.1.2, we assign the label of the closest sample, that is, the one included in the minimum radio hamming ball around the test sample.

During test, in the first step of geometric representation using AGH method, we notice that most of the test samples obtain a geometric code representation that lays on empty buckets, that is, generates geometric codes that have never seen during training. We wish to reduce the probability of test samples obtaining geometric codes unknown in the geometric training codes set. Then, we need to cover more area in this geometric space.

In order to reduce this probability of obtaining geometric codes falling in empty buckets we propose generating n geometric representations of the training samples that complement each other and thus reduce the number of geometric areas uncovered which are those in which an unknown geometric code would be obtained.

The proposal is based in generating different geometric representations that can be seen as different layers of SAGH method each one of them covering different areas of the geometric space.

We would need to run AGH method n times but using different anchors each time, in order to obtain different geometric codes for each representation. Therefore, k-nn is no longer suitable to obtain the anchors. We propose to use random samples obtained from the training set as anchors.

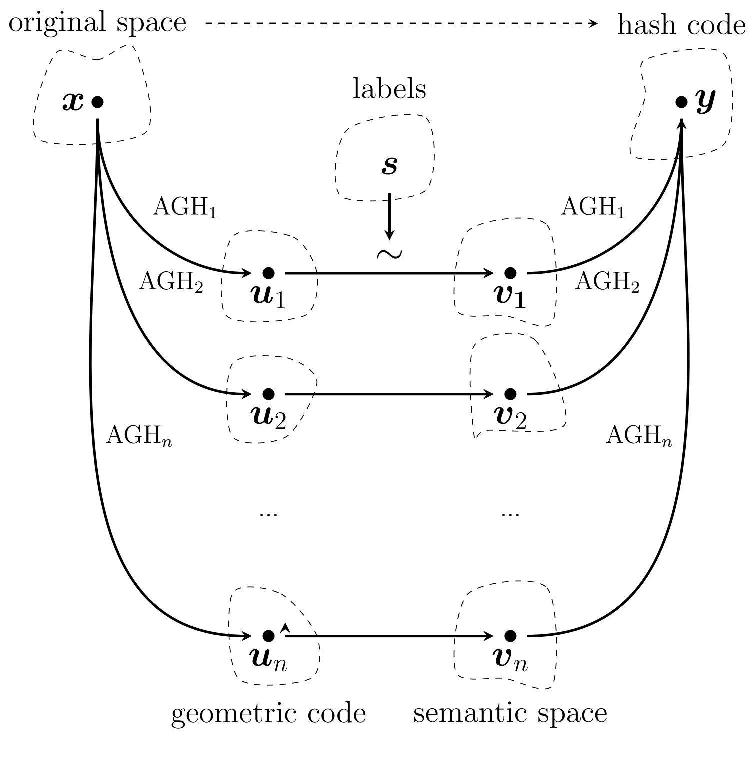


Figure 7: SMRAGH approach diagram.

In query time, the first geometric representation of the test sample is computed. In the case that a geometric code falling in an empty bucket is obtained, then the next geometric representation is computed. This is repeated until a geometric code falls in a non-empty bucket.

If during the process a sample $\hat{\boldsymbol{u}}$ falls in a non-empty bucket in an specific layer k, we proceed the same way as in SAGH, computing the intermediate representation by averaging the intermediate representation of the samples having the same geometric representation but only including the training samples of layer k. We will represent this as $\mathcal{B}_{R,k}$. In this case we already know that the minimum radio of the hamming ball containing the test sample is zero.

If none of the geometric representations in each one of the n layers obtains a geometric code that falls in a non-empty bucket, then we proceed similarly than in SAGH method (4.1.2) and compute the intermediate representation $\hat{\boldsymbol{v}}$ from the intermediate representation of the training sample/s inside the hamming ball with minimum radio around the test sample geometric codes among the n sets of training samples $\mathcal{B}_{R_{all}} = \mathcal{B}_{R_1} \cup \cdots \cup \mathcal{B}_{R_n}$.

The whole test process intermediate representation is expressed in equation 10. The equation should be called initially being k the number of AGH layers.

$$R_i = \min_r \{1, \ldots, p\} \quad s.t.\ \exists\ \mathbf{x} \in \mathcal{X} :\ d(\mathbf{u}_{i,\boldsymbol{x}}, \hat{\mathbf{u}}_i) = r, \qquad i = 1 \ldots n \tag{9}$$

$$\boldsymbol{v}(\hat{\boldsymbol{u}}, k) = \begin{cases} \frac{1}{|\mathcal{B}_{R_k}|} \sum_{[x] \in \mathcal{B}_{R_k}} \boldsymbol{v}_{\boldsymbol{x}} & \text{if } k < n \text{ and } d_H(\boldsymbol{u}_{\boldsymbol{x},k}, \hat{\boldsymbol{u}}) = 0 \\ \boldsymbol{v}(\hat{\boldsymbol{u}}, k-1) & \text{if } k < n \text{ and } d_H(\boldsymbol{u}_{\boldsymbol{x},k}, \hat{\boldsymbol{u}}) \neq 0 \\ \frac{1}{|\mathcal{B}_{R_{all}}|} \sum_{[x] \in \mathcal{B}_{R_{all}}} \boldsymbol{v}_{\boldsymbol{x}} & \text{if } k = 0 \end{cases} \tag{10}$$

5 Literature Experimentation

5.1 Measures

Precision: Ratio of relevant images to the total number of images retrieved in the query. Example: if 8 images are retrieved and only 4 belong to the category being searched we have 50% precision.

Recall: Ratio of relevant images retrieved in a query to the total number of images in the database. For example if only the top 8 matches are retrieved from a database that contains 80 images, recall is 10 percent.

Precision vs. Recall curves Precision is important for retrieval tasks and Recall for recognition tasks like retrieving the most similar points to a query. This curves are used to represent and compare the performance of a system. The curves show, in general, how precision decreases as larger fractions of the image database are retrieved. An ideal precision versus recall curve would have 100% precision for all values of recall: that would mean that all the relevant images are retrieved before any irrelevant ones.

Precision @ top returned images Shows the average precision for the first k retrieved images. This measures the hash ranking performance of the method.

Precision @ hamming distance Shows the average precision for an specific hamming radius hr. This measures the hash lookup performance of the method. Many articles in the literature provide Precision @ $hr = 2$ figures because this restriction provides very fast algorithms for retrieval.

5.2 Datasets used in literature

Below are listed the different datasets have been used to perform the experimentation of the different methodologies revised in the literature. Among them we chose CIFAR and NUS-WIDE to do the experimentation (section 6).

- LabelMe (GIST)
- MNIST (pixels)
- PhotoTourism (SIFT)
- Peekaboom (GIST)

- Nursery (8D attributes)
- 10D Uniform
- 80 million TinyImages
- CIFAR-10 and CIFAR-100
- CIFAR-11 (not published on the internet. Provided by Rob Fergus for [10])
- NUS-WIDE

6 Experiments

In order to assess the performance of the proposed SAGH technique, we have performed experiments using two datasets widely used in the literature. The first dataset is a version of the CIFAR[1] dataset [14], which consists 64,185 images selected from the Tiny Images dataset [23]. The original Tiny Images are 32×32 pixels, although they have been represented with grayscale GIST descriptors [19] computed at 8 orientations and 4 different scales, resulting in 320-dimensional feature vectors. These images have been manually grouped into 11 ground-truth classes (airplane, automobile, bird, boat, cat, deer, dog, frog, horse, ship and truck), thus we shall refer to this version of the dataset as CIFAR-11, and it is the same dataset that was used in [10].

The second dataset is the NUS-WIDE[2] dataset [5], which contains 269,648 images labelled using 81 concept tags. Each image is represented by an l_2 normalized 1024-dimensional sparse-coding feature vector [26] in the same way as it is done in [17]. Unlike CIFAR, images in NUS-WIDE are multi-labelled, i.e. each image can have multiple labels.

For comparison, we also ran the experiments with other hashing techniques found in the literature for which there was code freely available. Namely we have compared with the original Anchor Graph Hashing (AGH) [17], Locality Sensitive Hashing (LSH) [11], Spectral Hashing (SH)[28] and Iterative Quantization with Canonical Correlation Analysis (ITQ-CCA) [10]. Unfortunately, the only supervised hashing method for which we had code available was ITQ-CCA. In the results there is also a comparison with the feature vectors in the original space using the Euclidean distance, i.e. not doing hashing, and this is referred to as L2.

6.1 CIFAR-11

To estimate the performance of the different methods for the CIFAR-11 dataset, we have employed a 5-time repeated hold-out procedure. In each of the five rounds, 3,000 images were randomly selected for the test set and the remaining was left as the training set. The final results are the average over the five partitions.

For each of the methods being compared, the corresponding parameters were varied and only the best results are presented in each case. The number of anchors was kept fixed at 300, both for AGH and SAGH which was the value used in [17] for NUS-WIDE. The number of nearest anchors for AGH

[1]`http://www.cs.toronto.edu/~kriz/cifar.html`
[2]`http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm`

was varied for $s = 2, 3, 5$. For SAGH the number of nearest anchors to consider for the first geometric AGH also varied for $s_g = 2, 3, 5$ and for simplicity the second semantic AGH nearest anchors parameter s_s had the same value than s_g. The number of bits for the intermediate geometric hashing was varied for 10, 16, 32, 64, 128, 256 and 512 bits, although always higher or equal than the number of final semantic code size.

The results are presented in Figure 8. The performance is measured using the class labels as ground truth. In the figure one of the graphs presents the average precision for the first 500 retrieved images when varying the number of bits, this measures the hash ranking performance. For retrieved images having exactly the same hamming distance, a random reordering was applied. The other graph in the figure shows the average precision for a hamming radius of 2 and this measures the hash lookup performance.

As was expected, the two supervised methods, SAGH and ITQ-CCA, perform much better than all of the other unsupervised methods. This is quite understandable since the labels of CIFAR-11 are manually selected and not noisy, thus there is much to gain by using this additional available information. The performance of SAGH is better than ITQ-CCA. Note that in this case, because there are only 11 classes, ITQ-CCA is limited to a maximum of 10 bits, which is a severe limitation. As can be observed, the performance of the proposed SAGH is better than its unsupervised counterpart AGH. This confirms that the proposal effectively is capable of taking advantage the additional information to achieve a better performance. In these results the same behavior as in [17] is observed both for AGH and SAGH. The precision at a hamming radius of 2 does not decrease for large code sizes. Although the performance is not better than for fewer bits.

Figure 9 shows the top 10 images retrieved using the different hashing techniques. AGH shows a better performance for this particular query image than the other unsupervised techniques. For the supervised techniques, ITQ-CCA and SAGH show a very different performance. In this case ITQ-CCA code was formed by only 10 bits so it is not a truly fair comparison. Anyway, it is important to note that visually the results of SAGH are clearly more diverse than the other techniques but semantically all the images retrieved by SAGH are correct. This agrees with the behavior we were expecting of the proposed SAGH.

6.2 NUS-WIDE

For the NUS-WIDE dataset, the evaluation protocol is exactly the same as the one used in [17]. For test set, the 21 most frequent tags are considered, and for each of these tags, 100 images were randomly selected. Thus in total

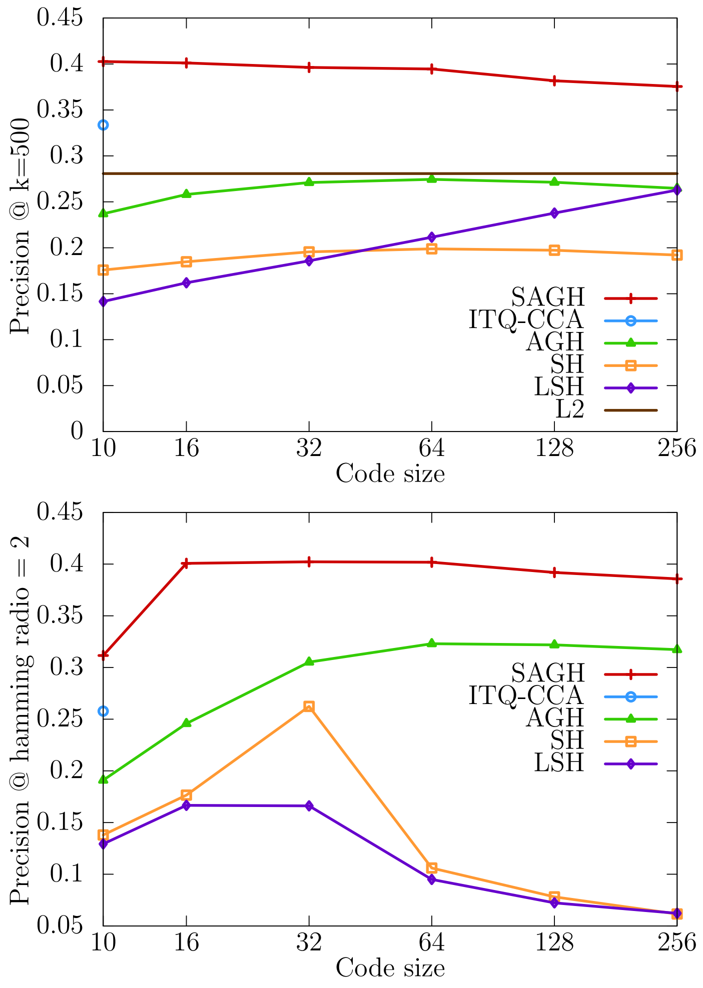


Figure 8: CIFAR-11 dataset. On top, average precision of the top-500 ranked images for each method varying the hash code size. Below, average precision for a hamming radius of 2 for each method varying the hash code size.

2,100 images are used for the test set, and the remaining are used as training.

Analogously to the CIFAR experiments, for each of the methods the corresponding parameters were varied and the best result is presented. The parameters tried for AGH and SAGH were the same than for the CIFAR-11 experiments.

The results are presented in Figure 10 using the same performance measures as for CIFAR, although in this case using precision for the first 5,000 retrieved samples since this was the value used in [17]. As ground truth labels, images are considered being semantically the same if they have at least one common tag. For more details on the protocol please refer to [17].

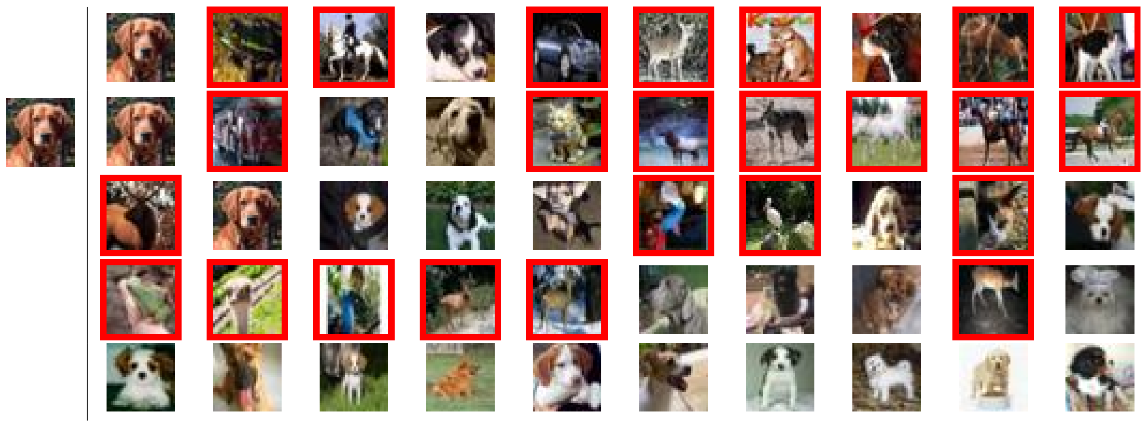


Figure 9: Query image and the 10-nearest images retrieved using different hashing techniques. From top to down: LSH, SH, AGH, ITQ-CCA (10 bits) and SAGH.

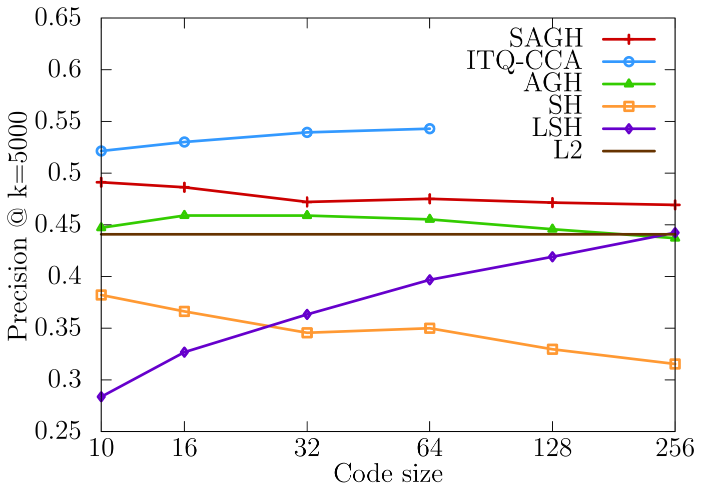
The results are somewhat similar to the ones for CIFAR. The supervised methods perform better than the unsupervised ones, and again SAGH is better than the original AGH. However, the difference between SAGH and AGH is much smaller and in this case ITQ-CCA has a better performance than SAGH.

In order to clarify this behavior we should take into account the proposed procedure for generating the intermediate semantic code in SAGH. Table 1 shows first the percentage of test samples for which the geometric codes fell into an non-empty geometric code, i.e. the radius of the minimum hamming ball is $R = 0$, and also shows the percentage of test samples for which the radius is very large $R > 3$. When $R > 0$, the intermediate semantic representation has to be obtained by means of averaging the semantic representations of the equivalence classes in $\mathcal{B}_R$.

Table 1: Percentage of test samples with radius 0 and higher than 3 w.r.t. the number of geometric bits p for the NUS-WIDE dataset (AGH).

p	$R = 0$	$R > 3$
32	80	0.2
64	58	20
128	44	32
256	20	43

When the value of R is high, it is highly probable that this averaging is done for very different semantic regions. This problem becomes important



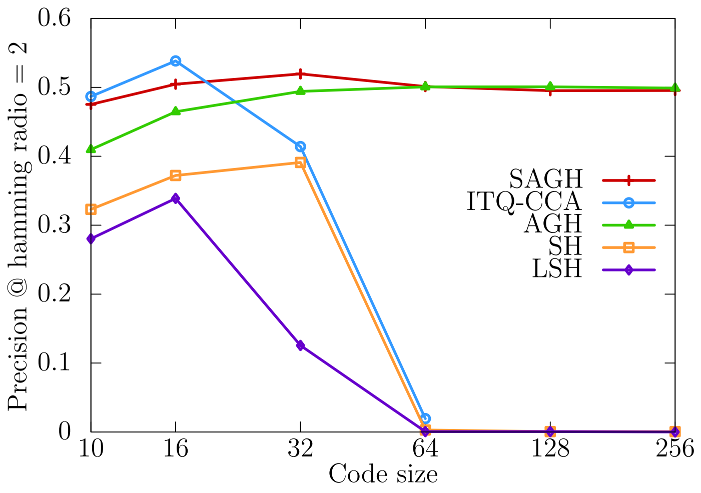


Figure 10: NUS-WIDE dataset. On top, average precision of the top-5,000 ranked images for each method varying the hash code size. Below, average precision for a hamming radius of 2 for each method varying hash code size.

when the number of geometric bits p increases, as can be observed in table 1. The results obtained for NUS-WIDE are motivating us to look for different approaches for obtaining a better semantic representations in such situations.

On the other hand, in the same way as the AGH behavior, SAGH maintains a good hash lookup precision even though code size increases, unlike the rest of the methods. SAGH obtains a better average precision than AGH for a hamming radius of 2 for small code sizes.

The percentage of test samples obtaining a training hash code has incremented as we expected. By generating codes with other AGH more geometric space is covered. This increments the probability of obtaining lower ham-

Table 2: Percentage of test samples with radius 0 and higher than 3 w.r.t. the number of geometric bits p for the NUS-WIDE dataset (SMRAGH) using number of randoms = 3.

p	$R = 0$	$R > 3$
32	91.43	0
64	68.81	8
128	59.81	21.14
256	40.05	28.95

ming distances between test and train codes. As we can see in figure 11, first we can notice that the behaviour of the technique slightly varies when using random anchors, but then as expected the minimum radio of the hamming ball decreases as the number of layers in the Multiple Random AGH increases. This finally produces an increment in the precision of the method even beating the results obtained with ITQ method (Figure 12).

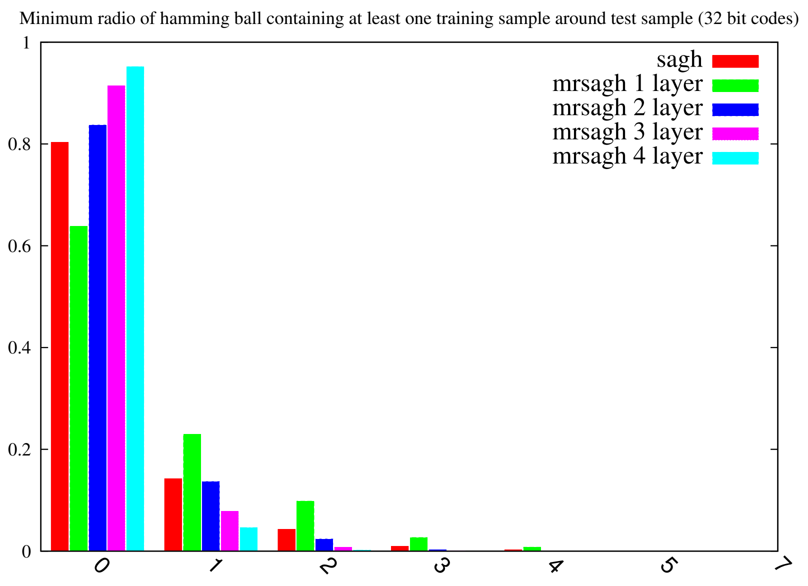


Figure 11: NUS-WIDE dataset: Minimum radio of hamming ball containing at least a training sample around the test sample geometric code. Methods SAGH, and SMRAGH with 1 to 5 layers

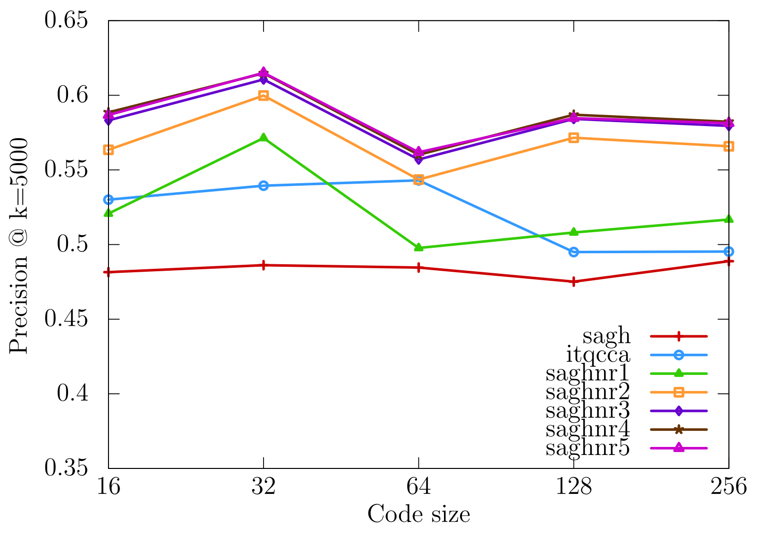


Figure 12: NUS-WIDE dataset: Hamming ranking precision of top-5000 ranked neighbours for each method varying the size of hash code, including random sagh method with 1 to 4 layers

7 Conclusions

In this master thesis we propose an extension to the Anchor Graph Hashing technique which is capable of taking advantage of supervised/label information. This extension is based on representing the samples in an intermediate semantic space that comes from the definition of an equivalence relation in an intermediate geometric code. The results show that our approach is a very effective way to incorporate such supervised information to the standard AGH. The standard AGH is clearly outperformed by our SAGH in the CIFAR dataset where the supervised information can be considered very clean. Moreover, SAGH is clearly the best technique on this dataset compared to the state-of-the-art ITQ-CCA. On the other hand, slight improvements are obtained using our approach in the NUS-WIDE dataset with respect to the AGH. In this multi-label dataset the label information is known to have an important presence of noise. In order to improve the results of the method, a new approach is defined to obtain a semantic representation for those test samples that fall far from a non-empty geometric code. This approach improved the precision in the results, making this technique achieve the best results for this dataset. There are other possible flaws of the proposed approach in this noisy scenario. Thus, future work will be focused on defining better equivalence relations should be derived which give the equivalence classes a more discriminative power under the semantic point of view, and furthermore being robust to the presence of noisy labels.

References

[1] *Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences*, 2007.

[2] Alexandr Andoni and Piotr Indyk. Near-Optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, Washington, DC, USA, 2006. IEEE Computer Society.

[3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[5] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., July 8-10, 2009.

[6] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[7] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In *In Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*. MIT Press, 2006.

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[9] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *The VLDB Journal*, pages 518–529, 1999.

[10] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[11] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[12] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[13] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms, 3rd Edition.* Addison-Wesley, 1997.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009.

[15] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1042–1050. 2009.

[16] Michael S. Lew, Nicu Sebe, and John P. Eakins. Challenges of image and video retrieval. In *Proceedings of the International Conference on Image and Video Retrieval*, CIVR '02, pages 1–6, London, UK, UK, 2002. Springer-Verlag.

[17] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1–8, New York, NY, USA, June 2011. ACM.

[18] Mohammad Norouzi and David Fleet. Minimal loss hashing for compact binary codes. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 353–360, New York, NY, USA, June 2011. ACM.

[19] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision*, 42:145–175, May 2001.

[20] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50:969–978, July 2009.

[21] Nicu Sebe, Michael S. Lew, Xiang Zhou, Thomas S. Huang, and Erwin M. Bakker. The state of the art in image and video retrieval. In

Proceedings of the 2nd international conference on Image and video retrieval, CIVR'03, pages 1–8, Berlin, Heidelberg, 2003. Springer-Verlag.

[22] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 750–, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30:1958–1970, November 2008.

[24] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[25] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5-6):544–557, 2009.

[26] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas S. Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *CVPR*, pages 3360–3367. IEEE, 2010.

[27] Peter Wegner. A technique for counting ones in a binary computer. *Commun. ACM*, 3:322–, May 1960.

[28] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS'08*, pages 1753–1760, 2008.