REAL-TIME LARGE-SCALE CONTENT-BASED IMAGE RETRIEVAL

by

ABDELRAHMAN KAMEL SIDDEK ABDELHAMED

B.Sc. Computer Science, Assiut University, 2009

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE (COMPUTER SCIENCE)

to

Department of Computer Science

Faculty of Computers and Information

ASSIUT UNIVERSITY

Supervised by

Prof.Dr. Youssef Bassyouni Mahdy

Dr. Khaled Fathy Hussain

Examined by

Prof.Dr. Usama Sayed Mohammed

Prof.Dr. Marghany Hassan Mohamed

Prof.Dr. Youssef Bassyouni Mahdy

Dr. Khaled Fathy Hussain

Assiut - Egypt

2014

THESIS APPROVAL

- Name: Abdelrahman Kamel Siddek Abdelhamed
- Thesis Title: REAL-TIME LARGE-SCALE CONTENT-BASED IMAGE RETRIEVAL
- Degree: Master of Science (MSc), Computers and Information (Computer Science)
- Supervisors:
    1. Prof. Dr. Youssef Bassyouni Mahdy
    2. Dr. Khaled Fathy Hussain
- Examination Committee:
    - External Referees:
        1. Prof. Dr. Usama Sayed Mohammed
        2. Prof. Dr. Marghany Hassan Mohamed
    - Internal Referees:
        1. Prof. Dr. Youssef Bassyouni Mahdy
        2. Dr. Khaled Fathy Hussain

Prof. Dr. Adel Abu El-Majed Sewisy

Vice Dean for Higher Studies and Research


بسم الله الرحمن الرحيم
إنه من سليمان وإنه

Abstract

The challenge of large-scale Content-Based Image Retrieval (CBIR) has been recently addressed by many promising approaches. In this thesis, a new approach that jointly optimizes the search precision and time for large-scale CBIR is presented. This is achieved by using binary local image descriptors, such as Binary Robust Independent Elementary Features (BRIEF) and Binary Robust Invariant Scalable Key-points (BRISK), along with binary hashing methods, such as Locality Sensitive Hashing (LSH) and Spherical Hashing (SH).

The proposed approach, named *Multi-Bin Search*, improves the retrieval precision of binary hashing methods. This improvement is done through computing, storing and indexing the nearest neighbor bins for each bin generated from a binary hashing method. Then, the search process does not only search the targeted bin, but also it searches the nearest neighbor bins.

In order to efficiently search inside the targeted bins, a fast exhaustive-search equivalent algorithm has been used. This algorithm is inspired by Norm Ordered Matching (NOM) which has been recently proved to yield the same or too close results to exhaustive-search results with a significant speedup. Also, a results reranking step that increases the retrieval precision is introduced, but with a slight increase in search time.

Experimental evaluations show that the proposed approach greatly improves the retrieval precision of recent binary hashing methods. Also, comparisons with some state-of-art methods have been carried out in order to further evaluate the effectiveness of the proposed methods. The compared methods are not depending on either binary hashing or binary descriptors. In addition, in order to experiment the proposed approach in a real-world applications, an image search Web application was built based on the implementation of proposed methods. The operation of the Web application demonstrated the effectiveness of the proposed approach for real-world applications.

Acknowledgements

Praise be to Allah, the Most Gracious, the Most Merciful. Prayer and peace be upon the Prophet Muhammad.

I would like to express my deepest gratitude to *Prof. Youssef Basyouni Mahdy*, for his patience, close guidance, and continual support.

I would like to thank *Dr. Khaled Fathy Hussain* for his insights and valuable remarks all through this work.

I would like to thank my colleagues at the Faculty of Computers and Information, for their help and valuable discussions.

I would like to thank my faculty for accepting me as a researcher and teaching assistant, and for providing me with resources to finish this work.

Finally, I would like to express my gratitude to my family, for their patience and support, words are useless in expressing my gratitude to my parents, to whom I simply owe everything.

Abdelrahman Kamel Siddek

2014

To my parents

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

List of Abbreviations

AGAST	Adaptive and Generic Accelerated Segment Test
ANN	Approximate Nearest Neighbor
BFROST	Binary Features from Robust Orientation Segment Tests
BoF	Bag of Features
BoVW	Bag of Visual Words
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Key-points
CBIR	Content-Based Image Retrieval
FAST	Features from Accelerated Segment Test
FREAK	Fast REtinA Key-points
LRP	Low Resolution Pruning
LSH	Locality-Sensitive Hashing
MAP	Mean Average Precision
MBEES	Multi-Bin Exhaustive-Equivalent Search
NOM	Norm Ordered Matching
ORB	Oriented and Rotated BRIEF
PCA	Principal Component Analysis

PDE	Partial Distortion Elimination
PK	Projection Kernel
QBIC	Query By Image Content
SBEES	Single-Bin Exhaustive-Equivalent Search
SH	Spherical Hashing
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
VLAD	Vector of Locally Aggregated Descriptors
WWW	World Wide Web

CHAPTER ONE

INTRODUCTION

This chapter reveals a background about the field of image retrieval and its traditional and recent applications. Then the objectives of the presented work are introduced, followed by a quick overview of the rest of this thesis.

1.1 Image Retrieval

Image retrieval is the process of searching and retrieving digital images from a database of digital images. Image retrieval can be divided into two major categories, image meta search and Content-Based Image Retrieval (CBIR) [1]. In image meta search, a keyword, caption or description text is added as a meta data for each image in the database so that the retrieval is performed over these meta data, as shown in Fig. 1.1. A keyword or search phrase is supplied as a query, then images with relevant meta data to the query are retrieved as results. These meta data are often manually attached to images, this is considered a limitation to such image retrieval methods.

On the other hand, Content-Based Image Retrieval does not include usage of textual meta data in the retrieval process, as shown in Fig. 1.2 compared to Fig. 1.1. It retrieves images based on visual similarities in their contents to a user-supplied query image. Such image contents can be colors, textures, shapes or any other information that can be derived from the image itself [2]. So the image content has to be analysed and described in a proper

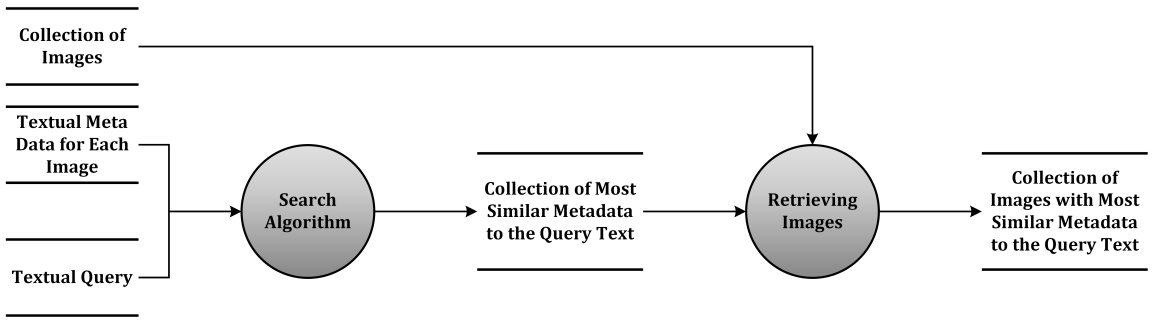

Fig. 1.1: General scheme of Image Meta Search.

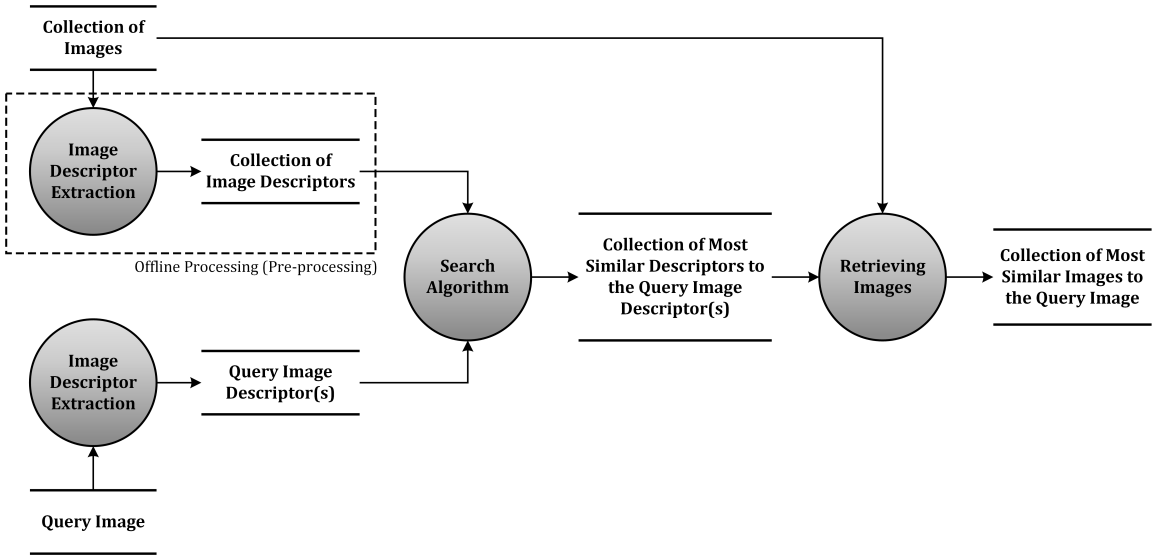

Fig. 1.2: General scheme of Content-Based Image Retrieval.

representation for the search process. The techniques, tools and algorithms that are used to describe image content originate from fields such as statistics, signal processing, computer vision, and pattern recognition. The next section provides a quick overview on the methods of image content representation.

1.2 Image Representation

Traditional image representation methods use the color, texture, or shape of the image to describe the content of an image. For example, in the *Query By Image Content* (QBIC) system [2], a histogram of image colors is used to represent images. The color histogram is simply the frequency of occurrence of each color value in the image. Examples of color

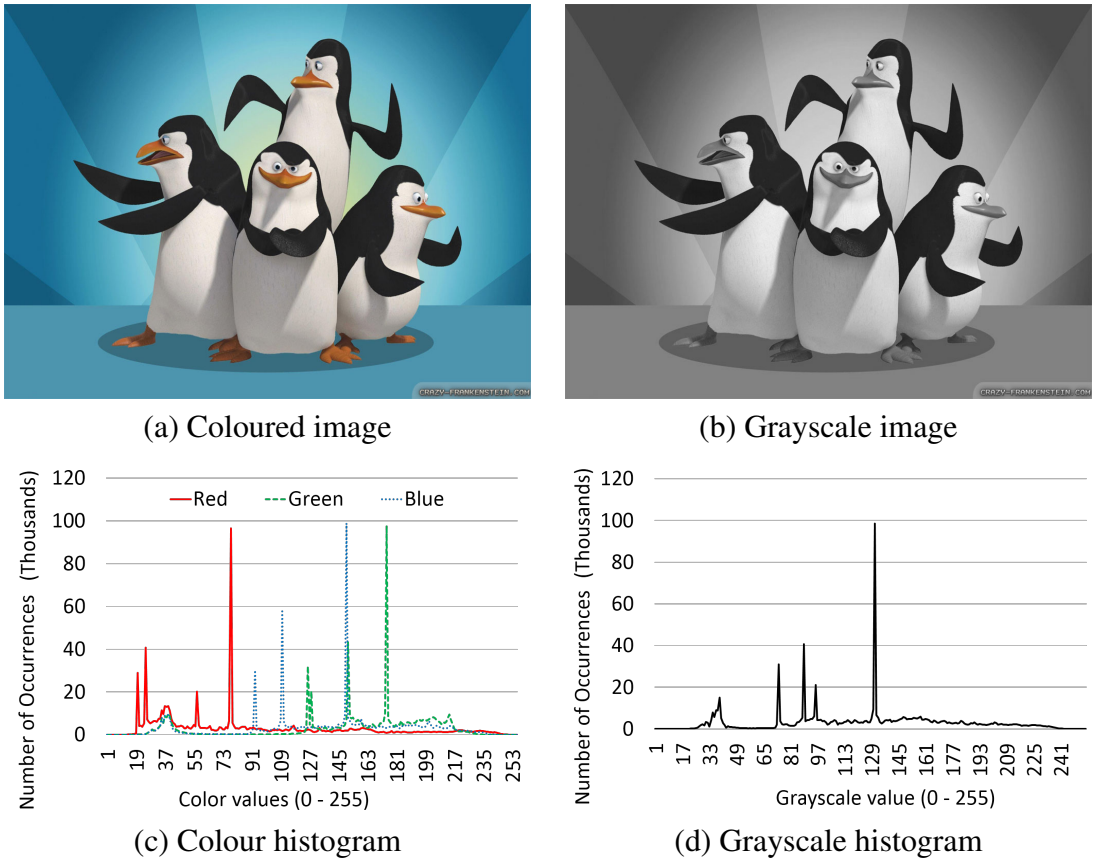

Fig. 1.3: Examples of image histograms. Part (c) shows the colour histogram of the image in part (a). Part (d) shows the grayscale histogram of the image in part (b).

and grayscale histograms are shown in Fig. 1.3. It is obvious that color histograms do not contain any information about image structure, they only describe the color distribution in the image. Recent image representation methods use more advanced techniques such as image keypoint descriptors, or interest point descriptors.

Image *keypoints* are points in the image that have a well-defined positions in image space and they are stable under local and global variations in the image scaling, rotation, illumination, etc. Such interest points can be reliably computed with high degree of reproducibility. These image keypoints are used to generate a numerical representation based on the pixel information in the image region around each keypoint. The resulting numerical representations are the keypoint *descriptor vectors*, or *feature vectors* [3]. Image descrip-

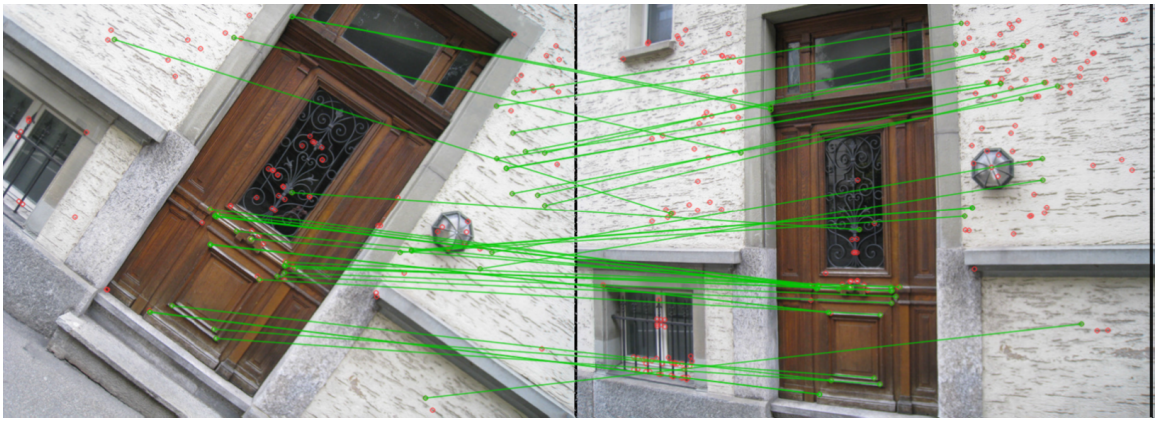

Fig. 1.4: An example of keypoints generated by BRISK from two different views of an image. The keypoints regenerated with a high degree of similarity in their descriptors are connected with green lines. This image is taken from [6].

tors can be local, i.e. one descriptor for each keypoint in the image, or global, i.e. one descriptor for the whole image.

Among the most famous local image detectors and descriptors are Scale Invariant Feature Transform (SIFT) [3] and Speeded Up Robust Features (SURF) [4]. Another example for global image descriptors is the GIST [5]. These descriptors have a very high discriminative power. Their drawback is the high dimensionality, where the size of a single descriptor normally ranges from 64 to 960 bytes, which requires much storage and processing compared to nowadays real-time and large-scale applications.

Other modern image representation methods involve *binary* keypoint descriptors such as Binary Robust Independent Elementary Features (BRIEF) [7], Binary Features from Robust Orientation Segment Tests (BFROST) [8], Binary Robust Invariant Scalable Keypoints (BRISK) [6], Oriented and Rotated BRIEF (ORB) [9], and Fast Retina Key-points (FREAK) [10]. These binary descriptors are smaller in size and faster in generation than previously mentioned traditional descriptors. Figure 1.4 shows an example of local image keypoints generated by BRISK from two different views of an image. The keypoints regenerated with a high degree of similarity in their descriptors are connected with green lines. Binary local image descriptors will be discussed in further detail later in this thesis.

1.3 Large-Scale Content-Based Image Retrieval

Large-scale image datasets are considered to contain thousands, millions, or more images. Examples of such datasets are the images on the World Wide Web (WWW), medical imaging datasets, video frames extracted from surveillance cameras, etc. Searching large-scale image datasets containing millions of images using brute-force matching, i.e., matching a query image with all images in the dataset, image by image, is not a practical task. This is true even if the dataset contains only a few hundreds of images. This is due to many reasons:

- The large number of images in the dataset.
- The *high dimensionality* of image descriptors which usually consist of tens or hundreds of dimensions per descriptor.
- The method used for *matching* pairs of images.
- The complexity of the *distance measure* used to match descriptors, which is a major factor in speeding up the retrieval process and increasing the retrieval precision.

To reduce the impact of the reasons mentioned above, approximations to the image retrieval algorithms have to be made. Many of these approximations are included under the general problem of Approximate Nearest Neighbor (ANN) search. One of the most notable contributions to the approximation of the image retrieval process is the Bags of Visual Words (BoVW) [11], or the Bags of Features (BoF), as called in some literature. This method aimed to represent each image with a single vector of unified dimensionality in order to avoid matching sets of descriptors from every image. This method was proved to gain a high rate of retrieval precision but with a drawback of ignoring spatial image information, as addressed in [12].

There are three reasons for the success of the BoVW representation. First, this representation benefits from powerful local descriptors, such as the SIFT descriptor [3]. Second, the BoVW vector representation can be compared with standard distances. Third, the BoVW

vector can be high dimensional, in which case the vectors are sparse and inverted lists can be used to implement efficient search. However, two factors limit the number of images that can be indexed in practice: the search time itself, which becomes prohibitive when considering large number of images, and the memory usage per image.

Another method towards scalable image retrieval is the Vocabulary Tree [13]. In the Vocabulary Tree method, all local image descriptors are hierarchically quantized using the k-means clustering algorithm. Each cluster represents a set of approximately matched descriptors. Then all the resulting clusters are indexed into a tree structure, which is called the Vocabulary Tree. This index also keeps track of the images to which each descriptor belongs. On a query process, the tree is searched using a breadth-first-search order, matching the query descriptor with all cluster centres at each level of the tree, until the nearest leaf cluster is found. Once the nearest leaf cluster to the query descriptor is found, all images having descriptors residing in this cluster are considered candidate results. These steps are repeated for each descriptor in the query image. Then all candidate images are sorted in descending order based on the number of descriptors found in all targeted leaf clusters.

Another state-of-art approach [14] jointly optimizes dimensionality reduction and indexing in order to obtain a precise vector comparison as well as a compact image representation. This approach uses the Vector of Locally Aggregated Descriptors (VLAD) [15] to aggregate local image descriptors into a compact image representation. Then, the dimensionality of the aggregated image descriptor is then reduced using Principal Component Analysis (PCA) [16], which is a useful tool for feature extraction and dimensionality reduction.

Focusing on the high dimensionality of image descriptors, many methods have been proposed to face this curse of dimensionality. These methods are based on Hashing algorithms, more specifically, Locality Sensitive Hashing (LSH). The basic idea is to hash the data points so that similar items are mapped to the same buckets or bins with high probability, the number of buckets or bins is much smaller than the universe of data points.

Such methods include LSH [17], Spectral Hashing [18], and Spherical Hashing [19]. The hashing methods are discussed in further detail later in this thesis.

1.4 Applications of CBIR

CBIR is the engine for many real-world, machine learning, and compute vision applications. Such applications include object recognition [13], finding partial image duplicates on the web [20], searching individual video frames [11], image classification [21], robot localization [22], and medical imaging [23].

Other applications include art gallery and museum management, architectural and engineering design, interior design, remote sensing and management of earth resources, geographic information systems, scientific database management, weather forecasting, fabric and fashion design, trademark and copyright database management, law enforcement and crime prevention, picture archiving and communication systems, and home entertainment [24].

One of the most interesting applications of CBIR is online image search engines. Such image search engines include Google Images [25] and TinEye Reverse Image Search [26].

1.5 The Objectives of The Thesis

The objectives of the presented work can be summarized as follows:

- Providing an overview on the field of Image Retrieval.
- Reviewing the recent state-of-the-art methods and algorithms in the area of Content-Based Image Retrieval.
- Introducing a new approach to Content-Based Image Retrieval that is suitable for nowadays large-scale and real-time applications. This approach is based on recent state-of-the-art algorithms and methodologies of the field.

1.6 The Organization of The Thesis

The rest of this thesis is organized as follows:

Chapter 2, Related Works, gives a detailed review on some recent state-of-the-art methods and algorithms that have been proposed in the area of CBIR. The chapter is focused on methods and algorithms that are used to build up the proposed approach of this work.

Chapter 3, Multi-Bin Search, provides a detailed presentation and explanation of the proposed approach of this work, which is called Multi-Bin Search, discussing its advantages and weakness points.

Chapter 4, Results, shows a detailed evaluations and comparisons of the proposed approach against other state-of-the-art approaches.

Chapter 5, Conclusions and Future Works concludes the work done through this thesis and presents the future works to be carried out based on this work.

CHAPTER TWO

RELATED WORKS

This chapter provides a quick review on recent research work related to the field of content-based image retrieval. It also provides a detailed review on the specific methods and algorithms on which this work is depending. Section 2.1 presents the recent work on *binary* local image descriptors, and discusses the Binary Robust Invariant Scalable Keypoints (BRISK) [6] in some detail. Section 2.2 presents the recent work on binary hashing methods, and provides more details on Locality-Sensitive Hashing (LSH) [17] and Spherical Hashing [19]. Section 2.3 presents the recent work on exhaustive-search equivalent algorithms and discusses Norm Ordered Matching (NOM) [27] in some detail.

2.1 Binary Local Image Descriptors

Decomposing an image into local keypoints of interest, or features, is a widely applied technique for representing digital images. Many computer applications, including those related to Content-Based Image Retrieval, rely on the presence of stable, representative features in the image. This is the work targeted by image keypoint detectors and descriptors.

The ideal keypoint detector aims to finds salient image regions such that they can be re-detected even if the image is modified or changed in terms of viewpoint, illumination, scale, etc. More generally, these salient image regions are robust to all possible image transformations. Similarly, the ideal keypoint descriptor captures and represents the most

important and distinctive information content enclosed in the detected salient regions, such that the same region or structure can be recognized if encountered. In addition to fulfilling these properties to achieve the desired quality of detected keypoints, the speed of detection and description needs to be optimized to fit within the time constraints of the task at hand.

In principle, state-of-the-art image keypoint detectors and descriptors target applications with either specific requirements in accuracy or speed of computation. Lowe's SIFT algorithm [3] is widely accepted as one of highest quality options currently available, this is due its promising distinctiveness and invariance to a variety of common image transformations. However, SIFT requires a high expense of computational cost. On the other end of the spectrum, a combination of the FAST [28] keypoint detector and the BRIEF [7] approach to image keypoint description presents a much more suitable alternative for real-time applications. However, despite the obvious advantage in computational speed, the latter approach suffers in terms of reliability and robustness. This is due to its minimal tolerance to image distortions and transformations, in particular to in-plane rotation and scale change.

The major difficulty in the process of extracting suitable features from an image lies in the balancing of two competing goals: high quality description and low computational power. Among recent approaches, the most relevant work targeting this problem is SURF [4] which has been demonstrated to achieve robustness and speed.

A recently introduced method for image keypoint detection and description is Binary Robust Invariant Scalable Keypoints (BRISK) [6], which is discussed in some detail in subsection 2.1.1. BRISK is the method used in this work for generating image keypoint descriptors. BRISK has been proven to achieve comparable quality to SURF with much less computation time for both generation and matching of keypoints.

2.1.1 Binary Robust Invariant Scalable Keypoints (BRISK)

Binary Robust Invariant Scalable Keypoints (BRISK) [6] is an approach to local image keypoint detection and description that has been proven to provide fast keypoint detection, description and matching with high precision. It is rotation invariant and scale invariant to a significant extent, achieving performance comparable to the state-of-the-art, with respect to SURF and SIFT, while dramatically reducing computational cost. Also, the modularity of the BRISK method allows the use of the BRISK detector in combination with any other keypoint descriptor and vice versa, BRISK descriptor can be used in combination with any other keypoint detector. This modularity allows optimization for the desired performance and the task at hand.

Scale-Space Keypoint Detection

BRISK detection methodology is inspired by Adaptive and Generic Accelerated Segment Test(AGAST) [29], an algorithm for detecting regions of interest in the image. AGAST is essentially an extension for accelerated performance of the now popular FAST (Features from Accelerated Segment Test) [28], which has been proven to be a very efficient basis for feature extraction.

With the aim of achieving invariance to scale, which is crucial for high-quality keypoints, BRISK searches for maxima not only in the image plane, but also in scale-space using the FAST score s as a measure for saliency. Despite discretizing the scale axis at coarser intervals than in alternative high-performance detectors, the BRISK detector estimates the true scale of each keypoint in the continuous scale-space.

In BRISK, the scale-space pyramid layers consist of n octaves c_i and n intra-octaves d_i, for $i = 0, 1, ..., n-1$ and typically $n = 4$. The octaves are formed by progressively half-sampling the original image (corresponding to c_0). Each intra-octave d_i is located in-between layers c_i and c_{i+1}, as illustrated in Fig. 2.1.

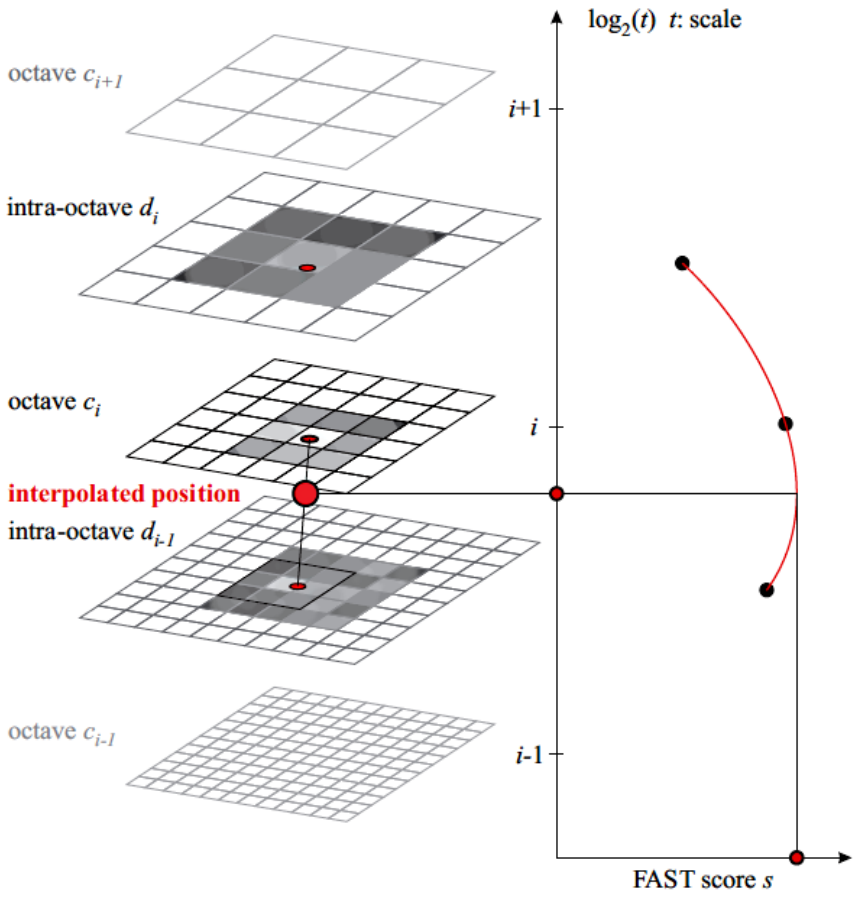

Fig. 2.1: BRISK scale-space interest point detection [6]

The first intra-octave d_0 is obtained by down-sampling the original image c_0 by a factor of 1.5, while the rest of the intra-octave layers are derived by successive half-sampling. Therefore, if t denotes scale then $t(c_i) = 2^i$ and $t(d_i) = 2^i \times 1.5$. It is important to note here that both FAST and AGAST provide different alternatives of mask shapes for keypoint detection. BRISK uses the 9-16 mask. This mask requires at least 9 consecutive pixels in the 16- pixel circle to either be sufficiently brighter or darker than the central pixel for the FAST criterion to be fulfilled, as shown in Fig. 2.2.

Keypoint Description

Given a set of keypoints (consisting of sub-pixel refined image locations and associated floating-point scale values), the BRISK descriptor is composed as a binary string by con-

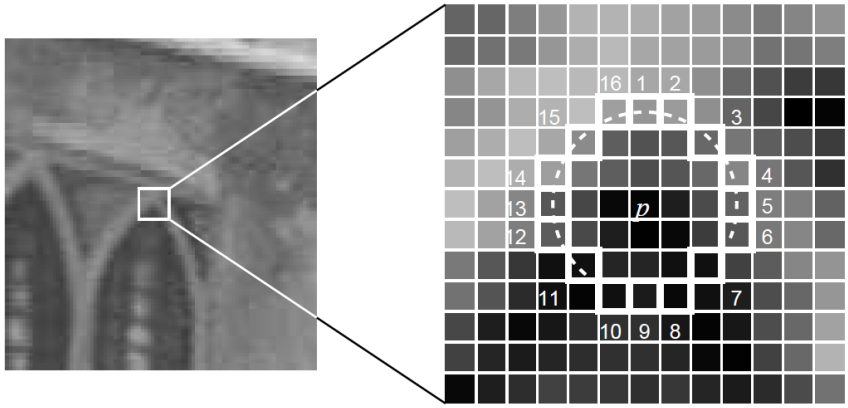

Fig. 2.2: FAST corner detection [28]

catenating the results of simple brightness comparison tests. In BRISK, the characteristic direction of each keypoint is determined in order to allow for orientation-normalized descriptors and hence achieve rotation invariance which is key to general robustness. Also, the brightness comparisons are carefully selected with the focus on maximizing descriptiveness.

The key concept of the BRISK descriptor makes use of a pattern used for sampling the neighborhood of the keypoint. The pattern, illustrated in Fig. 2.3, defines N locations equally spaced on circles concentric with the keypoint. In order to avoid aliasing effects when sampling the image intensity of a point p_i in the pattern, Gaussian smoothing is applied with standard deviation σ_i proportional to the distance between the points on the respective circle.

The sampling pattern is positioned and scaled accordingly for a particular keypoint k in the image. Considering one of the $N \times (N-1)/2$ sampling-point pairs (p_i, p_j), the smoothed intensity values at these points which are $I(p_i, \sigma_i)$ and $I(p_j, \sigma_j)$, respectively, are used to estimate the local gradient $g(p_i, p_j)$ by

$$g(p_i, p_j) = (p_j - p_i) \cdot \frac{I(p_j, \sigma_j) - I(p_i, \sigma_i)}{\| (p_j - p_i) \|^2} \tag{2.1}$$

Considering the set $\mathcal{A}$ of all sampling-point pairs:

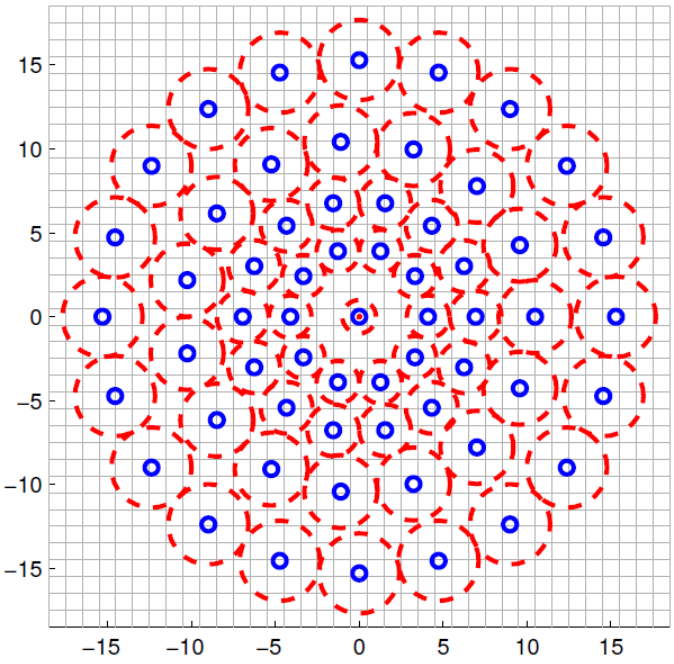

Fig. 2.3: BRISK sampling pattern [6]

$$\mathcal{A} = \left\{ (p_i, p_j) \in \mathbb{R}^2 \times \mathbb{R}^2 | i < N \wedge j < i \wedge i, j \in \mathbb{N} \right\} \tag{2.2}$$

a subset of short-distance pairings $\mathcal{S}$ and another subset of long-distance pairings $\mathcal{L}$ are defined:

$$\mathcal{S} = \{ (p_i, p_j) \in A | \parallel p_j - p_i \parallel < \delta_{max} \} \subseteq \mathcal{A} \tag{2.3}$$

$$\mathcal{L} = \{ (p_i, p_j) \in A | \parallel p_j - p_i \parallel > \delta_{min} \} \subseteq \mathcal{A} \tag{2.4}$$

The threshold distances are set to $\delta_{max} = 9.75t$ and $\delta_{min} = 13.67t$ (t is the scale of the keypoint k). Iterating through the point pairs in $\mathcal{L}$, the overall characteristic pattern direction of the keypoint k is estimated as:

$$g = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(p_i, p_j) \in \mathcal{L}} g(p_i, p_j) \tag{2.5}$$

where g_x and g_y are the components of the pattern direction along the x-axis and the y-axis, respectively.

For the formation of the rotation- and scale-normalized descriptor, BRISK applies the sampling pattern rotated by $\alpha = \arctan(g_y, g_x)$ around the keypoint k. The bit-vector descriptor d_k is assembled by performing all the short distance intensity comparisons of point pairs $(p_i^\alpha, p_j^\alpha) \in \mathcal{S}$ (p_i^α, p_j^α are p_i, p_j rotated by α), such that each bit b corresponds to:

$$b = \begin{cases} 1, & I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0, & \text{otherwise} \end{cases} \qquad \forall (p_i^\alpha, p_j^\alpha) \in \mathcal{S} \tag{2.6}$$

Descriptor Matching

Matching two BRISK descriptors is a simple computation of their Hamming distance. The Hamming distance d_H is simply the number of different bits in a pair of binary strings, i.e., the number of ones in the binary string resulting from the bitwise XOR operation between a pair of binary strings. The Hamming distance between two descriptors is a measure of their dissimilarity. The number of ones in a binary string is known as its population count ($POPCNT$), so the Hamming distance between two descriptors v_i, v_j can be written as:

$$d_H(v_i, v_j) = POPCNT(v_i \oplus v_j) \tag{2.7}$$

The operations in the Hamming distance involves a bitwise XOR followed by a bit count, which can both be computed very efficiently on modern architectures. The population count of binary strings can be efficiently computed by the POPCNT instruction which was recently introduced with the Nehalem-microarchitecture-based Core i7 processors [30].

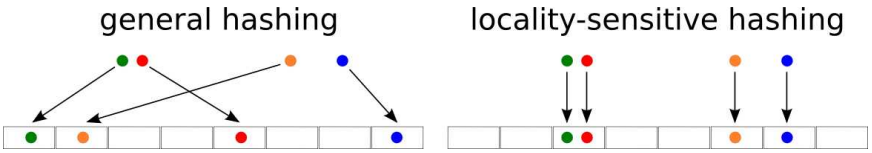

Fig. 2.4: General hashing compared to Locality-Sensitive Hashing

2.2 Binary Hashing Methods

The general idea of hashing is to represent data points with some identifiers or codes while avoiding collisions of these codes. The idea of LSH is to exploit collisions for mapping points which are nearby (in geometrical sense) into the same bin or bucket, as shown in Fig. 2.4. Binary hashing is the process of encoding high dimensional data into lower dimensional binary codes. The hashing functions ensures that near neighbor data points generates near or similar binary codes. This section provides a detailed discussion about some binary hashing methods used for solving Approximate Nearest Neighbor (ANN) problems. These methods are used in the proposed work for achieving better speed-up and accuracy.

2.2.1 Locality-Sensitive Hashing (LSH)

Locality-Sensitive Hashing (LSH) is an approach to solving approximate nearest neighbor problems that allows to quickly find similar entries in large databases. This approach belongs to an interesting class of algorithms that are known as randomized algorithms. A randomized algorithm does not guarantee an exact answer but instead provides a high probability guarantee that it will return the correct answer or one close to it. By investing additional computational effort, the probability can be pushed as high as desired [31].

LSH is based on the simple idea that, if two points are close together, then after a *projection* operation these two points will remain close together. This idea can be easily understood using the examples shown in Fig. 2.5. Two points that are close together on the sphere are also close together when the sphere is projected onto the two dimensional page. This is true no matter how the sphere is rotated. Two other points on the sphere that are far

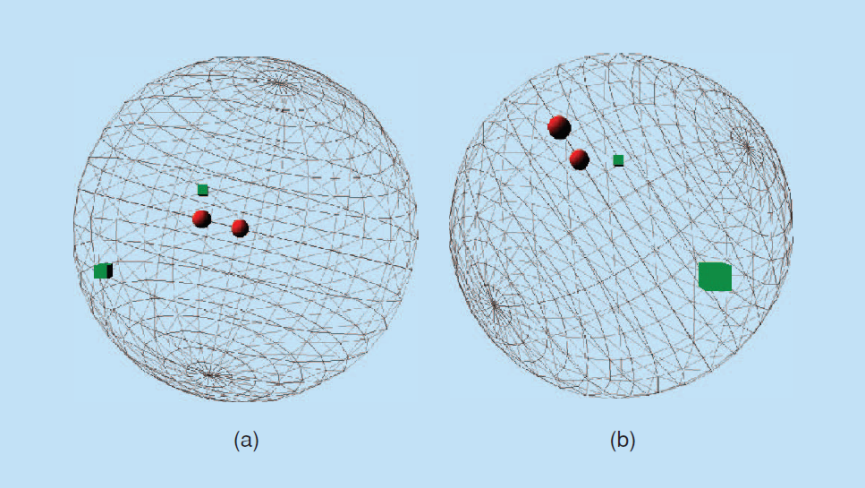

Fig. 2.5: Projections of two close (circles) and two distant (squares) points onto the printed page [31]

apart will, for some orientations, be close together on the page, but it is more likely that the points will remain far apart.

To further expand this basic idea, starting with a random projection operation that maps a data point from a high-dimensional point to a low-dimensional subspace. First, this keeps track of points that are close to the query point. Second, projections from a number of different directions are created to keep track of the nearby points. The list of nearby points that appear close to each other in more than one projection are the most close to the query point.

Random Projections: The Dot Product

At the core of LSH is the scalar projection (or the *dot* product), given by $h(\vec{v}) = \vec{v} \cdot \vec{x}$, where $\vec{x}$ is a query point in a high-dimensional space, and $\vec{v}$ is a vector with components that are selected at random from a Gaussian distribution. This scalar projection is then quantized into a set of *hash bins*, with the intention that nearby items in the original space will fall into the same bin. The resulting full hash function is given by

$$h^{x,b}(\vec{v}) = \left\lfloor \frac{\vec{x} \cdot \vec{v} + b}{w} \right\rfloor \tag{2.8}$$

where $\lfloor \cdot \rfloor$ is the floor operation, w is the width of each quantization bin, and b is a random variable uniformly distributed between 0 and w.

For the projection operator to be locality-sensitive, it must project nearby points to positions that are close together. This requires that:

- For any points p and q in $\mathbb{R}^d$ that are close to each other, there is a high probability P_1 that they fall into the same bucket

$$P_H[h(p) = h(q)] \geq P_1 \quad for \quad \| p - q \| \leq R_1 \tag{2.9}$$

- For any points p and q in $\mathbb{R}^d$ that are far apart, there is a low probability $P_2 < P_1$ that they fall into the same bucket

$$P_H[h(p) = h(q)] \leq P_2 \quad for \quad \| p - q \| \geq cR_1 = R_2 \tag{2.10}$$

  where c is a small constant.

In Equations 2.9 and 2.10, $\| \cdot \|$ is the L_2 vector norm and $R_2 > R_1$. Due to the linearity of the dot product, the difference between two image points $\| h(p) - h(q) \|$ has a magnitude whose distribution is proportional to $\| p - q \|$, therefore, $P_1 > P_2$.

Random Projections: The k Dot Products

The difference between P_1 and P_2 can be further magnified by performing k dot products in parallel, where $k \in \mathbb{Z}^+$. This increases the ratio of the probabilities that points at different separations will fall into the same quantization bin, since $(P_1/P_2)^k > (P_1/P_2)$. The resulting projection, is obtained by performing the k independent dot products to transform the query point $\vec{v}$ into k real numbers. As with the scalar (dot product) projection, the k inner products are quantized, as shown in Equation 2.8, with the intention that nearby points will fall in the same bucket in all dimensions.

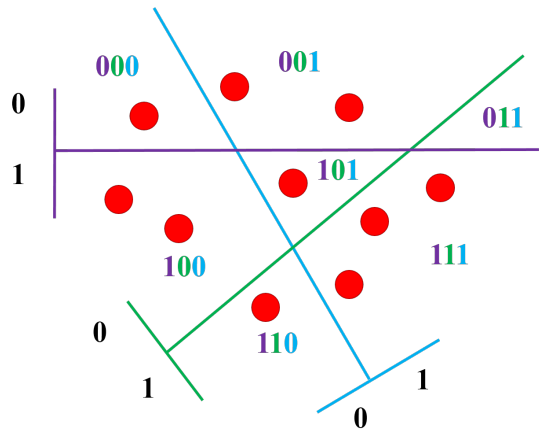

Fig. 2.6: An example of 2-dimensional binary projections for LSH

Increasing the quantization bucket width w will increase the number of points that fall into each bucket. To obtain the final nearest neighbor result, a linear search have to be performed through all the points that fall into the same bucket as the query, so varying w effects a trade-off between a larger table with a smaller final linear search, or a more compact table with more points to consider in the final search.

Binary LSH

Binary hashing can be simplified by defining a *projection hyperplane* for each hashing function (hyperplanes are planes defined in more than three dimensions). The projection hyperplane is represented by its perpendicular vector $\vec{x}$. Data points that fall on or above the hyperplane are given a code of 1 and data points that fall under the hyperplane are given a code of 0. The total bits resulting from the hash functions constitutes the binary code. The generated binary codes may be used to build one or more hash tables or any other indexing data structure which is used in the search process. A simplified 2-dimensional binary hashing example is shown in Fig. 2.6. A binary hashing function can be written as

$$h^x(\vec{v}) = \begin{cases} 1, & \vec{x}\cdot\vec{v} \geq 0 \\ 0, & \vec{x}\cdot\vec{v} < 0 \end{cases} \tag{2.11}$$

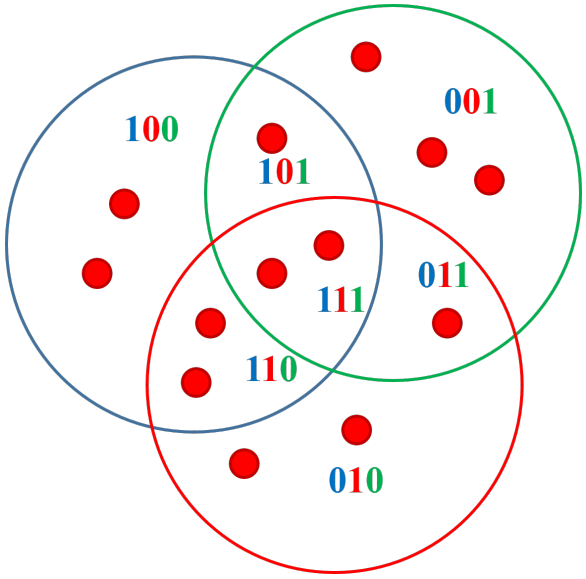

Fig. 2.7: Spherical Hashing simplified on two dimensions

2.2.2 Spherical Hashing

Spherical Hashing (SH) [19] is an efficient binary hashing approach, in which all data vectors are projected over hypersphere-based, instead of hyperplanes, hashing functions. Each hash function is represented by a hypersphere with some center vector and a radius, as shown in Fig. 2.7. In Spherical Hashing, the hash functions are optimized to balance the partitioning of data and the independency between them. Unlike LSH, each spherical hash function generates a binary bit depending on whether a vector resides inside or outside a hypersphere. Spherical Hashing has been proved more efficient than most of the state-of-the-art hashing methods.

Binary Code Generation with Spherical Hashing

Given a set of n data points in a D-dimensional space

$$X = \{x_1, ..., x_n\}, \quad x_i \in \mathbb{R}^D \tag{2.12}$$

A binary code corresponding to each data point x_i is defined by

$$b_i = \{0,1\}^c \tag{2.13}$$

where c is the length of the code.

A set of Spherical Hashing functions

$$H(x) = (h_1(x), ..., h_c(x)) \tag{2.14}$$

maps points in $\mathbb{R}^D$ into the binary hypercube $\{0,1\}^c$. Each spherical hashing function $h_k(x)$ is defined by a hypersphere. Each hypersphere is defined by a pivot $p_k \in \mathbb{R}^D$ and a distance threshold $t_k \in \mathbb{R}^+$ as the following:

$$h_k(x) = \begin{cases} 0, & d(p_k,x) > t_k \\ 1, & d(p_k,x) \leq t_k \end{cases} \tag{2.15}$$

where $d(\cdot,\cdot)$ denotes the Euclidean distance between two points in $\mathbb{R}^+$; various distance metrics (e.g., L_p metrics) can be used instead of the Euclidean distance. The value of each spherical hashing function $h_k(x)$ indicates whether the point x is inside the hypersphere whose center is p_k and radius is t_k.

The key difference between using hyperplanes and hyperspheres for computing binary codes is their abilities to define a closed region in $\mathbb{R}^D$ that can be indexed by a binary code. To define a closed region in a d-dimensional space, at least $d+1$ hyperplanes are needed, while only a single hypersphere is sufficient to form such a closed region in an arbitrarily high dimensional space. Furthermore, unlike using multiple hyperplanes a higher number of closed regions can be constructed by using multiple hyperspheres, while the distances between points located in each region are bounded. In addition, a hyperplane can be approximated by a large hypersphere (e.g. a large radius and a far-away center).

Spherical Hamming Distance

Most hyperplane-based binary embedding methods use the Hamming distance between two binary codes, which measures the number of different bits, i.e. $|b_i \oplus b_j|$, where $\oplus$ is the XOR bit operation and $|\cdot|$ denotes the number of 1 bits in a given binary code (recall Equation 2.7). This distance metric measures the number of hyperplanes that two given points reside in the opposing side of them. The Hamming distance, however, does not well reflect the property related to defining closed regions with tighter bounds, which is the core benefit of using spherical hashing functions.

To fully utilize desirable properties of spherical hashing functions, the following distance metric was proposed, the *spherical Hamming distance* ($d_{shd}(b_i, b_j)$), between two binary codes b_i and b_j generated by spherical hashing:

$$d_{shd}(b_i, b_j) = \frac{|b_i \oplus b_j|}{|b_i \wedge b_j|} \tag{2.16}$$

where $|b_i \wedge b_j|$ denotes the number of common 1 bits between two binary codes (the bitwise AND operation).

Having the common 1 bits in two binary codes gives tighter bound information than having the common 0 bits in spherical hashing functions. This is mainly because each common 1 bit indicates that two data points are inside its corresponding hypersphere, giving a stronger cue in terms of distance bounds of those two data points.

Independence between Spherical Hashing Functions

Achieving balanced partitioning of data points for each hashing function and the independence between hashing functions has been known to be important [18, 32, 33], since independent hashing functions distribute points in a balanced manner to different binary codes. It has been known that achieving such properties lead to minimizing the search time [32] and improving the accuracy [33].

To achieve this independency between spherical hashing functions, each hashing function h_k is defined to have equal probability for 1 and 0 bits as the following:

$$Pr[h_k(x) = 1] = Pr[h_k(x) = 0] = \frac{1}{2}, \quad x \in X, \quad 1 \leq k \leq c \tag{2.17}$$

this achieves balanced partitioning of data points for each bit.

It is known that two probabilistic events V_i and V_j to are independent if and only if $Pr[V_i \cap V_j] = Pr[V_i] \cdot Pr[V_j]$. If Equation 2.17 is achieved, the independence between two hashing function can satisfy the following equation:

$$Pr[h_i(x) = 1, h_j(x) = 1] = Pr[h_i(x) = 1] \cdot Pr[h_j(x) = 1] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \tag{2.18}$$

given that $x \in X$ and $1 \leq i, j \leq c$.

Iterative Optimization

An iterative optimization process is used in Spherical Hashing for:

- Computing c different hyperspheres, i.e. their pivots p_k and distance thresholds t_k.
- Satisfying balanced partitioning and independency constraints shown in Equations 2.17 and 2.18.

As the first phase of the iterative process, a sample subset $S = \{s_1, s_2, ..., s_m\}$ from data points X are chosen to approximate its distribution. Then the pivots of c hyperspheres are initialized with randomly chosen c data points in the subset S.

As the second phase of the iterative process, the pivots of hyperspheres are refined and their distance thresholds are computed. To help these computations, the following two

variables are computed, o_i and $o_{i,j}$, given $1 \leq i, j \leq c$:

$$\begin{aligned} o_i &= |\{s_k | h_i(s_k) = 1, \quad 1 \leq k \leq m\}| \\ o_{i,j} &= |\{s_k | h_i(s_k) = 1, h_j(s_k) = 1, 1 \leq k \leq m\}| \end{aligned} \tag{2.19}$$

where $|\cdot|$ is the cardinality of the given set. o_i measures how many data points in the subset S have 1 bit for ith hashing function and is used to satisfy balanced partitioning for each bit (recall Equation 2.17). Also, $o_{i,j}$ measures the number of data points in the subset S that are contained within both of two hyperspheres corresponding to ith and jth hashing functions. $o_{i,j}$ will be used to satisfy the independence between ith and jth hashing functions (recall Equation 2.18) during the iterative optimization process.

Once o_i and $o_{i,j}$ are computed with data points in the subset of S, two alternating steps are run to refine pivots and distance thresholds for hyperspheres. First, the pivot positions of two hyperspheres are adjusted in a way that $o_{i,j}$ becomes closer to or equal to $\frac{m}{4}$. Intuitively, for each pair of two hyperspheres i and j, when $o_{i,j}$ is greater than $\frac{m}{4}$, a *repulsive force* is applied to both pivots of those two hyperspheres (i.e. p_i and p_j) to place them farther away. Otherwise an *attractive force* is applied to locate them closer. Second, once pivots are computed, the distance threshold t_i of ith hypersphere is adjusted such that o_i becomes $\frac{m}{2}$ to meet balanced partitioning of the data points for the hypersphere (recall Equation 2.17).

The iterative optimization process is performed until the computed hyperspheres do not make further improvements in terms of satisfying constraints. Specifically, the mean and standard deviation of $o_{i,j}$ are considered as a measure of the convergence of the iterative process. Ideal values for the mean and standard deviation of $o_{i,j}$ are $\frac{m}{4}$ and zero respectively. However, in order to avoid over-fitting, the iterative process is stopped when the mean and standard deviation of $o_{i,j}$ are within $\varepsilon_m\%$ and $\varepsilon_s\%$, error tolerances, of the ideal mean and standard deviation of $o_{i,j}$ respectively; it was found that too low error tolerances lead to over-fitting while the values near 10% and 15% for $\varepsilon_m\%$ and $\varepsilon_s\%$, respectively, give reasonable results.

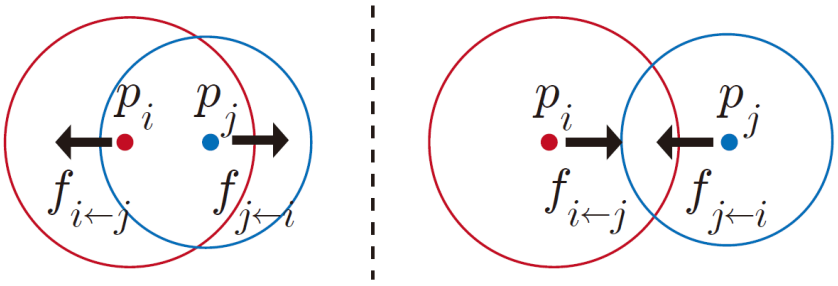

Fig. 2.8: Left: a repulsive force between two pivots p_i and p_j is computed since their overlap $o_{i,j}$ is larger than the desired amount. Right: an attractive force is computed because the overlap is smaller than the desired amount.

A (repulsive or attractive) force from pivot p_j to pivot p_i, $f_{i\leftarrow j}$, is defined as the following (Fig. 2.8):

$$f_{i\leftarrow j} = \frac{1}{2}\frac{o_{i,j} - m/4}{m/4}(p_i - p_j) \tag{2.20}$$

An accumulated force, f_i, is the average of all the forces computed from all the other pivots as the following:

$$f_i = \frac{1}{c}\sum_{j=1}^{c} f_{i\leftarrow j} \tag{2.21}$$

Once the accumulated force f_i is applied to p_i, then p_i is updated simply as $p_i + f_i$. The whole iterative optimization process is shown in Algorithm 1, Fig. 2.9.

2.3 Exhaustive-Search Equivalent Algorithms

Many exhaustive-search equivalent algorithms were introduced recently. These algorithms yields the same or too close results to exhaustive-search results with a significant speedup. Most of these algorithms depends on examining data points before matching them with the query point. Examples of these algorithms are Partial Distortion Elimination (PDE) [34], Projection Kernels (PK) [35], Low Resolution Pruning (LRP) [36] and Norm Ordered Matching (NOM) [27].

NOM has been proved to exceed other algorithms, in addition, it is easy to implement and adapt. Also, NOM and PDE can be used with any metric distance measure, including

Algorithm 1 Spherical hashing iterative optimization process

Input: sample data points $X = \{x_1, x_2, ..., x_m\}$, error tolerances ε_m, ε_s, and the number of hashing functions c

Output: pivot positions $p_1, ..., p_c$ and distance thresholds $t_1, ..., t_c$ for c hyperspheres

Initialize $p_1, ..., p_c$ with randomly chosen c data points from the set S
Determine $t_1, ..., t_c$ satisfy $o_i = \frac{m}{2}$
Compute $o_{i,j}$ for each pair of hashing functions
repeat
  for $i = 1$ to $c - 1$ **do**
    for $j = i + 1$ to c **do**
      $f_{i \leftarrow j} = \frac{1}{2} \frac{o_{i,j} - m/4}{m/4} (p_i - p_j)$
      $f_{j \leftarrow i} = -f_{i \leftarrow j}$
    end for
  end for
  for $i = 1$ to c **do**
    $f_i = \frac{1}{c} \sum_{j=1}^{c} f_{i \leftarrow j}$
    $p_i = p_i + f_i$
  end for
  Determine $t_1, ..., t_c$ satisfy $o_i = \frac{m}{2}$
  Compute $o_{i,j}$ for each pair of hashing functions
until $avg(o_{i,j}) \leq \varepsilon_m \frac{m}{4}$ *and* $std\text{-}dev(o_{i,j}) \leq \varepsilon_s \frac{m}{4}$

Fig. 2.9: Spherical hashing iterative optimization process

the Hamming distance which is used in this work, where PK and LRP are designed for L_2 distance. In NOM, not all descriptors are examined to find the nearest ones, but only a partition of them based on their norm values. This algorithm has been proved to significantly speed up exhaustive search. Follows a detailed overview about NOM.

2.3.1 Norm Ordered Matching (NOM)

NOM was originally proposed to solve the image template matching problem [27]. This section provides an explanation and adaptation of NOM in general form.

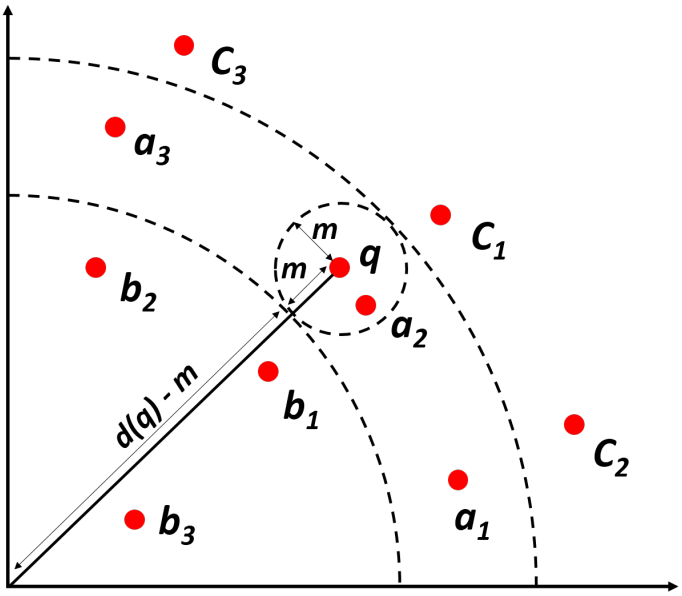

Fig. 2.10: NOM derivation. a_x, b_x, c_x are N-Dimensional vectors, q is the query vector. m is the current minimal distance.

Derivation of NOM

Representing a set of data points by N-dimensional vectors $x_i \in \mathbb{R}^N$ as shown in Fig. 2.10, let $d(p)$ be a norm defined over the vector space $\mathbb{R}^N$, so $d(p-q)$ represents the dissimilarity measure between the two N-Dimensional points p and q derived from the norm d.

Since d is a norm, it satisfies the triangle inequality, so we have:

$$|d(p) - d(q)| \leq d(p-q) \tag{2.22}$$

This splits into two inequalities:

$$d(p) - d(q) \leq d(p-q) \tag{2.23}$$

$$-(d(p) - d(q)) \leq d(p-q) \tag{2.24}$$

Equation 2.23 can be written as:

$$d(p) \leq d(q) + d(p-q) \tag{2.25}$$

If m is the current minimum distance to input query q as in Fig. 2.10, and for a candidate near neighbor p, it is found that (as the case for the c_x points in Fig. 2.10):

$$d(p) > d(q) + m \tag{2.26}$$

Comparing Equations 2.23 and 2.26, it is found that $m < d(p-q)$ so we can directly reject p without any further calculations, as it cannot improve the current minimum m.

In the same manner, Equation 2.24 can be written as:

$$-d(p) \leq d(p-q) - d(q) \tag{2.27}$$

$$d(p) \geq d(q) - d(p-q) \tag{2.28}$$

If for a candidate point p it is found that (as the case for the b_x points in the Fig. 2.10):

$$d(p) < d(q) - m \tag{2.29}$$

Comparing Equations 2.24 and 2.29, it could be found that $m < d(p-q)$ so p can be directly rejected without any further calculations, as it cannot improve the current minimum m.

Based on Equations 2.26 and 2.29 a lower and upper bounds can be derived on the allowable values of the norm of any candidate point $d(p)$. First two related quantities are defined:

$$QStart = d(q) - m, \quad QEnd = d(q) + m \tag{2.30}$$

Only points p where $QStart \leq d(p) \leq QEnd$ would need to be examined (a_x points in Fig. 2.10) and all other points (b_x, c_x points) would be skipped without any examination. An important thing to note here is that *QStart* and *QEnd* are both dependent on m, and finding a good m fast allows to tighten the range between *QStart* and *QEnd* fast and hence reducing the amount of examined points.

Evaluation Order

A crucial factor for the performance of any matching algorithm is how to find approximate values (either lower or upper bounds) of the true minimum distance early in the process, as this allows to reject most candidate points without the lengthy computing-intensive distance evaluation step. NOM proposed a novel evaluation order inspired by the bounding method derived previously in the text.

Considering Fig. 2.10, it can be noticed that there is a high probability of reaching closest point to query if points that are closer to query are examined first, in terms of norm.

It is true that this might allow distant points from q to be evaluated before other closer points (e.g. in Fig. 2.10 a_1 would be evaluated before b_1 and c_1 even though they are closer to q), but it was empirically proved that close points to q would be captured fast enough and with minimal number of unneeded evaluation.

The process is illustrated in Fig. 2.11. First the norm $d(p)$ is computed for all points and stored in an array, then the array is sorted based on the value of the norm $d(p)$, this is done once as an initialization step. Every time when a query point q is received, its norm $d(q)$ is computed, then binary search is performed over the list of sorted norms to find closest points to it (in terms of norm value), then candidates are evaluated in an up-down order, i.e., if binary search gave location x, candidates are evaluated in this order x, $x-1$, $x+1$, $x-2$, $x+2$, and every time a point that satisfies $d(p-q) < m$ is encountered, m, *QStart*, and *QEnd* are updated accordingly.

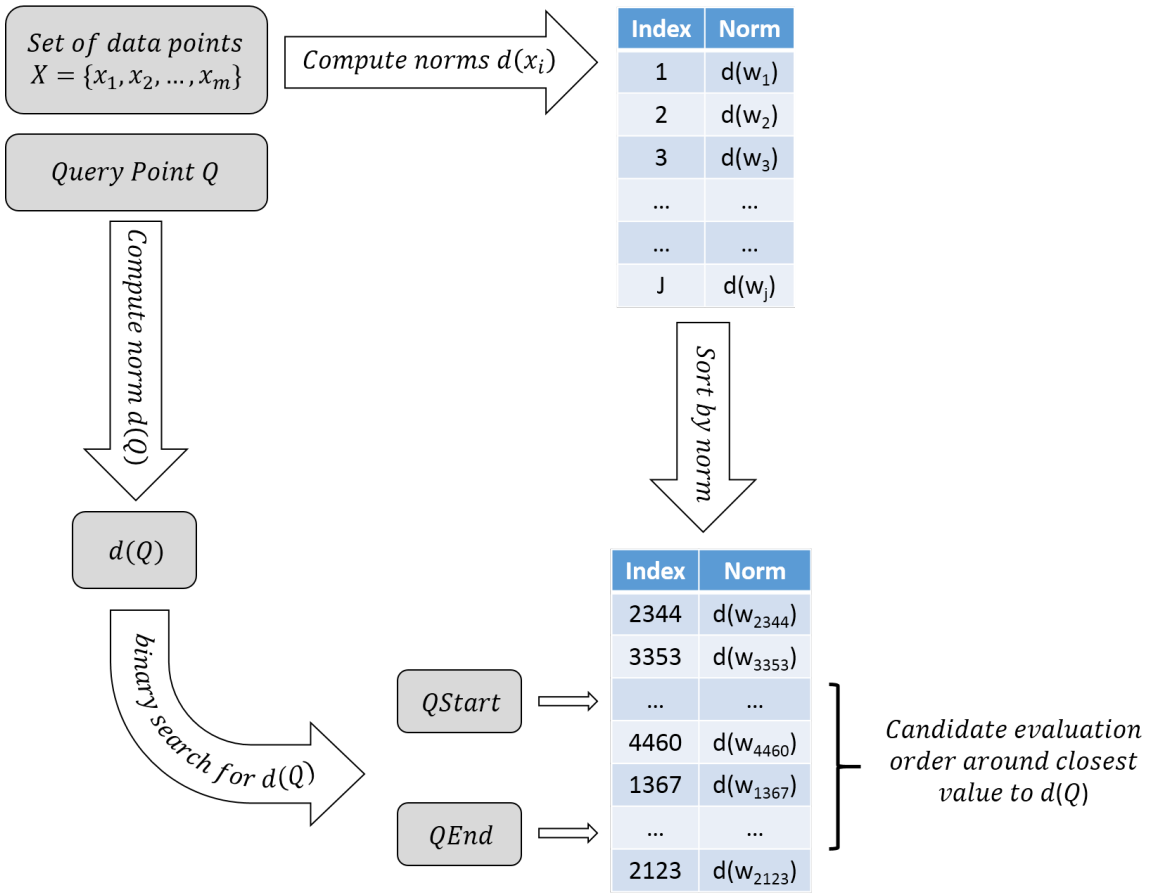

Fig. 2.11: NOM evaluation order of candidate points to input query Q according to dissimilarity measure derived from norm d.

Once a candidate point p is found that satisfies $d(p) < QStart$ or $d(p) > QEnd$, evaluation is stopped at that side, as this would be meaningless since the list of candidates is ordered by norm, for example, if $d(p) < QStart$ then it is time to stop searching the decreasing values of x since the list is sorted and all candidates in that direction would certainly be also less than $QStart$. Thus sorting the array gives two benefits, it allows to reach points close to real matching point fast and it also allows to prune whole side of the array at a single step without any further evaluation of inequality $QStart < d(p) < QEnd$. A Pseudo-code of NOM is demonstrated in Algorithm 2, Fig. 2.12.

Algorithm 2 Norm Ordered Matching algorithm (NOM)

Input: data points $X = \{x_1, x_2, ..., x_m\} \in \mathbb{R}^N$, query point Q
Output: nearest neigbor point to Q

for all data point $x_i \in X$ **do**
  Compute norm $d(x_i)$ and store it in an array *Norms*
end for
Sort the array *Norms* according to norm value
Compute norm of query point $d(Q)$
Perform binary search over the array *Norms* using $d(Q)$ as a key, store result in x
Let $QStart \leftarrow 0$, $QEnd \leftarrow minDist \leftarrow \infty$, $k \leftarrow -1$
for $i \leftarrow$ data points of indices $x, x-1, x+1, x-2, x+2, ...$ **do**
  $Id \leftarrow Norms[i]$
  if $Id < QStart$ **then**
    Skip all $i < Id$
  end if
  if $Id > QEnd$ **then**
    Skip all $i > Id$
  end if
  $dist \leftarrow Norm(Q - i)$
  if $dist < minDist$ **then**
    $minDist \leftarrow dist$
    $QStart \leftarrow d(Q) - min$
    $QEnd \leftarrow d(Q) + min$
    $k \leftarrow$ index of data point i in X
  end if
end for

Fig. 2.12: Norm Ordered Matching (NOM)

CHAPTER THREE

MULTI-BIN SEARCH

At first, it would be suitable to provide a formal definition for the problem of CBIR. Then, the approximated search process based on hashing methods is clarified. Then, a Single-Bin search method, based on exhaustive-search equivalent methods is presented, that greatly increases speedup. After that, the Multi-Bin search method is presented in detail. Finally, a result reranking step that increases search precision is discussed.

3.1 Formal Definitions

Given a set S of images

$$S = \{i_1, i_2, .., i_m\} \tag{3.1}$$

consisting of m images and a query image q, it is required to find a set R of k images,

$$R = \{i_1, i_2, .., i_k\} \quad where \quad R \subset S, \quad k << m \tag{3.2}$$

that contains the most similar k images for q in S. It is meant by similar images those containing a visually-similar scene or object with variations in illumination, viewpoint, scale, rotation, distortion, noise, etc.

In order to tell that two images are visually-similar, firstly, a way is needed to digitally describe the visual contents of these images. This is the task of image descriptors mentioned previously in Chapter 2 such as SIFT, SURF, BRIEF, BRISK, ORB, or FREAK. Image descriptors are no more than vectors of numbers describing specific regions of an image, as reviewed also in Chapter 2. So, an image I can be defined as a set of n vectors representing image descriptors:

$$I = \{v_1, v_2, .., v_n\} \tag{3.3}$$

This makes S can be replaced by:

$$S' = \{\{v_1, v_2, .., v_{n_1}\}_1, \{v_1, v_2, .., v_{n_2}\}_2, .., \{v_1, v_2, .., v_{n_m}\}_m\} \tag{3.4}$$

For simplicity, S' can be replaced by:

$$S'' = \{v_{11}, v_{12}, .., v_{1n_1}, v_{21}, v_{22}, .., v_{2n_2}, .., v_{m1}, v_{m2}, .., v_{mn_m}\} \tag{3.5}$$

3.2 Approximate Search with Binary Hashing

In large-scale image retrieval, the set of images S usually consists of thousands or millions of images. Knowing that each image usually consists of a few hundreds or a few thousands of descriptors, the total number of descriptors grows to the order of billions. In this work, hashing or quantization methods are used to approximate the search process.

Any binary hashing method can be defined as follows: given a set of data vectors as shown in Equation 3.5, each data vector v_i is represented by a binary code b_i as

$$b_i = \{0, 1\}^l \tag{3.6}$$

where l is the length of the binary code. In order to make this mapping from data vectors to binary codes, a set of l hashing functions is required

$$H(v) = (h_1(v), h_2(v), .., h_l(v)) \tag{3.7}$$

Each hash function generates a single bit in the binary code, 0 or 1, based on specific criteria defined by the function itself. The goal of the hashing functions is to generate the same binary code for the approximate nearest neighbor vectors.

After applying the hashing algorithm, the data vectors can be divided into a set G of *bins* or *buckets*

$$G = \{W_1, W_2, .., W_{n_w}\} \tag{3.8}$$

each bin contains the vectors with the same binary code. The maximum number of bins would be 2^l, i.e., $n_w \leq 2^l$.

The approximation here is to consider all the data vectors in a single bin as approximate nearest neighbors, i.e., matches to each other. So, the set of nearest neighbor vectors to a query vector x found inside a bin W is

$$NN(x) = \{y | y \in W, x \in W\} \tag{3.9}$$

Of course, the precision here depends on the algorithm used for hashing and the length of the generated binary code. It is clear that increasing the binary code length increases the precision.

After the hashing algorithm is applied and the binary codes are generated, it is time to index all data vectors inside one or more hash tables using the binary codes as table keys.

When a query vector is submitted, its binary code is computed and used to directly access the target bin in the hash table. A step further beyond this is to search all vectors inside the target bin for the nearest neighbor vectors within a distance threshold t_v, as shown

Algorithm 3 Searching a target bin for the nearest neighbors of a query vector using brute-force matching

Input: a query vector x, the target bin W_i where $x \in W_i$, and a distance threshold t_v
Output: a set $NN(x)$ containing the nearest neighbor vectors of the query vector x within the distance threshold t_v

```
1: for all y|y ∈ W_j do
2:     if d_H(x,y) < t_v then
3:         Add y to the set NN(x)
4:     end if
5: end for
```

Fig. 3.1: Searching a target bin for the nearest neighbors of a query vector using brute-force matching.

in Algorithm 3, Fig. 3.1. Of course, exhaustive or brute-force search would be very time consuming. In the next section, a first step toward solving this problem is presented.

3.3 Single-Bin Exhaustive-Equivalent Search (SBEES)

Exhaustively searching a target bin would be very time consuming due to the large number of vectors inside a single bin, even if the distance measure used to match vectors is very fast like the Hamming distance. So, up to now, the problem with approximate search is the time cost of searching for near neighbor vectors inside a target bin.

To solve this problem, an exhaustive-search equivalent algorithm is proposed inspired by NOM, discussed earlier in Section 2.3, has been used. Exhaustive-search equivalent algorithms yields same results as exact exhaustive-search algorithms with significant speedup. In the proposed algorithm, which is shown in Algorithm 4, Fig. 3.2, the population count, i.e. the number of ones in the binary string, is pre-computed offline for all vectors. The algorithm computes lower and upper bounds of population count that ensure a vector may result in a distance less than the threshold. If a vector's population count is outside these bounds, it is assured that it will result in a distance greater than the threshold, so it is skipped. As a result, the distance computation are skipped for a large number of

Algorithm 4 Searching a target bin for the nearest neighbors of a query vector using norm matching

Input: a query vector x, the target bin W_i where $x \in W_i$, a set of population counts of vectors inside the bin $P_i = \{p_y | y \in W_i\}$, and a distance threshold t_v

Output: a set $NN(x)$ containing the nearest neighbor vectors of the query vector x within the distance threshold t_v

1: $p_x \leftarrow POPCNT(x)$ ▷ compute number of 1 bits in x
2: $lowerBound \leftarrow p_x - t_v$
3: $upperBound \leftarrow p_x + t_v$
4: **for all** $y | y \in W_j$ **do**
5: **if** $lowerBound \leq p_y \leq upperBound$ **then**
6: **if** $d_H(x,y) < t_v$ **then**
7: Add y to the set $NN(x)$
8: **end if**
9: **end if**
10: **end for**

Fig. 3.2: Searching a target bin for the nearest neighbors of a query vector using NOM

the vectors inside the bin. This algorithm is different from NOM in some steps, there is no sorting of norm values and there is no change of the upper and lower bounds. The sorting step was skipped because, in the case of binary vectors, it adds much processing overhead while skipping a small group of vectors. The upper and lower bounds are not updated because the goal of this algorithm is to find the near neighbor vectors within a constant distance threshold t_v, not the exact nearest neighbor vector as in NOM.

3.4 Multi-Bin Exhaustive-Equivalent Search (MBEES)

Quantization errors are the major problem facing quantization, clustering, and hashing methods. A query descriptor may fall on the edge of a cluster or generate a hash code that is different from its neighbors' hash codes resulting in retrieving wrong results, i.e. false positives, as shown in Fig. 3.3. The proposed approach aims to minimize these quantization errors; instead of only examining the single bin or cluster that contains the query descriptor, the surrounding bins or clusters are also examined in order to minimize quanti-

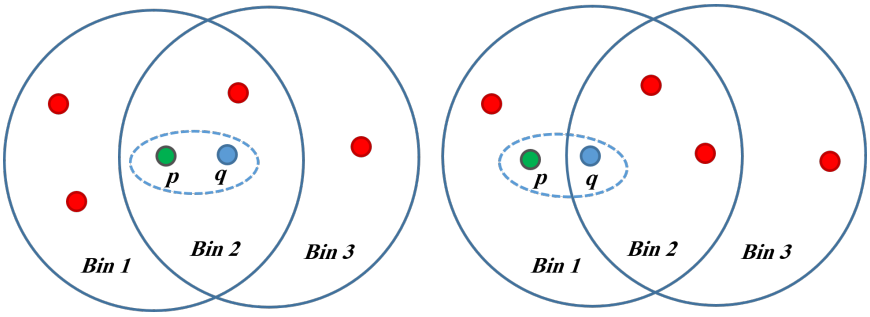

Fig. 3.3: Left: A query point q and its nearest neighbor p inside the same bin or cluster. Right: A query point q and its nearest neighbor p inside a different bin or cluster.

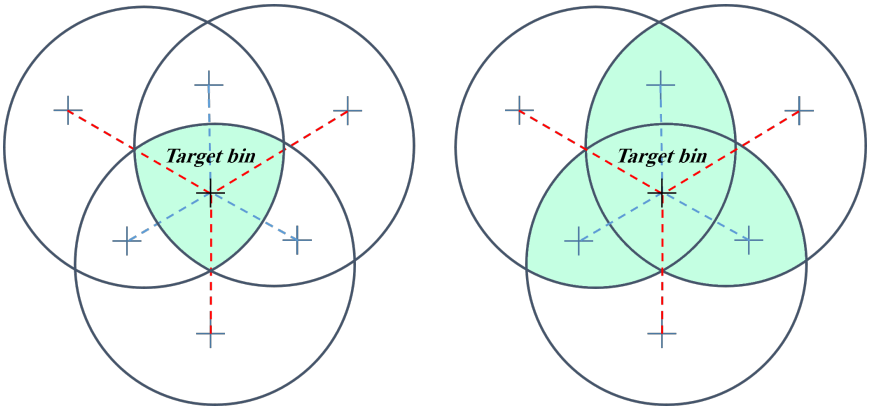

Fig. 3.4: Left: In Single-Bin search, only the targeted bin is searched for approximate nearest neighbor points (the colored area). Right: In Multi-Bin search, in addition to the targeted bin, nearest neighbor bins are also searched for approximate nearest neighbor points (the colored area).

zation errors, as shown in Fig. 3.4. Of course, examining more data will increase the search time, but the search accuracy would be greatly increased.

To minimize these quantization errors, for each bin W_i, Its nearest n_{w_i} bins are *precomputed* and stored, as shown in Algorithm 5, Fig. 3.5. These nearest bins are selected based on some distance threshold t_w, which is empirically chosen, where the distance between two bins $d(W_1, W_2)$ is computed as the distance between their centers, i.e., their average vector. This distance threshold is, of course, dependable on the binary code length. So, the set containing the indices of the nearest bins to some other bin w_i is

$$NN(W_i) = \{j | d(W_i, W_j) \leq t_w\} \tag{3.10}$$

Algorithm 5 Computing nearest neighbor bins for each bin

Input: bin centers $C = \{c_1, c_2, .., c_{n_w}\}$ of bins $G = \{W_1, W_2, .., W_{n_w}\}$, inter-bin distance threshold t_w

Output: a set containing the nearest neighbor bins for each bin $P = \{NN(W_1), NN(W_2), .., NN(W_{n_w})\}$ where $NN(W_i) = \{j | d(W_i, W_j) \leq t_w\}$

```
1: for i = 1 to n_w do
2:     Add i to the set NN(W_i)
3:     for j = i + 1 to n_w do
4:         if d(c_i, c_j) ≤ t_w then
5:             Add j to the set NN(W_i)
6:             Add i to the set NN(W_j)
7:         end if
8:     end for
9: end for
```

Fig. 3.5: Computing nearest neighbor bins for each bin

And a bin center c_i for some bin w_i with n_i vectors is

$$c_i = \frac{\sum_1^{n_i} x_i}{n_i} \tag{3.11}$$

Of course, the computation of nearest neighbor bins is done *offline* as a pre-processing step. After computing these nearest bins, they are searched, in addition to the target bin, for the nearest neighbors of the query vector x as shown in Algorithm 6, Fig. 3.6. Only vectors within a distance threshold t_v, which is impirically chosen, are considered near neighbors to x.

A remaining issue with the proposed approach is how to tell that an image in the dataset is a match for a query image depending on matching individual descriptors of the query image. To resolve this issue, an algorithm is used to tell a percentage of matching between a query image and the candidate images in the dataset. For each query image descriptor, if an approximate nearest neighbor is found, *one point* is added to the *score* of the image to which it belongs. At the end, these scores are normalized by dividing each image's score by the sum of descriptors in the image itself and the query image. The images are sorted in

Algorithm 6 Searching nearest neighbor bins for a query vector

Input: query vector x, a set containing the indices of the nearest neighbor bins $NN(W_i)$ where $x \in W_i$, and a distance threshold t_v

Output: a set $NN(x)$ containing the nearest neighbor vectors of the query vector

```
1:  p_x ← POPCNT(x)
2:  lowerBound ← p_x − t_v
3:  upperBound ← p_x + t_v
4:  for all j|j ∈ NN(W_i) do
5:     for all y|y ∈ W_j do
6:        if lowerBound ≤ p_y ≤ upperBound then
7:           if d_H(x,y) ≤ t_v then
8:              Add y to the set NN(x)
9:           end if
10:       end if
11:    end for
12: end for
```

Fig. 3.6: Searching nearest neighbor bins for a query vector

descending order by their scores and the top k results are picked. This method is shown in Algorithm 7, Fig. 3.7.

A major factor that significantly increases the precision of the resulting score of matching a pair of images is the actual distance between pairs of local descriptors. Of course, the shorter the distance between two descriptors is, the more similar they become. In other words, the similarity score between two descriptors is inversely proportional to the distance between them. So, it would be more precise to replace line 8 in Algorithm 7, Fig. 3.7 by

$$s_{imgIndex} \leftarrow s_{imgIndex} + (\frac{1}{d(v_i, y)}). \tag{3.12}$$

3.5 Re-ranking Results

Exhaustive image matching would not be time-costing if there are few image to match, and it should produce more accurate results than approximated matching techniques. So, in this

Algorithm 7 Computing image scores

Input: a set of images $S = \{I_1, I_2, .., I_m\}$, a set $NN(I_i) = \{NN(v_1), NN(v_2), .., NN(v_{n_i})$ containing the sets of the approximate nearest neighbors of each descriptor vector in the query image I_i, and number of top matched images k

Output: a set of the top k matched images $R = \{I_1, I_2, .., I_k\}$

Let $Scores = \{s_1, s_2, .., s_m\}$
for $k = 1$ to m **do**
    $s_k \leftarrow 0$
end for
for $i = 1$ to n_i **do**
    for all $y | y \in NN(v_i)$ **do**
        $imgIndex \leftarrow j | y \in I_j$
        $s_{imgIndex} \leftarrow s_{imgIndex} + 1$
    end for
end for
Sort *Scores* descending
$R \leftarrow$ first k elements in *Scores*

Fig. 3.7: Computing image scores

section, a results reranking method is presented that is based on the exhaustive matching of images.

In the reranking method, the top k retrieved images are exhaustively re-matched with the query image. This method depends on approximating the exhaustive matching of a pair of images. Exhaustively matching a pair of images, say I_1 and I_2, using their individual descriptors requires finding the *exact* nearest neighbor of each descriptor in I_1 from all descriptors in I_2 and vice versa. This is a time consuming method, so it is better to approximate it by using some distance threshold t and searching for the *first* descriptor in I_2 which is at distance $d \leq t$ from each descriptor in I_1. Once a match descriptor found, the rest is skipped.

This algorithm does not go backward, it only matches the descriptors from one image to the other. This method is shown in Algorithm 8, Fig. 3.8. This algorithm also benefits from the NOM algorithm, discussed earlier in the text, in speeding the search for matched descriptors. The results will show that this approximated reranking method highly in-

Algorithm 8 Approximated matching algorithm for matching a pair of images

Input: two sets of image descriptor vectors $I_1 = \{v_{11}, v_{12}, .., v_{1n}\}, I_2 = \{v_{21}, v_{22}, .., v_{2m}\}$ where $n \geq m$, their corresponding population counts $P_1 = \{p_{11}, p_{12}, .., p_{1n}\}, P_2 = \{p_{21}, p_{22}, .., p_{2m}\}$, and a distance threshold t_v

Output: a score s representing the matching percentage between I_1 and I_2

```
1:  s ← 0
2:  for i = 1 to n do
3:      lowerBound ← p_1i − t_v
4:      upperBound ← p_1i + t_v
5:      for j = 1 to m do
6:          if lowerBound ≤ p_2j ≤ upperBound then
7:              if d_H(v_1i, v_2j) ≤ t_v then
8:                  s ← s + 1
9:              end if
10:         end if
11:     end for
12: end for
13: s ← s/(n + m)
```

Fig. 3.8: Approximated matching algorithm for matching a pair of images

creases retrieval precision while costing constant and small amount of time per single query.

CHAPTER FOUR

EXPERIMENTAL RESULTS

This chapter presents the evaluation protocol of the proposed approach and the benchmarking dataset. Then, experimental evaluation results are presented and discussed thoroughly.

4.1 Evaluation Protocol

The proposed approach has been evaluated against three famous datasets:

- The University of Kentucky Benchmarking (UKB) [13]: consists of 10200 images grouped into 2550 categories, each category contains four images that are considered matches to each other. Given a query image, it is required to get the image itself and the other three images in the same category as the top four results. So, the precision measure is a floating-point *score* in the range from 0 to 4 representing the number of retrieved correct matches in the top four results averaged over the number of run queries. The first 5 categories of the UKB are shown in Fig. 4.1.

- The INRIA Holidays [37]: a collection of 1491 holiday images, 500 of them being used as queries. The dataset includes a very large variety of scene types (natural, man-made, water and fire effects, etc) and images are in high resolution. The accuracy is measured by the mean Average Precision (mAP) as defined in [38].

Fig. 4.1: The first 5 categories of University of Kentucky Benchmarking (UKB) dataset.

- The MIRFLICKR-1M [39]: a collection consists of one million images downloaded from the social photography site Flickr through its public API. This dataset is merged with the other datasets to evaluate the accuracy and efficiency on a large scale.

The proposed methods have been applied on the famous binary hashing methods, LSH and SH. For LSH. Another variation of the LSH algorithm is evaluated, that requires to center the data vectors around the origin vector, i.e. zero-centered, this variation is denoted LSHZC. BRISK local image descriptors were generated of size 512 bits using a detection threshold of 70 for the FAST keypoint detector, these parameters showed better recognition precision as stated in [6]. The proposed methods has been compared to the BoVW approach using the implementation of [40].

All experiments were run on an Intel®Core^{TM}i7-950 processor with 8 MB cache and 3.06 GHz clock speed, and 8 GB memory. An advantage was taken of the newly introduced POPCNT instruction which was introduced with the Nehalem-microarchitecture-based Core i7 processors. This instruction efficiently computes the population count of binary strings. POPCNT has been used in measuring Hamming distances between bins or vectors and in the offline pre-computing of the population counts for all bins and vectors.

Table 4.1: Distance thresholds used in computing nearest neighbor bins for each binary code length

	Binary code length							
	12	16	20	24	28	32	36	40
Distance threshold (decimal)	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Distance threshold (quantized)	2	2	3	3	4	4	5	5

In all experiments, all the 10200 images of UKB and the 500 queries of INRIA Holidays were run as queries and the average score or precision was computed. Hashing methods were tested with binary code lengths ranging from 12 to 40. For Multi-bin search, inter-bin distance thresholds were given values as a percent of the binary code lengths, a percent of 0.125 was picked, i.e. 1 bit for each 8 bits in the binary code. This threshold empirically showed the best trade-off between search time and accuracy. Threshold values for different binary code lengths are shown in Table 4.1. In the reranking step, only the top 50 results were reranked to make a trade-off between accuracy and time. For large-scale evaluation, the UKB dataset was merged with various portions of the MIRFLICKR-1M.

All evaluated methods were denoted as follows:

- BoVW: Bag of Visual Words.
- SH: Spherical Hashing.
- LSH: Locality Sensitive Hashing.
- LSHZC: Locality Sensitive Hashing with Zero-Centered vectors.
- SBEES: Applying the Single-Bin Exhaustive Equivalent Search on some hashing method.
- MBEES: Applying the Multi-Bin Exhaustive Equivalent Search on some hashing method.
- R: Applying a Reranking step.

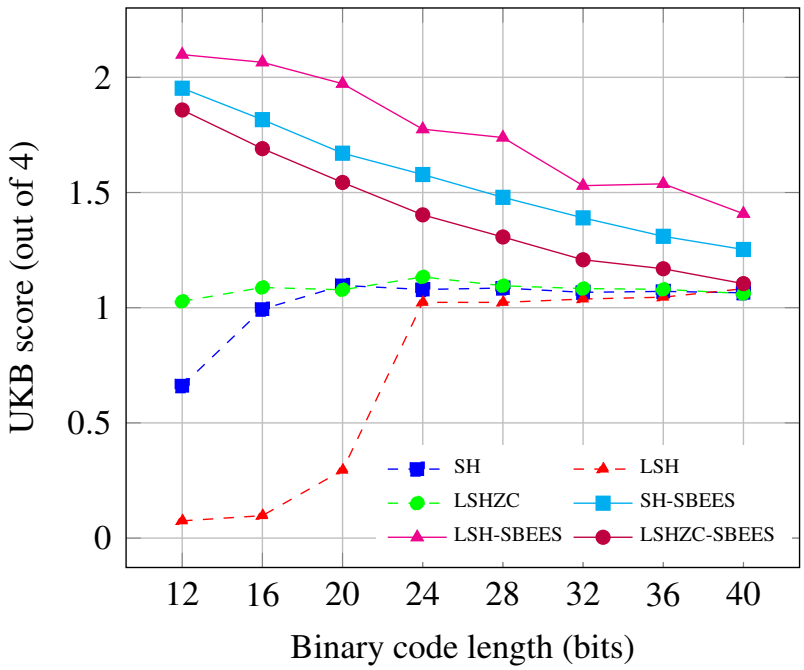

Fig. 4.2: Comparison of retrieval precision (UKB score) between hashing methods before and after applying the Single-Bin Exhaustive Equivalent Search.

4.2 Single-Bin Exhaustive-Equivalent Search (SBEES)

Figure 4.2 shows the scores of the evaluated methods SH, LSH, and LSHZC, along with the improvement in score resulting from applying the Single-Bin Exhaustive Equivalent Search (SBEES). It is shown that SBEES improves the retrieval precision (UKB score) of all evaluated methods by various amounts depending on binary code length. For instance, with binary code length 24, SBEES improves the retrieval precision of SH, LSH, and LSHZC by 46.24%, 73.46%, and 23.77% respectively.

Table 4.2 shows average query times over 10200 queries for SH, LSH, and LSHZC before and after applying SBEES. Of course, original hashing methods SH and LSHZC, especially with larger binary code lengths, have the shortest query times but with very low precision values as shown in Fig. 4.2. The LSH and LSH-SBEES query times are too long compared to other methods, this is due to the large sizes of bins generated by LSH. This is shown in Fig. 4.3, the total number of bins generated by each hashing method for

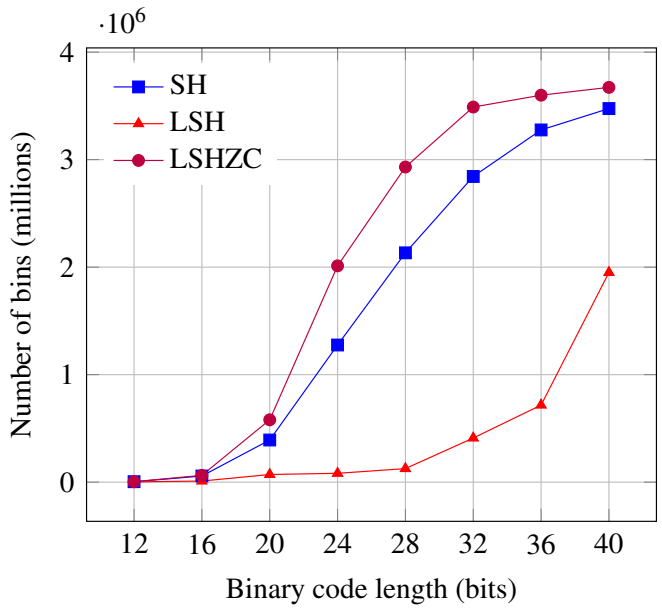

Fig. 4.3: Total number of bins generated by each hashing method.

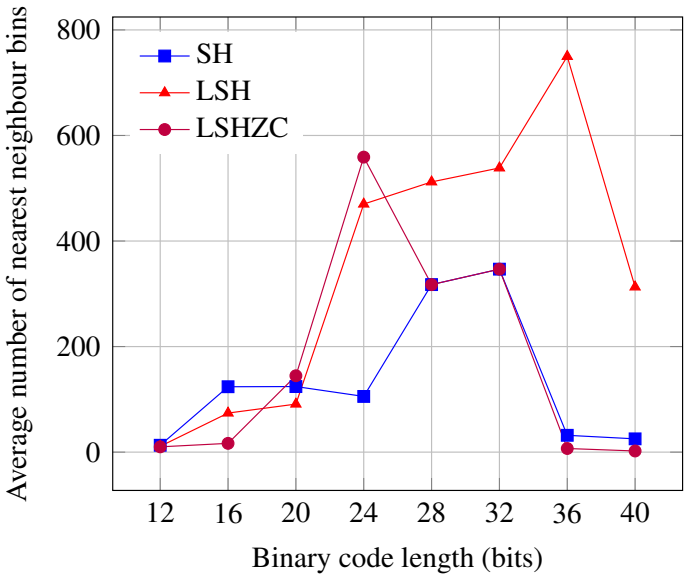

Fig. 4.4: Average number of nearest neighbor bins for each bin generated by each hashing method.

various binary code lengths. It is clear that LSH generates the smallest number of bins. This yields larger bins that require more time to examine. LSHZC-SBEES has the shortest times among all SBEES methods taking only 165 microseconds with binary code length of 24 bits.

Table 4.2: Average query times of SBEES compared to original hashing methods (milliseconds)

Search methods	Binary code length							
	12	16	20	24	28	32	36	40
SH	7.753	1.603	0.330	0.188	0.095	0.063	0.048	0.044
LSH	154.024	119.785	44.162	3.869	2.248	0.331	0.385	0.094
LSHZC	5.979	0.770	0.222	0.070	0.045	0.093	0.040	0.038
SH-SBEES	85.644	17.257	2.875	1.324	0.527	0.314	0.148	0.118
LSH-SBEES	3065.790	2125.170	586.660	35.178	20.531	2.127	2.980	0.381
LSHZC-SBEES	55.656	5.723	1.230	0.165	0.136	0.055	0.059	0.113

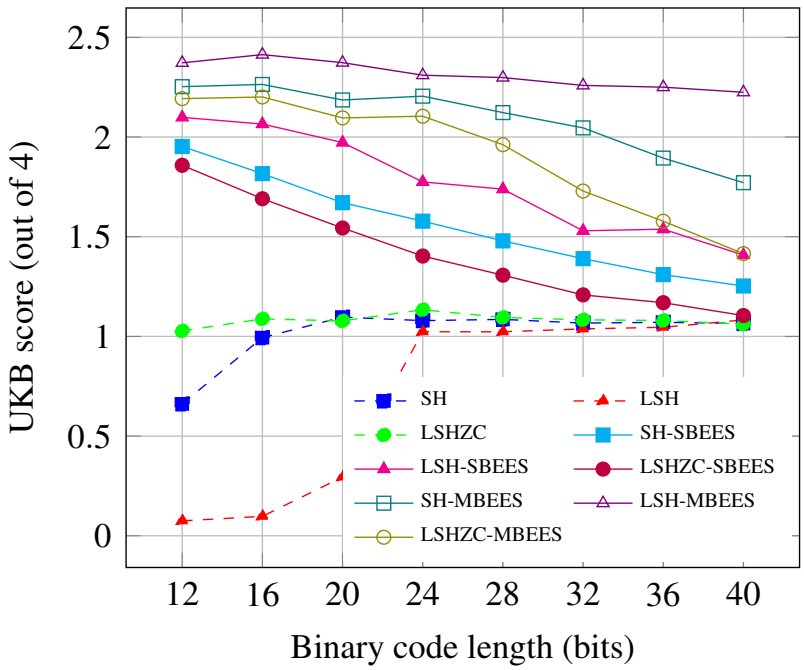

Fig. 4.5: Comparison of retrieval precision (UKB score) between hashing methods before and after applying the Single-Bin Exhaustive Equivalent Search (SBEES) and the Multi-Bin Exhaustive Equivalent Search (MBEES).

4.3 Multi-Bin Exhaustive-Equivalent Search (MBEES)

Figure 4.5 shows the scores of the evaluated methods SH, LSH and LSHZC, and the improvement in score resulting from applying SBEES and the Multi-Bin Exhaustive Equivalent Search. It is shown that MBEES improves the retrieval precision (UKB score) of all evaluated methods. For instance, with binary code length 24, MBEES improves the retrieval precision of SH, LSH, and LSHZC by 104.32%, 125.77%, and 85.64% respectively.

It is noticed in Fig. 4.2 and Fig. 4.5 that increasing the binary code length beyond specific values results in decreasing the retrieval precision, this is due to the increasing number of generated bins from the hashing algorithm, as shown in Fig. 4.3, which results in smaller bins. Smaller bins means less data to search in each bin, therefore, the precision decreases.

Table 4.3 shows average query times over 10200 queries for SH, LSH, and LSHZC before and after applying SBEES and MBEES. LSHZC-MBEES has the shortest times among all MBEES methods taking 86.607 milliseconds with binary code length of 24 bits.

Table 4.4 shows the retrieval precision (measured in mAP) and average query times (measured in milliseconds) of the proposed methods SBEES and MBEES compared to the BoVW method over the Holidays dataset. It is shown that the proposed methods mostly exceeds the BoVW in precision and time.

Figure 4.6 shows a comparison of the Recall@R (averaged over 500 queries) between SBEES, MBEES methods (using 24 bits) and BoVW (with two different vocabulary sizes) over the Holidays dataset. It is shown that SBEES and MBEES achieves higher recalls sooner than BoVW.

Table 4.3: Average query times of MBEES compared to original hashing methods and SBEES (milliseconds)

Search methods	Binary code length							
	12	16	20	24	28	32	36	40
SH	7.753	1.603	0.330	0.188	0.095	0.063	0.048	0.044
LSH	154.024	119.785	44.162	3.869	2.248	0.331	0.385	0.094
LSHZC	5.979	0.770	0.222	0.070	0.045	0.093	0.040	0.038
SH-SBEES	85.644	17.257	2.875	1.324	0.527	0.314	0.148	0.118
LSH-SBEES	3065.790	2125.170	586.660	35.178	20.531	2.127	2.980	0.381
LSHZC-SBEES	55.656	5.723	1.230	0.165	0.136	0.055	0.059	0.113
SH-MBEES	691.121	698.419	150.290	221.746	85.964	135.536	48.338	180.063
LSH-MBEES	14823.000	29605.400	13876.500	5668.350	3949.980	3035.400	3008.120	8978.240
LSHZC-MBEES	538.413	401.179	96.911	86.607	21.601	12.317	2.705	1.556

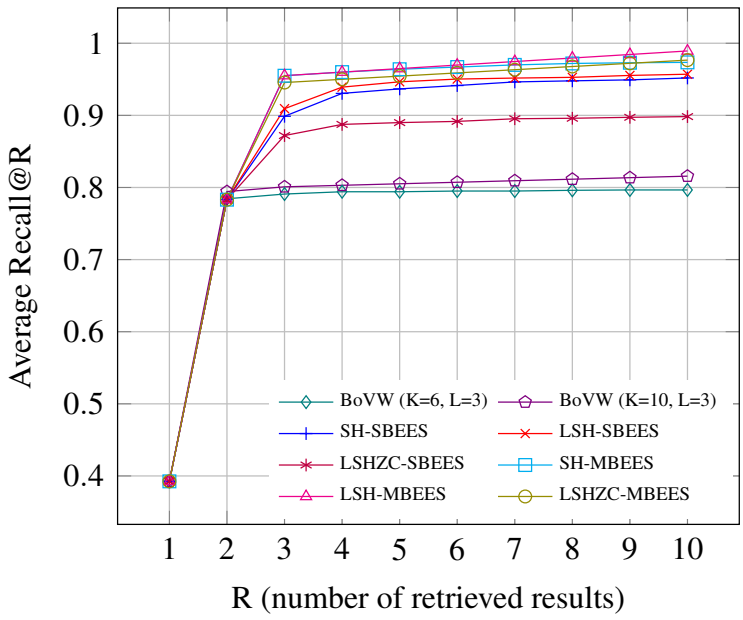

Fig. 4.6: Comparison of Recall@R (averaged over 500 queries) between SBEES, MBEES methods (using 24 bits) and BoVW over the Holidays dataset.

Table 4.4: Retrieval precision (mAP) and average query times (milliseconds) of the proposed methods compared to the BoVW method over the Holidays dataset.

Search methods	Retrieval precision (mAP)			Average query times (milliseconds)		
BoVW	(K=6, L=3)	(K=10, L=3)		(K=6, L=3)	(K=10, L=3)	
	20.5045	34.534		23.9736	62.345	
	(BCL=16)	(BCL=20)	(BCL=24)	(BCL=16)	(BCL=20)	(BCL=24)
SH-SBEES	31.840	37.095	35.515	1.766	0.484	0.201
LSH-SBEES	41.662	36.439	30.906	188.736	15.489	1.498
LSHZC-SBEES	35.364	33.422	29.233	0.906	0.167	0.077
SH-MBEES	38.209	41.973	42.308	79.178	22.177	33.793
LSH-MBEES	42.780	39.172	37.784	2609.070	538.968	339.850
LSHZC-MBEES	40.188	41.899	39.707	55.850	9.557	12.134

4.4 Re-ranking Results

Figure 4.7 shows a comparison of retrieval precision (UKB score) between MBEES before and after applying the reranking step. It shows that the reranking step adds some precision to all MBEES methods by about 10% with increasing in query time by around 50 millisec-

onds. The average query times of MBEES before and after applying the reranking step is shown in Table 4.5 and plotted for SH-MBEES and LSHZC-MBEES in Fig. 4.8.

Figure 4.9 shows the top four results of the first four distinct query images as found in the UKB dataset. Original LSH exceeded other original hashing methods in the first two queries due to its larger bins. Multi-Bin search improved the results of all queries except the third one. This is due to the complexity of the image's visual structure. The reranking step improved SH and LSHZC by retrieving another true positive result, i.e., another correct match.

Scalability Evaluation. Figure 4.10 reveals the scalability of the proposed methods applied on SH by showing the UKB scores of the proposed methods over various sizes of the MIRFLICKR-1M dataset merged with the UKB dataset. The labels on the data points represents the average query times in milliseconds. In this experiment, the number of descriptors per image was restricted to 50 descriptors/image, the most responsive descriptors were picked.

Figure 4.11 shows the speedup improvement introduced by the POPCNT instruction, mentioned previously, for the MBEES with SH and LSHZC. Speedup percentages are written over each binary code length. Of course this is not exactly the same speedup introduced only by POPCNT as there are many other computations that are not dependable on POPCNT.

Figure 4.12 shows the operation of the Web application which was built based on the implementation of the proposed methods. The Web application demonstrated the effectiveness of the proposed methods for real-time and real-world applications.

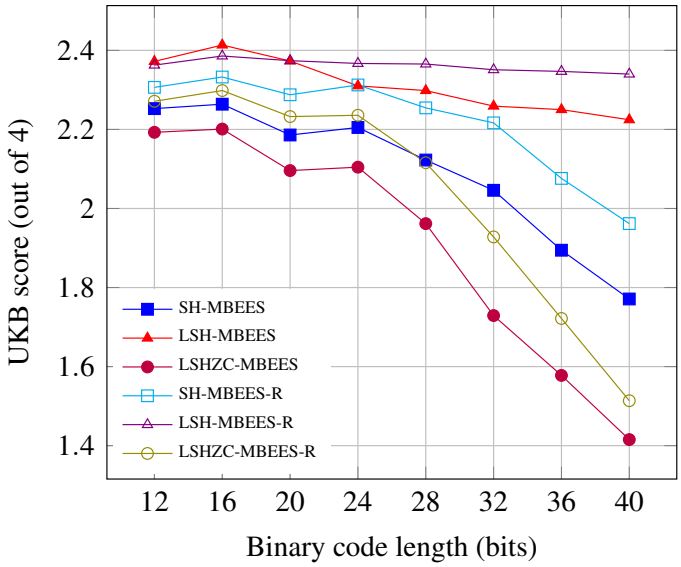

Fig. 4.7: Comparison of retrieval precision (UKB score) between MBEES before and after applying the re-ranking step.

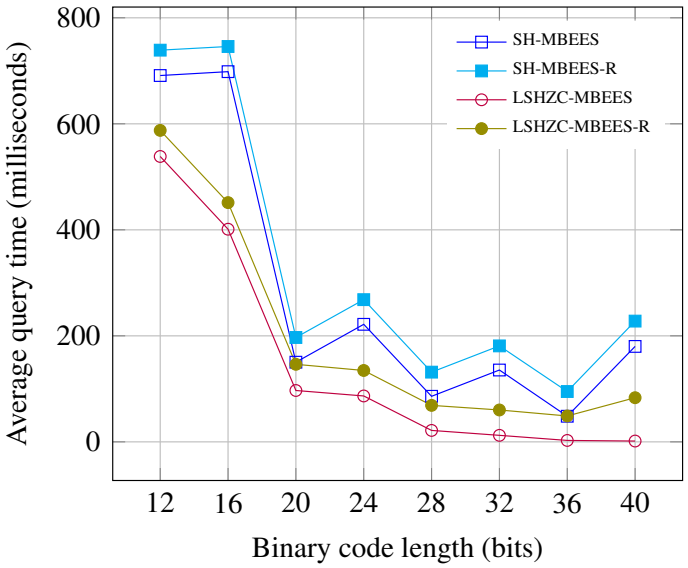

Fig. 4.8: Average query times before and after applying the reranking step on SH-MBEES and LSHZC-MBEES.

Table 4.5: Average query times of MBEES before and after applying the reranking step (milliseconds)

Search methods	Binary code length							
	12	16	20	24	28	32	36	40
SH-MBEES	691.121	698.419	150.290	221.746	85.964	135.536	48.338	180.063
LSH-MBEES	14823.000	29605.400	13876.500	5668.350	3949.980	3035.400	3008.120	8978.240
LSHZC-MBEES	538.413	401.179	96.911	86.607	21.601	12.317	2.705	1.556
SH-MBEES-R	738.969	745.868	196.933	268.293	131.5405	181.067	94.9854	227.83
LSH-MBEES-R	14873.660	29652.835	13924.115	5716.859	3999.084	3084.144	3294.375	9025.181
LSHZC-MBEES-R	587.607	451.363	146.307	134.680	68.847	60.160	48.852	83.299

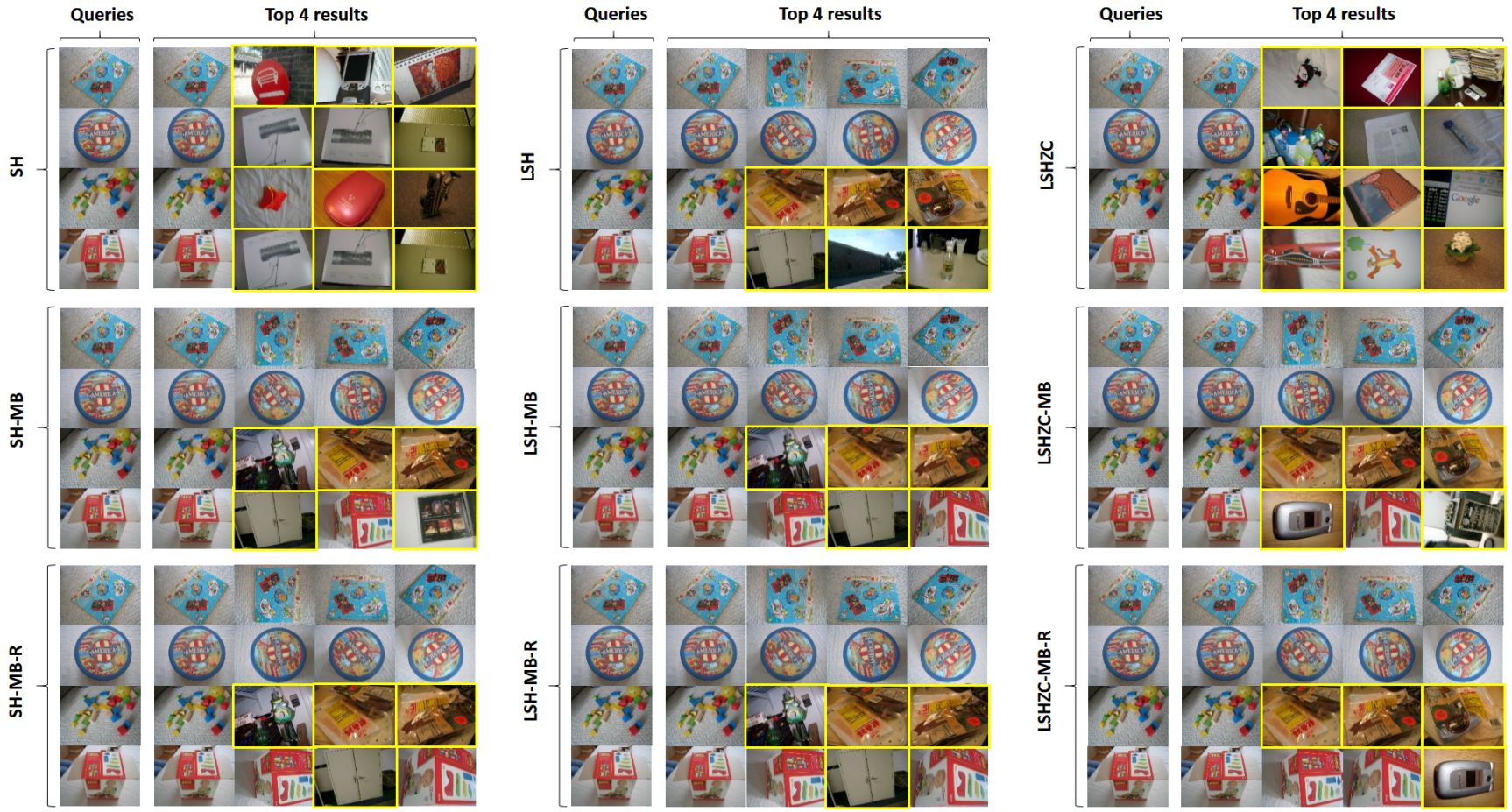

Fig. 4.9: Top four results of the first four distinct image queries as they appear in the UKB dataset.

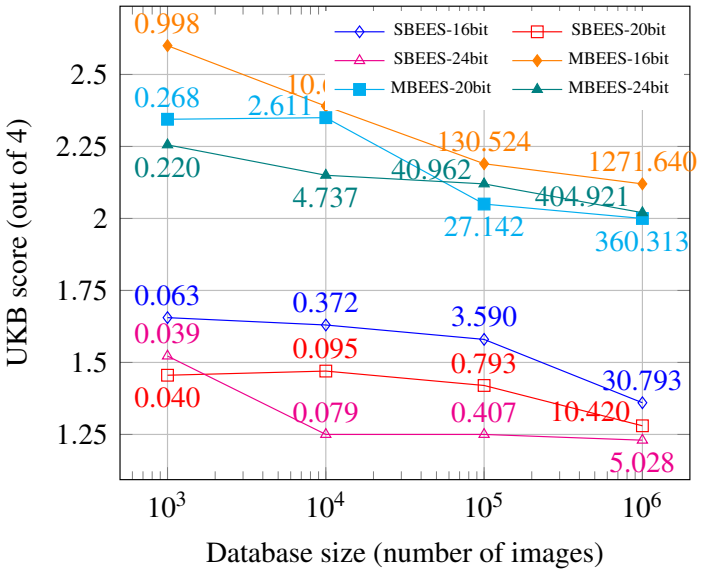

Fig. 4.10: UKB scores of the proposed methods applied on SH over various sizes of the MIRFLICKR-1M dataset merged with the UKB dataset. The labels on the data points represents the average query times in milliseconds.

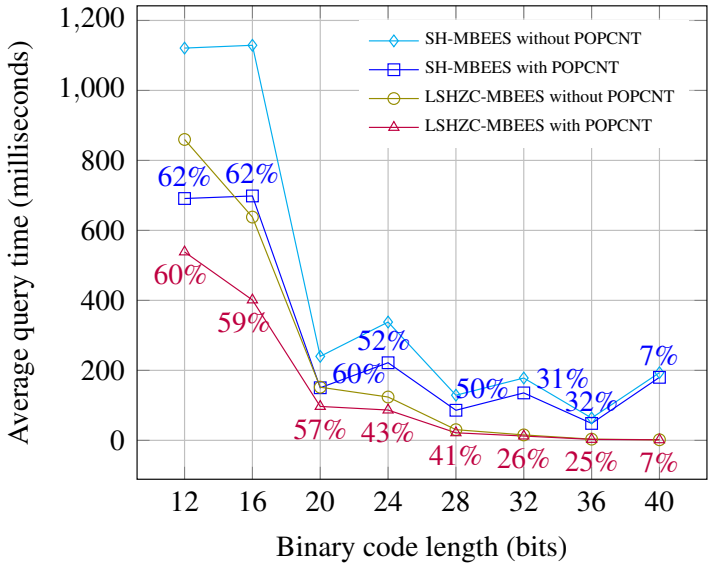

Fig. 4.11: Comparison between MBEES for SH-MBEES and LSHZC-MBEES with and without the instruction POPCNT. The labels on the lines represents speed up.

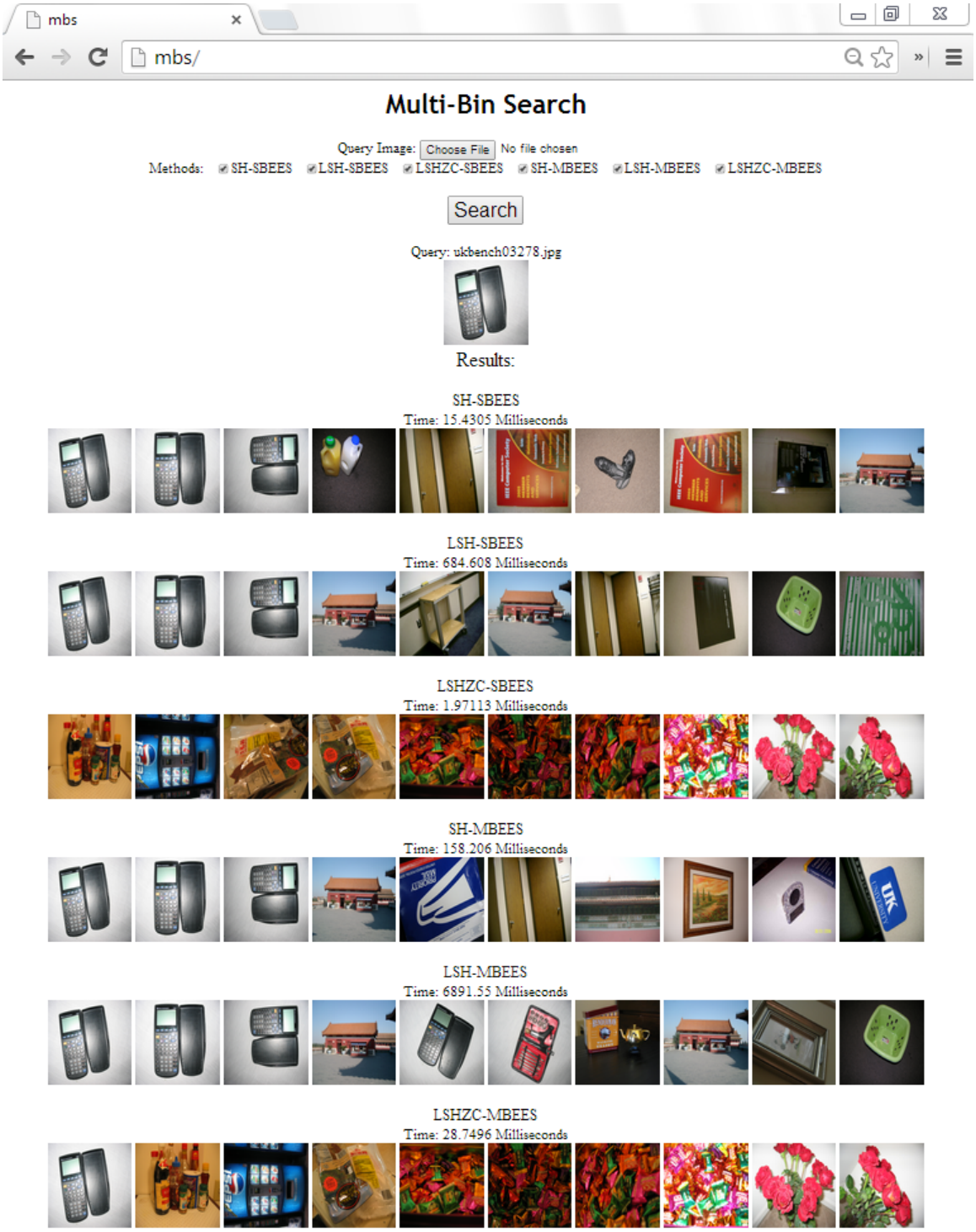

Fig. 4.12: Web application based on the implementation of the proposed methods.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions

In this work, a large-scale image retrieval method has been proposed which is based on binary hashing methods and binary local image descriptors. In this method, not only a target bin or bucket is searched, but also the nearest neighbor bins to a target bin are searched. The nearest neighbor bins are precomputed, then an exhaustive-search equivalent algorithm is used to examine those bins. The proposed approach can be applied to any hashing or clustering method in order to increase precision. The proposed approach has been applied on some hashing methods like Spherical Hashing and LSH, the evaluations showed significant improvement in retrieval precision over the original hashing methods.

In addition to comparisons with binary hashing methods, other comparisons with methods that are not depending on either binary hashing or binary descriptors have been carried out. These comparisons further evaluate and approve the effectiveness of the proposed approach.

To experiment the proposed approach in a real-world applications, an image search Web application was built based on the implementation of proposed methods. The operation of the Web application demonstrated the effectiveness of the proposed approach for real-world applications.

5.2 Future Works

Future work depending on the proposed approach can go on according to many aspects including the following:

- It would be convenient to evaluate the proposed approach on many other hashing and clustering methods.
- Some better exhaustive-search equivalent algorithms can be used for examining target bins.
- Many new binary image descriptors worth evaluation using the proposed approach, examples of these descriptors are FREAK and ORB.
- For more scalability assurance, it would be convenient to evaluate the proposed approach using other large-scale database.

REFERENCES

[1] N. Vasconcelos, "From pixels to semantic spaces: Advances in content-based image retrieval," *Computer*, vol. 40, no. 7, pp. 20–26, 2007.

[2] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, *et al.*, "Query by image and video content: The qbic system," *Computer*, vol. 28, no. 9, pp. 23–32, 1995.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[5] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision (IJCV)*, vol. 42, no. 3, pp. 145–175, 2001.

[6] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *International Conference on Computer Vision (ICCV)*, pp. 2548–2555, IEEE, 2011.

[7] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European Conference on Computer Vision (ECCV)*, pp. 778–792, Springer, 2010.

[8] J. Cronje, "Bfrost: binary features from robust orientation segment tests accelerated on the gpu," in *Proceedings of the 22nd Annual Symposium of the Pattern Recognition Society of South Africa*, pp. 25–30, 2011.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *International Conference on Computer Vision (ICCV)*, pp. 2564–2571, IEEE, 2011.

[10] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 510–517, IEEE, 2012.

[11] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *International Conference on Computer Vision (ICCV)*, pp. 1470–1477, IEEE, 2003.

[12] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, pp. 2169–2178, IEEE, 2006.

[13] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 2161–2168, 2006.

[14] H. Jégou, F. Perronnin, M. Douze, C. Schmid, *et al.*, "Aggregating local image descriptors into compact codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 34, no. 9, pp. 1704–1716, 2012.

[15] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3304–3311, IEEE, 2010.

[16] C. M. Bishop, *Pattern recognition and machine learning (information science and statistics)*. Springer, 2007.

[17] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on computational geometry*, pp. 253–262, ACM, 2004.

[18] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *The Neural Information Processing Systems Foundation (NIPS)*, pp. 1753–1760, NIPS, 2008.

[19] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2957–2964, IEEE, 2012.

[20] Z. Wu, Q. Ke, M. Isard, and J. Sun, "Bundling features for large scale partial-duplicate web image search," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 25–32, IEEE, 2009.

[21] L. Yu, J. Liu, and C. Xu, "Descriptive local feature groups for image classification," in *International Conference on Image Processing (ICIP)*, pp. 2501–2504, IEEE, 2011.

[22] T. Botterill, S. Mills, and R. Green, "Speeded-up bag-of-words algorithm for robot localisation through scene recognition," in *23rd International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pp. 1–6, IEEE, 2008.

[23] K. H. Hwang, H. Lee, and D. Choi, "Medical image retrieval: Past and present," *Healthcare Informatics Research*, vol. 18, no. 1, pp. 3–9, 2012.

[24] V. N. Gudivada and V. V. Raghavan, "Content based image retrieval systems," *Computer*, vol. 28, no. 9, pp. 18–22, 1995.

[25] http://images.google.com/ (last accessed: January 2014).

[26] http://www.tineye.com/ (last accessed: January 2014).

[27] M. Yousef and K. F. Hussain, "Fast exhaustive-search equivalent pattern matching through norm ordering," *Journal of Visual Communication and Image Representation*, vol. 24, no. 5, pp. 592–601, 2013.

[28] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision (ECCV)*, pp. 430–443, Springer, 2006.

[29] E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," in *European Conference on Computer Vision (ECCV)*, pp. 183–196, Springer, 2010.

[30] Intel, "Intel 64 and ia-32 architectures software developer's manual, volume 2b, instruction set reference, n–z," 2013.

[31] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors [lecture notes]," *Signal Processing Magazine, IEEE*, vol. 25, no. 2, pp. 128–131, 2008.

[32] J. He, R. Radhakrishnan, S.-F. Chang, and C. Bauer, "Compact hashing with joint optimization of search accuracy and time," in *International Conferenceon Computer Vision and Pattern Recognition (CVPR)*, pp. 753–760, IEEE, 2011.

[33] A. Joly and O. Buisson, "Random maximum margin hashing," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 873–880, IEEE, 2011.

[34] L. Liu, X. Shen, and X. Zou, "An improved fast encoding algorithm for vector quantization," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 1, pp. 81–87, 2004.

[35] Y. Hel-Or and H. Hel-Or, "Real-time pattern matching using projection kernels," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 27, no. 9, pp. 1430–1445, 2005.

[36] M. Gharavi-Alkhansari, "A fast globally optimal algorithm for template matching using low-resolution pruning," *IEEE Transactions on Image Processing*, vol. 10, no. 4, pp. 526–533, 2001.

[37] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *European Conference on Computer Vision (ECCV)*, pp. 304–317, Springer, 2008.

[38] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, IEEE, 2007.

[39] M. J. Huiskes and M. S. Lew, "The mir flickr retrieval evaluation," in *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*, (New York, NY, USA), ACM, 2008.

[40] D. Galvez-Lopez and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, pp. 1188–1197, October 2012.

PUBLICATIONS

- A. Kamel, Y. B. Mahdi, and K. F. Hussain, "Multi-Bin Search: Improved Large-Scale Content-Based Image Retrieval," *International Conference on Image Processing (ICIP)*, pp. 2597-2601, IEEE, 2013
- A. Kamel, Y. B. Mahdi, and K. F. Hussain, "Multi-Bin Exhaustive-Equivalent Search: Improved Large-Scale Content-Based Image Retrieval," *IEEE Transaction on Multimedia*, Under Review.

الملخص العربى

عنوان الرسالة: الاسْتِرْجَاعُ الْآنِيُّ لِلصُّوَرِ طِبْقًا لِلْمُحْتَوَى عَلَى نِطَاقٍ وَاسِعٍ

استرجاع الصور بناء على المحتوى على نطاق واسع يمثل تحدياً تمت مواجهته فى الآونة الأخيرة بالعديد من الطرق الواعدة. في هذه الرسالة، يتم تقديم طريقة جديدة والتى تحسّن بشكل مشترك دقة البحث والوقت المستغرق لاسترجاع الصور بناء على المحتوى على نطاق واسع. ويتحقق هذا عن طريق استخدام واصفات الصور المحلية الثنائية، مثل BRIEF أوBRISK، واستخدام طرق التجزئة الثنائية، مثل التجزئة الحساسة للمكان (LSH) والتجزئة الكروية.

الطريقة المقترحة، والمسماة بالبحث متعدد الحاويات (Multi-Bin Search)، تحسن دقة الاسترجاع لطرق التجزئة الثنائية. ويتم هذا التحسين من خلال حساب وتخزين وفهرسة أقرب الحاويات المجاورة لكل الحاويات المتولدة من طرق التجزئة الثنائية. وفى عملية البحث لا يتم البحث فقط فى الحاوية المستهدفة، ولكن يتم البحث أيضا فى أقرب الحاويات المجاورة.

من أجل البحث بكفاءة داخل الحاويات المستهدفة، تم استخدام خوارزمية سريعة مكافئة للبحث الشامل. وتستند هذا الخوارزمية على مطابقة المعايير المرتبة (NOM) والتى ثبت مؤخراً أنها تسفر عن نتائج مطابقة أو قريبة جداً من نتائج البحث الشامل مع تسريع كبير. أيضا، يتم تقديم خطوة إضافية لإعادة ترتيب النتائج والتى تزيد من دقة الاسترجاع ، ولكن مع زيادة طفيفة في وقت البحث.

تظهر التقييمات التجريبية أن الطرق المقترحة تحسن كثيرا من دقة استرجاع الصور لطرق التجزئة الثنائية الحديثة. أيضا، تم إجراء مقارنات مع بعض الطرق الحديثة لتقييم فعالية الطرق المقترحة. الطرق التى تمت مقارنتها لا تعتمد على أي من التجزئة الثنائية أو واصفات الصور الثنائية. بالإضافة إلى ذلك، و من أجل تجربة الطرق المقترحة في التطبيقات

الواقعية، تم بناء تطبيق ويب للبحث فى الصور على أساس الطرق المقترحة. وقد أظهر تشغيل التطبيق فعالية الطرق المقترحة في التطبيقات الواقعية وتلك التى تتطلب بحثا آنيا.

تتضمن هذه الرسالة خمسة فصول على النحو التالي:

الفصل الأول يعطى مقدمة عن مجال استرجاع الصور بناء على المحتوى وبعض تقنياته وتطبيقاته.

الفصل الثانى يعطي استعراضا مفصلا عن بعض الطرق والخوارزميات الحديثة التي تم اقتراحها في مجال استرجاع الصور بناء على المحتوى، ويركز على الأساليب والخوارزميات المستخدمة فى بناء الطريقة المقترحة فى هذه الرسالة.

الفصل الثالث يقدم شرحا مفصلا للطريقة المقترحة فى هذه الرسالة، والمسماة بالبحث متعدد الحاويات (Multi-Bin Search)، ويناقش مزاياه ونقاط ضعفه.

الفصل الرابع يقدم تقييمات تفصيلية ومقارنات بين الطريقة المقترحة و بعض الطرق الأخرى الحديثة.

الفصل الخامس يلخص العمل المنجز فى هذه الرسالة والأعمال المقترح إضافتها مستقبلا استنادا إلى هذا العمل.

الاسْتِرْجَاعُ الْآنِيُّ لِلصُّوَرِ طِبْقًا لِلْمُحْتَوَى عَلَى نِطَاقٍ وَاسِعٍ

عَبْد الرّحْمَنِ كَامِل صِدِّيق عَبْد الْحَمِيدِ

بَكَالُورْيُوسُ الْحَاسِبَاتِ وَالمعْلُومَاتِ (عُلُومُ الْحَاسِبِ) – جَامِعَةُ أَسْيُوط ٢٠٠٩

رِسَالَةٌ مِقَدَّمَةٌ إِلَى

قِسْمِ عُلُومِ الْحَاسِبِ – كُلِّيَةِ الْحَاسِبَاتِ وَالمعْلُومَاتِ – جَامِعَةِ أَسْيُوط

لِاسْتِيفَاءِ مُتَطَلَّبَاتِ الْحُصُولِ عَلَى دَرَجَةِ

الْمَاجِسْتِير فِى الْحَاسِبَاتِ وَالمعْلُومَاتِ (عُلُوم الْحَاسِبِ)

لَجْنَةُ الْإِشْرَافِ

أ.د. يُوسُفُ بَسْيُونِى مَهْدِى

د. خَالِدُ فَتْحِى حُسَيْن

لَجْنَةُ الْحُكْمِ والمنَاقَشَةِ

أ.د. أُسَامَة سَيِّـد مُحَمَّـد

أ.د. مَرْغَنِى حَسَن مُحَمَّـد

أ.د. يُوسُفُ بَسْيُونِى مَهْدِى

د. خَالِدُ فَتْحِى حُسَيْن

قِسْم عُلُومِ الْحَاسِبِ – كُلِّيَة الْحَاسِبَاتِ وَالمعْلُومَاتِ

جَامِعَة أَسْيُوط – أَسْيُوط – مِصْر

٢٠١٤