UNIVERSITÀ
DEGLI STUDI
FIRENZE

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione (DINFO)
Corso di Dottorato in Ingegneria dell'Informazione
Curriculum: Informatica (ssd ing-inf/05)

# Compact Hash Codes and Data Structures for Visual Descriptors Retrieval

*Candidate*
Simone Ercoli

*Supervisors*
Prof. Alberto Del Bimbo

Prof. Marco Bertini

*PhD Coordinator*
Prof. Luigi Chisci

ciclo XXIX, 2013-2016

Università degli Studi di Firenze, Dipartimento di Ingegneria
dell'Informazione (DINFO).

*"Memento audere semper"*

*A tutti quelli che hanno creduto in me*

# Acknowledgments

This thesis is the result of my work at Media Integration and Communication Center (MICC). I am most grateful to my supervisors Prof. Alberto Del Bimbo and Prof. Marco Bertini who have always supported my research and gave me the opportunity to work on challenging tasks. In addition, I would like to thank all my colleagues for the beautiful moments, the fruitful discussions and support given to me during these years. At the end I would like to thank my family who has always believed in me and I will always be grateful to my girlfriend Natascia for her love and understanding.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

We live in the digital age, burdened by information overload. Digital images play an important role in our everyday life. In many areas of science, commerce and government images are daily acquired and used. During the past decades we have been observing a permanent increase in image data, leading to huge repositories. This led to an increasing percentage of information in other mediums, for example in the form of audio, video and images, not to mention raw data which is created as a result of various industrial and scientific applications.
Naturally, such a massive amount of data would be close to being useless unless there are efficient ways to access it. This rapid evolution triggers the demand of qualitative and quantitative image retrieval systems. It can thus be said that we need efficient information retrieval systems.
Content-based image retrieval (CBIR) methods have tried to alleviate the access to image data. The broad range of image retrieval and classification applications demands a sort of generalization as well as highly specialized systems equipped with image features such as for example color, texture or structure-based ones. Content based image retrieval has been also an active research topic in computer vision and multimedia in the last decades, and it is still very relevant due to the emergence of social networks and the creation of web-scale image databases.
Efficient nearest neighbor (NN) search is one of the main issues in large scale information retrieval for multimedia and computer vision tasks. Computin Euclidean distances between high dimensional vectors is a fundamental requirement in many applications. Focusing on the $D$-dimensional Eu-

clidean space $\mathbb{R}^D$ of $n$ vectors minimizing the distance to the query vector $x \in \mathbb{R}^D$. Several multi-dimensional indexing methods, such as the popular KD-tree [26] or other branch and bound techniques, have been proposed to reduce the search time. However, for high dimensions it turns out that such approaches are not more efficient than the brute-force exhaustive distance calculation, whose complexity is $O(nD)$. There is a large body of literature [18], [31], [74] on algorithms that overcome this issue by performing approximate nearest neighbor (ANN) search. The key idea shared by these algorithms is to find the NN with high probability "only", instead of probability 1. Most of the effort has been devoted to the Euclidean distance, though recent generalizations have been proposed for other metrics [59]. One of the most popular ANN algorithms is the Euclidean Locality-Sensitive Hashing (E2LSH) [18], [88], which provides theoretical guarantees on the search quality with limited assumptions. It has been successfully used for local descriptors [52] and 3D object indexing [73], [88]. However, for real data, LSH is outperformed by heuristic methods, which exploit the distribution of the vectors. These methods include randomized KD-trees [89] and hierarchical k-means [75], both of which are implemented in the FLANN selection algorithm [74].

ANN algorithms are typically compared based on the tradeoff between search quality and efficiency. However, this tradeoff does not take into account the memory requirements of the indexing structure. In the case of E2LSH, the memory usage may even be higher than that of the original vectors. Moreover, both E2LSH and FLANN need to perform a final re-ranking step based on exact L2 distances, which requires the indexed vectors to be stored in main memory if access speed is important. This constraint seriously limits the number of vectors that can be handled by these algorithms. Only recently, researchers came up with methods limiting the memory usage. This is a key criterion for problems involving large amounts of data [98], i.e., in large-scale scene recognition [99], where millions to billions of images have to be indexed.

When dealing with high dimensional features also methods for multidimensional indexing obtain performance comparable to that of exhaustive search [47]. A typical solution is to employ methods that perform approximate nearest neighbor (ANN) search, typically using feature hashing and Hamming distances. Most of the works have addressed the development of effective visual features, from engineered features like SIFT and GIST to, more re-

cently, learned features such as CNNs [5]. To obtain scalable CBIR systems features are typically compressed or hashed, to reduce their dimensionality and size.
However, research on data structures that can efficiently index these descriptors has attracted less attention, typically simple inverted files (e.g. implemented as hash tables) are used, and require to use hash codes with a length of several tens of bits to obtain a reasonable performance in retrieval. This combination requires quite large amounts of memory to store large scale databases of features, thus their application to systems with relatively limited memory (e.g. most of the mobile devices still have 1-2 GB RAM only) or systems that involve a large-scale media analysis is not feasible.

## 1.1 Structure of the Document

This thesis is structured as follows. We begin by introducing a brief survey of related work on feature hashing for retrieval using local and global visual features illustrating all related methods (Chapter 2). In Chapter 3 we describe the standard components of an image retrieval system, including common performance measures for evaluation purposes. In Chapter 4 we introduce and describe a new approach for vector quantization based on kmeans which allows the possibility of assignment of a visual feature to multiple cluster centers during the quantization process. We show the goodness of this new approach by presenting an exhaustive comparison with all the methods presented in Chapter 2. We also introduce the usage of an efficient and recursive data structure to store datas. Finally, in the Chapter 5, we introduce a new approach for efficient image retrieval based on the m-k-means hashing introduced in Chapter 4 and we apply our method on CNN features. To conclude we introduce the usage of an efficient indexing structure based on Bloom filters and we show how the experimental validation outperforms the state-of-the-art hashing methods in terms of precision.

## 1.2 Contributions

The aim of this thesis is the investigation of several methods for visual feature hashing and multidimensional indexing related to the image and, more in general, large scale information retrieval problems. Moreover, we present a novel method for feature hashing, based on multiple k-means assignments,

that is unsupervised, that requires a very limited codebook size and that obtains a very good performance in retrieval, for local and global visual features, either engineered or learned, even with very compact hash codes. The proposed approach greatly reduces the need of training data and memory requirements for the quantizer, and obtains a retrieval performance similar or superior to more complex state-of-the-art approaches on standard large scale datasets. This makes it suitable, in terms of computational cost, for mobile devices, large-scale media analysis and content-based image retrieval in general.

We also propose a novel variation of this effective hashing method for CNN descriptors. To perform an immediate rejection of a search that should not return any result we store the hash code in a Bloom filter, i.e. a space efficient probabilistic data structure that is used to test the presence of an element in a set. To the best of our knowledge this is the first time that this data structure has been proposed for image retrieval since, natively, it has no facility to handle approximate queries. We perform extensive experimental validation on three standard datasets, showing how the proposed hashing method improves over state-of-the-art methods, and how the data structure greatly improves computational cost and makes the system suitable for application to mobile devices and distributed image databases.

# Chapter 2

# Literature Review

*This chapter gives a brief survey of related works on features hashing for retrieval using local and global visual features. Previous works on visual features hashing and retrieval can be divided in methods based on hashing functions, scalar quantization, vector quantization and, more recently, neural networks.*
*These methods typically rely on the use of inverted files to store the hash codes and to perform retrieval. A few works have also addressed specifically the study of efficient data structures for nearest neighbor retrieval.*

## 2.1 Visual Features

SIFT [70] descriptors have been successfully used for many years to perform CBIR [1]. Features have been aggregated using Bag-of-Visual-Words and, with improved performance, using VLAD [45] and Fisher Vectors [87].
The recent success of CNNs for image classification tasks has suggested their use also for image retrieval tasks. Babenko *et al.* [5] have proposed the use of different layers of CNNs as features, compressing them with PCA to reduce their dimensionality, and obtaining results comparable with state-of-the-art approaches based on SIFT and Fisher Vectors. Aggregation of local CNN features using VLAD has been proposed in [106], while Fisher Vectors computed on CNN features of objectness window proposals have been used in [101].

[1] Content-based image retrieval (CBIR) are systems which allow, given a query, to find a subset of similar elements within a database. In this case an element can be an image

## 2.2 Hashing Functions

Weiss *et al.* [104] have proposed to treat the problem of hashing as a particular form of graph partitioning, in their Spectral Hashing (SH) algorithm. Li *et al.* [64] have improved the application of SH to image retrieval optimizing the graph Laplacian that is built based on pairwise similarities of images during the hash function learning process, without requiring to learn a distance metric in a separate step. Heo *et al.* [39] have proposed to encode high-dimensional data points using hyperspheres instead of hyperplanes; Jin *et al.* [50] have proposed a variation of LSH, called Density Sensitive Hashing, that does not use random projections but instead uses projective functions that are more suitable for the distribution of the data.

Du *et al.* [21] have proposed the use of Random Forests to perform linear projections, along with a metric that is not based on Hamming distance.

Lv *et al.* [72] address the problem of large scale image retrieval learning two hashes in their Asymmetric Cyclical Hashing (ACH) method: a short one ($k$ bits) for the images of the database and a longer one ($mk$ bits) for the query, computed using similarity preserving Random Fourier Features, and computing the Hamming distance between the long query hash and the cyclically $m$-times concatenated compact hash code of the stored image.

Paulevé *et al.* [80] have compared structured quantization algorithms with unstructured quantizers (i.e. k-means and hierarchical k-means clustering). Experimental results on SIFT descriptors have shown that unstructured quantizers provide significantly superior performances with respect to structured quantizers.

## 2.3 Scalar Quantization

Zhou *et al.* [109] have proposed an approach based on scalar quantization of SIFT descriptors. The median and the third quartile of the bins of each descriptor are computed and used as thresholds, hashing is then computed coding the value of each bin of the descriptor with 2 bits, depending on this subdivision. The final hash code has a dimension of 256 bits, but only the first 32 bits are used to index the code in an inverted file. The method of [109] has been extended by Ren *et al.* [84], including an evaluation of the reliability of bits, depending on their quantization errors. Unreliable bits are then flipped when performing search, as a form of query expansion. To

avoid using codebooks, in the context of memory limited devices such as mobile phones, Zhou *et al.* [108] have proposed the use of scalable cascaded hashing (SCH), performing sequentially scalar quantization on the principal components, obtained using PCA, of SIFT descriptors. Chen and Hsieh [12] have recently proposed an approach that quantizes the differences of the bins of the SIFT descriptor, using the median computed on all the SIFT descriptors of a training set as a threshold.

## 2.4 Vector Quantization

Jégou *et al.* [47] have proposed to decompose the feature space into a Cartesian product of subspaces with lower dimensionality, that are quantized separately. This Product Quantization (PQ) method is efficient in solving memory issues that arise when using vector quantization methods such as k-means, since it requires a much reduced number of centroids. The method has obtained state-of-the-art results on a large scale SIFT features dataset, improving over methods such as SH [104] and Hamming Embedding [44]. This result is confirmed in the work of Chandrasekhar *et al.* [10], that have compared several compression schemes for SIFT features.

The success of the Product Quantization method has led to development of several variations and improvements. The idea of compositionality of the PQ approach has been further analyzed by Norouzi and Fleet [76], that have built upon it proposing two variations of k-means: Orthogonal k-means and Cartesian k-means (ck-means). Also Ge *et al.* [28] have proposed another improvement of PQ, called OPQ, that minimizes quantization distortions w.r.t. space decomposition and quantization codebooks; He *et al.* [38] have proposed an affinity-preserving technique to approximate the Euclidean distance between codewords in k-means method. Kalantidis and Avrithis [51] have presented a simple vector quantizer (LOPQ) which uses a local optimization over a rotation and a space decomposition and apply a parametric solution that assumes a normal distribution. More recently, Guo *et al.* [34] have improved over OPQ and LOPQ adding two quantization distortion properties of the Residual Vector Quantization (RVQ) model, that tries to restore quantization distortion errors instead of reducing it.

Babenko and Lempitsky [3] have proposed an efficient similarity search method, called inverted multi-index (IMI); this approach generalizes the inverted index by replacing vector quantization inside inverted indices with

product quantization, and building the multi-index as a multi-dimensional table (Multi-D-ADC). More recently, Babenko and Lempitsky [4] have addressed the problem of indexing CNN features, observing that IMI is inefficient to index such features, and proposing two extensions of IMI: the Non-Orthogonal Inverted Multi-Index (NO-IMI) and the Generalized Non-Orthogonal Inverted Multi-Index (GNO-IMI).

## 2.5 Neural Network

Lin *et al.* [67] have proposed a deep learning framework to create hash-like binary codes for fast image retrieval. Hash codes are learned in a point-wise manner by employing a hidden layer for representing the latent concepts that dominate the class labels (when the data labels are available). This layer learns specific image representations and a set of hash-like functions.

Do *et al.* [20] have addressed the problem of learning binary hash codes for large scale image search using a deep model which tries to preserve similarity, balance and independence of images. Two sub-optimizations during the learning process allow to efficiently solve binary constraints.

Guo and Li [35] have proposed a method to obtain the binary hash code of a given image using binarization of the CNN outputs of a certain fully connected layer.

Zhang *et al.* [107] have proposed a very deep neural network (DNN) model for supervised learning of hash codes (VDSH). They use a training algorithm inspired by alternating direction method of multipliers (ADMM) [8]. The method decomposes the training process into independent layer-wise local updates through auxiliary variables.

Xia *et al.* [105] have proposed an hashing method for image retrieval which simultaneously learns a representation of images and a set of hash functions.

A deep learning framework for hashing of multimodal data has been proposed by Wang *et al.* [103], using a multimodal Deep Belief Network to capture correlation in high-level space during pre-training, followed by learning a cross-modal autoencoder in fine tuning phase.

Lin *et al.* [66] have proposed to use unsupervised two steps hashing of CNN features. In the first step Stacked Restricted Boltzmann Machines learn binary embedding functions, then fine tuning is performed to retain the metric properties of the original feature space.

## 2.6 Data Structures

Babenko and Lempitsky [3] have proposed a data structure for efficient similarity search, called inverted multi-index, that generalizes the inverted index by replacing vector quantization inside inverted indices with product quantization, and building the multi-index as a multi-dimensional table. An efficient algorithm to produce an ordered sequence of multi-index entries for a query is also proposed. Very recently Norouzi *et al.* [77] have proposed a method to build multiple hashing tables for exact k-nearest neighbor search of hash codes, testing the method on a large scale SIFT dataset.

### 2.6.1 Bloom Filter

Bloom filter and its many variants have received an extremely limited attention from the vision and multimedia community, so far. Inoue and Kise [42] have used Bloomier filters (i.e. an associative array of Bloom filters) to store PCA-SIFT features of an objects dataset more efficiently than using an hash table; they perform object recognition by counting how many features stored in the filters are associated with an object. Bloom filter has been used by Danielsson [17] as feature descriptor for matching keypoints. Similarity of descriptors is evaluated using the "union" operator. Srijan and Jawahar have proposed to use Bloom filters to store compactly the descriptors of an image, and use the filter as postings of an inverted file index in [94].

# Chapter 3

# Content-Based Image Retrieval

*In this chapter we present an overview of the content-based image retrieval (CBIR) problem. Also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR) is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large database.*

## 3.1 Introduction

During the past decades we have been observing a permanent increase in image data, leading to huge repositories. Content-based image retrieval methods have tried to alleviate the access to image data. To date, numerous feature extraction methods have been proposed in order to improve the quality of content-based image retrieval and image classification systems.
Let there exist a database $D$ consisting of digital images. A user intends to access the database, but linear browsing is not practical since the databases are usually large. So how should this be performed? This is the core of the *image retrieval problem* and is depicted symbolically in Figure 3.1.
"Content-based" means that the search analyzes the contents of the image rather than the metadata such as keywords, tags, or descriptions associated with the image. The term "content" in this context might refer to colors, shapes, textures, or any other information that can be derived from the image itself. CBIR is desirable because searches that rely purely on metadata are dependent on annotation quality and completeness. Having

humans manually annotate images by entering keywords or metadata in a large database can be time consuming and may not capture the keywords desired to describe the image. The evaluation of the effectiveness of keyword image search is subjective and has not been well-defined. In the same regard, CBIR systems have similar challenges in defining success. In the end there is no unique best solution to this problem which holds for each and every database and for different retrieval scenarios.

The initial point of entry in a search engine is a *query*, which tells the system about what the user is looking for. Textual search engines, for example, can accept whole words or phrases as input query. A query in a text-based search engine may be seen as being a tiny document in itself. Indeed, most web search engines now include a *find similar web pages* extension, which can be viewed as a search request with a complete web page as the starting query. The most natural extension of a textual query model for use in image retrieval would be a *query-by-image-region* model. The image region should contain the object or objects which the user is looking for in an image. However, there are many practical differences compared to the corresponding textual query model. Unlike query text, which can be inputted from memory, a query image region cannot usually be constructed on the fly (an exception may exist for sketch retrieval, where the rough outline of the desired sketch may suffice). Another difference is that while words are trivially extracted from text documents (e.g. using whitespace information), image segmentation into meaningful regions is a very challenging problem whose solution still eludes us, save for some specific problem settings [16, 24]. Due to this, the most popular query mechanism in image retrieval literature has been the so-called *query-by-example* model, in which whole images are compared. Its main advantage is ease of validation; the quality of the results can be verified visually by viewing them alongside the query image. In a real system, however, finding an appropriate initial query image can be difficult. Indeed some could ask why to use of an image retrieval system if the user already possesses a similar looking image. While this criticism does hold for many image collections, there are scenarios where this is the preferred query model. An example could be in clinical use, where a doctor might want to retrieve the case history of patients with a similar radiographic image compared to that of the current patient.

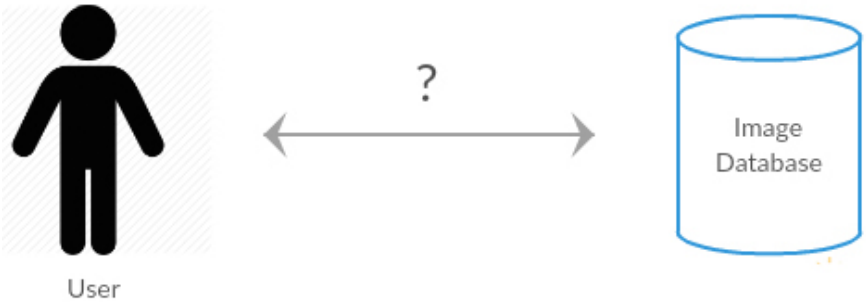



Figure 3.1: An image retrieval use case at its most abstract level

## 3.2 History

The term "content-based image retrieval" seems to have originated in 1992 when it was used by T. Kato to describe experiments into automatic retrieval of images from a database, based on the colors and shapes present [22].
Since then, the term has been used to describe the process of retrieving desired images from a large collection on the basis of syntactic image features. The techniques, tools, and algorithms that are used originate from fields such as statistics, pattern recognition, signal processing, and computer vision [63]. The earliest commercial CBIR system was developed by IBM and was called QBIC (Query by Image Content) [86]. Recent network and graph based approaches have presented a simple and attractive alternative to existing methods [6].

## 3.3 Technical Progress

The interest in CBIR has grown because of the limitations inherent to the metadata-based systems, as well as the large range of possible uses for efficient image retrieval. Textual information about images can be easily searched using existing technology, but this requires humans to manually describe each image in the database. This can be impractical for very large databases or for images that are generated automatically, e.g. those from surveillance cameras. It is also possible to miss images that use different synonyms in their descriptions. Systems based on categorizing images in semantic classes like "cat" as a subclass of "animal" can avoid the miscat-

egorization problem, but will require more effort by a user to find images that might be "cats", but are only classified as an "animal". Many standards have been developed to categorize images, but all still face scaling and miscategorization issues [22].
Initial CBIR systems were developed to search databases based on image color, texture, and shape properties. After these systems were developed, the need for user-friendly interfaces became apparent. Therefore, efforts in the CBIR field started to include human-centered design that tried to meet the needs of the user performing the search. This typically means inclusion of: query methods that may allow descriptive semantics, queries that may involve user feedback, systems that may include machine learning, and systems that may understand user satisfaction levels [63].

## 3.4 Components of a CBIR System

Figure 3.2 shows the building blocks of a typical content-based image retrieval system, and how they relate to each other. First of all, in an offline operation, meaningful features are extracted from all images in the database. This is a critical step and will be discussed in more detail in the next section. For computational efficiency, the features can be indexed, for example in a R-tree [36] or in a K-D-B tree [85].
The user starts the retrieval process by providing a query image. The features for the query image are extracted exactly in the same way as for the database images. The features of the query image are then compared with the features of the database images using some kind of a *similarity measure*. The database images which possess the highest similarity measure value are returned as the result images.
The reader can refer to [37] which contains a survey of the existing CBIR literature.

## 3.5 Features Extraction

Digital images possess a very high dimensionality. For example, an image of size $1024 \times 1024$ pixels has more than a million dimensions when interpreted as a vector. However, there is a huge amount of redundancy present in the images. Furthermore, it is often desirable to remove the information content which is not essential or even counter-productive for the particular

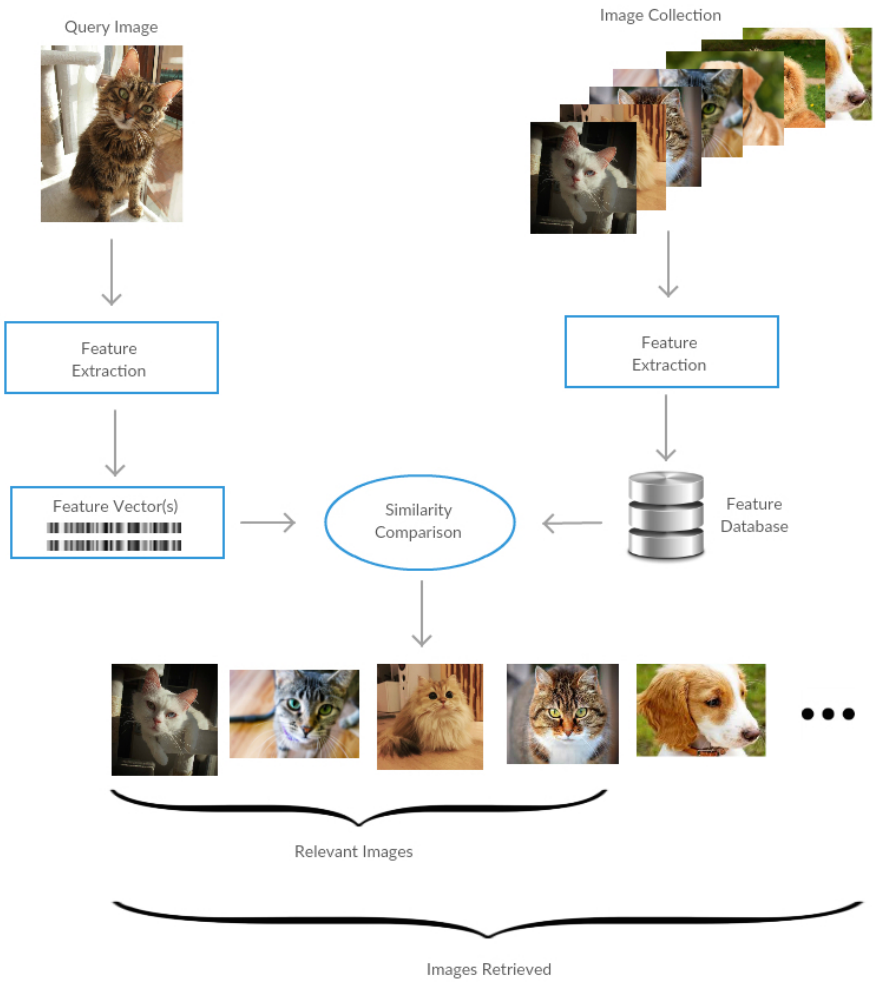



Figure 3.2: General scheme of content-based image retrieval

retrieval task at hand. This step of extracting meaningful information is termed as *features extraction*. The most critical properties to be analysed for any kind of features are its *information content* and *invariance* to certain transformations of the data. These properties will be discussed next.

### 3.5.1 Invariance

Given a transformation group $G$ with an element $g$ acting on the data $S$, a feature $F(S)$ is said to be invariant with respect to $G$ if

$$\mathbf{S}' = g\mathbf{S} \Rightarrow F(\mathbf{S}') = F(\mathbf{S}), \forall g \in G \tag{3.1}$$

This is the so-called necessary condition for invariance. For digital images, the transformation groups required in an application are typically one or more of the following:

- the group of translations
- the group of rotations
- the group of Euclidean motion
- the group of similarity transformations
- the group of affine transformations
- the group of monotonic intensity transformation

In the above list, all except the monotonic intensity transformation are what are known as *geometric transformations*. Only the location of the points is transformed, while the value remains unchanged. For example, given an initial source point $(p_x, p_y)^T$ , we have the destination point coordinates

$$\begin{pmatrix} p'_x \\ p'_y \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{3.2}$$

for the case of translation, and

$$\begin{pmatrix} p'_x \\ p'_y \end{pmatrix} = \begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \quad \begin{vmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{vmatrix} \neq 0 \tag{3.3}$$

for the general case of an affine transformation.
The number of independent tunable parameters in a transformation is known as the *degrees of freedom*, for example, two and six for the case of translation and affine transformations, respectively. In general, the variability within a class is exponentially proportional to the degrees of freedom in the transformation which the class is allowed to have.
There are in principle three ways with which one can obtain invariant features:

**Normalization** In this method one tries to find distinctive elements of a class and normalize other elements with respect to it. For example, an image of an object could be translated to have its center of gravity at a particular point, or rotated to have its major axis aligned with the horizontal axis of an image.

**Differentiation** The elements $g\mathbf{S}$ form an orbit in the feature space and can be controlled using a parameter vector $\lambda$ (whose dimensionality $f$ is equal to the degrees of freedom $G$). An invariant feature should take a constant value along the orbit, thus one tries to find features satisfying the differential equation

$$\frac{\delta g_\lambda \mathbf{S}}{\delta \lambda_i} = 0, \forall i = 1...f \tag{3.4}$$

**Integration** As early as in 1897, Hurwitz demonstrated the use of Haar Integrals for generating invariant features [41]. One integrates over the group elements which have been transformed using a (often non-linear) kernel function $f$.

$$F(\mathbf{S}) = \frac{1}{\mid G \mid} \int_G f(g\mathbf{S}) dg \tag{3.5}$$

### 3.5.2 Information Content

Invariance towards desired transformations alone does not guarantee good performance for a classification or retrieval task. It is also important that the features are discriminative for the classes in hand. As an extreme example, consider the feature described by the average gray value of the image, i.e. $f = 1/MN \sum_i \sum_j S(i,j)$. This feature is invariant against Euclidean motion of

objects present in the image, but unfortunately, it is invariant against many more transformations. For example, one may permute the pixel positions in an arbitrary manner without affecting the feature $f$ , which may have an undesirable behaviour. Mathematically, one can express this property in terms of what is known as the sufficient condition for invariance:

$$F(\mathbf{S}') = F(\mathbf{S}) \Rightarrow \mathbf{S}' = g\mathbf{S}, g \in G \tag{3.6}$$

Which means that if two patterns are mapped to the same point in the feature space, then they must belong to the same equivalence class. If both the necessary condition (Equation 3.1) as well as the sufficient condition are satisfied, we say that the invariant mapping is complete.

### Non-Vectorial Data

The scalar features extracted using the methods described in the previous section must be combined in some way. The simplest way to group them is to create a vector, with the scalar features being its members. This approach has the advantage that even unrelated features can be grouped together, as no special property about the features is assumed. The algorithms presented in this thesis are in general applicable to such vectorial features, as is indeed most work existing in the pattern recognition literature. However, other data representation formats such as strings, trees and graphs are getting popular, especially in domains such as text mining and bioinformatics. Even a segmented image could be thought of as an unordered set of its parts. Interactive retrieval is of interest in many bioinformatics applications, such as protein retrieval, where a vectorbased feature approach might not be the best choice.

### Non-Standard Similarity Measures

In some cases, it might be practicable to skip the feature extraction step, and instead build the complexity in the matching algorithm. For example, in the case of medical images, an algorithm might perform image registration and consider the quality of the registration as a similarity measure, or judge the similarity through the amount of deformation that must be carried out on one of the images [53].

### 3.5.3 Semantic Gap

Due to the fact that current feature extraction methods are not powerful enough to capture all kinds of subtle nuances present in natural images, we have what is known as the problem of the *semantic gap*. The following definition was provided in [93], which is arguably the most prevalent review paper on the field of content based image retrieval:

> *"The semantic gap is the lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given situation."*

However, different researchers have taken somewhat differing interpretations, which leads to some confusion. Especially, some authors chose to use the word *semantic* as in *semantic labelling* or *semantic classification* when clearly the terms *visual labelling* and *visual classification* would have been closer to the intended meaning. The primary difference is that in variability. An image would be a candidate for *semantic labelling* by a keyword if a human being could associate the keyword with the image either directly or indirectly. That is to say, on the basis of semantics. On the other hand, *visual labelling* merely postulates that the images possess a visual (low-level) content similarity. Figures 3.4 and 3.5 give examples of a possible label cars in its visual and semantic connotations. An example of the semantic gap problem is shown in Figure 3.3.
The semantic gap exists because the features or information that we can currently extract from the pixel content of digital images are not rich or expressive enough to capture higher-level information which on the other hand is readily performed by human beings. As an example, consider a face detection engine, which can detect, with a high enough precision (say 90% ), human faces in a digital image. The high-level information missing for a particular application could be, for example, the general *mood* present in the image. We wish to understand the hurdles in learning complex such high-level concepts from image data. There can be two fundamental approaches.

One is to train a *concept detector* for every high level concept such as *mood* directly from image data. This approach seems however, unlikely to work, as the number of possible high-level concepts can be very high, with possibly much higher intraconcept variability as compared to low-level tasks. The other approach is to have simple concept detectors and then use external

Figure 3.3: An example of the semantic gap problem. The two images possess very similar colour and texture characteristics, but differ vastly as far as the semantics are concerned.

rules to learn high-level concepts. These rules could be combinational based on certain observations and could also be *learned* automatically to a certain extent. Let us give an example. Assume that we can predict with medium precision (say 70%) the following information from the content of an image:

(a) the nationality of the people present

(b) their location estimate

(c) condition of the people's clothes

Figure 3.4: Sample images from a hypothetical *visual* class "cars"

Then rules such as the following might lead us to the desired result:

$$\forall x,$$
$$((\text{nationality}(x) = \text{`english'}) \wedge (\text{location}(x) = \text{`green\_grounds'}) \wedge (\text{clothes} = \text{`muddy'})) \Rightarrow \text{playing\_rugby}(x) \quad (\mathbf{R1})$$

and

$$\forall x,$$
$$\text{playing\_rugby}(x) \Rightarrow \text{mood}(x) = \text{`playful'} \quad (\mathbf{R2})$$

Although probabilistic logic would be more natural, the above example uses first-order predicate logic for simplicity.


Figure 3.5: Sample images from a hypothetical *semantic* class "cars"

The rule **R2** is simply dependent on the rule **R1**, however the rule **R1** is dependent on three sub-rules. Assuming that the detectors for the sub-rules are statistically independent, the output of **R1** is correct with a probability of $(0.7)^3 = 0.343$. Thus, we can observe that the output of a cascaded detector degrades much faster than the output of its individual units. This, together with the higher number of required training images, is the reason why semantic labelling still eludes us.

## 3.6 Performance Measurement

A timeless question in the field of content based image retrieval has been how to quantify the performance of a system. In our opinion, the real issue is not about finding good *performance measures*, but rather finding reliable *groundtruth* for the scenario under consideration. Ground truth here refers to the correct classification, labelling, or the retrieval result, which a system can achieve only in the best possible case[1].
Creating the ground truth for image retrieval or classification is in general a challenging task. Is mandatory to state clearly and unambiguously, the criterions that were used in creating the ground truth. For example, in a collection of images for different objects, each object must be reported to a separate class. The separation can be realized in a visual, or a more semantic level. For the *query-by-example* approach used in content-based image retrieval (CBIR), is possibile to create the so-called *relevance lists*. Each list contains, for each query image, all images in the database which would be perceived as being similar. The list indicates only a boolean information about each image which can be relevant or non-relevant. Since similarity is subjective, a common practice is to have multiple lists per query image, each based on the perceptions of a different individual, and to average out the results to reach a common ground truth.
The most common performance measures used in the literature are *precision* and *recall* [25]. Assume that the user has seen $k$ result images, out of which $k_R$ are good results (i.e. relevant), and the remaining $k_{NR}$ are nonrelevant. Further, let the total number of images in the database be $N$, out of which $N_R$ are relevant for the current query, and $N_{NR}$ are not. Then, the measures are defined as follows:

$$Recall_K = \frac{k_R}{N_R} \tag{3.7}$$

The recall is thus the ratio of retrieved relevant images to the total number of relevant images. By itself, it is not sufficient to measure system performance, as one could increase $k$ arbitrarily which would push `Recall` to 1 in the asymptotic case. Thus, one further formula defines:

$$Precision_K = \frac{k_R}{k} \tag{3.8}$$

[1]The term ground truth has its origins in cartography, satellite imagery and other remote sensing techniques, where the truth literally lies on the ground.

which measures the precision after $k$ images have been retrieved. Figure 3.6 gives a visual representation of the formulas introduced before.

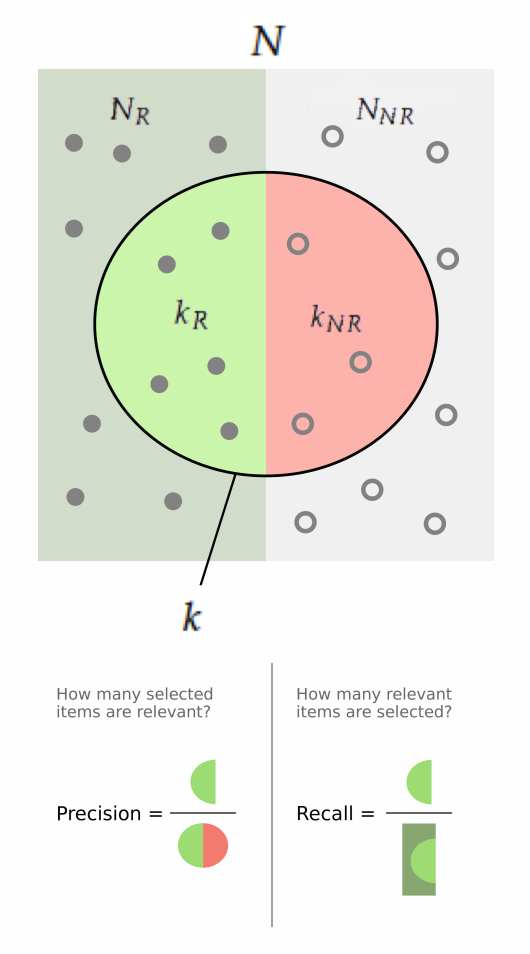



Figure 3.6: Interpretation of Precision and Recall

The `Precision` and `Recall` values can be plotted against each other for different values of $k$, the result being known as a *precision-recall graph*, which is well understood within the image retrieval community.

## 3.7 Single Query Image Scenario

In the *query-by-example* paradigm, the user begins the image search process by providing an initial query image. In the absence of any further information a CBIR system can only assume that the target images are distributed according to some symmetric function around the location of the query image in the feature space. These functions are known as *similarity measures.*

### 3.7.1 Similarity Measures

A measure is a valid distance metric, if the following conditions are satisfied:

1. **Non-negativity**: $d(\mathbf{x}, \mathbf{y}) \geq 0$ , with the equality holding only in the case of $\mathbf{x} = \mathbf{y}$
2. **Symmetry**: $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$
3. **Triangle Inequality**: $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$

In general it is desirable to use measures which are valid distance metrics, however it does not automatically mean that a non-metric function would perform poorly for a particular task. We give here a few common similarity measures which can be used in image retrieval. In the following, the elements of a feature vector will be represented by the notation $\mathbf{x} = (x_1, x_2, ..., x_n)$ with $n$ the dimensionality of the feature space.

**Minkowski distance** The general Minkowski distance of norm $p$ (also referred to as the distance induced by the $L_p$ norm) is given by :

$$d_{L_p}(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^{n} \mid x_i - y_i \mid^p \right)^{1/p} \tag{3.9}$$

which leads to the following special cases:

**Euclidean distance** is the distance measure induced by the $L_2$ norm:

$$d_E(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^{n} (x_i - y_i)^2 \right)^{1/2} \tag{3.10}$$

**Manhattan distance** Also known as the city-block distance, it is induced by the $L_1$ norm:

$$d_M(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mid x_i - y_i \mid \tag{3.11}$$

The ranking induced by the Euclidean and the Manhattan distance is illustrated in Figure 3.7.

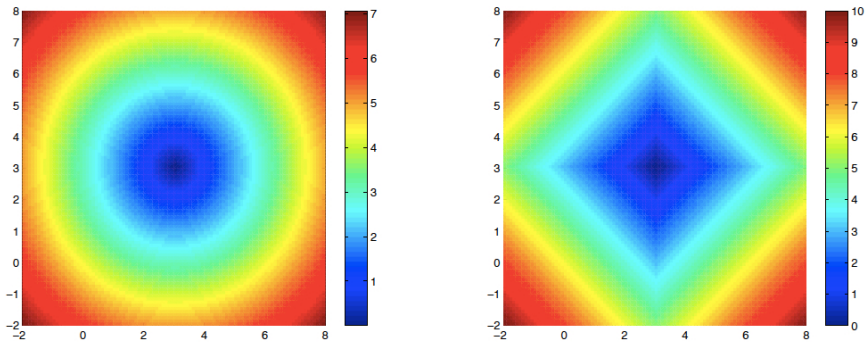


Figure 3.7: Ranking induced by similarity functions.Spherical ranking due to Euclidean distance (*left*) and ranking due to Manhattan distance (*right*).

## 3.8 Conclusion

In this chapter we have illustrated the task of image retrieval. We have shown a little history, some technical progress and illustrated each component of a CBIR system. Feature extraction techniques, performance measurements and the single-query scenario were illustrated at the end to complete the overview.

# Chapter 4

# Optimized Feature Hashing for Retrieval

*In this chapter we describe a novel version of the kmeans vector quantization approach introducing the possibility of assignment of a visual feature to multiple cluster centers during the quantization process. This approach greatly reduces the number of required cluster centers, as well as the required training data, performing a sort of quantized codebook soft assignment for an extremely compact hash code of visual features. We also introduce the usage of an efficient and recursive data structure to store datas. Results show that the our proposed technique is efficient,guarantees a low computational cost and results in a compact quantizer.* [1] [2]

## 4.1 Introduction

The technical features of modern smartphones have greatly improved in all aspects in recent years. Regarding the computational capabilities mobile

[1] This chapter has been partially published as "Compact hash codes and data structures for efficient mobile visual search" in *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on. IEEE, 2015* [23].

[2] *Acknowledgments:* this work was partially supported by the "Social Museum and Smart Tourism" project (CTN01 00034 231545). This research is based upon work supported [in part] by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA contract number 2014-14071600011

phones now have relatively fast processors and storage that is in the order of tens of GB. However certain characteristics are still quite lagging with respect to PCs. Because of these limitations, algorithms designed to run on desktop computers are not suitable for smartphones and mobile devices.
Regarding the problem of visual search, methods that aim at reducing computational costs typically use feature hashing, performing nearest neighbor search using Hamming distances. These methods generally use inverted files, e.g. hash tables, that require large quantities of memory to store the hash codes of the features. Moreover hash codes are relatively long (in the order of several tens of bits) to obtain a reasonable performance in retrieval, thus requiring fairly large amounts of memory when storing large scale databases of features.
In this chapter we present a novel method for feature hashing, based on k-means, that requires a very limited codebook size and that obtains good performance in retrieval even with very compact hash codes. We show also the benefit of using compact data structures to store a large database of features.
The proposed approach greatly reduces memory requirements and is suitable, also in terms of computational cost, for mobile visual search applications. The proposed method is compared to state-of-the-art approaches on a standard large scale dataset, showing a retrieval performance comparable or superior to more complex state-of-the-art approaches.

## 4.2 Proposed Method

The proposed method exploits a novel version of the k-means vector quantization approach, introducing the possibility of assignment of a visual feature to multiple cluster centers during the quantization process. This approach greatly reduces the number of required cluster centers, as well as the required training data, performing a sort of quantized codebook soft assignment for an extremely compact hash code of visual features. Table 4.1 summarizes the symbols used in the following.

The first step of the computation is a typical k-means algorithm for clustering. Given a set of observations $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ where each observation is a $D$-dimensional real vector, k-means clustering partitions the $n$ observations into $k(\leq n)$ sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the sum of distance functions of each point in the cluster to the $C_k$ centers. Its objective is to

Table 4.1: Notation table

| | |
|---|---|
| $x_i$ | feature to be hashed |
| $C_i$ | generic centroid |
| $S_i$ | cluster |
| $k$ | number of centroids; length of the hash code |
| $C_j$ | centroid associated to the $j^{th}$ bit of the hash code |
| $D$ | dimension of the feature |

find:

$$argmin_{\mathbf{s}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \| \mathbf{x} - C_i \|^2 \tag{4.1}$$

This process is convergent (to some local optimum) but the quality of the local optimum strongly depends on the initial assignment. We use the *k-means++* [2] algorithm for choosing the initial values, to avoid the poor clusterings sometimes found by the standard k-means algorithm.

### 4.2.1 Multi-k-means Hashing

K-means is typically used to compute the hash code of visual feature in unstructured vector quantization, because it minimizes the quantization error by satisfying the two Lloyd optimality conditions [47]. In a first step a dictionary is learned over a training set and then hash codes of features are obtained by computing their distance from each cluster center. Vectors are assigned to the nearest cluster center, whose code is used as hash code. Considering the case of 128-dimensional visual content descriptors, like SIFT or the FC7 layer of the VGG-M-128 CNN [11], this means that compressing them to 64 bits codes requires to use $k = 2^{64}$ centroids. In this case the computational cost of learning a k-means based quantizer becomes expensive in terms of memory and time because: *i)* there is the need of a quantity of training data that is several times larger than $k$, and *ii)* the execution time of the algorithm becomes unfeasible. Using hierarchical k-means (HKM) makes it possible to reduce execution time, but the problem of memory usage and size of the required learning set affects also this approach. Since the quantizer is defined by the $k$ centroids, the use of quantizers with a large number of centroids may not be practical or efficient: if a feature has a dimension $D$, there is need to store $k \times D$ values to represent the codebook of the quantizer. A possible solution to this problem is to reduce the length of the hash signature, but this typically affects negatively retrieval performance.

The use of product k-means quantization, proposed originally by Jégou *et al.* [47], overcomes this issue.

In our approach, instead, we propose to compute a sort of soft assignment within the k-means framework, to obtain very compact signatures and dimension of the quantizer, thus reducing its memory requirements, while maintaining a retrieval performance similar to that of [47].

The proposed method, called *multi-k-means* (in the following abbreviated as *m-k-means*), starts learning a standard k-means dictionary as shown in Eq. 4.1, using a very small number $k$ of centroids to maintain a low computational cost. Once we obtained our $\mathbf{C}_1, \ldots, \mathbf{C}_k$ centroids, the main difference resides in the assignment and creation of the hash code. Each centroid is associated to a specific bit of the hash code:

$$\begin{cases} \parallel x - C_j \parallel \leq \delta & j^{th}\ bit = 1 \\ \parallel x - C_j \parallel > \delta & j^{th}\ bit = 0 \end{cases} \tag{4.2}$$

where $x$ is the feature point and $\delta$ is a threshold measure given by

$$\delta = \begin{cases} (\prod_{j=1}^{k} \parallel x - C_j \parallel)^{\frac{1}{k}} & geometric\ mean \\ \frac{1}{k} \sum_{j=1}^{k} \parallel x - C_j \parallel & arithmetic\ mean \\ n^{th}\ nearest\ distance\ \parallel x - C_j \parallel & \forall j = 1, ..., k \end{cases} \tag{4.3}$$

i.e. centroid $j$ is associated to the $j^{th}$ bit of the hash code of length $k$; the bit is set to 1 if the feature to be quantized is assigned to its centroid, or to 0 otherwise.

A feature can be assigned to more than one centroid using two different approaches:

*i) m-k-means-$t_1$* - using Eq. (4.2) and one of the first two thresholds of Eq. (4.3). In this case the feature vector is considered as belonging to all the centroids from which its distance is below the threshold. Experiments have shown that the arithmetic mean is more efficient with respect to the geometric one, and all the experiments will report results obtained with it.

*ii) m-k-means-$n_1$* - using Eq. (4.2) and the third threshold of Eq. (4.3), i.e. assigning the feature to a predefined number $n$ of nearest centroids.

We also introduce two variants (*m-k-means-$t_2$* and *m-k-means-$n_2$*) to the previous approaches by randomly splitting the training data into two groups and creating two different codebooks for each feature vector. The final hash code is given by the union of these two codes.

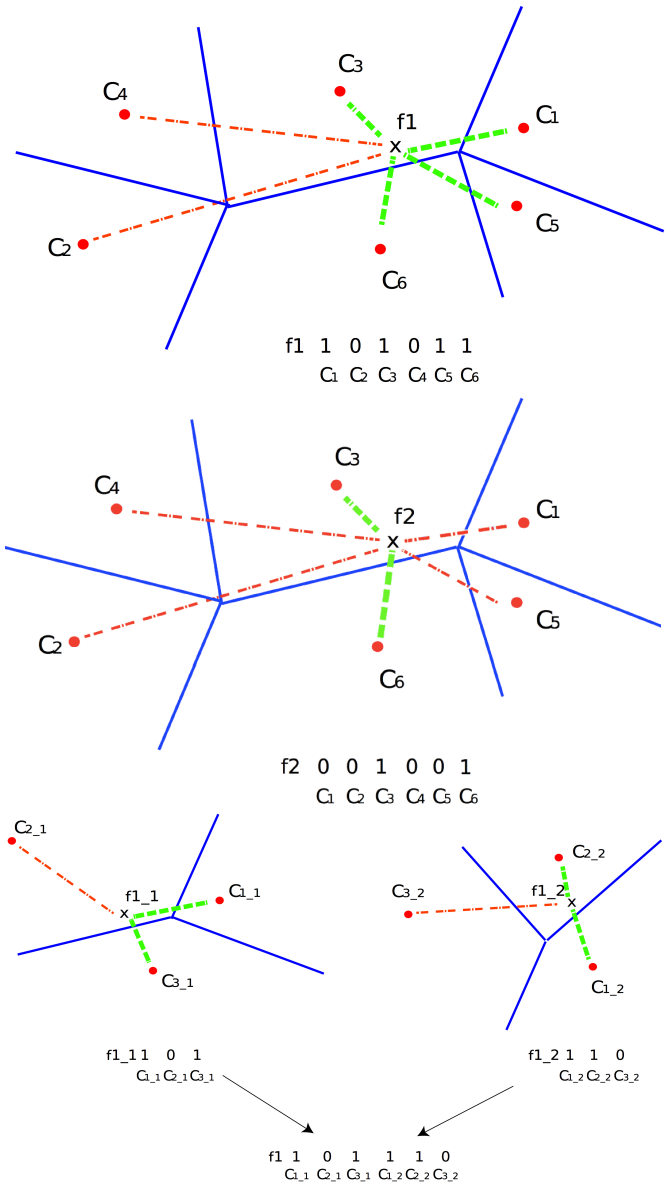



Figure 4.1: Toy examples illustrating the proposed method: *(top)* features can be assigned (green line) to a variable number of nearest clusters (e.g. those with distances below the mean $\delta$ - i.e. *m-k-means-*$t_1$); *(middle)* features can be assigned to a fixed number of clusters (e.g. the 2 nearest clusters - i.e. *m-k-means-*$n_1$);*(bottom)* hash code created from two different codebooks (*m-k-means-*$x_2$, where $x$ can be either $t$ or $n$). If a feature is assigned to a centroid the corresponding bit in the hash code is set to 1.

With the proposed approach it is possible to create hash signatures using a much smaller number of centroids than using the usual k-means baseline, since each centroid is directly associated to a bit of the hash code. This approach can be considered a quantized version of codebook soft assignment [102] and, similarly, it alleviates the problem of codeword ambiguity while reducing the quantization error.

Fig. 4.1 illustrates the quantization process and the resulting hash codes in three cases: one in which a vector is assigned to a variable number of centroids (*m-k-means-*$t_1$), one in which a vector is assigned to a predefined number of centroids (*m-k-means-*$n_1$) and one in which the resulting code is created by the union of two different codes created using two different codebooks (*m-k-means-*$t_2$ and *m-k-means-*$n_2$). In all cases the feature is assigned to more than one centroid. An evaluation of these two approaches is reported in Sect. 5.4.

Typically a multi probe approach is used to solve the problem of ambiguous assignment to a codebook centroid (in case of vector quantization, as in the coarse quantization step of PQ [47]) or quantization error (e.g. in case of scalar quantization, as in [84, 109]); this technique stems from the approach originally proposed in [71], to reduce the need of creating a large number of hash tables in LSH. The idea is that if a object is close to a query object $q$, but is not hashed to the same bucket of $q$, it is still likely hashed to a bucket that is near, i.e. to a bucket associated with an hash that has a small difference w.r.t. the hash of $q$. With this approach one or more bits of the query hash code are flipped to perform a query expansion, improving recall at the expense of computational cost and search time. In fact, if we chose to try all the hashes within an Hamming distance of 1 we have to create variations of the original hash of $q$ flipping all the bits of the hash, one at a time. This means that for a hash code of length $k$ we need to repeat the query with additional $k$ hashes. In the proposed method this need of multi probe queries is greatly reduced, because of the possibility of assignment of features to more than one centroid. For example, consider either Fig. 4.1 *(top)* or *(middle)*: if a query point nearby $f_1$ or $f_2$ falls in the Voronoi cell of centroid $C_6$, using standard k-means it could be retrieved only using a multi probe query, instead the proposed approach maintains the same hash code.

### 4.2.2 Computational Complexity

Let us consider a vector with dimensionality $D$, and desired hash code length of 64 bits. Standard k-means has an assignment complexity of $kD$, where $k = 2^{64}$, while the proposed approach instead needs $k' = 64$ centroids, has a complexity of $k'D$ and requires $k'D$ floats to store the codebook. Product Quantization requires $K^*D$ floats for the codebook and has an assignment complexity of $k^*D$, where $k^* = k^{1/m}$, using typically $k^* = 256$ and $m = 8$ values, for a 64 bit length [47]; in this case the cost of the proposed method is a quarter of that of PQ.

## 4.3 Indexing

In this section we present an efficient and recursive data structure to store datas. We also show,in the experimental section, how the resulting hash code from 4.2.1 is stored into that structure and how it is suitable for devices with very limited RAM such a mobile devices. Most of the approaches presented in scientific literature have relied on inverted files [47,84,109] that are typically implemented as hash tables[3] [12,77,92,100] or B-Trees. Instead we propose the use of a variant of radix tree (also known as 'patricia trie' or 'trie') for an extremely compact storage of the hash codes of visual features. The combination of these two solutions results in a greatly reduced consumption of memory and improved search speed.

### 4.3.1 Trie Data Structure

Radix tries are often used in approximate string matching algorithms [1], such as those required for spell checking. The trie can be used to implement an inverted file, where the key to search the data is stored in the position of the nodes. A memory efficient variant of radix tree is the patricia trie, a data structure that represents a space-optimized trie in which each node with only one child is merged with its parent. Fig. 4.2 compares a trie with a patricia trie.

In this work we propose the use of Matching Algorithm with Recursively Implemented StorAge (MARISA) trie. MARISA trie is a recursive data structure in which a patricia trie is used to represent another patricia trie. This recursion makes the data structure more compact at the expenses of

[3]E.g. Yael Library `http://yael.gforge.inria.fr`

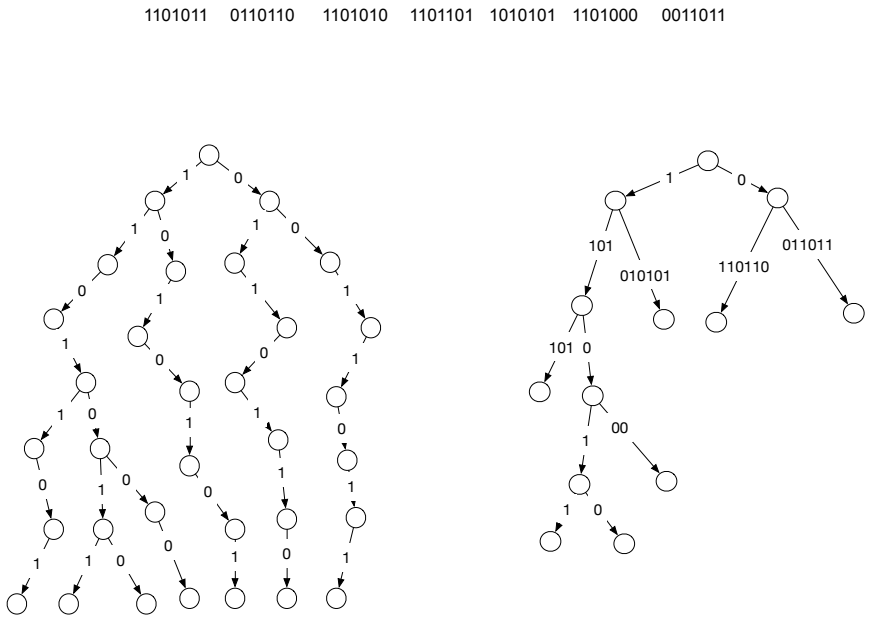



Figure 4.2: Comparison of trie (*left*) vs. patricia trie (*right*), used to store 7 hash codes (*top*).

search performance. In particular, to maintain a good time-space tradeoff, we have used one level of recursion depth.

Patricia tries have a better space complexity than hash tables, while time complexity is comparable only when considering imbalanced hash tables. However, the compactness of the data structure makes it amenable to be maintained in the CPU cache, thus greatly improving its speed.

## 4.4 Experimental Results

In this section we present an exhaustive comparation with all the method presented in chapter 2 and we evaluate the performance of the Trie data structure. The variants of the proposed method (*m-k-means-*$t_1$, *m-k-means-*$n_1$, *m-k-means-*$t_2$ and *m-k-means-*$n_2$) have been thoroughly compared to several state-of-the-art approaches using standard datasets, experimental setups and evaluation metrics.

### 4.4.1 Datasets

#### BIGANN Dataset

Is a large-scale dataset commonly used to compare methods for visual feature hashing and approximate nearest neighbor search [3,28,47,48,51,76,77]. The dataset is composed by three different sets of SIFT and GIST descriptors, each one divided in three subsets: a learning set, a query set and base set; each query has corresponding ground truth results in the base set, computed in an exhaustive way with Euclidean distance, ordered from the most similar to the most different. For SIFT1M and SIFT1B query and base descriptors have been extracted from the INRIA Holidays images [46], while the learning set has been extracted from Flickr images. For GIST1M query and base descriptors are from INRIA Holidays and Flickr 1M datasets, while learning vectors are from [97]. In all the cases query descriptors are from the query images of INRIA Holidays (see Figure 4.3). The characteristics of the dataset are summarized in Table 4.2.

#### DEEP1B Dataset

Is a recent dataset used to treat the problem of indexing CNN features [4] and produced using a deep CNN based on the GoogLeNet [95] architecture and trained on ImageNet dataset [19]. Descriptors are extracted from the outputs of the last fully-connected layer, compressed using PCA to 96 dimensions, and $l_2$-normalized. The characteristics of the dataset are summarized in Table 4.3.

Table 4.2: BIGANN datasets characteristics

| **vector dataset** | SIFT 1M | SIFT 1B | GIST 1M |
|---|---|---|---|
| descriptor dimensionality $D$ | 128 | 128 | 960 |
| # learning set vectors | 100,000 | 100,000,000 | 500,000 |
| # database set vectors | 1,000,000 | 1,000,000,000 | 1,000,000 |
| # queries set vectors | 10,000 | 10,000 | 1,000 |
| # nearest vectors for each query | 100 | 1000 | 100 |

Table 4.3: DEEP1B datasets characteristics

| | |
|---|---|
| descriptor dimensionality $D$ | 96 |
| # learning set vectors | 358,480,000 |
| # database set vectors | 1,000,000,000 |
| # queries set vectors | 10,000 |
| # nearest vectors for each query | 1 |

Figure 4.3: Sample images from INRIA Holiday dataset. Left column shows the query images, the other columns show similar images.

### CIFAR-10 Dataset

Consists of 60,000 colour images ($32 \times 32$ pixels) in 10 classes, with 6,000 images per class [56] (see Figure 4.4). The dataset is split into training and test sets, composed by 50,000 and 10,000 images respectively . A retrieved image is considered relevant for the query if it belongs to the same class. This dataset has been used for ANN retrieval in [67, 105].

### MNIST Dataset

Consists of 70,000 handwritten digits images [61] ($28 \times 28$ pixels, see Figure 4.5). The dataset is split into 60,000 training examples and 10,000 test examples. Similarly to CIFAR-10, a retrieved image is considered relevant if it belongs to the same class of the query. This dataset has been used for ANN retrieval in [67, 105].

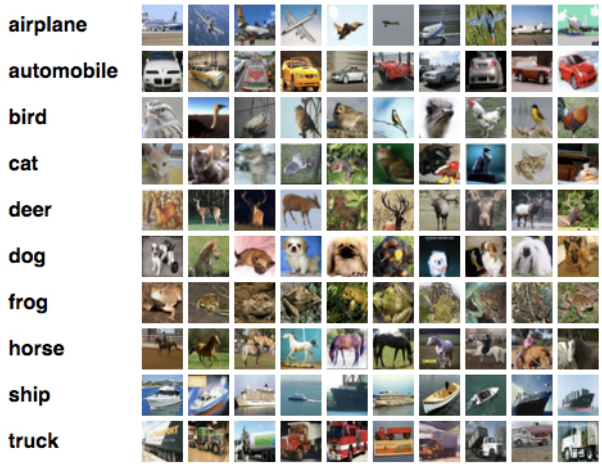



Figure 4.4: Sample images from CIFAR-10 dataset

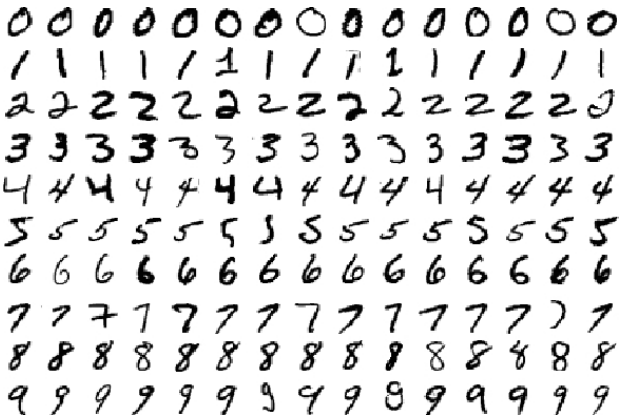


Figure 4.5: Sample images from MNIST dataset

### 4.4.2 Evaluation Metrics

The performance of ANN retrieval in BIGANN dataset is evaluated using *recall@R*, which is used in most of the results reported in the literature [3, 28, 47, 48, 51, 76] and it is, for varying values of R, the average rate of queries for which the 1-nearest neighbor is retrieved in the top R positions.

In case of $R = 1$ this metric coincides with *precision@1*. The same measure has been used by the authors of the DEEP1B dataset [4].

Performance of image retrieval in CIFAR-10 and MNIST is measured following the setup of [105], using Mean Average Precision:

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \tag{4.4}$$

where

$$AveP = \int_0^1 p(r)dr \tag{4.5}$$

is the area under the precision-recall curve and $Q$ is the number of queries.

### 4.4.3 Configurations and Implementations

**BIGANN** We use settings which reproduce top performances at 64-bit codes. We perform search with a non-exhaustive approach. For each query 64 bits binary hash code of the feature and Hamming distance measure are used to extract small subsets of candidates from the whole database set (Table 4.2). Euclidean distance measure is then used to re-rank the nearest feature points, calculating recall@R values in these subsets.

**DEEP1B** We use the CNN features computed in [4], hashed to 64-bit codes. Searching process is done in a non-exhaustive way, using Hamming distances to reduce the subsets of candidates from the whole database set. After we have extracted a shortlist of candidates we perform a re-rank step based on Euclidean distances and we calculate recall@R values.

**CIFAR-10** We use features computed with the framework proposed in [67] (Figure 4.6). The process is carried out in two steps: in the first step a supervised pre-training on the large-scale ImageNet dataset [57] is performed. In the second step fine-tuning of the network is performed, with the insertion of a latent layer that simultaneously learns domain specific feature representations and a set of hash-like functions. The authors used the pre-trained CNN model proposed by Krizhevsky *et al.* [57] and implemented in the Caffe CNN library [49]. In our experiments we use features coming from the $FCh$ Layer (Latent Layer H), which has a size of 48 nodes.

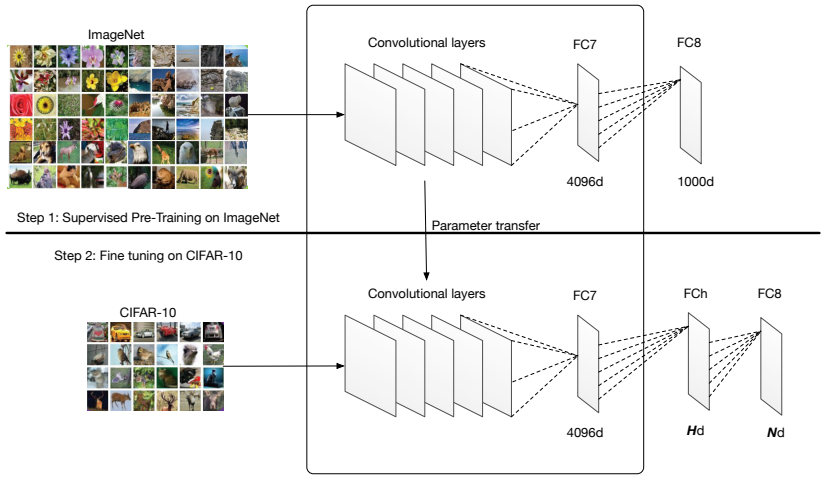



Figure 4.6: Framework used for CNN feature extraction on CIFAR-10 [67]: we use the values of the nodes of the FCh layer as feature (48 dimensions).

**MNIST** We use LeNet CNN to compute our features in MNIST. This is a network architecture developed by LeCun [60] that was especially designed for recognizing handwritten digits, reading zip codes, etc. It is a 8-layer network with 1 input layer, 2 convolutional layers, 2 non-linear down-sampling layers, 2 fully connected layers and a Gaussian connected layer with 10 output classes. We used a modified version of LeNet [49] and we obtain features from the first fully connected layer.

We perform search with a non-exhaustive approach on both CIFAR-10 and MNIST datasets. For each image we extract a 48-dimensional feature vector for CIFAR-10, and 500-dimensional feature vector for MNIST, from the respective network and then we generate a 48 bits binary hash code using the proposed methods of Sect. 4.2. Hamming distance is used to select the nearest hash codes for each query and similarity measure given by

$$similarity = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\| \mathbf{A} \| \| \mathbf{B} \|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4.6}$$

where $A_i$ and $B_i$ are the components of the original feature vectors $A$ and $B$, is used to re-rank the nearest visual features.

### 4.4.4 Results on BIGANN: SIFT1M, GIST1M

In this set of experiments the proposed approach and its variants are compared on the SIFT1M (Table 4.4) and GIST1M (Table 4.5) datasets against several methods presented in chapter 2 : Product Quantization (ADC and IVFADC) [47], PQ-RO [34], PQ-RR [34], Cartesian k-means [76], OPQ-P [28, 29], OPQ-NP [28, 29], LOPQ [51], a non-exhaustive adaptation of OPQ [28], called I-OPQ [51], RVQ [13] , RVQ-P [34] and RVQ-NP [34].

ADC (*Asymmetric Distance Computation*) is characterized by the number of sub vectors $m$ and the number of quantizers per sub vectors $k^*$, and produces a code of length $m \times log_2 k^*$.

IVFADC (*Inverted File with Asymmetric Distance Computation*) is characterized by the codebook size $k'$ (number of centroids associated to each quantizer), the number of neighbouring cells $w$ visited during the multiple assignment, the number of sub vectors $m$ and the number of quantizers per sub vectors $k^*$ which is in this case fixed to $k^* = 256$. The length of the final code is given by $m \times log_2 k^*$.

PQ-RO [34] is the Product Quantization approach with data projection by randomly order dimensions.

PQ-RR [34] is the Product Quantization approach with data projection by both PCA and randomly rotation.

Cartesian k-means (ck-means) [76] models region center as an additive combinations of subcenters. Let $m$ be the number of subcenters, with $h$ elements, then the total number of model centers is $k = h^m$, but the total number of subcenters is $h \times m$, and the number of bits of the signature is $m \times log_2 h$.

OPQ-P [28, 29] is the parametric version of Optimized Product Quantization (OPQ), that assumes a parametric Gaussian distribution of features and performs space decomposition using an orthonormal matrix computed from the covariance matrix of data.

OPQ-NP [28, 29] is the non-parametric version of OPQ, that does not assume any data distribution and alternatively optimizes sub-codebooks and space decomposition.

LOPQ (Locally optimized product quantization) [51] is a vector quantizer that combines low distortion with fast search applying a local optimization over rotation and space decomposition.

I-OPQ [51] is a non-exhaustive adaptation of OPQ (Optimized Product Quantization [28]) which use either OPQ-P or OPQ-NP global optimization.

RVQ [13] approximates the quantization error by another quantizer instead of discarding it. In this method several stage-quantizers, each one with its corresponding stage-codebook, are connected sequentially. Each stage-quantizer approximates the residual vector of the preceding stage by one of centroids of its stage-codebook and generates a new residual vector for the next stage.

RVQ-P [34] is a parametric version of RVQ, where stage-codebooks and space decomposition of RVQ are optimized using SVD.

RVQ-NP [34] is a non-parametric version of RVQ, using the same techniques of RVQ-P, but optimizing a space decomposition for all the stages.

The parameters of the proposed methods are set as follows: for *m-k-means-*$t_1$ we use as threshold the arithmetic mean of the distances between feature vectors and centroids to compute hash code; *m-k-means-*$n_1$ creates hash code by setting to 1 the corresponding position of the first 32 (SIFT1M) and first 48 (GIST1M) nearest centroids for each feature; *m-k-means-*$t_2$ and *m-k-means-*$n_2$ create two different sub hash codes for each feature by splitting into two parts the training phase and combine these two sub parts into one single code to create the final signature. Since we have a random splitting during the training phase, these experiments are averaged over a set of 10 runs.

Table 4.4: *Recall@R* on SIFT1M - Comparison between our method (m-k-means-$t_1$, m-k-means-$n_1$ with n=32, m-k-means-$t_2$ and m-k-means-$n_2$ with n=32), the Product Quantization method (PQ ADC and PQ IVFADC) [47], Cartesian k-means method (ck-means) [76], a non-exhaustive adaptation of the Optimized Product Quantization method (I-OPQ), a Locally optimized product quantization method (LOPQ) [51], OPQ-P and OPQ-NP [28, 29], and PQ-RO, PQ-RR, RVQ-P and RVQ-NP [34].

| Method | R@1 | R@10 | R@100 | R@1000 | R@10000 |
|---|---|---|---|---|---|
| PQ(ADC) [47] | 0.224 | 0.600 | 0.927 | 0.996 | 0.999 |
| PQ(IVFADC) [47] | 0.320 | 0.739 | 0.953 | 0.972 | 0.972 |
| PQ-RO [34] | 0.177 | 0.501 | 0.854 | N/A | N/A |
| PQ-RR [34] | 0.107 | 0.331 | 0.695 | N/A | N/A |
| ck-means [76] | 0.231 | 0.635 | 0.930 | **1** | **1** |
| OPQ-P [28, 29] | 0.219 | 0.563 | 0.917 | N/A | N/A |
| OPQ-NP [28, 29] | 0.242 | 0.627 | 0.938 | N/A | N/A |
| I-OPQ [51] | 0.299 | 0.691 | 0.875 | 0.888 | 0.888 |
| LOPQ [51] | 0.380 | 0.780 | 0.886 | 0.888 | 0.888 |
| RVQ [13] | 0.264 | 0.659 | 0.949 | **1** | **1** |
| RVQ-P [34] | 0.397 | 0.821 | 0.983 | N/A | N/A |
| RVQ-NP [34] | 0.271 | 0.686 | 0.958 | N/A | N/A |
| **m-k-means-$t_1$** | 0.501 | 0.988 | **1** | **1** | **1** |
| **m-k-means-$t_2$** | **0.590** | **0.989** | **1** | **1** | **1** |
| **m-k-means-$n_1$** | 0.436 | 0.986 | **1** | **1** | **1** |
| **m-k-means-$n_2$** | 0.561 | 0.986 | **1** | **1** | **1** |

Table 4.5: *Recall@R* on GIST1M - Comparison between our method (m-k-means-$t_1$, m-k-means-$n_1$ with n=48, m-k-means-$t_2$ and m-k-means-$n_1$ with n=48), the Product Quantization method (ADC and IVFADC) [47], Cartesian k-means method (ck-means) [76], a non-exhaustive adaptation of the Optimized Product Quantization method (I-OPQ) [51], a Locally optimized product quantization method (LOPQ) [51], OPQ-P and OPQ-NP [28, 29], and PQ-RO, PQ-RR, RVQ-P and RVQ-NP [34].

| Method | R@1 | R@10 | R@100 | R@1000 | R@10000 |
|---|---|---|---|---|---|
| PQ(ADC) [47] | 0.145 | 0.315 | 0.650 | 0.932 | 0.997 |
| PQ(IVFADC) [47] | 0.180 | 0.435 | 0.740 | 0.966 | 0.992 |
| PQ-RO [34] | 0.034 | 0.056 | 0.136 | N/A | N/A |
| PQ-RR [34] | 0.033 | 0.062 | 0.124 | N/A | N/A |
| ck-means [76] | 0.135 | 0.335 | 0.728 | 0.952 | 0.985 |
| OPQ-P [28, 29] | 0.095 | 0.297 | 0.629 | N/A | N/A |
| OPQ-NP [28, 29] | 0.089 | 0.277 | 0.642 | N/A | N/A |
| I-OPQ [51] | 0.146 | 0.410 | 0.729 | 0.862 | 0.866 |
| LOPQ [51] | 0.160 | 0.461 | 0.756 | 0.860 | 0.866 |
| RVQ [13] | 0.095 | 0.276 | 0.656 | 0.936 | **1** |
| RVQ-P [34] | **0.309** | 0.700 | 0.950 | N/A | N/A |
| RVQ-NP [34] | 0.107 | 0.314 | 0.678 | N/A | N/A |
| **m-k-means-$t_1$** | 0.111 | 0.906 | **1** | **1** | **1** |
| **m-k-means-$t_2$** | 0.123 | 0.890 | **1** | **1** | **1** |
| **m-k-means-$n_1$** | 0,231 | **0,940** | **1** | **1** | **1** |
| **m-k-means-$n_2$** | 0,265 | 0,905 | **1** | 0,999 | 0,999 |

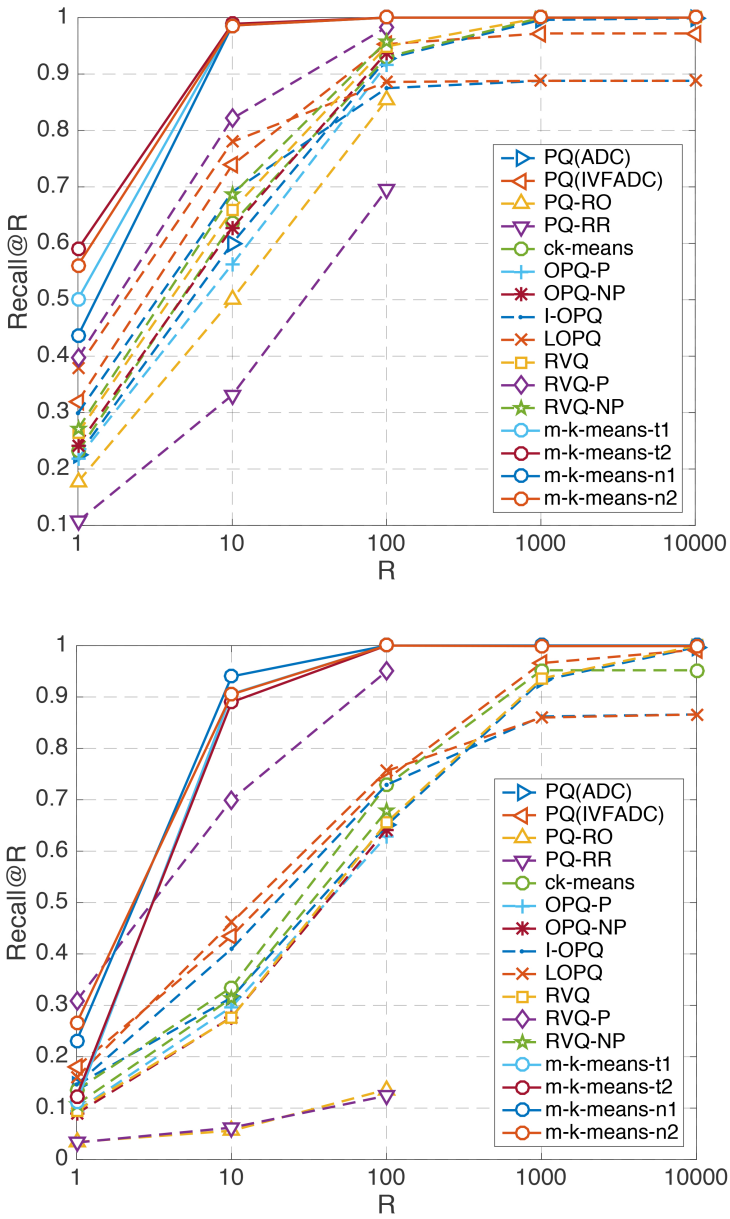



Figure 4.7: Results on SIFT1M (*top*). Results on GIST1M (*bottom*)

### 4.4.5 Results on BIGANN: SIFT1B

In this experiment we compare our method on the large scale SIFT1B dataset (Table 4.6) against LOPQ and a sub-optimal variant LOR+PQ [51], single index PQ approaches IVFADC+R [48] and ADC+R [48], I-OPQ [28] and ck-means [76], and a multi-index method Multi-D-ADC [3]. [48] differ from the standard IVFADC [47] and ADC [47] in using short quantization codes to re-rank the NN candidates. *m-k-means-*$t_1$ uses the same setup of the previous experiment; *m-k-means-*$n_1$ uses the first 24 nearest centroids for each feature.

Table 4.6: *Recall@R* on SIFT1B - Comparison between our method (m-k-means-$t_1$, m-k-means-$n_1$ with n=24), the Product Quantization method [48], a non-exhaustive adaptation of the Optimized Product Quantization method (I-OPQ) [51], a multi-index method (Multi-D-ADC), a Locally optimized product quantization method (LOPQ) with a sub-optimal variant (LOR+PQ) [51].

| Method | R@1 | R@10 | R@100 |
|---|---|---|---|
| PQ(ADC+R) [48] | 0.656 | 0.970 | 0.985 |
| PQ(IVFADC+R) [48] | 0.630 | 0.977 | 0.983 |
| ck-means [76] | 0.084 | 0.288 | 0.637 |
| I-OPQ [51] | 0.114 | 0.399 | 0.777 |
| Multi-D-ADC [3] | 0.165 | 0.517 | 0.860 |
| LOR+PQ [51] | 0.183 | 0.565 | 0.889 |
| LOPQ [51] | 0.199 | 0.586 | 0.909 |
| **m-k-means-$t_1$** | 0.775 | 0.917 | 0.928 |
| **m-k-means-$n_1$** | **0.787** | **0.990** | **1** |

### 4.4.6 Results on DEEP1B

Experiments on DEEP1B [4] are shown in Table 4.7. We use a configuration with an hash code length of 64 bits for the *m-k-means-t* and *m-k-means-n* variants. The comparison is made against IMI [4], NO-IMI [4] and GNO-IMI [4], for which we report the results obtained by the authors using a rerank approach for codes of 64 bits. Following the experimental setup used in [4], we considered $R = 1$, $R = 5$ and $R = 10$ for the *recall@R* measure. The proposed method obtains best results in both configurations (*m-k-means-*$t_1$ and *m-k-means-*$n_1$) and considering $R = 1$ obtains a result approximately three times greater than the others methods; for the other values of R the improvement is between 2 and 1.5×.

Table 4.7: *Recall@R* on DEEP1B - Comparison between our method (m-k-means-$t_1$, m-k-means-$n_1$ with n=24), Inverted Multi-Index (IMI) [4], Non-Orthogonal Inverted Multi-Index (NO-IMI) [4] and Generalized Non-Orthogonal Inverted Multi-Index (GNO-IMI) [4].

| Method | R@1 | R@5 | R@10 |
|---|---|---|---|
| NO-IMI [4] | 0.272 | 0.492 | 0.593 |
| IMI [4] | 0.241 | 0.450 | 0.545 |
| GNO-IMI [4] | 0.276 | 0.508 | 0.613 |
| **m-k-means-$t_1$** | 0.694 | 0.892 | 0.912 |
| **m-k-means-$n_1$** | **0.768** | **0.988** | **0.999** |

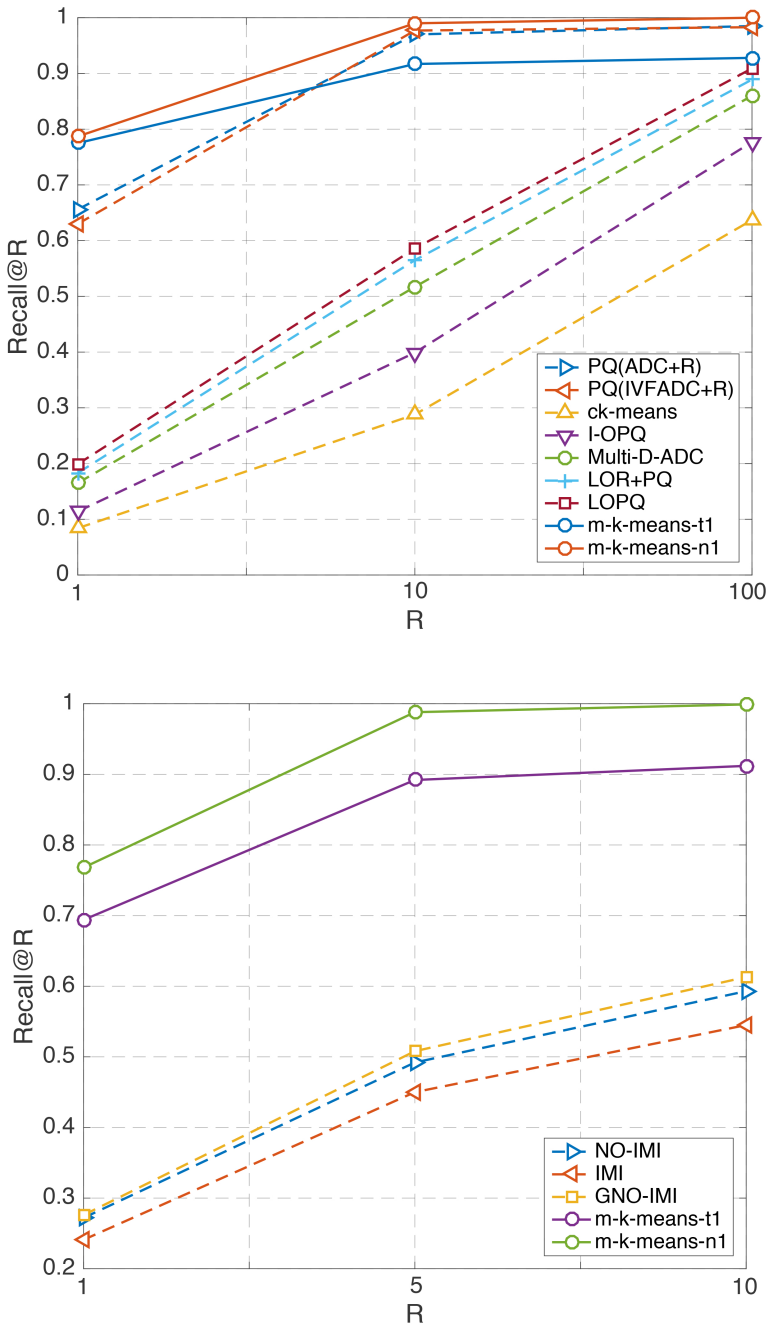



Figure 4.8: Results on SIFT1B (*top*). Results on DEEP1B (*bottom*)

### 4.4.7 Results on CIFAR-10, MNIST

In the experiments on CIFAR-10 [56] and MNIST [61] images dataset we use the following configurations for the proposed method: hash code length of 48 bits (the same length used by the compared methods),arithmetic mean for the *m-k-means-*$t_1$ variant, $n = 24$ for *m-k-means-*$n_1$.

Queries are performed using a random selection of 1,000 query images (100 images for each class), considering a category labels ground truth relevance ($Rel(i)$) between a query $q$ and the $i^{th}$ ranked image. So $Rel(i) \in \{0, 1\}$ with 1 for the query and $i^{th}$ images with the same label and 0 otherwise. This setup has been used in [67, 105]. Since we select queries in a random way the results of these experiments are averaged over a set of 10 runs.

Table 4.8: MAP results on CIFAR-10 and MNIST. Comparison between our method (m-k-means-$t_1$, m-k-means-$n_1$ with n=24, m-k-means-$t_2$ and m-k-means-$n_2$ with n=24) with KSH [69], ITQ-CCA [33], MLH [78], BRE [58], CNNH [105], CNNH+ [105], KevinNet [67], LSH [30], SH [104], ITQ [33]. Results from [67, 72, 105].

| Method | CIFAR-10 (MAP) | MNIST (MAP) |
|---|---|---|
| LSH [30] | 0.120 | 0.243 |
| SH [104] | 0.130 | 0.250 |
| ITQ [33] | 0.175 | 0.429 |
| BRE [58] | 0.196 | 0.634 |
| MLH [78] | 0.211 | 0.654 |
| ITQ-CCA [33] | 0.295 | 0.726 |
| KSH [69] | 0.356 | 0.900 |
| CNNH [105] | 0.522 | 0.960 |
| CNNH+ [105] | 0.532 | 0.975 |
| ACH [72] | 0.600 | - |
| KevinNet [67] | 0.894 | **0.985** |
| **m-k-means-**$t_1$ | 0.953 | 0.972 |
| **m-k-means-**$t_2$ | 0.849 | 0.964 |
| **m-k-means-**$n_1$ | **0.972** | 0.969 |
| **m-k-means-**$n_2$ | 0.901 | 0.959 |

### 4.4.8 Trie Data Structure Performance

In experiments we evaluate the performance of MARISA trie w.r.t. the data structures typically used to implement inverted files, i.e. hash table and binary tree. The $10^6$ feature vectors of the database set (SIFT1M) have been coded using the proposed multi-k-means quantization, then stored using C++ implementations of the data structures. The STL versions of `unordered_multimap` and `multimap` provided by GCC C++ compiler have been used for hash tables and B-Tree, respectively. Results reported in Tab. 4.9 show that MARISA trie obtains a dramatic improvement both in terms of speed and size. In particular hash table and B-Tree would occupy a very large percentage of RAM in a mobile phone (e.g. $\sim 10\%$ in an Apple iPhone 6), while the trie fits the L1 cache of an ARM CPU commonly used in mobile phones.

Table 4.9: Comparison of data structures used in the experiments to store 1 million SIFT hash codes (computed with the proposed approach). Search time measures the time required to perform the series using all the 10,000 query vectors of the dataset.

| | MARISA trie | Hash table | Binary Tree |
|---|---|---|---|
| Dimension (Kb) | **19** | 90,112 | 93,184 |
| Search Time (ms) | **25.3** | 245.31 | 370 |

## 4.5 Conclusion

We introduced a novel version of the k-means vector quantization obtaining very high performances compared with other state-of-the art methods. We show how this approach produces very compact hash codes maintaining high performances in retrieval. We also introduced a recursive data structure to store datas which has a better space complexity than has tables obtaining remarkable achievements in terms of speed and size.

# Chapter 5

# Efficient and Distributed Image Retrieval

*In this chapter we introduce a new approach for efficient image retrieval based on the m-k-means hashing applied on CNN features and the use of an indexing structure based on Bloom filters. These filters are used as gatekeepers for the database of image features, allowing to avoid to perform a query if the query features are not stored in the database and speeding up the query process, without affecting retrieval performance. Thanks to the limited memory requirements the system is suitable for mobile applications and distributed databases, associating each filter to a distributed portion of the database. Experimental validation has been performed on three standard image retrieval datasets, outperforming state-of-the-art hashing methods in terms of precision, while the proposed indexing method obtains a* $2\times$ *speedup.*

## 5.1 Introduction

Content based image retrieval (CBIR) has been an active research topic in computer vision and multimedia in the last decades, and it is still very relevant due to the emergence of social networks and the creation of web-scale image databases. Most of the works have addressed the development of effective visual features, from engineered features like SIFT and GIST to, more recently, learned features such as CNNs [5]. To obtain scalable CBIR systems

features are typically compressed or hashed, to reduce their dimensionality and size. However, research on data structures that can efficiently index these descriptors has attracted less attention, and typically simple inverted files (e.g. implemented as hash tables) are used.
In this chaper we address the problem of approximate nearest neighbor (ANN) image retrieval proposing a simple and effective data structure that can greatly reduce the need to perform any comparison between the descriptor of the query and those of the database, when the probability of a match is very low. Considering the proverbial problem of finding a needle in a haystack, the proposed system is able to tell when the haystack probably contains no needle and thus the search can be avoided completely.
To achieve this we use an effective hashing method for CNN descriptors, and use this code to perform ANN retrieval in a database. To perform an immediate rejection of a search that should not return any result we store the hash code in a Bloom filter, i.e. a space efficient probabilistic data structure that is used to test the presence of an element in a set. To the best of our knowledge this is the first time that this data structure has been proposed for image retrieval since, natively, it has no facility to handle approximate queries. We perform extensive experimental validation on three standard datasets, showing how the proposed hashing method improves over state-of-the-art methods, and how the data structure greatly improves computational cost and makes the system suitable for application to mobile devices and distributed image databases.

## 5.2 Convolutional Neural Network

Traditional non-convolutional neural net is a stack of fully connected layers. The layers extracts different levels of concepts from the input.When the input has spatial information, for instance, the input is an image, this spatial information is lost during the process.
While in Convolutional Neural Network (CNN), the feature maps generated from the convolutional layers has the same layer out as the input image. Thus the spatial information is still there. Convolution scans the whole image with a square filter, that extracts local information from the underlying patches. It works just like a detector. Convolutional Neural Networks (CNN) are a variant of the Multi-Layer Perceptron Network (MPL). This kind of networks are composed of more levels and the neurons at each level

are completely connected to the neurons of the previous and following level. At the end CNNs were presented to cope the computational and memory cost of MLPs during the training phase.
A convolutional neural network is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. CNNs consist of multiple layers of receptive fields and may include local or global pooling layers, which combine the outputs of neuron clusters [15,57]. They also consist of various combinations of convolutional and fully connected layers, with pointwise nonlinearity applied at the end of or after each layer [14]. One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces memory footprint and improves performance [62].

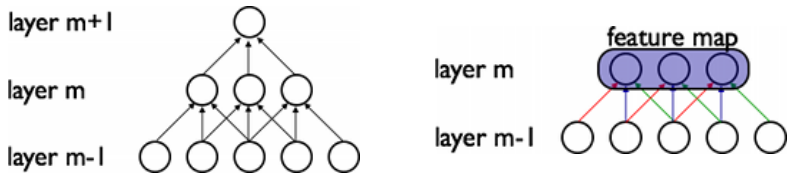



Figure 5.1: Two properties which distinguish CNNs from MLPs. Sparse connectivity (*left*) and Shared Weights (*right*)

Training a CNN means to adjust the weight of the layers through the back-propagation algorithm [40].

Initially CNN were designed for classification task but subsequently it was realized that CNNs could be used to extract global features from images. Now we analize the two Neural Network architectures used in this chapter.

### 5.2.1 VGG16

VGG16 [90] is a network with a first input level of 224×224 with 3 RGB channels. Following we have small convolutional filters (3×3 with stride 1). Some of these convolutional levels have also a max-pooling step which is applied over areas of 2X2 with stride 1. After the convolutional levels stack we can found three fully connected layers: first two have a dimension of 4096 while the third have a dimension of 1000. The last level is a softmax step for a

classification over 1000 classes. Table 5.1 shows a schematic representation of VGG16 (column D). This network was used for our experiments and we used features extracted from the second fully connected layer of 4096 dimension (FC-4096).
Unfortunately this network has a major drawback: the file which contains the weights of the net is considerable big (550 MB). This means that we get several seconds to extract featuters from an image and and it is not suitable in a mobile approach.

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |

| input (224×224 RGB image) | | | | | |
|---|---|---|---|---|---|
| conv3-64 | conv3-64 **LRN** | conv3-64 **conv3-64** | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 **conv3-128** | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 **conv1-256** | conv3-256 conv3-256 **conv3-256** | conv3-256 conv3-256 **conv3-256** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Table 5.1: Table with architectures showed in [90]. VGG16 configuration is represented by the column D

### 5.2.2 GoogLeNet with Batch Normalization

In this subsection we present some basic concepts [68] which distinguish the Google Net. After that we introduce the Inception architecture [96] and at the end we talk about Batch Normalization [43].

#### Network in Network

The CNN is a extension of non-convolutional deep networks, by replacing each fully connected layers with a convolutional layer. Usually the first layer of convolution is a detection layer of edges, corners and other low level features. Then the second layer detects higher features like parts etc. For face classification, the second layer may learn features such as eyes, noses etc. Then the third layer may combine the eyes noses into a intact face. If we convolve the image with a aligned face classifier, there is no such cascade. The abstraction of each local patch is done through a linear classifier and a non-linear activation function. Which is definitely not a strong abstraction. Weak abstraction resolves the combinatorial explosion to some extent, but not as potent as strong abstraction. In [68] is presented a new convolutional approach (Network in Network or NiN) which tries to increase the abstraction degree. Intead of the classical convolutional model authors presented a *micro Multilayer Perception Network* (MPL) which creates a new level in the net called *mlpconv*.

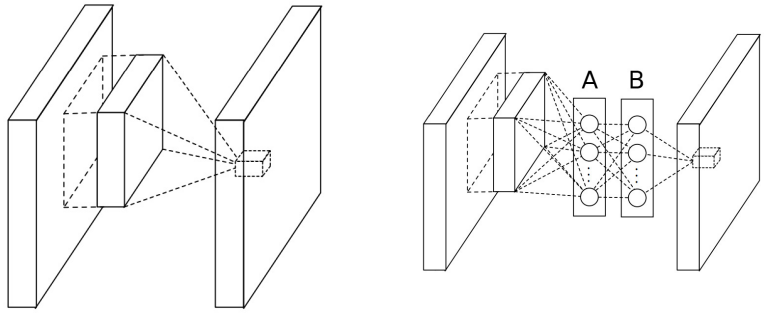



Figure 5.2: Linear convolutional level (*left*) and mplconv (*right*)

In this level the *Rectified Linear Unit* (ReLU) is used like activation function. This configuration was used inside GoogLeNet to obtain a dimen-

sionality reduction of the parameters and to increase the non-linearity degree between the levels.

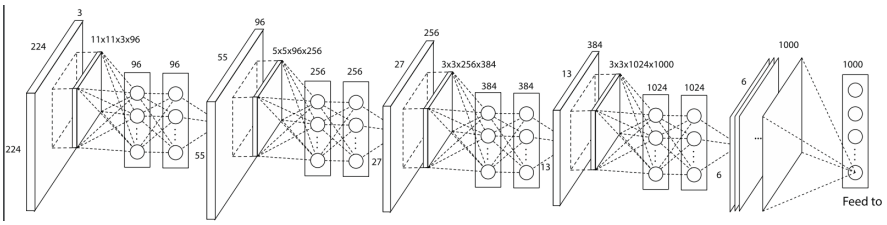



Figure 5.3: Network in Network example net

### Inception Architecture and GoogLeNet

The inception architecture [96] was structured for a quest aimed at reducing the computational burden of deep neural networks.
By now, Fall 2014, deep learning models were becoming extremely useful in categorizing the content of images and video frames. Most skeptics had given in that Deep Learning and neural nets came back to stay this time. Given the usefulness of these techniques, the internet giants like Google were very interested in efficient and large deployments of architectures on their server farms.
A google team has thought a lot about ways to reduce the computational burden of deep neural nets while obtaining state-of-art performance (on ImageNet, for example), or be able to keep the computational cost the same, while offering improved performance.
Result was the Inception module showed in Figure 5.4, which at a first glance is basically the parallel combination of 1×1, 3×3, and 5×5 convolutional filters. But the great insight of the inception module was the use of 1×1 convolutional blocks (NiN) to reduce the number of features before the expensive parallel blocks.

The inception architecture was used to generate GoogLeNet [96] which was used in ILSVRC14 competition[1]. Table 5.2 shows the net structure while Figure 5.5 shows the network topology in a graphical way.

[1]Imagenet Large Scale Visual Recognition Challenge which evaluates algorithms for object detection and image classification at large scale

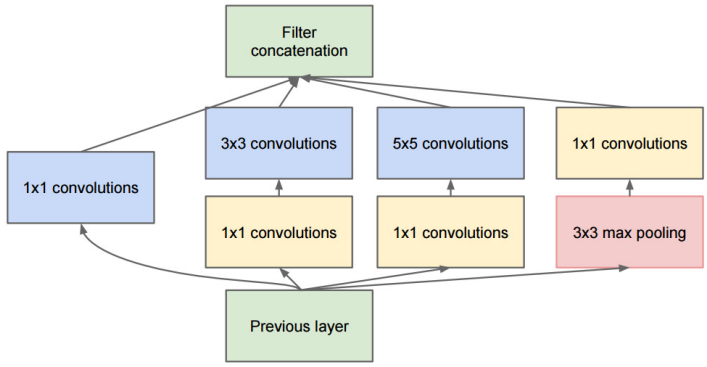



Figure 5.4: Inception module

| type | patch size/ stride | output | depth | # 1 × 1 | # 3×3 reduce | # 3×3 | # 5×5 reduce | # 5×5 | proj pool | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Table 5.2: GoogLeNet architecture

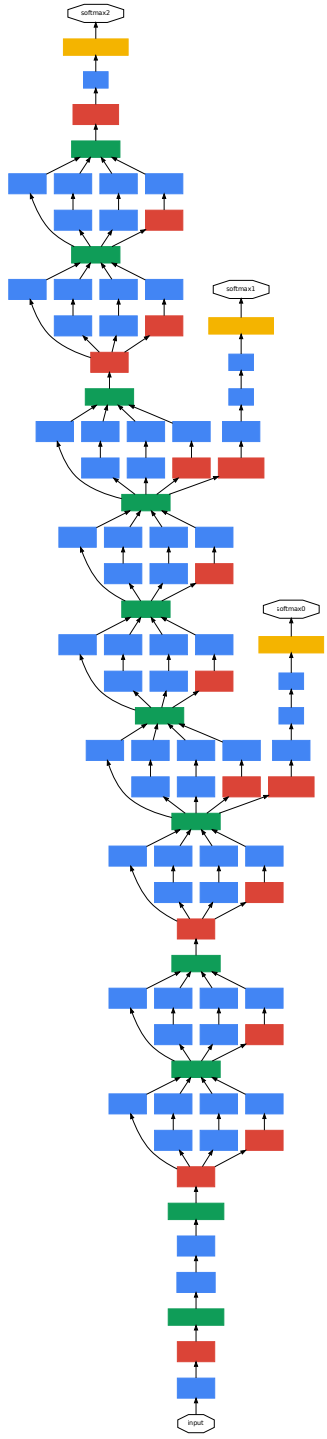



Figure 5.5: GoogLeNet architecture

#### Batch Normalization

Batch Normalization (BN) was introduced in [43]. It is a normalization which tries to eliminate the *internal covariance shift* phenomenon, i.e the distribution of the inputs change on various levels during network training. The approch is based on the *whitening technique* [79], which consists of a linear transformation of the inputs in such a way that they have zero mean, unit variance and that are not correlated between them. This technique allows a greater convergence speed and then reduces training time. The idea is to extend the whitening approach to the inputs at each net levels in such a way as to have a fixed distribution of the inputs themselves and a removal of the internal covariance shift.
This work is done working on the *mini-batch* distributions, which are small sample sets used during the training of the net and that are used in the gradient calculation which is used for the backpropagation step during network upgrade. The mini-batch technique speeds up the training because the feedforward operations and gradiend calculation can be vectorized. So the proposed solution is to normalize each feature map with mean and variance of each mini-batch. Therefore the BN approach consists in making the normalization an integral part of the network architecture.

## 5.3 The Proposed Method

In the proposed approach, differently from [27], we learn a vector quantizer separately from the CNN features, so to easily replace different and pretrained CNN networks for feature extraction, without need of retraining. Moreover, we propose to include Bloom filters into feature indexing structures to improve the speed of queries. Bloom filters act as gatekeepers that rule out immediately, with a very limited memory cost, if a query should be completely performed or if it can be avoided. The proposed data structure is very suitable for mobile and distributed applications.

### 5.3.1 Quantization Algorithm

The proposed approach is based on the *multi-k-means* method [23] (Chapter 4), which is an efficient method for mobile visual search based on a multiple assignment k-means hashing schema that obtained very good results, compared to Product Quantization [47], on the BIGANN dataset.

The first step of the method consists in learning a standard k-means dictionary with a small number of centroids (to maintain a low computational cost). Each centroid is associated to a bit of the hash code, that has thus length equal to the number of centroids. The bit is set to 1 if the feature is assigned to the centroid, 0 otherwise. A feature can be assigned to more than one centroid, and it is assigned to it if the distance from the centroid is less than the mean distance from all the centroids (Figure 5.6, top). Instead, in this work we select a fixed number $N$ of distances and we set to 1 all the bits associated to the smaller $N$ distances (Figure 5.6, bottom). In the following we refer to this method as MINx. This change has proven to be more efficient when coding CNN feature descriptors, that were used in the experiments.

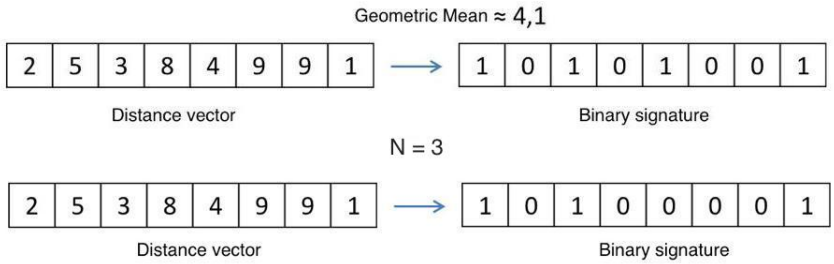



Figure 5.6: Binarization examples with a distance vector of 8 elements: (*top*) geometric mean (MEAN method); (*bottom*) smaller distances $N = 3$ (MINx method).

Approximate nearest neighbor retrieval of image descriptors is performed in two steps: in the first step is performed an exhaustive search over the binary codes using Hamming distances, to reduce negative effects of quantization errors. All the binary codes with Hamming distance below a threshold are selected. In the second step the candidate neighbors are ranked according to the distance computed using the full feature vector using *cosine distance* (Eq:4.6), that proved to be more effective than $L_2$ during the experiments.

### 5.3.2 Bloom Filter Algorithm

To improve search of feature vectors we also introduced the use of *Bloom filters* [7]. Typically this type of structures are used to speed up the answers in a key-based storage system (Figure 5.7).

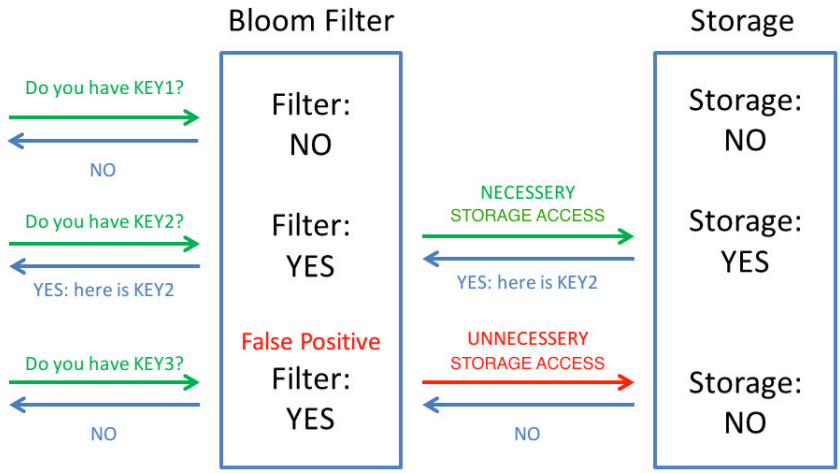



Figure 5.7: Memory accesses with Bloom filters

A Bloom filter is an efficient probabilistic data structure used to test if an element belongs to a set or not. This structure works with binary signatures, and can provide false positive response but not false negative and more elements are inserted into the structure and more high is the probability to obtain a false positive. To insert an element inside a Bloom filter we need to define $k$ hash functions which locate $k$ positions inside the array, setting them to 1.

To check the presence of an element inside a Bloom filter we compute the $k$ hash functions over the element and check the related positions inside the array. If just one bit of these positions is equal to 0 it means that the element is not present inside the array; if all the checked bits are equal to 1 it means that either the element is inside the array or we have a false positive. We used the method of [55] to create the $k$ functions from just two hash functions.

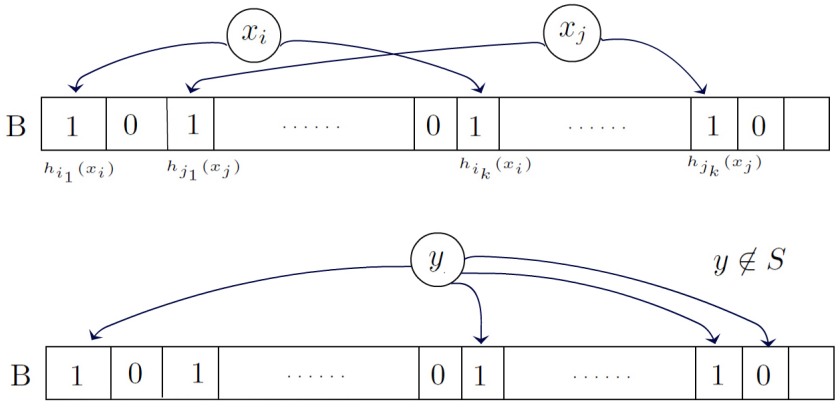



Figure 5.8: Example Bloom filter: (*top*) Insertion, (*bottom*) Search

A useful property of Bloom filter is that we can measure the presence of a false positive with probability:

$$(1 - e^{-kn/m})^k = (1 - p)^k = \epsilon \tag{5.1}$$

where $m$ is the bit number of the array, $n$ is the number of inserted items, $p$ is the probability that one position of the array is equal to 0, and $k$ is the number of hash functions. We can obtain the optimal value $k$ which minimizes false positive probability:

$$\tilde{k} = \ln 2(m/n). \tag{5.2}$$

Supposed that $p = 0.5$ we can write out $\epsilon$ like

$$\epsilon = 0.5^{\tilde{k}} = (0.6185)^{m/n} \tag{5.3}$$

So $n$ is strictly related to $m$, and in general $m = O(n)$ it is a good compromise.

Storing in the Bloom filter hash codes that are designed for ANN, as those of Sect. 5.3.1, results in a data structure that is similar, from a practical point of view, to distance-sensitive Bloom filters proposed in [54], where LSH functions are used as $k$ hash functions.

Table 5.3: Bloom Filter false positive probability related to $m$

| $m$ | $\epsilon$ | % |
|---|---|---|
| n | 0.61 | 61% |
| 2n | 0.38 | 38% |
| 5n | 0.09 | 9% |
| 10n | 0.008 | 0.8% |

### 5.3.3 Retrieval System

Our proposed retrieval system merges the methods introduced in 5.3.1 and 5.3.2. Regarding visual feature hashing we have applied the proposed method to CNNs features. Our system (Figure 5.9) provides a initial phase were descriptors are extracted from base images, binarized following one of the methods introduced in 5.3.1 and saved inside a data structure composed by a set of inverted files of hashes implementing an horizontal partition of data (allowing to distribute the database as "shard"), each one guarded by a Bloom filter. The hash code is also added to the Bloom filter of the corresponding inverted file.

During the search phase we extract the CNN descriptor from query images, we compute the hash code, and check the presence of the hash in the Bloom filters, each of which guard a subset of the base. If one of this Bloom filters gives a positive response (this means that we have a positive or a false positive match), all the hash codes within an Hamming distance threshold are used to select the full feature vector. This provides a great speedup in the approximate nearest neighbor retrieval since we consider only descriptors from base coded by a Bloom filter, and below the Hamming threshold value. For each resulting original CNN descriptor we compute the distance and we rearrange results to obtain a ranked list of vectors.

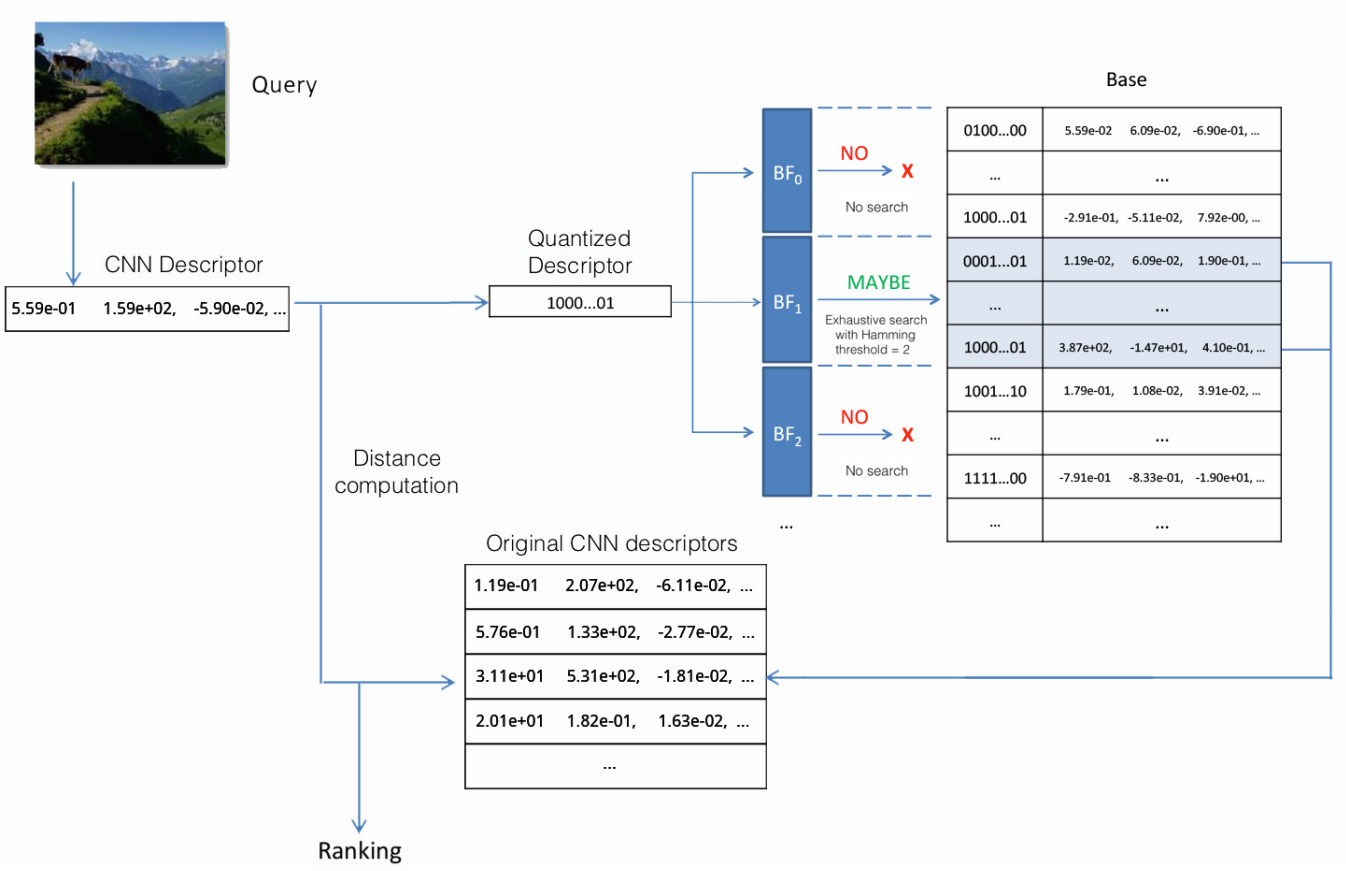



Figure 5.9: System overview.

## 5.4 Experimental Results

In this section we present an experimental validation performed on three standard image retrieval datasets.

### 5.4.1 Datasets and Configurations

We tested our system using three standard dataset typically used to evaluate image retrieval systems: INRIA Holidays [46], Oxford 5K [81] and Paris 6K [82]. For each dataset are given a number of query images, and the associated ground truth. INRIA Holidays (Figure: 4.3) is composed by 1,491 images, of which 500 are used as queries; Oxford 5K (Figure: 5.10) is composed by 5,062 images with 55 query images, and Paris 6K 5.11) is made of 6,412 images with 55 query images. We used the query images and ground truth provided for each dataset, adding 100,000 distractor images from *Flickr 100K* [81].
When testing on a dataset training is performed using the other two datasets. Features have been hashed to 64 bits binary codes, a length that has proved to be the best compromise between compactness and representativeness. Other parameters used for hashing were:
– number of $N$ nearest distances used in the hash code computation ($N \in \{6, 10, 16, 32, 40\}$);
– Hamming distance threshold $\in \{2, 4, 6, 10, 16, 30\}$.

For the evaluation we used the Mean Average Precision (MAP) metric defined as follow:

given the *Precision at k* value

$$P@k := \frac{\#imgRelevant\ with\ rank\ \leq k}{k} \tag{5.4}$$

.

which is the ratio between the relevant images in the first k position and k the number of retrieved images, we can define the *Average Precision* value for a single query $q$

$$AP(q) := \frac{1}{R} \sum_{i=1}^{R} P@k_i \tag{5.5}$$

and compute the mean over $Q$ queries with:

$$MAP(Q) := \frac{1}{|Q|} \sum_{j=1}^{|Q|} AP(q_j) \tag{5.6}$$

The CNN features used in the following experiments have been extracted using the 1024d average pooling layer of GoogLeNet with Batch Normalization (Inception BN) [95] and using the FC7 layer of VGG [91] used in [101].

Figure 5.10: Sample images from Oxford


Figure 5.11: Sample images from Paris

### 5.4.2 Results on VGG16

VGG16 5.2.1 is the first Neural Network used in our experiments. We evaluate the effects of the method parameters, comparing the proposed hashing approach (MINx) with the original method [23] (MEAN) and a baseline that uses no hashing. Figure 5.12, 5.13 and 5.14 show results over Inria Holidays, Paris Buildings and Oxford Buildings with feature quantization set to 64 bits. Tables 5.4, 5.5 and 5.6 show instead numerical results.

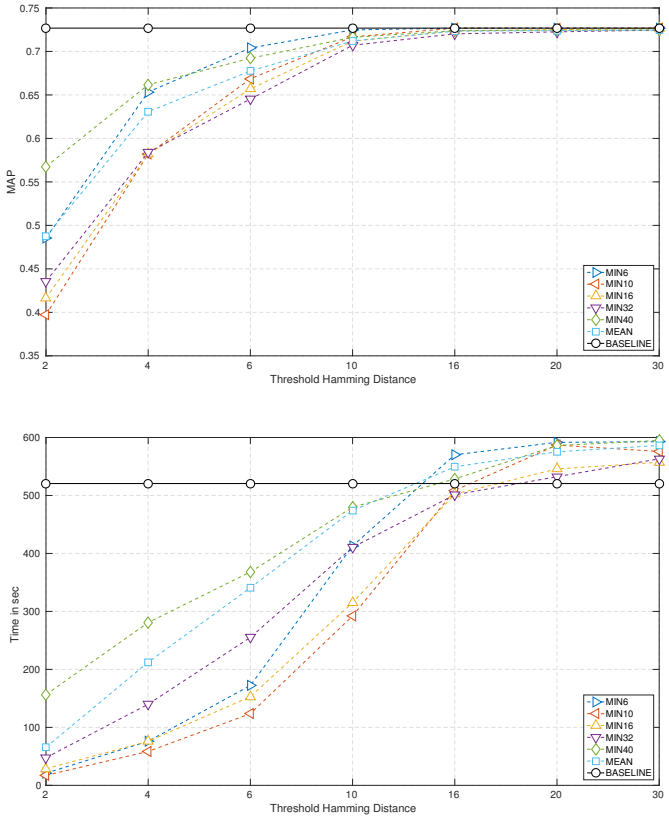



Figure 5.12: Map (*top*) and Execution Time (*bottom*) for Inria Holidays

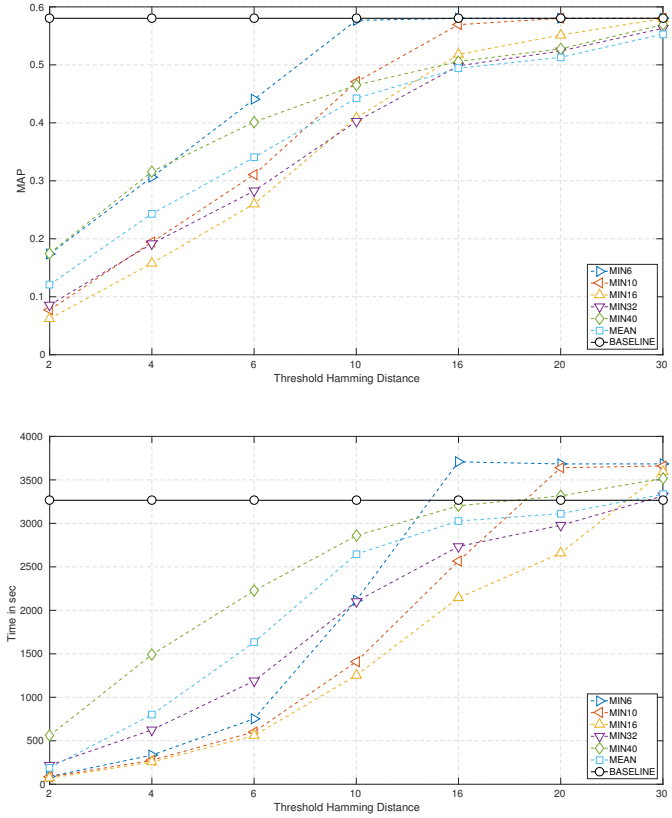



Figure 5.13: Map (*top*) and Execution Time (*bottom*) for Paris Buildings

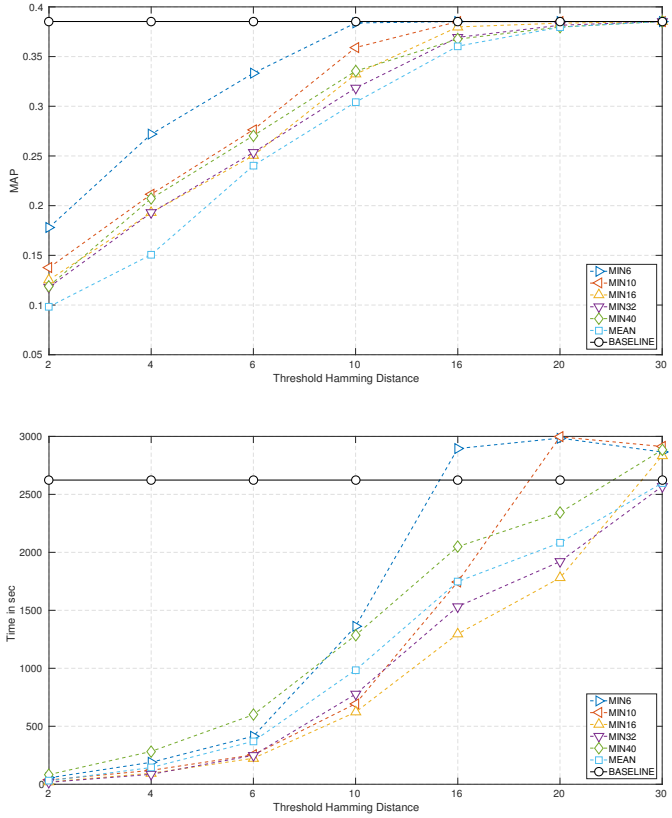



Figure 5.14: Map (*top*) and Execution Time (*bottom*) for Oxford Buildings

Table 5.4: VGG16 numerical Results for Inria Holidays dataset

| | MAP | | | | | | | Tempo (sec) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **2** | **4** | **6** | **10** | **16** | **20** | **30** | **2** | **4** | **6** | **10** | **16** | **20** | **30** |
| **MIN6** | 0.48546 | 0.65339 | 0.70396 | 0.72478 | 0.72684 | 0.72684 | 0.72684 | 22.1838 | 76.6519 | 173.376 | 419.146 | 560.489 | 567.853 | 560.387 |
| **MIN10** | 0.39720 | 0.58187 | 0.66859 | 0.71677 | 0.72680 | 0.72684 | 0.72684 | 17.4546 | 56.2703 | 121.757 | 283.830 | 503.768 | 611.230 | 591.357 |
| **MIN16** | 0.41639 | 0.58187 | 0.65692 | 0.71194 | 0.72355 | 0.72501 | 0.72684 | 28.6139 | 75.8825 | 152.928 | 315.155 | 502.210 | 545.864 | 557.215 |
| **MIN32** | 0.43559 | 0.58392 | 0.64526 | 0.70711 | 0.72023 | 0.72254 | 0.72456 | 51.2062 | 149.971 | 266.948 | 422.159 | 520.333 | 558.560 | 590.420 |
| **MIN40** | 0.56763 | 0.66170 | 0.69221 | 0.71622 | 0.72389 | 0.72460 | 0.72484 | 169.92 | 299.816 | 391.363 | 482.617 | 542.281 | 558.346 | 570.134 |
| **MEAN** | 0.48734 | 0.63076 | 0.67753 | 0.71209 | 0.72340 | 0.72389 | 0.72458 | 66.5149 | 211.404 | 333.677 | 462.711 | 531.258 | 550.540 | 564.158 |
| **Baseline** | 0.72684 | | | | | | | 520.566 | | | | | | |

Table 5.5: VGG16 numerical Results for Paris Buildings dataset

| | MAP | | | | | | | Tempo (sec) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **2** | **4** | **6** | **10** | **16** | **20** | **30** | **2** | **4** | **6** | **10** | **16** | **20** | **30** |
| **MIN6** | 0.173853 | 0.306208 | 0.440519 | 0.576764 | 0.580334 | 0.580334 | 0.580334 | 89.157 | 335.23 | 753.087 | 2114.33 | 3707.3 | 3683.34 | 3683.83 |
| **MIN10** | 0.0774561 | 0.194167 | 0.310534 | 0.470674 | 0.569312 | 0.580334 | 0.580334 | 84.0938 | 276.385 | 601.77 | 1409.53 | 2567.37 | 3639.17 | 3660.7 |
| **MIN16** | 0.0621038 | 0.157804 | 0.260079 | 0.408719 | 0.518021 | 0.551434 | 0.579773 | 68.8062 | 255.445 | 559.012 | 1252.52 | 2142.93 | 2657.14 | 3593.88 |
| **MIN32** | 0.0857685 | 0.191491 | 0.282481 | 0.402354 | 0.498396 | 0.523985 | 0.563083 | 219.181 | 625.117 | 1187.47 | 2102.64 | 2733.88 | 2977.03 | 3315.83 |
| **MIN40** | 0.174999 | 0.315817 | 0.401305 | 0.465136 | 0.505779 | 0.527712 | 0.569156 | 560.244 | 1490.26 | 2225.62 | 2857.88 | 3202.4 | 3315.8 | 3516.39 |
| **MEAN** | 0.12088 | 0.243217 | 0.340479 | 0.442362 | 0.494162 | 0.512963 | 0.552908 | 188.535 | 800.487 | 1632.92 | 2646.11 | 3027.26 | 3112.33 | 3334.68 |
| **Baseline** | 0.580334 | | | | | | | 3265.54 | | | | | | |

Table 5.6: VGG16 numerical Results for Oxford Buildings dataset

| | MAP | | | | | | | Tempo (sec) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **2** | **4** | **6** | **10** | **16** | **20** | **30** | **2** | **4** | **6** | **10** | **16** | **20** | **30** |
| **MIN6** | 0.177951 | 0.271898 | 0.333289 | 0.383842 | 0.385201 | 0.385201 | 0.385201 | 55.2494 | 189.145 | 414.549 | 1361.07 | 2895.35 | 2985.01 | 2865.13 |
| **MIN10** | 0.137537 | 0.21141 | 0.276424 | 0.359071 | 0.385223 | 0.385201 | 0.385201 | 36.9503 | 119.832 | 253.718 | 694.655 | 1746.08 | 2996.33 | 2912.88 |
| **MIN16** | 0.125302 | 0.192899 | 0.250409 | 0.33225 | 0.379823 | 0.383617 | 0.385207 | 20.2284 | 94.5917 | 224.075 | 621.784 | 1296.83 | 1779.63 | 2833.26 |
| **MIN32** | 0.117939 | 0.192885 | 0.253495 | 0.318336 | 0.369331 | 0.381529 | 0.38525 | 17.1714 | 85.4339 | 249.617 | 777.084 | 1532.52 | 1920.31 | 2569.46 |
| **MIN40** | 0.119043 | 0.207129 | 0.270422 | 0.335373 | 0.367562 | 0.379725 | 0.385272 | 83.0668 | 283.074 | 599.718 | 1284.81 | 2049.09 | 2342.8 | 2883.16 |
| **MEAN** | 0.0982769 | 0.150642 | 0.24033 | 0.304124 | 0.36046 | 0.379531 | 0.385462 | 29.6716 | 144.074 | 370.738 | 985.035 | 1747.7 | 2082.61 | 2598.34 |
| **Baseline** | 0.385272 | | | | | | | 2623.38 | | | | | | |

As expected the uncompressed features perform better than MIN6 and MIN40 which provide anyway good results. To undertand better this beaviour we calculated mean distances between features and their binarized signatures[2]. Table 5.7 shows these distances and we can see how for hight values of n (MINn) we obtain a worst reconstruction of the features.

Table 5.7: Mean distances between features and binarized signatures. Distance is calculated using the cosine formula, so an high value correspond to a greater similarity

| | **Inria** | **Paris** | **Oxford** |
|---|---|---|---|
| **MIN6** | 0.7153 | 0.7363 | 0.7502 |
| **MIN10** | 0.6852 | 0.7147 | 0.7320 |
| **MIN16** | 0.6528 | 0.6900 | 0.7095 |
| **MIN32** | 0.5905 | 0.6486 | 0.6650 |
| **MIN40** | 0.5687 | 0.6324 | 0.6464 |
| **MEAN** | 0.5782 | 0.6395 | 0.6581 |

This means that centroids computed by k-means are very scattered in the multi-dimensional space and that a signature with a low number of bits set to 1 is more accurate (Fig 5.15).
Another interesting aspect is the distribution of the signatures. This value indicates how many image descriptors are quantized in the same way. For simplicity we show only distributions related to MIN6 and MIN40 for Inria Holidays dataset in Figure 5.16 (for the other dataset the behavior is almost the same). We ca see how MIN6 approach has a more uniform distribution of the signatures while MIN40 is more concentrated. This means that MIN40 approach works well also with low hamming thresholds because we analyze an higher signatures subspace and at the end of the story we have a MAP with high values but at the same time we have also high values for the searching time. For MIN6 instead we have a good MAP but lower values for searching time respect to MIN40 and this is a proof of the goodness for the quantization approach with a low values for n.

[2]we consider the center of gravity of centroids

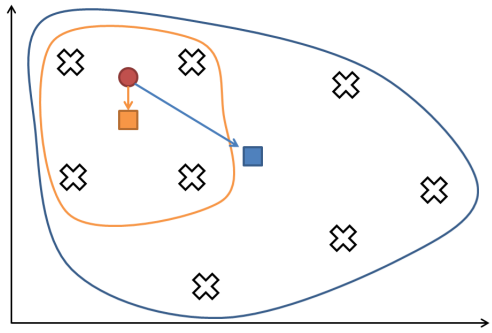
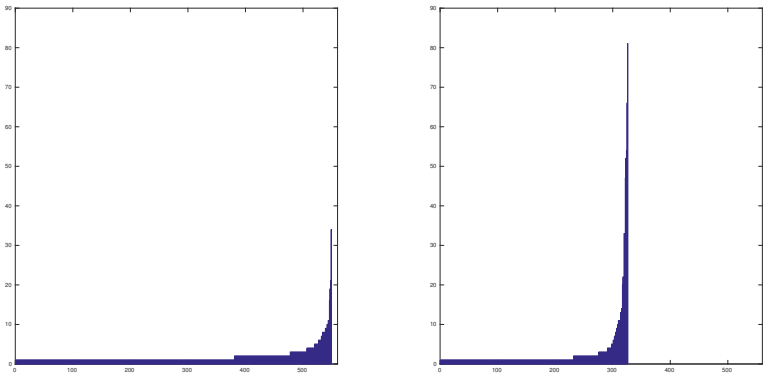


Figure 5.15: Example wich shows the difference during a feature (red) reconstruction for MIN4 (orange) and MIN8 (blue). Crosses are codebook centroids.



(a) MIN6 (b) MIN40

Figure 5.16: Histogram of signatures distribution for MIN6 (a) and MIN40 (b).

### 5.4.3 Results on Inception BN

Experiments executed in 5.4.2 have shown that features extracted using the FC7 layer of VGG are not performing in terms of time search. This is why it has been necessary to use Inception BN 5.2.2, a more lightweight neural network.
Tabel 5.8 shows a general comparison. We can observe how the values are more high for the Inception Bn Network and furthermore there is not much difference between MIN6 and MIN40. This means that reconstruction in Inception BN is more accurate respect to VGG16 and it is high along different values of n.

Table 5.8: Mean distances between features and their binarized signatures computed for each quantization approach, each Neural Network and each dataset. Distance is calculated using the cosine formula, so an high value correspond to a greater similarity

| | **VGG16** | | | **Inception BN** | | |
|---|---|---|---|---|---|---|
| | **Inria** | **Paris** | **Oxford** | **Inria** | **Paris** | **Oxford** |
| **MIN6** | 0.7153 | 0.7363 | 0.7502 | 0.7935 | 0.8204 | 0.8175 |
| **MIN10** | 0.6852 | 0.7147 | 0.7320 | 0.7757 | 0.8088 | 0.8067 |
| **MIN16** | 0.6528 | 0.6900 | 0.7095 | 0.7581 | 0.7962 | 0.7950 |
| **MIN32** | 0.5905 | 0.6486 | 0.6650 | 0.7313 | 0.7737 | 0.7719 |
| **MIN40** | 0.5687 | 0.6324 | 0.6464 | 0.7224 | 0.7654 | 0.7625 |
| **MEAN** | 0.5782 | 0.6395 | 0.6581 | 0.7300 | 0.7782 | 0.7776 |

Figures 5.17, 5.18 and 5.19 show experimental results on Inria holidays, Paris Buildings and Oxford Buildings dataset. We can note some differences respect to VGG16 regardings MAP and Time search. Time improvement is due to the dimensionality of the features: VGG16 provides float features of 4096 length while Inception BN provides features of 1024 length. We can see also an improvment respecto to MAP related to baseline. This means that the 1024 space dimension generated by Inception BN is more representative than the VGG16 one.
Interestingly that best results for MAP are given by MIN6 while worst result is given by MEAN and that, respect to VGG16, MIN40 does not represent a good approach for MAP computing with low hamming distance treshold. Tables 5.10, 5.11 and 5.12 show all numerical results.

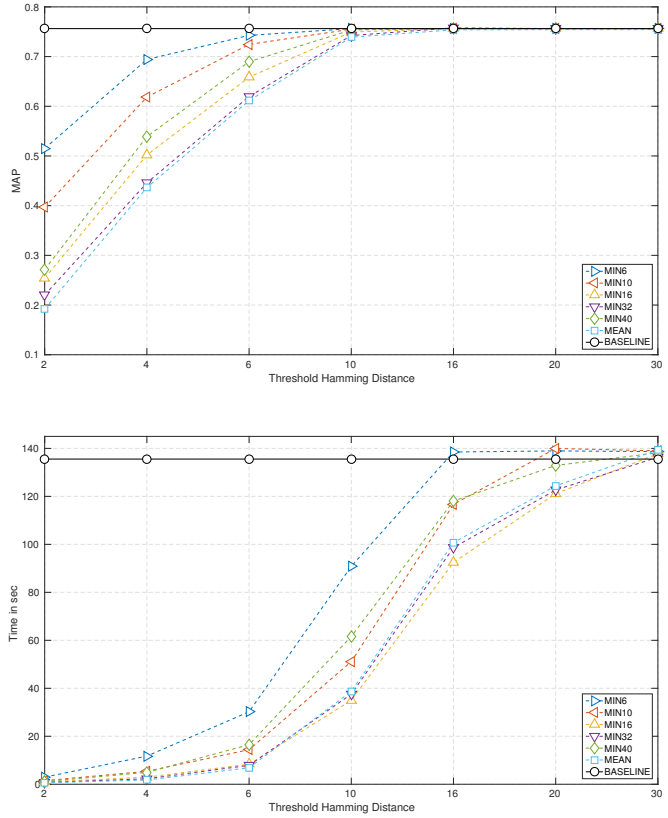



Figure 5.17: Map (*top*) and Execution Time (*bottom*) for Inria Holidays

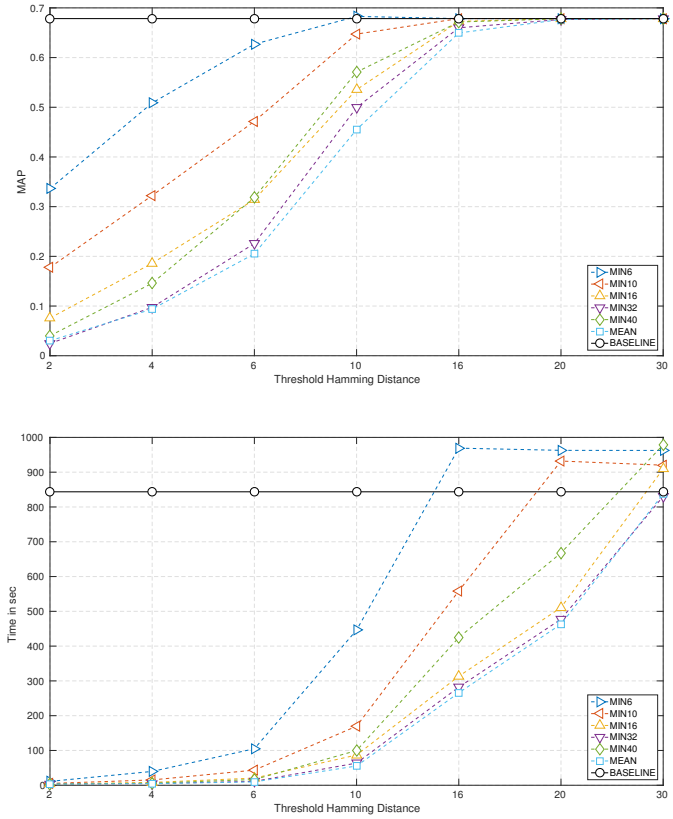



Figure 5.18: Map (*top*) and Execution Time (*bottom*) for Paris Buildings

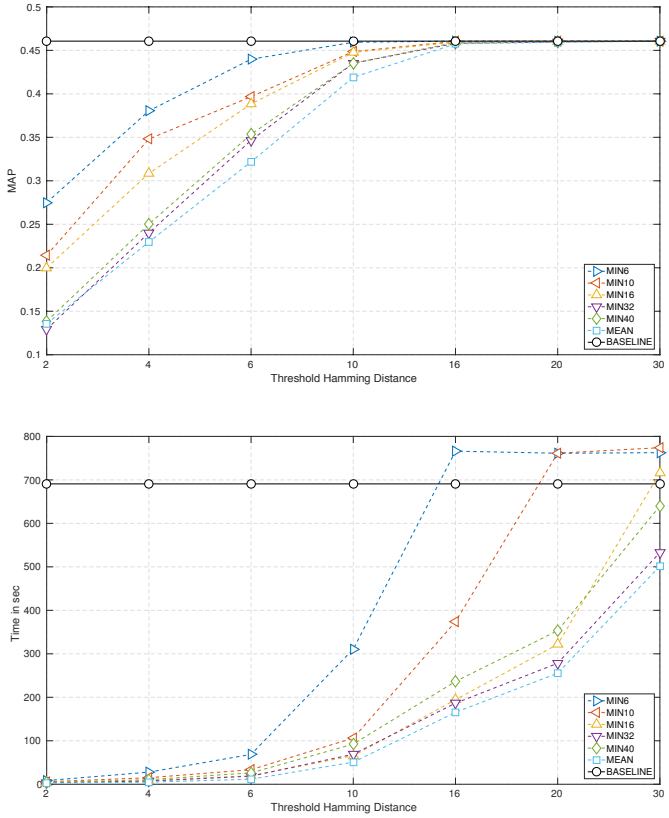



Figure 5.19: Map (*top*) and Execution Time (*bottom*) for Oxford Buildings

In the last experiment we evaluated the effects of the method parameters, comparing the proposed hashing approach (MINx) with the original method of [23] (MEAN), a baseline that uses no hashing, and several state-of-the-art methods, among which the recent UTH method [65]. The best combinations of MINx are reported, compared on the three datasets in terms of MAP. As expected the uncompressed features perform better, but the MIN6 setup, with an Hamming distance $\geq 6$ has comparable results, and greatly outperforms any state-of-the-art hashing method.

Table 5.9: MAP results on Holidays, Oxford 5K and Paris 6K datasets. The proposed MINx method outperforms all the current state-of-the-art methods. All hashes are 64 bit long.

| **Method** | **Holidays** | **Oxford 5K** | **Paris 6K** |
|---|---|---|---|
| ITQ [33] | 53.68 | 23.00 | - |
| BPBC [32] | 38.10 | 22.51 | - |
| PCAHash [33] | 52.80 | 23.90 | - |
| LSH [18] | 43.08 | 23.91 | - |
| SKLSH [83] | 24.09 | 13.39 | - |
| SH [104] | 52.22 | 23.24 | - |
| SRBM [9] | 51.58 | 21.23 | - |
| UTH [65] | 57.10 | 24.00 | - |
| **MIN6 Thr. 10** | **75.62** | 45.93 | **68.28** |
| **MIN6 Thr. 16** | **75.62** | **46.06** | 67.84 |
| **MIN10 Thr. 10** | 75.51 | 44.86 | 64.72 |
| **MIN10 Thr. 16** | **75.62** | 46.03 | 67.84 |
| **MEAN [23] Thr. 10** | 73.91 | 41.89 | 45.50 |
| **Baseline** | 75.62 | 46.06 | 67.84 |

Table 5.10: Inception BN numerical Results for Inria Holidays dataset

| | MAP | | | | | | | Tempo (sec) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **2** | **4** | **6** | **10** | **16** | **20** | **30** | **2** | **4** | **6** | **10** | **16** | **20** | **30** |
| **MIN6** | 0.51506 | 0.69421 | 0.74263 | 0.75628 | 0.75628 | 0.75628 | 0.75628 | 2.963 | 11.724 | 30.228 | 90.839 | 138.497 | 138.944 | 138.666 |
| **MIN10** | 0.39690 | 0.61842 | 0.72449 | 0.75510 | 0.75628 | 0.75628 | 0.75628 | 1.493 | 5.401 | 14.419 | 51.123 | 116.661 | 139.962 | 138.988 |
| **MIN16** | 0.25424 | 0.50181 | 0.65851 | 0.74932 | 0.75634 | 0.75627 | 0.75628 | 0.959 | 2.947 | 8.357 | 34.975 | 92.452 | 121.119 | 137.780 |
| **MIN32** | 0.22012 | 0.44575 | 0.61967 | 0.74234 | 0.75778 | 0.75628 | 0.75628 | 0.722 | 2.283 | 7.869 | 37.542 | 98.646 | 122.668 | 136.324 |
| **MIN40** | 0.27121 | 0.53897 | 0.68969 | 0.75360 | 0.75636 | 0.75628 | 0.75628 | 1.109 | 4.963 | 16.496 | 61.593 | 118.037 | 132.900 | 138.170 |
| **MEAN** | 0.19229 | 0.43638 | 0.61138 | 0.73914 | 0.75380 | 0.75428 | 0.75428 | 0.605 | 1.777 | 6.861 | 38.701 | 100.717 | 124.373 | 139.385 |
| **Baseline** | 0.75628 | | | | | | | 136.456 | | | | | | |

Table 5.11: Inception BN numerical Results for Paris Buildings dataset

| | MAP | | | | | | | Tempo (sec) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **2** | **4** | **6** | **10** | **16** | **20** | **30** | **2** | **4** | **6** | **10** | **16** | **20** | **30** |
| **MIN6** | 0.336309 | 0.50876 | 0.626968 | 0.682864 | 0.678424 | 0.678424 | 0.678424 | 11.1328 | 39.7277 | 104.978 | 446.163 | 969.086 | 962.748 | 962.51 |
| **MIN10** | 0.17771 | 0.322388 | 0.471783 | 0.647291 | 0.678409 | 0.678424 | 0.678424 | 5.19482 | 15.3159 | 43.5377 | 169.808 | 558.622 | 932.004 | 920.107 |
| **MIN16** | 0.075801 | 0.186052 | 0.314223 | 0.535456 | 0.672823 | 0.678376 | 0.678412 | 3.74478 | 8.18897 | 20.5611 | 86.9614 | 313.326 | 510.891 | 909.636 |
| **MIN32** | 0.024746 | 0.097367 | 0.225794 | 0.499426 | 0.659921 | 0.676726 | 0.678392 | 3.22912 | 4.86233 | 11.2225 | 63.749 | 282.084 | 477.427 | 830.361 |
| **MIN40** | 0.040081 | 0.145862 | 0.318565 | 0.570833 | 0.671578 | 0.678191 | 0.678431 | 3.36214 | 5.996 | 16.4596 | 99.8146 | 424.567 | 667.834 | 978.704 |
| **MEAN** | 0.030127 | 0.093854 | 0.205108 | 0.455086 | 0.649518 | 0.676606 | 0.678419 | 3.32751 | 4.63821 | 9.57446 | 55.4068 | 265.784 | 463.114 | 837.478 |
| **Baseline** | 0.678424 | | | | | | | 843.598 | | | | | | |

Table 5.12: Inception BN numerical Results for Oxford Buildings dataset

| | MAP | | | | | | | Tempo (sec) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **2** | **4** | **6** | **10** | **16** | **20** | **30** | **2** | **4** | **6** | **10** | **16** | **20** | **30** |
| **MIN6** | 0.274981 | 0.380422 | 0.439791 | 0.459339 | 0.460614 | 0.460614 | 0.460614 | 8.40198 | 27.9177 | 68.6569 | 310.741 | 766.072 | 761.215 | 762.646 |
| **MIN10** | 0.214395 | 0.348057 | 0.396712 | 0.448613 | 0.460359 | 0.460614 | 0.460614 | 5.66545 | 15.0693 | 33.4678 | 106.29 | 373.755 | 761.433 | 774.001 |
| **MIN16** | 0.199912 | 0.308418 | 0.388412 | 0.447595 | 0.459411 | 0.460182 | 0.460616 | 3.77203 | 8.44777 | 19.5041 | 65.5873 | 194.385 | 321.81 | 716.188 |
| **MIN32** | 0.129356 | 0.240127 | 0.34654 | 0.435165 | 0.458162 | 0.459625 | 0.460587 | 3.27494 | 7.26284 | 18.6175 | 69.0486 | 186.43 | 278.267 | 532.951 |
| **MIN40** | 0.138421 | 0.250189 | 0.353441 | 0.435194 | 0.459095 | 0.460184 | 0.460616 | 3.95866 | 10.5687 | 26.2706 | 92.3946 | 236.457 | 353.24 | 639.706 |
| **MEAN** | 0.135638 | 0.229622 | 0.321836 | 0.418972 | 0.457484 | 0.459269 | 0.460536 | 2.82485 | 4.79286 | 11.6194 | 50.7517 | 165.334 | 255.416 | 501.609 |
| **Baseline** | 0.460614 | | | | | | | 690.80 | | | | | | |

### 5.4.4 Results on Bloom Filter

The second part of the experiments has involved the application of the Bloom Filters 5.3.2 . We evaluated a use case in which a database of images is queried with a large number of images that do not belong to it. Hash codes have been computed with different variants of the proposed hashing method using the Inception BN features. The most useful context in which a Bloom Filter can be used is a large scale application. This is why experiments were performed over Paris Buildings dataset which is the largest one.
The database contains the Paris 6K images, and it is queried with all the query images of Paris 6K and all the 100,000 distractor images 5.4.1.

We analized the three most interesting configurations:

- MIN6 with maximun hamming distance 10
- MIN10 with maximun hamming distance 16
- MIN16 with maximun hamming distance 20.

For each of these configurations we excecuted the methods with one Bloom Filter, two bloom Filters and five Bloom Filters with size factor equal to $2n$ and $5n$ where $n$ is the number of stored elements. MAP values and query time in seconds are reported in Tab 5.13 and 5.14.
The speedup obtained is about $2\times$ since a large number of distractor queries are immediately stopped by the system; the slight increase in MAP is due to the beneficial effect of elimination of some false positives of the Paris 6K images, that do not result in retrieving wrong dataset images. We can see this effect also in Table 5.15; when we have a ratio equal to one means that all the queries rejected by the bloom Filter are distractors.

Table 5.13: MAP obtained on Paris 6K with the proposed system with different numbers of Bloom filters (1, 2 and 5) and with a baseline without filters. $2n$ and $5n$ are the size of the filters, where $n$ is the number of stored elements. *Thr.* is the maximum Hamming distance used for hash code retrieval.

| | **MAP** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **No BF** | **1 BF** | | **2 BF** | | **5 BF** | |
| | | **2n** | **5n** | **2n** | **5n** | **2n** | **5n** |
| **MIN6 Thr. 10** | 0.6828 | 0.6828 | 0.6828 | 0.6899 | 0.6911 | 0.6828 | 0.6855 |
| **MIN10 Thr. 16** | 0.6784 | 0.6784 | 0.6784 | 0.6735 | 0.6860 | 0.6784 | 0.6889 |
| **MIN16 thr. 20** | 0.6783 | 0.6783 | 0.6783 | 0.6859 | 0.6931 | 0.6791 | 0.6709 |

Table 5.14: Time (secs.) obtained on Paris 6K with the proposed system with different numbers of Bloom filters (1, 2 and 5) and with a baseline without filters. $2n$ and $5n$ are the size of the filters, where $n$ is the number of stored elements. *Thr.* is the maximum Hamming distance used for hash code retrieval.

| | **Time (secs)** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **No BF** | **1 BF** | | **2 BF** | | **5 BF** | |
| | | **2n** | **5n** | **2n** | **5n** | **2n** | **5n** |
| **MIN6 Thr. 10** | 479.15 | 332.80 | 236.65 | 391.67 | 276.08 | 483.94 | 398.38 |
| **MIN10 Thr. 16** | 580.75 | 342.89 | 224.94 | 463.30 | 285.26 | 550.62 | 446.15 |
| **MIN16 Thr. 20** | 546.23 | 300.12 | 160.727 | 410.18 | 228.28 | 524.91 | 416.99 |

Table 5.15: The ratios represent the number of distractors rejected by the Bloom Filter over the total rejections number. We report different numbers of Bloom filters (1, 2 and 5). $2n$ and $5n$ are the size of the filters, where $n$ is the number of stored elements. *Thr.* is the maximum Hamming distance used for hash code retrieval.

| | **1 BF** | | **2 BF** | | | |
|---|---|---|---|---|---|---|
| | **2n** | **5n** | **2n** | | **5n** | |
| | $BF_0$ | $BF_0$ | $BF_0$ | $BF_1$ | $BF_0$ | $BF_1$ |
| **MIN6** | 25972 | 39888 | 11860 | 14367 | 36770 | 35144 |
| **Thr. 10** | 25972 | 39888 | 11865 | 14372 | 36780 | 35155 |
| **MIN10** | 39175 | 58545 | 19285 | 19632 | 48226 | 49746 |
| **Thr. 16** | 39175 | 58545 | 19287 | 19636 | 48238 | 49758 |
| **MIN16** | 44747 | 70238 | 19409 | 19409 | 56878 | 54400 |
| **Thr. 20** | 44747 | 70238 | 19412 | 19412 | 56890 | 54409 |

| | **5 BF** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **2n** | | | | | **5n** | | | | |
| | $BF_0$ | $BF_1$ | $BF_2$ | $BF_3$ | $BF_4$ | $BF_0$ | $BF_1$ | $BF_2$ | $BF_3$ | $BF_4$ |
| **MIN6** | 414 | 1759 | 777 | 963 | 1118 | 15004 | 17954 | 18343 | 16836 | 18619 |
| **Thr. 10** | 414 | 1759 | 777 | 963 | 1118 | 15011 | 17962 | 18351 | 16845 | 18622 |
| **MIN10** | 791 | 1109 | 363 | 514 | 567 | 1904 | 19312 | 18839 | 19751 | 19423 |
| **Thr. 16** | 791 | 1109 | 363 | 514 | 567 | 1909 | 19317 | 18846 | 19758 | 19433 |
| **MIN16** | 1726 | 344 | 1405 | 1785 | 2250 | 19880 | 24365 | 19939 | 20008 | 24312 |
| **Thr. 20** | 1726 | 344 | 1406 | 1785 | 2252 | 19892 | 24371 | 19951 | 20017 | 24319 |

In the third experiment we evaluate a more challenging and large scale experiment: three datasets composed by distractor images and Holidays, Paris 6K and Oxford 5K images are built and stored in the proposed data structure. The standard dataset query images are then used to query the system. In this case we have used 10 filters to "shard" the database that, thus, can be distributed. Tab. 5.16 reports the results in terms of MAP and time (secs.). For semplicity we report only results for MIN6 and Hamming threshold 10 for Paris 6K and Holidays, while for Oxford we report MIN10 with threshold 16. Using the proposed method results in speed improvement of 2× while improving MAP, except the Holidays dataset that only improves speed. The size of each Bloom filter is $\sim 6 - 62$ KB, allowing the use of the method in a mobile environment.

Table 5.16: MAP+time (secs.) obtained on Paris 6K, Oxford 5K and Holidays with the proposed system with 10 Bloom filters of varying size and with a baseline without filters. The database contains 100,000 distractor + the database images of each dataset.

| # BF | Paris 6K | Oxford 5K | Holidays |
|---|---|---|---|
| No BF | 58.52 | 42.05 | **59.56** |
| | 3.35 | 2.35 | 56.71 |
| 10 BF | 59.44 | 41.45 | 52.15 |
| 10n | 2.66 | 1.95 | 47.21 |
| 10 BF | 61.21 | 42.01 | 42.36 |
| 20n | 1.93 | 1.53 | 36.36 |
| 10 BF | 62.82 | **42.29** | 39.26 |
| 30n | 1.43 | 1.25 | 31.37 |
| 10 BF | **63.52** | 42.24 | 34.29 |
| 50n | **1.21** | **1.11** | **26.59** |

## 5.5 Conclusion

We have presented a simple and effective method for CNN feature hashing that outperforms current state-of-the-art methods on standard datasets. A novel indexing structure, where Bloom filters are used as gatekeepers to inverted files storing the hash codes, results in a 2× speedup for ANN, without loss in MAP.

# Chapter 6

# Conclusions and Perspectives

### 6.0.1 Conclusion

In this thesis we have investigated the content-based image retrieval problem related to the large scale information retrieval which is very relevant due to the emerge of social networks and mostly the creation of web-scale image databases. We have proposed and presented a new version of the k-means based hashing schema called multi-k-means. This approach provides 4 variants: *mk-means-$t_1$*, *m-k-means-$t_2$*, *m-k-means-$n_1$* and *m-k-means-$n_2$* and uses a small number of centroids to guarantee a low computational cost and to result in a compact quantizer. These characteristics are achieved thanks to the association of the centroids to the bits of the hash code, that greatly reduce the need of a large number of centroids to produce a code of the needed length. Another advantage of the method is that it has no parameters in its *m-k-means-$t_1$* and *m-k-means-$t_2$* variants, and only one parameter for the other two variants; anyway, as shown by the experiments, it is quite robust to variations of such parameter, as well as hash code length.
Our compact hash signature is able to represent high dimensional visual features obtaining a very high efficiency in approximate nearest neighbor (ANN) retrieval, both on local and global visual features. This characteristic stems from the multiple-assignment strategy, that reduces the need of multi probe strategy to retrieve hash codes that differ by few bits, typically due to quantization errors, and results in better approximated nearest neighbour estimation using Hamming distance.
We presented also a recursive data structure which provides, in conjunction

with our compact hash signature, best results in terms of memory requirements and search time compared to the traditional approaches represented by hash table and binary tree.
The method has been tested also on large scale datasets of engineered (SIFT and GIST) and learned (deep CNN) features, obtaining results that outperform or are comparable to more complex state-of-the-art approaches. Finally, we noted that the *m-k-means-*$n_1$ variant typically performs better than *m-k-means-*$t_1$ when dealing with modern CNN features.
Correlated to our approach we also presented a new system which uses inside it, in conjunction with CNN features, a novel indexing structure, where we used our compact hash signatures and an efficient probabilistic data structures called Bloom filters as gatekeepers. This structure helped us to have an inverted file storing and to obtain a higher compression ratio that has allowed us a more quick search. This approach has shown to be very efficient and in our experiments it results in a 2X speedup for ANN, without loss in MAP.

### 6.0.2 Perspectives

In this work we have shown a new kind of quantization method for features and how these compact hash signatures can be used inside a system which makes use of a new kind of probabilistic data structure.
To conclude, we would like to share some ideas about possible future improvements; we are interested in "exploring" a new kind of features and their impact with our method, especially we are interested in an extensive evaluation of our system on bigger real dataset. We are also interested in an extension of the system in the video field with particular focus on video surveillance applications and last but not least we are interested in a development of a real mobile system based on our retrieval approach.

# Appendix A

# Publications

## International Conferences and Workshops

1. G. D'Amico, S. Ercoli, and A. Del Bimbo, "A Framework for Itinerary Personalization in Cultural Tourism of Smart Cities" AI* HCI@ AI* IA. 2013.
2. S. Ercoli, M. Bertini, and A. Del Bimbo, "Compact hash codes and data structures for efficient mobile visual search". Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on. IEEE, 2015
3. A. Salvi, S. Ercoli, M. Bertini, A. Del Bimbo, "Bloom Filters and Compact Hash Codes for Efficient and Distributed Image Retrieval". IEEE ISM, 2016 The IEEE International Symposium on Multimedia

## International Journals

### Submitted

1. S. Ercoli, M. Bertini, A. Del Bimbo, "Compact Hash Codes for Efficient Visual Descriptors Retrieval in Large Scale Databases". *IEEE Transactions on Multimedia*

# Bibliography


[1] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[2] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proc. of ACM-SIAM SODA*, 2007.

[3] A. Babenko and V. Lempitsky, "The inverted multi-index," in *Proc. of CVPR*, 2012.

[4] ——, "Efficient indexing of billion-scale datasets of deep descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2055–2063.

[5] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *Proc. of ECCV*, 2014.

[6] S. J. Banerjee, M. Azharuddin, D. Sen, S. Savale, H. Datta, A. K. Dasgupta, and S. Roy, "Using complex networks towards information retrieval and diagnostics in multidimensional imaging," *Scientific reports*, vol. 5, 2015.

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[8] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[9] V. Chandrasekhar, J. Lin, O. Morere, A. Veillard, and H. Goh, "Compact global descriptors for visual search," in *Proc. of DCC*, 2015.

[10] V. Chandrasekhar, M. Makar, G. Takacs, D. Chen, S. S. Tsai, N.-M. Cheung, R. Grzeszczuk, Y. Reznik, and B. Girod, "Survey of SIFT compression schemes," in *Proc. of WMPP*, 2010.

[11] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *Proc. of BMVC*, 2014.

[12] C.-C. Chen and S.-L. Hsieh, "Using binarization and hashing for efficient SIFT matching," *Journal of Visual Communication and Image Representation*, vol. 30, pp. 86–93, 2015.

[13] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11 259–11 273, 2010.

[14] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 3642–3649.

[15] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1237.

[16] D. Cremers, M. Rousson, and R. Deriche, "A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape," *International journal of computer vision*, vol. 72, no. 2, pp. 195–215, 2007.

[17] O. Danielsson, "Category-sensitive hashing and Bloom filter based descriptors for online keypoint recognition," in *Proc. of SCIA*, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-19665-7_27

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. of SoCG*, 2004.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. of CVPR*, 2009.

[20] T.-T. Do, A.-Z. Doan, and N.-M. Cheung, "Discrete hashing with deep neural network," *arXiv preprint arXiv:1508.07148*, 2015.

[21] S. Du, W. Zhang, S. Chen, and Y. Wen, "Learning flexible binary code for linear projection based hashing with random forest," in *Proc. of ICPR*, 2014.

[22] J. P. Eakins and M. E. Graham, "Content based image retrieval: A report to the jisc technology applications programme," 1999.

[23] S. Ercoli, M. Bertini, and A. Del Bimbo, "Compact hash codes and data structures for efficient mobile visual search," in *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on.* IEEE, 2015, pp. 1–6.

[24] A. X. Falcão, P. A. Miranda, and A. Rocha, "A linear-time approach for image segmentation using graph-cut measures," in *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, 2006, pp. 138–149.

[25] D. Feng, W.-C. Siu, and H. J. Zhang, *Multimedia information retrieval and management: Technological fundamentals and applications*. Springer Science & Business Media, 2013.

[26] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[27] L. Gao, J. Song, F. Zou, D. Zhang, and J. Shao, "Scalable multimedia retrieval by deep learning hashing with relative similarity learning," in *Proc. of ACM MM*, 2015. [Online]. Available: http://doi.acm.org/10.1145/2733373.2806360

[28] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proc. of CVPR*, 2013.

[29] ——, "Optimized product quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 744–755, 2014.

[30] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. of VLDB*, 1999.

[31] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, no. 6, 1999, pp. 518–529.

[32] Y. Gong, S. Kumar, H. Rowley, and S. Lazebnik, "Learning binary codes for high-dimensional data using bilinear projections," in *Proc. of CVPR*, 2013.

[33] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[34] D. Guo, C. Li, and L. Wu, "Parametric and nonparametric residual vector quantization optimizations for {ANN} search," *Neurocomputing*, 2016.

[35] J. Guo and J. Li, "CNN based hashing for image retrieval," *arXiv preprint arXiv:1509.01354*, 2015.

[36] A. Guttman, *R-trees: a dynamic index structure for spatial searching*. ACM, 1984, vol. 14, no. 2.

[37] A. Halawani, A. Teynor, L. Setia, G. Brunner, and H. Burkhardt, "Fundamentals and applications of image retrieval: An overview." *Datenbank-Spektrum*, vol. 18, no. 14-23, p. 6, 2006.

[38] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proc. of CVPR*, 2013.

[39] J. P. Heo, Y. Lee, J. He, S. F. Chang, and S. E. Yoon, "Spherical hashing: Binary code embedding with hyperspheres," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 11, pp. 2304–2316, 2015.

[40] S.-i. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm," *IEEE transactions on Neural Networks*, vol. 3, no. 5, pp. 801–806, 1992.

[41] A. Hurwitz, "Ueber die erzeugung der invarianten durch integration," *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, vol. 1897, pp. 71–2, 1897.

[42] K. Inoue and K. Kise, "Compressed representation of feature vectors using a Bloomier filter and its application to specific object recognition," in *Proc. of ICCV Workshops*, 2009.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[44] M. Jain, H. Jégou, and P. Gros, "Asymmetric Hamming embedding: Taking the best of our bits for large scale image search," in *Proc. of ACM MM*, 2011.

[45] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid, "Aggregating local descriptors into compact codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1704–1716, 2012.

[46] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Proc. of ECCV*, 2008.

[47] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[48] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," in *Proc. of ICASSP*, 2011.

[49] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. of ACM MM*, 2014.

[50] Z. Jin, C. Li, Y. Lin, and D. Cai, "Density sensitive hashing," *IEEE Transactions on Cybernetics*, vol. 44, no. 8, pp. 1362–1371, 2014.

[51] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. of CVPR*, 2014.

[52] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar, "Efficient near-duplicate detection and sub-image retrieval," in *ACM Multimedia*, vol. 4, no. 1. Citeseer, 2004, p. 5.

[53] D. Keysers, T. Deselaers, C. Gollan, and H. Ney, "Deformation models for image recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 8, pp. 1422–1435, 2007.

[54] A. Kirsch and M. Mitzenmacher, "Distance-sensitive Bloom filters," in *Proc. of ALENEX*, 2006.

[55] ——, "Less hashing, same performance: Building a better Bloom filter," *Random Structures & Algorithms*, vol. 33, no. 2, pp. 187–218, 2008. [Online]. Available: http://dx.doi.org/10.1002/rsa.v33:2

[56] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Master's thesis, Department of Computer Science, University of Toronto, 2009.

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012.

[58] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Proc. of NIPS*, 2009.

[59] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *2009 IEEE 12th international conference on computer vision*. IEEE, 2009, pp. 2130–2137.

[60] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[61] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998.

[62] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, 2015.

[63] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 2, no. 1, pp. 1–19, 2006.

[64] P. Li, M. Wang, J. Cheng, C. Xu, and H. Lu, "Spectral hashing with semantically consistent graph for image indexing," *IEEE Transactions on Multimedia*, vol. 15, no. 1, pp. 141–152, 2013.

[65] J. Lin, O. Morère, J. Petta, V. Chandrasekhar, and A. Veillard, "Tiny descriptors for image retrieval with unsupervised triplet hashing,"

*arXiv preprint arXiv:1511.03055*, 2015. [Online]. Available: http://arxiv.org/abs/1511.03055

[66] ——, "Tiny descriptors for image retrieval with unsupervised triplet hashing," in *Proc. of DCC*, 2016.

[67] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen, "Deep learning of binary hash codes for fast image retrieval," in *Proc. of CVPR*, 2015.

[68] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[69] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proc. of CVPR*, 2012.

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

[71] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proc. of VLDB*, 2007.

[72] Y. Lv, W. W. Y. Ng, Z. Zeng, D. S. Yeung, and P. P. K. Chan, "Asymmetric cyclical hashing for large scale image retrieval," *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp. 1225–1235, 2015.

[73] B. Matei, Y. Shan, H. S. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert, "Rapid object indexing using locality sensitive hashing and joint 3d-signature space estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1111–1126, 2006.

[74] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[75] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 2161–2168.

[76] M. Norouzi and D. Fleet, "Cartesian k-means," in *Proc. of CVPR*, 2013.

[77] M. Norouzi, A. Punjani, and D. Fleet, "Fast exact search in Hamming space with multi-index hashing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 6, pp. 1107–1119, 2014.

[78] M. Norouzi and D. J. Fleet, "Minimal loss hashing for compact binary codes," in *Proc. of ICML*, 2011.

[79] G. B. Orr and K.-R. Müller, *Neural networks: tricks of the trade*. Springer, 2003.

[80] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

[81] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proc. of CVPR*, 2007.

[82] ——, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *Proc. of CVPR*, 2008.

[83] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *Proc. of NIPS*, 2009.

[84] G. Ren, J. Cai, S. Li, N. Yu, and Q. Tian, "Scalable image search with reliable binary code," in *Proc. of ACM MM*, 2014.

[85] J. T. Robinson, "The kdb-tree: a search structure for large multidimensional dynamic indexes," in *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*. ACM, 1981, pp. 10–18.

[86] Y. Rui, T. S. Huang, and S.-F. Chang, "Image retrieval: Current techniques, promising directions, and open issues," *Journal of visual communication and image representation*, vol. 10, no. 1, pp. 39–62, 1999.

[87] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the Fisher vector: Theory and practice," *International Journal of Computer Vision*, vol. 105, no. 3, pp. 222–245, 2013.

[88] G. Shakhnarovich, P. Indyk, and T. Darrell, *Nearest-neighbor methods in learning and vision: theory and practice*, 2006.

[89] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[90] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[91] ——, "Very deep convolutional networks for large-scale image recognition," in *Proc. of ICLR*, 2015.

[92] J. Sivic and A. Zisserman, "Video Google: a text retrieval approach to object matching in videos," in *Proc. of CVPR*, 2003.

[93] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 12, pp. 1349–1380, 2000.

[94] K. Srijan and C. V. Jawahar, "Towards exhaustive pairwise matching in large image collections," in *Proc. of ECCV*, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33863-2_22

[95] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. of CVPR*, 2015.

[96] ——, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[97] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[98] ——, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[99] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on.* IEEE, 2008, pp. 1–8.

[100] ——, "Small codes and large image databases for recognition," in *Proc. of CVPR*, 2008.

[101] T. Uricchio, M. Bertini, L. Seidenari, and A. Del Bimbo, "Fisher encoded convolutional bag-of-windows for efficient image retrieval and social image tagging," in *Proc. of ICCV Workshops*, 2015.

[102] J. van Gemert, C. Veenman, A. Smeulders, and J.-M. Geusebroek, "Visual word ambiguity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1271–1283, 2010.

[103] D. Wang, P. Cui, M. Ou, and W. Zhu, "Learning compact hash codes for multimodal representations using orthogonal deep structure," *IEEE Transactions on Multimedia*, vol. 17, no. 9, pp. 1404–1416, 2015.

[104] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. of NIPS*, 2009.

[105] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning." in *Proc. of AAAI*, 2014.

[106] J. Yue-Hei Ng, F. Yang, and L. S. Davis, "Exploiting local features from deep networks for image retrieval," in *Proc. of CVPR Workshops*, June 2015.

[107] Z. Zhang, Y. Chen, and V. Saligrama, "Supervised hashing with deep neural networks," *arXiv preprint arXiv:1511.04524*, 2015.

[108] W. Zhou, M. Yang, H. Li, X. Wang, Y. Lin, and Q. Tian, "Towards codebook-free: Scalable cascaded hashing for mobile image search," *IEEE Transactions on Multimedia*, vol. 16, no. 3, pp. 601–611, 2014.

[109] W. Zhou, Y. Lu, H. Li, and Q. Tian, "Scalar quantization for large scale image search," in *Proc. of ACM MM*, 2012.