



Bag of indexes: a multi-index scheme for efficient approximate nearest neighbor search

Federico Magliani¹ · Tomaso Fontanini¹ · Andrea Prati¹

Received: 27 June 2019 / Revised: 2 September 2020 / Accepted: 9 December 2020 /

Published online: 15 January 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC part of Springer Nature 2021

Abstract

During the last years, the problem of Content-Based Image Retrieval (CBIR) was addressed in many different ways, achieving excellent results in small-scale datasets. With growth of the data to evaluate, new issues need to be considered and new techniques are necessary in order to create an efficient yet accurate system. In particular, computational time and memory occupancy need to be kept as low as possible, whilst the retrieval accuracy has to be preserved as much as possible. For this reason, a brute-force approach is no longer feasible, and an Approximate Nearest Neighbor (ANN) search method is preferable. This paper describes the state-of-the-art ANN methods, with a particular focus on indexing systems, and proposes a new ANN technique called Bag of Indexes (BoI). This new technique is compared with the state of the art on several public benchmarks, obtaining 86.09% of accuracy on Holidays+Flickr1M, 99.20% on SIFT1M and 92.4% on GIST1M. Noteworthy, these state-of-the-art accuracy results are obtained by the proposed approach with a very low retrieval time, making it excellent in the trade off between accuracy and efficiency.

Keywords Content-based image retrieval · Approximate nearest neighbor search · Hashing · LSH

1 Introduction

The methods applied for the resolution of the Content-Based Image Retrieval (CBIR) problem on small datasets recently reached excellent results. However, with the growth

✉ Federico Magliani
federico.magliani@studenti.unipr.it

Tomaso Fontanini
tomaso.fontanini@studenti.unipr.it

Andrea Prati
andrea.prati@unipr.it

¹ IMP lab, University of Parma, Parco Area delle Scienze, 181/A, Parma, Italy

of images, the large-scale retrieval task became more interesting. The retrieval on large datasets, composed by almost millions of images, is not yet capable of achieving performances comparable to the small-scale scenarios, because it brings with it several additional challenges. Moreover, it is worth to note that these challenges are not only related to computer vision tasks, but they are typical also, e.g., to the information retrieval field. First, the amount of data representing the descriptors of the images does not fit into the memory, because they are usually high-dimensional features. There are two main solutions: the first one is to reduce the size of the descriptors in order to fit into the memory (e.g., with descriptors of maximum size of 32 bits); an alternative solution is to avoid storing the descriptors into the memory by adopting a similarity metric between images not based on image descriptors distance. Of course, these two strategies can be also applied simultaneously.

A second challenge to take into account is related to the retrieval time which tends to be quite slow if a brute-force approach is employed. In fact, the brute-force strategy linearly increases the time spent on the retrieval with the growth of the number of images in the dataset and becomes quickly inapplicable for the resolution of the Nearest Neighbor (NN) search problem due to the well-known curse of dimensionality problem [1]. For solving this problem, the Approximate Nearest Neighbor (ANN) search can be a good strategy. ANN search consists in a non-exhaustive search, since it finds only the elements that have a high probability to be the nearest neighbor of the query descriptor. More specifically, ANN consists in returning a point that has a distance from the query equals to at most c times the distance from the query to its nearest points, where $c = (1 + \epsilon) > 1$ and ϵ is an infinitesimal value. The trade-off between retrieval performance and query time is at the basis of this task.

Using indexing algorithms is the common approach to efficiently solve ANN problems. The first step is to organize the descriptors of the dataset in an efficient way and then, during the retrieval, to search only the interesting elements. Finally, a re-ranking method based on real distances between descriptors can be implemented to obtain a short list of results. This two-steps approach is intended to improve efficiency, while preserving accuracy. In fact, the reduction of candidate elements through ANN search should retain in the short list all the relevant elements, by applying the brute-force approach on a limited set of elements, for the sake of efficiency. In other words, thanks to this pipeline, the complexity in the search can be reduced up to $O(\log(n))$, considering that the brute-force approach has a complexity equals to $O(n)$, where n is the number of the images to be checked in the dataset.

In this paper we present a novel indexing method called Bag of Indexes (BoI). This unsupervised approach allows to index image descriptors in a fast and representative way and then execute efficiently the query, obtaining excellent retrieval results. In order to demonstrate the goodness and generality of the proposed approach, experiments are conducted on several different descriptors: SIFT [21] and GIST [32] features and locVLAD [24] descriptors. Firstly, the image descriptors are hashed multiple times through LSH [13] projections. The system works also with different hashing functions for the projection phase, but LSH is preferred because it is faster to execute and simpler to implement than other techniques. Secondly, the descriptors are searched during the retrieval phase only in the query bucket and in its neighborhoods, assigning weights to the retrieved descriptors and creating a ranking list. This approach allows our system to reduce the search space by obtaining both a fast and accurate retrieval, supposing that the hash dimension and the number of hash tables are appropriate. Finally, in order to improve the results, only the top elements of the ranking are re-ranked based on the Euclidean distance between the query and the database descriptors. This step exploits the brute-force strategy, but it is executed only for a subset of database

elements, usually not more than the 0.1% of the database descriptors, so the impact on the query time is negligible.

The main contributions of this paper are the followings:

- a very effective Bag of Index structure, similar to Bag of Words, that contains the indexes of the database images;
- a high-quality weighting strategy applied to the BoI structure, that allows to improve the quality of the retrieval phase;
- an efficient search strategy based on multi-probe LSH that allows to select the number of buckets to check every hash table (gap), in order to both reduce the computational time and preserve the retrieval accuracy;
- a complete ablation analysis of the parameters and the strategy adopted in the BoI adaptive multi-probe LSH algorithm (e.g. different weighting strategies);
- good retrieval performance and excellent trade-off between performance and computational time on several public image datasets.

This paper is organized as follows. Section 2 explains the approaches used in the state of the art. Section 3 presented the proposed Bag of Indexes (BoI) approach, which is then evaluated in Section 4 on public benchmarks: Holidays+Flickr1M, SIFT1M, and GIST1M. Finally, concluding remarks are reported.

2 Related works

This section will briefly describe the most relevant previous works on ANN search. The methods are subdivided based on their approach in: scalar quantization, permutation pivots, hashing functions, vector quantization and neural networks.

2.1 Scalar quantization

Zhou et al. [38] proposed to apply a scalar quantization on a SIFT descriptor in order to obtain a descriptive and discriminative bit vector. Moreover, the proposed quantizer is independent from the collection of the images. The threshold used for the quantization is computed through the calculation of the median and the third quartile of the bins of each descriptor. After that, the vector is hashed using 2 bits. Considering in input a SIFT descriptor of 128D, the final input will be 256 bits long. It is worth to note that only the first 32 bits are used for indexing the descriptor in an inverted file and that no codebook is needed to be trained in their quantization scheme. Ren et al. [33] extended the previous work, evaluating the reliability of bits and then checking their quantization errors. The proposed strategy for the unreliable bits is to flip them during the search phase, similar to a query expansion technique.

Zhou et al. [39] proposed to use scalable cascade hashing (SCH) sequentially for calculating the scalar quantization. Also in this case, no codebook is required for the scalar quantization application. The method is applied on vectors using PCA on SIFT descriptors. This process allows to ensure a final high recall. Chen and Hsieh [4] proposed a fast image retrieval scheme, where SIFT descriptors are transformed in binary values, allowing to use hashing for the retrieval phase. The quantization process is executed using a threshold, calculated as the median of all the SIFT descriptors.

2.2 Permutation pivots

Permutation Pivots [7] represents the image descriptors through permutation of a set of randomly selected reference objects.

Firstly, a reduced number of random database descriptors are chosen. Generally speaking, the lower the number of the reference objects, the lower the retrieval time is, but with losses in the performance. Secondly, all the database descriptors are ordered by the distance to the reference objects. Usually Euclidean distance is adopted. Thirdly, at query side, the distances between the query descriptor and the reference objects are computed. In this way only the best reference objects are evaluated because if a query is not close to a reference object, then probably the database descriptors that are close to this reference objects are not close to the query descriptor. The distances between query and database descriptors are computed following the Spearman Footrule Distance [2]. The distance value is the sum of the difference of ranking positions between the query descriptor and the database descriptors on the best, previously selected, reference objects. Finally, re-ranking is applied through the calculation of the original distance between the query descriptor and the retrieved database descriptors.

This method is simple to implement, easy to understand and produces also good results.

2.3 Hashing functions

Locality-Sensitive Hashing (LSH) [13] is probably the most famous hashing technique used for compression and indexing tasks. It projects points that are close to each other into the same bucket with high probability. It is defined as follows [35]: a family of hash functions \mathcal{H} is called (R, cR, P_1, P_2) -sensitive if, for any two items \mathbf{p} and \mathbf{q} , it holds that:

- if $\text{dist}(\mathbf{p}, \mathbf{q}) \leq R$, $\text{Prob}[h(\mathbf{p}) = h(\mathbf{q})] \geq P_1$
- if $\text{dist}(\mathbf{p}, \mathbf{q}) \geq cR$, $\text{Prob}[h(\mathbf{p}) = h(\mathbf{q})] \leq P_2$

with $c > 1$, $P_1 > P_2$, R is a distance threshold, and $h(\cdot)$ is the hash function. In other words, the hash function h must satisfy the property to project “similar” items (with a distance lower than the threshold R) to the same bucket with a probability higher than P_1 , and have a low probability (lower than $P_2 < P_1$) to do the same for “not-similar” items (with a distance higher than $cR > R$).

The hash function used in LSH for Hamming space is a scalar projection:

$$h(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{v}) \quad (1)$$

where \mathbf{x} is the feature vector and \mathbf{v} is a vector with the components randomly selected from a Gaussian distribution $\mathcal{N}(0, 1)$.

The process is repeated L times, with different Gaussian distributions, to increase the probability to satisfy the above constraints. The retrieval phase using LSH has a complexity of $O(\log(N))$, where N is the number of descriptors.

Performing a similarity search query on an LSH index consists of two steps:

1. apply LSH functions to the query image;
2. search in the corresponding bucket possible similar images, ranking the candidate objects according to their distances from the query image.

In order to reduce the number of hash tables, a different LSH implementation called multi-probe LSH [23], that checks also the buckets near the query bucket, can be used. This

approach allows to improve the final performance because usually exploiting the LSH projections similar elements are projected in the same bucket or in near buckets. Sensitive Hashing (DSH) [16] is a variant of LSH, that explores the geometrical structure of the data in order to better project in a more reduced space than the original LSH. Spectral Hashing [36] is a data-dependent hashing method used for the projection of data. It learns the projection matrix by solving a minimization problem. All the data-dependent hashing methods are slower than LSH and therefore they are not suitable for our purposes. Du et al. [5] proposed the use of random forest for calculating an hash signature through the application of linear projections. It is also worth to note that they adopted a different metric from the common Hamming distance, used in the majority of hashing task.

2.4 Vector quantization

Product Quantization (PQ) [15] decomposes the space into a Cartesian product of low dimensional subspaces and quantizes each subspace separately. A vector is represented by a short code composed of its subspace quantization indices.

PQ works as follow:

- the input vector x is split into m distinct sub-vectors u_j , $1 \leq j \leq m$ of dimension $D^* = D/m$, where D is a multiple of m ;
- the sub-vectors are quantized separately using m distinct quantizers;
- a given vector x is therefore mapped as follows:

$$\underbrace{x_1, \dots, x_{D^*}}_{u_1(x)}, \dots, \underbrace{x_{D-D^*+1}, \dots, x_D}_{u_m(x)} \rightarrow q_1(u_1(x)), \dots, q_m(u_m(x)) \quad (2)$$

where q_j is a low-complexity quantizer associated with the j^{th} sub-vector;

- the index set \mathcal{I}_j , the codebook \mathcal{C}_j and the corresponding reproduction values c_j are associated with the subquantizer q_j ;
- the codebook is therefore defined as the Cartesian product:

$$\mathcal{C} = \mathcal{C}_1 \times \dots \times \mathcal{C}_m \quad (3)$$

where a centroid of this set is the concatenation of centroids of the m subquantizer.

Recently, PQ has been improved thanks to the work of [8, 18] that addressed the problem of optimal space decomposition. OPQ [8] minimizes quantization distortion, while LOPQ [18] locally optimizes an individual product quantizer per cell and uses it to encode residuals.

Norouzi and Fleet [30] proposed two variants of k-means (namely, orthogonal k-means and Cartesian k-means), exploiting the principle approach of PQ: the compositionality of the application of several sub-quantization on the input vector. Furthermore, Gao et al. [11] proposed a Residual Vector Quantization (RVQ): the basic idea is to try to restore the quantization errors using multi-stage quantizers, instead of the decomposition used by PQ and its variants.

Ercoli et al. [6] proposed a novel version of k-means vector quantization approach. The main advantage of this technique is the reduced number of clusters needed for the quantization. First, a classical k-means is computed in order to create a dictionary, using a very small number k of centroids for maintaining low the computational cost. The compression is obtained through the association between each centroid and specific bit of the hash code.

The authors proposed different thresholds to be used and the best results are obtained using the arithmetic mean. The proposed approach creates different hash signatures using a much smaller number of centroids than using the k-means baseline algorithm.

2.5 Neural networks

Recently, with the advent of the Convolutional Neural Networks (CNNs) and the deep learning approach, many computer vision tasks are solved using these new architectures and techniques. The CNNs are used in order to create a compact binary code of an image, inserted in the input of a deep neural network [25]. By following the strategy of *end-to-end training*, it is possible to improve the representativeness of the binary codes and, therefore, the retrieval performance increases [9, 27, 28]. Lu et al. [22] proposed a new deep supervised hashing method for efficient image retrieval, designing a novel deep network with a hash layer as the output layer. Zhu et al. [40] proposed the Deep Hashing Network (DHN) architecture for supervised hashing. The architecture is composed by a fully-connected hashing layer, a pairwise cross-entropy loss layer and a pairwise quantization loss. The cross-entropy loss allows to maintain the similarity between the image descriptors, whilst the quantization loss controls the hashing quality. A potential weakness of deep learning and of all the supervised approaches is the need of huge amount of data in order to train the classifier to recognize the different pictures (elements/views) of a class. To solve this problem Cao et al. [3] introduced HashGAN, that creates binary hash codes from real images and different images synthesized by generative models.

Lin et al. [20] proposed a deep learning framework for quickly generating binary hash codes for image retrieval. The CNN is used for learning the image representation and, then, the hash signature is learned using a hidden layer composed by a set of attributes. Xia et al. [37] proposed a supervised hashing method for image retrieval. Starting from the pairwise similarity matrix of the training set, the approach tried to decompose the dot product in order to learn the best transformation matrix. Then, using a set of hash functions, the deep neural network learns the best representation. Wang et al. [34] proposed an hashing pipeline based on multimodal deep learning. The pre-training is executed using a Deep Belief Network and the fine-tuning through an Orthogonal Multimodal AutoEncoder. Lin et al. [19] proposed to use Unsupervised Triplet Hashing, a fully unsupervised method in order to calculate the hash signatures for each input image. Firstly, a Stacked Restricted Boltzmann Machines is applied and, then, triplet networks are used for fine-tuning the previously-learned binary embeddings.

The proposed method belongs to the latter strategy. However, our proposal adopts several different techniques in order to assure that elements similar to the query are projected in nearby buckets. As such, our method can be considered as an hybrid method. In fact, an important difference between our method and those based on scalar quantization is that no codebook (and, therefore, no training) is needed.

In comparison to permutation pivots, our proposal results much faster since it does not require to store a large number of reference objects. Moreover, in case of large and complex datasets, methods based on permutation pivots might struggle to find the right reference objects to compare with.

Vector quantization techniques are pretty fast but with a generally-low accuracy. Finally, compared with neural network-based approaches, ours does not require cumbersome tuning of parameters and the best configuration is easy to find, as will be shown in the ablation analysis in Section 4.3.

3 Bag of indexes

3.1 Overview

Bag of Indexes (BoI) [26]¹ borrows concepts from the well-known Bag of Words (BoW) approach. It is a form of multi-index hashing method [10, 31] for the resolution of ANN search problem.

Figure 1 shows the steps of BoI algorithm. For each input query image a BoI structure is created and stored in memory. Then, the query image is projected with the same hash functions used for the database images. During the retrieval phase, elements similar to the query are searched, by computing the query neighbors as it will be described in Section 3.4. For all the database elements retrieved, the corresponding weights are updated as described in Section 3.3.

In order to reduce the computational time, the gap can be reduced based on the current number of hash tables processed (see Section 3.4). Finally, a re-ranking step is executed in order to increase the retrieval performance.

3.2 BoI LSH

Following the LSH approach, L hash tables composed by 2^δ buckets (that will contain the indexes of the database descriptors) are created. The parameter δ represents the hash dimension in bits. The list of parameters of BoI and chosen values are reported in Table 1 in Section 3.4. Then, the descriptors are projected L times using hashing functions. The L projection values obtained are the buckets of the hash tables that will contain the index of the images. The complete method for the calculation of the projection is reported, for the sake of completeness, in the Appendix A. It is worth to note that this approach can be used in combination with different projection functions, not only hashing and LSH functions. However, in the following explanation and experiments, the LSH approach has been used.

At query time, for each query, a BoI structure is created, that is a vector of n weights (each corresponding to one image of the database) initialized to zero. Every element of the vector will be filled based on the weighting method explained in Section 3.3. By applying this procedure for each of the L hash tables, a coarse-grain evaluation of similarity between the query and the database images can be achieved, by taking into account the frequencies computed for each bucket. This avoids the computation of all pairwise Euclidean distances, employed by a standard brute-force approach. Subsequently, at the end of the retrieval phase, the ε elements of the vector with the highest weights are re-ranked according to their Euclidean distance from the query. The nearest neighbors are then searched only in this short re-ranked list. By computing the Euclidean distances only at the end of the retrieval phase and only on this short list (instead of computing them on each hash table like in standard LSH), the computational time is greatly reduced. When the dataset used is bigger, for obtaining good retrieval performance the BoI structure needs to be filled with a lot of elements. So, in order to reduce the insertions of useless elements, a threshold (α) was set. If the weight of an image, at the end of the process of all the hash tables, is less than the threshold, the index will not be inserted in the BoI structure. Reducing the length of this list allows to improve the speed of the sort function, that depends by the number of the elements contained in the BoI structure since it has a complexity equals to $O(N \log N)$, where

¹The C++ code is available on [GitHub](#)

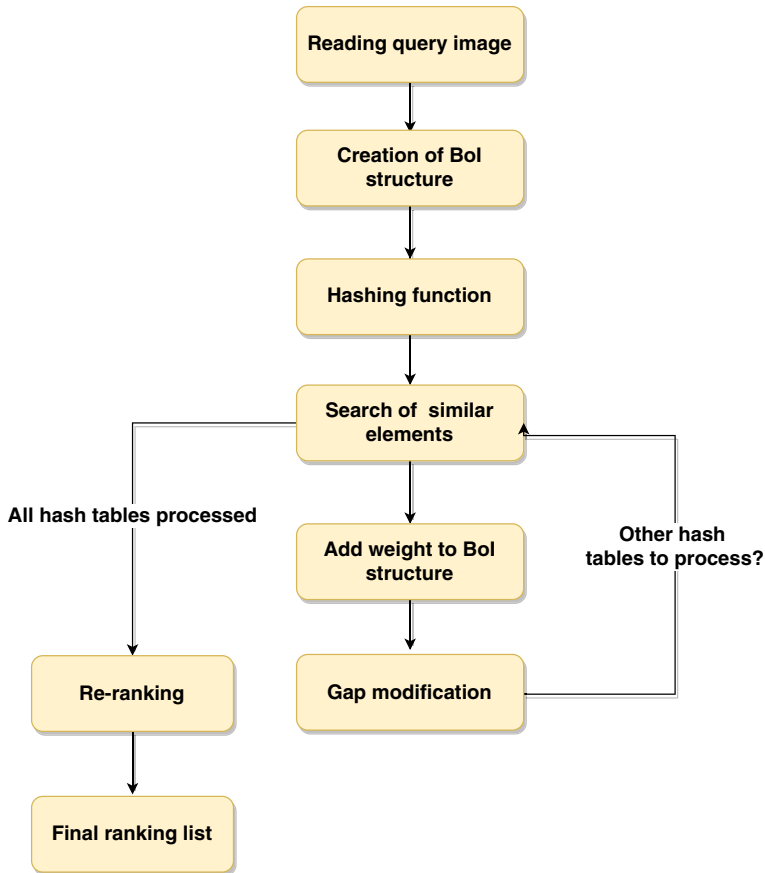


Fig. 1 Workflow figure of BoI approach

N are the elements on which the function is applied. Furthermore, this approach, unlike LSH, does not require to maintain a ranked list without duplicates for all the L hash tables. The detailed analysis of the memory occupation of BoI is reported in Appendix B.

3.3 BoI multi-probe LSH

As previously reported, BoI can be used in combination with different hashing functions. When used with baseline LSH, at query time only the corresponding bucket, called *query bucket* and obtained through the application of LSH projection at the query image, will be checked. In this case, even though it is faster than LSH, the accuracy suffers a significant loss. Conversely, when BoI is combined with multi-probe LSH, also the l -neighboring buckets are considered. Following the idea of multi-probe LSH, several buckets of a hash table are checked considering the Hamming distance between each bucket and the query bucket. More formally, we consider the l -neighbors, that are the buckets that have a Hamming distance less than or equal to l from the query bucket. Different weighting strategy are

Table 1 Summary of BoI algorithm notation

Symbol	Definition	Chosen value
n	number of images	-
δ	hash dimension	$2^{16} = 65536$
L	number of hash tables	50
γ_0	initial gap (at the first hash table processed)	320
l	neighbors bucket	3-neighbors
α	threshold applied before re-ranking	$L * 0.10$
ϵ	elements in the re-ranking list	500
-	reduction	sublinear

implemented. The first one is called *exponential* strategy:

$$w(i, q, l) = \begin{cases} \frac{1}{2^{H(i,q)}} & \text{if } H(i, q) \leq l \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where i is a generic bucket, q is the query bucket and $H(i, q)$ is the Hamming distance between i and q .

Alternatively, two *linear* strategies are implemented, which reduce the weight by a value β at each step (with $\beta = 0.2$ for the so-called *Linear1* strategy, and $\beta = 0.3$ for *Linear2*):

$$w(i, q, l) = \begin{cases} 1 - \beta * H(i, q) & \text{if } H(i, q) \leq l \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The BoI multi-probe LSH approach increases the number of buckets considered during the retrieval and, thus, the probability of retrieving the correct result, by exploiting the main principle of LSH that similar objects should fall in the same bucket or in the ones that are close to it. However, even if we want to account for some uncertainty in the selection of the correct bucket, we also want to weight less as soon as we move farther from the “central” bucket.

Figure 2 shows an exemplar overview of the BoI computation. With $L = 3$ hash tables and 1-neighbours (i.e., $l = 1$), a query can be projected in different buckets. The corresponding weights (in case of exponential strategy, see (4)) are accumulated in the BoI (see the graph on the right of the image). Only the ϵ images with the highest weights are considered for the last step (re-ranking) for improving the recall.

3.4 BoI adaptive multi-probe LSH

The described BoI multi-probe LSH approach has the drawback of increasing the computational time since it also needs to search in neighboring buckets (which are $\sum_{i=0}^l \binom{\delta}{i}$, being δ the hash dimension). To mitigate this drawback, we introduce a further variant, called *BoI adaptive multi-probe LSH*. The main idea of this approach is to iteratively refine the search bucket space, by starting with a large gap γ_0 (e.g., 250) and slowly reduce γ when the number of hash tables increases. The gap γ represents the number of the neighboring buckets to check during the retrieval phase at a certain hash table number. At the beginning $\gamma = \gamma_0$, but during the retrieval process it is modified. At the i -th hash table it is equals to $\gamma = \gamma_i$. This adaptive increase of focus can, on the one hand, reduce the computational time and,

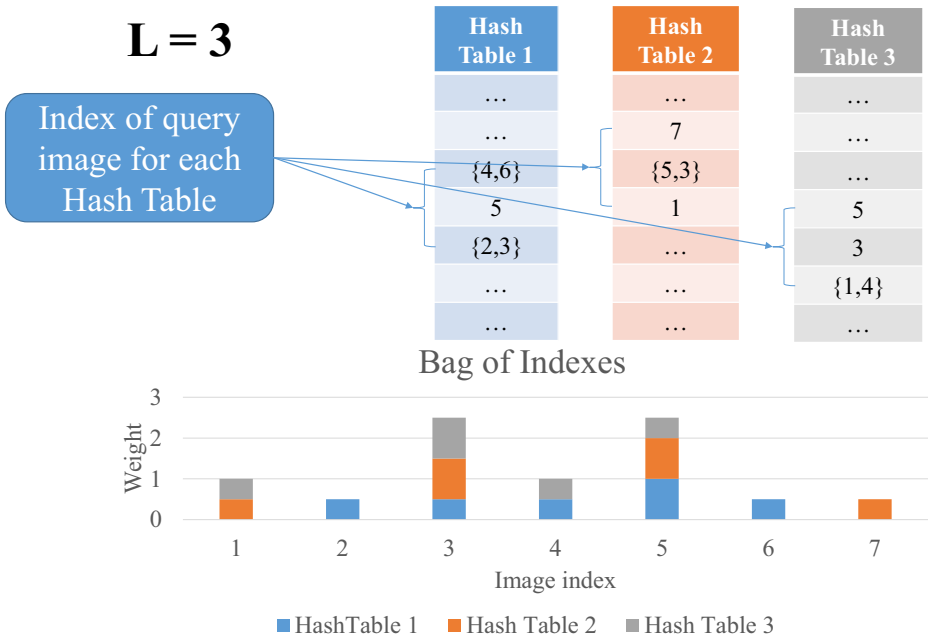


Fig. 2 Overview figure of the retrieval through BoI multi-probe LSH

on the other hand, reduce the noise. In fact, at each iteration, the retrieval results are supposed to be more likely correct and the last iterations are meant to just confirm them, so there is no need to search on a large number of buckets. In order to avoid checking the same neighbors during different experiments, the list of neighbors to check is shuffled randomly at each experiment.

Two different techniques for the reduction of the number of hash tables are evaluated:

- *linear*: the number of neighboring buckets γ is reduced by a quantity (x), that depends on the starting gap (γ_0), every fifth of the L hash tables, i.e.:

$$\gamma_i = \begin{cases} \gamma_{i-1} - x & \text{if } r \bmod 5 = 0 \quad \wedge \quad \gamma_i > 1 \\ \gamma_{i-1} & \text{otherwise} \end{cases} \quad (6)$$

with $r = \{1, \dots, L\}$;

- *sublinear*: the number of neighboring buckets γ is reduced by a quantity (x), that depends on the starting gap (γ_0), each fourth of hash tables, but only after the first half of hash tables, i.e.:

$$\gamma_i = \begin{cases} \gamma_{i-1} & \text{if } r \leq L/2 \quad \wedge \quad \gamma_i > 1 \\ \gamma_{i-1} - x & \text{if } r > L/2 \quad \wedge \quad r \bmod 4 = 0 \quad \wedge \quad \gamma_i > 1 \\ \gamma_{i-1} & \text{otherwise} \end{cases} \quad (7)$$

with $r = \{1, \dots, L\}$

The quantity x should be greater than 0, and proportional to the number of hash tables used. The objective is that, in the last part of the hash table processing, the retrieval is limited to a

small number of hash tables. The condition $\gamma_i > 1$ is set for avoiding a negative gap. Also, in this case, the computation time will be very huge and it is not desirable for our application.

The proposed approach contains several parameters, summarized in Table 1 and some chosen value for the parameters. Their values were chosen after an extensive parameter analysis, reported in Section 4.3 that depend to the used dataset. In general, L , δ and l should be as low as possible since they directly affect the number of buckets \mathcal{N}_q^l to be checked and therefore the computational time at each query q , as follows:

$$\mathcal{N}_q^l = L \sum_{i=0}^l \binom{\gamma_i}{i} = L \sum_{i=0}^l \frac{(\gamma_i)!}{i! (\gamma_i - i)!} \quad (8)$$

where $\gamma_i = \gamma_0 = \delta, \forall i$ for standard BoI multi-probe LSH, whereas, in the case of BoI adaptive multi-probe LSH, γ_i can be computed using the (6) or (7).

4 Experimental results

All the code is implemented in C++ and compiled with pragma omp parallel in order to execute the script on multiple threads. The proposed approach and other state-of-the-art approaches are tested and compared on different public image datasets in order to detect which algorithm reached the best trade-off between retrieval accuracy and average retrieval time.

4.1 Datasets

There are many different image datasets used for Content-Based Image Retrieval (CBIR). The most used datasets are the following:

- **Holidays** [14] is composed of 1491 high-resolution images representing the holidays photos of different locations and objects, subdivided in 500 classes. The database images are 991 and the query images are 500, one for every class;
- **Flickr1M** [12] contains 1 million Flickr images under the Creative Commons license. It is used for large scale evaluation. The images are divided in multiple classes and are not specifically selected for the image retrieval task.

Images taken from Flickr1M are usually employed as distractors to make more difficult the retrieval phase, creating the Holidays+Flickr1M dataset, often used in the literature.

The public benchmarks used for large-scale evaluation are the following:

- **SIFT1M** [15] is composed by 1 million 128D SIFT descriptors. The queries, also represented by 128D SIFT descriptors, are 10k and the nearest vectors for each query are 100;
- **GIST1M** [15] is composed by 1 million 960D GIST descriptors. The queries, also represented by 960D GIST descriptors, are 1000 and the nearest vectors for each query are 100.

The information about the tested datasets are summarized in Table 2.

4.2 Evaluation metrics

To evaluate the accuracy in the retrieval phase, mean Average Precision (mAP) is used on Holidays+Flickr1M. The mAP is the mean of average precision that identifies how many

Table 2 Datasets characteristics

Dataset	Db images	Query images	Metric
Holidays [14]	991	500	mAP
Holidays+Flickr1M	1000991	500	mAP
SIFT1M [15]	1M	10k	recall@R
GIST1M [15]	1M	1k	recall@R

correct elements are found. In the other datasets, in order to fairly compare with the state of the art, Recall@R is used, which is the average rate of queries for which the 1-nearest neighbor is ranked in the top R positions. In order to compare a query image with the database, L_2 distance is employed. It is also reported the average time required for the execution of a single query, which allows to calculate the ratio accuracy/queries per second. The ratio accuracy/queries per second represents a measure of the trade-off between retrieval accuracy and the number of queries that the system is able to perform in a second (e.g., if, on average, a query is retrieved in 1 msec, the query per second will be 1000 queries per second).

4.3 Ablation analysis

This section reports different experiments aiming at selecting the best set of parameters for the proposed approach. Each of the analysed parameters has been selected by extensive experimentation.

4.3.1 Weighting strategy

The different weighting strategies (reported in Section 3.3) have been tested on SIFT1M and GIST1M datasets, in order to find which one results more convenient in terms of accuracy (the efficiency is basically the same). The three strategies (*exponential*, *linear1* and *linear2*) have been compared using slightly different parameters for the two datasets. In the case of SIFT1M dataset, we used the values $L = 50$, $\delta = 8$, and $\varepsilon = 500$, while in the case of GIST1M, chosen values are $L = 100$, $\delta = 8$, and $\varepsilon = 500$. Tables 3 and 4 report the Recall@R, for R=1,10,100.

The results reported in Tables 3 and 4 showed that the *Linear2* strategy produced the best retrieval results in both cases.

4.3.2 Hash dimension and number of hash tables

Two crucial parameters (tightly correlated) are the hash dimension δ and the number of hash tables L . The hash table is composed by some buckets, the total number of buckets is 2^δ . On

Table 3 Results in terms of Recall@R on SIFT1M, changing weighting strategy on BoI multi-probe LSH

Method	R=1	R=10	R=100
Exponential	95.75%	96.33%	96.33%
Linear1	95.02%	95.64%	95.64%
Linear2	96.80%	97.44%	97.44%

Table 4 Results in terms of Recall@R on GIST1M, changing weighting strategy on BoI multi-probe LSH

Method	R=1	R=10	R=100
Exponential	65.50%	66.20%	66.20%
Linear1	59.30%	59.70%	59.70%
Linear2	68.60%	69.50%	69.50%

the one hand, by setting this value high, the probability of having similar objects mapped on the same bucket is reduced (which is not desirable), but, on the other hand, decreasing this value too much will bring, during the retrieval phase, in the creation of a large number of indexes (especially for the large-scale scenario), affecting the efficiency of the retrieval. This is known as the problem of overpopulation and, in order to solve it, the only strategy is to increase the number of buckets of each hash table.

A similar consideration can be made for the number of hash tables L , where increasing L improves the accuracy of the results, but at the cost of computational time.

In order to find the optimal values of L and δ , we run the algorithm on SIFT1M using different values: $L = 25, 50, 75$ and $\delta \in [7, \dots, 17]$. Table 5 shows the obtained Recall@100. As foreseeable, increasing the number of hash tables results in a higher accuracy. This is also the case when increasing the hash dimension, but with $\delta = 17$ the performance is slightly worse, while the best accuracy is obtained with $\delta = 16$. This is probably due to the fact that increasing too much the hash dimension brings to project similar elements in different buckets. The combination with a too high value for hash dimension and a reduced number of hash tables do not allow to achieve the best retrieval results, but only to reduce the retrieval time.

Despite this reasonable and expected result, increasing the number of hash tables and the hash dimension has a cost in terms of computational time. Therefore, we also compute the indexing time (on the database images - Fig. 3) and the retrieval time (on the query image - Fig. 4). As said above, there is an overall tendency of higher computational time (both for indexing and retrieval) when the the number of hash tables and the number of buckets increase.

Table 5 Accuracy on SIFT1M changing L and hash dimension

δ	L = 75	L = 50	L = 25
7	98.81%	96.33%	76.22%
8	99.20%	96.70%	81.24%
9	99.42%	97.65%	84.44%
10	99.58%	98.18%	86.36%
11	99.83%	98.58%	87.27%
12	99.76%	98.73%	88.88%
13	99.79%	99.12%	90.31%
14	99.82%	99.21%	90.48%
15	99.87%	99.17%	91.17%
16	99.85%	99.15%	91.57%
17	99.82%	99.08%	91.27%

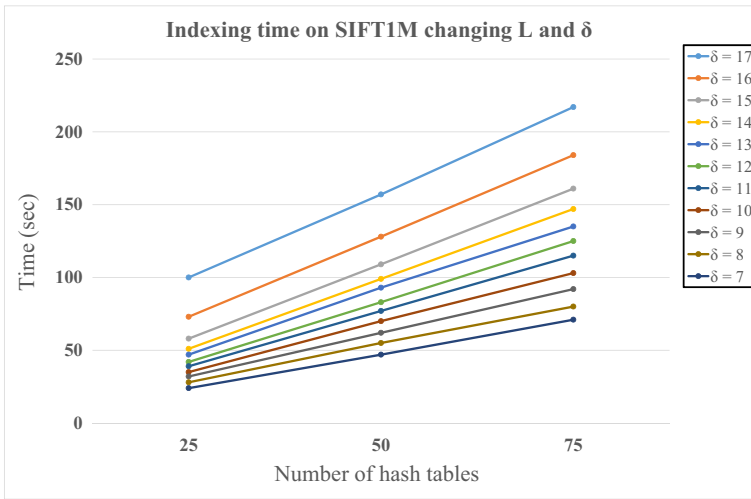


Fig. 3 Indexing time on SIFT1M changing L and hash dimension

As a conclusion, the considered values affect accuracy and efficiency in a conflicting manner. Therefore, we propose a possible score that accounts for both these needs to compute an overall trade-off value TO :

$$TO = \frac{Recall@100 + \frac{\min(I_{time})}{I_{time}} + \frac{\min(R_{time})}{R_{time}}}{3} \tag{9}$$

where $Recall@100$ stands for the retrieval accuracy (which can be also computed with other metrics, such as mAP), I_{time} is the time (in msec) needed for indexing all the database items, and R_{time} is the average retrieval time (also in msec). In order to best evaluate the

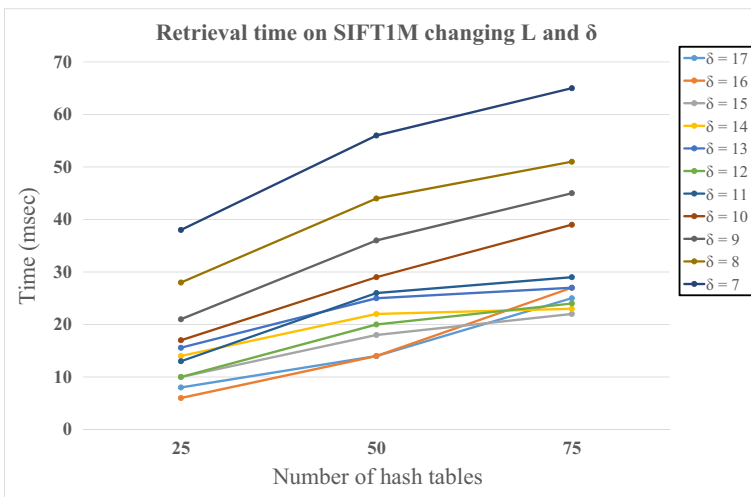


Fig. 4 Retrieval time on SIFT1M changing L and hash dimension

Table 6 Calculation of Trade-Off value for the SIFT1M dataset

δ	L	TO
16	25	74.81%
16	50	78.62%
16	75	73.30%
17	25	63.42%
17	50	76.33%
17	75	73.51%

time (both indexing and retrieval), it is needed that the lower the time is, the higher the trade-off should be. By using the ratio between the minimum time and the computed time we can weigh more very low time values. Finally, the three values are equally weighted through the application of an arithmetic mean. For obtaining a percentage as a final result, the mean is multiplied by 100. By using this trade-off value, we can select, among the most reasonable values coming from the above analysis, the optimal set of value. From Table 6, it is evident that the best trade-off for the SIFT1M dataset is obtained by the $\delta = 16$ and $L = 50$.

4.3.3 Bucket reduction strategy

The initial gap (γ_0) is directly connected to the number of neighbors bucket (L), since with the increase of neighbors buckets to be checked, also the number of neighbors increases as a consequence. Therefore, the proposed BoI adaptive multi-probe LSH allows to reduce the query time by reducing the buckets to check (and, consequently, the corresponding indexes). In order to evaluate which of the two proposed strategies (linear and sublinear) performs better, we tested them on SIFT1M. The results, reported in Table 7, show that the sublinear strategy obtains better results than the linear one, thanks to its ability to keep a larger focus than the linear one, therefore collecting more indexes and increasing the likelihood to find the correct nearest neighbors. It is also worth to note that the linear strategy is slightly faster in calculating the nearest neighbors of a query.

4.3.4 Number of top elements ϵ

Once the candidate elements are found through indexing and hashing, the ϵ top elements are re-ranked by computing (only on them) the Euclidean distance. Therefore, this value is crucial to improve the final retrieval accuracy. However, selecting a high ϵ value results in having the Euclidean distance computed on a larger number of elements, which can affect the retrieval time. In the case of large datasets, the original descriptors can be pre-loaded in memory to improve the time needed for the re-ranking.

Table 8 reports the retrieval results by changing the number of the re-ranking elements. Increasing the value of ϵ allows to obtain better results, but with an extra cost in the retrieval

Table 7 Comparison between linear and sublinear strategy on SIFT1M dataset

Strategy	Recall@100	Avg retrieval time	Ratio
Linear	87.36%	4.5 msec	222.22
Sublinear	91.57%	6 msec	166.66

Table 8 Comparison between different values for the parameter ε on SIFT1M dataset

ε	Recall@100	Avg re-ranking time
500	91.57%	0.2 msec
2500	97.60%	1 msec
5000	98.68%	2 msec
10000	99.39%	4 msec

time. For the sake of completeness, the table shows only the average re-ranking time and not the average retrieval time.

Otherwise, in the case of SIFT1M, re-ranking the top 500 elements directly from the disk needs approximately 4 msec more than loading them from memory, while for the top 10k elements the difference raises to 40 msec.

4.3.5 Cut-off parameter α

The cut-off parameter (α) allows to reduce the number of elements in BoI structure in order to optimize the execution and to reduce the time required for the sorting phase, before the re-ranking step. The cut-off step is executed in an unsupervised manner and only the indexes of the BoI that did not exceed the threshold are removed. This value is usually set to the 10% of the number L of hash tables. It is advisable to keep this value static and not to compute it dynamically because there is no exact knowledge about the distribution of the weights.

Table 9 reports the retrieval results by changing the values for the α parameter. As foreseeable, increasing the value of this parameter, the average retrieval time decreases, but the accuracy slightly diminishes.

4.4 Results on Holidays+Flickr1M dataset

Table 10 shows the current state-of-the-art methods on Holidays+Flickr1M with reference to the retrieval accuracy and the average retrieval time.

LSH and Multi-probe LSH obtained the best results, but requiring an huge average query time (3103 msec for LSH and 16706 msec for Multi-probe LSH). Instead, PP-index [7] reduced the retrieval time, but with a loss in terms of accuracy.

FLANN [29] is an open source library for ANN and one of the most popular for approximate nearest neighbor matching. It includes different algorithms and has an automated configuration procedure for finding the best algorithm to search in a particular data set. It represents the state of the art in indexing techniques, and the achieved accuracy is rather good (83.97%). On the contrary, the accuracy achieved by LOPQ [17] is only 67.22%. Moreover, LOPQ and FLANN are faster than the others method: the first one employed only 4 msec and the second one 995 msec.

Table 9 Comparison between different values for the parameter α on SIFT1M dataset

α	Recall@100	Avg re-ranking time
0.01 * L	91.38%	17 msec
0.02 * L	91.26%	15 msec
0.1 * L	91.15%	8 msec

Table 10 Results in terms of mAP, average retrieval time and ratio on Holidays+Flickr1M. * indicates our re-implementation

Method	ϵ	mAP	Avg retrieval time	Ratio
LSH*	250	86.03%	3103 msec	0.32
Multi-probe LSH* ($L = 50$)	250	86.10%	16706 msec	0.05
PP-index* [7]	250	82.70%	2844 msec	0.35
LOPQ [17]	250	36.37%	4 msec	250
FLANN [29]	250	83.97%	995 msec	1.00
BoI LSH	250	78.10%	5 msec	200
BoI multi-probe LSH	250	85.16%	12 msec	83.33
BoI adaptive multi-probe LSH	250	85.35%	8 msec	125
PP-index* [7]	10k	85.51%	15640 msec	0.06
LOPQ [17]	10k	67.22%	72 msec	13.88
FLANN [29]	10k	85.66%	1004 msec	0.99
BoI adaptive multi-probe LSH	10k	86.09%	16 msec	62.50

The proposed BoI adaptive multi-probe LSH achieved an excellent trade-off between accuracy (85.35%) and retrieval time (8 msec for a query), using $l = 3$ and $L = 100$.

The bottom part of Table 10 reports a comparison with some of the state-of-the-art methods by increasing the number ϵ of re-ranked images from 250 to 10k. This new configuration allowed to obtain better accuracy than the previous one for all the tested methods. However, while other methods (e.g., PP-index) also suffer for a more-the-linear increase in the retrieval time, our BoI adaptive multi-probe LSH reached the best result (86.09%) in 16 msec for query. The ratio values, for $\epsilon = 250$, confirm that the proposed approach is faster than the other ones, except LOPQ, that obtains an horrible retrieval performance. When $\epsilon = 10k$, our method is faster than the others.

4.5 Results on SIFT1M and GIST1M datasets

As mentioned in Section 4.2, in the case of SIFT1M and GIST1M datasets, the standard evaluation metrics is the Recall@R. Table 11 reports the comparison with other methods for SIFT1M at different values of R: 1, 10 and 100.

Table 11 Results in terms of Recall@R, average retrieval time and ratio on SIFT1M. * indicates our re-implementation

Method	ϵ	R=1	R=10	R=100	Avg retrieval time	Ratio
PP-index* [7]	500	94.32%	94.98%	94.98%	16999 msec	0.05
LOPQ [17]	500	19.93%	44.80%	52.92%	3 msec	333.33
FLANN [29]	500	54.47%	54.83%	54.83%	16 msec	62.50
BoI adaptive multi-probe LSH	500	98.36%	99.19%	99.19%	18 msec	55.55
LOPQ [17]	10k	36.34%	80.11%	96.18%	104 msec	9.61
FLANN [29]	10k	95.06%	95.86%	95.86%	31 msec	32.20
BoI adaptive multi-probe LSH	10k	99.20%	100.00%	100.00%	25 msec	40

Table 12 Results in terms of Recall@R, average retrieval time and ratio on GIST1M. * indicates our re-implementation

Method	ϵ	R=1	R=10	R=100	Avg retrieval time	Ratio
PP-index* [7]	500	54.80%	55.30%	55.30%	17909 msec	0.05
LOPQ [17]	500	7.00%	18.50%	27.80%	17 msec	58.82
FLANN [29]	500	28.30%	28.60%	28.60%	1262 msec	0.79
BoI adaptive multi-probe LSH	500	79.80%	80.80%	80.80%	60 msec	16.66
LOPQ [17]	10k	9.60%	28.50%	58.50%	360 msec	2.77
FLANN [29]	10k	75.90%	76.50%	76.50%	1352 msec	0.73
BoI adaptive multi-probe LSH	10k	92.40%	93.40%	93.40%	108 msec	9.25

PP-index [7] achieved a recall equals to 94.32% with R = 1 and almost 95% with R=10 and R=100, but at the cost of a high retrieval time (about 17 sec for a single query). LOPQ confirmed to be very fast (only 3 msec for a query), but the the retrieval accuracy is very poor (recall of 19.93% when R = 1). FLANN obtained better results than LOPQ, but always a way lower than PP-index. Finally, the proposed BoI adaptive multi-probe LSH reached a recall of 98.36% at R = 1 in only 18 msec. This excellent trade-off between accuracy and efficiency is even better when ϵ is set to 10k. In particular, the proposed solution achieved a recall equal to 99.20% on R = 1, in only 25 msec.

Results are even better when applied to GIST1M dataset (Table 12). In this case, the number of hash tables L is set to 100 and the number of buckets for hash table δ is set to 16. Notably, the proposed method achieved, for both the values of ϵ , higher accuracy and lower retrieval time then the competing methods. In this case, moreover, the differences with other methods are even more large.

5 Conclusions

This paper presented a new multi-index scheme applied to the resolution of the large-scale Content-Based Image Retrieval problem using Approximate Nearest Neighbor (ANN) techniques. The proposed *BoI adaptive multi-probe LSH* follows a Bag-of-Words paradigm to build an indexed hash-based representation of database image descriptors which can be used for efficiently find the nearest neighbors of a query image. Experiments have been conducted on several public datasets, in some cases with a large number of samples. The proposed solution resulted to be very promising in terms of trade off between accuracy and retrieval time. It is also worth mentioning that the proposed approach is based on LSH as multi-probe hashing function, but can be also combined with other hashing/projection functions, making the approach extendible to other fields, such as information retrieval.

A future line of research (partially in place) is to extend our analysis to datasets with billions of data, with the aim of maintaining low the retrieval time and the memory occupancy, without affecting the accuracy. Another possible extension of this work is related to the large-scale retrieval for mobile devices, where additional considerations must be made regarding computational issues, as well as memory occupation.

Acknowledgements This research benefits from the HPC (High Performance Computing) facility of the University of Parma, Italy.

This work is partially funded by Regione Emilia Romagna under the “Piano triennale alte competenze per la ricerca, il trasferimento tecnologico e l’imprenditorialità”.

Appendix A - LSH projection algorithm

As mentioned in Section 2.3, the hash function used for projecting the vectors is the following:

$$h(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{v}) \quad (10)$$

where \mathbf{x} represents the input feature vector and \mathbf{v} represents the projecting vector. Before starting the calculation, it is important to generate the vector \mathbf{v} for the projection. The values of this vector are sampled from a Gaussian distribution $\mathcal{N}(0, 1)$. The dimensions of this list of vectors depend from the hash dimension (δ), the number of hash tables (L) and the dimension of the input vector. After that, it is possible to calculate the correct bucket using LSH projections. Assuming to have $\delta = 8$ and $L = 10$, 80 LSH projections will be generated for each image descriptor, but only 10 buckets for each image are obtained for projecting the descriptors (one for each hash table). This is due to the fact that, for each hash table, hash dimension (δ) projections are calculated, so in the end the vectors will be projected in only L buckets. Applying more projections for each hash table (thus, increasing the hash dimension δ) allows to improve the robustness of LSH approach and reduce the possibility of projecting different elements in close buckets because increasing the number of bits used (δ) increases the number of possible buckets for the final projection. To summarize, once we have the projection vectors, the dot product between the input vector and the projection vector is computed. If it is greater than zero, the bucket value (*index*) is increased by a power of two.

B - Memory requirements

The memory requirements of the ANN algorithm depend from the considered number of images. For example, if 1M images are represented by 1M descriptors of 128D (float = 4 bytes) are employed, the brute-force approach requires 0.5Gb (1M x 128 x 4). For the same task, LSH needs only 100 Mb, because it needs to store 1M indexes for each of the $L = 100$ hash tables, since each index is stored using a single byte (8 bit). In addition, the proposed BoI only requires additional 4 Mb to store 1M weights, allowing to scale better the proposed approach on larger datasets than the brute-force approach.

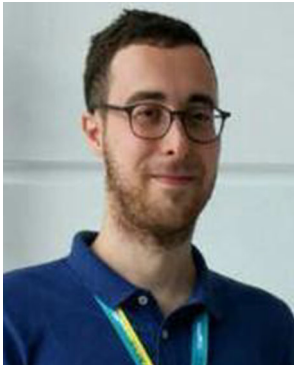
References

1. Böhm C., Berchtold S, Keim DA (2001) Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)* 33(3):322–373
2. Brandenburg FJ, Gleißner A, Hofmeier A (2013) The nearest neighbor spearman footrule distance for bucket, interval, and partial orders. *J Combinatorial Optim* 26(2):310–332
3. Cao Y, Liu B, Long M, Wang J, Kliss M (2018) Hashgan: Deep learning to hash with pair conditional wasserstein GAN. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp 1287–1296

4. Chen CC, Hsieh SL (2015) Using binarization and hashing for efficient SIFT matching. *J Vis Commun Image Represent* 30:86–93
5. Du S, Zhang W, Chen S, Wen Y (2014) Learning flexible binary code for linear projection based hashing with random forest. In: *Proceedings of the 22nd international conference on pattern recognition*. IEEE, pp 2685–2690
6. Ercoli S, Bertini M, Del Bimbo A (2017) Compact hash codes for efficient visual descriptors retrieval in large scale databases. *IEEE Transactions on Multimedia* 19(11):2521–2532
7. Esuli A (2012) Use of permutation prefixes for efficient and scalable approximate similarity search. *Information Processing & Management* 48(5):889–902
8. Ge T, He K, Ke Q, Sun J (2014) Optimized product quantization. *IEEE Trans Pattern Anal Mach Intell* 36(4):744–755
9. Gordo A, Almazán J, Revaud J, Larlus D (2016) Deep image retrieval: Learning global representations for image search. In: *European conference on computer vision*. Springer, pp 241–257
10. Greene D, Parnas M, Yao F (1994) Multi-index hashing for information retrieval. In: *Proceedings of the 35th Annual symposium on foundations of computer science*. IEEE, pp 722–731
11. Guo D, Li C, Wu L (2016) Parametric and nonparametric residual vector quantization optimizations for ANN search. *Neurocomputing* 217:92–102
12. Huiskes MJ, Lew MS (2008) The MIR flickr retrieval evaluation. In: *Proceedings of the 1st ACM international conference on multimedia information retrieval*. ACM, pp 39–43
13. Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: *Proceedings of the 30th annual ACM symposium on theory of computing*. ACM, pp 604–613
14. Jégou H, Douze M, Schmid C (2008) Hamming embedding and weak geometric consistency for large scale image search. In: *European conference on computer vision*. Springer, pp 304–317
15. Jegou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search. *IEEE Trans Pattern Anal Mach Intell* 33(1):117–128
16. Jin Z, Li C, Lin Y, Cai D (2014) Density sensitive hashing. *IEEE Trans Cybern* 44(8):1362–1371
17. Kalantidis Y, Avrithis Y (2014) Locally optimized product quantization for approximate nearest neighbor search. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp 2321–2328
18. Kalantidis Y, Mellina C, Osindero S (2016) Cross-dimensional weighting for aggregated deep convolutional features. In: *European conference on computer vision*. Springer, pp 685–701
19. Lin J, Morere O, Petta J, Chandrasekhar V, Veillard A (2016) Tiny descriptors for image retrieval with unsupervised triplet hashing. In: *Data Compression Conference (DCC)*. IEEE, pp 397–406
20. Lin K, Yang HF, Hsiao JH, Chen CS (2015) Deep learning of binary hash codes for fast image retrieval. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp 27–35
21. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. *Int J Comput Vis* 60(2):91–110
22. Lu X, Song L, Xie R, Yang X, Zhang W (2017) Deep hash learning for efficient image retrieval. In: *IEEE international conference on multimedia & expo workshops*. IEEE, pp 579–584
23. Lv Q, Josephson W, Wang Z, Charikar M, Li K (2007) Multi-probe LSH: efficient indexing for high-dimensional similarity search. In: *Proceedings of the 33rd international conference on very large data bases*. VLDB Endowment, pp 950–961
24. Magliani F, Bidgoli NM, Prati A (2017) A location-aware embedding technique for accurate landmark recognition. In: *Proceedings of the 11th international conference on distributed smart cameras*. ACM, pp 9–14
25. Magliani F, Fontanini T, Prati A (2018) A dense-depth representation for VLAD descriptors in content-based image retrieval. In: *International symposium on visual computing*. Springer, pp 662–671
26. Magliani F, Fontanini T, Prati A (2018) Efficient nearest neighbors search for large-scale landmark recognition. *Proceedings of the 13th international symposium on visual computing* 11241:541–551
27. Magliani F, Prati A (2018) An accurate retrieval through R-MAC+ descriptors for landmark recognition. In: *Proceedings of the 12th international conference on distributed smart cameras*. ACM, p 6
28. Mohedano E, McGuinness K, O'Connor NE, Salvador A, Marques F, Giro-i Nieto X (2016) Bags of local convolutional features for scalable instance search. In: *Proceedings of the international conference on multimedia retrieval*. ACM, pp 327–331
29. Muja M, Lowe DG (2014) Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans Pattern Anal Mach Intell* 36(11):2227–2240

30. Norouzi M, Fleet DJ (2013) Cartesian k-means. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 3017–3024
31. Norouzi M, Punjani A, Fleet DJ (2012) Fast search in hamming space with multi-index hashing. In: IEEE conference on computer vision and pattern recognition. IEEE, pp 3108–3115
32. Oliva A, Torralba A (2001) Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int J Comput Vis* 42(3):145–175
33. Ren G, Cai J, Li S, Yu N, Tian Q (2014) Salable image search with reliable binary code. In: Proceedings of the 22nd ACM international conference on multimedia. ACM, pp 769–772
34. Wang D, Cui P, Ou M, Zhu W (2015) Learning compact hash codes for multimodal representations using orthogonal deep structure. *IEEE Transactions on Multimedia* 17(9):1404–1416
35. Wang J, Shen HT, Song J, Ji J (2014) Hashing for similarity search: A survey. arXiv:1408.2927
36. Weiss Y, Torralba A, Fergus R (2009) Spectral hashing. In: Advances in neural information processing systems, pp 1753–1760
37. Xia R, Pan Y, Lai H, Liu C, Yan S (2014) Supervised hashing for image retrieval via image representation learning. In: AAAI, vol 1, p 2
38. Zhou W, Lu Y, Li H, Tian Q (2012) Scalar quantization for large scale image search. In: Proceedings of the 20th ACM international conference on multimedia. ACM, pp 169–178
39. Zhou W, Yang M, Li H, Wang X, Lin Y, Tian Q et al (2014) Towards codebook-free: Scalable cascaded hashing for mobile image search. *IEEE Transactions of Multimedia* 16(3):601–611
40. Zhu H, Long M, Wang J, Cao Y (2016) Deep hashing network for efficient similarity retrieval. In: AAAI, pp 2415–2421

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Federico Magliani graduated in Computer Engineering at the University of Parma in 2016. Where he is following a PhD course and he is working on Content-Based Image Retrieval, focusing on large-scale retrieval. He worked with Bridgestone on a project called “Detection of cracks in the tires under effort”. He also visited for 5 months the Insight Center for Data Analytics, a research lab of DCU (Dublin City University) under the supervision of the Assistant Professor Kevin McGuinness. He worked as a reviewer for the scientific community (WACV, BMVC).



Tomaso Fontanini graduated in Computer Engineering at the University of Parma in 2017. He then worked on a project in collaboration with Adidas with the objective of extraction and classification of clothes and views of clothes from Adobe Illustrator files. Since November 2017 he is a PhD student funded by Adidas in machine learning and computer vision at University of Parma working on image generation and colorization of clothes with neural network models like Generative Adversarial Networks.



Andrea Prati graduated in Computer Engineering at the University of Modena and Reggio Emilia in 1998. He got his PhD in Information Engineering in 2002 from the same University. After some post-doc positions at University of Modena and Reggio Emilia, he was appointed as Assistant Professor at the Faculty of Engineering of Reggio Emilia (University of Modena and Reggio Emilia) from 2005 to 2011, and then as Associate Professor at the Department of Design and Planning in Complex Environments of the University IUAV of Venice, Italy. In December 2015 he moved to the Department of Engineering and Architecture of the University of Parma and was promoted to Full Professor in February 2019. Author of 9 book chapters, 39 papers in international referred journals (including 11 papers published in highly-ranked journals) and more than 100 papers in proceedings of international conferences and workshops. Andrea Prati is Senior Member of IEEE, Fellow of IAPR (“For contributions to low- and high-level algorithms for video surveillance”), and member of GIRPR.