Published in Image Processing On Line on 2018–06–03.
Submitted on 2017–01–06, accepted on 2017–12–21.
ISSN 2105–1232 © 2018 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2018.199

Efficient Large-scale Image Search with a Vocabulary Tree

Esteban Uriza[1], Francisco Gómez-Fernández[1], Martín Rais[2]

[1] DC, FCEN, UBA, Argentina (euriza@dc.uba.ar, fgomez@dc.uba.ar)
[2] CMLA, ENS Cachan, France (mrais@cmla.ens-cachan.fr)

Abstract

The task of searching and recognizing objects in images has become an important research topic in the area of image processing and computer vision. Looking for similar images in large datasets given an input query and responding as fast as possible is a very challenging task. In this work the Bag of Features approach is studied, and an implementation of the visual vocabulary tree method from Nistér and Stewénius is presented. Images are described using local invariant descriptor techniques and then indexed in a database using an inverted index for further queries. The descriptors are quantized according to a visual vocabulary, creating sparse vectors, which allows to compute very efficiently, for each query, a ranking of similarity for indexed images. The performance of the method is analyzed varying different factors, such as the parameters for the vocabulary tree construction, different techniques of local descriptors extraction and dimensionality reduction with PCA. It can be observed that the retrieval performance increases with a richer vocabulary and decays very slowly as the size of the dataset grows.

Keywords: image processing; scalable recognition; bag of features; vocabulary tree

Source Code

A software written in C++ is available at the IPOL web page of this article[1], which is the code used in the online demo. Program usage and compilation instructions are also provided with the code. For reporting bugs, issues and questions related to the source code, please refer to: https://github.com/fragofer/voctree

1 Introduction

In this work the problem of searching for similar images in a large-scale set of images is addressed. Similar images are defined as images of the same object (or scene) viewed under different imaging conditions: variations in rotations and scale changes, different illumination conditions, partial occlusions, noise, etc. A large-scale object recognition system, takes an image query as input, performs a search process over a large-scale image dataset (possibly containing more than one million images), and retrieves a ranked list of similar images in the dataset as output.

[1]https://doi.org/10.5201/ipol.2018.199

Since the size of the dataset is large, it is not possible to perform a linear scan, taking image by image and computing its similarity against the query at run-time. Thus, this kind of systems make use of preprocessed data in a training stage. Query response time is constrained and should be in the order of the state-of-the-art text document retrieval web engines (milliseconds). It is necessary to find efficient data structures to index precomputed retrieval data, that are used during the search. Indexing structures must fit entirely in RAM, accessing to hard disk drive will be too slow to meet this requirements. These kinds of restrictions make the problem even more challenging.

A local descriptor is an image pattern centered at a pixel with characteristic properties. This pattern is usually associated with a change of an image property or several properties simultaneously, such as intensity, color, or texture. Most distinctive descriptors on common images are located in special positions, such as mountain peaks, building corners, or patches of snow. These special positions are called keypoints. Given two or more images of the same object, a good keypoint detection algorithm should find a high percentage of the same keypoints in all the given images despite different imaging conditions. Typically, corners, edges or blobs in images are used as good keypoints.

Once keypoints are detected, it is possible to apply a descriptor extraction algorithm which gives as a result a set of descriptors per image. A descriptor is often expressed as a D-dimensional vector that describes the appearance of the patches of pixels surrounding the keypoint locations. Examples of popular keypoint detector/descriptor extraction algorithms are SIFT [15], SURF [3], ORB [21], KAZE [1], AKAZE [2], among others. Many local methods, specify both a keypoint detection and descriptor extraction, such as SIFT and SURF, although there are other methods that only focus on keypoint detection (HARRIS [8]), or in a description method (BRIEF [4], BRISK [14]). Thus, one could employ different combinations of keypoint detection and descriptor extraction depending on the particular application.

The amount of descriptors extracted per image depends on the chosen method, its parameters, the size of the image, as well as the image contents (images containing trees with many leaves, or rough surfaces, will give many more descriptors than images of flat surfaces). The memory required to store the local descriptors of an image is simply the memory consumed by a single extracted local descriptor, multiplied by the amount of all extracted descriptors for that image. Memory consumption of a single descriptor depends on the description technique used. For example, each SIFT descriptor has 128 dimensions, which, if they are represented by chars, it will consume 128 bytes per descriptor. Then, if we have a set of one million images, and let's assume that the images produce 1,000 SIFT descriptors in average (an optimistic number with real world images), we would need: $128 \text{ B} \times 1,000 \times 1,000,000 \sim 120$ GB to store all descriptors. A dataset of this size *does not fit in RAM* for current standard hardware machines, and neither does not scale well if we add more images.

Therefore, in order to achieve a fast image-search engine in a dataset that scales well with the number of objects, it is necessary to keep the used memory space limited and perform very fast query searches using efficient data structures.

1.1 Related Work

There are currently two main approaches to face the image similarity search problem. The first approach known as "holistic" uses global image descriptors such as [19, 24, 25, 11, 22]. The second one is based on processing local invariant descriptors, and often employs a variant of the "bag of features" (BoF) technique. Both approaches have their strengths and weaknesses, however, BoF has proven to be good enough to work with image sets of the order of one million images with a standard consumer hardware, and achieving better accuracy than holistic methods.

In this work the Bag of Features approach is studied, in particular the variant called vocabulary tree. The BoF representation is built from a local descriptors set. These methods first extract local descriptors for all the images, generating a set of descriptors. Most authors use SIFT [15] as description technique. The BoF main idea comes from text search engines that, in particular, exploit inverted files indexing data structures [27]. An inverted file or inverted index is a convenient data structure to compute distances between sparse vectors using *Term Frequency, Inverse Document Frequency* (TF-IDF) weighting mechanisms. Typically K-means clustering is applied over the descriptors set to create a set of *words*, in image retrieval this set is called visual vocabulary. Each descriptor is then quantized in its nearest visual word (or nearest cluster centroid) making the visual vocabulary.

Josef Sivic and Andrew Zisserman originally proposed this technique in [23]. Since then different authors have contributed with improvements. David Nistér and Henrik Stewénius [18] proposed using hierarchical K-means, to arrange visual words in a tree. Their approach, named "vocabulary tree", allows to work with a bigger dataset and the potential addition of new images at runtime. Works as [13] and [20] used randomized kd-trees, allowing to build the vocabulary much faster, without losing performance. In [20] a variant of RANSAC [5] is used to re-rank the obtained results using local spatial consistency verification of the keypoints distribution, improving the search results. In [17] two methods are compared: hierarchical K-means (HKM) and Randomized kd-trees (RKD). The authors show that the performance depends on the particular dataset considered and the system requirements: precision, vocabulary building time, memory constraints, etc. In [12] two contributions are proposed. The first one is a contextual dissimilarity measure correction factor, which improves precision considerably. And the second one allows to speed up query time using an inverted index structure of two levels. In [9] and [10] additional information is embedded into the descriptors to improve the results. In the work of Zhang et al. [26] a greater descriptiveness of the local descriptors is explored in order to construct the vocabulary. Also, the concept of Bag of Features is extended with visual phrases, which have a context and a spatial disposition, in the same way words have in a text document.

The use of the SIFT descriptor in BoF has proven to be successful, but the computation is expensive compared to other description techniques. In [7], the use of ORB [21] within the BoF approach is used to save computation time, showing that computing ORB is up to two orders of magnitude faster than SIFT.

1.2 Contributions

This work proposes a careful study and implementation of the vocabulary tree method from David Nistér and Henrik Stewénius [18]. Nistér's work produced a high impact on the computer vision community, having more than 3200 citations to date (2017) and being awarded with the CVPR's Longuet-Higgins prize in 2016. The vocabulary tree approach is a powerful yet simple and intuitive method, used in image-search and classification. It is commonly used in applications requiring fast queries and low memory consumption while dealing with a large number of images to be processed.

In this publication, a complete C++ implementation of the vocabulary tree for an image-search system is described. This implementation allows to use different descriptors and parameters in order to compare their performance in different scenarios. Also, it is released as open source and available online for research reproducibility.

This work is organized as follows. Bag of Features is covered in detail in Section 2. Section 3 introduces the vocabulary tree approach and the derived large-scale image-search system. Results and experimental evaluation over public domain datasets, parameters analysis and method variations, are carried out in Section 4. Finally, Section 3.4 depicts several examples with possible applications of this method. Section 6 presents conclusions and future work.

2 Bag of Features Representation

The Bag of Features representation was originally proposed by Sivic and Zisserman in [23]. The idea of Bag of Features was taken from text retrieval systems.

Text retrieval systems generally employ a number of standard steps. The documents are first parsed into words. Second, the words are represented by their stems, for example "walk", "walking" and "walks" would be represented by the stem "walk". Third, a stop list is used to reject very common words, such as "the" and "an", which occur in most documents and therefore are not discriminative for a particular document. The remaining words are then assigned a unique identifier, and each document is represented by a vector with components given by the frequency of occurrence of the words the document contains. All of the above steps are carried out before the retrieval, and the set of vectors representing all the documents in a corpus are organized as an inverted file to facilitate efficient retrieval. An inverted file or inverted index is structured like an ideal book index. It has an entry for each word in the corpus followed by a list of all the documents (and position in that document) in which the word occurs. A text is retrieved by computing its vector of word frequencies and returning the documents with the closest vectors. In addition, the ordering and separation of the words may be used to rank the returned documents.

For image retrieval, instead of using text words, the idea is to use "visual words". In Sivic and Zisserman's work, descriptors are clustered using the K-means algorithm, although other clustering methods are possible. Once the clustering is done, each cluster center represents a visual word, and the union of all the visual words represents the visual vocabulary. When a new image is processed, each descriptor of this image is assigned to the nearest cluster, and this immediately generates matches for all the images in the image set.

The BoF representation of an image is a vector $d \in \mathbb{R}^n$, that can be seen as a histogram counting the frequencies of each visual word in the image. This histogram is weighted using TF-IDF (see Section 2). Figure 1 shows an example of this representation. Images are compared by computing the L_p norm between these BoF vectors to measure their similarity.

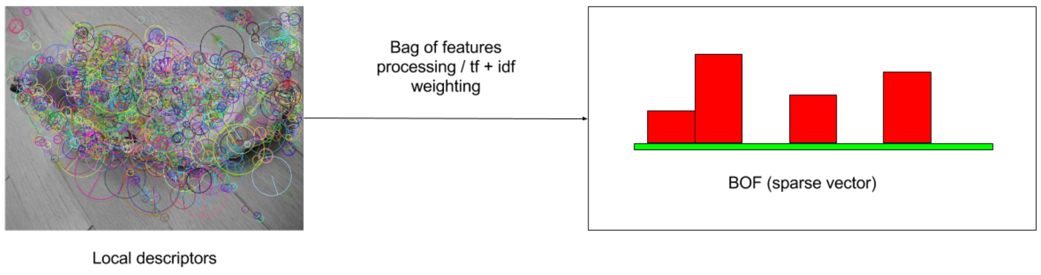


Figure 1: Each BoF d can be thought as a sparse vector histogram of visual words frequencies.

Object retrieval exploits the use of inverted index or inverted files. In a classical file structure all the words are stored in the document they appear in. An inverted file structure has an entry for each word where all occurrences of the word in all documents are stored. Sivic and Zisserman used an inverted file with an entry for each visual word, which stores all the occurrences of the same visual word in all the images in the set, see Figure 2. Since the BoF vector is very sparse, the use of an inverted index makes the retrieval very fast.

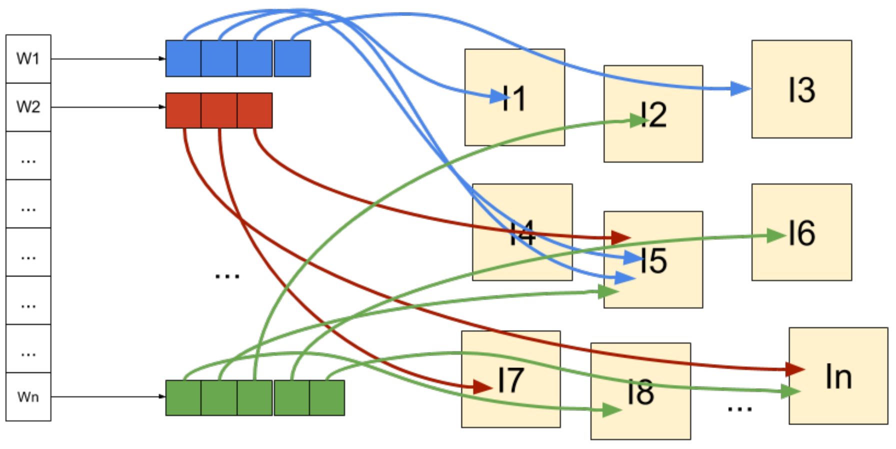


Figure 2: Each cell in the vertical vector represents a visual word, making the visual vocabulary. Each visual word has a list of pointers to the images where this visual word occurs.

2.1 Visual Indexing and Weighting with TF-IDF

In text retrieval each document is represented by a vector of word frequencies. However, it is usual to apply a weighting to the components of this vector, rather than use the frequency vector directly for indexing. The weighting mechanism employed is known as "Term Frequency, Inverse Document Frequency" (TF-IDF). Given a vocabulary of V words, each visual word $i = 1, \dots, V$ is assigned a weight $w_i \in \mathbb{R}$ based on entropy

$$w_i = \log \frac{N}{N_i}, \tag{1}$$

where N_i is the number of occurrences of the word i in the whole database and N is the number of total training images. Note that this weighting mechanism penalizes the most frequent words (least descriptive) with lower weights. In text retrieval, this weight w_i is commonly known as inverse document frequency (idf).

Each processed image j has descriptors that are quantized into visual words. In order to measure the relevance of a particular word i in image j, the term frequency n_i^j is defined as follows

$$n_i^j = \frac{\text{the number of times that word } i \text{ appears in image } j}{\text{the total count of words in image } j}. \tag{2}$$

Thus, each image j produces a Bag of Features d^j, defined as follows

$$d_i^j = n_i^j \cdot w_i. \tag{3}$$

2.2 Geometric Consistency

Text retrieval algorithms increase the ranking for documents where the searched words appear close together in the retrieved texts (measured by word order). Analogously, the disposition of visual words within the images can be used to improve retrieval performance, discarding or re-ranking images where visual words disposition is not geometrically consistent with the words in the query. Geometric consistency can be measured quite loosely simply by requiring that neighboring matches in the query region lie in a surrounding area in the retrieved image. It can also be measured very strictly by requiring that neighboring matches have the same spatial layout in the query region and retrieved image. This kind of geometric consistency considerations could be time consuming, but since it is only applied over the short ranked list of retrieved images, it can be implemented in such a way that does not degrade the algorithm retrieval time.

2.3 Bag of Features Method Summary

In order to satisfy response-time constraints, the BoF technique has two phases: off-line phase and on-line phase.

Off-line phase (or training phase). It will be run once, in order to create the visual vocabulary and the inverted index. This phase can take a considerable amount of time, depending on the number of input images, and the local feature technique employed (for example SIFT is much more time consuming than ORB). In this phase, first, local descriptors are extracted from all the images in the dataset. Then, a clustering technique is applied in order to build the visual vocabulary. All descriptors are quantized to its nearest visual words, producing sparse vectors named Bag of Features. Finally, BoFs quantizations of dataset descriptors are arranged in a convenient way using an inverted index. Note that the inverted index does not depend on the training image dataset and can be computed using a smaller, bigger or a different vocabulary.

On-line phase (or run-time phase). When the training phase is completed, multiple queries can be returned quickly. Figure 3 summarizes the main steps to solve the query. When the system receives a query image, local descriptors are extracted for that query image. The resulting descriptors are quantized to its nearest visual words, generating the Bag of Features for this query. The query BoF is compared with the BoFs computed in the training phase. This comparison process is performed efficiently using the inverted index, producing a short list of best ranked images. A further process will improve the output by checking the geometric consistency on the short list to re-rank or filter the results.

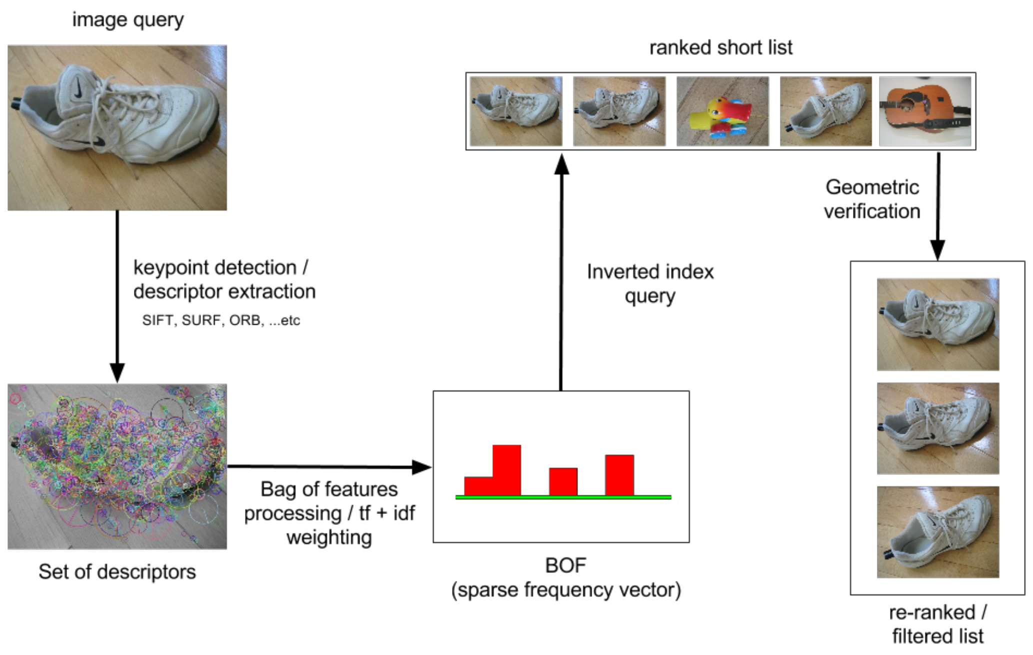


image query
keypoint detection / descriptor extraction
SIFT, SURF, ORB, ...etc
Set of descriptors
Bag of features processing / tf + idf weighting
BOF
(sparse frequency vector)
Inverted index query
ranked short list
Geometric verification
re-ranked / filtered list

Figure 3: BoF query process summary

3 Vocabulary Tree Approach

Nistér et al. [18] approach generalizes Sivic's idea using hierarchical clustering, while generating a "visual vocabulary tree", and results a particular case of considering max tree level equal to 1. This approach has several advantages over the previous work. It allows to use a larger vocabulary, traverse smaller portions of the database to perform the query, and perform potential on-the-fly insertion of new objects into the database in the run-time phase.

In this Section, we describe how to build a visual vocabulary and perform a query, corresponding to the training and run-time phase of the BoF method.

3.1 Training Phase

Algorithm 1 shows a high level overview of the training phase process. First, it extracts descriptors for all the images in the dataset, second it builds the visual vocabulary, then the inverted indexes for each leaf, and finally computes the BoFs. The inverted index structure can be thought as an image database which can be queried several times efficiently.

Algorithm 1: Training phase overview

```
TrainingPhase
input  : $\Gamma$ a set of $N$ images, $K$ number of children per node, $H$ maximum height for the tree
output: $T$ the vocabulary tree, $BoFs$ the bag of features vectors
begin
    $D \leftarrow$ ExtractDescriptors($\Gamma$)
    $T_0 \leftarrow$ BuildVocabularyTree($D, 0, K, H$)
    $T \leftarrow$ BuildInvertedIndex($D, T_0$)
    $BoFs \leftarrow$ ComputeBoFs($T$)
```

Keypoint detection can be expressed as a function $Detect_{md}$ that takes an image as input and returns a set of keypoints.

$$Detect_{md}(I) = \{ft_1, \dots, ft_n\}, \tag{4}$$

where md is a keypoint detection method technique. Generally, descriptor extraction can be expressed as a function $Extract_{mx}$ that takes an image I and a set of detected keypoints Ft, and returns a set of local descriptors vectors.

$$Extract_{mx}(I, Ft) = \{\alpha_1, \dots, \alpha_n\}, \tag{5}$$

where mx is a local descriptor method and $\alpha_i \in \mathbb{R}^D$ are the are the resulting descriptor vectors of the image I. Descriptor extraction is applied to all the images of Γ storing the result descriptors into a set. Algorithm 2 shows this process.

3.1.1 Building the Visual Vocabulary

K-clustering is applied on the descriptors set, generating K cluster centers. This K centers are used as the first level of nodes in the vocabulary tree. The descriptors set is then split into K subsets, assigning each descriptor to its closest center (called visual word). In the same way, K-clustering is recursively applied for each subset. This process is repeated until the maximum tree height H is reached or the descriptor set is empty. Figure 4 shows an example of this procedure.

Algorithm 3 shows a pseudo-code of tree building, where $|D|$ is the number of elements in the set D. The **ComputeClusters** function performs descriptor clustering. For clustering, K-means or

Algorithm 2: Descriptors extraction procedure.

```
ExtractDescriptors
input  : $\Gamma$ a set of $N$ images
output: $D$ the set of extracted descriptors
begin
    $D \leftarrow \{\}$
    foreach image $I_j \in \Gamma$ do
        $Keypoints_j \leftarrow \textbf{Detect}_{md}(I_j)$
        $Descriptors_j \leftarrow \textbf{Extract}_{mx}(I_j, Keypoints_j)$
        $D \leftarrow D \cup Descriptors_j$
```

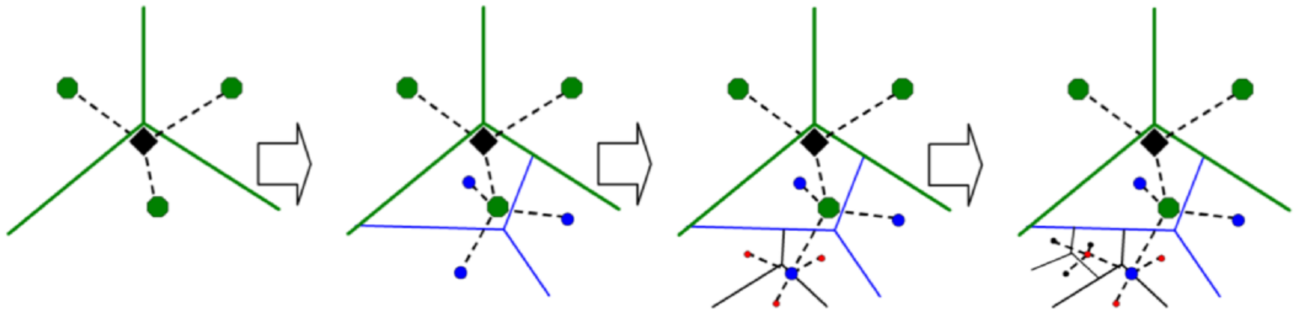


Figure 4: Nistér's hierarchical clustering. From left to right a new level is added to the tree. Same color denotes nodes at the same level on the tree, the root node is black. Figure taken from [18].

any other clustering technique can be used. The **GetCentroid** function returns the centroid for the given cluster. If K-means is used, the centroid will be the mean.

For a tree node T, $T.children$ represents its children node set, $T.weight$ its weighting value, $T.\mu$ its centroid and $\#T$ the number of nodes of its subtree. Also, leaf nodes have the property $T.InvertedIndex$ representing the bag of inverted indexes computed for that node. To see an example of how this algorithm works, see Section 3.4.

When comparing binary descriptor vectors (ORB, AKAZE, etc), usually the Hamming distance is employed, so the mean vector is undefined. In [7] a workaround to this problem is introduced allowing to use the vocabulary tree technique with binary descriptors. This clustering algorithm is called "K-majority". K-majority uses a voting scheme to decide which components of the centroid would have 0 or 1. It can also be seen as applying K-means over binary vectors, by reinterpreting them as real-valued vectors where each component is 1.0 or 0.0. In this way it is feasible to calculate the mean. Mean vector components are rounded to 0 or 1 to reinterpret it as a binary vector.

3.1.2 Building the Inverted Index

After the vocabulary tree is created, it is time to build the inverted index. Algorithm 4 shows this process. The inverted index structure is physically stored as small inverted indexes in the leaf nodes. Although it could be a separate structure, having the inverted index directly on the leaves of the tree is important to reflect the efficiency of the method to obtain all the BoFs.

At the beginning each leaf starts with an empty inverted index. An inverted index is a bag (a set with repetitions) which stores the ids of images as elements. Given a bag b, N_b is defined as the number of different elements in b, and n_b^i as the number of repetitions of each i in b. Also, given two

Algorithm 3: Process to create the visual vocabulary tree

```
BuildVocabularyTree
input  : $D$ a set of descriptors, $L$ current level of the tree, $K$ number of children per node, $H$
         maximum height for the tree
output: $T$ the vocabulary tree
begin
    if $|D| < K$ or $L \geq H$ then
        $T \leftarrow$ build a new leaf node
    else
        $Clusters \leftarrow$ ComputeClusters($D, K$)
        $T \leftarrow$ build a new branch node
        $T.children \leftarrow \{\}$
        foreach cluster $C \in Clusters$ do
            $Child \leftarrow$ BuildVocabularyTree($C, L+1, K, H$)
            $Child.\mu \leftarrow$ GetCentroid($C$)
            $T.children \leftarrow T.children \cup \{Child\}$
```

bags, $\cup_{bag}$ is defined as the union function that sums repetitions if an element is on both bags.

Each descriptor vector of the image j is traversed along the tree from the root to the leaves following in each level the nearest center (e.g. according to the L_2 norm). When the descriptor reaches a leaf, one repetition for j is added into the multi-set of that leaf. Later, inverted indexes for branch nodes (virtual inverted indexes), will be computed combining inverted indexes of children nodes. To see an example of how this algorithm works, see Section 3.4.

Note that the norm employed by **FindLeaf** is the same norm that was used for clustering: L_2 norm for K-means, and Hamming if K-majority is used.

Algorithm 4: Procedure to build the inverted index given a vocabulary tree.

```
BuildInvertedIndex
input  : $D$ a set of descriptors, $T$ vocabulary tree
output: $T$ tree with inverted indexes on its leaves
begin
    foreach descriptor $\alpha^j \in D$ do
        $A \leftarrow$ FindLeaf($T, \alpha^j$)
        $A.InvertedIndex \leftarrow A.InvertedIndex \cup_{bag} \{j\}$

FindLeaf
input  : $T$ the vocabulary tree, $\alpha^j$ a descriptor of the image $j$
output: $A$ the leaf after traversing $\alpha^j$
begin
    $A \leftarrow T$
    while $A$ is not a Leaf do
        $A \leftarrow \underset{C \in A.children}{\operatorname{argmin}} ||\alpha^j - C.\mu||$
```

3.1.3 Building the Bag of Features

In [18] different ways to adapt the weighting mechanisms are evaluated. Term frequency (TF) and inverse document frequency (IDF) are proposed in the following way:

IDF. Each node i is assigned with a weight $w_i \in \mathbb{R}$ based on entropy

$$w_i = \log \frac{N}{N_i}, \tag{6}$$

where N is the number of images in the database, and N_i is the number of images in the database with at least one descriptor vector path through node i.

Note N_i is the number of different elements of the inverted index of the subtree with node i as root. If node i is a leaf, N_i is simply the number of different elements of the inverted index stored on that node. If node i is a branch, the virtual inverted index will be computed as the union of its children inverted index. Note that, TF and BoF for each query descriptor are actually computed at run-time phase, and are computed in an analogous way for the images in the database.

TF. Let $n_i^j \in \mathbb{N}$ be the number of descriptor vectors of the image j with a path through the node i. Similarly, let $m_i \in \mathbb{N}$ be the number of descriptor vectors of the query image with a path through the node i.

***d*-vectors and *q*-vector.** Let $q \in \mathbb{R}^n$, $q = (q_1, \dots, q_n)$ be the query BoF-vector and $d^j \in \mathbb{R}^n$, $d^j = (d_1^j, \dots, d_n^j)$ be a database BoF-vector, defined as follows

$$\begin{aligned} d_i^j &= n_i^j w_i, \\ q_i &= m_i w_i, \end{aligned} \tag{7}$$

where j is the database image index, $1 \leq j \leq N$, and n is the number of tree nodes.

Note that n_i^j is the number of repetitions of element j in the inverted index of node i. Also, this equation establishes a mapping between the nodes and the components of d-vectors or q-vector. Each node is mapped to a different d component. This vectors can also be thought as a big sparse matrix with N rows mapped to the database images and n columns mapped to tree nodes. Sub-index A for d_A^j is used to indicate the component of d^j that is mapped to node A.

Algorithm 5 shows a pseudo-code for d-vectors computation. It iterates over all the nodes in the tree, and for each node computes its inverted index. Then it computes the node weight, and for each image in the inverted index computes the corresponding d-vector component value. This d-vector component is stored in the inverted index of the node as a pair $\langle j, d_i^j \rangle$. The function **getVirtualInvertedIndexes** returns the inverted index for the specified node.

3.1.4 Normalizing BoFs

To achieve fairness between database images with few and many descriptor vectors, a L_p-norm normalization is made to each vector

$$\bar{d^j} = \frac{d^j}{||d^j||}, \quad \bar{q} = \frac{q}{||q||}.$$

As suggested, L_1-norm normalization is done for this implementation, and it is computed as the sum of all the components of each database vector and dividing each component by each sum. For an example see "d-vectors normalization" on Section 3.4.

Algorithm 5: Procedure for Bag of Features computation.

```
ComputeBoFs
input  : $T$ a vocabulary tree with inverted indexes, $N$ number of images
output: $T$ tree nodes with its weight computed
output: $BoFs$ the bag of features
begin
    $BoFs \leftarrow$ new sparse matrix of size ( $N \times \#T$ )
    foreach $node\ index\ i < \#T$ do
        $A \leftarrow T[i]$
        $ii \leftarrow$ getVirtualInvertedIndexes($A$)
        $A.weight \leftarrow \log(\frac{N}{N_{ii}})$
        foreach $j \in ii$ do
            $BoFs[j, i] \leftarrow A.weight * n_{ii}^{j}$
    // BoFs normalization
    foreach $j < N$ do
        $d \leftarrow BoFs[j].row$
        $\bar{d} \leftarrow \frac{d}{||d||_p^p}$
        $BoFs[j].row \leftarrow \bar{d}$

getVirtualInvertedIndexes
input  : $A$ a tree node
output: $ii$ a bag containing the inverted indexes of A
begin
    if $A$ is a Leaf then
        // a real inverted index
        $ii \leftarrow A.invertedIndex$
    else
        // computing virtual inverted index
        $ii \leftarrow \{\}_{bag}$
        foreach $node\ C \in A.children$ do
            $cii \leftarrow$ getVirtualInvertedIndexes($C$)
            $ii \leftarrow ii \cup_{bag} cii$
```

3.2 Runtime Phase

Scoring. The score between a query and an image j is defined as the distance between its normalized BoF vectors $\bar{q}$ and $\bar{d^j}$. The score is computed as the p-th power of the L_p-norm of the difference

$$s^j(\bar{q}, \bar{d^j}) = ||\bar{q} - \bar{d^j}||_p^p. \tag{8}$$

Without the p-th power, this metric is also known as Minkowsky distance, and can be considered as a generalization of both the Euclidean distance and the Manhattan distance. In [18] the authors show that if the vectors are normalized, any Minkowsky distance can be computed by inspecting only components where both vector components are non zero. Equation (9) is then used to compute the scoring. Because it will be applied to BoFs, in a real world case most of the components will have zero value. This will reduce considerably the number of operations.

$$||\bar{q} - \bar{d^j}||_p^p = 2 + \sum_{i|\bar{d}_i^j \neq 0, \bar{q}_i \neq 0}^{n} \left(|\bar{q}_i - \bar{d}_i^j|^p - |\bar{q}_i|^p - |\bar{d}_i^j|^p \right). \tag{9}$$

3.2.1 Performing a Query

Algorithm 6 shows the query process. Descriptor extraction is performed over Q with the same function we used for training images. Function **ComputeQBoF** accumulates weights on q components while traverses each descriptor of Q through the tree from root to leaves. The function **indexOf**(A) returns the index of the *q-vector* component mapped to the node A, and it can be computed in $O(1)$ time. Equation (9) is applied on $\bar{d}$ and $\bar{q}$ vectors to compute the scoring, only components where q is different from zero are evaluated.

3.3 Vocabulary Tree Memory Consumption

Once K and H are fixed, we can estimate some aspects of memory consumption. For example each level of the tree has at most K children, then the maximum number of nodes is limited by the geometric sum

$$\sum_{i=0}^{H} K^i = \frac{K^{H+1} - 1}{K - 1}. \tag{10}$$

Since each node stores a D-dimensional cluster center or visual word, the amount of memory needed to store the vocabulary, will be the number of used nodes multiplied by the consumption of a single visual word. For example, using $D = 128$, $K = 10$ and $H = 6$, it will produce at most $1,111,111$ cluster centers (or nodes), and it would consume less than $136MB$ of memory. This amount of memory is promising in terms of scale because is fixed for any database size, and can perfectly fit in RAM. It is important to note that, the data type used to represent the components of feature vectors also affects the memory consumption, and this is not a minor decision. In the case of SIFT descriptors, most common implementations use integer values between 0 and 127, thus it is possible to represent them using 1 byte for each dimension which in $C++$ would be *unsigned char* data type. However, real-valued representations may use a $C++$ *float* data type which consumes 4 bytes. In the above example, a floating point representation requires $136MB * 4 = 544MB$ and still perfectly fits in RAM. Consequently, the float data type is a trade-off between efficiency and amount of memory required for the vocabulary. Note that we considered only the amount of memory needed to store the vocabulary, however other data structures may be needed to compute the score.

3.4 Example

Suppose we have a set with three images $\Gamma = \{I_1, I_2, I_3\}$. Now we apply SIFT descriptor extraction over all the images in the set to obtain the following image descriptors: $Extract_{SIFT}(I_1) = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, $Extract_{SIFT}(I_2) = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ and $Extract_{SIFT}(I_3) = \{\gamma_1, \gamma_2, \gamma_3\}$. Since we used SIFT, each descriptor belongs to $\mathbb{R}^{128}$.

Building the Tree

Figure 5 shows an example of the tree building process. Note that when a node is created, the original set used to compute cluster centers is no longer needed, so it can be freed in order to gain memory.

Algorithm 6: Procedure to perform a query.

```
Query
input  : $Q$ the image query
input  : $T$ the vocabulary tree
input  : $BoFs$ the set of precomputed bag of features
input  : $t$ size of the resulting short-list
output: $R$ the short-list with the indexes of top ranked results
begin
    $E \leftarrow$ ExtractDescriptors($Q$)
    $q \leftarrow$ ComputeQBoF($E, T$)
    $\bar{q} \leftarrow \frac{q}{||q||_p^p}$
    $X \leftarrow \{ i \mid 1 \leq i \leq \#T$ and $\bar{q}_i \neq 0\}$
    $S \leftarrow \{\}$
    for $1 \leq j \leq N$ do
        $d^j \leftarrow BoFs[j]$
        $score \leftarrow 2$
        foreach $i \in X$ do
            $score \leftarrow score + |\bar{q}_i - \bar{d}_i^j|^p - |\bar{q}_i|^p - |\bar{d}_i^j|^p$
        $s_i \leftarrow \langle i, score \rangle$
        $S \leftarrow S \cup \{s_i\}$
    $R \leftarrow [r_1, r_2, \ldots, r_t]$ where $r_i.score < r_j.score$ / $r_i, r_j \in S$ and $1 \leq i < j \leq t$

ComputeQBoF
input  : $E$ a set of descriptors
input  : $T$ the vocabulary tree
output: $q$ the BoF for the query
begin
    $q \leftarrow$ new sparse vector of size $\#T$
    foreach $\alpha$ in $E$ do
        $A \leftarrow T$
        while $A$ is not a Leaf do
            $A \leftarrow \underset{C \in A.children}{\operatorname{argmin}} ||\alpha - C.\mu||$
            $i \leftarrow$ indexOf($A$)
            $q_i \leftarrow q_i + A.weight$
```

Construction of the Inverted Index

Figure 6 shows the inverted index construction of the example described above, and Figure 7 shows inverted indexes for three example nodes. In order to reduce memory consumption actually inverted indexes are stored only in the leaves of the tree. For the internal nodes, virtual inverted indexes are recursively computed combining child inverted indexes. One efficient way to implement the virtual inverted indexes is to store inverted indexes entries ordered by image id, and merge them using a method similar to the merge part of the merge-sort algorithm.

The weights for these nodes are

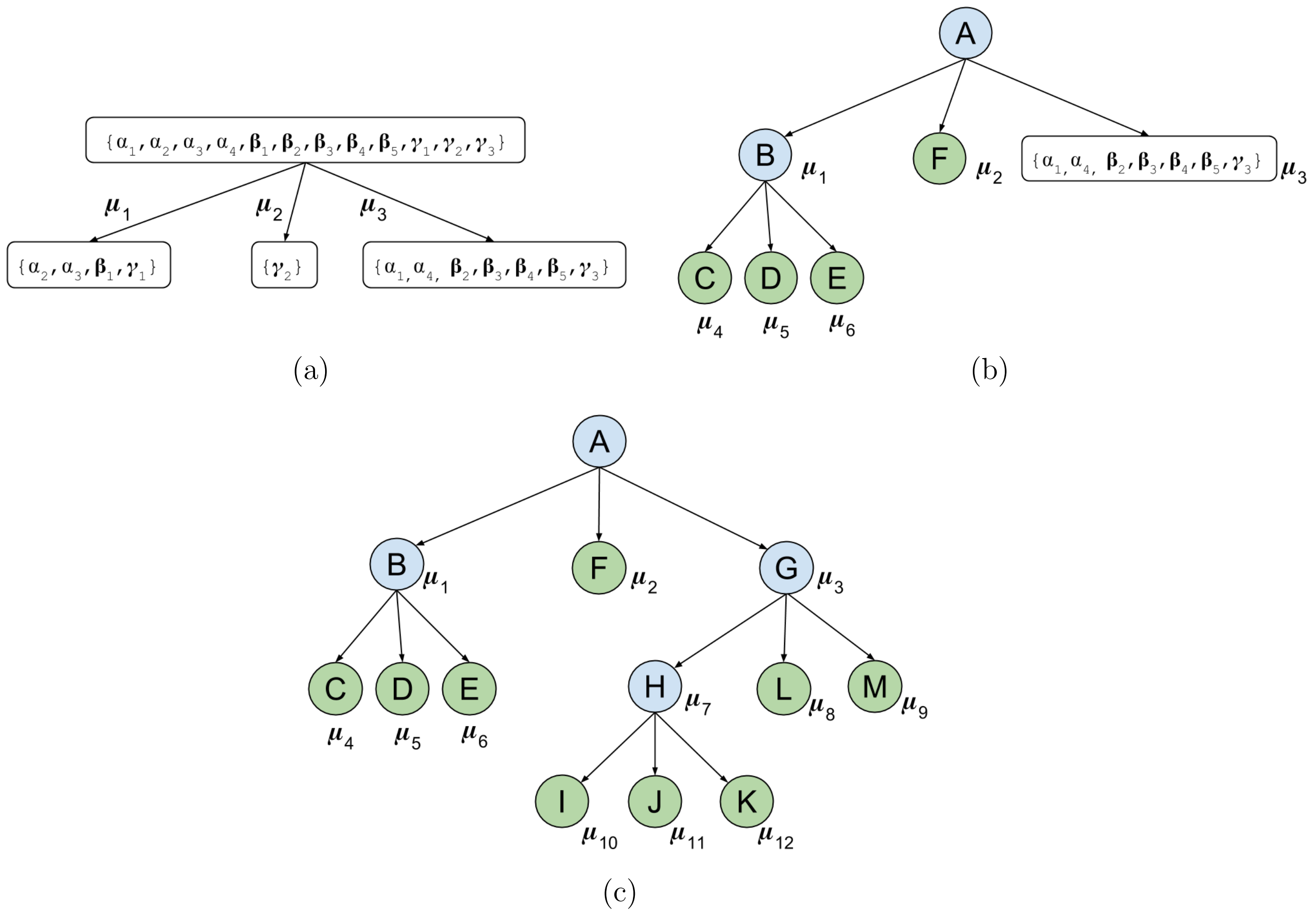


Figure 5: Tree building process example using $K = 3$ parameter for K-means, and $H = 4$ as maximum height for the tree. (a) The extracted descriptors are put together in a set and clustered with K-means. The result is 3 different clusters, each one with its corresponding cluster center μ_1, μ_2 and μ_3. This process is applied recursively. (b) An intermediate step. (c) The resulting tree. Since the tree has reached the maximum height $H = 4$, the leaf nodes *I*, *J* and *K* cannot be split further.

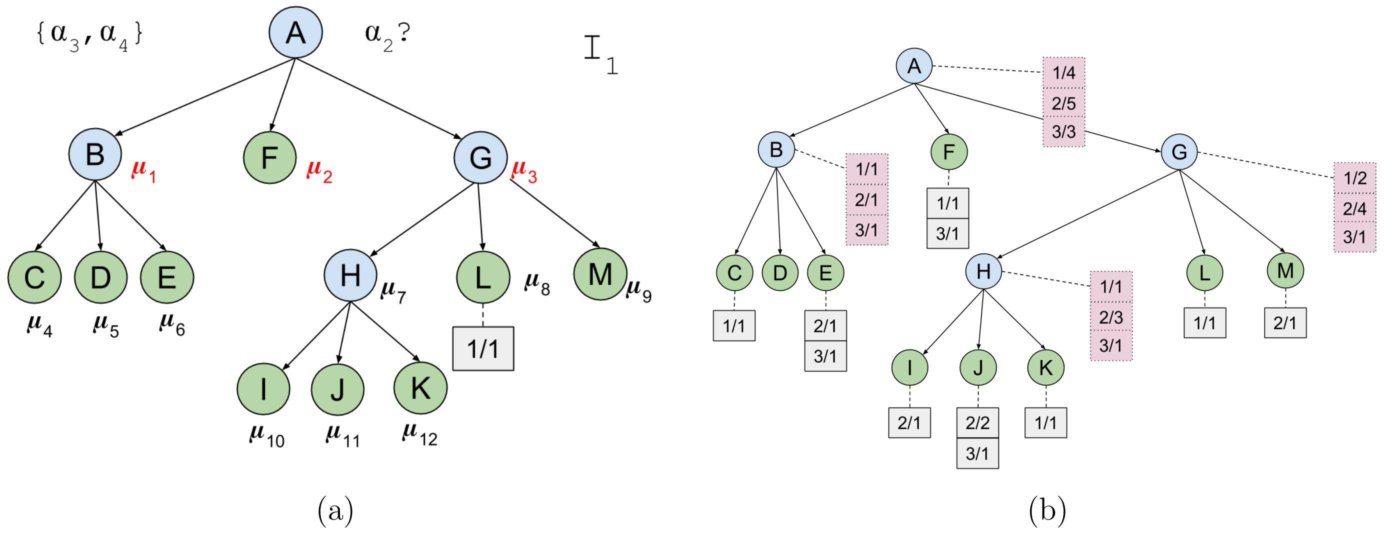


Figure 6: Inverted indexes construction. (a) The first descriptor α_1 of the first image 1 has been added. (b) The same procedure has been repeated with all the remaining descriptors of image 1, and the descriptors of all the remaining images in the dataset.

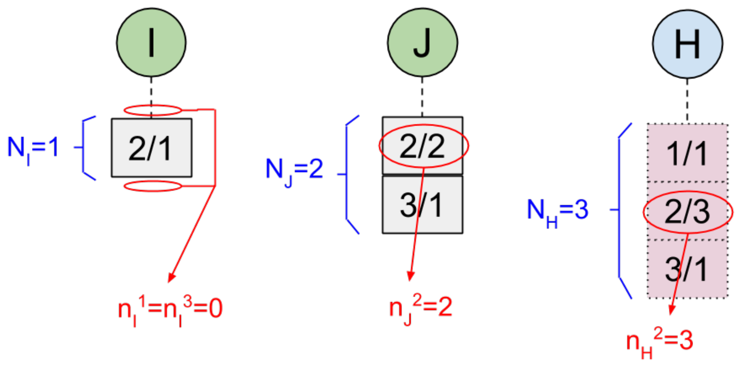


Figure 7: Inverted index for three example nodes *I*, *J* and *H*. As there is only one image with descriptors that passes through node *I* (the image 2) then $N_I = 1$. In the same way, there are two different images with descriptors that pass through node *J* (images 2 and 3) then $N_J = 2$, and all the images of the database pass through node *H*, then $N_H = 3$.

$$\begin{aligned} w_I &= \log N/N_I &&= \log 3/1 &&\approx 0.47712, \\ w_J &= \log N/N_J &&= \log 3/2 &&\approx 0.17609, \\ w_H &= \log N/N_H &&= \log 3/3 &&= 0. \end{aligned}$$

Note that the weight is smaller when the node has more different images with descriptors that pass through it. For the root node H, since all the images passes through it, does not provide any information (it weights 0). Table 1 shows the complete weight vector for the example.

	A	B	C	D	E	F	G	H	I	J	K	L	M
$w =$	0	0	$\log 3$	0	$\log 3$	$\log 3/2$	0	0	$\log 3$	$\log 3/2$	$\log 3$	$\log 3$	$\log 3$

Table 1: weight vector example

Figure 7 shows n_i^j, the number of repetitions in the inverted index entry. For example, for node I we have no inverted index entries for images 1 and 3, then $n_I^1 = n_I^3 = 0$, and $n_I^2 = 1$. For node J we have $n_J^1 = 0$, $n_J^2 = 2$ and $n_J^3 = 1$. And for node H we have $n_H^1 = 1$, $n_H^2 = 3$ and $n_H^3 = 1$. Table 2 shows n-vectors for the complete example.

	A	B	C	D	E	F	G	H	I	J	K	L	M
$n^1 =$	4	1	1	0	0	1	2	1	0	0	1	1	0
$n^2 =$	5	1	0	0	1	0	4	3	1	2	0	0	1
$n^3 =$	3	1	0	0	1	1	1	1	0	1	0	0	0

Table 2: n-vectors example

Using Equation (7) we compute d-vectors and obtain the results shown in Table 3.

	A	B	C	D	E	F	G	H	I	J	K	L	M
$d^1 =$	0	0	$\log 3$	0	0	$\log 3/2$	0	0	0	0	$\log 3$	$\log 3$	0
$d^2 =$	0	0	0	0	$\log 3$	0	0	0	$\log 3$	$2\ \log 3/2$	0	0	$\log 3$
$d^3 =$	0	0	0	0	$\log 3$	$\log 3/2$	0	0	0	$\log 3/2$	0	0	0

Table 3: d-vectors example

The d-vectors can be represented as a vector of pointer-to-vector pairs. These pairs have the image id as first component and the d value as second component, and can be computed at the same time that the inverted index is built.

Performing a Query

Let Q be a query image. SIFT descriptor extraction over Q is applied in the same way we did with the images in the entire image set, and then we obtain the following image descriptors: $Extract_{SIFT}(Q) = \{\delta_1, \delta_2, \delta_3, \delta_4\}$.

First, the vector $m \in \mathbb{R}^n$ is initialized with zeros. Then, the tree is traversed by each query descriptor δ_i, following in each level the nearest center, until a leaf is reached. While the tree is traversed, the m_j position corresponding to the node with nearest center is incremented. Figure 8 shows the result when all the descriptors of Q have been traversed in the tree, and vector m is completely computed. Table 4 shows the vectors m, w and q for the example above. The vector q is

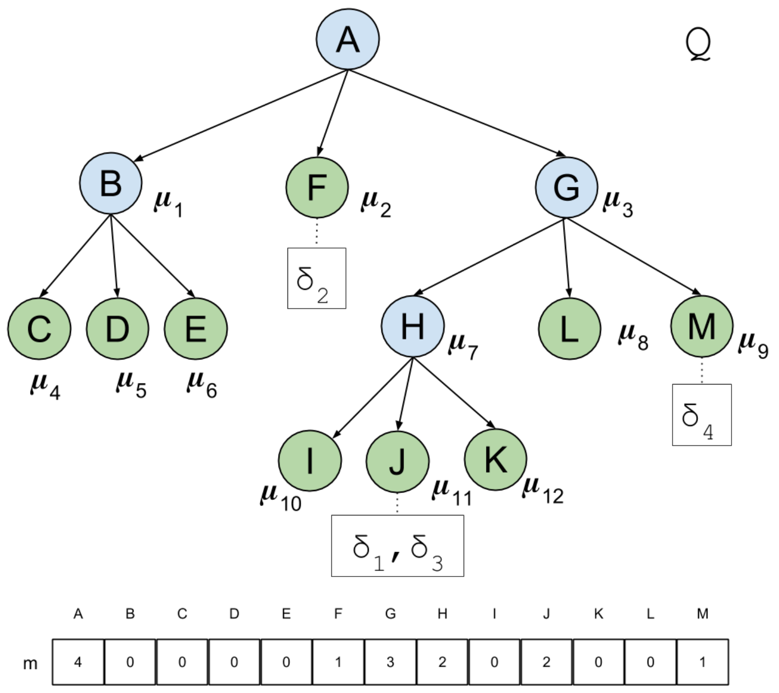


Figure 8: Vector m computed for query image Q with descriptors $\delta_1 \ldots \delta_4$.

computed as the element-wise multiplication of m and w (see Equation (7)).

	A	B	C	D	E	F	G	H	I	J	K	L	M
$m =$	4	0	0	0	0	1	3	2	0	2	0	0	1
$w =$	0	0	$\log 3$	0	$\log 3$	$\log 3/2$	0	0	$\log 3$	$\log 3$	$\log 3$	$\log 3$	$\log 3$
$q =$	0	0	0	0	0	$\log 3/2$	0	0	0	$2\log 3$	0	0	$\log 3$

Table 4: q-vector example

With the q vector computed, we need to normalize it using the L_1-norm to obtain $\bar{q}$. Finally, the score for each image in the database is computed using Equation (9). Note that it is only needed to perform operations over components where both vectors have values different from zero, in this case F, J and M. The resulting scores $s^1 \ldots s^3$ are

$$\begin{aligned} s^1 &\approx 1.78091, \\ s^2 &\approx 1.07005, \\ s^3 &\approx 1.35623. \end{aligned}$$

Therefore, for this example, the most similar image to Q is I_2, followed by I_3 and finally by I_1.

4 Results

In this section we perform the following experiments:

- Different combinations of K and H and descriptor extraction techniques: SIFT, SURF, KAZE, ORB and AKAZE.
- The impact on performance in the following cases:
  - Different sizes of the vocabulary.
  - Applying dimensionality reduction of descriptor vectors using Principal Component Analysis (PCA).
  - Increasing database size, up to 1 million images.

All the experiments were carried out using the L_1-norm for scoring, the L_2 norm for K-means clustering, and the Hamming distance when K-majority is used. Experiments using L_2-norm for scoring gave worse performance, and their results have not been shown here. Except when mentioned, the default parameters for descriptor extraction techniques were used.

4.1 Datasets

For all the experiments, we use the UKBench and Flikr1M datasets. However, we also tried the Holidays dataset [9] obtaining similar results and conclusions than UKBench, and therefore were not included in this work.

UKBench. Ukbench is the set of images used in [18] to perform their experiments. This dataset consists of pictures of different objects mostly taken indoors, such as toys, clocks and CD covers. The pictures were taken with different illumination, perspective and distance conditions. There are pictures of 2250 objects, each object has 4 pictures, totaling 10,200 images. All images have 640x480 pixels resolution and it can be downloaded from http://vis.uky.edu/~stewe/ukbench/.

MirFlickr1M. MirFlickr1M has 1 million Flickr images under the Creative Commons license. This set is commonly used for image retrieval evaluation and consist of images downloaded from the social photography site Flickr. It has over 600 Google Scholar citations to date (2017) and 32 thousand downloads from universities and companies worldwide. More information can be found here: http://press.liacs.nl/mirflickr/. In this work, MirFlickr1M images were used only to increase the database size and not to build the vocabulary tree.

4.2 Evaluation Metrics

In order to measure the algorithm effectiveness, a fixed image dataset Γ is considered. Each image in Γ belongs to exactly one known object.

Let $\mathcal{Q} \in \Gamma$ be the query set (images that will be used to test effectiveness), $\mathcal{V} \in \Gamma$ the images that were used to compute the vocabulary and $\mathcal{I} \in \Gamma$ the indexed images.

The ground truth is expressed as a function $isRelevant(q, r) : \mathcal{Q} \times \mathcal{I} \to \{0, 1\}$, defined as 1 if q and r are images of the same object, and 0 if not.

Now it is necessary to define a function $Query(q)$ that takes an image query $q \in \mathcal{Q}$, and gives as result a ranking for that query. A ranking R is a list of images of size t, $r_i \in \mathcal{I}$, $R = [r_1, r_2, \dots, r_t]$ sorted in descending order of relevance, that is: for each $1 \leq i \leq t$, r_i should be more relevant to q than r_{i+1}. Given a query q, the purpose of an evaluation criteria based on relevant images to q, is to compare two or more algorithms and decide which of them worked better in terms of the ground truth. We present two main metrics to evaluate retrieval performance: the metric used in [18] which we call $Metric_4$, and mAP (mean Average Precision), used in many other works [20, 10].

Metric$_4$. This metric is based on the UKBench dataset. Given a query image q it counts how many images of the same object are ranked within the top 4 better results. For a single image, it is defined as

$$metric_4\_image(q) = \sum_{i=1}^{4} isRelevant(q, r_i), \tag{11}$$

where $Query(q) = [r_1, r_2, r_3, r_4]$. It is basically a real number between 0 and 4. For example, getting all relevant results gives a score of 4, and getting none gives a score of 0. Each individual query performance is averaged for all the objects in the database, resulting in a global performance value

$$Metric_4(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} metric_4\_image(q). \tag{12}$$

$Metric_4$ assumes exactly 4 different images of the same object or class. Datasets usually do not have a fixed number of object classes. Moreover, if the performance is measured with a number between 0 and 4, it is hard to compare with results from other datasets.

mean Average Precision (mAP). The mean average precision (mAP) is a common metric used to evaluate retrieval effectiveness. This metric is similar to the previous one. The main difference is that this metric will favor algorithms where good results are put on top of the ranking.

Given a query image q its average precision is defined as follows

$$average\_precision(q) = \frac{\sum_{i=1}^{t} precision(q, i)}{total\_relevant(q)}, \tag{13}$$

where $total\_relevant(q)$ is the total relevant-to-q images in $\mathcal{I}$ and $precision(q, i)$ is the precision for q in the position i, defined as follows

$$precision(q, i) = \frac{\sum_{k=1}^{i} isRelevant(q, r_k)}{i}. \tag{14}$$

Then, combining all the individual average precisions, a global single value mAP is obtained according to

$$mAP(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} average\_precision(q). \tag{15}$$

Figure 9 shows an example of mAP calculation. mAP is commonly used because it combines precision as well as recall in a single value. A filter or re-rank stage will improve considerably this metric. In Niéster et al. [18] work, these improvements are not applied.

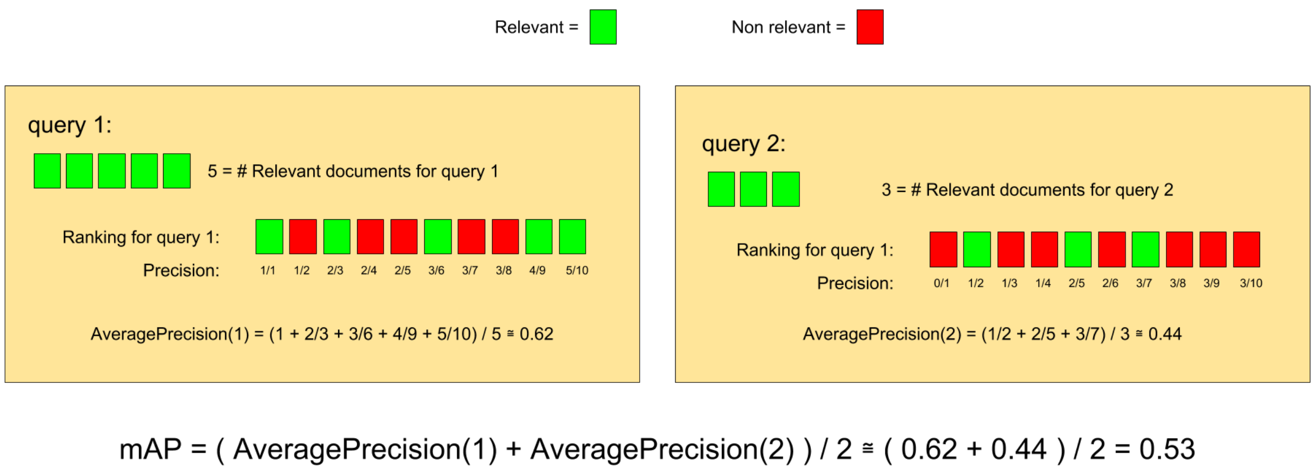

Relevant =
Non relevant =
query 1:
5 = # Relevant documents for query 1
Ranking for query 1:
Precision:
1/1 1/2 2/3 2/4 2/5 3/6 3/7 3/8 4/9 5/10
AveragePrecision(1) = (1 + 2/3 + 3/6 + 4/9 + 5/10) / 5 ≅ 0.62
query 2:
3 = # Relevant documents for query 2
Ranking for query 1:
Precision:
0/1 1/2 1/3 1/4 2/5 2/6 3/7 3/8 3/9 3/10
AveragePrecision(2) = (1/2 + 2/5 + 3/7) / 3 ≅ 0.44
mAP = (AveragePrecision(1) + AveragePrecision(2)) / 2 ≅ (0.62 + 0.44) / 2 = 0.53

Figure 9: Example of computing mean Average Precision.

4.3 K-H Best Values

We computed mAP varying different combinations of K and H, and different feature extraction algorithms, with the first 1000 images from the UKBench dataset, to find which combinations performs better. K and H were chosen, in such a way that vocabulary construction could fit in a fixed amount of RAM. Thus, for $H = 6$ only a few values of K were tested. Figures 10 and 11 show the mAP behavior when varying K between 8 and 32 for $H = 2, 3, 4, 5, 6$ values, using L_2 norm, and Hamming distance, respectively.

Results are consistent with [18], where values $K = 10$ and $H = 6$ (see Section 3) were proposed as good parameters to build the vocabulary tree.

Note that methods that use L_2 norm exhibit results superior to the ones using Hamming distance. Methods using L_2 norm have mAP peaks of 0.92 while the ones that use Hamming distance obtain at most 0.88. The descriptors methods KAZE [1] and AKAZE [2], in some cases resulted slightly better than the classic ones. One important advantage of using KAZE and AKAZE is that they are open source, while SIFT [16] and SURF [6] are patented and non-free.

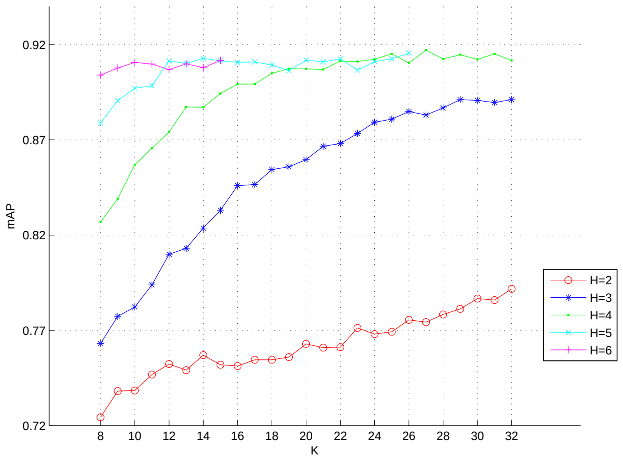
4.4 Performance vs. Training Size

Figure 12 shows the result of varying the number of images used to train the vocabulary. For each feature description technique, K and H parameters were chosen according to the best results of the K-H exploration experiment in Section 4.3 and the descriptors were extracted using SIFT.

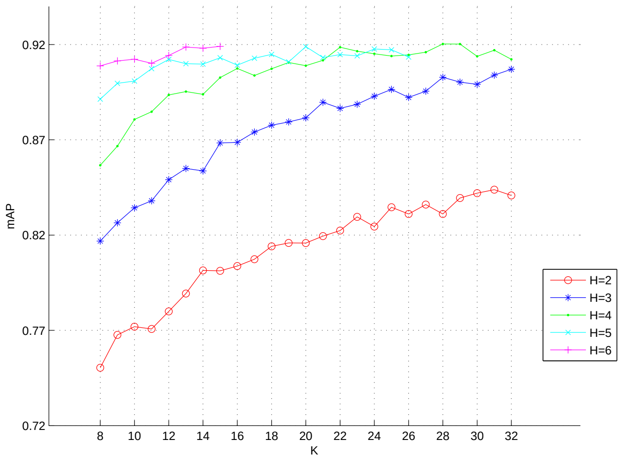
The UKBench dataset was used both for training and testing. Training was performed varying incrementally the number of images from 1000 to 10000. Metrics in runtime were computed using always the entire dataset (10200 images). Results show that a bigger vocabulary gives a better performance, conclusion that is consistent with Niéster work.

4.5 Performance vs. Dimensionality Reduction

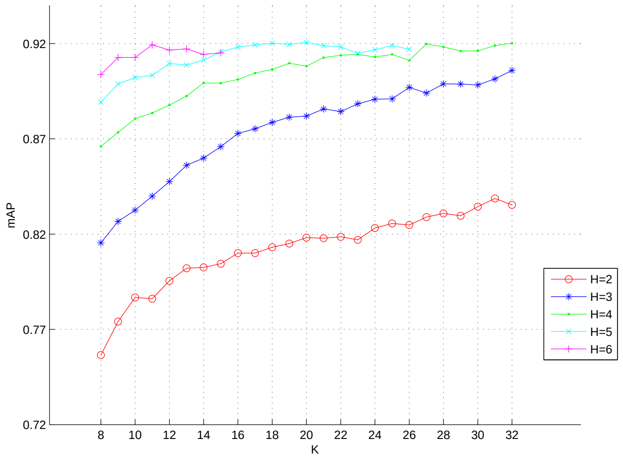
A good improvement can be achieved by using Principal Component Analysis (PCA) to reduce the dimensionality of the features. Dimensionality reduction is done before generating the vocabulary, as well as in the runtime phase. Figure 13 shows the results of applying PCA on the training features of the first 1000 images of the UKBench dataset. Figure 14 shows the results of the same experiment, but applying PCA to reduce dimensionality of SIFT descriptors from 128 to 2 dimensions. The



(a) SIFT.

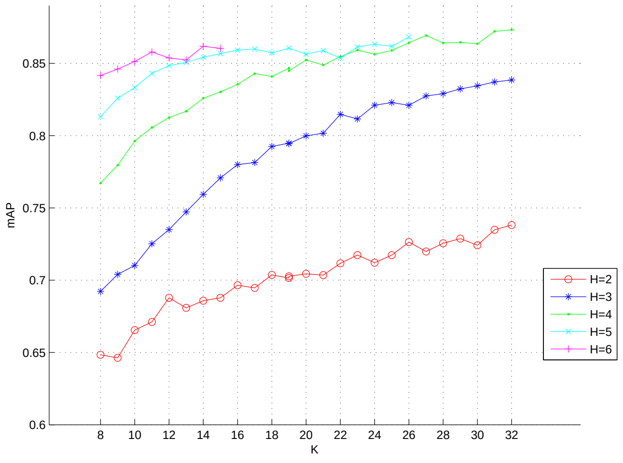


(b) KAZE.

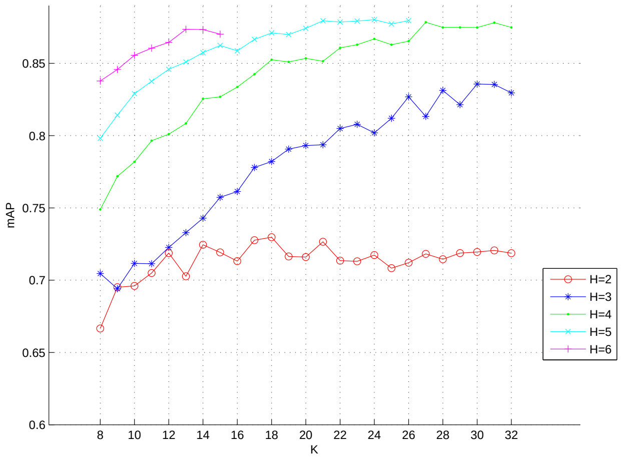


(c) SURF.

Figure 10: mAP behavior varying K between 8 and 32 for $H = 2, 3, 4, 5, 6$ values, using descriptors (a) SIFT, (b) KAZE with parameters $extended = true$ and $threshold = 0.0001$, and (c) SURF.



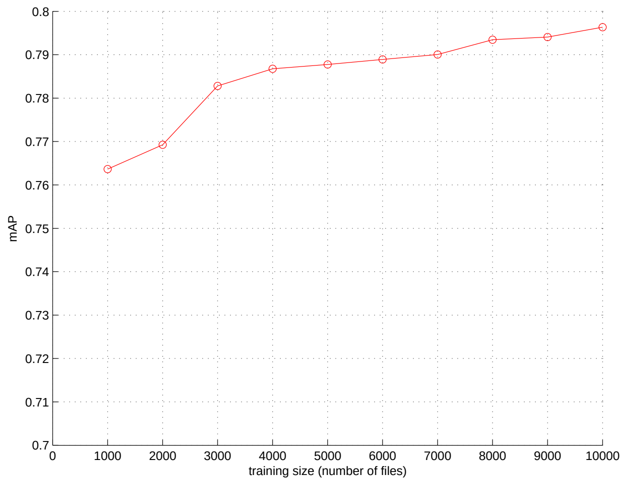
(a) ORB.



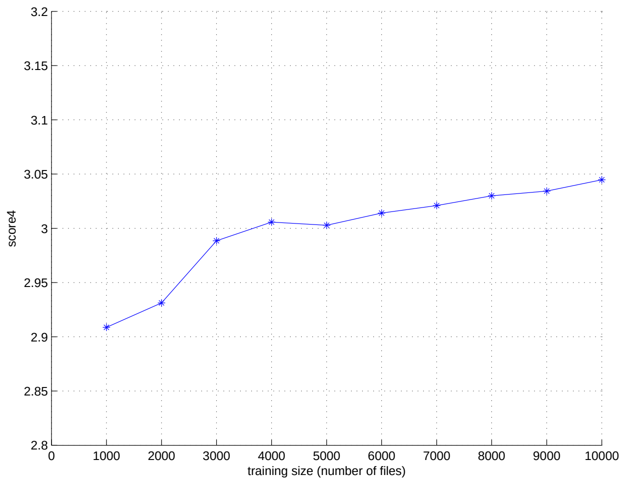
(b) AKAZE.

Figure 11: mAP behavior varying K between 8 and 32 for $H = 2, 3, 4, 5, 6$ values, using Hamming distance and L_1-norm for scoring (a) ORB (b) AKAZE with parameter $threshold = 0.0001$.

results show that if the dimensionality of the features is reduced with PCA, the performance is not deteriorated, and can even grow. The reduction of dimensionality has no impact on the query execution time and could even slightly improve it. However, a greater advantage can be achieved



(a) mAP vs training size.



(b) $Metric_4$ vs training size.

Figure 12: Performance vs. training size measured with (a) mAP, and (b) $Metric_4$. The vocabulary was built varying the number of input training files from 1000 to 10000, and parameters $K = 27$, $H = 4$. The SIFT method was used to extract descriptors.

by reducing the required storage space and execution time in the training phase. For example SIFT experiments show that half of the descriptor dimensions could be discarded without making a worse performance, and requiring half space to store features. One of the possible causes of this small performance increase could be related to the fact that less dimensions avoid the effects of the curse of dimensionality, and K-clustering algorithms can deal better with less dimensions.

4.6 Performance vs. Database Size

Figure 15 shows the result of varying N given a fixed vocabulary. The vocabulary was built using the entire UKBench dataset. First, UKBench images were indexed (10200 images), and then images from MirFlickr1M were added progressively up to 1 million images. Descriptors were extracted using SIFT, limiting their amount to 500 per image, and dimensions were reduced to 64 using PCA. The UKBench dataset was used as query set for computing mAP. This experiment shows how this method scales well with the dataset size.

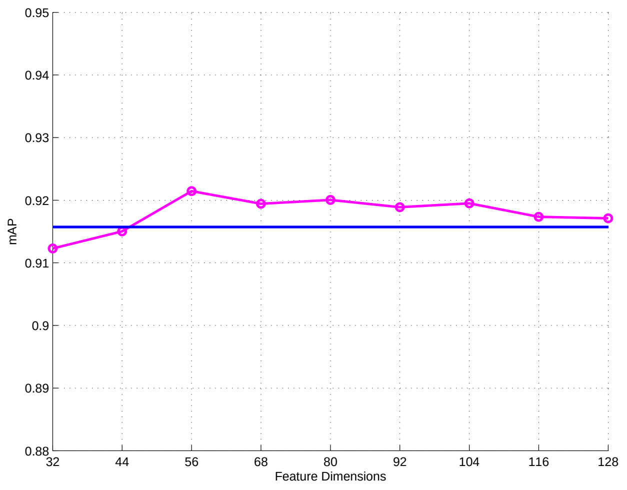
5 Examples

In this section, several example applications that make use of the vocabulary tree to search images are presented.

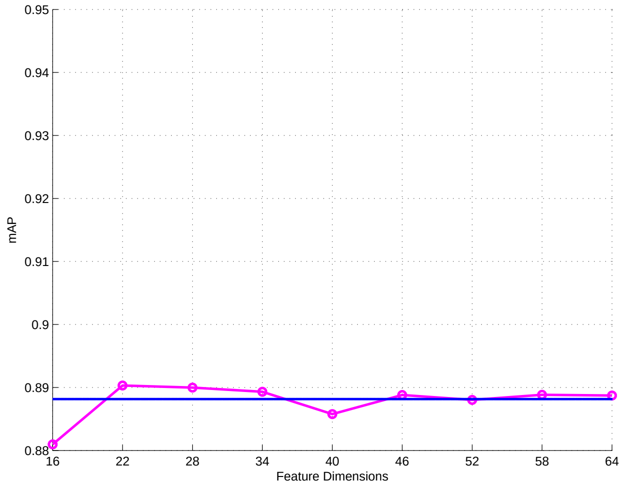
- UKBench dataset: Figures 16 and 17.
  Object recognition examples using the dataset for Section 4.

- Object/Scene search in videos: Figure 18.
  A video can be thought as a sequence of images, so the vocabulary tree can be employed to recognize objects and scenes within the frames of a movie.

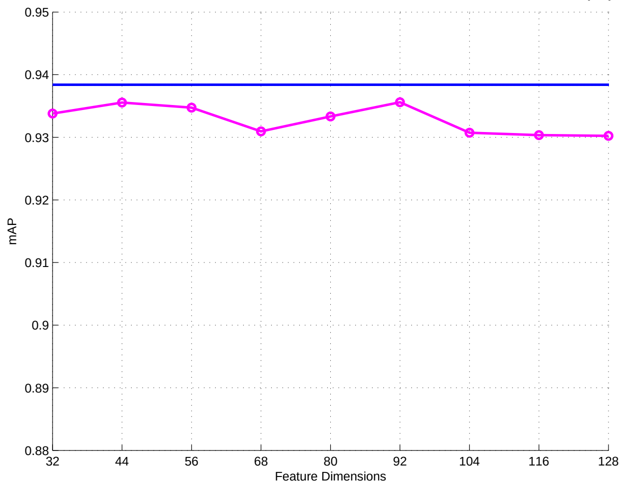
- Bank notes: Figures 19 and 20.
  Image classification example for recognizing Argentinean bank notes.



(a) SIFT.



(b) KAZE.



(c) KAZE extended.

Figure 13: mAP as a function of feature dimensions for the UKBench dataset. Blue line represents mAP without applying PCA for the same input. Vocabulary tree parameters $K = 27$, $H = 4$. (a) SIFT descriptors, (b) KAZE descriptors, (c) KAZE descriptors with parameters $extended = true$ and $threshold = 0.0001$

- Wine labels: Figure 21.
  Example application to recognize wine labels. Once a wine label is detected, a post-processing algorithm can be applied in order to detect the vintage and the variety of grape.

6 Conclusions

An implementation of an image search system using a vocabulary tree was presented, as well as a detailed explanation of the involved algorithms. Source code in $C++$ is provided and its functionality can be executed with the online IPOL interactive demo. Even though the vocabulary creation depends strongly on the amount of memory available, an external clustering implementation is provided that allows using training sizes bigger than RAM availability.

This implementation was evaluated using different feature extraction methods, and varying vocabulary construction parameters, in order to measure how these elements affect retrieval performance. The experiments showed that:

- Larger values of branch factor K and tree maximum height H increase the number of tree nodes,

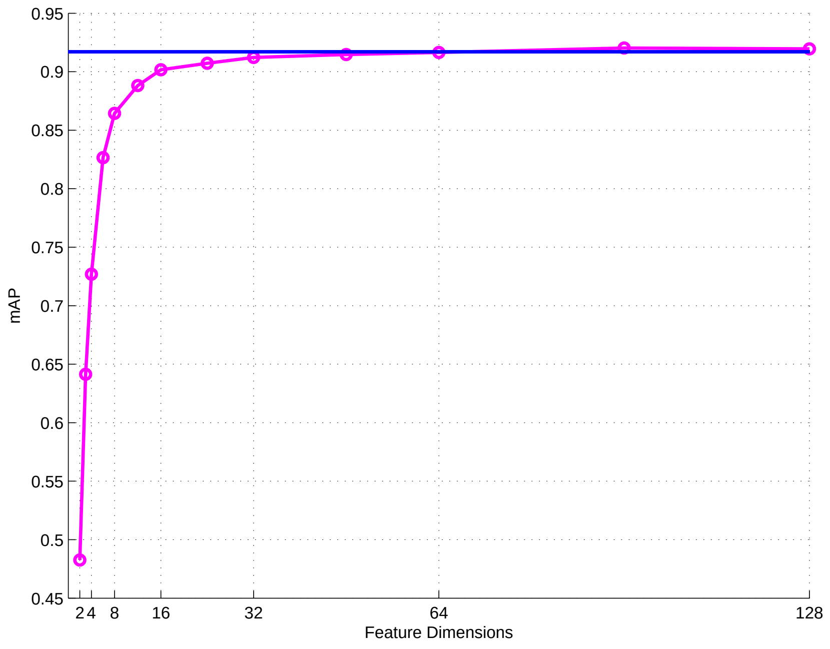


Figure 14: mAP as a function of feature dimensions. SIFT descriptors were reduced from 128 to 2 dimensions using PCA. mAP was computed using the first 1000 images of the UKBench dataset.

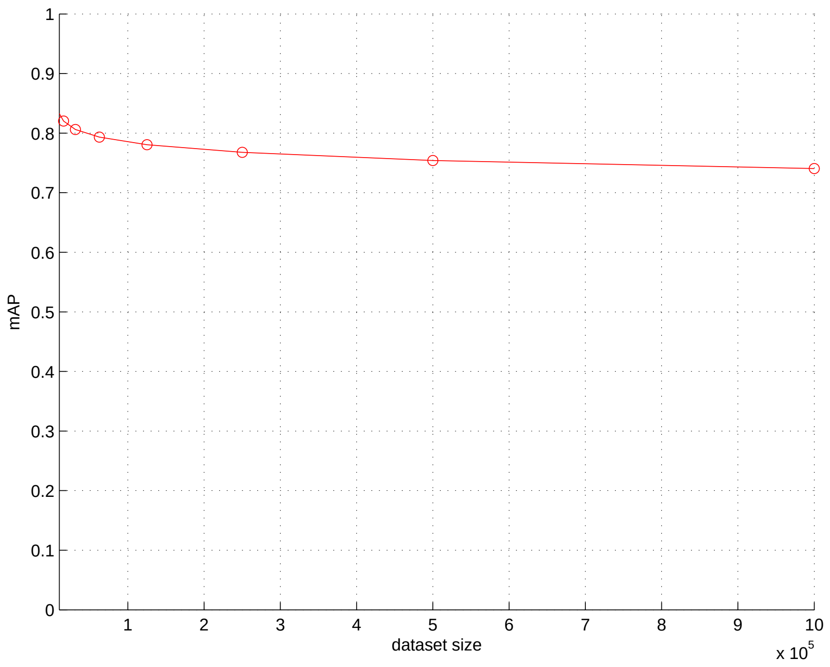


Figure 15: mAP as a function of dataset size ,up to 1 million of image files using SIFT and PCA with 64 dims.

allowing to work with more visual words that generate a richer vocabulary, and providing better performance.

- More training samples increase the retrieval performance, both increasing the number of images, and the number of features per image.

- The method scales well with the size of the input dataset.

Standard hardware gives a constrained amount of memory, and hierarchical clustering is a suitable approach to address this problem. These constrained memory requirements could be exploited in

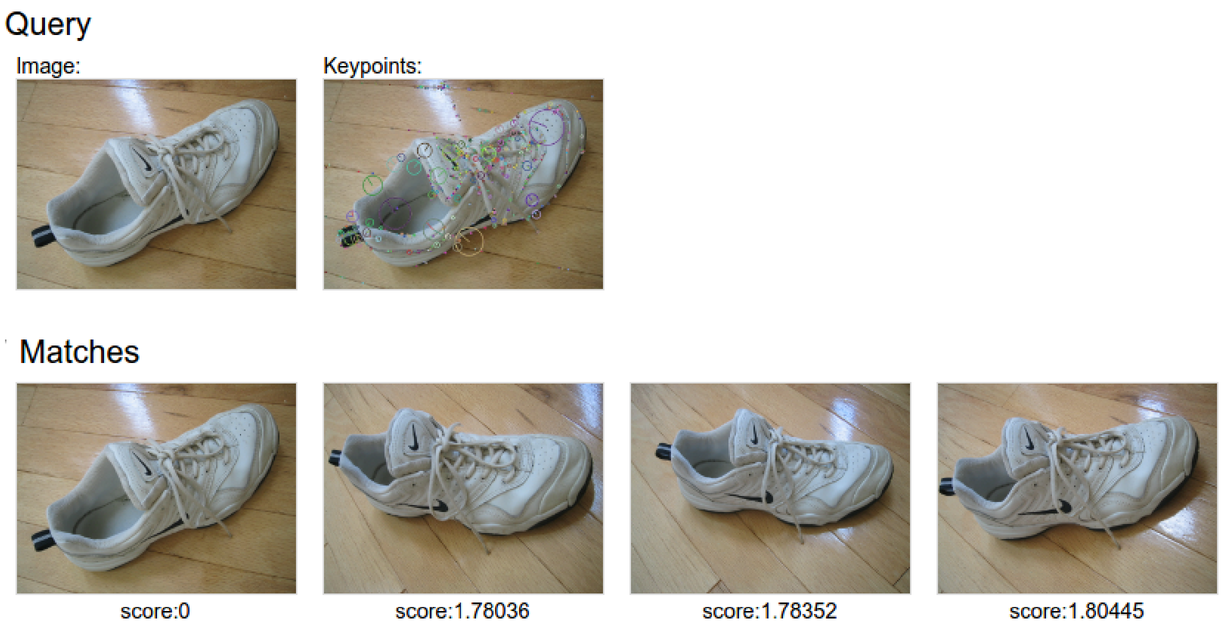


Figure 16: UKBench dataset example 1. Query was done using a query image within the dataset. First row: image query and its keypoints. Second row: ranked list sorted by similarity and their score.

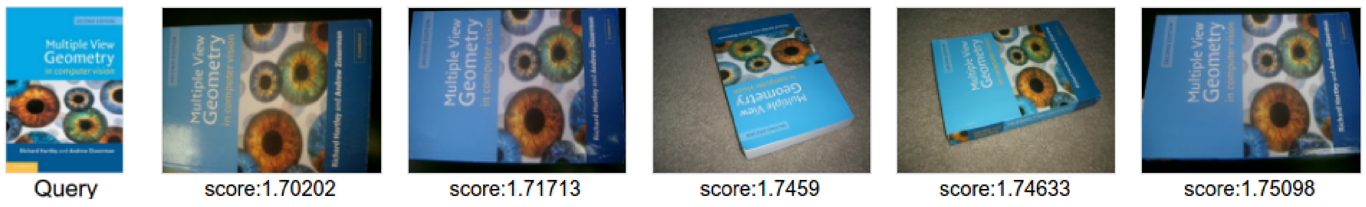


Figure 17: UKBench dataset example 2. Query was done using an image downloaded from internet showing the ability of the vocabulary tree for object recognition.

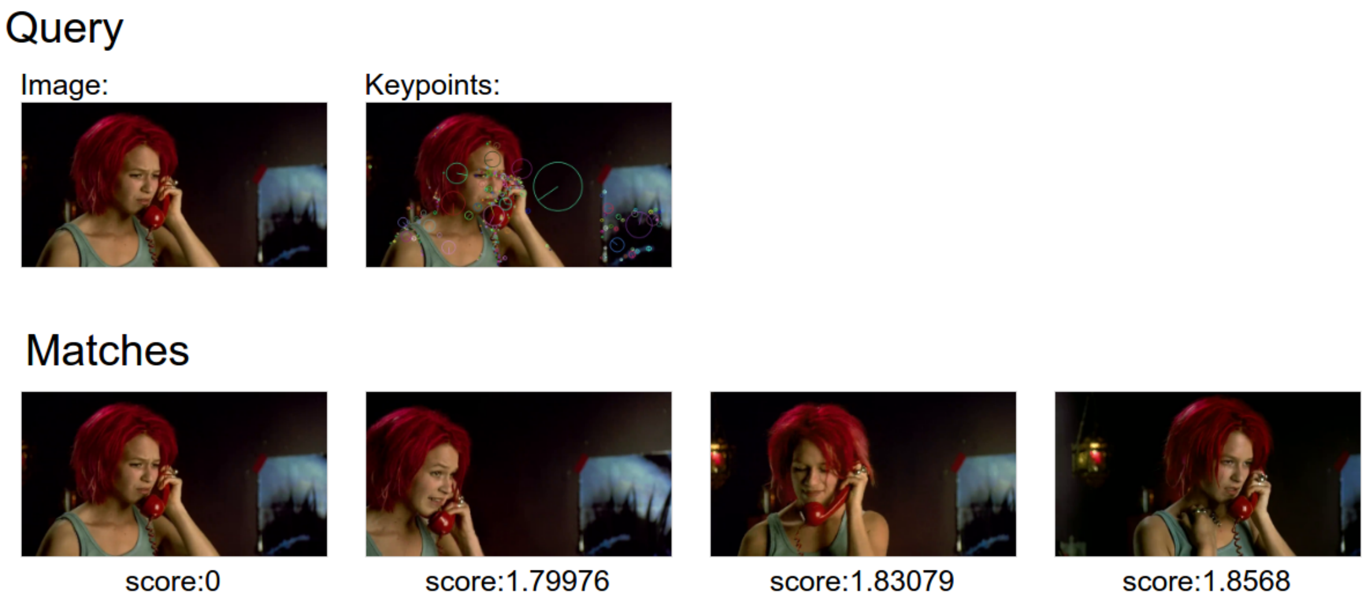


Figure 18: Scene retrieval from the movie Run Lola Run. Almost all the frames from the movie were used to train the vocabulary tree. Query image was done using a frame from the same movie. First row: image query and its keypoints. Second row: ranked list sorted by similarity and their score.

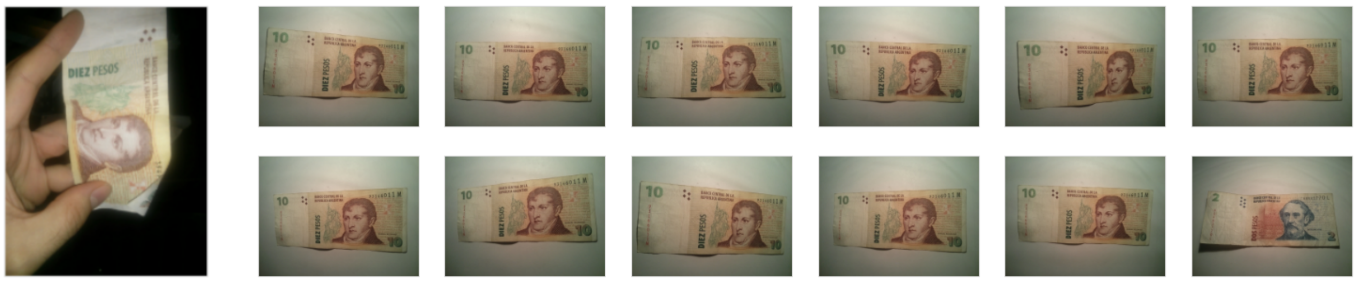


Figure 19: Bank note recognition example. 11 samples of each denomination were used to train. A new image (showed in the first column) is queried, obtaining the image list sorted by similarity on the right. There are enough elements on the top of the list that vote for the same class. Therefore, there is a majority consensus and the result: "10 pesos".

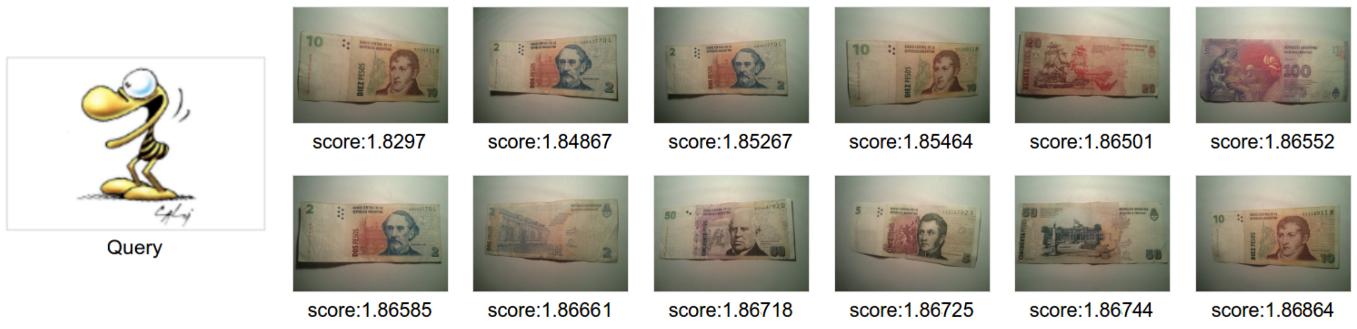


Figure 20: Bank note recognition, rejection example. A picture of "Clemente" is given as query. There is no consensus on the same class and the response is "not a bank note".

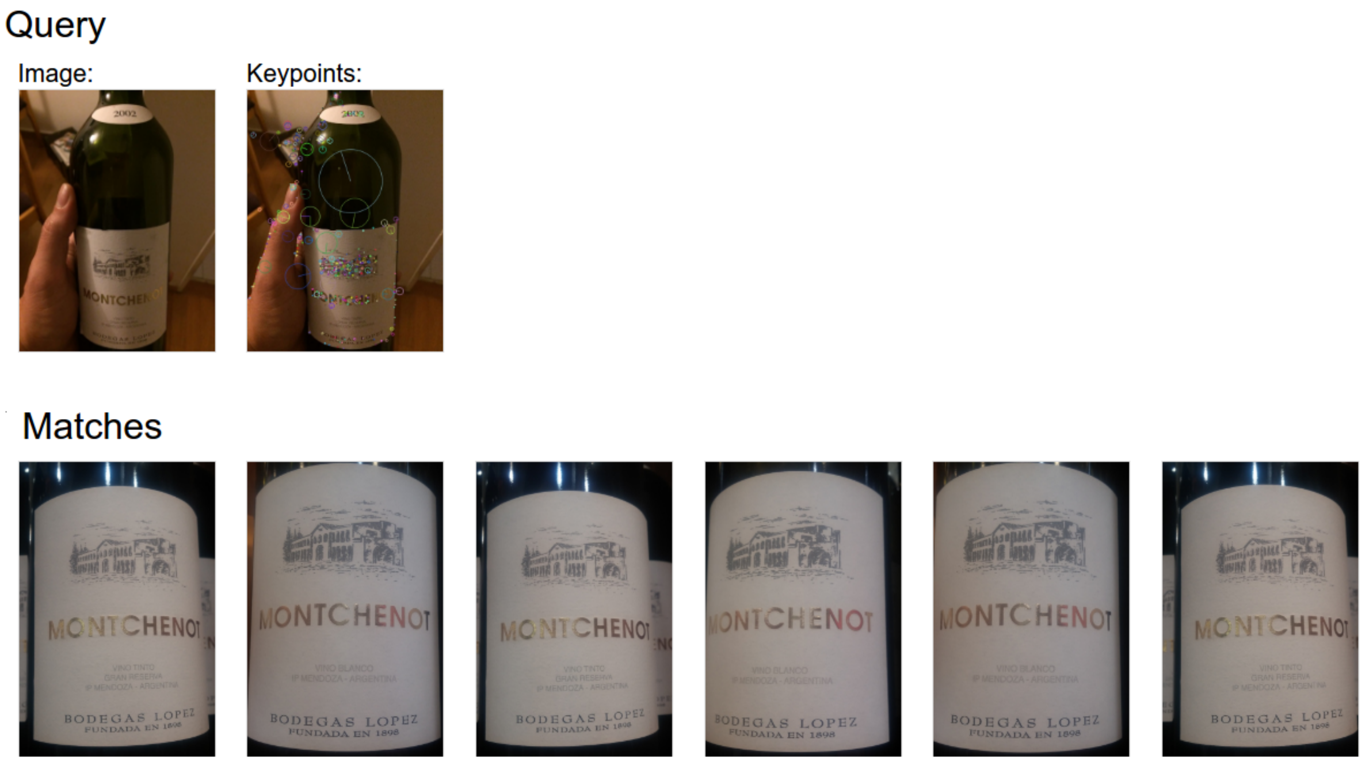


Figure 21: Wine label recognition example. The training set consist of 1053 images with an average of 6 images per wine label. First row: image query and its keypoints. Second row: ranked list sorted by similarity.

several applications on reduced hardware devices such as smart phones or mobile robots, among others.

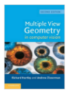
As contributions to this method, it can be mentioned: First, dimensionality reduction, using techniques such as PCA at least does not worsen retrieval performance, but allows to work with less memory storage. Second, the use of the new feature extraction methods like *KAZE* and *AKAZE* performs as well as the classic ones and provide an open source alternative to the non-free ones.

Many parts of this implementation could be improved, or further developed. As future extensions can be mentioned:

- Adding a re-ranking stage for results. This could be achieved using geometric consistency as proposed in [20].
- Improving descriptiveness to features used for building the vocabulary, and the use of visual phrases as proposed in [26]
- Embedding extra information to the features to improve obtained results as proposed in [9], and [10]
- Exploring different alternatives for the clustering method. Actually K-means and K-majority were experimented. While there are many other clustering techniques, many of them do not scale well with the input size. It would be interesting to explore different clustering variants. Also, K-clustering forces a fixed branch factor, but it would be interesting to experiment with an adaptive branch factor, that perhaps would give better description of the data.

Image Credits

All images by the authors (license CC-BY-SA) except images from datasets UKBench and MirFlickr1M, and the following:

- Figure 18: keyframes extracted from the movie 'Run Lola Run' by Tom Tykwer.
- 'Clemente' cartoon character by Carlos Loiseau.
-  Book cover of 'Multiple view geometry in computer vision' by R. Hartley and A. Zisserman.

References

[1] P.F. Alcantarilla, A. Bartoli, and A.J. Davison. Kaze features. In *European Conference on Computer Vision (ECCV)*, pages 214–227. Springer, 2012. https://doi.org/10.1007/978-3-642-33783-3_16.

[2] P.F. Alcantarilla, J. Nuevo, and A. Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, 2011. https://doi.org/10.5244/c.27.13.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006. https://doi.org/10.1007/11744023_32.

[4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European Conference on Computer Vision (ECCV)*, pages 778–792. Springer, 2010. https://doi.org/10.1007/978-3-642-15561-1_56.

[5] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. https://doi.org/10.1016/b978-0-08-051581-6.50070-2.

[6] R. Funayama, H. Yanagihara, L. Van Gool, T. Tuytelaars, and H. Bay. Robust interest point detector and descriptor, 2009. US Patent App. 12/298,879, https://www.google.com/patents/US20090238460.

[7] C. Grana, D. Borghesani, M. Manfredi, and R. Cucchiara. A fast approach for integrating orb descriptors in the bag of words model. In *IS&T/SPIE Electronic Imaging*, pages 866709–866709. International Society for Optics and Photonics, 2013. https://doi.org/10.1117/12.2008460.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, volume 15, pages 10–5244. Manchester, UK, 1988. https://doi.org/10.5244/c.2.23.

[9] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *European Conference on Computer Vision (ECCV)*, pages 304–317. Springer, 2008. https://doi.org/10.1007/978-3-540-88682-2_24.

[10] H. Jégou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):316–336, 2010. https://doi.org/10.1007/s11263-009-0285-2.

[11] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3304–3311. IEEE, 2010. https://doi.org/10.1109/cvpr.2010.5540039.

[12] H. Jégou, H. Harzallah, and C. Schmid. A contextual dissimilarity measure for accurate and efficient image search. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007. https://doi.org/10.1109/cvpr.2007.382970.

[13] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 775–781. IEEE, 2005. https://doi.org/10.1109/CVPR.2005.288.

[14] S. Leutenegger, M. Chli, and R.Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *International Conference on Computer Vision (ICCV)*, pages 2548–2555. IEEE, 2011. https://doi.org/10.1109/iccv.2011.6126542.

[15] D.G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. https://doi.org/10.1023/b:visi.0000029664.99615.94.

[16] D.G. Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image, 2004. US Patent 6,711,293, https://www.google.com/patents/US6711293.

[17] M. Muja and D.G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 2, 2009.

[18] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society. https://doi.org/10.1109/CVPR.2006.264.

[19] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001. https://doi.org/10.1023/A:1011139631724.

[20] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007. https://doi.org/10.1109/cvpr.2007.383172.

[21] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE, 2011. https://doi.org/10.1109/iccv.2011.6126544.

[22] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013. https://doi.org/10.1007/s11263-013-0636-x.

[23] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *International Conference on Computer Vision (ICCV)*, pages 1470–1477. IEEE, 2003. https://doi.org/10.1109/iccv.2003.1238663.

[24] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008. https://doi.org/10.1109/cvpr.2008.4587633.

[25] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1753–1760. Curran Associates, Inc., 2009.

[26] S. Zhang, Q. Huang, G. Hua, S. Jiang, W. Gao, and Q. Tian. Building contextual visual vocabulary for large-scale image applications. In *ACM International Conference on Multimedia*, pages 501–510. ACM, 2010. https://doi.org/10.1145/1873951.1874018.

[27] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM computing surveys (CSUR)*, 38(2):6, 2006. https://doi.org/10.1145/1132956.1132959.