Packing bag-of-features

Hervé Jégou
INRIA
herve.jegou@inria.fr

Matthijs Douze
INRIA
matthijs.douze@inria.fr

Cordelia Schmid
INRIA
cordelia.schmid@inria.fr

Abstract

One of the main limitations of image search based on bag-of-features is the memory usage per image. Only a few million images can be handled on a single machine in reasonable response time. In this paper, we first evaluate how the memory usage is reduced by using lossless index compression. We then propose an approximate representation of bag-of-features obtained by projecting the corresponding histogram onto a set of pre-defined sparse projection functions, producing several image descriptors. Coupled with a proper indexing structure, an image is represented by a few hundred bytes. A distance expectation criterion is then used to rank the images. Our method is at least one order of magnitude faster than standard bag-of-features while providing excellent search quality.

1. Introduction

Searching for images of the same object or scene in a large number of images has recently received increasing attention [9, 4, 11, 5]. The most popular approach today, initially proposed in [12], relies on a bag-of-features (BOF) representation of the image. The idea is to quantize local invariant descriptors, for example obtained by an affine invariant interest point detector [8] and a description with SIFT [6], into a set of visual words. The frequency vector of the visual words then represents the image and an inverted file system is used for efficient comparison of such BOFs. Recent extensions of this method improve the quantization and its speed [9, 11], the post-processing based on a global spatial geometric verification [10], the matching distance of descriptors [4] as well as the efficiency and compactness of the representation [1, 2].

The main advantages of the BOF representation are 1) its compactness, i.e., reduced storage requirements and 2) the rapidity of search due to an inverted file system. In detail, instead of storing a set of 128 dimensional SIFT descriptors for each image, we only have to store one entry for each existing visual word. Furthermore, the inverted file system compares vectors by taking into account only the non-zero entries of the vectors. It is, therefore, very efficient if the vector is sparse. Such a BOF approach allows a single machine to handle several million images. However, it is impossible to scale up to one billion (1000 million) of images, i.e., to web-scale applications.

In this paper, we first show the advantages and limitations of index compression applied to a binary BOF, which obtains excellent search results for large vocabulary sizes. Compressing the inverted file significantly reduces the memory requirements, typically by a factor 4, without modifying the search results. To our knowledge, this well established method in the text retrieval community has never been considered for large scale image indexing.

Second, we present an approach scalable to web-scale search, which keeps the data in main memory and indexes it efficiently. We, therefore, need to generate an extremely compressed feature vector for each image, in order to store a large set of images in main memory. Our approach first produces several small descriptors from a BOF representation of the image, referred to as *miniBOF* vectors. Each miniBOF provides partial information about the original BOF and is indexed separately. A fusion strategy based on a distance expectation criterion is then proposed to merge the answers returned for the different miniBOFs. It allows the ranking of the database images even if only a subset of images is returned by each indexing structure. This method is at least one order of magnitude more efficient than standard BOF. Most importantly, the memory usage per image is typically a few hundred bytes, which is two order of magnitude lower than for a standard BOF.

Most similar to our work is the min-Hash method of Chum et al. [1, 2] which stores a constant amount of data per image and has a reduced search complexity. The technique is very appropriate for near duplicate detection where a small number of "sketches" (combinations of visual words) is sufficient to represent an image. However, in the presence of complex viewing changes, as for example in the University of Kentucky dataset [2], the number of bytes stored is similar to a classic BOF. Also related to our approach is the work of Weiss et al. [13] which reduces the global GIST descriptor to a small binary code with spectral

hashing. However, the subsequent search is exhaustive and the GIST descriptor not invariant to rotation, cropping and strong changes in viewpoint. Furthermore, their approach is only applied to evaluate the similarity of patterns with the same layout.

The paper is organized as follows. Section 2 presents the bag-of-features implementation used in this paper as well as the datasets and the performance measures. A preliminary study analyzes binary BOFs as well as inverted file compression in Section 3. Our indexing method based on miniBOFs is, then, presented in Section 4 and supported by the experiments in Section 5.

2. Background: bag-of-features and datasets

In the following, we briefly present the BOF representation and the evaluation datasets used in this paper.

2.1. Bag-of-features representation

Producing a BOF representation is performed by first extracting local image descriptors. Interest regions are extracted with the Hessian-Affine detector [8] and described with the SIFT descriptor [6]. Clustering of the descriptors is then performed with a k-means quantizer, where the number of clusters k is a parameter of the approach. Note that an independent dataset is used in our experiments to compute the clusters. The cluster centroids are in the following referred to as visual words.

Each SIFT descriptor of a given image i is assigned to the closest visual word (Euclidean distance). The histogram of visual word occurrences is weighted using the *tf-idf* weighting scheme of [12] and subsequently normalized with the L2 norm, producing a frequency vector f_i of length k.

2.2. Datasets and evaluation criteria

We perform our experiments on two annotated datasets: the *INRIA Holidays* dataset [4] and the *University of Kentucky recognition benchmark* [9]. For both datasets we have used the descriptor extraction procedure of [4] [1]. To evaluate large scale image search we use a distractor dataset of one million images downloaded from Flickr, *Flickr1M*. A second dataset of one million images, *Flickr1M**, is used to learn the parameters of our approach. These datasets have the following characteristics:

dataset	#images	#queries
Holidays	1,491	500
Kentucky	10,200	10,200
Flickr1M	1,000,000	0
Flickr1M*	1,000,000	0

We use the evaluation measures proposed by the authors, i.e., the mean the average precision (mAP) [10] for Holidays: for each query image we obtain a precision/recall curve, compute its average precision and then take the mean value over the set of queries. We also evaluate the recall in the top N returned images, denoted by *recall@N*, which measures how the system filters the dataset images. On Kentucky the performance measure is the number of relevant images in the top 4 images, i.e., this score is equal to 4 $\times$ recall@4.

Finally, as the goal of this paper is to go beyond the current limitations on the number of database images, we will evaluate the memory usage, the amount of memory to be read when performing a query, and the average number of "hits", i.e., the number of documents returned by the system. For an inverted file system, this quantity is the average number of database BOF vectors that share at least one non-zero position with the query BOF vector.

3. Preliminary discussion

We want an image representation that is invariant to a large class of transformations (rotation, cropping, change in viewpoint, etc) and meets the following requirements: 1) it represents an image with a low number of bytes and 2) it is associated with an efficient indexing strategy that limits the amount of memory to be read at query time. The second point is especially important when the data is stored on a low efficiency memory device, such as a hard drive. It is also a good measure for the search complexity when the data is stored in main memory.

In the following, we review the current methods that, from our point of view, best address these objectives, and discuss their limitations. In particular, we evaluate index compression [15, 16], an approach that has never been considered in large-scale image search.

3.1. Binary BOF

A straightforward way of compacting a BOF vector is to use a binary BOF representation, i.e., to discard the information about the exact number of occurrences of a given visual word in the image. In that case the BOF vector components only indicates the presence or absence of a particular visual word in the image. This choice was first proposed in [12], and shown to be slightly inferior compared to the BOF vector for a vocabulary size of about $k = 10000$ visual words. However, to our knowledge this approach has not been evaluated for smaller (< 2000) or larger (> 20000) vocabularies. Fig. 1 shows a comparison performed on the INRIA Holidays dataset for a varying vocabulary size. One can see that the binary BOF representation is weak for small vocabularies, but obtains slightly better results than the full BOF for large ones (> 10000).

The number of bits used to represented a binary vector strongly depends on the encoding method. The naive

[1] This procedure is described at http://lear.inrialpes.fr/people/jegou/data.php

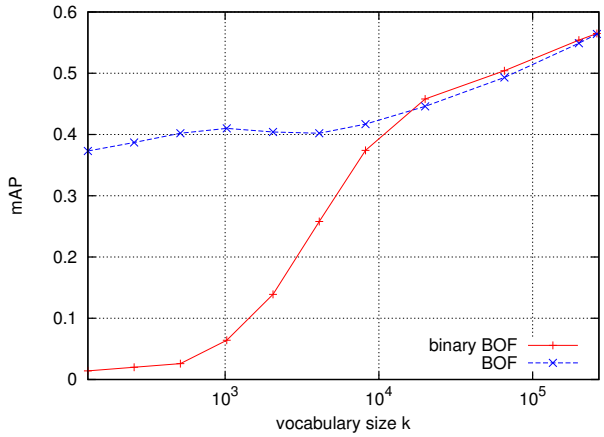

Figure 1. Search quality: BOF *vs* binary BOF

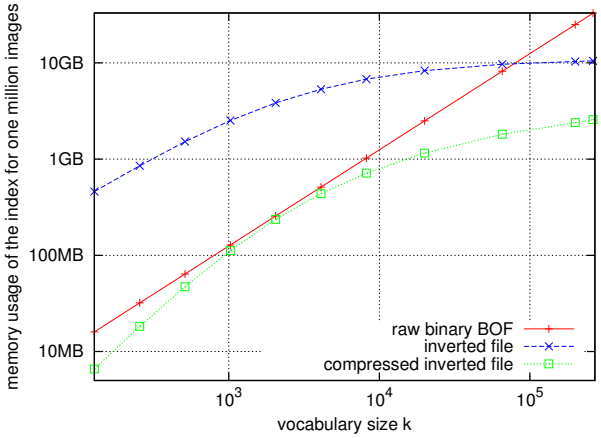

Figure 2. Binary BOF vectors: memory usage of different indexing structures for one million images.

method, denoted by *raw binary BOF*, consists in a sequential coding using 1 bit per component. The memory usage is, then, $\lceil k/8 \rceil$ bytes per image. As the binary BOF representation is good for large vocabularies only, the memory usage per image would be typically 10 kB per image, see Fig. 2.

A better strategy is to take into account the sparsity of large binary BOF vectors: most of their components are zeros. Sparse vectors are usually represented by tuples of the form (non-zero position index, value), which in the binary case consists in coding only the positions of non-zeros components. Storing vectors with an inverted file [16] is a better strategy, as it allows the efficient computation of the distances between a given query vector and a set of sparse vectors representing the images in the dataset.

3.2. Compressed inverted file

Another possibility is to use index compression, an established method in text retrieval [16]. To our knowledge, it has never been considered in the image search literature. The motivation behind index compression is to exploit the distribution of the visual words components. Here we only consider the binary case, i.e., we exploit the probability mass function of binary BOF vectors.

It is well know in information theory that data can theoretically be compressed close to the vector entropy. This is, however, difficult to measure, as it requires to estimate the probability mass function of a space of cardinality 2^k. An upper bound, which is reached when the components are independent, is obtained by summing the individual marginal entropies of the binary components. This compression performance can almost be reached by coupling, within the inverted file, run-length encoding with arithmetic coding [14]. Several index compression methods offering different trade-offs have been considered in the text retrieval literature. Some of them compress the index close to the entropy bound, see [15] for a recent survey.

Fig. 2 gives the storage requirements, computed from the measured binary component entropies, associated with a compressed inverted file index storing one million images. These measurements, extrapolated from the INRIA Holidays dataset, clearly show the interest of this approach. Compared with a standard inverted file, about 4 times more images can be indexed using the same amount of memory. Moreover, the amount of memory to be read is proportionally reduced at query time. This may compensate the decoding cost of the decompression algorithm.

As a final remark, note that for a vocabulary size of $k = 2000$, the memory usage of the compressed inverted file is equivalent to the one obtained by directly coding the raw binary vectors. This corresponds to the maximum entropy case: the probability that a given visual word is present or not in the image is close to 0.5.

3.3. Discussion

Index compression coupled with a binary BOF representation is a promising method, as for large vocabularies it guarantees an excellent search quality, see Fig. 1. It also provides a reasonable memory usage of 1-2 kB per image (see Fig. 2), which is five times less than a standard inverted file for binary vectors, and even more compared to an inverted file storing the full component values.

This memory requirement is of the same order as the memory used by the min-Hash method in a near-duplicate detection setup, where 768 bytes per image are used for 64 "sketches" [1]. This number of sketches is appropriate for near-duplicate detection only. In the case of object recognition a larger number of sketches is necessary. To obtain reasonable results on the Kentucky benchmark, at least 500 sketches are necessary [2], which corresponds to about 6 kB

per image. For near-duplicate detection, min-Hash may be a better choice, as the number of document "hits", i.e., the number of documents which receive a non-zero score, is significantly lower than for a binary BOF representation. Min-Hash typically returns 5% of the total number of documents [2], a large improvement over an inverted file, binary or not, compressed or not.

The method proposed in the next section goes beyond index compression and min-Hash: (1) it offers accuracy comparable to the BOF with a few hundred bytes per image; (2) it reads a limited amount of memory at query time; and (3) it returns a reduced number of documents.

4. MiniBOFs

The central problem of efficiently indexing a BOF image representation is that, to our knowledge, there exists no efficient approximate nearest neighbor search algorithm for sparse vectors. Hence, using a state-of-the-art indexing algorithm such as locality-sensitive hashing [3] is less efficient than using an inverted file structure, which computes the exact distances by visiting only the non-zeros positions.

The approach proposed in this section is a way of retrieving approximate nearest BOF vectors. Fig. 3 gives an overview of our approach. A query is performed by 1) producing several descriptors for a single BOF vector, by 2) indexing them in separate structures and by 3) fusing the distances returned by these structures using a distance expectation criterion. These steps are explained in the following subsections.

4.1. Projection of a BOF: vocabulary aggregators

The first step of our approach produces a set of image descriptors given a BOF. Each descriptor represents a visual vocabulary for a coarse partition of the feature space, hence providing an independent representation of the image.

To obtain these descriptors, we introduce a set of sparse projection matrices $\mathcal{A} = \{A_1, \ldots, A_m\}$ of sizes $d \times k$, where d is the dimension of the output descriptor and k the dimension of the initial BOF. A projection matrix A_j is called an *aggregator*, as it aggregates the vocabulary by grouping several components of the input BOF vector into a single one. For instance, for $k = 12$ and $d = 3$, we can define the first aggregator as

$$A_1 = \left[\underbrace{\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}}_{k} \right] \Bigg\} d \quad (1)$$

For each projection vector (a matrix row), the number of non-zero components is $n_z = k/d$, and is chosen such that the output vector is of reasonable dimension. We typically set $n_z = 8$ for $k = 1000$, resulting in descriptors of dimension $d = 125$. Note that the choice of A_1 in (1) is performed without loss of generality because there is no particular order between the bag-of-features components.

The other aggregators are defined by shuffling the input BOF vector components using random permutations. For $k = 12$ and $d = 3$, the random permutation (11, 2, 12, 8, 9, 4, 10, 1, 7, 5, 6, 3), results in the aggregation matrix

$$A_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2)$$

Multiplying aggregator j by the BOF frequency vector f_i produces a vector

$$\omega_{i,j} = A_j \times f_i \quad (3)$$

of dimension d, which can be seen as a BOF vector for the quantizer resulting from the aggregation of the Voronoi cells in the original k-means codebook. Such a vector is called a *miniBOF*. An image i is, then, described by the m miniBOFs $\omega_{i,j}$, $1 \leq j \leq m$, resulting from the projections of the input BOF by the aggregators $A_1, \ldots, A_m$.

4.2. Indexing structure

We build upon the Hamming Embedding (HE) approach introduced in [4] to index local descriptors. The first step is a quantization into visual words. Within each quantization cell the Hamming Embedding refines the search. The two steps, i.e., quantization and binary signature generation, produce a compact representation of the miniBOFs. Note that we create a separate indexing structure for each miniBOF type, i.e., for each aggregator A_j, see Fig. 3.

Quantization. The miniBOF $\omega_{i,j}$ is quantized using the quantizer q_j associated with the j^{th} aggregator, producing a quantization index $c_{i,j} = q_j(\omega_{i,j}) \in \{1, .., k'\}$, where k' is the number of codebook entries of the indexing structure.

The set of k-means codebooks $q_j(.)$, $1 \leq j \leq m$, is learned off-line using a large number of miniBOF vectors, here extracted from the Flickr1M* dataset. The dictionary size k' associated with the minBOFs is not related to the one associated with the initial SIFT descriptors, hence we may choose $k \neq k'$. We typically set $k' = 20000$.

Binary signature generation. The objective of this step is to compute a binary signature $b_{i,j}$ of length d that refines the localization of the miniBOF within the cell. The binary signature generation is performed using the method of [4]:

1. The miniBOF is projected using a random rotation matrix R, producing d components.

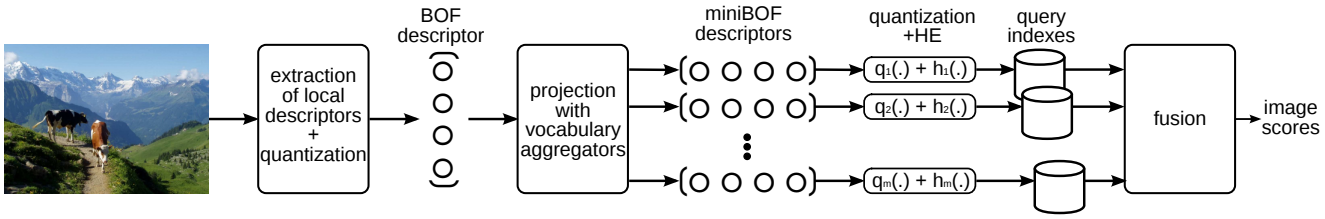

Figure 3. Overview of our image search approach.

2. Each bit of the vector $b_{i,j}$ is obtained by comparing the value projected by R to the median value of the elements having the same quantized index. The median values for all quantizing cells and all projection directions are learned *off-line* on our independent dataset.

At this point, the j^{th} miniBOF associated with image i is represented by the tuple $(c_{i,j}, b_{i,j})$. This tuple is stored in an inverted file [16, 17]. This stucture is an array of k' inverted lists, one per output value of quantizer q_i. The tuple is stored in list no. $c_{i,j}$ of the array, as an entry of the form $(i, b_{i,j})$. The memory used to index a miniBOF is then

- 4 bytes to store the image identifier i ;
- $\lceil d/8 \rceil$ bytes to store the binary vector $b_{i,j}$,

where the ceiling ensures that the entries are byte-aligned in memory. As we use m inverted files in parallel, one per aggregator, the total memory usage per image is $C_i = m\ (4 + \lceil d/8 \rceil)$ bytes.

Querying the j^{th} indexing structure with the tuple $(c_{q,j}, b_{q,j})$ associated with the j^{th} miniBOF vector of the query image amounts to returning all the elements assigned to the inverted list associated with quantization index $c_{q,j}$. By contrast with [4], the role of the binary signature is not to filter elements that are above a given distance threshold, but to provide a distance between the query miniBOF and those in the inverted list. This measure will be used by the fusion algorithm detailed in the next subsection.

Multi-probe strategy. On the query side only, returning a single inverted list associated with the quantized index $c_{i,j}$, as proposed in [4], is not sufficient for miniBOF vectors. This is because the noise on miniBOFs is higher than on SIFT vectors: the vector is severely modified by strong cropping or clutter. To overcome this problem, we adopt a strategy similar to the one proposed for locality-sensitive hashing in [7], namely *multi-probe* querying. In our case, this strategy consists in retrieving not only the inverted list associated with the quantized index $c_{i,j}$, but the set of inverted lists associated with the closest t centroids of the quantizer codebook.

This strategy does not modify the amount of memory needed for the database. However, it increases the number of image hits because t times more inverted lists are visited.

4.3. Fusion: expected distance criterion

The output of the indexing structure is, for each aggregator j, a set of potential relevant images and the Hamming distances[2] of their binary signatures with that of the query miniBOF. Hereafter, we explain how the set of observed distances is used to rank to images.

Expectation based criterion. For the j[th] aggregator, let us denote by $b_{q,j}$ the signature associated with the query image q, and by $b_{i,j}$ the signature of the database image i, respectively. The concatenation $b_q = [b_{q,1}, \ldots, b_{q,m}]$ of the binary signatures over all aggregators is a representation of the query image. Similarly, $b_i = [b_{i,1}, \ldots, b_{i,m}]$ represents the database image i. The distance between these two vectors can be computed as

$$h(b_q, b_i) = \sum_{1 \le j \le m} h(b_{q,j}, b_{i,j}), \tag{4}$$

where $h(x, y)$ represents the Hamming distance between binary vectors x and y. However, for most of the database images and aggregators, the distances $h(b_{q,j}, b_{i,j})$ are unknown because only a small proportion of the indexed miniBOF vectors are stored in the t inverted lists visited for this aggregator. Nevertheless, due to the median partitioning used in the binary signature learning stage (subsection 4.2), we known that the expectation of this distance is $\hat{h}(b_{q,j}, b_{i,j}) = d/2$. Hence, we can compute the expectation of the image distance (4) as

$$\hat{h}(b_q, b_i) = \sum_{1 \le j \le m} \hat{h}(b_{q,j}, b_{i,j}) \tag{5}$$

where

$$\hat{h}(b_{q,j}, b_{i,j}) = \begin{cases} h(b_{q,j}, b_{i,j}) & \text{if } b_{i,j} \text{ observed} \\ d/2 & \text{otherwise} \end{cases} \tag{6}$$

[2]The Hamming distance between two vectors is the number of components that are different.

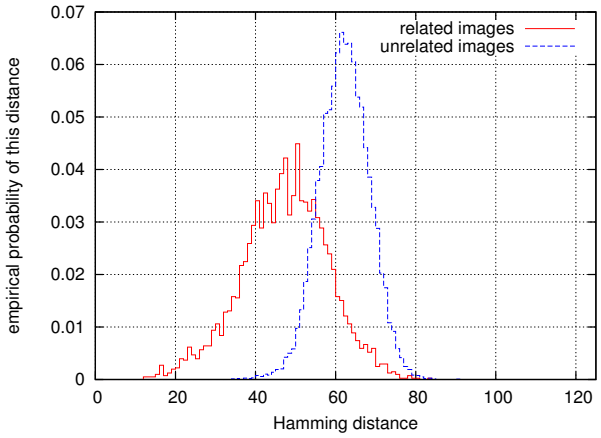

Figure 4. Empirical probability mass functions of the Hamming distance between the binary signatures associated with the same inverted list for 1) matching images and 2) unrelated images. The measures have been performed on the Holidays dataset.

Note that in (4), the quantization index does not appear. It is only used by the indexing structure to select the miniBOF vectors that are likely to be close to the miniBOF query vector. Having filtered most of the miniBOF vectors, we prefer to rely only on the binary signatures to compare miniBOF vectors. Fig. 4 shows the probability mass function of the distances in a particular cell. The probability of having a small distance between vectors of two unrelated images is on average lower than for corresponding images.

Implementation details. For most of the images, we have no Hamming distance measurements in (5), which means that the distance defaults to $m \times d/2$. The score associated with an image is then obtained as a summation over the *observed* distances, as

$$\text{score}_q(i) = \sum_{j/b_{i,j} \text{ is observed}} d/2 - h(b_{q,j}, b_{i,j}), \qquad (7)$$

which is equal to 0 for images having no observed binary signatures, i.e., most of the database images, and equal to $d \times m/2$ if the database image i is the query image itself. This score, which provides the same image ranking as the one of (5), is computed while reading the indexing structure, by accumulating the image scores in a hash table.

As observed in [4], the query speed is improved by a threshold τ on the Hamming distance, as it decreases 1) the number of score updates in the hash table storing the image scores and 2) reduce the number of "documents hits". In the following, we use $\tau = d/2$, which roughly divides by 2 the number of score updates. It amounts to penalizing the images having a large distance in the same quantization cell the same way as those that have not been retrieved at all.

method	k	mAP	memory usage	image hits
BOF	1k	0.414	3,087	1,484
BOF	20k	0.446	10,364	1,471
BOF	200k	0.549	12,886	1,412
binary BOF	20k	0.458	8,291	1,471
binary BOF	200k	0.554	10,309	1,412
compressed binary BOF*	20k	0.458	1,174	1,471
compressed binary BOF*	200k	0.554	1,830	1,412
miniBOF, m=1	1k	0.255	20	19
miniBOF, m=4	1k	0.368	80	48
miniBOF, m=8	1k	0.403	160	68
miniBOF, m=16	1k	0.426	320	93
miniBOF, m=32	1k	0.452	640	120

Table 1. Comparison of the different BOF approaches on the Holidays dataset: search quality (mAP), memory usage (bytes per database image), and average number of image hits per query image. The hits values should be compared to the total number of images (1491). m is the number of miniBOFs; *estimation based on the binary BOF vector entropy.

5. Experiments

We evaluate the miniBOF approach by measuring the performance on the reference datasets Holidays and Kentucky using the evaluation measures introduced in Subsection 2.2. The results on these datasets are presented in Tables 1 and 2. The experiments performed on Holidays + one million images (Flickr1M dataset) are presented in Table 3 and Fig. 5. The approach presented in Section 4 is denoted by *miniBOF* in all these figures and tables.

Unless specified otherwise, we have used the following parameters in all the miniBOF experiments

$k =$	1000	for the SIFT codebook size
$n_z =$	8	for the aggregator parameter
$d = k/n_z =$	125	for the miniBOF dimension
$k' =$	20000	for the miniBOF codebook size
$t =$	100	for the multi-probe strategy

Using a value of n_z between 8 and 12 provides the best accuracy for vocabulary sizes ranging from 1k to 20k. In order to limit the memory usage, which strongly depends on $d = k/n_z$, we use a small vocabulary, i.e., k=1000. This leads to binary signatures of length 125.

BOF *vs* binary and compressed binary BOF. Tables 1 to 3, show the excellent results obtained by the binary BOF, which, for the vocabulary sizes considered, slightly outperforms the BOF, with a reduced memory usage. The number of image hits remains the same, as it depends only on the number of images having at least one shared non-zero position with the query BOF vector. The number of bytes used

method	k	score	memory usage	image hits
BOF	20k	2.92	6,662	9,928
binary BOF	20k	3.02	5,329	9,928
miniBOF, m=1	1k	2.07	20	94
miniBOF, m=8	1k	2.72	160	383
miniBOF, m=16	1k	2.83	320	567
miniBOF, m=64	1k	2.93	1,280	1,078

Table 2. Experiments on the University of Kentucky object recognition benchmark: score (4× recall at 4), number of image hits and corresponding memory usage per image.

per inverted list entry is 4 bytes (for the image identifier) for binary BOF and 5 bytes for BOF, for which we optimize the memory usage by storing the number of occurrences of the visual word, and by performing the *tf-idf* and the L2 normalization on-the-fly.

The improvement due to a compressed binary BOF is shown in Table 1. As the image representation is the same as the binary BOF, the results are identical and obtained with a reduced memory usage, as discussed in Section 3.

MiniBOF. The behavior of miniBOF on the two reference datasets is shown in Tables 1 and 2. One can see that, for a memory usage of at least one order of magnitude lower, the results are quite similar to those obtained by BOF. On Holidays, the mAP is 0.452 for $m = 16$ aggregators, which is comparable to 0.446 obtained by BOF for a vocabulary size of $k = 20$k visual words. Most importantly, this performance is obtained using 320 bytes per image, i.e., 32 times less than for the BOF approach. These results are confirmed by the measurements on Kentucky, where for $m = 16$ we obtain a score of 2.83, to be compared with the score of 2.85 obtained in [2] for 512 "sketches", that require an order of magnitude more memory.

Surprisingly, for $m \geq 16$ the mAP obtained using our approach is better than the BOF computed for the same vocabulary size of 1000 visual words. This may be due to the sub-optimality of the Euclidean distance as a BOF comparison criterion. Indeed, for the Kentucky dataset, it is well known [9] that the Euclidean distance between L2-normalized BOF vectors is poor, and significantly outperformed by the histogram intersection distance. The other interesting measure is the number of images hits, which shows the excellent selectivity of the approach. The system returns about 6% of the images for $m = 8$.

The accuracy of our method is inferior in terms of mAP to the method of [4], which obtained mAP=0.751 in its best setup. However, let us underline that their approach requires 35 kB of memory per image on Holidays, i.e., two orders of magnitude more than MiniBOFs.

method	k	mAP	memory usage	memory scanned	query time
BOF	20k	0.227	7,322	860	22163
BOF	200k	0.315	8,885	148	2827
binary BOF	20k	0.307	5,858	688	14073
binary BOF	200k	0.381	7,108	117	2562
miniBOF, m=1	1k	0.066	20	0.19	71
miniBOF, m=8	1k	0.196	160	1.54	132
miniBOF, m=32	1k	0.244	640	6.14	352

Table 3. Experiments on Holidays + Flickr1M: query time per image (in ms, for one processor core), memory usage (MB) of the indexing structure, memory to be scanned (MB) and search quality of the miniBOF approach compared with BOF.

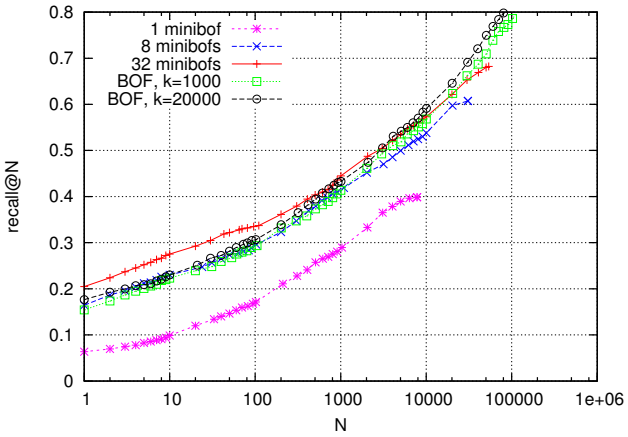

Figure 5. Holidays+Flickr1M: repartition of the true positives (recall@N) for one million images for BOF and miniBOF.

Holidays+Flickr1M: large scale experiments. Fig. 5 gives the recall@N, which reflects how a system is able to filter a large number of images to produce an image shortlist to be treated in a post-ranking stage [6, 10]. Our results are comparable to those of BOF, which is excellent considering the respective memory usages and query times. Our approach requires 160 MB for $m = 8$ and the query is performed in 132ms, to be compared, respectively, with 8 GB and 3 s for BOF. See Table 3 for additional measurements, which confirm the excellent mAP values and query times. As a final note, we want to underline that using $m = 8$, we could index about 350 million images in memory on a present-day server machine (64GB of RAM). We were limited in our experiments by the number of images we could download from Flickr and the disk space (about 250GB is required per million images).

Typical queries from Holidays are shown in Fig. 6. The results returned by the system, although not always correct according to the groundtruth, are visually satisfactory.

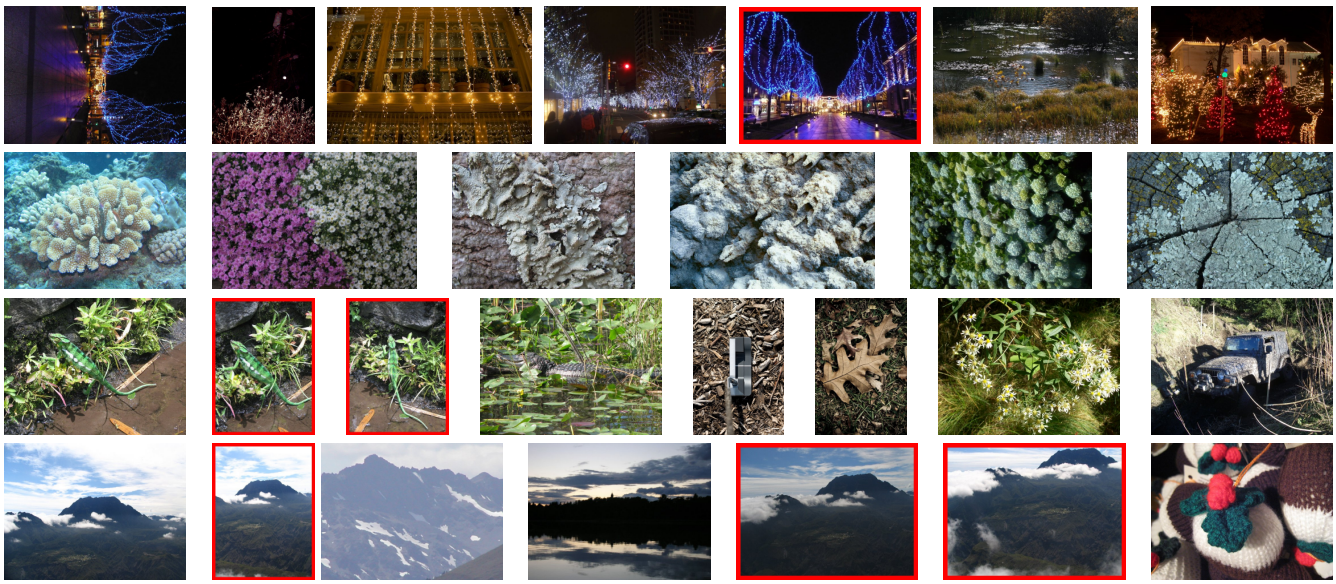

Figure 6. A few sample queries from Holidays +Flickr1M and the first search results with miniBOF. Note that we do not use color. The true positives are framed in red.

6. Conclusion

In this paper we have introduced a way of packing BOFs: miniBOFs. This representation is based on aggregations of several visual vocabulary cells and is extremely compact. By using a set of them we obtain a redundant representation and increase the search speed. An efficient indexing structure based on Hamming Embedding allows for rapid access and an expected distance criterion for the fusion of the scores. Our approach reduces memory usage by more than one order of magnitude. Furthermore, it reduces the quantity of memory scanned (hits) as well as the query time. Experimental results demonstrate an excellent accuracy even for very compact representations.

Acknowledgements

We would like to thank the ANR projects RAFFUT and GAIA, as well as the QUAERO project for their financial support.

References

[1] O. Chum, J. Philbin, M. Isard, and A. Zisserman. Scalable near identical image and shot detection. In *CIVR*, 2007.

[2] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-Hash and tf-idf weighting. In *BMVC*, 2008.

[3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Symposium on Computational Geometry*, pages 253–262, 2004.

[4] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, 2008.

[5] H. Jégou, M. Douze, and C. Schmid. On the burstiness of visual elements. In *CVPR*, June 2009.

[6] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[7] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[8] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *IJCV*, 60(1):63–86, 2004.

[9] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[10] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007.

[11] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008.

[12] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477, 2003.

[13] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[14] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[15] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 387–396, 2008.

[16] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6, 2006.

[17] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, 1998.