Extremely low bit-rate nearest neighbor search using a Set Compression Tree

Relja Arandjelović Andrew Zisserman
Department of Engineering Science, University of Oxford
{relja,az}@robots.ox.ac.uk

Abstract

The goal of this work is a data structure to support approximate nearest neighbor search on very large scale sets of vector descriptors. The criteria we wish to optimize are: (i) that the memory footprint of the representation should be very small (so that it fits into main memory); and (ii) that the approximation of the original vectors should be accurate.

We introduce a novel encoding method, named a Set Compression Tree (SCT), that satisfies these criteria. It is able to compress 1 million descriptors using only a few bits per descriptor, and at high accuracy. The large compression rate is achieved by not compressing on a per-descriptor basis, but instead by compressing the set of descriptors jointly, i.e. *if the set of descriptors is* $\{x_1, x_2, \ldots, x_n\}$ *then the compressed set is* $\mathrm{com}\{x_1, x_2, \ldots, x_n\}$ *rather than* $\{\mathrm{com}(x_1), \mathrm{com}(x_2), \ldots, \mathrm{com}(x_n)\}$*. We describe the encoding, decoding and use for nearest neighbor search, all of which are quite straightforward to implement.*

The method is compared on standard benchmarks (SIFT1M and 80 Million Tiny Images) to a number of state of the art approaches, including Product Quantization, Locality Sensitive Hashing, Spectral Hashing, and Iterative Quantization. In all cases SCT has superior performance. For example, SCT has a lower error using 5 bits than any of the other approaches, even when they use 16 or more bits per descriptor. We also include a comparison of all the above methods on the standard benchmarks.

1. Introduction

Nearest neighbor search is ubiquitous in computer vision with numerous applications across the field. With ever larger data sets generating millions or billions of vector descriptors, two particular problems have become critical: (i) how to keep all the original vectors in main memory, and (ii) how to obtain the nearest neighbors of a query vector as fast as possible. Recently there have been two quite distinct threads of work aimed at addressing problem (i), which both proceed by obtaining a low dimensional representation of the original vectors, such that the distance between two low dimensional representations is a good approximation of the distance between the original vectors. The consequence is that the low dimensional vectors for the entire database then fit in main memory which in turn alleviates problem (ii) as no (expensive) hard disk access is required.

One thread is the product quantization of Jégou and collaborators [10]. This follows on from earlier work on Hamming embedding [9]. Both were aimed at obtaining a more accurate distance between SIFT [12] vectors than that obtained by straight forward k-means vector quantization of the entire vector and representation as visual words [20]. Product quantization divides the vector into sub-vectors, and then vector quantizes each sub-vector (if there are m sub-vectors each quantized independently into k cluster centers, then there are k^m possible code words with $m \cdot log_2(k)$ bits required to represent each vector). It was originally applied to SIFT vector matching, but has since been employed in large scale category retrieval [17] and used with inverted indexes for immediate retrieval [3, 16] (again addressing problem (ii)).

The second thread is the binary string representation of vectors used for retrieval in the 80 Million tiny images series of papers [7, 15, 21, 22, 23]. Here the goal is to represent the original vectors by short binary strings, such that the Hamming distance between the binary representations approximates the distance between the original vectors (or more accurately, that the ranking of Hamming distances equals the ranking of distances between the original vectors).

Current methods achieve reasonable performance only when 16 or 32 or more bits per descriptor are used, but none is capable of functioning at the extremely low bit-rate regime which is certainly needed for huge scale datasets. In this paper we propose a *Set Compression Tree* (SCT) approach to lossy descriptor compression that is capable of accurately representing vectors using only 4 to 7 bits per vector. The key idea in SCT coding is not to store information directly on a per-descriptor basis, but instead to store information jointly across sets of the descriptors in the database. Because the code applies to a *set* of vectors, the number of bits required for *each* vector is far less. The coding is achieved through a simple and very fast scheme where the feature space is partitioned into disjoint cells in a k-d tree like manner using binary splits, such that each descriptor is uniquely associated with one cell and approximated by the centroid of that cell. The SCT coding is explained in section 2, and compared in section 3 to previous compression methods including: Product Quantization of Jégou *et al*. [10], Locality Sensitive Hashing [2] (LSH), Spectral Hashing of Weiss *et al*. [23], SIK of Raginsky and Lazebnik [15], and Iterative Quantization (ITQ) of Gong and Lazebnik [7]. As will be seen, in all cases SCT has superior performance at the very low bit end of the compression scale.

Such a low bit compression can find immediate application in multiple places, and we mention two use cases here: the first is direct representation of descriptors. These may be descriptors of local image regions (such as SIFT) or descriptors of the entire image (such as GIST [14], PHOW [4], VLAD [11] etc). The second is in large scale object retrieval systems, where descriptors are first quantized and an inverted index then used for fast access, and the *residual* (the difference between the cluster center and original descriptor) is compressed for finer descriptor matching [9, 10].

As far as we know there has been no scheme similar to SCT proposed before. The closest work is Chandrasekhar *et al.* [5], which, like SCT, exploits the fact that descriptor ordering is not important. However, their method is aimed at lossless compression for sets of binary vectors rather than the approximation for real vectors targeted here. Note, k-d trees have long been employed for efficient and approximate nearest neighbor algorithms [1, 13, 18]. Our objective is very different though – we aim for compression, whereas previous uses have stored all the original vectors (leading to an *increase* in storage requirements as the parameters of the tree must also be stored).

2. Set Compression Tree (SCT) Encoding

The objective of the Set Compression Tree (SCT) is to provide an extremely low bit-rate lossy compression of a set of descriptors. It can be used for brute-force nearest neighbor search by decompressing the descriptors from the SCT encoded set one by one and comparing them to the query vector. Section 4 explains how SCTs can be used for approximate nearest neighbor search.

Current methods for obtaining a compressed representation of descriptors all focus on compressing each descriptor *individually*. For very low bit-rate scenarios this approach is infeasible as many descriptors get assigned to the same value thus making it impossible to discriminate between them. Nearest neighbor search in such a system would have extremely low precision.

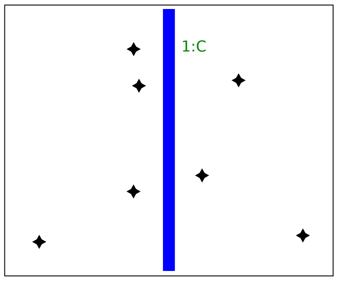
We instead focus on compressing all descriptors *jointly* such that the amortized cost of storing a descriptor is extremely low. This is achieved by not storing any information on a per-descriptor basis, but representing all descriptors together via a single binary tree. All of the storage requirements for descriptor representation are contained by the encoding of this binary tree. The compression method, *i.e*. tree building and encoding is described here, followed by implementation details in section 2.1.

The method proceeds by dividing the descriptor space by a sequence of binary splits that are predetermined and independent of the data to be compressed. After a number of divisions, a cell will only contain a single vector, and that vector is represented by the centroid of the cell. The method is best explained through an example, and Figure 1 shows the compression and encoding steps for the case of seven descriptors in a 2D space. The resulting representation uses only 2.14 bits per descriptor.

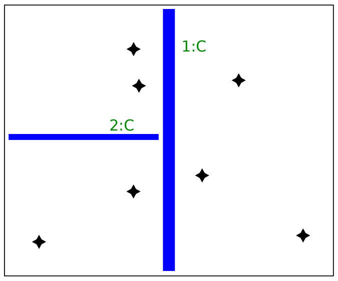
The key idea underpinning SCT encoding is that a single split contributes information to many descriptors; for the example in figure 1(a) the first split, encoded with two bits, halves the space for all seven descriptors, so for 0.29 bits per descriptor their positional uncertainty is halved. If, instead, every descriptor is encoded individually halving the feature space would cost 1 bit per descriptor.

To summarize the method, the encoding algorithm starts from a root bounding space which contains all descriptors and proceeds by a sequence of binary space partitions. A bounding space S is divided in a predefined manner independent of the descriptors it contains, for example, an axis aligned split is chosen such that the longest edge of the space is divided in half (the split sequence is explained in full in section 2.1). The split of the space S partitions it into two spaces A and B (*i.e*. such that $A \cup B = S$ and $A \cap B = \emptyset$), then *all* that is recorded at each split is the "outcome" of that split, *i.e*. information on the number of descriptors in A and B, denoted as $|A|$ and $|B|$ respectively. All outcomes fall into six categories that are recorded (figure 1(f)): (i) $|A| = 0$, $|B| > 1$, (ii) $|A| > 1$, $|B| = 0$, (iii) $|A| > 1$, $|B| > 1$, (iv) $|A| > 1$, $|B| = 1$, (v) $|A| = 1$, $|B| > 1$, (vi) $|A| = 1$, $|B| = 1$. Note, a space is split only if the number of elements $|S|$ is larger than one. Consequently, options $\{|A| = 0, |B| = 1\}$ and $\{|A| = 1, |B| = 0\}$ are not possible since, $|S| > 1$ and $A \cup B = S$ thus $|A| + |B| > 1$.

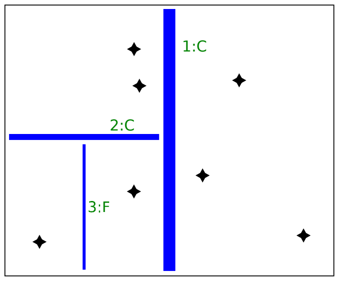
After the encoding, the entire data set is simply represented by the sequence of splits outcomes. This sequence is all that is required to reconstruct the space partitions (and thereby the centroid of the cell which represents the original vector). Decoding is analogous to the encoding procedure: the process starts from the root bounding space retracing the same sequence of binary splits as the encoding. A bounding space S is divided in the same predefined manner as



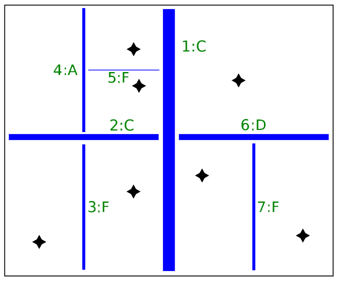
(a) 1st step; code = C



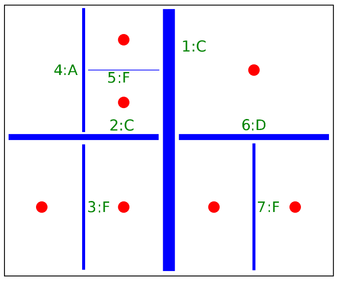
(b) 2nd step; code = C



(c) 3rd step; code = F



(d) Final tree



(e) Reconstruction

Symbol	Code	Number in child 1	Number in child 2
A	0000	$= 0$	> 1
B	0001	> 1	$= 0$
C	01	> 1	> 1
D	0010	> 1	$= 1$
E	0011	$= 1$	> 1
F	1	$= 1$	$= 1$

(f) Split outcomes and codes

Final tree encoding: CCFAFDF
Bitrate: $15/7 = 2.14$ bits per vector

Set tree encoding: 01 01 1 0000 1 0010 1

Figure 1: **SCT encoding.** The aim is to encode the seven 2-D vectors (*black stars*) by a sequence of binary partitions of the space (delimited by the outside rectangle). In *(a)-(e)* the space splits are shown in blue, thicker lines correspond to splits at levels closer to the root of the tree. *(a)-(c)* The first three steps in the tree construction. The construction (i.e. splitting bounding spaces) stops once all regions contain at most one descriptor; the final tree separating all seven vectors is shown in *(d)*. Next to each split its ordinal number in the sequence of splits is shown together with the code recording the outcome of the split. *(f)* The list of possible outcomes and their corresponding codes. For example, the second split *(b)* is shown as *2:C*, the table states that *C* means there is more than one vector in each of the newly created cells. *(e)* the vectors are represented by the centroid (*red circles*) of the final cells. Vector decompression is based solely on the tree, which is encoded by the sequence of split outcomes (15 bits, i.e. 2.14 bits per vector), and the centroids generate the reconstructed vectors.

used in the encoding (*e.g.* by halving the longest edge) and the split outcome is obtained from the SCT. If any of the children cells contains only a single descriptor, it is reconstructed using the cell centroid. The splitting of remaining cells which are known to contain more than one descriptor is continued in the same fashion, until none remains and the whole set of descriptors will then have been decompressed.

Figure 2 illustrates the benefit of using the SCT representation. 400 2-D data points are generated by sampling from a Gaussian mixture model with 16 Gaussian components. Compressing this data with 4 bits per descriptor by approximating a point with its closest cluster center, out of $2^4 = 16$ clusters, yields very large quantization errors. The centers in this example are obtained using k-means (note that at such a low bit-rate product quantization [10] degenerates to simple k-means), but any division of the space into 16 regions is bound to do similarly poorly. This is because on average $400/16 = 25$ points are assigned the same 4-bit code (corresponding to one region) and are thus completely indistinguishable between each other. However, quantizing the points jointly by sharing information between them performs very well, achieving a 21 times smaller mean squared error than k-means with only 3.05 bits per descriptor.

2.1. Implementation details

In this section technical details are discussed; they are sufficient to fully reproduce the experiments. The summary of all the steps of the SCT algorithm is given in figure 3.

In the case of nearest neighbor search there are three datasets involved: a training set, the database to be encoded, and the query set. The query set contains all the query vectors, and the database set contains the vectors on which nearest neighbor search will be performed, *i.e.* it contains the vectors that should be compressed. The training set is used to learn all the required parameters in an unsupervised way (for example PCA) and, in general, should be distinct

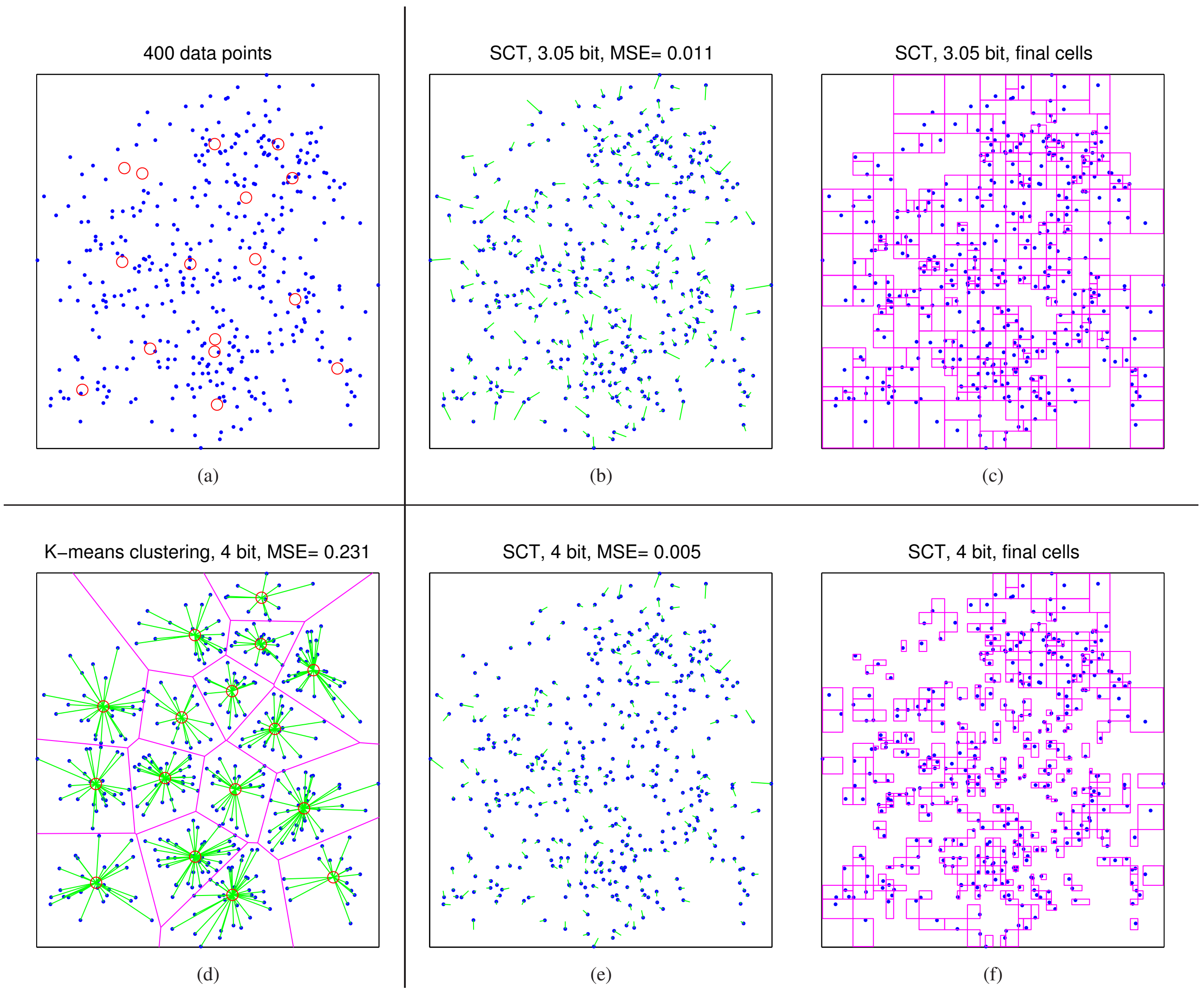


Figure 2: **Compression example.** All plots show a 2-D feature space, 400 synthetically generated 2-D points are shown in *blue*, *green* lines show the displacement between the original point and the approximation, *magenta* lines depict cells (*i.e.* uncertainties) associated with quantized descriptors. *(a)* The 400 points and the 16 GMM means (*red circles*) used to generate them. *(d)* Representing the data with 4 bits per point by using the closest cluster center (16 clusters, *red circles*, are obtained through k-means); the errors (depicted by long *green* lines) are large. *(b)* SCT encoding; with as little as 3.05 bits per point the errors are very small, with MSE (mean squared error between the original and reconstructed points) being 21 times smaller than for k-means; *(c)* shows the cells associated with each point; the point is finally represented by the centroid of the cell. *(e)* Tree encoding; at the same bit-rate (4 bits per point) as the k-means method; MSE is 46 times smaller. *(f)* shows the cells associated with each point.

from the database to be encoded in order to avoid implicitly storing database information in the learnt parameters. For an extreme example, one could "compress" a database of 1 million vectors by "learning" a 1 million dictionary identical to the database, and then representing the database using 20-bits per vector word ID thus achieving perfect representation at quite a low bitrate.

Requirements. An important requirement for SCT is that each component of the vector has a lower and an upper bound. The requirement is not very restrictive as this is the case with all commonly used descriptors, *e.g.* SIFT is bounded by 0 and 255 in each dimension.

```
(a) Training
1. Compute PCA, keep $D$ principal components for the training set $V$
2. Use a random rotation matrix $R$ ($D \times D$) to rotate the training set to get $\tilde{V}$
3. For each dimension $i$ in $[1, D]$:
       Store the distribution of values $\tilde{V}$ at dimension $i$, and lower ($L_i$) and upper ($U_i$) bounds

(b) Encoding
1. Rotate the $D$ principal components of database descriptors by $R$
2. Create the root bounding space, bounded by $[L_i, U_i]$ for dimension $i$, and associate all database descriptors with it
3. Find a bounding space $S$ (depth-first) containing more than one descriptor; if none found go to step (7)
4. Set $d = \arg\max_i (u_i^S - l_i^S)$ and $s = median(\{\tilde{V}_{i,d} | i : \tilde{V}_{i,d} \in [l_i^S, u_i^S]\})$
5. Create cells $A^S$ and $B^S$ by splitting cell $S$ at dimension $d$ and position $s$, move each descriptor from $S$ into $A^S$ or $B^S$ depending
   on which space they are bounded by
6. Encode the split outcome, see figure 1(f), and go to step (3)
7. Compute optimal Huffman coding for the recorded frequency of split outcomes, store it using 18 bits and re-encode the tree by
   replacing old codes with new, optimal ones
8. Refine final non-empty bounding spaces by splitting them additionally, using one bit per split

(c) Decoding
1. Create the current cell $C$ bounded by $[l_i^C, u_i^C] = [L_i, U_i]$ for dimension $i$. Set $|C|$ to $> 1$
2. If $|C| = 1$, refine $C$ by reading codes created in (b.8) and output its centroid rotated by $R^{-1}$
3. If $|C| \leq 1$, set $C$ to be equal to its parent, go to step (2)
4. If this is the first time $C$ is visited:
       Create cells $A^C$ and $B^C$ by splitting dimension $d$ at $s$, obtained in the same way as in (a.4)
       Decode the split outcome and assign $|A^C|$ and $|B^C|$ to values 0, 1 or $> 1$ accordingly
5. If $A^C$ was not visited, set $u_d^C = s$ (i.e. $C \leftarrow A^C$) and go to step (2)
6. If $B^C$ was not visited, set $l_d^C = s$ (i.e. $C \leftarrow B^C$) and go to step (2)
7. If parent exists, set $C$ to be equal to it and go to step (2), otherwise exit as all nodes have been visited

(d) Nearest neighbor search
1. Project the query descriptor to the $D$ principal components to obtain $q$
2. Use (c) to decode the tree, at step (c.2) compare $q$ with the outputted value
```

Figure 3: **SCT algorithm summary.** Lower and upper bounds for cell S in dimension d are denoted as l_d^S and d_d^S, respectively. Training data is referred to as V, the value of vector i at dimension d is $V_{i,d}$. Note that for nearest neighbor search one would actually also rotate q in step (d.1) by R and modify step (c.2) not to rotate the outputted centroid by R^{-1}, thus improving the speed.

Split choice. For a given cell, splits are determined in the following manner: The dimension to be split is chosen to be the one with the largest difference between the upper and lower bounds. It is straightforward to see that choosing to split the cell across its largest extent minimizes the expected approximation error for a descriptor (for example, imagine a degenerate 2-D cell where the extent in the x direction is long while the extent in the y direction is infinitesimal – splitting y would not be beneficial). Experiments confirm that pseudo-randomly choosing splitting dimensions (and recording the random seed in order to be able to reconstruct the tree) yields an inferior MSE to our scheme. The order of the splits is determined by always choosing the left child region first.

For a given cell and splitting dimension, the place of the split is determined with the aim of balancing the tree; a balanced tree is likely to produce similar sized cells for all descriptors thus resulting in similar magnitudes of reconstruction errors. The split is chosen according to the median value of the training data in the splitting dimension (clipped by the cell). Alternatively one can simply split the space in half, however this choice could lead to an unbalanced tree depending on the data distribution. A histogram for each component is retained to efficiently compute the median.

In summary, only the following information is stored from the training data in order to generate the split sequence: (i) upper and lower bounds for each dimension (in order to create the first, root cell); (ii) a histogram of training data values for each dimension (e.g. for SIFT, 128 histograms are stored).

Note, it is essential to have a cell splitting strategy that does not depend on the data inside the cell (as above). Otherwise the sequence of splits needs also to be stored, as one would do with a k-d tree, and this will incur a considerable storage cost.

Optimal binary encoding. The six outcomes of splits are encoded using optimal variable length binary codes. To achieve this tree encoding is performed in two stages. In the first stage splits are simply encoded using predefined suboptimal binary codes. During this stage, occurrence counts for each of the six outcomes are obtained. In the second stage, the tree is re-encoded by replacing the initial suboptimal codes with optimal ones. Huffman coding [6] is used to obtain optimal variable length binary codes by utilizing the recorded frequency of the six split outcomes. Storing the Huffman tree requires 18 bits in total and is certainly worth the reduced storage requirement for the tree representation.

Finer representation. It is simple to obtain a finer representation of a descriptor by increasing the bit-rate: the cell associated with it can be additionally split with a rate of 1 bit per split, encoding on which side of the split the descriptor is. Based on the desired bit-rate, the additional available bits can be equally distributed to refine each of the final bounding spaces, however, it is better to bias the refinement towards large bounding spaces (*i.e*. descriptors which have been represented poorly).

Dimensionality reduction. SCT encoding is not appropriate to use when the dimensionality of the descriptor vector is large compared to $log_2 N$, where N is the number of descriptors to be compressed. For example, compressing 1 million descriptors is expected to yield a tree of depth 20 ($log_2(1M)$), as it is approximately balanced, meaning that at most 20 dimensions will be split in order to obtain a final bounding space for each descriptor. Trying to compress 128-D SIFT descriptors with 20 splits would result in large approximation errors as at least 108 dimensions would remain unchanged and thus not convey any information about the value of the descriptor in any of the 108 dimensions. For this reason it is recommended to first perform dimensionality reduction on the data. We chose to zero-center the data followed by principal component analysis (PCA), keeping only D dominant directions. Since the variance of different components is not balanced we subject the D-dimensional vectors to a random rotation [7, 11], otherwise one would need to bias the choice of splitting directions towards the components which carry more information.

Obtaining the ID corresponding to a vector. As decompressing the SCT permutes input vectors according to the depth-first traversal of the tree, one might wonder how can one preserve the identity of the decompressed vectors without storing additional (costly) information. For example, in an image retrieval system, a retrieved feature vector (*e.g*. GIST [14], VLAD [11]) must be associated with its original image.

Here we distinguish two cases based on the nearest neighbor search strategy: (a) linear traversal, and (b) large scale retrieval using indexing.

In case (a), existing compression methods produce compressed vectors in the same sequence as input vectors, by for example product quantizing each vector in turn, and thus do not need to store any additional meta information to preserve the compressed vector identity (the original order may simply be alphabetical from file names). For example, returning the third compressed image descriptor means the system should return the *third image* in the image database. Thus the correspondence between compressed vector and original image is stored implicitly.

For SCT, the order of the reconstructed vectors depends on the order of the tree traversal and not the original order, thus SCT seemingly requires extra information to store the permutation of the vectors. However, this is ***not*** the case, for example the original images can simply be permuted so that the 'canonical ordering' of the images is the same as that of the decompression order.

More practically, the correspondence between the reconstructed vector order number can be stored in a look up table (LUT) that maps from order number to identity (e.g. its URL). *All* existing methods employ a LUT or some predefined ordering of the data. For example, an image retrieval system which represents each image using a single descriptor needs to be able to identify an image in some way from the retrieved descriptor. For images of Flickr or the web, a LUT is used to map between feature descriptor and URL.

In case (b), large scale retrieval, the situation is different, as the data is typically not traversed linearly. Instead, the dataset feature vectors may be grouped in some fashion, for example by vector quantizing, or accessed via an inverted index [20] (an example is discussed in more detail in section 4). In such cases an explicit LUT is required (e.g. between the entries of a posting list and the URLs of the images). Again, this requirement applies to any compression method.

To summarise, in spite of permuting the order of the input vectors, the SCT is capable of preserving vector identities without any additional storage requirements over those of other compression methods.

3. Evaluation and results

In this section we compare the compression, accuracy and retrieval speed of SCT to a number of other standard methods.

3.1. Datasets and evaluation procedure

We evaluate the performance of the SCT on two standard datasets: (i) the SIFT1M dataset of Jégou *et al*. [10], and (ii) the 580k GIST descriptors used by [7], which is a subset of the 80M Tiny Images dataset [21]. For both we follow the standard evaluation procedure of the authors, as summarized next.

1M SIFT descriptors (SIFT1M) [10]. This dataset is commonly used for evaluating approximate nearest neighbor methods, with an emphasis on low bit-rate descriptor representations for image search, as it consists of SIFT [12] descriptors. It contains 10k query descriptors, 100k descriptors used for training, and 1 million database descriptors. The descriptors are from the Holidays dataset [9] and Flickr images. Search quality is evaluated as the average recall of the first nearest neighbor at R retrievals (usually set to 100) for each query, *i.e*. the proportion of query vectors for which the Euclidean nearest neighbor using SIFT is ranked within the first R retrievals using the approximating method.

580k Tiny Images (Tiny580k) [7]. This dataset contains 580k images and is a subset of the Tiny Images dataset [21]; the images are represented by 384-D GIST descriptors [14]. It is randomly split into 1k query descriptors and 579k database descriptors which are also used for training; performance is measured across five different random splits of the data.

For this dataset the search performance is evaluated in three different ways, we will refer to them as *mAP-50NN*, *AP-thres* and *mAP-thres*. The first method (*mAP-50NN*), is based on the precision-recall curves proposed by the creators of the Tiny Images dataset [21]. The 50 true nearest neighbors for each query are labelled as positives, and the performance is evaluated as mean average precision (mAP) across the queries.

The latter two criteria (*AP-thres* and *mAP-thres*) concentrate on measuring distance preservation from the original descriptor space (in this case GIST) to the new space, thus measuring the effectiveness of hashing schemes to preserve neighborhood relations. For these cases, a set of "good neighbors" is determined by choosing a global distance threshold T (obtained by taking the mean distance to the 50th nearest neighbor), and labelling a descriptor as a positive for a particular query if its Euclidean distance from the query is lower than the threshold T. This method for ground truth generation was proposed by [23] and adopted by [7, 15]; they measure the performance as average precision (AP) of retrieval of "good neighbors" across all queries simultaneously, and we refer to it as *AP-thres*.

However, the test data associated with *AP-thres* is extremely unbalanced, such that 50% of the queries have less than 22 positives, while 7% have more than 30k positives. This means that as long as a system performs well on the 7% of the queries (i.e. these are retrieved correctly and first) then it will reach a "good" AP value, regardless of its poor performance on 93% of the queries. For example, if all of the 7% of the queries and their respective positives belong to a single cluster counting 30k descriptors, a simple 1-bit encoding being the indicator of the descriptor cluster membership (1: in the cluster, 0: out of the cluster) would yield a large AP.

To account for this imbalance we propose to also measure performance in terms of mean average precision (mAP), *i.e*. the mean of the average precision for each query, and refer to this as *mAP-thres*. The 1-bit encoding example would correctly perform poorly under this measure.

In summary, the *mAP-50NN* criterion is the most useful one of the three when evaluating image retrieval as it directly measures the ability of a system to return relevant results for any query image.

3.2. Baselines

SCT is compared to several state-of-the-art methods at various bit-rates, we briefly summarize them here.

(i) Product Quantization (PQ) [10]: Each descriptor is split into m parts and each part is vector quantized independently using k cluster centers, thus having a $m \cdot log_2(k)$ bit code per descriptor.

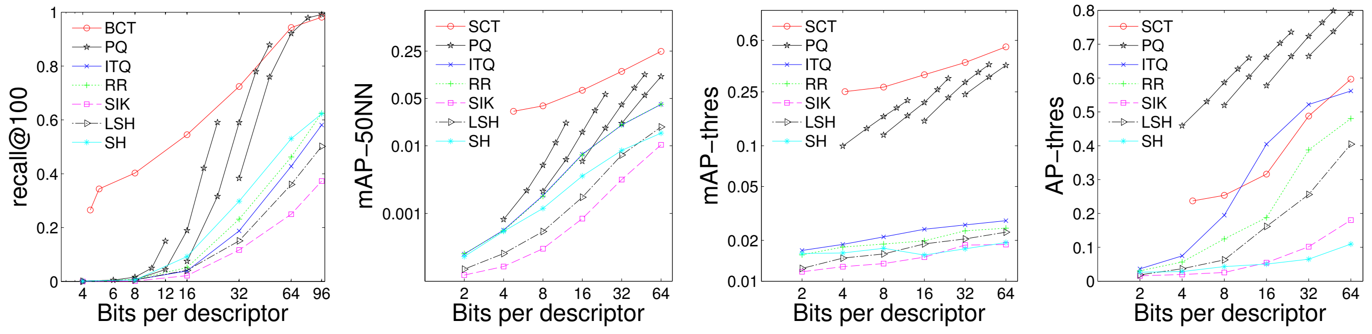
(ii) Locality Sensitive Hashing (LSH) [2]: The code is computed by taking the signs of descriptor projections onto random hyperplanes (normals are sampled from an isotropic Gaussian) with random offsets, the Hamming distance between descriptors encoded in such a way is closely related to the cosine similarity between the original vectors.

(iii) Shift Invariant Kernels (SIK) of Raginsky and Lazebnik [15]: The code is computed by taking the signs of random Fourier features with random offsets.

(iv) PCA with random rotation (RR) [7, 11]: Data is zero-centered and the most significant D principal components are kept and randomly rotated. The D dimensional code is obtained by taking signs of each dimension.

(v) Iterative quantization (ITQ) [7]: Gong and Lazebnik use the RR method and then proceed to iteratively find the rotation which minimizes quantization errors.

(vi) Spectral Hashing (SH) [23]: The coding scheme is obtained deterministically by trying to ensure that similar training descriptors get hashed to similar binary codes. This is a NP-hard problem so their approach is to solve a relaxed



(a) SIFT 1M (b) Tiny 580k (c) Tiny 580k (d) Tiny 580k

Figure 4: **Comparison of SCT with state-of-the-art.** Best viewed in color; note the log scale for mAPs in *(b)* and *(c)*. The different curves associated with PQ correspond to different numbers of sub-quantizers (from left to right: 1, 2, 4, 8), the combinations of settings reproduce the ones used in the original paper [10]. SCT outperforms all methods at low bit-rates for both datasets and all evaluation measures apart from *AP-thres*, where the test data is extremely unbalanced and the measure inappropriate (see the discussion in section 3.1). SCT continues to dominate all competing methods at higher bit-rates as well.

optimization problem.

Methods (ii)-(vi) rank descriptors based on their Hamming distance from the query which is binarized in the same fashion. For product quantization (i) the asymmetric distance computation method [10] is used since Jégou *et al.* report it to be superior; the distance between the query and a database vector is approximated by the distance between the raw (non-quantized) query and the quantized representation of the database vector.

We use publicly available code for all of the baseline methods and replicate the results in [7, 10], these two papers together cover all the baseline methods.

3.3. Results and discussion

Figure 4 shows the comparison of SCT with the baseline methods. SCT clearly outperforms all competing methods at very low bit-rates on both datasets and using all evaluation metrics. For example, on the SIFT 1M dataset SCT achieves a recall@100 of 0.344 at 4.97 bits per descriptor, while the next best method, product quantization, achieves 0.005 at 6 bits, *i.e.* 69 times worse. Even at 16 bits, *i.e.* 3.2 times larger, PQ only reaches 55% of SCT performance at 4.97 bits per descriptor. The next best method after SCT is PQ, and PQ is then superior to all other methods.

Despite the very low bit-rate, SCT by construction represents *all* database descriptors distinctly, thus making it possible to distinguish between them and rank them appropriately. In contrast, all other methods, by quantizing each descriptor individually into a small number of bits, effectively identically represent a (possibly very large) set of database descriptors, as previously illustrated in Figure 2. This means that, for example, using the Tiny 580k dataset and four bits per descriptor, on average a descriptor has the same representation as another 36k ones; it is impossible to rank the first 36k results for a given query, as any of the 36k factorial permutations is equally likely under these representations. This argument also shows that performance evaluation based on *AP-thres* is very misleading. For example, PQ under *AP-thres* achieves an AP of 0.459 with 4 bits per descriptor. The reason for such a "good" performance is the fact that the test data associated with *AP-thres* is extremely unbalanced, as explained in section 3.1.

Figure 5 shows qualitative retrieval results on the Tiny 580k dataset. As expected from the quantitative evaluation shown in figure 4, SCT performs much better than the state-of-the-art methods while representing descriptors using three times fewer number of bits.

Dimensionality reduction. Figure 6 shows the performance of SCT as a function of the number of principal components (PC) used and bit-rate. It can be seen that, as expected, at extremely low bit-rates performance drops with increasing number of PCs due to increasing approximation errors. However, increasing the bit-rate by further splitting final bounding spaces makes it again appropriate to use SCT with this larger number of PCs, which in turn improves the performance as the underlying data is represented more accurately using more PCs.

For a given number of PCs, it is also important to note that the SCT performance reaches the upper bound (*i.e.* the performance that is obtained by using raw descriptors with the same number of PCs) at quite low bit-rates. For example this point is reached at 32 bits per descriptor for 16 PCs, so only two bits per dimension are sufficient to encode a value which would commonly be represented using 32 or 64 bits (single or double floating point number).

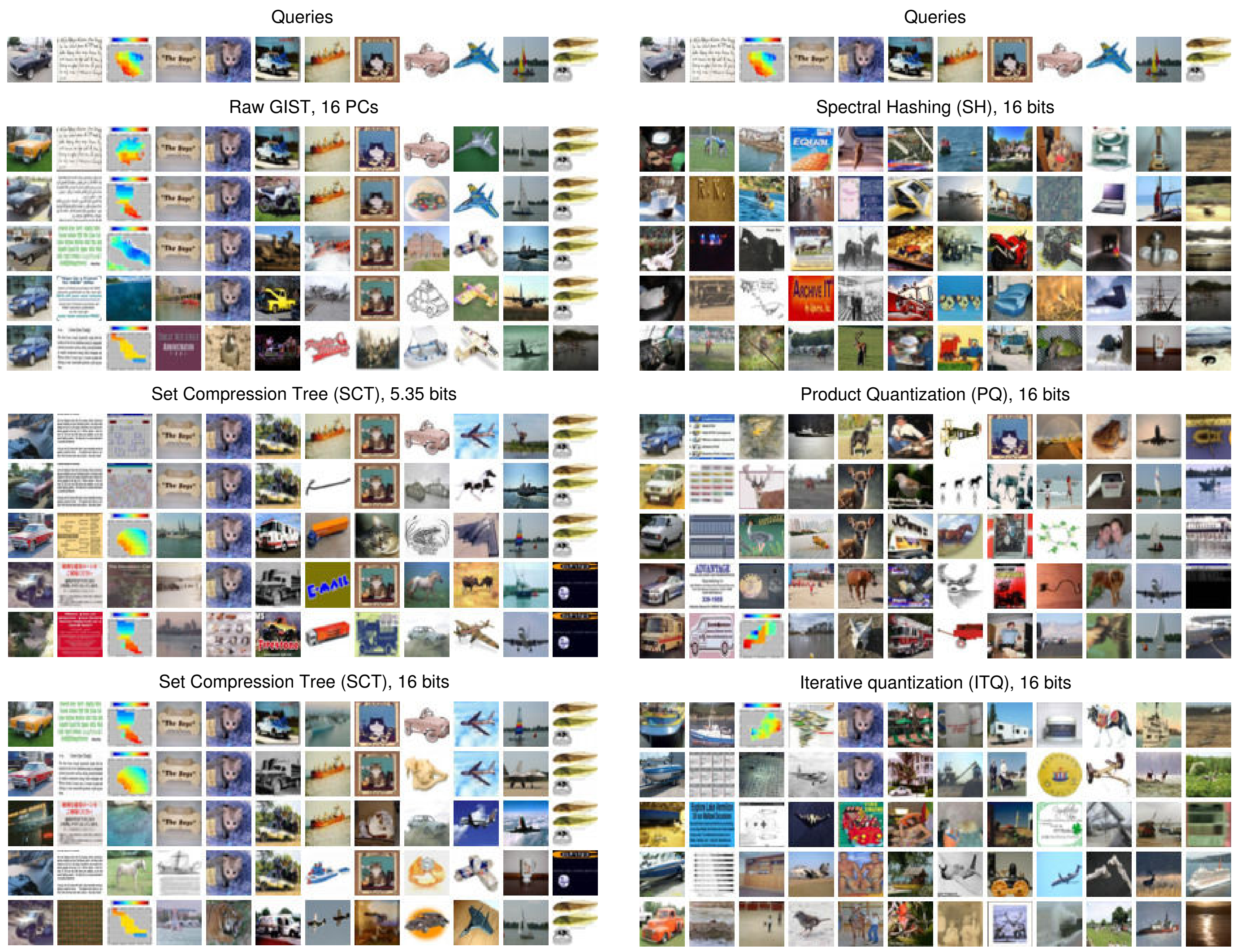

Queries
Queries
Raw GIST, 16 PCs
Spectral Hashing (SH), 16 bits
Set Compression Tree (SCT), 5.35 bits
Product Quantization (PQ), 16 bits
Set Compression Tree (SCT), 16 bits
Iterative quantization (ITQ), 16 bits

Figure 5: **Retrieval quality on Tiny 580k dataset.** Best viewed in color. *First row* shows twelve example queries, each of the remaining *six blocks* show the top five retrieved images for each of the queries. Set Compression Tree (SCT) at only 5.35 bits per descriptor outperforms state-of-the-art methods, namely Spectral Hashing, Product Quantization and Iterative Quantization, while using three times less storage for descriptor representation. Using SCT with 16 bits per descriptor further improves retrieval quality. Note that other methods are not capable of even finding near-duplicate images at such low bit-rates.

Fast and low memory footprint of coding and decoding. All timings are measured on a laptop using a single 2.66 GHz processor. The un-optimized program compresses 1 million SIFT descriptors using 32 principal components in 14 seconds, while decompression and nearest neighbor search take 0.5 seconds. The search time scales linearly with the number of database descriptors, searching 580k GIST descriptors (again using 32 principal components) takes 0.26 seconds. In the decoding stage not all of the cells need to be stored in memory at once – the tree is traversed depth-first, so only a single cell representing the "current" node is kept at any one time. Once a leaf node or a node with already visited children is encountered, the current bounding space is reverted to its parent's bounding space. For this to be possible only a small amount of additional information needs to be maintained for each previous split in the current path from the root node, which is quite short as for 1 million descriptors the depth of a balanced tree is equal to 20.

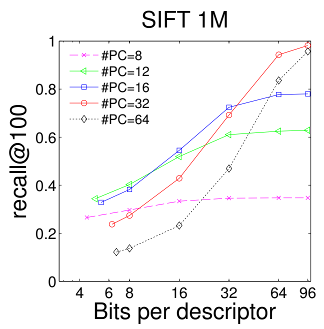


Figure 6: **Influence of the number of principal components (PC) on SCT performance with varying bit-rates.** At extremely low bit-rates performance drops with increasing number of PCs due to increasing approximation errors. However, increasing the bit-rate by further splitting final bounding spaces makes it again appropriate to use SCT with this larger number of PCs, which in turn improves the performance as the underlying data is represented more accurately using more PCs. The performance for each scenario (number of PCs) saturates at "large" bit-rates, each of the saturation levels has been verified to be equal to the actual performance when using raw database descriptors with the same number of PCs. This shows that SCT reaches maximal possible performance at what is actually quite a low bit-rate (see figure 4(a)).

4. Discussion and recommendations

In this section we discuss the properties of SCT, its advantages and disadvantages compared to other encoding methods, and three use cases for it.

Unique description. Every descriptor is assigned a unique bounding space, and all bounding spaces are disjoint. This means that even in areas of high descriptor density it is still possible to discriminate between descriptors, a characteristic which does not exist in any of the competing methods. This could potentially be used to disambiguate between descriptors using a second nearest neighbor test [12].

Asymmetric in nature. As noted by [11], it is beneficial not to have to quantize query vectors when performing nearest neighbor search, as this obviously discards relevant information. SCT is asymmetric at its core as query vectors are compared directly to the reconstructed database vectors.

Making the representation finer. As described in section 2.1, by increasing the bit-rate: the cell associated with it can be additionally split with a rate of 1 bit per split, encoding on which side of the split the descriptor is. Other schemes are certainly possible, *e.g.* representing the residual (the difference between the original descriptor and the reconstruction) with any of the other methods such as product quantization, LSH or ITQ.

Small dimensionality requirement. As discussed in section 2.1, SCT is not applicable when the number of splits per descriptor is smaller than the data dimensionality, since in this case many of the dimensions are not split thus causing large approximation errors. To compress vectors of large dimensionality using the SCT requires dimensionality reduction via PCA. Note that the requirement for "small" descriptor dimensionality is not as limiting as it might seem as PCA is commonly used for compacting descriptors such as VLAD [11] or even SIFT [17]. There are many cases where "small" descriptors are used, *e.g.* Google's CONGAS descriptors are 40-D [24], Simonyan *et al.* [19] achieve impressive performance with 60-D, while Jégou and Chum [8] demonstrate very good performance when reducing 130k-D VLAD vectors using PCA down to 128-D.

Furthermore, all baselines apart from PQ perform dimensionality reduction: RR and ITQ start from PCA (bitrate equals the number of PCs), LSH and SIK use random projections or Fourier features (bitrate equals the number of projections), and SH learns the projections from training data (bitrate equals the number of projections). Thus all methods apart from PQ actually suffer from a much worse problem than SCT since, for example, for 8 bits ITQ is forced to use only 8 PCs.

Adding descriptors to the database. For applications where the image database grows in time it is important to be able to add descriptors efficiently to the SCT. Adding a single descriptor does not disturb the tree structure much (*e.g.* imagine the example in figure 1 where a point is left out and added to the tree after the tree is built) as the sequence of splits is independent of the compressed data. Thus, all the split outcomes will remain the same (*i.e.* the tree does not need to be re-encoded) apart from the split that creates the leaf node (cell) that contains the new descriptor. By construction, if the cell is not empty then it contains exactly one descriptor and new splits are added in the same manner as when the original tree was built, *i.e.* until a split is encountered which discriminates between the two descriptors, namely the "old" and the "new" one. Thus, adding a descriptor to the SCT is very efficient and only requires access to at most one original database descriptor (zero if the relevant leaf cell is empty). Note that for the same operation none of the baselines requires access to the original descriptors. However, this limitation is not significant as the original descriptors do not need to be stored in RAM but can be stored on disk.

Use cases. SCT has been demonstrated to perform well for the problem of large scale image retrieval (section 3.3), searching a database of 580k images (represented by GIST descriptors) in 0.26 seconds. Since query-time speed is linear in the number of descriptors, with no changes to the system up to 7 million images images can be searched immediately (3 seconds per query) on a single core. SCT can easily be parallelized, thus enabling 30 million images to be searched on a typical quad-core computer. Note that storing 30 million descriptors at 6 bits per descriptor requires only 23 MB.

For larger scale databases with billions of images memory requirements would still remain low, however processing power would be the limiting factor as a linear scan through the data is infeasible. In this case one can, in the manner of Jégou *et al*. [9, 10], vector quantize the database descriptors coarsely and use SCT to compress the residual. At search time the query descriptor is vector quantized and only compared to database descriptors quantized to the cluster through the use of an inverted index [20], while the SCT encoded residual is used for refining the matches. Searching 1 billion images quantized into 1k clusters would thus take about half a second using a single core processor (i.e. to decompress a single cluster containing 1M descriptors).

The same system can be used for large scale object retrieval where database images are typically represented using 1k local descriptors (*e.g*. SIFT [12]). For this use case a query image is also represented using 1k descriptors, thus the same number of nearest neighbor searches would need to be issued. Searching 1 million images by quantizing the 1 billion database descriptors into 100k clusters and using 4 processing cores would yield results in 1.25 seconds (i.e. to decompress 1k clusters each containing 10k descriptors).

Summary. The Set Compression Tree (SCT) hugely outperforms all competing methods at extremely low bit-rates, making it the only tool of choice for very large scale retrieval purposes, where it is critical for fast retrieval that all the relevant data fits in RAM.

Acknowledgements. We are grateful for financial support from ERC grant VisRec no. 228180.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 1997.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Comm. ACM*, 2008.

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

[4] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Proc. ICCV*, 2007.

[5] V. Chandrasekhar, S. Tsai, Y. Reznik, G. Takacs, D. Chen, and B. Girod. Compressing feature sets with digital search trees. In *International Workshop on Mobile Vision*, 2011.

[6] C. Cormen, T.and Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, 1990.

[7] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. CVPR*, 2011.

[8] H. Jégou and O. Chum. Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening. In *Proc. ECCV*, 2012.

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

[10] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE PAMI*, 2011.

[11] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Proc. CVPR*, 2010.

[12] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[13] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithmic configuration. In *Proc. VISAPP*, 2009.

[14] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 2001.

[15] M. Raginsky and S. Lazebnik. Locality sensitive binary codes from shift-invariant kernels. In *NIPS*, 2009.

[16] M. Rastegari, C. Fang, and L. Torresani. Scalable object-class retrieval with approximate and top-k ranking. In *Proc. ICCV*, 2011.

[17] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proc. CVPR*, 2011.

[18] C. Silpa-Anan and R. Hartley. Localization using an image-map. In *Australasian Conf. on Robotics and Automation*, 2004.

[19] K. Simonyan, A. Vedaldi, and A. Zisserman. Descriptor learning using convex optimisation. In *Proc. ECCV*, 2012.

[20] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proc. ICCV*, 2003.

[21] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE PAMI*, 2008.

[22] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *Proc. ECCV*, 2012.

[23] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008.

[24] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven. Tour the world: building a web-scale landmark recognition engine. In *Proc. CVPR*, 2009.