Scalability of Local Image Descriptors: A Comparative Study

Herwig Lejsek, Friðrik Heiðar Ásmundsson, Björn Þór Jónsson, Laurent Amsaleg

Scalability of Local Image Descriptors: A Comparative Study

Herwig Lejsek, Fridrik H. Ásmundsson, Björn Thór Jónsson
Reykjav´ık University
Ofanleiti 2
IS-103 Reykjav´ık
Iceland
{herwig, fridrik01, bjorn}@ru.is

Laurent Amsaleg
IRISA–CNRS
Campus de Beaulieu
35042 Rennes
France
laurent.amsaleg@irisa.fr

ABSTRACT

Computer vision researchers have recently proposed several local descriptor schemes. Due to lack of database support, however, these descriptors have only been evaluated using small image collections. Recently, we have developed the PvS-framework, which allows efficient querying of large local descriptor collections. In this paper, we use the PvS-framework to study the scalability of local image descriptors. We propose a new local descriptor scheme and compare it to three other well known schemes. Using a collection of almost thirty thousand images, we show that the new scheme gives the best results in almost all cases. We then give two stop rules to reduce query processing time and show that in many cases only a few query descriptors must be processed to find matching images. Finally, we test our descriptors on a collection of over three hundred thousand images, resulting in over 200 million local descriptors, and show that even at such a large scale the results are still of high quality, with no change in query processing time.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Multimedia Databases, Query Processing*

General Terms

Algorithms, Performance

Keywords

Local image descriptors, scalability, PvS-framework, high-dimensional indexing, median rank aggregation

1. INTRODUCTION

With the proliferation of digital media and online access, multimedia retrieval and image retrieval, in particular, is growing in importance. The computer vision community has recently started a trend towards advanced image description schemes using *local descriptors* (e.g., see [7, 1, 15, 11]). The applications of local descriptor schemes include face recognition, shape recognition and image copyright protection [3, 12]. With these schemes, each image yields many descriptors (several hundreds for high-quality images), where each descriptor describes a small "local" area of the image. Two images are typically considered similar when many of their descriptors are found to be similar.

All of these approaches, however, have only been studied and compared at a small scale. In [15], 112 images were used, resulting in about 150 thousand local descriptors, while in [11], only 20 images were used. A performance evaluation of state-of-the-art local descriptor schemes was presented in [16]. In that study, the SIFT descriptors of [15] obtained the best results. Unfortunately, the measurements only consisted of comparing the original query image with images of the same subjects, taken from different perspectives, which makes it hard to predict how they will perform with collections of tens of thousands of images or more.

There are three primary issues associated with scalability of image descriptors.

- First, it is key to answer queries efficiently, even with very large image (and descriptor) collections. This issue demands efficient database support techniques.
- Second, it is necessary to ensure effectiveness; that queries return useful results from very large collections. Note that studying effectiveness at a large scale requires first having efficient database support.
- Finally, the descriptors must be created efficiently, in particular in a high-throughput environment.

This paper addresses all three issues.

1.1 Scalable Database Support

Up until recently, scalable database support for local descriptors has been sorely lacking [1]. Two approaches specifically geared to local descriptor retrieval were proposed in [2] and [12]. The retrieval scheme proposed in [2] was based on pre-clustering the data and running approximate nearest neighbor queries, while the scheme proposed in [12] was based on locality sensitive hashing. Neither approach has been shown (nor is likely) to scale to very large collections.

As far as we know, only two approaches achieve efficient query processing for large collections of local descriptors. In [10], a system is proposed to verify, in real time, whether

MM'06, October 23–27, 2006, Santa Barbara, California, USA.

a video broadcast on TV comes from a reference collection containing over 40,000 hours of video. It uses a smart encoding of local descriptors together with a statistical similarity search which uses a Hilbert space filling curve. The processing time is sublinear in collection size for reasonable collection sizes (albeit with an asymptotic linear behavior).

The second scheme that achieves efficient query processing is the PvS-framework [13]. This framework consists of two key parts. First, it transforms costly nearest neighbor searches in multi-dimensional space into efficient unidimensional B^+-tree accesses using a combination of projections of vectors to random lines and segmentation of the projected space. Second, descriptor distance is computed efficiently using median rank distance [6], which approximates the expensive Euclidean distance function. The PvS-framework has been designed to return high-quality search results in a time which is dependent only on the number of query descriptors. With the PvS-framework, computer vision researchers can analyze the quality of their local descriptors at a large scale; this paper describes such a study.

1.2 Contributions of the Paper

We have compared three major local descriptor schemes, namely the SIFT [15], the PCA-SIFT [11], and the RDTQ descriptors [1]. While we observed that PCA-SIFT and SIFT descriptors give good and similar results, we saw that in some cases they missed opportunities for matches, in particular in the context of very large image collections. Furthermore, we felt that the descriptor creation could be made more efficient, which is very important in high-throughput applications. To address both problems, we propose in this paper a new local descriptor scheme of the SIFT family. This scheme is called Eff^2 as these descriptors are computed *Eff*iciently and yield *Eff*ective search results.

We then perform a detailed comparison of these four local descriptor schemes using a collection of almost thirty thousand images, focusing on the quality of the results. The comparison shows that our new scheme performs the best in almost all cases. In some cases it detects heavily modified images that the other schemes fail to recognize. We also study the likelihood of false positives and determine that our new scheme performs best on that metric also.

With the PvS-framework, query processing time is fixed per query descriptor. Therefore, one way to reduce the query time is to allow the search to stop before all query descriptors have been processed. We propose two stop rules and show that in many cases as few as ten query descriptors must be processed in order to obtain answers of high quality. As before, our new descriptor scheme performs best.

Finally, we apply our new local descriptor scheme to an image collection of over three hundred thousand images, resulting in over 200 million 72-dimensional descriptors. We show that even at such a large scale, high quality results are returned, with no increase in query processing time.

1.3 Overview of the Paper

This paper is organized as follows. Section 2 describes the PvS-framework. Section 3 briefly reviews the three local descriptor schemes used in this study and then describes our Eff^2 descriptors. Section 4 describes the stop rules for early termination of query processing. Section 5 presents our experimental setup and Section 6 the performance results. Finally, Section 7 concludes and gives open issues.

2. THE PvS-FRAMEWORK

The PvS-framework is a novel multi-dimensional indexing scheme specifically geared to local descriptor retrieval. It provides efficient and scalable database support for local descriptors. The PvS-framework borrows heavily from the OMEDRANK approach, which is described in Section 2.1. The PvS-framework, however, overcomes severe limitations of OMEDRANK as explained in Section 2.2.

2.1 OMEDRANK

With OMEDRANK, Fagin et al. proposed an approach for efficiently evaluating single descriptor nearest-neighbor queries in high-dimensional collections [6]. Offline, that approach first creates a set of empty B^+-trees. In addition, it associates to each B^+-tree a line that randomly crosses the multi-dimensional space. Each descriptor of the collection is then projected onto each of the random lines. For one particular random line, a pair containing the identifier of the descriptor and the value of that descriptor along the random line is inserted into the appropriate B^+-tree. Each B^+-tree keeps pairs ordered by increasing projected values.

At search time, the query descriptor is first projected onto each of the random lines. Each B^+-tree is then probed with the appropriate projected value to find a starting point for the query. Next, two cursors are started for each index, respectively reading successively lower and higher values. The cursors are used in a round-robin fashion, to simultaneously traverse all B^+-trees and retrieve descriptor identifiers. The algorithm keeps track of how often each descriptor is encountered, while the cursors are moved. When a particular descriptor has been seen in more than half of the B^+-trees, it is returned as the nearest neighbor. Processing then continues, until the k nearest neighbors have been returned. Overall, each random line gives a ranking of the database descriptors with respect to the query descriptor, and these rankings are aggregated to get the nearest neighbors.

The analysis in [13] demonstrated that applying OMEDRANK directly to a large set of local descriptors is very inefficient, both in terms of response time and quality of the search results. At large scale, random projections introduce a significant level of noise as many descriptors that are far apart in the space have near ranks on some lines; furthermore, vectors near in space tend to have distant ranks as they get separated by other vectors far in space. When indexing a large set of vectors (several millions), results of high quality can only be returned by OMEDRANK by using many random lines (many B^+-trees) for projections. Aggregating ranks and cross-analyzing evidences found on numerous lines counter-balances the noise-like effect of projections, restoring quality. The cost of probing each B^+-tree and then aggregating the results, however, becomes prohibitive. The only way to reduce the cost of executing OMEDRANK is to reduce the number of B^+-trees that are created, which, in turn, dramatically degrades the result quality as in most cases none of the vectors returned by the search are actual neighbors of the query point.

2.2 Overview of the PvS-Framework

We designed the PvS-framework to overcome the problems observed with OMEDRANK when used with large collections of local descriptors. By replacing each B^+-tree used by OMEDRANK with a PvS-index, results of good quality are obtained with excellent performance. As for OMED-

RANK, the PvS-framework uses random projections to build the PvS-indices. The creation of one PvS-index differs, however, because it combines multiple levels of random projections of the descriptors with multiple levels of data segmentation. Querying a PvS-index returns the identifiers and the ranks of some descriptors from the database; those ranks are then aggregated with the results from other PvS-indices using the OMEDRANK approach.

The PvS-framework has been designed to return approximate k-neighbors of high quality using only a single I/O operation per PvS-index. We achieve this goal by 1) using many different random lines for projections to build a single PvS-index, 2) segmenting the data space along those lines until each segment can be fetched in one I/O, and 3) introducing redundancy into the index to increase the likelihood of finding the correct results.

The creation of one particular PvS-index is detailed in the next section. As for the B^+-trees used in Fagin's approach, one PvS-index cannot be used in isolation; several PvS-indices must be created to support the rank aggregation algorithm of [6]. Therefore, the process detailed next must be repeated for each PvS-index.[1]

2.2.1 Creating One PvS-Index

The creation of a single PvS-index is done using a repeated combination of projecting and segmenting the high-dimensional descriptors. Initially, all descriptors of the collection are considered to be part of a single temporary segment. Then, the descriptors are projected onto one random line (as for OMEDRANK). The descriptors are then segmented based on the value of their projection to this line into a set of new temporary sub-segments of identical cardinality.

This process of projecting and segmenting is repeated for all the new temporary segments (of decreasing cardinality), using a new random line at each level. The process stops when the number of descriptors in a temporary segment drops below a limit, designed to be disk I/O friendly. When this limit is reached, the descriptors in the temporary segment are projected again to a new line and the descriptor identifiers are written into a B^+-tree on disk according to the rank of their projected value. The intermediate nodes of the PvS-index keep track of the lines used for projection, the cut-points along lines, and pointers to sub-nodes or leaves. Only leaves contain references to descriptors. The nodes of the PvS-index form a balanced tree.

One thing is worth noting here. Repeatedly projecting and segmenting descriptors tends to keep in a single segment the descriptors that are close in the feature space, thereby producing high quality results at query time, even at a very large scale. To further improve result quality, however, we introduce redundancy by allowing the segments to overlap, thus making sure that near neighbors are likely to be together in at least one segment.

Figure 1 illustrates the data structure of a single PvS-index. In the figure, a "P" is a pointer to the random line to project to (each "P" indicates a different random line). In the nodes, the flat lines represent the random lines, while the marks on the flat lines indicate the cut-points which guide the index creation and search. The overlapping segments are shown above and below the lines, each pointing to a sub-node or a leaf. The PvS-index in Figure 1 is said to have two levels with a [3, 5] segmenting strategy, yielding $3 \times 5 = 15$ final segments stored on disk. Each final segment is organized as a B^+-tree and contains a fixed number of descriptor identifiers. Note that the PvS-framework is a static approach.

[1] In [13], using only three PvS-indices was sufficient to obtain search results of high quality with short response times; this setting is used in Section 5.

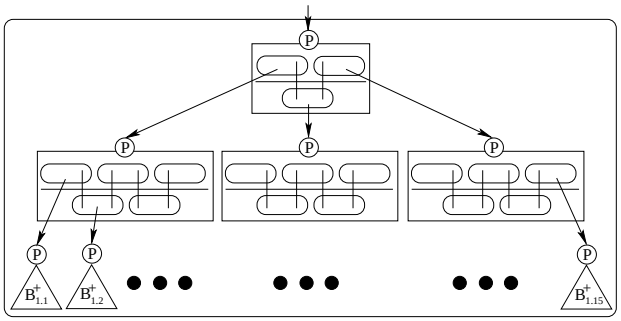

Figure 1: A single two-level PvS-index with a [3, 5] segmentation strategy.

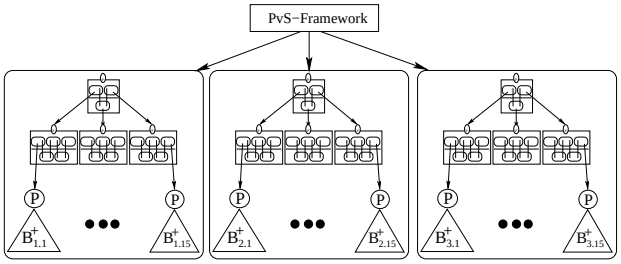

Figure 2: A complete PvS-framework with three PvS-indices (2 levels, [3, 5] segmentation strategy).

2.2.2 PvS-Framework Search

As mentioned above, several PvS-indices must be created before allowing searches. These indices have strictly identical structure, but each uses different random lines, resulting in different cut-points and segments containing different vectors. Figure 2 shows a complete PvS-framework made of three PvS-indices, each having two levels.

As for OMEDRANK, the PvS-indices created to index a particular collection are probed simultaneously. During the processing of one query descriptor, each PvS-index is traversed as follows. At each level of the index, the query descriptor is projected to the line associated with the current node. Using the cut-points for that level, the search is then directed to the appropriate sub-level. This process of projection and choosing the right sub-level is repeated until the search reaches a leaf segment.

The leaf segment (which is a B^+-tree) is fetched into memory for all indices. Then, as for OMEDRANK, cursors are initialized and moved inside each B^+-tree retrieving descriptor identifiers until the aggregation of their ranks has returned the k nearest neighbors of the query point.

For each query descriptor, the processing time is only dependent on the number of PvS-indices searched because the PvS-framework requires *a single* disk read per index (CPU cost is negligible). Response time is therefore independent of the size of the indexed collection. Guaranteeing effectiveness when searching a very large collection is clearly crucial; the results of [13] show that large collections must be segmented and projected enough to provide high-quality results.

3. LOCAL DESCRIPTORS

In this section, we first briefly describe the three state-of-the-art local descriptor schemes used in this paper. We then discuss scalability issues in Section 3.2 and present our new Eff^2 descriptors in Section 3.3.

3.1 Background

Local descriptors are typically computed on small areas of images in a two step process. The first step identifies points of interest in the image and the second step computes the values of the descriptors using the characteristics of the signal around each point of interest. Local descriptor schemes differ in the way points of interest are determined, in the number of points subsequently used and in the way the image signal is manipulated to compute each local descriptor. Overall, local descriptor schemes handle rotations, translations of objects in images, changes in color and to some extent compression and scale changes. See [16] for a comparison of several schemes and their invariance properties.

In this paper, we evaluate the recognition performance of three state-of-the-art local descriptor schemes, namely the RDTQ, the SIFT and the PCA-SIFT schemes. The RDTQ scheme by Amsaleg and Gros [1] uses the seminal corner and edge detector by Harris and Stephens [9] and generates vectors of 24 dimensions. Local descriptors are then computed around the points of interest by convoluting the local signal with a Gaussian and by using derivatives up to the third order. The SIFT descriptors by Lowe [15, 4] use the Difference of Gaussian (DoG) interest point detector which better handles significant changes in the scale of images (see [5, 14, 19]). The SIFT descriptors then encode the image gradients and their orientations around the points into a 128-dimensional histogram. The PCA-SIFT descriptors [11, 12] simply apply PCA on Lowe's SIFT descriptors, reducing their dimensionality to 36.

3.2 Scalability Issues

We obtained access to two programs to generate SIFT descriptors for our experiments. The original binary code by Lowe [15] creates a large number of 128-dimensional descriptors per image (unfortunately, the source code is not available). We did, however, have access to the source-code of version 2.2 by Nowozin[2], which created fewer descriptors, with a choice of 36, 72, and 128 dimensions.

As mentioned in the introduction, the primary issues associated with scalability of image descriptors are efficiency of query processing, effectiveness of the search and efficiency of the descriptor creation. We now address the problems and opportunities associated with these issues.

Efficiency of the Search: While the query performance of the PvS-framework is only dependent on the number of query descriptors, the index creation time is dependent on both the size and dimensionality of the descriptor collection. The original SIFT implementation creates on average over 1,200 descriptors for our images, and for some rescaled query images it creates more than 9,000 descriptors. With so many query descriptors, query processing takes a significant time. We then observed that despite Nowozin's implementation creating fewer descriptors, it actually gave better results in many cases. Furthermore, we observed that while there was a large difference in the result quality between 36 and 72 dimensions, using Nowozin's descriptors, the quality was not improved much when going to 128 dimensions. For these reasons, we decided to restrict the dimensionality of our proposed descriptors to 72 dimensions and to also restrict the number of descriptors that were generated. In particular we do not consider creating descriptors from an "up-scaled" image octave (rescaled to twice the size of the original image) as this image octave yields many low-level descriptors; this method was also used by Nowozin to reduce the number of descriptors compared to SIFT.

Effectiveness of the Search: Aside from the dimensionality issues described above, we also observed that in some cases corresponding descriptors of slightly modified images were quite distant from the original descriptors in the high-dimensional space. With the limited image collection of [15], these distant descriptors were still found as matches. With a very large collection, however, it is likely that many unrelated descriptors fall between the two descriptors, causing the votes to be lost. Furthermore, due to the segmentations of the PvS-indices, the two descriptors may fall in different segments, again causing the votes to be lost. We therefore studied closely the effects of image modifications and attempted to limit the distance between modified descriptors such that they are more likely than not to fall within the same segment in the PvS-indices.

Efficiency of the Descriptor Creation: When querying for images in a high-throughput application, descriptor creation may become a bottleneck. In the creation of our descriptors, we attempted to code the creation process as efficiently as possible, e.g., by using lookup-tables for difficult computations when possible. Additionally, by not considering the "up-scaled" image octave, many descriptor computations are saved.

3.3 The Eff^2 Descriptors

In this section we describe the creation of the Eff^2 descriptors. The pseudo-code in Figure 3 describes the order in which the descriptors are created, and the rule used to limit the number of descriptors. As the figure shows, the descriptors are created in the order from high-level descriptors to low-level descriptors. In the following, we describe in more detail the interest point detection and the descriptor creation, which are at the heart of Figure 3.

3.3.1 Interest Point Detection

As with the SIFT descriptors, the interest points are detected using Difference of Gaussians over different scales. In order to detect the local maxima and minima each sample point is compared to its eight neighbors in the current DoG scale and nine neighbors in the scales above and below it.

Finding DoG maxima at very high scales is difficult since strongly rescaled images have low contour information, as if they were strongly smoothed. The lower scales of an image tend to generate almost all the interest points, leaving only few interest points at high scales, which precludes any robust handling of strong rescalings. Several normalization methods have been proposed to address this issue (e.g., see [17]). We used for our Eff^2 descriptors a trick which, in a certain way, reinforces the contrast of images at higher DoG scales. We applied increasing gamma correction to the DoG as the scales increase. For the experiments reported in this

[2]See http://user.cs.tu-berlin.de/~nowozin/autopano-sift.

```
Create all image octaves
loop from smallest octave to largest
    Create all Gaussian scales
    Calculate all Differences of Gaussians
    loop from largest DoG scale to smallest
        loop for all pixels in the DoG scale
            Detect interest point
            Create the descriptor
        end loop
        if more than 800 descriptors have been found then
            Terminate descriptor creation
    end loop
end loop
```

Figure 3: Descriptor creation pseudo-code.

paper we used a parameter of $2 - (0.87)^n$ where n is the scale of the Difference of Gaussians; further experiments are needed to better understand the effect of tweaking gamma and whether direct contrast enhancements are of interest.

For exact "inter-pixel" localization of keypoints, we used Brown's Taylor approximation method [4]. Afterwards we applied the basic edge and low contrast filters proposed in [15]. Two comments are in order here. First, the edge filter applied here only detects horizontal and vertical edges; after the descriptor is created we apply a much stronger edge filter which can handle any orientation of the edges. Second, for the contrast filtering we use a variable threshold parameter which decreases with decreasing octave size, as smaller octaves are likely to have lower contrast. Again, this is done to increase the number of high-level descriptors created.

3.3.2 Descriptor Creation

Once an interest point is found, we must now create a descriptor. As done in [15] we assign to each keypoint the dominant gradient orientation using the gradient magnitudes and orientations of the surrounding pixels. We have observed, however, that it is beneficial to reduce the influence of high gradient magnitudes for making the descriptors robust against brightness and contrast changes. This is achieved by using the fourth root of the gradient magnitude wherever the gradient magnitude was previously used.

For the encoding of the descriptors we use a 3×3 grid around the point, rotated with the dominating gradient orientation. Each of the cells contains 8 orientation buckets, resulting in $3 \times 3 \times 8 = 72$ dimensions in total. This structure is the same as for 72-dimensional SIFT descriptors. The radius of pixels considered for the descriptor computation, however, is calculated differently using the formula 6×1.2^n where n is the scale of the Difference of Gaussians. The motivation for this exponential increase of the radius is that the Gaussian blur is aggregating the visual contents of an increasingly larger area at higher scales. We compensate for this aggregation by increasing the area by 20% at each scale.

Since the extrema in the Difference of Gaussians of a keypoint can be either local maxima or local minima, and we wish to distinguish between the two, we multiply the resulting descriptor vector by -1 in the case of local minima.

The edge filtering proposed in [15] has the disadvantage that it only filters vertical and horizontal edges, but not slanted edges. We argue that all lines should be filtered out, as lines appear in very many images and have little information content (similar to common words in information retrieval). Fortunately, detecting lines in a 72 dimensional descriptor is easy, since the descriptor has already been rotated to match the strongest gradient orientation. This means that for straight lines the first orientation bucket in each grid cell has by far the highest value. By summing all these first buckets and comparing with the sum of the other 63 buckets, edges can be filtered out. Similar steps are taken to eliminate bright spots, such as spotlights and raindrops, as these also have little information content.

When comparing matching descriptors from a modified image to the original descriptors, we found that the linear interpolation used for assignment of gradients to orientation bins is not accurate enough. In order to reduce the error we perform a smoothing filter onto the eight orientations, where we assign to each bin the value $(\mathit{left} + 2 \times \mathit{current} + \mathit{right})/4$.

We then normalize the descriptor to the unit length. In order to reduce the effects of dimensions with high values, any value higher than 0.25 is reduced to 0.25. This is also used in [15], albeit to 0.20 due to the higher dimensionality of the descriptors. The descriptor is then normalized again to the unit length, and output.

3.3.3 Discussion

As we have pointed out, the Eff^2 descriptors borrow heavily from the SIFT methodology proposed by Lowe [15]. We have constructed them, however, with an aim for extreme scalability by restricting the number of descriptors, focusing the efforts on high-level descriptors and by studying the location of descriptors and adjusting parameters such that corresponding descriptors are likely to land in the same segment in PvS-indices. As our performance results show, our efforts have been rewarded, since in most cases they perform significantly better than the SIFT descriptors.

4. STOPPING RULES

In our experiments, we have observed that images that match a query image well typically receive tens or hundreds of votes. Two unrelated images, on the other hand, may have a few random descriptors in common among the hundreds of descriptors created, but these random votes are generally few and far between. Since the query processing time with the PvS-framework is solely dependent on the number of query descriptors processed, in this section we consider stopping as soon as the search is either confident that a match has been found or that no match is likely to be found with further processing.

Assuming that there is no matching image in the collection, any descriptor is equally likely to be returned as a neighbor to a specific query descriptor. Thus, the probability of any descriptor from a particular image matching the query descriptor is $p = k/n$, where k is the number of nearest neighbors returned and n is the number of images in the collection. The likelihood that this particular image gets more than x votes after evaluating m query descriptors then follows the binomial distribution $Bin_{m,p}(x)$. As a result, the probability that *any* image gets more than x votes by coincidence is given with the following formula:

$$P(max(X_1..X_n) > x) = (1 - Bin_{m,p}(x))^n \qquad (1)$$

By choosing a fixed probability value P for the likelihood $P(max(X_1..X_n) > x)$ we can, for a given m, calculate x_m, which is the number of votes that an image must have such that the odds of the image being a random image are equal to P. When considering intermediate (or final) results,

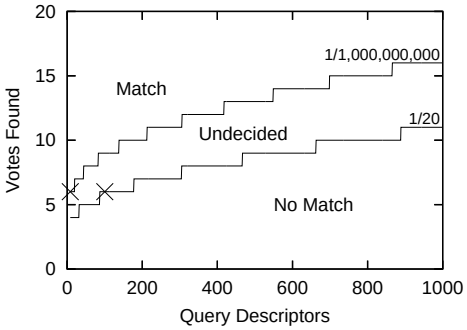

Figure 4: Visualization of the stop rules.

this method is used to determine which images are highly likely to be matches, which images are highly unlikely to be matches, and which images fall somewhere between the two.

Consider Figure 4, which demonstrates the processing of a fictional query image with 1,000 query descriptors against the collection used in most of our experiments. The figure shows two lines which divide the figure into three distinct parts. The upper line corresponds to a likelihood of $P = 1/1,000,000,000$ that an image is a random image; any image scoring above the line will be considered a match to the query image. The lower line corresponds to a likelihood of $P = 1/20$; any image scoring below this line will be considered not to be a match. Images scoring between the two lines are called "undecided".

To give an example of this classification, an image that has received its tenth vote after processing 100 query descriptors will be considered a match. If, however, the tenth vote is seen after processing 500 descriptors, it is considered undecided. Finally, if the tenth vote is seen only after processing 900 descriptors, the image is not considered a match.

Furthermore, Figure 4 has two "×" marks that are notable as they signal the application of two distinct stopping rules.

Stop Rule 1 (Matches Found): The first mark, on the upper line, indicates that this stop rule starts considering intermediate results when 8 query descriptors have been processed. After that, whenever a situation arises where (at least) one image is considered a match, and no other images are undecided (i.e., all other images are decidedly non-matches), query processing is terminated and the results returned.

Stop Rule 2 (No Matches Found): The second mark, on the lower line, indicates that this stop rule starts considering the possibility of not finding any matches when 100 query descriptors have been processed. After that, whenever a situation arises where all images are considered non-matches, query processing is terminated and the results returned, indicating a failure to find a match.

As long as any images are labeled undecided, further processing is required to determine whether they are a match or not. Note that if processing continues until all query descriptors have been considered, the images that pass the upper line are considered to match the query image; all other images are considered non-matches. We use this method, in fact, to determine the result quality in our experiments.

While applying the stop rules may obviously improve performance, it may also potentially affect the result quality. On one hand, Stop Rule 1 may generate false positives. This possibility arises when a random image generates, by chance, a number of its votes early in the search. As the preceding discussion points out, however, we take two precautions in order to avoid this possibility. First, we require a very low threshold of $P = 1/1,000,000,000$, which means that matching images are extremely likely to be at least partially similar to the query image. Second, we take care not to apply this rule until at least eight query descriptors have been processed, as before that time one or two random votes might trigger a false positive.

Stop Rule 2, on the other hand, may generate misses, when for some reason most of the votes are accumulated late in the query processing. As the description of Stop Rule 2 points out, however, we also take two precautions in order to avoid misses. First, we require only a threshold of $P = 1/20$ to make images undecided, which means that only images that have very low scores are ruled out entirely. Second, we take care to apply this rule conservatively, typically once 100 query descriptors have been processed.

To summarize, the two stop rules are used to determine when a) a matching image is most likely found, or b) no matching image is likely to be found at all. While neither of these conditions is determined, processing continues. The stop rules are first and foremost a performance issue, trading improved query performance for potentially weaker results. Both rules, however, have two parameters (the probability threshold and the starting point) that can be used to make them stronger or weaker as applications require.

5. EXPERIMENTAL SETUP

All experiments were run using the PvS-framework [13]. This section describes the images and descriptors used in our experiments, as well as the performance metrics used to analyze the results. Different workloads were used for different experiments; the workloads are therefore described along with the experimental results.

5.1 Images and Descriptors

We used two image collections consisting of high-quality news photos of varied content. The smaller collection contained 29,277 images, while the larger collection contained 316,545 images (including the smaller collection).

Each descriptor collection was produced by first resizing each image such that its larger edge was 512 pixels and then calculating the local descriptors. Table 1 shows the descriptor collections created for our experiments. As the table shows, we compared four different local descriptor schemes of various dimensionalities including our Eff^2 descriptors. The table furthermore shows that the SIFT and PCA-SIFT schemes create many more descriptors than the other two schemes, and that the creation of the Eff^2 descriptors is the most efficient, requiring only 1.5 seconds per image, compared to 2.8 seconds for SIFT descriptors. We did not measure the creation time of RDTQ descriptors.

We created three PvS-indices for each descriptor collection. For each query descriptor we searched for 30 nearest neighbors (as done in [1]), requiring a descriptor to be found in two of the three indices to be returned as a neighbor. In each case we set the segment size of the PvS-indices to 128 KB and created four-level PvS-indices (except for the large collection, which is described in Section 6.4). Since the segmentation of the PvS-indices is dependent on the num-

Collection	Scheme	Number of Dimensions	Number of Descriptors	Per Image	Time Per Image	Segmentation Strategy	Index Size
Small	Eff^2	72	22,983,895	785	1.5 sec.	[13,13,13,13]	3 × 3,7 GB
	SIFT	128	35,484,770	1,212	2.8 sec.	[15,15,15,13]	3 × 6,6 GB
	PCA-SIFT	36	35,484,770	1,212	15 sec.	[15,15,15,13]	3 × 6,6 GB
	RDTQ	24	20,506,800	700	-	[13,13,13,11]	3 × 3,2 GB
Large	Eff^2	72	208,264,284	661	-	[15,13,13,13,13]	3 × 56,1 GB

Table 1: The descriptor collection and index properties for each descriptor scheme.

ber of descriptors extracted from the image collection, we adopted the segmentation strategy accordingly, as shown in Table 1. The table also shows the size of the indices; creating each setup of three indices took 2 to 6 hours.

5.2 Performance Metrics

The experiments were run on DELL PowerEdge 1850 computers, each having two 3GHz Intel Pentium 4 processors, 2GB of DDR2-memory, 1MB CPU cache, and two (or more) 140GB 10Krpm SCSI disks. The machines were running Gentoo Linux (2.6.7 kernel) and the ReiserFS file system. We found that query processing performance depends solely on the number of query descriptors; our server is able to process more than 20 query descriptors per second. Although SIFT and PCA-SIFT are already at a disadvantage due to higher number of descriptors, we have chosen to focus instead on result quality in our analysis.

We have studied several metrics to measure the effectiveness of the descriptor schemes; the primary metrics used in our presentation follow. We use the method presented in Section 4 to determine the (possibly empty) result set for each query and then consider the top image from this set. For queries using image modifications, it is considered a *miss* when the original image is not the top image in the result set. When a miss occurs (or if there is no matching image in the collection) the query returns a *false positive* if any image is found in the result set. Both metrics should preferably be zero. Finally, when querying for modified images, we also studied the *descriptor ratio*, which is the percentage of query descriptors which yield a vote for the original image.

6. EXPERIMENTAL RESULTS

In this section we present the main results from our detailed performance experiments. In the first experiment, we study an image copyright violation workload, where all query images are modifications of existing images. In the second experiment, we study a workload of images that do not have a match in the collection. In the third experiment, we consider the effects of early stopping on both workloads. In the fourth and final experiment, we consider the effects of scaling the descriptor collection by a factor of ten.

6.1 Experiment 1: Matching Images

For this experiment we randomly chose 109 different "original" images from the small collection. For each of these images, we used the StirMark benchmarking tool [18], Version 4 to create 24 different modifications of these images. While StirMark generates over 100 different modifications, we chose to focus mostly on the most difficult ones (actually strengthening several of the distortions) in order to better test the descriptor schemes. Additionally we created two contrast modifications using the ImageMagick library. We thus created $109 \times 26 = 2{,}834$ different query images. Table 2 shows the complete listing of all image modifications.

Table 2 also shows the retrieval quality of the four descriptor schemes, using four-level PvS-indices, in terms of misses and false positives (empty cells indicate no misses or false positives). As the table clearly shows, our proposed Eff^2 descriptors perform better than the other schemes, finding almost 99% of all query images. They fail to retrieve one and two images from the difficult CONV 4 and COTR 2 modifications, respectively. The most difficult modification, the CONV 5 embossing filter, is found for 80 of the 109 query images; in one case is a false positive returned instead.

The SIFT and PCA-SIFT descriptors both miss a few images from four new modifications (CONV 2, COTR 1, MEDIAN 9, and NOISE 5), about 80% of the CONV 4 modification, one-third of the COTR 2 modification, and all of the embossed images. Overall, the PCA-SIFT scheme shows slightly fewer misses and false positives.

Finally, Table 2 shows that the RDTQ descriptors, which were used in [13], fail significantly for many of these modifications, producing both misses and false positives. Notable failures include the RESC 200 and RESC 75 modifications[3] and the JPEG 15 image compression.

To present the result quality in more detail, Figure 5 shows the descriptor ratio of the four descriptor schemes for all the image modifications. As explained above, the descriptor ratio is the percentage of query image descriptors that yield a vote for the original image. As the figure shows, the Eff^2 descriptors return the highest proportion of votes for nearly all image modifications, followed by the SIFT and PCA-SIFT schemes, and finally the RDTQ descriptors.

A few exceptions from Figure 5 are worth noting. For the PSNR modification all descriptor schemes return very good results; this modification is easily handled as it does not change the visual content of the images. For the CONV 1 and CROP 75 modifications, the Eff^2 descriptors perform the worst of the four schemes. In both cases, however, they return votes for more than 50% of all query descriptors.

The relatively poor results for the CROP 75 modification can be explained by the fact that the descriptor extraction process of the Eff^2 descriptor scheme stops when the maximum threshold of 800 descriptors is reached. Therefore, when an image includes many low-level details, some of those details are skipped and not represented in the database. In contrast, because of its limited size, a cropped image generally yields fewer high-level descriptors, which means that more of the fine-grained details yield a descriptor. These detailed descriptors, however, have no match in the database.

The relatively poor results for the CONV 1 modification (low brightness) can be explained similarly. Because many interest points disappear in this modification, the Eff^2 scheme creates more low-level descriptors for the query image than for the original image; these descriptors have no

[3]It should be noted that a related descriptor scheme, IDTQ, is designed to handle rescaling better. The improvement, however, is at the cost of poorer handling of rotations.

Modification	Description	Eff^2		SIFT		PCA-SIFT		RDTQ	
		Miss	False Pos.	Miss	False Pos.	Miss	False Pos.	Miss	False Pos.
AFFINE 1	Shear in X								
AFFINE 2	Shear in Y								
AFFINE 3	Shear in X and Y								
CONV 1	Low brightness								
CONV 2	High brightness			4	1	3	1	25	24
CONV 3	Sharpen							9	1
CONV 4	Strong sharpen	1		85	8	79	6	106	6
CONV 5	Emboss filter	29	1	109	11	109	2	107	6
COTR 1	High contrast (from ImageMagick)			3				53	1
COTR 2	Low contrast (from ImageMagick)	2		38		30		90	13
CROP 75	Crop 75% from center								
JPEG 15	15% quality							90	7
JPEG 80	80% quality								
MEDIAN 9	9x9 median filter			15		12		107	7
NOISE 5	Applied 5% noise			2		1		89	19
PSNR	Watermark removal								
RESC 200	Image scaled to 200%							107	22
RESC 75	Image scaled to 75%							100	3
ROT 10	10° rotation								
ROT 90	90° rotation								
ROTCROP 2	2° rotation and cropping								
ROTCROP 5	5° rotation and cropping								
ROTSCAL 2	ROTCROP 2 + scaling								
ROTSCAL 5	ROTCROP 5 + scaling								
SS 1	Change in color space							2	1
SS 2	Change in color space								
Total (across all modifications)		**32**	**1**	**256**	**20**	**234**	**9**	**885**	**110**

Table 2: A quality comparison of the descriptor schemes for different image modifications.

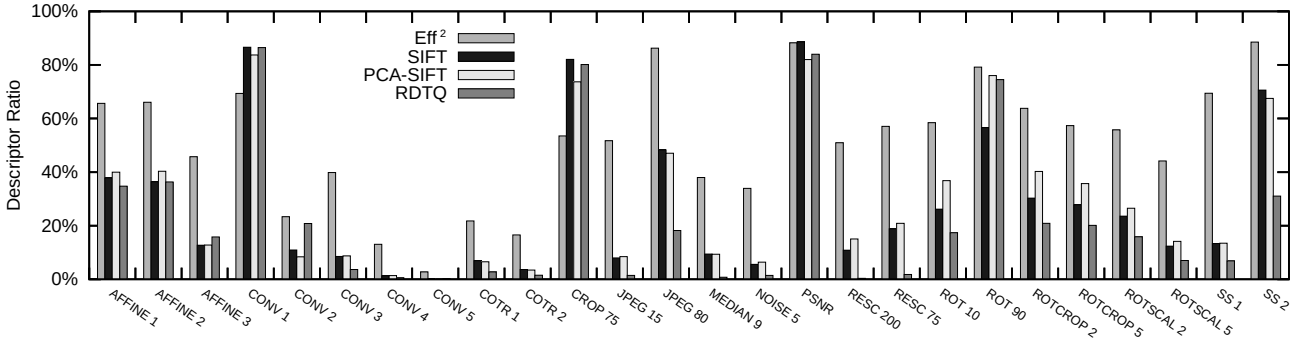

Figure 5: A comparison of the descriptor ratio of the four descriptor schemes for all image modifications.

match in the database. The SIFT scheme, on the other hand, stores many low-level descriptors in the database and thus receives a higher score for both CONV 1 and CROP 75.

Finally, Figure 5 shows that the RDTQ descriptors manage to beat the SIFT descriptors on a few modifications, but as already demonstrated in Table 2 they completely fail to retrieve original images for many other modifications.

6.2 Experiment 2: Non-Matching Images

For many applications, such as image copyright protection, it is important to have as few false positives as possible. For this reason, just over one thousand random query images were selected, that were not in the small collection.

All methods report a few false positives. Both the SIFT and PCA-SIFT descriptors return 12 false positives. In this workload, the RDTQ descriptors perform better, returning 8 false positives. The Eff^2 descriptors perform the best, as before, returning only 4 false positives, or for about 0.4% of all images. The primary reason for the poorer performance of the SIFT and PCA-SIFT schemes appears to be that these schemes produce many more descriptors, particularly low-level descriptors (recall that SIFT, e.g., had poor edge filtering), which generate more random matches.

Note that for a copyright protection application it is possible to post-process the potential matches (e.g., by examining the relative descriptor locations as done in [15]) to practically eliminate false positives; it is beneficial, however, to require such post-processing for as few images as possible.

6.3 Experiment 3: Stop Rules

In this experiment we study the effects of stopping the query processing, using the method presented in Section 4, as soon as it is likely that a match has been found.

We consider first the workload from Experiment 1, where each query image corresponds to an image in the collection. Figure 6 shows how many descriptors the Eff^2 and SIFT descriptor schemes must process before the match is found. (The PCA-SIFT scheme generally requires slightly

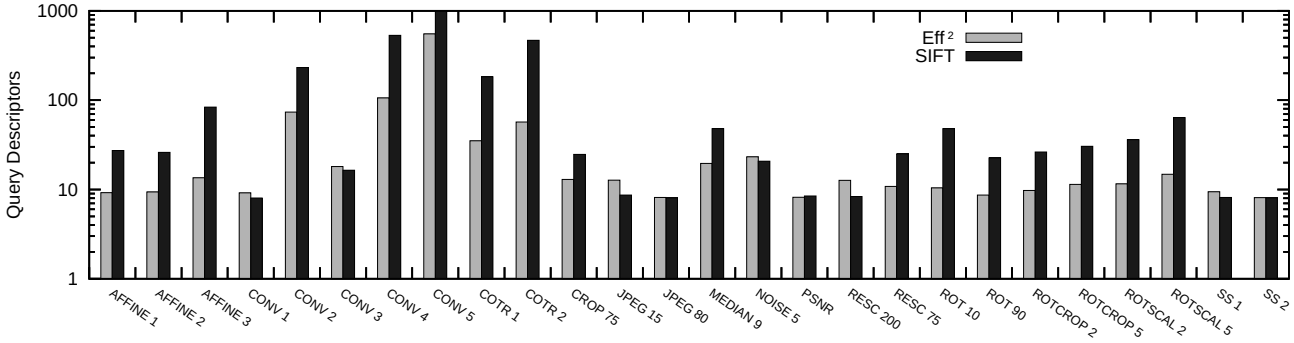

Figure 6: Comparison of performance of Eff^2 and SIFT descriptors with the stop rules.

Starting Point	CONV 2		CONV 4		CONV 5		COTR 1		COTR 2		Other		Total	
for Stop Rule 2	Eff^2	SIFT	Eff^2	SIFT	Eff^2	SIFT	Eff^2	SIFT	Eff^2	SIFT	Eff^2	SIFT	Eff^2	SIFT
200 query desc.	4	28	2	47	65	108	0	13	0	54	0	5	71	255
150 query desc.	5	38	2	47	75	109	0	22	1	63	0	5	83	284
100 query desc.	8	60	12	49	93	109	0	40	3	78	0	13	116	349
50 query desc.	17	82	27	51	103	109	5	72	12	95	0	76	164	485
30 query desc.	36	96	56	59	108	109	8	98	34	104	10	274	252	677

Table 3: Number of misses with Eff^2 and SIFT descriptors with varying starting point for Stop Rule 2.

more descriptors than the SIFT scheme, while the RDTQ descriptors perform poorly, even with full evaluation.) Note that the y-axis has a logarithmic scale, as many modifications require fewer than ten descriptors before the processing is stopped. Overall, about 70% fewer descriptors are required with the Eff^2 scheme than with the SIFT scheme. With the SIFT descriptors, five difficult modifications require processing of more than 100 descriptors; with the Eff^2 descriptors, only CONV 4 and CONV 5 require such extensive processing. Furthermore, with the Eff^2 descriptors, 19 modifications require fewer than 20 descriptors, and 8 modifications require fewer than 10 descriptors. Given that our modest server can search for more than 20 descriptors per second, these results mean that a match is usually found in less than a second for 19 out of 26 modifications.

As mentioned in Section 4, the number of misses is affected by Stop Rule 2. Table 3 shows the number of misses as the starting point for Stop Rule 2 is varied from the 200th query descriptor to the 30th query descriptor. Overall, the table shows that the Eff^2 descriptors perform much better than the SIFT desriptors, having only about one-third of the misses seen with the SIFT descriptors. Additionally, much fewer modifications have misses with the Eff^2 descriptors. Finally, the results show that when the rule is applied after the 100th query descriptor (after less than 5 seconds of processing in our system), only about 4% of all images are missed while SIFT misses more than 12% of all images. Recall, that in our experiments we include the most difficult image modifications of the StirMark benchmark, so these results are still of very high quality with this setting.

We now turn to the results for the workload from Experiment 2, where images have no match in the collection. When Stop Rule 2 is applied after 100 query descriptors have been processed, we see no change in the number of false positives for the Eff^2 and SIFT descriptors, which return 4 and 12 false positives, respectively, as before. Processing is terminated after 104 and 113 descriptors (on average) for the Eff^2 and SIFT descriptors, respectively.

6.4 Experiment 4: Scalability

In this final experiment we demonstrate the scalability of the Eff^2 descriptors with the PvS-framework, using the large collection which contains 316,545 images. With the Eff^2 descriptor scheme, these images yield in total 208,264,284 descriptors, or about 660 descriptors per image. We chose to use three five-level PvS-indices in an [15,13,13,13,13] configuration. A single index consumes about 56 GB of disk space and takes nearly 28 hours to construct. In total, therefore, the index creation took about three and a half day.

Consider first the workload of Experiment 1. As expected, the query time is unaffected by the size of the indices (it was actually reduced, as we needed to use two disks to fit all three indices). Figure 7 shows the descriptor ratio for the Eff^2 descriptors in both collections. As expected, fewer votes are returned with the larger collection, but in all cases the descriptor ratio is reduced by less than 20% of the query descriptors. The results are still of very high quality, as only about 90 images are missed (two-thirds from the CONV 5 modification), while 7 images are returned as false positives.

Turning to the workload of Experiment 2, which consisted of over 1,000 images that were not in the collection, we observed that only three images returned false positives.

7. CONCLUSIONS

In this paper we have addressed scalability issues of local image descriptor retrieval. We have proposed a new local descriptor scheme of the SIFT family and compared it, using the recently proposed PvS-framework, to three other well known descriptor schemes. Using a collection of almost thirty thousand images, we have shown that the new descriptor scheme yields the best results. We have proposed two stop rules which improve the retrieval performance significantly. Finally, we have tested these new descriptors using a collection of over three hundred thousand images and shown that even at such a large scale the descriptors still yield results of high quality, with no changes in response time. To our knowledge, this is by far the largest local image descrip-

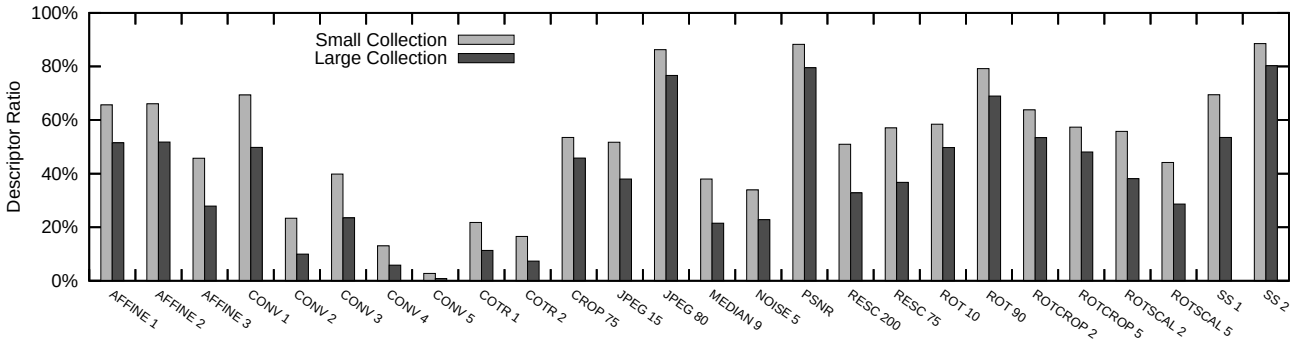

Figure 7: Comparison of descriptor ratio of Eff^2 descriptors with the small and large collections.

tor collection ever studied; these results demonstrate well the extreme scalability of both our Eff^2 descriptors and the PvS-framework. Using the PvS-framework, researchers in computer vision will be able to study the quality of their advanced image descriptor schemes at a very large scale.

There are several interesting avenues for our future work on scalability of local descriptors. First, more experimentation is needed to fully understand the effect of some parameters (such as the normalization of the DoG scales) on the Eff^2 descriptors. Second, we wish to analyse the performance of new descriptors, such as the fast approximate SIFT descriptors [8], which are also geared towards efficient descriptor creation; perhaps these techniques may be merged into our approach for even further performance improvements. Finally, with the stopping rules, a clever search algorithm could choose to process first those query descriptors whose segments are already stored in memory, opening opportunities for clever buffer management and prefetching strategies.

8. ACKNOWLEDGMENTS

This work was partly supported by INRIA *Eff*^{2} Associate Teams grant, Rannís Grant 60036021, and an ÉGIDE travel grant. We also wish to thank Utz Westermann for his help.

9. REFERENCES

[1] L. Amsaleg and P. Gros. Content-based retrieval using local descriptors: Problems and issues from a database perspective. *Pattern Analysis and Applications*, 4(2/3), 2001.

[2] S.-A. Berrani, L. Amsaleg, and P. Gros. Approximate searches: k-neighbors + precision. In *ACM CIKM*, 2003.

[3] S.-A. Berrani, L. Amsaleg, and P. Gros. Robust content-based image searches for copyright protection. In *ACM MMDB*, 2003.

[4] M. Brown and D. G. Lowe. Invariant features from interest point groups. In *British Machine Vision Conf.*, 2002.

[5] Y. Dufournaud, C. Schmid, and R. Horaud. Matching images with different resolutions. In *CVPR*, 2000.

[6] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *ACM SIGMOD*, 2003.

[7] L. M. J. Florack, B. M. ter Haar Romeny, J. J. Koenderink, and M. A. Viergever. General intensity transformation and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2), 1994.

[8] M. Grabner, H. Grabner, and H. Bischof. Fast approximated SIFT. In *ACCV*, 2006.

[9] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conf.*, 1988.

[10] A. Joly, C. Frélicot, and O. Buisson. Robust content-based video copy identification in a large reference database. In *CIVR*, 2003.

[11] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *CVPR*, 2004.

[12] Y. Ke, R. Sukthankar, and L. Huston. Efficient near-duplicate detection and sub-image retrieval. In *ACM Multimedia*, 2004.

[13] H. Lejsek, F. H. Ásmundsson, B. Th. Jónsson, and L. Amsaleg. Efficient and effective image copyright enforcement. In *BDA*, 2005.

[14] T. Lindeberg. Feature detection with automatic scale selection. *Int. Journal of Computer Vision*, 30(2), 1998.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2), 2004.

[16] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *CVPR*, 2003.

[17] K. S. Pedersen and M. Nielsen. The Hausdorff dimension and scale-space normalisation. In *Scale-Space*, 1999.

[18] F. A. P. Petitcolas et al. A public automated web-based evaluation service for watermarking schemes: StirMark benchmark. In *Electronic Imaging, Security and Watermarking of Multimedia Contents III*, 2001.

[19] A. P. Witkin. Scale-space filtering. In *IJCAI*, 1983.