Efficient Near-duplicate Detection and Sub-image Retrieval

Yan Ke, Rahul Sukthankar, Larry Huston

IRP-TR-04-07
August 2004

Proceedings of ACM International Conference on Multimedia (MM), 2004

Research at Intel

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications.
Intel may make changes to specifications and product descriptions at any time, without notice.

Efficient Near-duplicate Detection and Sub-image Retrieval

Yan Ke[†]
yke@cmu.edu

Rahul Sukthankar[‡†]
rahuls@cs.cmu.edu

Larry Huston[‡]
larry.huston@intel.com

[†]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
U.S.A.

[‡]Intel Research Pittsburgh
417 S. Craig Street Suite 300
Pittsburgh, PA 15213
U.S.A.

ABSTRACT

We introduce a system for near-duplicate detection and sub-image retrieval. Such a system is useful for finding copyright violations and detecting forged images. We define near-duplicates as images altered with common transformations such as changing contrast, saturation, scaling, cropping, framing, etc. Our system builds a parts-based representation of images using *distinctive local descriptors* which give high quality matches even under severe transformations. To cope with the large number of features extracted from the images, we employ *locality-sensitive hashing* to index the local descriptors. This allows us to make approximate similarity queries that only examine a small fraction of the database. Although locality-sensitive hashing has excellent theoretical performance properties, a standard implementation would still be unacceptably slow for this application. We show that, by optimizing layout and access to the index data on disk, we can efficiently query indices containing millions of keypoints. Our system achieves near-perfect accuracy (100% precision at 99.85% recall) on the tests presented in Meng *et al.* [16], and consistently strong results on our own, significantly more challenging experiments. Query times are interactive even for collections of thousands of images.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Image databases*; I.4.10 [**Computing Methodologies**]: Image Processing and Computer Vision—*Image Representation*

General Terms

Algorithms, Performance, Experimentation

Keywords

Sub-image retrieval, Near-duplicate image detection, Interest points, Local image descriptors, Locality-sensitive hashing (LSH)

1. INTRODUCTION

Near-duplicate image detection and sub-image retrieval is an important problem with several applications. Our system is motivated by two practical scenarios: finding (potentially modified) copyrighted images [1] and detecting forged images [6].

As more images are published on the Web, and as image manipulation software becomes more powerful and user-friendly, pirating images is becoming increasingly easy. Although digital watermarking techniques exist, these schemes are very difficult to design and there is an inherent trade-off between the robustness of the watermark and the amount of degradation induced in the image. To circumvent digital watermarking, the pirated images are often altered slightly — for instance, by cropping and rescaling. The problem of matching a slightly altered photograph to its original is termed *near-duplicate image detection*. A photo publishing agency could use a system like ours to automatically identify potential copyright violations and dispense with digital watermarks altogether.

A more insidious form of image manipulation, one that is becoming increasingly popular in propaganda, is the creation of fake photographs by cutting and pasting pieces extracted from different original sources. For instance, one could crop political figures from two different photographs and create a fake composite image showing them shaking hands, even though the individuals may never have met in reality [6]. Figure 1 shows an example where a girl's head from one painting was grafted into a scene from another painting. The problem of matching a small portion of one image to its original is termed *sub-image retrieval*. If the original images were stored in our system, we could detect query images that were composites and accurately identify the exact sources used in their creation.

We believe that practical systems that address the applications discussed above should satisfy the following requirements:

1. High recall. All images in the database that contain sub-images that are present in the query image should be found, even if the sub-images only occupy a small portion of the original.
2. High precision. If the database images and the query image do not have sub-images in common, then they should not be matched. This is important because incorrectly-flagged images will waste the user's time.
3. Efficiency. The time needed to query an image should be small, enabling the system to scale to very large databases.

Near-duplicate image detection and sub-image retrieval have both been studied extensively in recent years [3, 6, 12, 14, 16, 19]. However, previous work has typically cast the task into a traditional content-based image retrieval (CBIR) context, which tends to suffer from the following two problems. First, many techniques calcu-

MM'04, October 10-16, 2004, New York, New York, USA.
Copyright 2004 ACM 1-58113-893-8/04/0010 ...$5.00.

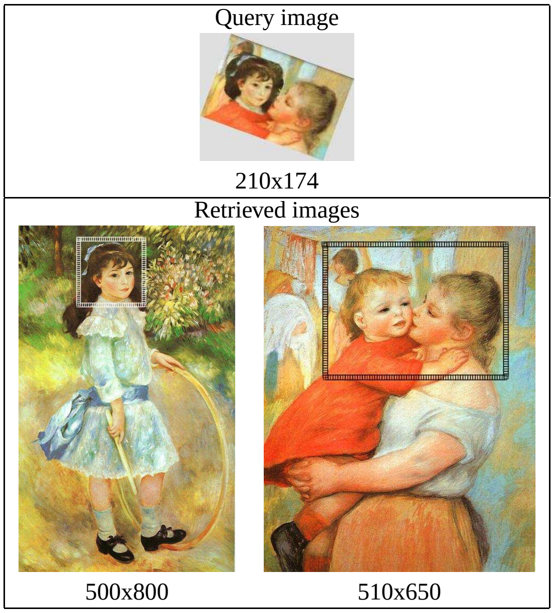


Figure 1: Example of sub-image retrieval applied to forgery detection. Given the top (forged) image as a query, our system correctly retrieves the bottom two (source) images, without any false matches, from a database containing 6100 images of similar paintings. Note that the query image contains only a very small portion from each source image, and that these portions have been cropped, resized and rotated.

late and store global statistics for each image, which is efficient but insufficiently accurate: recall can suffer when a significant transformation perturbs global statistics; and precision can be poor because global statistics, such as histograms (that are robust to geometric transforms), tend to generate many false positives. Second, those systems that compute local statistics of an image (e.g., by partitioning an image into smaller pieces) can suffer from low precision [14, 19]. This is because they typically concatenate all of an image's local statistics into a single feature vector describing the image. They must then relax the matching threshold in order to match small parts of the feature vectors, which makes the vectors less distinctive.

We argue that, instead of using a single feature vector to describe an entire image, one should identify and *independently index* a large number of local features, each of which is highly distinctive, while being robust to typical image transforms. Such an approach would selectively identify local features that match extremely well, rather than seeking loose partial matches between complicated global image features. Unlike existing techniques, such a scheme would be highly resistant to occlusions and cropping, both of which can destroy a significant fraction of the features. The main drawback of the proposed approach is that a single image could generate thousands of local features, and a single query would require the system to search for each of these features in a database containing millions or billions of features. Since features would not generate exact matches, each of the individual searches would become a similarity query in a very high-dimensional feature space. Consequently, such approaches have previously been dismissed as computationally impractical. However, this paper shows that these ideas can become the foundation for near-duplicate and sub-image retrieval system that is both extremely accurate and that scales well to large image collections.

The research that is most similar to ours is [1], where local features are extracted from images and matched using an approximate similarity search. Our system differs significantly in the following respects. First, we use scale- and rotation-invariant interest point detectors, more distinctive local descriptors, and perform geometric verification on the matched features. Second, instead of an *ad hoc* approximate similarity search, we employ locality-sensitive hashing [7], an algorithm with provable performance bounds. These contribute to the dramatic improvements in accuracy shown in Section 5.1. Third, we build offline indices that are optimized for disk access and search for all of the query local descriptors in a single pass. This enables us to query large image collections in interactive time.

The remainder of this paper is organized as follows. Section 2 reviews the relevant research for each of our components. Section 3 describes our system and gives implementation details. Section 4 details our evaluation metrics, experimental methodology, and the different datasets. Section 5 presents results of our system's retrieval accuracy and explores the impact of individual optimizations on execution time. Finally, we discuss some limitations of our work in Section 6 and conclude in Section 7.

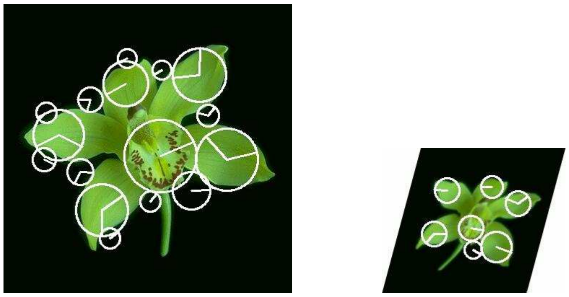
2. BACKGROUND

We first introduce the three building blocks of our system: distinctive interest points, locality-sensitive hashing, and efficient layout of data on disk. Using distinctive interest points allows us to achieve high recall and precision, while using locality-sensitive and efficient data layout gives our system interactive query times.

2.1 Distinctive Interest Points

Interest points [8, 13, 21] are commonly employed in a number of real-world applications such as object recognition [4] and image retrieval [17] because they can be computed efficiently, are resistant to partial occlusion, and are relatively insensitive to changes in viewpoint. There are three considerations to using interest points in these applications. First, the interest points should be localized in position and scale. Typically, interest points are placed at local peaks in a scale-space search, and filtered to preserve only those that are likely to remain stable over transformations. Second, the neighborhood surrounding each interest point should be modeled by a local descriptor. Ideally, this description should be distinctive (reliably differentiating one interest point from others), concise, and invariant over expected geometric and photometric transformations. Finally, the matching between local descriptors must be accurate and computationally efficient (discussed further in Section 2.2).

For interest point detection, we use Lowe's Difference of Gaussian [13] (DoG) detector because it has been shown to be robust and efficient. The DoG detector consists of three major stages: (1) scale-space peak selection; (2) interest point localization; (3) orientation assignment. In the first stage, potential interest points are identified by scanning the image over location and scale. This is implemented efficiently by constructing a Gaussian pyramid and searching for local peaks, termed *keypoints*, in a series of difference-of-Gaussian images. In the second stage, candidate keypoints are localized to sub-pixel and sub-scale accuracy, and eliminated if found to be unstable. The third stage identifies the dominant orientations for each keypoint based on its local image patch. The assigned orientation(s), scale and location for each keypoint enables us to construct a canonical view for the keypoint that is invariant to similarity transforms. For a 640x480 pixel image, we typically find hundreds to thousands of keypoints in the image, depending on the complexity of the image. Figure 2 shows the keypoints found in



(a) Original image (b) Rotated, scaled, and sheared

Figure 2: The keypoints located in this pair of images are shown as white circles, with lines denoting dominant orientations and radius denoting scale. Note that the keypoints are found at the same locations in each image, enabling us to accurately match the transformed image to the original. Note that the size and orientation of the keypoints reflects how the image was scaled, rotated and sheared. For illustration purposes, keypoints with a very small scale are not shown.

two images of a plant. One image is a rotated, scaled and sheared version of the other. Notice how the size and orientation of the keypoints are consistent with the applied transform. Although some of the smaller keypoints from Figure 2(a) are not detected in Figure 2(b), our system can still reliably match the larger keypoints.

For interest point representation, we use PCA-SIFT [11], a local descriptor that has been shown to be both more distinctive and compact than the original SIFT [13] descriptor. Given the location, size, and orientation of a keypoint, PCA-SIFT extracts a 41×41 pixel patch at the given scale and rotates it to a canonical orientation. The extracted patch covers an area in the original image proportional to the size of the keypoint. PCA-SIFT then generates its compact feature vector by computing the local gradient image of the patch, normalizing it, and projects it onto a precomputed eigenspace. As described in [11], this eigenspace is generated once (off-line) from a large number of keypoints extracted from images of natural scenes, and is not specific to our image collection. The top 36 components of the projected vector are used as the local descriptor.

The use of local descriptors has several characteristics that are ideal for solving the near-duplicate image detection problem. First, the interest points are scale and rotation invariant. This allows us to detect and match the same set of interest points even after images have been arbitrarily rotated or scaled. Our approach is also robust to deformations such as Gaussian blurring, median filtering, and the addition or removal of noise, which can degrade or destroy the high frequency content of the original image. This is because a subset of interest points in the original image will continue to match those interest points that encode lower frequency content in the transformed image (corresponding to larger image areas). Second, the descriptors are robust to image deformations such as affine warp, changes in brightness and contrast, etc. Furthermore, PCA-SIFT ignores color and operates on gray-scale images, making the algorithm immune to transforms that manipulate the color content of the image, such as saturation and colorization. Finally, because we use *local* descriptors, our system can find matches even if there is significant occlusion or cropping in the images. The system requires as few as *five* interest points (out of hundreds) to match between two images in terms of descriptor similarity and geometric constraints. Despite the small number of interest points needed to match, we maintain a low false positive rate because the local descriptors are highly distinctive and the geometric constraints further discard many false positives. In practice, the smallest sub-image we can reliably match between two images is approximately 100×100 pixels. This technique is also well suited to approximate similarity search algorithms, where one achieves a much faster query time at the cost of missed matches; note that although recall may suffer at the keypoint level, the overall recall of the system can continue to be very high because so few keypoint matches are needed.

Because of the large number of keypoints present in each image, it is cost prohibitive to do a linear search through the database for each query. Therefore, we employ an approximate similarity search that is well suited for high dimensional data.

2.2 Locality Sensitive Hashing

Locality-sensitive hashing (LSH), proposed by Indyk & Motwani [10], is an approximate similarity search technique that works efficiently even for high-dimensional data. Traditional data structures for similarity search suffer from the *curse of dimensionality*, in that they scale poorly for data with dimensions greater than 20, where they perform no better than an exhaustive linear search through the entire database. It has been shown that LSH out-performs tree-based structures such as the Sphere/Rectangle-tree (SR-tree) by at least an order of magnitude. Given that our data consists of many, high-dimensional (36-dimensional) feature vectors, LSH becomes an attractive indexing scheme.

Locality-sensitive hashing solves the following similarity search problem, termed (r, ε)-NN, in sub-linear time. If, for a point $\mathbf{q}$ (query) in d-dimensional space, there exists an indexed point $\mathbf{p}$ such that $d(\mathbf{p}, \mathbf{q}) \leq r$, then LSH will, with high probability, return an indexed point $\mathbf{p}'$ such that $d(\mathbf{p}', \mathbf{q}) \leq (1+\varepsilon)r$. If no indexed point lies within $(1+\varepsilon)r$ of q, then LSH will return nothing with high probability. This is accomplished using a set of special hash functions that satisfy the intuitive notion that the probability of a hash collision for two points be related to the similarity (distance) between the points. By using multiple such hash functions in parallel, LSH reduces the rate of false negatives.

A popular algorithm for LSH, introduced by Gionis *et al.* [7] conceptually transforms each point $\mathbf{p}$ into a binary vector by concatenating the unary representation of each (discretized) coordinate of $\mathbf{p}$. The resulting bit string is a point in a high-dimensional Hamming space, where L1 distances between points in the original space are preserved. Hash functions that simply select a subset of the bits that satisfy the desired locality-sensitive properties. The algorithm builds a set of l such hash functions, each of which selects k bits from the bit string (each function uses a different, randomly-selected set of k bits). These k bits are hashed once more to index into the buckets in our hash table, and a 32-bit checksum hash value is also generated. The two parameters, k and l enable the designer to select an appropriate trade-off between accuracy and running time. Our implementation of this algorithm is described in Section 3. In our experiments, we use k=450 and l=20, based on performance on a separate validation dataset. As seen in Section 5.2, our choice of (k, l) favors execution speed over similarity point recall; given that each image contains hundreds of keypoints, we are willing to risk missing a significant fraction of them in exchange for speed.

2.3 Efficient Disk Access

Locality-sensitive hashing was originally designed to work efficiently in memory, where random access is fast. For large datasets, one must store the database on disk, and a naive implementation of LSH fails miserably. This is because random access on disk is expensive, on the order of 10ms per seek. Multiple queries into a hash table, by definition, requires random seeks on disk. Our initial experiments revealed that querying our database for the keypoints from just one image took several minutes, indicating that the standard LSH implementation could never be practical for our problem.

The key difference between our system and other systems that use LSH for other applications is that all of our queries occur in batches of hundreds or thousands (corresponding to all of the keypoints in the query image). We extract the keypoints from the query image, and search on the entire set of keypoints to determine if any of them match the keypoints in the database. An earlier disk-based implementation of LSH by Gionis *et al.* [7] was designed for efficient *single point* queries rather than the batch queries required by our system. Since disk seek times are the bottleneck, our approach relies on organizing the batch queries so as to minimize the motion of the disk heads. We do this by precomputing all of the hash bins that we need to access, sorting them, and accessing them in sequential order. Reducing the disk head motion in this manner translates to a dramatic improvement in effective seek time — cutting it to approximately 1ms per seek. Gionis *et al.* also suggested inlining the data in the hash table instead of storing only the pointers as one would for an in-memory implementation. Their goal was to halve the number of seeks because one would not need to follow a pointer to the actual data. However, for our application, inlined data led to a massive increase in required disk space (20x for our dataset) and actually *slowed* our search. Since our searches do not require random seeks, we achieve better performance by employing a small hash table with an auxiliary keypoint database (and scanning both in-order) rather than a large hash table with inlined data.

All of these components are required to make our system practical. The use of robust interest point detectors and distinctive local descriptors enables us to query images with high recall and precision, as shown in section 5. By using locality-sensitive hashing and optimizing the data layout on disk, we achieve interactive response times for queries.

3. IMPLEMENTATION

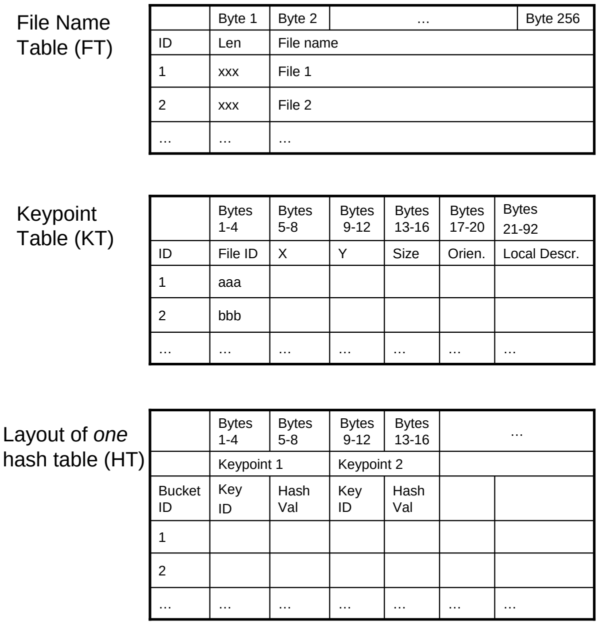
This section describes the implementation details of our system. Our algorithm consists of two stages. First, in the *index construction* phase, we process the image collection and index all of the extracted keypoints. Then, in the *database query* phase, the user can issue queries to find near-duplicates or to perform sub-image retrieval. These are summarized in Figure 4 and detailed below.

3.1 Index Construction

Given the collection of images to be indexed, we first use the SIFT DoG detector to locate all of the interest points. Then, we use PCA-SIFT to build local descriptors using a small image patch centered around each interest point. The source code for these two steps was downloaded from the web and the default parameters for each algorithm were employed.

We create three disk-based data structures, which are carefully laid sequentially on disk.[1] The data structures store a list of file names (FT), a list of keypoints from all the images (KT), and the locality-sensitive hash table of pointers to the keypoints (HTs). We construct the data structures, illustrated in Figure 3, as follows.

First, we create the file name table (FT) using a list of fixed-sized records on disk. Each record is 256 bytes in length, where the first byte denotes the length of the file name and the rest are used to store the string. Implicitly, the *id* of each file is its index location in the name table.



File Name Table (FT)

	Byte 1	Byte 2	…	Byte 256
ID	Len	File name		
1	xxx	File 1		
2	xxx	File 2		
…	…	…		

Keypoint Table (KT)

	Bytes 1-4	Bytes 5-8	Bytes 9-12	Bytes 13-16	Bytes 17-20	Bytes 21-92
ID	File ID	X	Y	Size	Orien.	Local Descr.
1	aaa					
2	bbb					
…	…	…	…	…	…	…

Layout of *one* hash table (HT)

	Bytes 1-4	Bytes 5-8	Bytes 9-12	Bytes 13-16	…	
	Keypoint 1		Keypoint 2			
Bucket ID	Key ID	Hash Val	Key ID	Hash Val		
1						
2						
…	…	…	…	…	…	…

Figure 3: Format of the disk-based data structures.

Similarly, we create the keypoint table (KT) using fixed-sized records. Each record stores one keypoint and consists of a file id (where the keypoint came from), its x and y location, orientation, scale, and its local descriptor. In total, each record is 92 bytes in length. Assuming that there are a thousand keypoints per image, it takes approximately 90MB to store the keypoints from one thousand images. Wherever possible, our implementation optimizes disk read access. For instance, given a list of keypoints that need to be read from disk, we first sort the list by keypoint id, thus ordering the disk reads to be efficient, and thereby reducing the average seek time.

Finally, we create the locality-sensitive hash tables (HTs). Recall that the LSH algorithm builds l independent hash tables, each with its own hash function. Below, we describe the layout of just one of these hash tables. All of the independent hash tables are concatenated and stored sequentially on disk. The hash tables are of fixed size, so the number of keypoints that we need to store must be determined before we create the hash tables. Each hash table consists of B buckets, where each bucket can store up to m keypoints. With a utilization value of α, we need $B = n/(\alpha m)$ buckets to store n keypoints. A higher α will lead to better space utilization, with an increased risk that some keypoints will not be indexed due to full buckets. A smaller bucket size m will lead to faster search times, but also a higher risk of dropped keypoints. For our experiments, we set $m = 20$ and $\alpha = 0.5$. Two items are stored per keypoint: the keypoint id and a checksum hash value that enables the system to avoid verifying every keypoint in the same bucket. Therefore, the system utilizes approximately 16MB of storage, per independent hash table, per million keypoints. One of our major system performance optimizations is that each hash table is created separately and entirely in memory before being written to disk. It would be impractical to create the hash table otherwise. This is feasible because we're only storing pointers to keypoints, and thus we can easily accommodate databases of 50 million keypoints (from 50 thousand images) in 1GB of main memory.

[1] In practice, it is difficult for a user program to control the data layout on disk; we start with a defragmented disk.

3.2 Database query

Once the index is created on disk, we can issue queries on new images using a parallel set of operations. First, we locate interest points in the query image and build local descriptors, as described above. Then, we calculate the bucket id's of each keypoint using the locality hash functions *without accessing the disk*. If we were to read data from the hash buckets (from HTs) as we hashed each keypoint, then this would be equivalent to doing random seeks on disk and would be unfeasibly slow. Instead, we sort the bucket id's and read the buckets in order, which corresponds to a linear seek on disk. We read all of the keypoints within a bucket and confirm that the checksum hash values match. All of the candidate keypoints are stored in a list sorted by keypoint id.

Finally, we read the keypoint data (location, orientation, size, and descriptor) from the keypoint table (KT) in order to generate a list of candidate matches for the query keypoints. Because LSH only returns approximate matches with respect to the L1 (Manhattan) norm, we need to check both for false positives and for points outside the threshold distance under the L2 (Euclidean) norm. We discard false matches by checking that the distance between the local descriptors of the query keypoint and the candidate keypoints is within the threshold distance under L2.

At this point, we look up the file id (in FT) corresponding to matched keypoints and separate them according to file id. The greater the number of matches found per file, the more likely it is that the image is a near-duplicate. However, it is still likely that there are significant false positives at the keypoint match phase. In other words, although some keypoints are within the threshold distance, they belong to patches of images that are not near-duplicates. We do affine geometric verification using RANSAC [5] to eliminate such outliers. The affine transformation between two images can be derived using three pairs of matched keypoints. RANSAC verifies whether the majority of the other matched keypoints support this transform and discards any outliers. The remaining pairs of matched keypoints correspond to the target image under an affine warp from the query image. The affine transformation includes rotation, scale, and shearing along the axes. The remaining set of images are discarded if fewer than θ matches are found, where θ is an adjustable parameter that controls the recall-precision of the system.

4. EVALUATION METHODOLOGY

We present a number of experiments to show the effectiveness of our system. For our initial experiments, we use the methodology from [16] so that our results can be compared. A direct comparison should ideally test both algorithms on the same dataset. Unfortunately, neither their source code nor their image database was available. Therefore, we have created a very similar dataset by using identical image transformations for our experiments, and compare our algorithm against Meng *et al.*'s published results [16].

The experiments employ a small set of query images (probes) and a much larger set of test images (gallery). The gallery is composed of transformed versions of the probe images, augmented by a large number of similar-looking random images that serve as distractors. We use the transformations described in [16] and build a gallery with the same fraction of distractors.

We also created a significantly more challenging gallery by applying difficult transformations to the probe images to gain additional insight into the performance of our algorithms. All of our datasets are detailed below, and they can be downloaded from `http://www.cs.cmu.edu/~yke/retrieval/`.

```
Index Construction

For each image in the gallery:
    Find keypoints using DoG detector
    Build PCA-SIFT local descriptors for each keypoint
Build and store file name table (FT)
Build and store keypoint table (KT)
For each of the $l$ hash tables (HTs):
    For each keypoint:
        Hash keypoint and store id in table (in memory)
    Store hash table (HT) on disk

Database Query

Find keypoints in query image using DoG detector
For each keypoint:
    Build its PCA-SIFT local descriptor
    Compute the $l$ LSH hashes for the descriptor
Sort hashes by bucket id, scan hash tables (HTs)
Sort returned keypoint ids and scan KT linearly
For each returned image:
    Determine best affine transform using RANSAC
    Discard if a valid transform was not found
Print matched file names by reading FT
```

Figure 4: Summary of our algorithm.

4.1 Evaluation Metrics

In order to evaluate our system's performance, we measure the recall and precision of our algorithm. Intuitively, we want to maximize the number of correct positives and minimize the number of false positives. *A correct positive* is defined as a match between a probe image and one of its transformed versions in the gallery. *Recall* and *precision* are defined as:

$$recall = \frac{\text{number of } \textit{correct-positives}}{\text{total number of positives}}$$

and

$$precision = \frac{\text{number of } \textit{correct-positives}}{\text{total number of matches (correct or false)}}.$$

4.2 Experimental setup

For our first experiment, our image database consists of 6261 images of fine art downloaded from an online art gallery [2]. We resize each image so that the size of its larger dimension is 512 pixels. We randomly select 150 images to be the probes, and use the rest are added to the gallery as distractors. Each probe is modified according to the following 40 transformations, and these 6000 near-duplicates are added to the gallery to create a dataset with 12,111 images.

Using the Difference of Gaussian interest point detector, we identify 13.6 million keypoints in the images, and apply PCA-SIFT to build local descriptors. There are 1100 keypoints per image on average, although some images generate as few as 67 keypoints and others as many as 3000, depending on the complexity of the image content. We use locality-sensitive hashing to index all of the keypoints with parameters k=450 and l=20. We define two PCA-SIFT descriptors as matching if their L2 distance is within 3000. The minimum match threshold θ is 5 for our experiments. These parameter values were empirically selected using tests on a small validation set. All of our experiments use a 3GHz Intel® Pentium® 4 machine with 1GB of memory running Linux 2.4. The algorithms are implemented in C++.

As discussed above, our image transforms are identical to those described in [16], and are implemented using ImageMagick [9]. These 40 transforms are described below. The number in brackets next to each operation denotes the number of near-duplicate images generated by that particular operation.[2]

1. *Colorizing [3]:* Tint the (a) red, (b) green, or (c) blue channels of the image by 10%.
2. *Changing contrast [2]:* (a) Increase or (b) decrease image contrast using default parameters.
3. *Cropping [4]:* Crop the image by (a) 5%, (b) 10%, (c) 20%, or (d) 30%, preserving the center region. Resize cropped image to original size.
4. *Despeckling [1]:* ImageMagick's despeckle operation.
5. *Downsampling [7]:* Downsample (without Gaussian blurring) the image so that its size is reduced by (a) 10%, (b) 20%, (c) 30%, (d) 40%, (e) 50%, (f) 70%, or (g) 90%.
6. *Changing format [1]:* Convert JPEG source image to GIF format. This compresses the color space to a palette of 256 colors.
7. *Framing [4]:* Add an outer frame to the image, where the size of the frame is 10% of the framed image. Four images are produced, each with a frame of a random color.
8. *Rotating [4]:* Rotate image by (a) 90°, (b) 180°, or (c) 270°.
9. *Scaling [6]:* Scale the size of image *up* by (a) 2, (b) 4, (c) 8 times; or *down* by (d) 4, (e) 4, (f) 8 times. The scaled image is resized to the original size.
10. *Changing saturation [5]:* Change image saturation amplitude by (a) 70%, (b) 80%, (c) 90%, (d) 110%, or (e) 120%.
11. *Changing intensity. [4]:* Change image intensity amplitude by (a) 80%, (b) 90%, (c) 110%, or (d) 120%.

Note that because we use rotation-, scale-, and illumination-invariant grayscale local descriptors, we are *inherently robust* to all of the above transforms (confirmed by results in Section 5). To examine where our system could fail, we constructed a second set of more difficult experiments with larger data sets. We randomly selected 15,582 photos from the MM270K [15] database and randomly chose 314 of them as query images. We applied the following more difficult transformations to both the art and MM270K databases:

12. *Cropping [3]:* Crop the image by (a) 50%, (b) 70%, or (c) 90%, preserving the center region. Resize cropped image to original size.
13. *Shearing [3]:* Apply an affine warp along the *x* axis by (a) 5°, (b) 10°, or (c) 15°.
14. *Changing intensity [2]:* Change the brightness of the image by (a) 50% or (b) 150%.
15. *Changing contrast [2]:* (a) Increase contrast by 3x, or (b) decrease contrast 3x.

Figure 5 shows a small subset of these transformations.

Our last series of experiments verifies that our system can apply sub-image retrieval to detect forged images. For the first test, we manually generated three fake images by carefully tracing and pasting images of people from six source photographs. Using the fake images as probes, our system correctly identifies all of the source images with no false positives. The second test used automatically-generated composite images as queries. Each composite was generated by drawing a pair of random images from a gallery of 1000 images, and the center 10% region from one image was selected and pasted on to the other. We created a probe set of 1000 images, and judged the system's response to be correct only if both of the source images for that query were correctly retrieved.

Table 1: Recall-Precision for standard transformations.

recall	precision
Baseline - select 40 random images	
0.3%	0.01%
Weighted Sampling Threshold method from [16]	
90%	67%
100%	6%
Our method on art database of 12,000 images	
99.85%	100%

Table 2: Recall-Precision for difficult transformations.

	recall	precision
Art database of 7,611 images		
	98.40%	99.86%
MM270K database of 18,722 images		
Original	96.78%	88.78%
Same scene removed	96.78%	96.12%

5. RESULTS

We now present our image retrieval accuracy results and the effect of our design choices on running time performance.

5.1 Retrieval results

Table 1 shows the results on the first experiment using the set of transformations from [16] on 12,111 images. We see that we perform extremely well on both the recall and precision. There are 150 query images, each with 40 near-duplicates in the database. Therefore, there are 6000 possible total correct matches. For $\theta = 5$, our system only fails to find 9/6000 near-duplicates and generates zero false positives. By comparison, a baseline strategy that randomly selected 40 images from the gallery to match each probe would have a recall of 0.3% and a precision of 0.01%. Meng *et al.*'s system achieves a recall of 90% with a precision of 67% (as shown by the ROC plots published in [16] on their dataset).

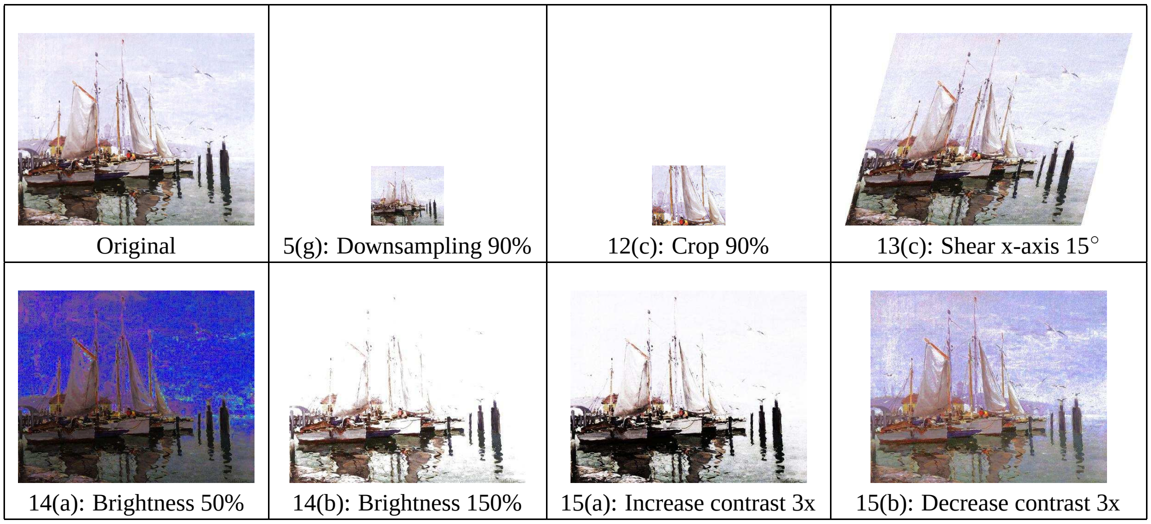
Table 2 shows the results for the second set of experiments. The accuracy remains very high, despite the more challenging transformations used to generate the near-duplicates. For the art database, we used 150 query images and for the MM270K database, we used 314 query images. Each query image has 10 near-duplicates in the database. Although the recall is high for both databases, the precision is significantly lower for the MM270K database. The drop in precision is due to the fact that this database contains several images that are slightly-different views of the same scene, taken from the same location, at the same time. A manual inspection of the false positives shows them to be panned or zoomed versions of others in this database, or photos of the same scene taken with a different color filter. If these photos of the same scene are not penalized as false positives, our system's precision remains above 95%.

Finally, Table 3 shows the results of querying the 1000 composite "forged" images to detect their sources. Again, our system does extremely well.

5.2 Running time results

While the previous section focused on the accuracy of our near-duplicate detector, we now turn to the system design and show that

[2] Our tests do not include the flipping transform; those are easily matched using mirrored versions of the probe images.


Original

5(g): Downsampling 90%

12(c): Crop 90%

13(c): Shear x-axis 15°

14(a): Brightness 50%

14(b): Brightness 150%

15(a): Increase contrast 3x

15(b): Decrease contrast 3x

Figure 5: Examples of automatically-generated near-duplicates. Only 7 out of 50 transforms are shown; all were correctly identified.

Table 3: Recall-Precision for composite images.

recall	precision
98.85%	99.65%

Table 4: Efficiency of LSH versus linear search.

	linear search	LSH
Running time in sec. (σ)	80.3 (0.06)	0.97 (0.04)
Pairs of keys checked	268 million	2656
Pairs of keys matched	5464	1611

each of our optimizations were necessary in order to make our approach practical. All of the following experiments were done using a small test set of 200 random images combined with 10 near-duplicates of the query image from the difficult test set, for a total of with 265,000 extracted keypoints. There are 1010 keys in the query image.

First, we compare the execution time of locality-sensitive hashing versus an exhaustive linear search through the keypoints, for the selected LSH parameters. Table 4 shows the performance of searching for matches of 1010 keys in the database. We give the running time and the number of keys checked by each. We see that LSH, due to the approximate nature of its search, misses a very large fraction (71%) of keys that were correctly matched by the exhaustive search. On the other hand, our execution speed is faster by two orders of magnitude. Despite the large number of keys missed by LSH, our system still performs well because we extract hundreds of interest points per image, while requiring as few as θ=5 matching keys to identify near-duplicates.

Recall that our implementation of LSH assumes L1 (Manhattan) distance in the analysis of near neighbors, while PCA-SIFT requires L2 (Euclidean) distance to be calculated during the actual matching of keypoints. Our algorithm makes the implicit assumption that using L1 in the LSH stage does not force us to examine and discard too many points during the PCA-SIFT matching stage. Our next experiment quantifies the inefficiency induced by this assumption. Table 5 shows that most of the keys (94%) that are checked but not matched under L2 are due to hash table collisions. The number of keys matched under L1 but not L2 only account for 2% of the total keys checked. Therefore, the L1 distance assumption is acceptable for our data.

Table 5: Inefficiency due to L1 assumption.

	No. of keys
Checked by LSH	2656
Matched under L1 ($d \leq 18000$)	1674
Matched under L2 ($d \leq 3000$)	1611
Checked because of hash table collision	982
Matched under L1 but not L2	63

Table 6: Importance of building hash table in memory.

	Running time in sec. (σ)
Build directly on disk	325 (1.8)
Build in memory, stream to disk	48 (0.1)

While the use of LSH gives us tremendous theoretical gains in performance, careful system design is required to realize LSH's benefits for our application. When creating the hash tables, we build the independent hash tables in memory, and then stream them sequentially to disk. Table 6 compares the running time of building the hash table in memory and directly on disk. We see that eliminating the random seeks to disk reduces the running time by a factor of 7.

For queries, we must also linearize the disk accesses to remove as many random seeks as possible. By sorting the hash bucket id's and keypoint id's before reading them from disk, we get a dramatic improvement in running time. Table 7 shows that this optimization results in a 20x speed-up.

Table 7: Importance of linearizing disk accesses in queries.

	Running time in sec. (σ)
Unoptimized disk reads	65 (1.8)
Sorted disk reads	3.4 (0.1)

It is only by combining all of these optimizations that we are able to create a practical near-duplicate and forgery detector that scales to large image databases.

6. LIMITATIONS

While our experiments show excellent results in retrieving near-duplicates and sub-images, there are limitations to our technique. Our system can match *similar* images of the same scene even if they were not near-duplicates nor forged images, as in our MM270K database. This scenario could occur if the copyrighted image database contains pictures of famous landmarks, where there are likely to be many pictures of the same landmarks on the web. Our system may find similar keypoints on the landmarks and *incorrectly* match them. Others have exploited this property as a feature to detect images of the same scene taken from different camera locations as in [17, 18, 20]. To informally test the performance of our system under such conditions, we gathered 59 pictures from the Web of *Big Ben*, *Eiffel Tower*, and *Half Dome*, and queried them against 18 professional pictures of the same landmarks. We were pleased to see only one match (i.e., false positive) among the 59 queries. The reason why we do not accidentally identify similar scenes as forged images more frequently is due to well-known limitations in applying current interest point detectors to recognize real-world objects. They include the instability of the DoG interest point detector on three dimensional objects and appearance changes due to self occlusion or shadowing. For example, during the course of the day, Big Ben's appearance changes significantly due to self-shadows. Similarly, Half Dome's appearance varies with seasonal vegetation. The sensitivity of keypoints to these appearance variations is seen as a limitation in the object recognition domain. However, for our application, this inability to generalize is a benefit. Since our goal is to detect perturbations of an image, we can set our system's appearance matching and geometric verification thresholds to much tighter bounds. This allows us to detect simple manipulations to copyright images while rejecting different views of the same scene.

7. CONCLUSION

The primary contributions of our paper are the synthesis of recent advances in robust interest point detection (DoG detector), local descriptor representation (PCA-SIFT), and efficient similarity search of high-dimensional data (LSH). By using a robust interest point detector and distinctive local descriptors, we accurately solve the near-duplicate and sub-image retrieval problem. Because we use a parts-based approach, our system is highly resistant to cropping, scaling, and other common transforms that traditional approaches based on global features can not cope with. A potential drawback in using a parts-based approach is that the system needs to query hundreds to thousands of features at a time, which could be slow. We make our system theoretically efficient through locality-sensitive hashing. Further, we make our system practical through careful data placement and batched disk accesses to minimize random seeks. We show experimentally that our system has *near perfect* accuracy (99.85% recall and 100% precision) on a standard test set. For future work, we plan to further optimize the data structures to gain additional query performance and further improve accuracy.

Acknowledgments

Yan Ke is supported by the Intel Research Scholar program. We thank M. Satyanarayanan and Derek Hoiem for valuable feedback.

8. REFERENCES

[1] S. Berrani, L. Amsaleg, and P. Gros. Robust content-based image searches for copyright protection. In *Proceedings of ACM Workshop on Multimedia Databases*, 2003.

[2] CGFA - A Virtual Art Museum. `http://cgfa.sunsite.dk/`.

[3] E. Chang, J. Wang, C. Li, and G. Wiederhold. RIME: A replicated image detector for the world-wide web. In *Proceedings of SPIE*, 1998.

[4] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, June 2003.

[5] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), June 1981.

[6] J. Fridrich, D. Soukal, and J. Lukas. Detection of copy-move forgery in digital images. In *Digital Forensic Research Workshop*, 2003.

[7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of International Conference on Very Large Databases*, 1999.

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

[9] ImageMagick. `http://www.imagemagick.org/`.

[10] P. Indyk and R. Motwani. Approximate nearest neighbor - towards removing the curse of dimensionality. In *Proceedings of Symposium on Theory of Computing*, 1998.

[11] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, 2004.

[12] A. Loui and M. Wood. A software system for automatic albuming of consumer pictures. In *Proceedings of ACM International Conference on Multimedia*, 1999.

[13] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[14] J. Luo and M. Nascimento. Content based sub-image retrieval via hierarchical tree matching. In *Proceedings of ACM Workshop on Multimedia Databases*, 2003.

[15] Media Graphics International. 270,000 Multimedia Graphics Pack, 1998.

[16] Y. Meng, E. Chang, and B. Li. Enhancing DPF for near-replica image recognition. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, 2003.

[17] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *Proceedings of International Conference on Computer Vision*, July 2001.

[18] F. Schaffalitzky and A. Zisserman. Automated location matching in movies. *Computer Vision and Image Understanding*, 2003.

[19] N. Sebe, M. Lew, and D. Huijsmans. Multi-scale sub-image search. In *Proceedings of ACM International Conference on Multimedia*, 1999.

[20] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of International Conference on Computer Vision*, Oct. 2003.

[21] L. Van Gool, T. Moons, and D. Ungureanu. Affine/photometric invariants for planar intensity patterns. In *Proceedings of European Conference on Computer Vision*, 1996.