Simple Representation and Approximate Search of Feature Vectors for Large-Scale Object Recognition

Koichi Kise, Kazuto Noguchi, and Masakazu Iwamura
Osaka Prefecture University, Japan
{kise,masa}@cs.osakafu-u.ac.jp, noguchi@m.cs.osakafu-u.ac.jp

Abstract

This paper presents two methods of large-scale recognition of planar objects with a simple representation and approximate search of local feature vectors. A central problem of the use of local feature vectors is the burden of computation and memory for finding nearest neighbors. To solve this problem, the proposed methods embody the following: (1) a simple bit representation of feature vectors and hashing enable us to fast access with less memory, (2) approximate search with query perturbation allows us to find approximate nearest neighbors efficiently. From large-scale experiments using 10,000 objects in the database and 2,000 query images, it was found that only 10–20% of correct nearest neighbors were enough for achieving recognition rate of 98.0%. The processing time for achieving this rate was 8.3 ms / query (excluding time for feature extraction). We have also tested the scalability of a proposed method using the database of 100,000 objects and obtained the result of 92.3% accuracy in 4.5 ms /query.

1 Introduction

Local feature vectors with high stability and discriminability such as SIFT [8] have been opening a new vista of robust and accurate object recognition. Extensive efforts have so far been made on, for example, recognition of planar objects [6, 9], 3D objects and their parts [4], object and scene categories [7], and structuring videos [10]. This paper is concerned with the application of PCA-SIFT [5] to the task of large-scale recognition of planar objects such as posters, book covers, and pictures of natural scenes, people, artifacts, etc. The objective of such recognition is to use the recognition technology for *tagging* planar objects in the real world: the user can deal with planar objects like *barcodes* for accessing their relevant information. Moreover the user is capable of establishing his/her own *links* that are clickable using mobile cameras.

One of the most prominent ways of the use of local feature vectors is often called *bag-of-words* or *bag-of-features* which represent images as a set of *visual words* obtained by clustering feature vectors [10]. When we scale up the number of objects to be recognized by the bag-of-features approach, at least the following problems need to be addressed. The first one is how to improve the discriminability of visual words. It is reported in the literature (e.g., [9]) that better performance is obtained by a larger number of visual

BMVC 2007 doi:10.5244/C.21.37

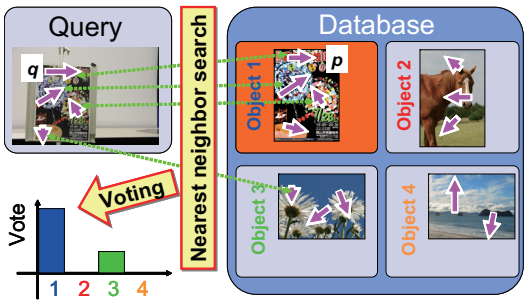


Figure 1: Object recognition by voting.

words, which increase the computational burden for indexing, storage and retrieval. Some researchers have attempted to solve this problem by applying a tree structure designed for dealing with local feature vectors [9]. The second problem is recomputation of visual words; if a number of new objects are added to the database, it would be necessary to recompute all visual words from scratch for better indexing. The last problem is about minority feature vectors. If some objects are mainly described by such minorities, it is difficult to recognize them correctly because few visual words appropriately represent such objects.

A simple way to solve the second and the third problems is just to take all feature vectors as visual words for indexing, which is an approach called memory-based or case-based vision. The input (query) image is recognized by matching its feature vectors to those in the database. In this scenario, *nearest neighbor search* of feature vectors is a fundamental processing. However this exacerbates the first problem. For the recognition of 100,000 objects, for example, the number of feature vectors is more than a hundred million and the number of times of matching with a simple sequential search exceeds 100 billion for recognizing a single query image; it is prohibitive to keep the recognition efficient. Thus the important problem here is how to deal with a huge number of feature vectors for storage and retrieval.

The main contribution of this paper is to solve this problem by (1) a simple bit vector representation of feature vectors and their hashing for efficient storage and retrieval, (2) an *approximate* nearest neighbor search with query perturbation for keeping the accuracy of nearest neighbor search enough for object recognition. For example, a proposed method is capable of recognizing planar objects using the database of 10,000 objects with the accuracy of 98.0% in 8.3 ms/query (excluding time for feature extraction). A different setting allows us 92.3% accuracy in 4.5 ms/query for the database of 100,000 objects.

2 Object Recognition by Voting

In this paper, we employ a simple scheme of object recognition as shown in Fig. 1. The task here is to find the corresponding image of an object from the database (DB) by matching the input image captured by a camera to each image in the DB. In this simple model of object recognition, both the camera-captured image, or *query image* and images in the database are described by a number of feature vectors produced by the local descriptor PCA-SIFT.

An outline of the recognition procedure is as follows. For each feature vector $\boldsymbol{q}$ of the query image, the corresponding feature vector $\boldsymbol{p}$ is found in the DB as the approximate nearest neighbor of $\boldsymbol{q}$. As a result, an object to which the vector $\boldsymbol{p}$ belongs obtains a vote from $\boldsymbol{q}$. Finally, the query image is recognized as the object with the largest number of votes.

It is easy to realize that, for such an object recognition scheme, the key for the memory efficiency is how to index and store feature vectors. Because the recognition process is dominated by finding nearest neighbors, nearest neighbor search is the key component to improve overall recognition efficiency. It is desirable to minimize both time and memory required to find nearest neighbors without losing the recognition accuracy.

3 Approximate Nearest Neighbor Search

An effective way of improving the efficiency of nearest neighbor search is to introduce "approximation" to the notion of "nearest neighbors". In this section we briefly describe two major methods: ANN (Approximate Nearest Neighbor) by Arya et al. [2], and LSH (Locality Sensitive Hashing) by Indyk et al. [3], both of which were utilized for comparison in Sect. 5.

ANN is a method of nearest neighbor search that employs a tree structure: each non-leaf node corresponds to a test that splits feature vectors into two disjoint sets, and each leaf node represents a feature vector. A node also corresponds to a cell, i.e., a region in the feature space. ANN traverses the tree structure to collect feature vectors for computing the distance to the query feature vector. The accuracy and the efficiency of computing nearest neighbors depend on the number of feature vectors for distance calculation. The approximation factor ϵ, which defines $(1+\epsilon)$-approximate nearest neighbors and changes the radius of search in the feature space, is to control the number. The smaller number of feature vectors is obtained with the larger ϵ.

LSH is a method of approximate nearest neighbor search using hash tables. We briefly describe an implementation of LSH called E^2LSH (Exact Euclidean LSH; simply called LSH for simplicity in this paper) [1]. To store a d-dimensional feature vector, LSH converts it into L different k dimensional integer vectors each of which is recorded in a distinct hash table. To find an approximate nearest neighbor of the query feature vector it is likewise converted into L integer vectors, and then they are utilized to find entries (feature vectors) in L hash tables. The distances from the query feature vector to each retrieved feature vector is calculated and the one with the minimum distance is returned as the result. With the same k, a larger L increases the probability of finding the exact nearest neighbor but also increases the computation time as well as the required memory.

4 Proposed Methods

4.1 Concepts

Both ANN and LSH are methods for finding the approximate nearest neighbor of a single vector. The simplest way of using these methods for object recognition is to apply them for each query feature vector to find its correspondence for vote. It is worth noting that our objective is not to find the best candidate for each feature vector, but to recognize

objects by voting. Thus as long as objects are correctly recognized, there should be no problem even with a certain amount of votes for wrong objects. This means that there is still some room of drastic approximation for improving the efficiency of recognition.

The most time-consuming part of object recognition by voting is the distance calculation with a large number of feature vectors. Thus the efficiency of recognition is improved by reducing the number. The most part of memory required for the recognition is occupied for recording a large number of feature vectors. Thus the memory requirement is eased by reducing the amount of memory for recording feature vectors.

In this paper we attempt to apply some simple methods for this purpose. A major drawback of the use of LSH for a large-scale object recognition is the memory consumption by multiple hash tables. The proposed methods employ only a single hash table to solve this problem. To compensate the lack of multiple hash tables, we introduce perturbation of query feature vectors that simulates errors caused by different imaging conditions. With this common strategy, we propose two methods for object recognition: the method with distance calculation (Method A) first retrieves feature vectors close to the query feature vector and then find the nearest one for voting by the distance calculation. The method without distance calculation (Method B) is just to skip the computation of the nearest feature vector of the method A; it simply votes for all feature vectors with a hope that the correct object accumulates the maximum votes. The details are explained in the following.

4.2 Indexing of Feature Vectors

In order to index feature vectors in the hash table, it is required to convert real-valued feature vectors into integers. A common solution is to apply the vector quantization. However, in order to keep our methods simple and efficient, we apply scalar quantization as follows. Let $\boldsymbol{p}$ be a n-dimensional feature vector $\boldsymbol{p} = (p_1, p_2, ..., p_n)$ obtained by applying PCA-SIFT. We binarize each dimension by

$$u_j = \begin{cases} 1 & \text{if } p_j \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

to produce a bit vector $\boldsymbol{u} = (u_1, ..., u_d)$ where the first $d(\leq n)$ elements are employed for the indexing. The threshold 0 is reasonable for values obtained by PCA-SIFT, because the their averages are around 0. Then the following hash function

$$H_{\text{index}} = \left(\sum_{i=1}^{d} u_i 2^{(i-1)} \right) \mod H_{\text{size}} \tag{1}$$

is applied to obtain the hash value of $\boldsymbol{p}$ and store it at the corresponding slot of the hash table.

For the method A with distance calculation it is necessary to record the original feature vectors. For the method B without distance calculation, on the other hand, it is unnecessary to keep original vectors. This greatly helps reducing the amount of memory.

Multiple vectors with the same hash value are stored by chaining. If the length of the chain exceeds the threshold c, all entries with the same hash value are deleted. This is the strategy called "stopword elimination". Although it is simple, it improves the memory consumption as well as the accuracy of recognition.

4.3 Method with Distance Calculation (Method A)

Next we describe the object recognition by using the method A. The elementary task is to retrieve a set of feature vectors $\{\boldsymbol{p}\}$ close to the query vector $\boldsymbol{q}$ from the hash table. Then the method finds the nearest neighbor $\boldsymbol{p}^*$ of $\boldsymbol{q}$ in $\{\boldsymbol{p}\}$ and votes for the object to which $\boldsymbol{p}^*$ belongs. The object in a query image is recognized by voting with all feature vectors extracted from it.

The most important step here is to find the set $\{\boldsymbol{p}\}$, which should include the nearest feature vector. A way is to access the hash table based on the hash value obtained from the query vector. In this case feature vectors having the same hash value are retrieved. Unfortunately it is unsatisfactory since variation of a query vector may change its bit vector, which results in accessing the slot of the hash table that does not contain its nearest neighbor.

In order to ease this problem, the query vector is expanded into several bit vectors to find its nearest neighbor. Because the query vector is transformed into the bit vector using the threshold of 0 for each dimension, the query vector having values close to 0 at some dimensions can be converted into the bit vector different from its nearest neighbor. We simply utilize the error range e as a parameter for generating different bit vectors. For the dimension q_j satisfying $|q_j| \leq e$ in the query vector $\boldsymbol{q} = (q_1, ..., q_d)$, the other bit value $u'_j = (u_j + 1)$ mod2 (if $u_j = 0$ then 1, otherwise 0) is also utilized to generate bit vectors. This means that the error of the value is estimated within the range defined by e. For the query vector $\boldsymbol{q} = (-10, 100, 2)$ and $e = 5$, for example, two bit vectors (0,1,1) and (0,1,0) is employed for the access of the hash table.

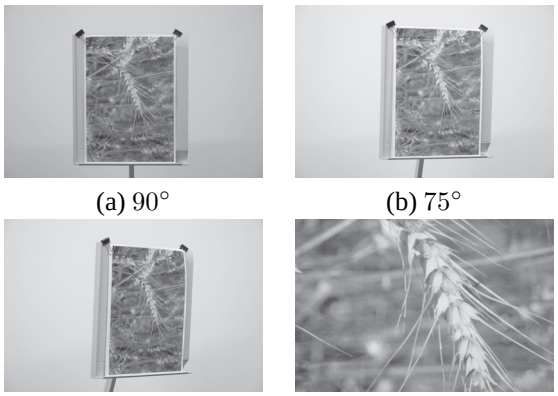
Unlimited application of this strategy increases the expanded bit vectors exponentially. In order to avoid this problem we utilize the limit b of the number of dimensions for the application. The method applies the strategy for q_j from $j = 1$, i.e., the dimension with the largest eigenvalue. If the number of dimensions that satisfy the threshold e exceeds the limit b, the application stops and the remaining dimensions are simply utilized as they are. In other words, the number of expanded bit vectors for a query vector is at most 2^b. For the above example of $\boldsymbol{q}$ with $e = 20$ and $b = 1$, two bit vectors (0,1,1) and (1,1,1) are obtained.

4.4 Method without Distance Calculation (Method B)

The method B shares most of the processing with the method A. Thus the parameters of the processing are also c, e, b and d. The difference is as follows. The method A computes the distance from the vector $\boldsymbol{q}$ to each element of the set $\{\boldsymbol{p}\}$ to find the nearest neighbor of $\boldsymbol{q}$ in the set. The method B, on the other hand, skips this calculation and simply votes for all vectors in the set $\{\boldsymbol{p}\}$. Because there is no need to calculate the distance, it is not necessary to record the original feature vectors. This enables us to reduce drastically the amount of memory needed for the processing.

(a) (b) (c)

Figure 2: Examples of images in the database



(a) 90° (b) 75°

(c) 60° (d) Part

Figure 3: Examples of query images. (a) – (c) covers the whole area, and (d) covers only 1/4 of the whole area. (a) and (d) are frontal view (90 °), while (b) and (c) are cross shots.

5 Experiments

5.1 Experimental Settings

The objective of experiments is to clarify the following relations: (1) relation between the accuracy of nearest neighbor search and that of object recognition, (2) relation between the accuracy of object recognition and the processing time, (3) relation between the number of objects to be recognized and the amount of memory.

As the images in the database, we have prepared in total 100,000 images collected using Google image search (Fig. 2(a)) and Flickr (Fig. 2(b)), as well as images available at the site of PCA-SIFT [1] such as Fig. 2(c). Images were resized to have their longest side less than 640 pixels. The average number of feature vectors extracted from an image was about 2,000.

Images used as queries were prepared as follows. We first selected at random 500 images from the database. Then these images were printed out onto A4 paper and converted into images by taking their pictures in four different ways as shown in Fig. 3. The size of the images were reduced to 512×341 pixels and then PCA-SIFT was applied to extract feature vectors. About 600 feature vectors were obtained on an average from an image.

[1] `http://www.cs.cmu.edu/~yke/pcasift/`

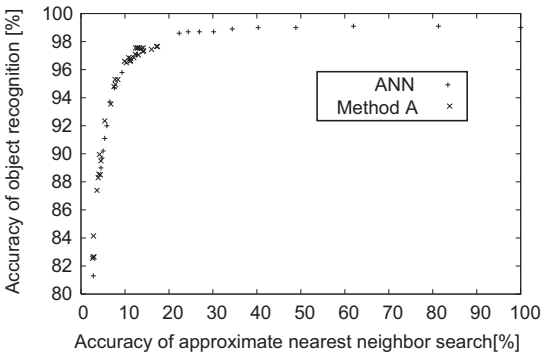


Figure 4: Relation between the accuracy of nearest neighbor search and that of object recognition.

The methods A and B were compared with methods of object recognition using ANN and LSH as engines of approximate nearest neighbor search [2]. Parameters for each method were set as follows. The number of dimensions of feature vectors for ANN and LSH was set to 36. For ANN, the approximation factor ϵ was changed from 2 to 1000. Although it may seem extremely large, we show that it is meaningful at least up to $\epsilon = 10$. For other parameters, the default values were employed. It means, for example, that we employed standard search with kd-trees. LSH has two major parameters, the number of hash tables L and the number of dimensions k for indexing feature vectors. We tested it by combining the values $L = 1, 3, 6, 15, 28$ and $k = 20, 24$. For the methods A and B, we have four parameters, i.e., the number of dimensions m for indexing feature vectors, the number of collisions c for deleting feature vectors, the error range e of a feature value, and the limit b of the number of dimensions for the expansion of the original bit vector. The tested ranges were as follows. $m = 24, 26, 28$, $c = 1, ..., 100$ and ∞, $e = 200, 500, 1000$, and $b = 0, ..., 15$.

In the following results, computation time excludes the time required for extracting feature vectors. We employed a computer with AMD Opteron 2.8GHz CPU and 16GB RAM.

5.2 Accuracy of Approximate Nearest Neighbor Search

For measuring the relation between the accuracy of nearest neighbor search and that of object recognition, we applied methods A and that with ANN using the database including 10,000 images. The accuracy of nearest neighbor search was measured as the ratio of the number of *exact* nearest neighbors to the total number of feature vectors.

Figure 4 shows the results where points were obtained by changing the parameters. We can see the consistency of the results obtained by the two methods. It is also shown that if the accuracy of nearest neighbor search is more than 20%, there is a little impact on the accuracy of object recognition. In addition, only 5% accuracy of nearest neighbor search is needed to achieve the object recognition rate of 90%. This means that we have a large margin of approximation for reducing the computational cost.

[2]The implementations utilized in the experiments were provided by authors of these engines: ANN at `http://www.cs.umd.edu/~mount/ANN/` and LSH at `http://www.mit.edu/~andoni/`

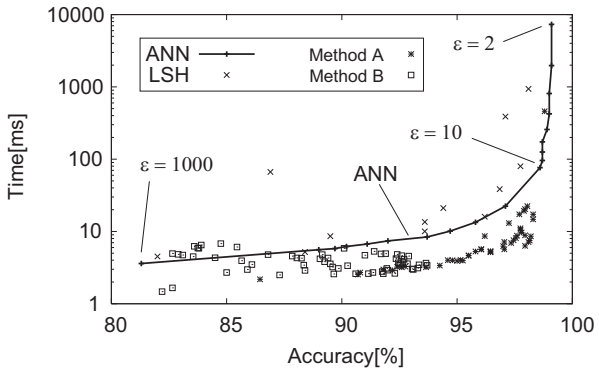


Figure 5: Accuracy of recognition and time for the database of 10,000 images.

5.3 Accuracy of Recognition and Time

Next we evaluated the accuracy of recognition and the time required for achieving the accuracy. Figure 5 illustrates the results. In this graph each point represents a result of a method with a set of parameter values.

The results by ANN are shown with the solid line: the upper right end indicates the result with $\epsilon = 2$ (accuracy:99.1%, time:7.3×10^3 ms) and the other end corresponds to the result with $\epsilon = 1000$ (81.3%, 3.6 ms). Because the accuracy 98.6% in 76.3 ms was obtained with $\epsilon = 10$, ANN was successful to cut down the computation time up to this value. With $\epsilon > 10$, the computation time can be further cut down but it comes with a price of lowering the accuracy.

Next, taking the results by ANN as the baseline, let us discuss results by other methods Most of the results obtained by LSH were inferior to those by ANN. This was mainly due to the cost of using multiple hash tables. The method A, on the other hand, was capable of achieving the processing about three times faster than the method with ANN. Unless the user is interested in results with very high accuracy, the method A is a good choice. For example, 98.0% accuracy was obtained in 8.3 ms/query. If the user is interested mainly in fast recognition while there is no problem with the accuracy around 90%, the method B can be recommended: e.g., 93.1% was in 1.75 ms/query.

5.4 Effect of the Number of Objects

We also tested the effect of the number of objects to the accuracy, the computation time and the memory consumption. The memory consumption was almost proportional to the number of objects in the database. For the recognition with ANN, 16GB RAM ran out for the database of 20,000 objects. The situation is better for the method A because of the use of deletion, and worse for LSH due to the use of multiple hash tables. The best was the method B that utilizes no original feature vectors. The amount of memory for the recognition of 100,000 objects was 7GB.

Time and accuracy using the method B are shown in Fig. 6. As shown in this figure, the time was sub-linear to the number of objects in the database. It is also shown that the accuracy almost stayed unchanged. These results indicate the scalability of the method B.

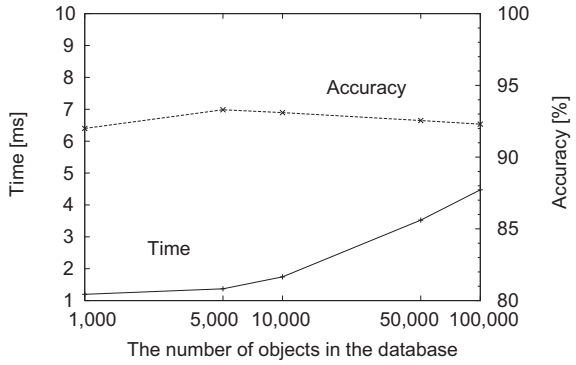


Figure 6: Relation among the number of images in the database, the accuracy of recognition and the processing time, all of which were measured using the method B with the parameters $e = 200$, $d = 28$, $b = 5$ and $c = 5$.

6 Related Work

We focus here on methods for large-scale recognition of planar objects. Ke et al. have proposed a method of sub-image retrieval for near-duplicate detection taking LSH as an engine for approximate nearest neighbor search. The database they employed for experiments included about 20,000 images. Although the size of the database is large, the task was much easier than ours. Their task of retrieval takes as queries automatically generated near-duplicates, while our task is with camera-captured images as queries. For our setting, it is required to cope with much wider variations on images.

Closer tasks are found in the paper by Nister et al. [9], which describes the vocabulary tree for indexing and retrieval of feature vectors. The computation time was reported as 25ms for a database of 50,000 images. They also evaluated their method using 6376 images and reported that the best result was obtained with 10M visual words. This seems almost the size to keep all feature vectors as distinct visual words. In such a case the tree structure is only to find approximate nearest neighbors efficiently. Another database was with 1 million images consisting of all frames from several movies. Although the size was extremely huge, the number of visual words should be much less because most of a sequence of images contains same objects under equal lighting conditions.

As compared to these methods as well as methods with ANN and LSH, the proposed methods can be characterized as follows. A simple representation by the bit vectors and the hashing mechanism improve the efficiency of storage and retrieval. Query perturbation enables us to keep the accuracy of recognition with such simple representation. The number of images used in the experiments and the number of feature vectors employed for the recognition were equivalent or even more than those employed for the above methods.

7 Conclusion

In this paper, we have proposed two methods for the use of individual feature vectors as indices of objects to be recognized. A simple bit vector representation of feature vec-

tors and search methods for their approximate nearest neighbors with query perturbation enable us to keep the processing efficient. From the experimental results using up to 100,000 objects it has been shown that the proposed methods are superior to recognition using ANN and LSH. The future work includes further improvement of accuracy and time based on the combination of nearest neighbor searchers. Recognition of 3D objects is another task to be pursued.

Acknowledgment

This research was supported in part by the Grant-in-Aid for Scientific Research (B) (19300062) from Japan Society for Promotion of Science.

References

[1] Alexandr Andoni, Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab Mirrokni. Locality-sensitive hashing using stable distributions. In Gregory Shakhnarovich, Trevor Darrel, and Piotr Indyk, editors, *Nearest-Neighbor Methods in Learning and Vision*, pages 61–72. The MIT Press, 2005.

[2] Sunil Arya, David M. Mount, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45(6):891–923, 1998.

[3] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of the 20th annual symposium on Computational Geometry*, pages 253–262, 2004.

[4] Boris Epshtein and Shimon Ullman. Identifying semantically equivalent object fragments. In *Proc. CVPR2005*, pages 2–9, 2005.

[5] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR2004*, volume 2, pages 506–513, 2004.

[6] Y. Ke, R. Sukthankar, and L. Huston. Efficient near-duplicate detection and sub-image retrieval. In *MM2004*, pages 869–876, 2004.

[7] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bag of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. CVPR2006*, pages 2169–2178, 2006.

[8] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[9] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *Proc. CVPR2006*, pages 775–781, 2006.

[10] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proc. ICCV2003*, volume 2, pages 1470–1477, 2003.