Scaling Object Recognition: Benchmark of Current State of the Art Techniques

Mohamed Aly[1] *Peter Welinder*[1] *Mario Munich*[2] *Pietro Perona*[1]
[1] Computational Vision Group, Caltech
Pasadena, CA 91106
vision.caltech.edu
{malaa, welinder, perona}@caltech.edu

[2] Evolution Robotics
Pasadena, CA 91106
www.evolution.com
mario@evolution.com

Abstract

Scaling from hundreds to millions of objects is the next challenge in visual recognition. We investigate and benchmark the scalability properties (memory requirements, runtime, recognition performance) of the state-of-the-art object recognition techniques: the forest of k-d trees, the locality sensitive hashing (LSH) method, and the approximate clustering procedure with the tf-idf inverted index. The characterization of the images was performed with SIFT features. We conduct experiments on two new datasets of more than 100,000 images each, and quantify the performance using artificial and natural deformations. We analyze the results and point out the pitfalls of each of the compared methodologies suggesting potential new research avenues for the field.

1. Introduction

Techniques for visual recognition of individual objects have come of age in the past few years bringing to life commercial deployments of high performance, reliable recognition systems that are able to recognize hundreds, sometimes thousands, of objects. Since the seminal work on SIFT features by David Lowe [11], there has been much research activity in the area of feature detectors and descriptors with embedded invariance to image transformations. Systems based on SIFT have been already commercialized [13] in products that successfully use object recognition in real life conditions. And most recently, a number of companies have started using object recognition in mobile search, see [bzhjkd][1].

If one wished, say, to build a system to recognize any building of interest to tourists around the world or any CD cover ever published, one would have to deal with at least millions of images. Therefore, it is easy to conceive of applications that require recognition of 10^6 or 10^8 individual objects. Some researchers have already acknowledged the fact that current techniques can only handle up to few thousand objects. The challenge is then to scale up these techinques by three to five orders of magnitude. Kd-trees are used canonically to search for approximate nearest neighbors [12]. Indyk [7] approximates nearest neighbor search with a set of efficient hash functions. Nistér and Stewénius [14] propose hierarchical clustering of descriptors to produce a vocabulary which is scored with the vector space model. Philbin *et al*. [15] improve on the this method by replacing the hierarchical vocabulary with a flat k-means clustering. These different methods all attempt to tackle scalability by trading off memory usage and time complexity. Some approaches propose clustering vocabularies to save memory by not having to store all descriptors. Others propose simpler matching models, such as the vector space model, to reduce complexity.

We are interested in exploring the computational cost and the precision-recall performance of these methods as the database size increases. We explore why, and with what tradeoffs, these performance decreases are manifested under different recognition models. We have three contributions: (a) we collected two large databases of images for testing scaling of recognition algorithms, we will make this benchmark public, (b) we developed a number of performance metrics that analyze both end-to-end performance and module performance of recognition algorithms, (c) we measured compared the performance of the most promising current approaches to scalable recognition, using off-the-shelves implementations when possible.

2. Related Work

Scaling object recogntion to millions of objects has a dual in the problem of specific object retrieval in large image collections. Indeed, many techniques developed to solve one of the problems may be applied successfully to the other. This section summarizes the latest research in these fields.

Many effective object retrieval approaches are based on

[1] We use tinyurl.com throughout the paper, thus [bzhjkd] means http://tinyurl.com/bzhjkd

2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops
978-1-4244-4441-0/09/$25.00 ©2009 IEEE

techniques developed in the textual information retrieval community. The idea was introduced by Sivic *et al*. [18], and later improved on by [14, 17, 15, 3, 16]. In particular, the importance of large vocabularies of visual words, the analog of textual words, for retrieval performance was studied by [14]. When dealing with vocabularies of 1 million words, exact clustering methods such as K-means become infeasible, as their computational cost is $O(N \cdot K)$ for K words, where N is the size of the training set. Philbin *et al*. [15] improved upon the hierarchical K-means scheme used by Nistér and Stewénius [14] by using a Kd-tree based clustering algortihm. Vocabularies built taking into account the information gain of features in the vocabulary construction perform better [17]. Further gains in classification rates can be achieved by considering contextual distance metrics, proposed by [9], which adapts the metric for comparing bag-of-words vectors to better discriminate between database images. The problem with the latter method is that it requires pre-computing the nearest neighbors to all images in the database, which takes $O(N^2)$ for large databases. Recent work [9, 16] has shown that assigning many words to a visual feature, so-called soft word assignment, can further improve the retrieval performance.

A related class of techniques uses hashing to group similar images in large image databases [4, 19]. Here the whole image is represented by a single short descriptor, similar to the bag-of-words representation. The hashing techniques then become an alternative for finding nearest neighbors in the image database. Although still in the early stages, hashing-based algorithms might be very important for large databases recognition given their $O(N^c)$, $c < 1$ search time, as current methods such as the popular vector space search have $O(N)$ search time.

As shown in this section, there is a deep connection between object recogntion and object retrieval in image collections. However, it is important to reiterate that the emphasis of this paper is to benchmark the best techniques developed for both problems specifically on the object recognition task with large databases.

3. Datasets

We used three testing scenarios, each consisting of a *model* set and a *probe* set. Each model set consists of a large number of objects, any of which can be recognized. Each probe set consists of a smaller set of additional images of objects in the model set, which are used to index into the model set. Probe sets are used to benchmark recognition performance. We collected two *model* sets and three *probe* sets.

3.1. CD/DVD Covers Dataset

The first model set is a set of 197,311 medium resolution (640×480) CD/DVD covers from [kgaxg]. The set included 1,979 covers of software packages, 11,444 games covers, 102,353 movies and TV shows covers, and 81,535 music records covers. In this dataset is that there are duplicates and highly similar images, e.g. covers of the same game on different console or different language versions of the same movie, see Fig 1. We pruned the duplicates from the dataset to avoid having competing matches when trying to recognize an input probe image. Duplicates were eliminated (see below) yielding the final dataset of 132,380 unique images.

Fig. 2 explains the database cleaning algorithm. We used SIFT [11] features[2], and a set of 4 randomized Kd-trees [10] as the dictionary. The thresholds were determined empirically for the dataset, and were set to $T_m = T_a = 100$ features, where T_m is the minimum number of matching features an image must have to be further considered and T_a is the maximum number of matching features an image must have to be considered unique. The dictionary keeps only a set of $T_d = 50$ unique images, making use of the fact that the images were in alphabetical order and that we only need to match images to the previous images already in the dictionary. If new images are matched to one in the dictionary, then it is marked as similar, otherwise it is considered a unique image and is added to the list of unique images. Figure 1 shows examples of similar images found by the pruning algorithm.

We used two probe sets with the model set above. The first probe set was obtained by applying 5 synthetic transformations to 100 randomly chosen images from the model set, which gives a set of 500 probe images. Having the ground truth transformations allows us to quantify the performance of the nearest neighbor algorithms, see fig. 8. The transformations are: 1) identity (image without any change, to test retrieval performance), 2) subsampling to half the size, 3) rotation and sheer using the affine transformation defined by $\left(\begin{bmatrix} 0.5 & -0.87 \end{bmatrix}^T \begin{bmatrix} 0.5 & 0.87 \end{bmatrix}^T\right)$, 4) rotation and sheer using the affine transformation defined by $\left(\begin{bmatrix} 0.87 & -0.5 \end{bmatrix}^T \begin{bmatrix} 0.87 & 0.5 \end{bmatrix}^T\right)$, and 5) adding salt-and-paper noise with noise density of 0.1.

The second probe is obtained by selecting photographs of 97 CDs from the dataset used in [14] available from [ddoht2]. This dataset has 4 medium resolution photos of each object taken from different viewpoints, providing $4 \times 97 = 388$ probe images in total.

[2] A. Vedaldi's implementation: [argwtg]

Figure 1. Examples of images from CD/DVD dataset. The first two rows show examples of similar images in the dataset: two similar images of a 007 game on Xbox and Playstation2 (first row), and an English and a French version of the movie Ratatouille (second row). The third row shows a CD cover in the dataset (left) and its photographed image (right) cited in [14].

3.2. Pasadena Buildings

A second model-probe set was based on 750 photos of facades of 103 houses in the Pasadena area and 22 buildings from the Caltech campus. Each building was photographed six times, three times in the afternoon, and three times in the morning the next day. Thus, the light conditions vary between the two sets. Each time, the buildings were photographed from the front, and from the left and right at approximately 30°. For increased generality, the photographs were taken with two different digital cameras. For each building, one image was added to the model database (frontal view, afternoon), and the remaining five were used as its test images, which gives us a total of $125 \times 5 = 625$ images in the probe set.

Extract local features from all images
foreach image i **do**
- Set $m_j = 0$ for all images j in the dictionary
- **foreach** feature in image i get the matching feature f_i and its image label l_i from the dictionary and update $m_j = \sum_i \delta(l_i, j)$
- **foreach** image j with $m_j \geq T_m$ perform RANSAC affine verification and update m_j with the number of features consistent with affine transform
- Get $m = \max_k\{m_k\}$ and $j = \text{argmax}_k\{m_k\}$
- **if** $m < T_a$ **then** add image i to the dictionary **else** mark image i as similar to image j
- **if** size(dictionary)$\geq T_d$ **then** remove earliest images from dictionary and rebuild

Figure 2. Dataset Cleaning Algorithm

Figure 3. Small subset of the Pasadena buildings dataset. Each row show a different building. Photos in the first two columns were taken in the afternoon, while photos in the last column were taken the next morning with a different camera.

In order to test the algorithms' performance on large database sizes, we diluted the Pasadena buildings model set by adding to it 99,599 photos downloaded from Flickr by searching for the 145 most popular tags. All photos were downsampled to a size of 640×480 pixels.

4. Recognition Methods

We focus on three methods, that fall under two broad approaches for object recognition. The first, which we call the **SIFT/Match/Hough/RANSAC** pipeline, is the one proposed by Lowe [11], which represents each image by a set of SIFT features extracted at interesting points. During training the features extracted from all the images in the training set (one per object) are placed in a database. During recognition, the features of the test image are extraced, and for each such feature the database is searched for its (approximate) nearest neighbors. Potential object match candidates are checked for spatial consistency by using the Generalized Hough Transform followed by a RANSAC affine fitting. The database image with the maximum number of inliers is considered the matching image. We consider two methods for building a database that supports fast approximate nearest neighbors: Kd-trees and Locality Sensitive Hashing (LSH).

The second approach is the bag-of-words approach, which we call **SIFT/Quantize/Rank** pipeline [14, 15]. Here SIFT features are extracted and quantized into a codebook of visual words. Each image is represented by a histogram of word occurences. Given a test image, its features are extracted and quantized using the codebook computed during training, and its histogram is used to compute

its similarity to all images in the database, which are thus ranked. Next we explain these methods in more detail.

4.1. Kd-tree and Kd-forest

Exhaustive nearest neighbor search scales linearly with the number of features in the database, and the run time becomes unacceptable when the size of the database exceeds millions of features. Building the Kd-tree scales as $O(dN)$ where N is the number of features in the database and d is the number of dimensions (there are $2N-1$ nodes in the tree with N leaves). Memory requirements scale as $O(N)$ as we need to store split information at every node. However, searching through the Kd-tree scales as $O(\log N)$ where $\log N$ is the depth of the tree.

Kd-trees work best in low dimensions [6] (up to 10 rather than SIFT's 128), so following [11] we use an approximate version called Best-Bin-First Kd-tree [2] (we just call it Kd-tree hereinafter for simplicity), where promising branches are placed in a heap-based priority queue, and only a certain number of branches popped off the top of the queue are processed. It is approximate because it is not guaranteed to return the exact nearest neighbor in the database, however it provides a speed up of several orders of magnitude over exhaustive search while producing an acceptable number of false matches [11].

The number of false matches can be reduced by using a set of Kd-trees [10] (Kd-forest[3]). In order to build a Kd-forest, the individual Kd-trees are randomized at each step when choosing a dimension upon which to split the data, so that a random dimension among those with top variance is chosen rather than picking the dimension with maximum variance (as is the case with Kd-tree). When searching the Kd-forest, there is a single priority queue in which branches from all the trees are pushed in. This improves performance considerably by decreasing the chance of missing the true nearest neighbor.

4.2. Locality Sensitive Hashing

The key idea of the LSH approximate nearest neighbor (NN) algorithm is to construct a set of hash functions such that the probability of nearby points being close after transformation with the hash function is larger than the probability of two distant points being close after the same transformation. The range space of the function is discretized into buckets and we say that there is a 'collision' when two points end up in the same bucket. LSH has been shown to work well on high-dimensional datasets, and has a query time that scales sub-linearly with the dataset size under certain conditions [7].

[3]We use Kd-tree and Kd-forest interchangeably in the paper, and the distinction should be clear from context

A locality sensitive family $\mathcal{H}$ of hash functions is defined such that for any two points $p, q \in \mathbb{R}^d$ and $h \in \mathcal{H}$ chosen uniformly at random, $\Pr[h(q) = h(p)] \geq P_1$ if $\|p - q\| \leq R$, where R and P_1 are parameters specific to the application and dataset. In order to achieve the desired collision probability, we choose L functions $g_j(q) = (h_{1,j}(q), \ldots, h_{k,j}(q))$ where $h_{t,j}\,(1 \leq t \leq k, 1 \leq j \leq L)$ are chosen uniformly at random from $\mathcal{H}$ [7]. The parameters k and L are chosen to minimize the search time and maximize the collision probability of nearby points while minimizing collisions of distant points. We have omitted some of the details of the definitions, see [5] for a precise definition and review of LSH.

Given a query point q, all points stored in the buckets $g_1(q), \ldots, g_L(q)$ are retrieved. In the final step, the distance between q and all the points in all the buckets is computed to determine the NN of q. This scheme could in the worst case make the query time grow like $O(N)$.

We use the E^{2}LSH implementation available from [1], which measures distances according to the l_2 norm. For optimization reasons, that implementation defines the parameter m as $L = m \cdot (m-1)/2$. We tuned m and k by hand on a separate training set to keep the average search time per query point low (on the order of 1 ms), even for large datasets, while keeping the collision probability for nearby points as high as possible.

It is crucial to use a training set that resembles the anticipated test set when tuning the parameters in LSH. This is particularly important when using LSH to find NNs of SIFT descriptors, as we show in the following paragraphs. The final step of the LSH pipeline is to use RANSAC to fit an affine transformation from the database image to the test image, and then keep only the descriptors in the test image with NNs in the database image that are consistent with this transformation. Therefore, we focus on tuning the parameters of LSH to find true NNs for descriptors in test images, i.e. descriptors which survive the RANSAC spatial verification step. The task thus becomes to separate the set of all features, $\mathcal{F}^t$, in the test image into two separate subsets: features that have the potential of surviving RANSAC, $\mathcal{F}^t_+$, and all other features $\mathcal{F}^t_-$ (that may still be matched to the correct image in the database, but will not survive RANSAC). The NNs of $\mathcal{F}^t_+$ and $\mathcal{F}^t_-$ in the database image are denoted by $\mathcal{F}^d_+$ and $\mathcal{F}^d_-$ respectively. We optimize LSH to find accurate NNs for features in $\mathcal{F}^t_+$.

The two sets $\mathcal{F}^t_+$ and $\mathcal{F}^t_-$ are created by taking a test image and its corresponding database image and finding the NNs of the test image features in the database image.

After the NNs are found, we apply RANSAC to fit an affine transformation, H. Using H we back-project the database features to the test image and prune all the features which are inconsistent with the transfomation. The features that remains (that 'survived' RANSAC) and their

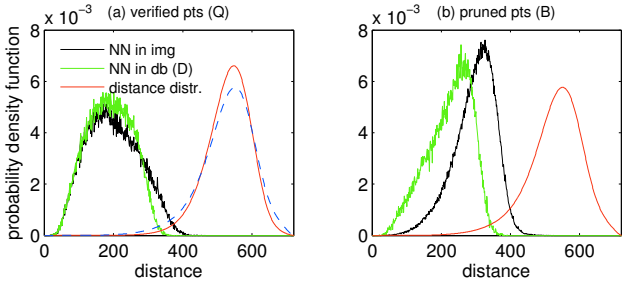

Figure 4. Distance distributions for SIFT descriptors in $\mathcal{Q}$ (left) and $\mathcal{B}$ (right). The curves show the distance distributions to the NNs in the ground truth database image (black), in the whole database $\mathcal{D}$ (green), and the overall distance to all points (red). The distance distribution from the pruned points has been inset as a blue curve in the left plot to show how the shapes differ (see text for analysis).

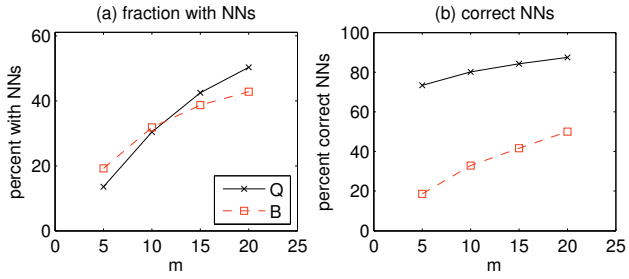

Figure 5. Importance of an appropriate training set when tuning LSH. The black and red curves show results for points in $\mathcal{Q}$ and $\mathcal{B}$ respectively. (a) The increase of NNs reported as a fraction of the the size of the query set, as m is increased. (b) The percentage of the reported NNs that are also exact NNs nearest neighbors for the same data as used in (a).

corresponding NNs in the test image make up $\mathcal{F}_+^d$ and $\mathcal{F}_+^t$, while all other features belong to $\mathcal{F}_-^d$ or $\mathcal{F}_-^t$. We repeat the procedure on ~ 100 pairs of images, and merge all the $\mathcal{F}_+^t$'s into a query set $\mathcal{Q}$. Similarly, we take all the features from the $\mathcal{F}_-^t$'s and call them $\mathcal{B}$. We take all the features from all the database images (no matter if they survived RANSAC or not), and call them the database set, $\mathcal{D}$. Fig. 4a (black curve) shows the distances of descriptors in $\mathcal{Q}$ to NNs in their corresponding database images. The green curve shows the distances from descriptors in $\mathcal{Q}$ to their NNs in the database, $\mathcal{D}$. The red curve shows the overall distance distribution from $\mathcal{Q}$ to all points in $\mathcal{D}$, be they NNs or not. Fig. 4b shows the corresponding curves for features in $\mathcal{B}$. Fig. 4 shows that features in $\mathcal{Q}$ have NNs in the matching database image at a shorter distance than features in $\mathcal{B}$ (black curves). Thus, LSH should be tuned to retrieve NNs only if they are "close enough" to the query point, otherwise the NN will probably not survive the final RANSAC step anyway.

Fig. 5 highlights the differences in performance when using descriptors from $\mathcal{Q}$ and $\mathcal{B}$ when querying $\mathcal{D}$. Fig. 5a shows what percentage of the query points for which LSH found a NN, as the parameter m is varied. The next plot shows that even though NNss are found for a similar fraction of the query points for the two query sets, the points in $\mathcal{Q}$ consistently return higher quality neighbors (i.e. true nearest neighbors).

We picked R by examining the curves in fig. 4, and optimized the other parameters to minimize the query time while keeping a high collision probability for points in $\mathcal{Q}$. We found $R = 250, m = 5, k = 30$ to yield good results.

4.3. Bag-of-Words Search

Textual information retrieval algorithms such as the vector space search method were introduced in image retrieval by [18, 14]. In this scheme, each image in the database is represented as a sparse vector of "visual words" occurrences. The bag-of-words vector space method was further improved by [15] and shown to work well on photos of Oxford buildings. We implement the method outlined by [15], with minor modifications, as described briefly below.

In [14, 15] it was found that vocabularies on the order of 10^5 or more words are necessary for good retrieval performance. Thus, our vocabularies are built using approximate K-means (AKM) which scales like $O(N \log K)$, and was shown to give best results in [15]. The approximate nearest neighbor search is performed using the publicly available FLANN package [12] with a forest of 8 Kd-trees trees and 512 checks. The number of features used for training the codebooks with 10^4 and 5×10^5 codewords was 5 and 20 million respectively.

Each image in the database is represented as a normalized histogram of its visual words occurrences using the tf-idf weighting scheme, which downweights frequently occuring, less discriminative words [18]. Test images are compared to all images in the index using the dot-product, measuring the cosine distance between the query and database vectors.

Enhanced retrieval performance may be achieved by a spatial, RANSAC-based, re-ranking of the M most similar database images [15]. We have on purpose omitted this step in here, since its purpose is not to re-rank, but to find the true positive amongst the top M results from the vector space search. Because M is an application-specific parameter that represents a tradeoff between classification accuracy and computational efficiency, we will mention it only briefly in this paper.

Although a query expansion method can improve performance in image retrieval [3], it does not help in the object recognition setting this study is concerned with. This is because we assume that there is only one example image per object in the database, and so there are no near-identical examples to expand the query against.

	Model	#feat	Probe	#ims
Scenario 1 § 3.1	CD/DVD	$\sim 10^8$	Synthetic	500
Scenario 2 § 3.1	CD/DVD	$\sim 10^8$	Photographed	388
Scenario 3 § 3.2	Flickr	$\sim 0.7 \times 10^8$	Buildings	625

Table 1. The three testing scenarios investigated. The third column lists the total number of features in the model set, and the fifth colum lists the number of images in the probe set.

The big advantage of the vector space indexing method is its efficient memory storage. Because of the quantization into visual words, the storage requirements in RAM is $O(N_{\text{img}} \cdot F)$, where F is the average number of features per database image and N_{img} is the number of images in the database. This is completely independent of the dimensionality of the feature vectors, unlike the requirements for the Kd-forests and LSH algorithms, which use $O(N_{\text{img}} \cdot F \cdot d)$ memory, where d is the dimensionality of the descriptor ($d = 128$ in the case of SIFT).

5. Experiments

5.1. Setup

We performed extensive experiments comparing the three methods in §4.1-4.3 on the two datasets described in §3 in the three testing scenarios, summarized in Table 1. In each of the three scenarios, we benchmarked the recognition performance by increasing the problem size using 1, 4, 8, 16, 64, and 128 thousand images in the model set. Experiments were run on a Quadcore Intel Xeon 2.83GHz machine with 32GB of memory. We used the standard SIFT feature descriptor with DoG interest point detector [11] extracted by the code written by Vedaldi and Fulkerson [20]. Benchmark results for these three scenarios are shown in Fig. 6.

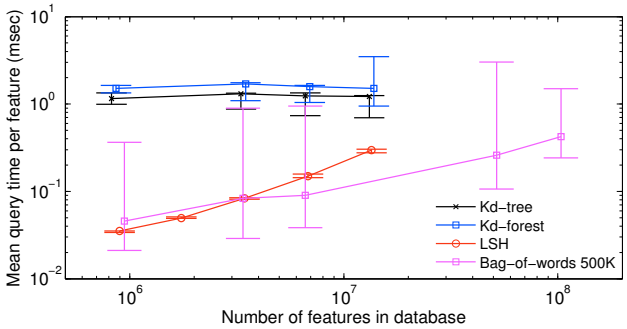

Figure 7. Comparison of how the query time per feature vary with database size for Kd-trees and LSH for the three scenarios. The curve for the Bag-of-words method shows the query time divided by the number words used in the query vector. The error bars show the maximum and minimum times, while the point indicates the median time. The curves have been shifted slightly horizontally to make error bars more visible.

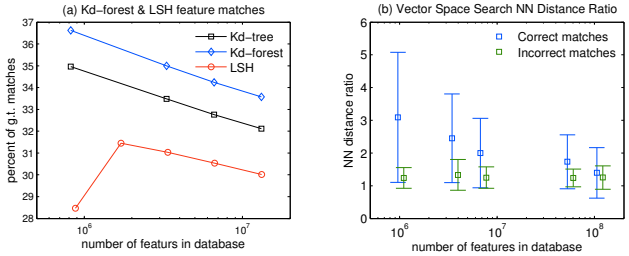

Figure 8. Degradation at large database sizes. (a) The percent of matching feature pairs that are consistent with the ground truth transformation for Kd-trees and LSH. (b) The ratio of the next-NN and NN distances when querying with bag-of-word histograms in the vector space search. The blue and green points show the correctly and incorrectly classified images respectively. Points were artificially shifted horizontally to be more visible.

5.2. Kd-tree and Kd-forest

In our experiments we considered two implementations of Kd-forests: FLANN, the implementation available from [12], and our implementation. The FLANN implementation only accepts floating-point inputs, so we could only fit in memory a bit over 1 million features. Our implementation of the kd-forest takes integer-valued features, enabling us to reach the maximum number of features that we have in our databases. Thus, we used our implementation in mex/Matlab for most of the experiments. We compared our implementation with FLANN and we found they give comparable results, see Fig. 9(a).

We also checked the effect of the Kd-forest size on the recognition performance. Fig. 9(b) shows results of comparing Kd-forests with 1, 5, 10 and 15 trees. Performance increases with the number of trees in the forest, though there is almost no gain from increasing the size above 10 trees. In our experiments we used single Kd-trees and Kd-forests with 5 trees, which provide a significant improvement while being manageable computationally.

The Kd-forest provides the best recognition performance in all three test scenarios, as shown in Fig. 6. The Kd-tree provides comparable results to the other methods, but is worse than Kd-forest. Performance of both degrades as the database size increases, which is expected as the probability of finding the true nearest neighbor decreases as the Kd-tree size increases. Due to the limited size of RAM available, we were not able to run the Kd-forest for database sizes 64K images and above (over 6×10^7 features).

5.3. Locality Sensitive Hashing

LSH was used to find the nearest neighbor in the databse to each feature in a probe image. In all the figures we used Lowe's [11] suggestion of using the Hough-transform to find the correct model in the database.

The performance of the LSH algoritm was comparable

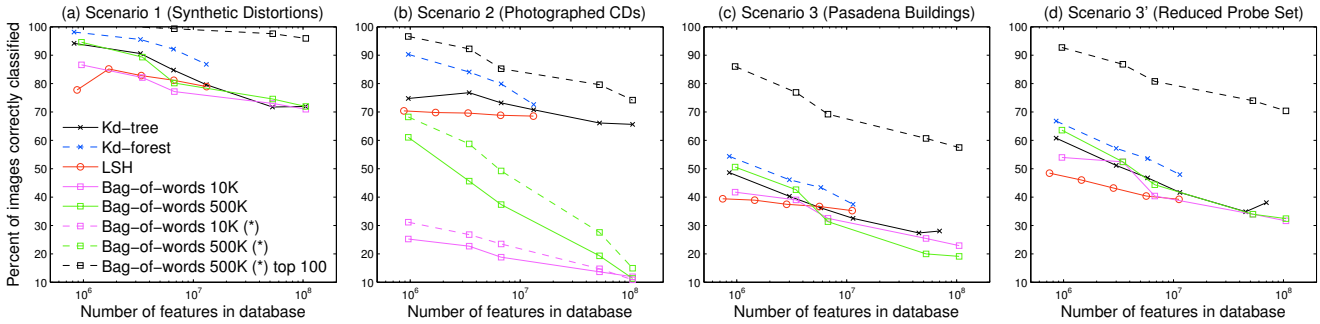

Figure 6. Benchmark on classification performance for the different techniques in the different testing scenarios. (*) denotes the instances where the codebook was trained on images of the same type as the test set. (d) shows results on scenario 3 with a reduced probe set, where for every house image we use the two test images that are taken at the same time of day so as to neutralize the effect of lighting change.

to, but slightly worse than, Kd-trees, see fig. 6. However, LSH seems to handle larger database sizes better; its performance stays approximately constant. The available implementation does not scale to experimnts larger than 16,000 images. This does not appear to be an instrinsic limit of the algorithm.

5.4. Bag-of-words Search

When benchmarking the bag-of-words nearest neighbor search with the tf-idf weighted index, we have defined a classification as correct when the test image gives the highest ranking score to the ground truth training image. It may be argued that this is an unfair comparison with the feature-based methods, which also include a spatial verification step (the Hough-transform). Indeed, it has been shown spatial verification of the top M images does improve recognition results [17]. However, this re-ranking technique would have to be compared to spatially verifying the M models with the highest numbers of matched features in the Kd-tree and LSH methods. The only fair comparison would be a comparison with a method where geometric verification has been incorporated into the indexing scheme [8]. As an upper bound, we have included in figure 6 the results when an image is considered as correctly classified if it is ranked among the top 100 images, i.e. $M = 100$.

In fig. 6, it is shown that the method performes as well as the Kd-trees on both scenarios 1 & 3. However, the bag-of-words representation fares much worse on scenario 2. This is most likely due to the fact that the vocabulary used for quantization was trained only on the scanned DVD covers, so background features in the test image makes the query vector more noisy. Indeed, as is shown in fig. 6(b), training the vocabulary on a combination of other photographs from the dataset in [14] and the scanned CD/DVD covers yields better results, although not as good as the Kd-trees and LSH.

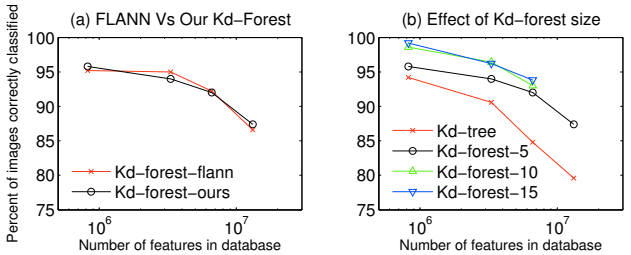

Figure 9. Kd-tree recognition performance vs number of database features in Scenario 1. (a) Comparison of our implementation of Kd-forest with FLANN shows comparable results. (b) Effect of Kd-forest size on recognition performance, with comparison of Kd-forests with 1, 5, 10, and 15 trees. Performance increases with increasing the number of trees.

6. Discussion

On both scenarios 1 & 3, Kd-forest outperformed LSH and Bag-of-words for smaller databases. This difference in performance was reduced as the database size was increased. It is worth reiterating here that it may be possible to tune the individual algorithms more extensively to do better on one or more of the datasets. However, our aim was to use the most basic implementation of each algorithm, in order to benchmark the core performance of the methods.

We noticed the performance was very poor on scenario 3, and we believe it is because some of the photos in the probe set were taken under different lighting conditions (at different times of the day). To verify this, we plotted peformance for a reduced probe set with photos taken in the same lighting condition as the database image, see fig. 6(d). It shows improvement of about 25%.

Fig. 7 shows the mean query time per features as the database size increases. It is interesting to see that the query time for the Kd-tree is almost constant, and in fact slightly decreasing with increasing database size. This is because as the Kd-tree size increases, the number of nodes deep down the tree increases. This increases the chance of having these nodes at the top of the priority queue, which decreases the search time with increased database sizes. On the other

hand, the searching time for LSH increases with increasing the database size as expected.

The performance of both the Kd-trees and the bag-of-words search degrade as the size of the database is increased. This is not the case for the LSH algortihm, which performs more or less constant for all database sizes we tried it on. Since both LSH and Kd-trees require the same amount of memory, this could make LSH preferable for larger databases, if the trend continues. Fig. 8(a) shows that the number of feature matches obtained with LSH decreases as database size increases (similar to kd-trees). However, the performance of LSH stays constant while the performance of the kd-trees deteriorates. We have no explanation for this behavior, but we hypothesize that the NNs provided by LSH are more likely to create a proper match given the special tuning of LSH parameters (§ 4.2).

The performance of the bag-of-words search also falls off with increasing database size, but for a slightly differnet reason. In the case of bag-of-words, the nearest neighbor search is between the images, represented as weighted bag-of-words histograms, and not between individual features. The bag-of-words representation breaks down as the distance between vectors in the database approaches the noise level for the query vectors. Fig. 8(b) shows the ratio of the next-NN and the NN distances in the index for the query images in scenario 2 (photographed CDs). For correctly classified images, this ratio is much higher (albeit with large variance) than for mis-classified query images, for which the ratio is ~ 1, independent of the database size. However, as the size of the database is increased, the ratio decreases even for correctly classified images, and for more than 100,000 images the ratio approaches unity. One solution to this problem might be to apply a local distance metric to the images in the database [9].

In summary, our main findings are:

1. The performance of all algorithms is very different in the three scenarios. In particular, performance on synthetic distortions (scenario 1) overestimates performance on 'real' data, and is overall quite uninformative. Furthermore, differences in performance between CD covers and buildings show that the statistics of images count for a lot. Thus, one should use a diverse collection of benchmark datasets of real images for the purpose of evaluating recognition algorithms.
2. Performance of bag-of-words techniques, which were designed to scale recognition to large datasets, degrades sharply with increasing database size in both real image scenarios. Our experiments do not show recognition performance advantages of these techniques with respect to Lowe's original method. There is, however, a significant advantage in memory usage, allowing larger experiments to be carried out. This suggests that a spatial consistency check among the top M images is crucial for this method, as indicated by the upper-bound curve in fig. 6.
3. Kd-trees and LSH scale differently. Recognition time with Kd-trees remains virtually constant as database size increases, while recognition performance decreases sharply. Instead, LSHs performance decreases slowly with database size, while recognition time increases quickly. The query time for the Bag-of-words search also increases fast.
4. RAM size will be a serious bottleneck if scaling to millions of images is an objective, it appears essential to develop approaches that can use RAM more efficiently and effectively, ideally allowing the user to set an upper bound on memory usage. Solutions allowing to distribute the database across multiple servers would of course be useful, but only if the number of servers grows sublinearly with the number of objects to be recognized.

References

[1] A. Andoni and P. Indyk. E2lsh: Exact euclidian locality-sensitive hashing, 2004. `http://web.mit.edu/andoni/www/LSH`.

[2] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[3] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, 2007.

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

[5] T. Darrell, P. Indyk, and G. Shakhnarovich, editors. *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2006.

[6] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. on Mathematical Software*, 3:209–226, 1977.

[7] P. Indyk and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. In *Symposium on Theory of Computing*, 1998.

[8] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, volume I, pages 304–317, oct 2008.

[9] H. Jegou, H. Harzallah, and C. Schmid. A contextual dissimilarity measure for accurate and efficient image search. In *CVPR*, 2007.

[10] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *CVPR*, pages 775–781, 2005.

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

[12] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Int. Conf. on Comp. Vision Theory and Applications*, 2009.

[13] M. Munich, P. Pirjanian, E. Di Bernardo, L. Goncalves, N. Karlsson, and D. Lowe. Application of Visual Pattern Recognition to Robotics and Automation. *IEEE Robotics & Automation Magazine*, pages 72–77, 2006.

[14] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, volume 2, pages 2161–2168, June 2006.

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

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

[17] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR*, Minneapolis, June 2007.

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

[19] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large databases for recognition. In *CVPR*, 2008.

[20] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.