Mobile Visual Search

Bernd Girod, *Fellow, IEEE,* Vijay Chandrasekhar, *Member, IEEE*, David M Chen, *Member, IEEE*, Ngai-Man Cheung, *Member, IEEE*, Radek Grzeszczuk, *Member, IEEE*, Yuriy Reznik, *Senior Member, IEEE,* Gabriel Takacs, *Member, IEEE,* Sam S Tsai, *Member, IEEE,* Ramakrishna Vedantham, *Member, IEEE,*

MOBILE phones have evolved into powerful image and video processing devices, equipped with high-resolution cameras, color displays, and hardware-accelerated graphics. They are increasingly also equipped with GPS, and connected to broadband wireless networks. All this enables a new class of applications which use the camera phone to initiate search queries about objects in visual proximity to the user (Fig 1). Such applications can be used, e.g., for identifying products, comparison shopping, finding information about movies, CDs, real estate, print media or artworks. First deployments of such systems include Google Goggles [1], Nokia Point and Find [2], Kooaba [3], Ricoh iCandy [4], [5], [6] and Amazon Snaptell [7].

Mobile image retrieval applications pose a unique set of challenges. What part of the processing should be performed on the mobile client, and what part is better carried out at the server? On the one hand, transmitting a JPEG image could take tens of seconds over a slow wireless link. On the other hand, extraction of salient image features is now possible on mobile devices in seconds or less. There are several possible client-server architectures:

- The mobile client transmits a query image to the server. The image retrieval algorithms run entirely on the server, including an analysis of the query image.
- The mobile client processes the query image, extracts features and transmits feature data. The image retrieval algorithms run on the server using the feature data as query.
- The mobile client downloads data from the server, and all image matching is performed on the device.

One could also imagine a hybrid of the approaches mentioned above. When the database is small, it can stored on the phone and image retrieval algorithms can be run locally [8]. When the database is large, it has to be placed on a remote server and the retrieval algorithms are run remotely.

In each case, the retrieval framework has to work within stringent memory, computation, power and bandwidth constraints of the mobile device. The size of the data transmitted over the network needs to be as small as possible to reduce network latency and improve user experience. The server latency has to be low as we scale to large databases. Further, the retrieval system needs to be robust to low quality camera-phone images. This paper reviews recent advances in content-based image retrieval with a focus on mobile applications.

Bernd Girod, Vijay Chandrasekhar, David Chen, Ngai-Man Cheung, Gabriel Takacs and Sam Tsai are with Stanford University, CA.

Radek Grzeszczuk and Ramakrishna Vedantham are with Nokia Research Center, Palo Alto, CA.

Yuriy Reznik is with Qualcomm Inc., San Diego, CA.

Fig. 1. Example of a mobile visual search application. The user points his camera phone at an object and obtains relevant information about it.

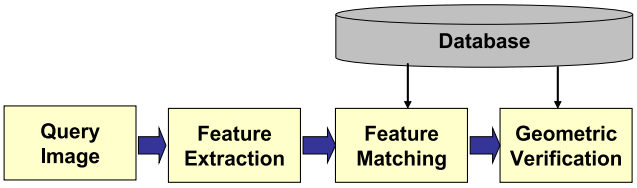


Fig. 2. Pipeline for image retrieval. Local features are extracted from the query image. Feature Matching finds a small set of images in the database that have many features in common with the query image. The Geometric Verification step rejects all matches with feature locations that cannot be plausibly explained by a change in viewing position.

We first review large-scale image retrieval highlighting recent progress in mobile visual search. As an example, we then present the Stanford Product Search system, a low latency interactive visual search system. Several sidebars invite the interested reader to dig deeper into the underlying algorithms.

Robust Mobile Image Recognition

The most successful algorithms for content-based image retrieval today use an approach that is referred to as "Bag of Features" (BoF) or "Bag of Words" (BoW). The BoW idea is borrowed from text retrieval. To find a particular text document, such as a web page, it is sufficient to use a few well-chosen words. In the database, the document itself can likewise be represented by a "bag" of salient words, regardless of where these words appear in the text. For images, robust local features take the analogous role of "visual words." Like text retrieval, BoF image retrieval does not consider where in the image the features occur, at least in the initial stages

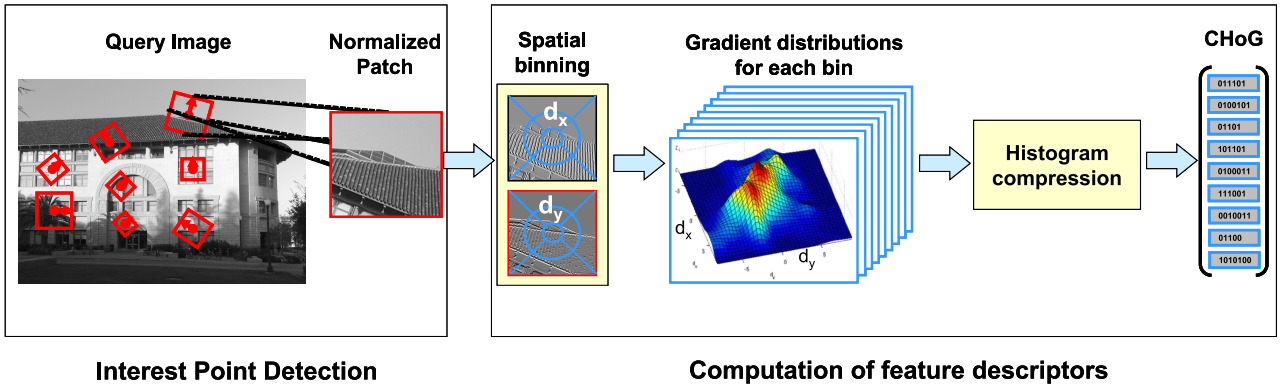


Fig. 3. Illustration of feature extraction. We first compute interest points (e.g., corners, blobs) at different scales. The patches at different scales are oriented along the dominant gradient. Feature extraction is followed by computation of feature descriptors that capture the salient characteristics of the image around the interest point. Here, we illustrate how the CHoG descriptor is computed. The scaled and oriented canonical patches are divided into localized spatial bins, which gives robustness to interest point localization error. The distribution of gradients in each spatial bin is compressed to obtain a very compact description of the patch.

of the retrieval pipeline. However, the variability of features extracted from different images of the same object makes the problem much more challenging.

A typical pipeline for image retrieval is shown in Fig. 2. First, local features are extracted from the query image. The set of image features is used to assess the similarity between query and database images. For mobile applications, individual features must be robust against geometric and photometric distortions encountered when the user takes the query photo from a different viewpoint, and with different lighting, compared to the corresponding database image.

Next, the query features are quantized [9], [10], [11], [12]. The partitioning into quantization cells is precomputed for the database, and each quantization cell is associated with a list of database images in which the quantized feature vector somewhere appears. This "inverted file" circumvents a pair-wise comparison of each query feature vector with all the feature vectors in the database and is the key to very fast retrieval. Based on the number of features they have in common with the query image, a short list of potentially similar images is selected from the database.

Finally, a geometric verification step is applied to the most similar matches in the database. Geometric Verification finds a coherent spatial pattern between features of the query image and the features of the candidate database image to ensure that the match is plausible.

For mobile visual search, there are considerable challenges to provide users with an interactive experience. Current deployed systems typically transmit an image from the client to the server, which might require tens of seconds. As we scale to large databases, the inverted file index becomes very large, with memory swapping operations slowing down the Feature Matching stage. Further, the Geometric Verification step is computationally expensive and thus increases response time. We discuss each block of the retrieval pipeline in the following, focusing on how to meet the challenges of mobile visual search.

Feature Extraction

Interest Point Detection: Feature extraction typically starts by finding salient interest points in the image. For robust image matching, we desire interest points to be repeatable under perspective transformations (or, at least, scale changes, rotation and translation) and real-world lighting variations. An example of feature extraction is illustrated in Fig. 3. To achieve scale invariance, interest points are typically computed at multiple scales using an image pyramid [13]. To achieve rotation invariance, the patch around each interest point is canonically oriented in the direction of the dominant gradient. Illumination changes are compensated by normalizing the mean and the standard deviation of the pixels of the gray values within each patch [14].

Numerous interest point detectors have been proposed in the literature. Harris Corners [15], SIFT Difference-of-Gaussian (DoG) [13] keypoints, Maximally Stable Extremal Regions (MSER) [16], Hessian Affine [14], FAST [17] and Hessian-blobs [18] are some examples. The different interest point detectors provide different trade-offs in repeatability and complexity. E.g., the SIFT DoG points are slow to compute, but highly repeatable, while the FAST corner detector is extremely fast but offers lower repeatability. In [19], Mikolajczyk et al. compare different interest point detectors in a common framework.

The Stanford Product Search system can perform feature extraction and compression on the client, to reduce system latency. Current generation smart phones have limited compute power, typically only a tenth of what a desktop PC provides. We require interest points that are fast to compute and highly repeatable. We choose the Hessian-blob detector sped up with integral images [18] which provides a good trade-off of repeatability and complexity. For VGA images, Hessian-blob interest point detection can be carried out in $\sim$1 second on current-generation smart phones [20].

Feature Descriptor Computation: After interest point detection, we compute a "visual word" descriptor on the normalized

patch. We would like descriptors to be robust to small distortions in scale, orientation and lighting conditions. Also, we require descriptors to be discriminative, i.e, characteristic of an image or a small set of images. Descriptors that occur in almost every image (the equivalent of the word "and" in text documents) would not be useful for retrieval. Since Lowe's paper in 1999 [21], the highly discriminative SIFT descriptor remains the most popular descriptor in computer vision. Other examples of feature descriptors are Gradient Location and Orientation Histogram (GLOH) by Mikolajczyk and Schmid [19], Speeded Up Robust Features (SURF) by Bay et al. [22] and our own Compressed Histogram of Gradients (CHoG) [23], [24]. Winder and Brown [25], [26], and Mikolajczyk et al. [19] evaluate the performance of different descriptors.

As a 128-dimensional descriptor, SIFT descriptor is conventionally stored as 1024 bits (8 bits/dimension). Alas, the size of SIFT descriptor data from an image is typically larger than the size of the JPEG compressed image itself. Several compression schemes have been proposed to reduce the bitrate of SIFT descriptors. In our recent work [27], we survey different SIFT compression schemes. They can be broadly categorized into schemes based on hashing [28], [29], [30], transform coding [31], [27] and vector quantization [32], [10], [11]. We note that hashing schemes like Locality Sensitive Hashing (LSH), Similarity Sensitive Coding (SSC) or Spectral Hashing (SH) do not perform well at low bitrates. Conventional transform coding schemes based on Principal Component Analysis (PCA) do not work well due to the highly non-Gaussian statistics of the SIFT descriptor. Vector quantization schemes based on the Product Quantizer [32] or a Tree Structured Vector Quantizer [10] are complex and require storage of large codebooks on the mobile device.

Through our experiments, we came to realize that simply compressing an "off-the-shelf" descriptor does not lead to the best rate-constrained image retrieval performance. One can do better by designing a descriptor with compression in mind. Of course, such a descriptor still has to be robust and highly discriminative. Ideally, it would permit descriptor comparisons in the compressed domain for speedy feature matching. To meet all these requirements simultaneously, we designed the Compressed Histogram of Gradients (CHoG) descriptor [23], [24]. Descriptors based on the distribution of gradients within a patch of pixels have been shown to be highly discriminative [25], [19]. Lowe [13], Bay *et al.* [22], Dalal and Triggs [33], Freeman and Roth [34], and Winder *et al.* [26] have proposed Histogram of Gradient (HoG) based descriptors. The CHoG descriptor is designed to work well at low bitrates (see *Box - CHoG: A Low Bitrate Descriptor*). CHoG achieves the performance of 1024-bit SIFT at less than 60 bits/descriptor. Since CHoG descriptor data are an order of magnitude smaller than SIFT or JPEG compressed images, it can be transmitted much faster over slow wireless links. A small descriptor also helps if the database is stored in the mobile device. The smaller the descriptor, the more features can be stored in limited memory.

Box 1 - CHoG: A Low Bitrate Descriptor

CHoG builds upon the principles of HoG descriptors with the goal of being highly discriminative at low bitrates. Fig. 3 illustrates how CHoG descriptors are computed.

- The patch is divided into spatial bins, which provides robustness to interest point localization error. We divide the patch around each interest point into soft log polar spatial bins using DAISY configurations proposed in [26]. The log polar configuration has been shown to be more effective than the square grid configuration used in SIFT [26], [35], [19].
- The joint (d_x, d_y) gradient histogram in each spatial bin is captured directly into the descriptor, as illustrated in Fig. 4. CHoG histogram binning exploits the skew in gradient statistics that are observed for patches extracted around interest points.
- CHoG retains the information in each spatial bin as a distribution. This allows the use of more effective distance measures like KL divergence, and more importantly, allow us to apply quantization and compression schemes that work well for distributions, to produce compact descriptors.

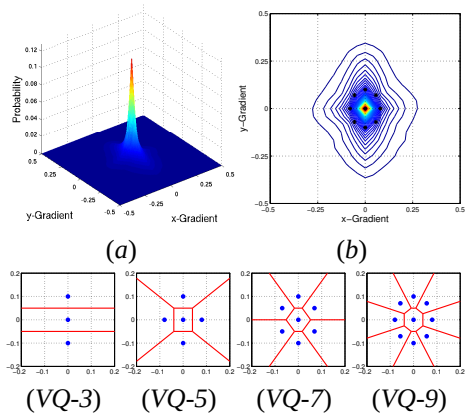


Fig. 4. The joint (d_x, d_y) gradient distribution (*a*) over a large number of cells, and (*b*), its contour plot. The greater variance in y-axis results from aligning the patches along the most dominant gradient after interest point detection. The quantization bin constellations VQ-3, VQ-5, VQ-7 and VQ-9 and their associated Voronoi cells are shown at the bottom.

Typically, 9 to 13 spatial bins and 3 to 9 gradient bins are chosen resulting in 27 to 117 dimensional descriptors. For compressing the descriptor, we quantize the gradient histogram in each spatial bin individually. In [23], [24], we have explored several novel quantization schemes that work well for compressing distributions: Quantization by Huffman Coding, Type Coding and optimal Lloyd-Max Vector Quantization (VQ). Here, we briefly discuss one of the schemes: Type Coding, which is linear in complexity to the number of histogram bins and performs close to optimal Lloyd-Max VQ.

Let m represent the number of histogram bins. m varies from 3 to 9 for the CHoG descriptor. Let $P = [p_1, p_2, ...p_m] \in R_+^m$ be the original distribution as described by the gradient

histogram, and $Q = [q_1, q_2, ...q_m] \in R_+^m$ be the quantized probability distribution. First, we first construct a lattice of distributions (or *types*) $Q_n = Q(k_1, \ldots, k_m)$ with probabilities

$$q_i = \frac{k_i}{n}, \quad k_i, n \in Z_+, \quad \sum_i k_i = n \tag{1}$$

We show several examples of such sets in $m = 3$ dimensions in Fig. 5.

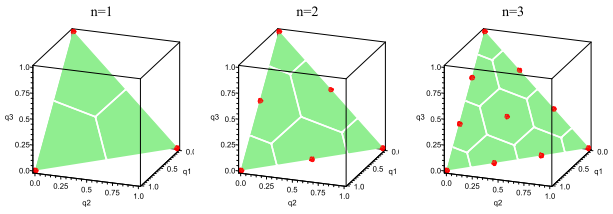


Fig. 5. Type lattices and their Voronoi partitions in 3 dimensions ($m = 3, n = 1, 2, 3$).

The parameter n controls the fidelity of quantization and higher the value of n parameter, higher the fidelity. Second, after quantizing the distribution P, we compute an index for the type. The total number of types $K(m, n)$ is the number of partitions of n into m terms $k_1 + \ldots + k_m = n$

$$K(m, n) = \binom{n + m - 1}{m - 1}, \tag{2}$$

The algorithm that maps a type to its index $f_n : \{k_1, \ldots, k_m\} \rightarrow [0, K(m, n) - 1]$ is described in [24].

Finally, we encode the index in each spatial cell with fixed-length or entropy codes. Fixed-length encoding provides the benefit of compressed domain matching at the cost of a small performance hit. The Type Quantization and coding scheme described here performs close to optimal Lloyd-Max VQ and does not require storage of codebooks on the mobile client. The CHoG descriptor with Type Coding at 60 bits matches the performance of the 128 dimensional 1024-bit SIFT descriptor [24].

As illustrated in Fig. 3, each interest point has a location, scale and orientation associated with it. Interest point locations are needed in the geometric verification step to validate potential candidate matches. The location of each interest point is typically stored as two numbers: x and y co-ordinates in the image at sub-pixel accuracy [13]. In a floating point representation, each feature location would require 64 bits, 32 bits each for x and y. This is comparable in size to the CHoG descriptor itself. We have developed a novel histogram coding scheme to encode the x, y coordinates of feature descriptors [36] (see *Box - Location Histogram Coding*). With location histogram coding, we can reduce location data by an order of magnitude compared to their floating point representation, without loss in matching accuracy.

Box 2 - Location Histogram Coding

Location Histogram Coding is used to compress feature location data efficiently. We note that the interest points in images are spatially clustered, as shown in Fig. 6. To encode their locations, we first generate a 2-D histogram from the locations of the descriptors, Fig. 7. Location histogram coding provides two key benefits. First, encoding the locations of a set of N features as a histogram reduces the bitrate by $\log(N!)$, compared to encoding each feature location in sequence [36]. This gain arises because ordering information ($N!$ unique orderings) is discarded when a histogram is computed. Second, we exploit the spatial correlation between the locations of different descriptors as illustrated in Fig. 6.

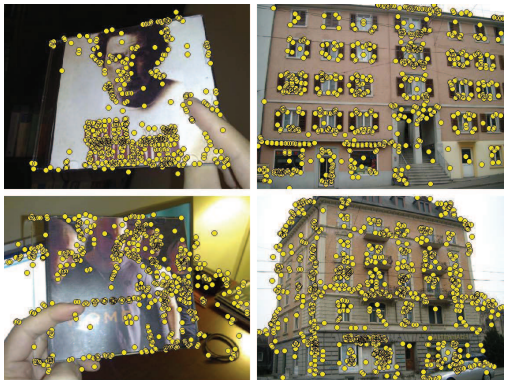


Fig. 6. Interest point locations in images tend to cluster spatially.

We divide the image into spatial bins and count the number of features within each spatial bin. We compress the binary map, indicating which spatial bins contains features, and a sequence of feature counts, representing the number of features in occupied bins. We encode the binary map using a trained context-based arithmetic coder, with neighbouring bins being used as the context for each spatial bin. Using location histogram coding, we can transmit each location with $\sim$5 bits/descriptor with little loss in matching accuracy - a $\sim$12.5$\times$ reduction in data compared to transmitting the location using a 64-bit floating point representation [37].

A few hundred descriptors per query image are sufficient for achieving high matching accuracy for large databases [24], [20]. Table I summarizes data reduction using CHoG and location histogram coding for 500 descriptors per image.

TABLE I
DATA REQUIRED TO REPRESENT AN IMAGE FOR MOBILE VISUAL SEARCH.

Scheme	Data (KB)
JPEG Compressed Image	30-40
SIFT + Uncompressed Location Data	66.4
CHoG + Uncompressed Location Data	7.6
CHoG + Compressed Location Data	4.0

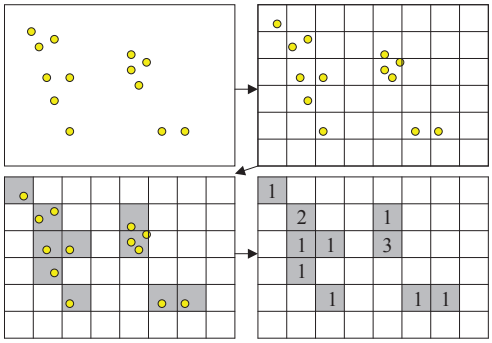


Fig. 7. We represent the location of the descriptors using a location histogram. The image is first divided into evenly spaced blocks. We enumerate the features within each spatial block generating a location histogram.

Feature Indexing and Matching

For a large database of images, comparing the query image against every database image using pairwise feature matching is infeasible. A database with millions of images might contain billions of features. A linear scan through the database would be too time-consuming for interactive mobile visual search applications. Instead, we must use a data structure that can quickly return a shortlist of the database candidates most likely to match the query image. The shortlist may contain false positives, as long as the correct match is included. Slower pairwise comparisons can subsequently be performed on just the shortlist of candidates rather than the entire database.

Many data structures have been proposed for efficiently indexing all the local features in a large image database. Lowe proposes approximate nearest neighbour (ANN) search of SIFT descriptors with a best-bin-first strategy [13]. One of the most popular methods is Sivic and Zisserman's Bag-of-Features (BoF) approach [9]. The BoF codebook is trained by k-means clustering of many training descriptors. During a query, scoring the database images can be made fast by using an inverted file index associated with the BoF codebook. To generate a much larger codebook, Nister and Stewenius utilize hierarchical k-means clustering to create a Vocabulary Tree (VT) [10]. The VT is explained in greater detail in the box "Vocabulary Tree and Inverted Index." Alternatively, Philbin et al. use randomized k-d trees to partition the feature descriptor space [12]. Subsequent improvements in tree-based quantization and ANN search include greedy N-best paths [38], query expansion [39], efficient updates over time [40], soft binning [12], and Hamming embedding [11].

As database size increases, the amount of memory used to index the database features can become very large. Thus, developing a memory-efficient indexing structure is a problem of increasing interest. Chum et al. use a set of compact min-hashes to perform near-duplicate image retrieval [41], [42]. Zhang et al. decompose each image's set of features into a coarse signature and a refinement signature [43]. The refinement signature is subsequently indexed by a locality sensitive hash (LSH). To support the popular VT scoring framework, inverted index compression methods for both hard-binned and soft-binned VT's have been developed by us [44], as explained in the box "Inverted Index Compression." The memory for BoF image signatures can alternatively be reduced using the mini-BoF approach [45]. Very recently, visual word residuals on a small BoF codebook have shown promising retrieval results with low memory usage [46], [47]. The residuals are indexed either with PCA and product quantizers [46] or with LSH [47].

Box 3 - Vocabulary Tree and Inverted Index

A Vocabulary Tree (VT) with an inverted index can be used to quickly compare images in a large database against a query image. If the VT has L levels excluding the root node and each interior node has C children, then a fully balanced VT contains $K = C^L$ leaf nodes. Fig. 8 shows a VT with $L = 2$, $C = 3$, and $K = 9$. The VT for a particular database is constructed by performing hierarchical k-means clustering on a set of training feature descriptors representative of the database, as illustrated in Fig. 8(a). Initially, C large clusters are generated from all the training descriptors by ordinary k-means with an appropriate distance function like L2-norm or symmetric KL divergence. Then, for each large cluster, k-means clustering is applied to the training descriptors assigned to that cluster, to generate C smaller clusters. This recursive division of the descriptor space is repeated until there are enough bins to ensure good classification performance. Typically, $L = 6$ and $C = 10$ are selected [10], in which case the VT has $K = 10^6$ leaf nodes.

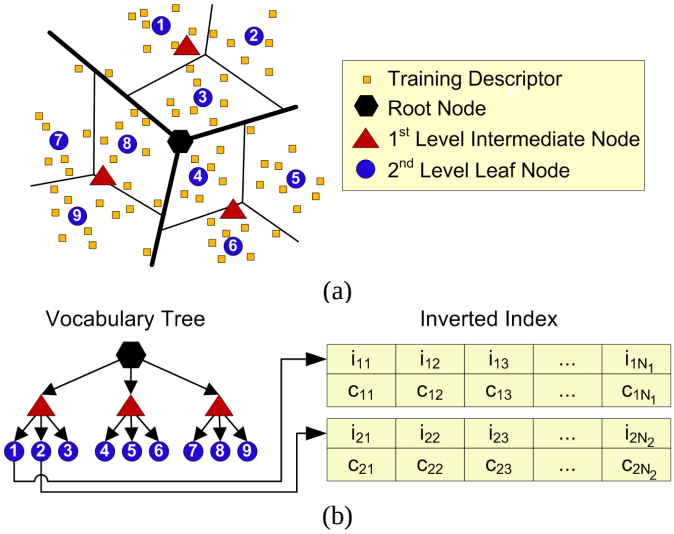


Fig. 8. (a) Construction of a Vocabulary Tree by hierarchical k-means clustering of training feature descriptors. (b) Vocabulary Tree and the associated inverted index.

The inverted index associated with the VT maintains two lists per leaf node, as shown in Fig. 8(b). For node k, there is a sorted array of image IDs $\{i_{k1}, i_{k2}, \cdots, i_{kN_k}\}$ indicating which N_k database images have visited that node. Similarly, there is a corresponding array of counts $\{c_{k1}, c_{k2}, \cdots, c_{kN_k}\}$ indicating the frequency of visits. During a query, a database of N total images can be quickly scored by traversing only

the nodes visited by the query descriptors. Let $s(i)$ be the similarity score for the i^{th} database image. Initially, prior to visiting any node, $s(i)$ is set to 0. Suppose node k is visited by the query descriptors a total of q_k times. Then, all the images in the inverted list $\{i_{k1}, \cdots, i_{kN_k}\}$ for node k will have their scores incremented according to

$$s(i_{kj}) := s(i_{kj}) + \frac{w_k^2 c_{kj} q_k}{\Sigma_{i_{kj}} \Sigma_q} \qquad j = 1, \cdots, N_k \quad (3)$$

where w_k is an inverse document frequency (IDF) weight used to penalize often-visited nodes, $\Sigma_{i_{kj}}$ is a normalization factor for database image i_{kj}, and Σ_q is a normalization factor for the query image.

$$w_k = \log(N/N_k) \quad (4)$$

$$\Sigma_{i_{kj}} = \sum_{n=1}^{K} w_n \,(\text{count for DB image } i_{kj} \text{ at node } n) \quad (5)$$

$$\Sigma_q = \sum_{n=1}^{K} w_n \,(\text{count for query image at node } n) \quad (6)$$

Scores for images at the other nodes visited by the query image are updated similarly. The database images attaining the highest scores $s(i)$ are judged to be the best matching candidates and kept in a shortlist for further verification.

Soft binning [12] can be used to mitigate the effect of quantization errors for a large VT. As seen in Fig. 8(a), some descriptors lie very close to the boundary between two bins. When soft binning is employed, the visit counts are then no longer integers but rather fractional values. For each feature descriptor, the m nearest leaf nodes in the VT are assigned fractional counts

$$c_i = 1/C \cdot \exp\left(-0.5 d_i^2/\sigma^2\right) \qquad i = 1, \cdots, m \quad (7)$$

$$C = \sum_{i=1}^{m} \exp\left(-0.5 d_i^2/\sigma^2\right) \quad (8)$$

where d_i is the distance between the i^{th} closest leaf node and the feature descriptor, and σ is appropriately chosen to maximize classification accuracy.

Box 4 - Inverted Index Compression

For a database containing one million images and a VT that uses soft binning, each image ID can be stored in a 32-bit unsigned integer and each fractional count can be stored in a 32-bit float in the inverted index. The memory usage of the entire inverted index is $\sum_{k=1}^{K} N_k \cdot 64$ bits, where N_k is the length of the inverted list at the k^{th} leaf node. For a database of one million product images, this amount of memory reaches 10 GB, a huge amount for even a modern server. Such a large memory footprint limits the ability to run other concurrent processes on the same server, such as recognition systems for other databases. When the inverted index's memory usage exceeds the server's available random access memory (RAM), swapping between main and virtual memory occurs, which significantly slows down all processes.

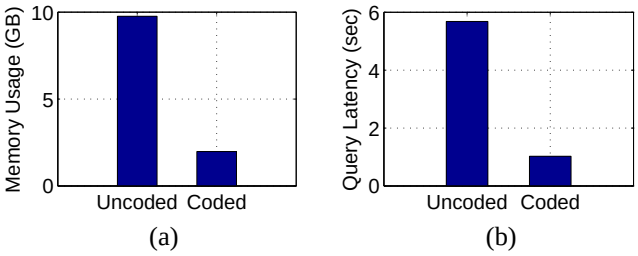


Fig. 9. (a) Memory usage for inverted index with and without compression. A $5\times$ savings in memory is achieved with compression. (b) Server-side query latency (per image) with and without compression. The RBUC code is used to encode the inverted index.

A compressed inverted index [44] can significantly reduce memory usage without affecting recognition accuracy. First, because each list of IDs $\{i_{k1}, i_{k2}, \cdots, i_{kN_k}\}$ is sorted, it is more efficient to store consecutive ID differences $\{d_{k1} = i_{k1}, d_{k2} = i_{k2} - i_{k1}, \cdots, d_{kN_k} = i_{kN_k} - i_{k(N_k-1)}\}$ in place of the IDs. This practice is also commonly used in text retrieval [48]. Second, the fractional visit counts can be quantized to a few representative values using Lloyd-Max quantization. Third, the distributions of the ID differences and visit counts are far from uniform, so variable-length coding can be much more rate-efficient than fixed-length coding. Using the distributions of the ID differences and visit counts, each inverted list can be encoded using an arithmetic code (AC) [49]. Since keeping the decoding delay low is very important for interactive mobile visual search applications, a scheme that allows ultra-fast decoding is often preferred over AC. The carryover code [50] and recursive bottom up complete (RBUC) code [51] have been shown to be at least $10\times$ faster in decoding than AC, while achieving comparable compression gains as AC. The carryover and RBUC codes attain these speed-ups by enforcing word-aligned memory accesses.

Fig. 9(a) compares the memory usage of the inverted index with and without compression, using the RBUC code. Index compression reduces memory usage from nearly 10 GB to 2 GB. This $5\times$ reduction leads to a substantial speed-up in server-side processing, as shown in Fig. 9(b). Without compression, the large inverted index causes swapping between main and virtual memory and slows down the retrieval engine. After compression, memory swapping is avoided and memory congestion delays no longer contribute to the query latency.

Geometric Verification

Geometric Verification (GV) typically follows the Feature Matching step. In this stage, we use location information of query and database features to confirm that the feature matches are consistent with a change in viewpoint between the two images. We perform pairwise matching of feature descriptors

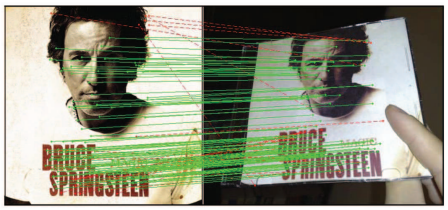


Fig. 10. In the GV step, we match feature descriptors pairwise and find feature correspondences that are consistent with a geometric model. True feature matches are shown in red. False feature matches are shown in green.

and evaluate geometric consistency of correspondences as shown in Fig. 10. The geometric transform between query and database image is estimated using robust regression techniques like RANSAC [52] or the Hough transform [13]. The transformation can be represented by the fundamental matrix which incorporates 3-D geometry, or simpler homography or affine models. Geometric Verification tends to be computationally expensive, which limits the list of candidate images to a small number.

A number of groups have investigated different ways to speed up the GV process. In [53], [54], Chum et al. investigate how to optimize steps to speed up RANSAC. Jegou et al. [11] use weak geometric consistency checks based on feature orientation information. Some authors have also proposed to incorporate geometric information into the VT matching step [55], [42].

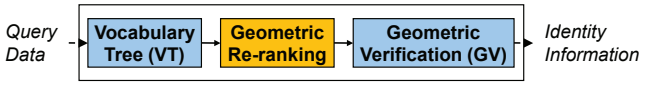


Fig. 11. A image retrieval pipeline can be greatly sped up by incorporating a geometric re-ranking stage.

To speed up geometric verification, one can add a geometric re-ranking step before the RANSAC GV step as illustrated in Fig. 11. In [56], we propose a re-ranking step that incorporates geometric information directly into the fast index look up stage, and use it to re-order the list of top matching images (see *Box - Fast Geometric Re-ranking*). The main advantage of the scheme is that it only requires x, y feature location data, and does not use scale or orientation information as in [11]. As scale and orientation data are not used, they need not be transmitted by the client, which reduces the amount of data transferred. We typically run fast geometric re-ranking on a large set of candidate database images, and reduce the list of images that we run RANSAC on.

Box 5 - Fast Geometric Re-ranking

We have proposed a fast geometric re-ranking algorithm in [56], that uses x, y locations of features to rerank a short-list of candidate images. First, we generate a set of potential feature matches between each query and database image based on VT quantization results. After generating a set of feature correspondences, we calculate a geometric score between them. The process used to compute the geometric similarity score is illustrated in Fig. 12. We find the distance between two features in the query image and the distance between the corresponding matching features in the database image. The ratio of the distance corresponds to the scale difference between the two images. We repeat the ratio calculation for features in the query image that have matching database features. If there exists a consistent set of ratios (as indicated by a peak in the histogram of distance ratios), it is more likely that the query image and the database image match.

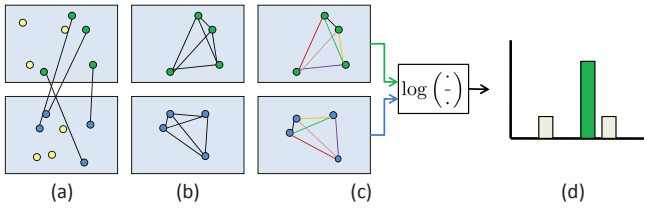


Fig. 12. The location geometric score is computed as follows: (a) features of two images are matched based on VT quantization, (b) distances between pairs of features within an image are calculated, (c) log distance ratios of the corresponding pairs (denoted by color) are calculated , and (d) histogram of log distance ratios is computed. The maximum value of the histogram is the geometric similarity score. A peak in the histogram indicates a similarity transform between the query and database image.

The geometric re-ranking is fast because we use the vocabulary tree quantization results directly to find potential feature matches and using a really simple similarity scoring scheme. The time required to calculate a geometric similarity score is 1-2 orders of magnitude less than using RANSAC.

System Performance

What performance can we expect for a mobile visual search system that incorporates all the ideas discussed so far? To answer this question, we have a closer look at the experimental Stanford Product Search System (Fig. 13). For evaluation, we use a database of one million CD, DVD and book cover images, and a set of 1000 query images (500×500 pixel resolution) [57] exhibiting challenging photometric and geometric distortions, as shown in Fig. 14. For the client, we use a Nokia 5800 mobile phone with a 300MHz CPU. For the recognition server, we use a Linux server with a Xeon E5410 2.33GHz CPU and 32GB of RAM. We report results for both 3G and WLAN networks. For 3G, experiments are conducted in an AT&T 3G wireless network, averaged over several days, with a total of more than 5000 transmissions at indoor locations where such an image-based retrieval system would be typically used.

We evaluate two different modes of operation. In *Send Features* mode, we process the query image on the phone and transmit compressed query features to the server. In *Send Image* mode, we transmit the query image to the server and all operations are performed on the server.

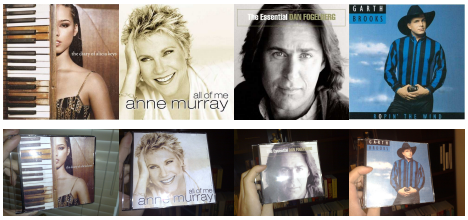


Fig. 14. Example image pairs from the dataset. A clean database picture (*top*) is matched against a real-world picture (*bottom*) with various distortions.

We discuss results of three key aspects that are critical for mobile visual search applications: retrieval accuracy, system latency and power. A recurring theme throughout this section will be the benefits of performing feature extraction on the mobile device compared to performing all processing on a remote server.

Retrieval Accuracy

It is relatively easy to achieve high precision (low false positives) for mobile visual search applications. By requiring a minimum number of feature matches after RANSAC geometric verification, we can avoid false positives entirely. We define Recall as the percentage of query images correctly retrieved. Our goal is to then maximize Recall at a negligibly low false positive rate.

Fig. 15 compares the Recall for three schemes: *Send Features (CHoG)*, *Send Features (SIFT)* and *Send Image (JPEG)*. For the JPEG scheme, the bitrate is varied by changing the quality of compression. For the SIFT scheme, we extract SIFT descriptors on the mobile device, and transmit each descriptor uncompressed as 1024 bits. For the CHoG scheme, we need to transmit about 60 bits per descriptor accross the network. For SIFT and CHoG schemes, we sweep the Recall-bitrate curve by varying the number of descriptors transmitted.

First, we observe that a Recall of 96% is achieved at the highest bitrate for challenging query images even with a million images in the database. Second, we observe that the performance of the JPEG scheme rapidly deteriorates at low bitrates. The performance suffers at low bitrates as the interest point detection fails due to JPEG compression artifacts. Third, we note that transmitting uncompressed SIFT data is almost always more expensive than transmitting JPEG compressed images. Finally, we observe that the amount of data for CHoG descriptors are an order of magnitude smaller than JPEG images or SIFT descriptors, at the same retrieval accuracy.

System Latency

The system latency can be broken down into 3 components: processing delay on client, transmission delay, and processing delay on server.

Client and Server Processing Delay: We show the time for the different operations on the client and server in Table II. The *Send Features* mode requires $\sim$1 second for feature extraction on the client. However, this increase in client processing time is more than compensated by the decrease in transmission latency, compared to *Send Image*, as we illustrate in Fig. 16 and 17. On the server, using VT matching with a compressed inverted index, we can search through a million image database in 100 milliseconds. We perform GV on a short list of 10 candidates after fast geometric re-ranking of the top 500 candidate images. We can achieve $<$1 second server processing latency while maintaining high recall.

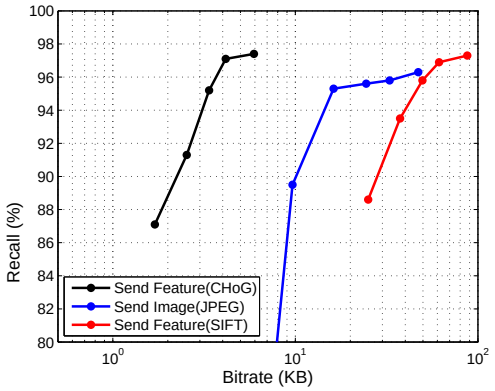
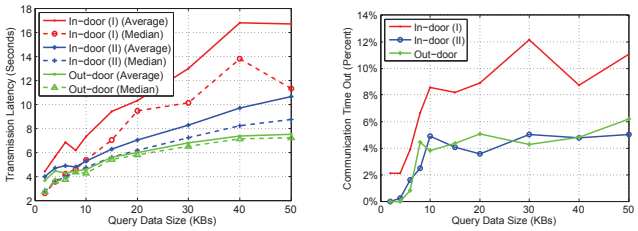


Fig. 15. Bitrate comparisons of different schemes. CHoG descriptor data are an order of magnitude smaller compared to JPEG images or uncompressed SIFT descriptors.

TABLE II
PROCESSING TIME

Client side operations	Time (sec)
Image capture	1-2
Feature extraction and compression (for *Send Features* mode)	1-1.5
Server side operations	Time (msec)
Feature extraction (for *Send Image* mode)	100
Vocabulary tree matching	100
Fast geometric re-ranking (per image)	0.46
Geometric verification (per image)	30



(a) (b)

Fig. 16. Measured transmission latency (a) and time-out percentage (b) for transmitting queries of different size over a 3G network. “In-door (I)” is tested in-doors with poor connectivity. “In-door (II)” is tested in-doors with good reception. “Out-door” is tested outside of buildings.

Transmission Delay: The transmission delay depends on the type of network used. In Fig. 17, we observe that data transmission time is insignificant for a WLAN network due to the high bandwidth available. However, transmission time turns out to be a bottleneck for 3G networks. In Fig. 16, we present experimental results for sending data over a 3G wireless network. We vary query data sizes from that of typical compressed query features (3-4 KB) to typical JPEG query

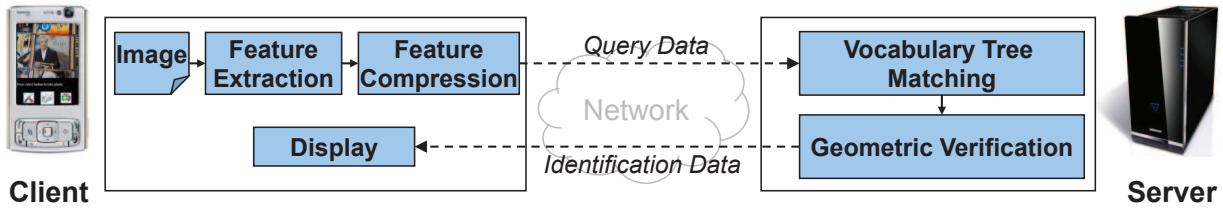


Fig. 13. Stanford Product Search System. Due to the large database, the image recognition server is placed at a remote location. In most systems [3], [7], [1], the query image is sent to the server and feature extraction is performed. In our system, we show that by performing feature extraction on the phone, we can significantly reduce the transmission delay and provide an interactive experience.

images (50 KB) to learn how query size affects transmission time. The communication time-out was set to 60 seconds. We have conducted the experiment continuously over several days. We tested at three different locations, typical locations where a user might use the visual search application.

The median and average transmission latency of our experiments are shown in Fig. 16. Sending the compressed query features typically takes 3-4 seconds. The time required to send the compressed query image is several times longer and varies significantly at different locations. However, transmission delay does not include the cases when communication fails entirely, which increases with query size. We show the percentage of transmissions that experience a time-out in Fig. 16(b). The time-out percentage of transmitting compressed query features is much lower than that of transmitting compressed query images because of their smaller query size.

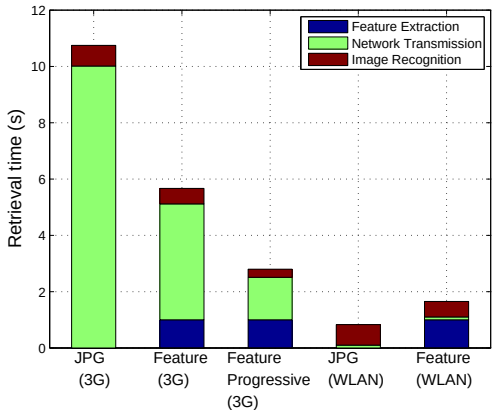


Fig. 17. End-to-end latency for different schemes. Compared to *Send Image* scheme, we achieve approximately 4× reduction in average system latency using progressive transmission of CHoG feature descriptors in a 3G network.

End-to-End Latency: We compare end-to-end latency for different schemes in Fig. 17. For WLAN, we observe that < 1 second query latency is achieved for *Send Image* mode. *Send Features* mode is slower due to the processing delay on the client. With such fast response times over WLAN, we are able to operate our system in a continuous Mobile Augmented Reality mode [58].

For 3G networks, network latency remains the bottleneck as seen in Fig. 17. In this scenario, there is significant benefit in sending compressed features. *Send Features* reduces system latency by 2× compared to *Send Image* mode.

Energy Consumption

On a mobile device, we are constrained by the energy of the battery, and hence, conserving energy is critical. We measure the average energy consumption associated with a single query using the Nokia Energy Profiler [1] on the Nokia 5800 phone.

We show the average energy consumption for a single query using *Send Features* and *Send Image* for WLAN and 3G network connections in Fig. 18. For 3G connections, the energy consumed in *Send Image* mode is almost 3× as much as *Send Features*. The additional time needed to transmit image data compared to feature data results in a greater amount of energy being consumed. For WLAN transmission, *Send Image* consumes less energy, since feature extraction on the mobile client is not required.

Finally, we compute the number of image queries the mobile can send before the battery runs out of power. A typical phone battery has voltage of 3.7 V and a capacity of ∼1000 mAH (or ∼13.3K Joules). Hence, for 3G connections, the maximum number of images that the mobile can send is 13.3K Joules / 70 Joules = ∼ 190 total queries. For *Send Features*, we would be able to perform 13.3 K joules / 21 Joules = ∼630 total queries, which is 3× as many queries as *Send Image* can perform. This difference becomes even more important as we move towards streaming augmented reality applications.

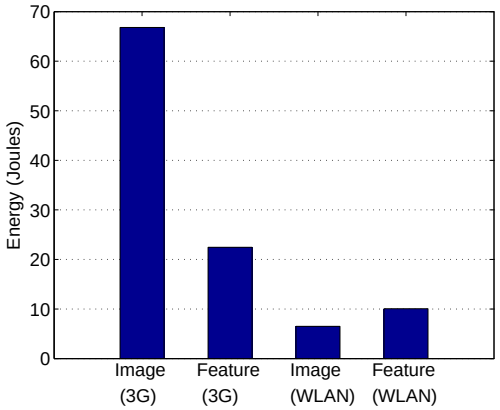


Fig. 18. Average energy consumption of a single query using *Send Image* and *Send Features* mode for various types of transmission.

[1]Nokia Energy Profiler: http://store.ovi.com/content/17374

CONCLUDING REMARKS

Mobile Visual Search is ready for prime-time. State-of-the-art systems today achieve over 95% recall at negligible false positive rate for databases with over 1M classes, a recognition performance that is sufficient for many applications. The key are robust and discriminative local features that are used in a "Bag-of-Visual-Words" approach. Robustness against scale changes and rotation is achieved by interest point detection algorithms that robustly yield not only a location, but also feature scale and dominant orientation. That permits feature descriptors computed on canonical patches in a local coordinate system. Robustness against brightness and contrast variations is achieved by normalizing the patch variance and using only the image gradient to compute feature descriptors. This takes care of much of the variability among corresponding query and database features, but not all. Foreshortening affects the feature descriptor and correspondences can no longer be established, if the angle becomes extreme. Adding foreshortened descriptors to the database or using affine invariant descriptors are possible remedies. Robustness against blur (particularly motion blur) is typically lacking, but would be highly desirable. By using clever hierarchical data structures, such as vocabulary trees, fast retrieval can be achieved in less than 1 sec on a typical desktop CPU for databases of more than a 1M images. Such speed requires a precomputed inverted file index which is stored entirely in RAM - seeking data on a hard disk would slow retrieval considerably. Inverted index compression can be used to fit even larger data structures in RAM. For large databases, vocabulary trees must be combined with a geometric verification stage to avoid false positives. Geometric consistency checks are computationally expensive, so it makes sense to use simple geometric re-ranking to reduce the number of candidate images.

Feature compression is a key problem for visual search to work with mobile devices. Sending a JPEG image as a query over a slow wireless link can take a long time; it is better to extract salient features on the phone instead and send these features as the query to the server. Note that this approach also provides a certain degree of privacy. For a small database, compressed database features could be stored on the mobile device, and matching could be performed directly on the device without wireless communication. One can compress well-known feature descriptors, such as SIFT, trading off recognition performance against size. Better performance, however, is achieved by the CHoG descriptor that was designed directly for rate-constrained recognition. CHoG needs about 60 bits per feature, including its location.

As an example throughout, we have used the Stanford Product Search system which implements many of the ideas discussed. We observe that the processing latency on the client and server is typically on the order of $\sim$1 second. The transmission delay depends on the network being used. For WLAN, the transmission delay is insignificant, and transmitting a JPEG image as a query is faster than extracting CHoG features on the phone. For 3G networks, the transmission delay is the bottleneck in the end-to-end system latency. By transmitting feature data, we can reduce end-to-end system latency to 2-3 seconds, compared to sending the image which takes several times as long. Somewhat counter to intuition, extracting and transmitting feature data on the mobile client requires $3\times$ less energy than sending the image.

Numerous open problems remain. Accurate and near-instantaneous web-scale visual search with billions of images will likely remain one of the Grand Challenges of multimedia technology for years to come. And we would like to perform mobile visual search at video rates, without ever pressing a button. While faster processors and networks will get us closer to this goal, lower-complexity image analysis algorithms are urgently needed. Hardware support on mobile devices should also help. Ultimately, we may expect to see ubiquitous mobile augmented reality systems that continuously superimpose information and links on everything the camera of a mobile device sees, thus seamlessly linking the virtual world and the physical world.

REFERENCES

[1] *Google Goggles*, http://www.google.com/mobile/goggles/.

[2] *Nokia Point and Find*, http://www.pointandfind.nokia.com.

[3] *Kooaba*, http://www.kooaba.com.

[4] B. Erol, E. Antúnez, and J. Hull, "Hotpaper: multimedia interaction with paper using mobile phones," in *Proc. of the 16th ACM Multimedia Conference*, New York, NY, USA, 2008.

[5] J. Graham and J. J. Hull, "Icandy: a tangible user interface for itunes," in *Proc. of CHI '08: Extended abstracts on human factors in computing systems*, Florence, Italy, 2008.

[6] J. J. Hull, B. Erol, J. Graham, Q. Ke, H. Kishi, J. Moraleda, and D. G. Van Olst, "Paper-based augmented reality," in *Proc. of the 17th International Conference on Artificial Reality and Telexistence (ICAT)*, Washington, DC, USA, 2007.

[7] *SnapTell*, http://www.snaptell.com.

[8] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod, "Outdoors augmented reality on mobile phone using loxel-based visual feature organization," in *Proc. of ACM International Conference on Multimedia Information Retrieval (ACM MIR)*, Vancouver, Canada, October 2008.

[9] J. Sivic and A. Zisserman, "Video Google: A Text Retrieval Approach to Object Matching in Videos," in *Proc. of IEEE International Conference on Computer Vision (ICCV)*, Washington, DC, USA, 2003.

[10] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, New York, USA, June 2006.

[11] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Proc. of European Conference on Computer Vision (ECCV)*, Berlin, Heidelberg, 2008.

[12] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Lost in quantization - improving particular object retrieval in large scale image databases," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, Alaska, June 2008.

[13] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[14] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A Comparison of Affine Region Detectors," *International Journal on Computer Vision*, vol. 65, no. 1-2, pp. 43–72, 2005.

[15] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. of 4th Alvey Vision Conference*, 1988.

[16] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *Proc. of British Machine Vision Conference (BMVC)*, Cardiff, Wales, UK, September 2002.

[17] E. Rosten and T. Drummond, "Machine Learning for High Speed Corner Detection," in *Proc. of Euproean Conference on Computer Vision (ECCV)*, Graz, Austria, May 2006.

[18] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Proc. of European Conference on Computer Vision (ECCV)*, Graz, Austria, May 2006.

[19] K. Mikolajczyk and C. Schmid, "Performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[20] S. S. Tsai, D. M. Chen, V. Chandrasekhar, G. Takacs, N. M. Cheung, R. Vedantham, R. Grzeszczuk, and B. Girod, "Mobile Product Recognition," in *Proc. of ACM Multimedia (ACM MM)*, Florence, Italy, October 2010.

[21] D. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, August 1999.

[22] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust feature," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[23] V. Chandrasekhar, G. Takacs, D. M. Chen, S. S. Tsai, R. Grzeszczuk, and B. Girod, "CHoG: Compressed Histogram of Gradients - A low bit rate feature descriptor," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, June 2009.

[24] V. Chandrasekhar, Y. Reznik, G. Takacs, D. M. Chen, S. S. Tsai, R. Grzeszczuk, and B. Girod, "Study of Quantization Schemes for Low Bitrate CHoG descriptors," in *Proc. of IEEE International Workshop on Mobile Vision (IWMV)*, San Francisco, California, June 2010.

[25] S. Winder and M. Brown, "Learning Local Image Descriptors," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, Minneapolis, Minnesota, 2007, pp. 1–8.

[26] S. Winder, G. Hua, and M. Brown, "Picking the best daisy," in *Proc. of Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, June 2009.

[27] V. Chandrasekhar, M. Makar, G. Takacs, D.M. Chen, S. S. Tsai, N. M. Cheung, R. Grzeszczuk, Y. Reznik, and B. Girod, "Survey of SIFT Compression Schemes," in *Proc. of International Mobile Multimedia Workshop (IMMW), IEEE International Conference on Pattern Recognition (ICPR)*, Istanbul, Turkey, August 2010.

[28] C. Yeo, P. Ahammad, and K. Ramchandran, "Rate-efficient visual correspondences using random projections," in *Proc. of IEEE International Conference on Image Processing (ICIP)*, San Diego, California, October 2008.

[29] A. Torralba, R. Fergus, and Y. Weiss, "Small Codes and Large Image Databases for Recognition," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, Alaska, 2008.

[30] Y. Weiss, A. Torralba, and R. Fergus, "Spectral Hashing," in *Proc. of Neural Information Processing Systems (NIPS)*, Vancouver, BC, Canada, December 2008.

[31] V. Chandrasekhar, G. Takacs, D. M. Chen, S. S. Tsai, and B. Girod, "Transform coding of feature descriptors," in *Proc. of Visual Communications and Image Processing Conference (VCIP)*, San Jose, California, January 2009.

[32] H. Jegou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *Accepted to IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[33] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Diego, CA, June 2005.

[34] W. T. Freeman and M. Roth, "Orientation histograms for hand gesture recognition," in *Proc. of International Workshop on Automatic Face and Gesture Recognition*, 1994, pp. 296–301.

[35] E. Tola, V. Lepetit, P. Fua, "A Fast Local Descriptor for Dense Matching," in *Conference on Computer Vision and Pattern Recognition*, 2008.

[36] S. S. Tsai, D. M. Chen, G. Takacs, V. Chandrasekhar, J. P. Singh, and B. Girod, "Location coding for mobile image retreival systems," in *Proc. of International Mobile Multimedia Communications Conference (MobiMedia)*, London, UK, September 2009.

[37] S. S. Tsai, D. Chen, G. Takacs, V. Chandrasekhar, J. P. Singh, and B. Girod, "Location coding for mobile image retrieval," in *Proc. of International Mobile Multimedia Communications Conference (Mobimedia)*, London, UK, September 2009.

[38] G. Schindler, M. Brown, and R. Szeliski, "City-scale location recognition," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, Minnesota, June 2007.

[39] 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 *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, Minnesota, 2007.

[40] T. Yeh, J. J. Lee, and T. J. Darrell, "Adaptive vocabulary forests for dynamic indexing and category learning," in *Proc. of IEEE International Conference on Computer Vision (ICCV)*, Rio de Janeiro, Brazil, 2007.

[41] O. Chum, J. Philbin, and A. Zisserman, "Near duplicate image detection: min-hash and tf-idf weighting," in *Proc. of British Machine Vision Conference (BMVC)*, Leeds, United Kingdom, September 2008.

[42] O. Chum and M. Perdoch and J. Matas, "Geometric min-Hashing: Finding a (thick) needle in a haystack," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, USA, June 2009.

[43] X. Zhang, Z. Li, L. Zhang, W.-Y. Ma, and H.-Y. Shum, "Efficient indexing for large-scale visual search," in *Proc. of IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, September 2009.

[44] D. M. Chen, S. S. Tsai, V. Chandrasekhar, G. Takacs, R. Vedantham, R. Grzeszczuk, and B. Girod, "Inverted Index Compression for Scalable Image Matching," in *Proc. of IEEE Data Compression Conference (DCC)*, Snowbird, Utah, March 2010.

[45] H. Jegou, M. Douze, and C. Schmid, "Packing bag-of-features," in *Proc. of IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, September 2009.

[46] H. Jegou, M. Douze, C. Schmid, and P. Perez, "Aggregating local descriptors into a compact image representation," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, USA, June 2010.

[47] F. Perronnin, Y. Liu, J. Sanchez, and H. Poirier, "Large-scale image retrieval with compressed fisher vectors," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, USA, June 2010.

[48] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys*, vol. 38, no. 2, pp. 6, July 2006.

[49] A. Said, "Comparative analysis of arithmetic coding computational complexity," in *HP Labs Technical Report*, 2004.

[50] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," *Information Retrieval*, vol. 8, no. 1, pp. 151–166, January 2005.

[51] A. Moffat and V. N. Anh, "Binary codes for non-uniform sources," in *Proc. of IEEE Data Compression Conference (DCC)*, Snowbird, Utah, March 2005.

[52] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[53] O. Chum, J. Matas, and J. V. Kittler, "Locally optimized RANSAC," in *Proc. of DAGM Symposium*, Magdeburg, Germany, September 2003.

[54] O. Chum, T. Werner, and J. Matas, "Epipolar geometry estimation via ransac benefits from the oriented epipolar constraint," in *Proc. of International Conference on Pattern Recognition (ICPR)*, Cambridge, UK, August 2004.

[55] Z. Wu, Q. Ke, M. Isard, and J. Sun, "Bundling features for large scale partial-duplicate web image search," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, USA, 2009.

[56] S. S. Tsai, D. M. Chen, G. Takacs, V. Chandrasekhar, R. Vedantham, R. Grzeszczuk, and B. Girod, "Fast Geometric Re-ranking for Image based Retrieval," in *Proc. of IEEE International Conference on Image Processing (ICIP)*, Hong Kong, September 2010.

[57] D. M. Chen, S. S. Tsai, R. Vedantham, R. Grzeszczuk, and B. Girod, *CD Cover Database - Query Images*, April 2008, http://vcui2.nokiapaloalto.com/ dchen/cibr/testimages/.

[58] D. M. Chen, S. S. Tsai, R. Grzeszczuk, R. Vedantham, and B. Girod, "Streaming mobile augmented reality on mobile phones," in *Proc. of International Symposium on Mixed and Augmented Reality (ISMAR)*, Orlando, Florida, USA, October 2009.