Mobile Product Search with Bag of Hash Bits and Boundary Reranking

Junfeng He, Jinyuan Feng, Xianglong Liu, Tao Cheng, Tai-Hsu Lin, Hyunjin Chung, Shih-Fu Chang
EE Department, Columbia University
jh2700@columbia.edu, fengjy01.cu@gmail.com, xlliuchina@gmail.com, tc2569@columbia.edu
lord.lth@gmail.com, hyunjinc.chung@gmail.com, sfchang@ee.columbia.edu

Abstract

Rapidly growing applications on smartphones have provided an excellent platform for mobile visual search. Most of previous visual search systems adopt the framework of "Bag of Words", in which words indicate quantized codes of visual features. In this work, we propose a novel visual search system based on "Bag of Hash Bits" (BoHB), in which each local feature is encoded to a very small number of hash bits, instead of quantized to visual words, and the whole image is represented as bag of hash bits. The proposed BoHB method offers unique benefits in solving the challenges associated with mobile visual search, e.g., low transmission cost, cheap memory and computation on the mobile side, etc. Moreover, our BoHB method leverages the distinct properties of hashing bits such as multi-table indexing, multiple bucket probing, bit reuse, and hamming distance based ranking to achieve efficient search over gigantic visual databases. The proposed method significantly outperforms state-of-the-art mobile visual search methods like CHoG, and other (conventional desktop) visual search approaches like bag of words via vocabulary tree, or product quantization. The proposed BoHB approach is easy to implement on mobile devices, and general in the sense that it can be applied to different types of local features, hashing algorithms and image databases. We also incorporate a boundary feature in the reranking step to describe the object shapes, complementing the local features that are usually used to characterize the local details. The boundary feature can further filter out noisy results and improve the search performance, especially at the coarse category level. Extensive experiments over large-scale data sets up to 400k product images demonstrate the effectiveness of our approach.

1. Introduction and Related Works

1.1. Mobile Visual Search

The advent of smartphones provides a perfect platform for mobile visual search, in which many interesting applications [7, 15, 10] have been developed, such as location search, product search, augmented reality, etc. Among them mobile product search is one of the most popular, because of the commercial importance and wide user demands. There are several preliminary commercial systems on mobile product search such as Google " Goggles", Amazon "Snaptell", and Nokia "Point and Find". For mobile product search, local feature (LF) like SIFT or SURF[1] is a popular choice, since global features usually cannot support object-level matching, which is crucial for product search.

Similar as conventional desktop visual search problems, mobile visual search has the same requirement of efficient indexing and fast search. However, besides it, mobile visual search has some unique challenges, e.g.,reducing the amount of data sent from the mobile to the server[1], having low computation and cheap memory on the mobile side, etc.

Early mobile visual search systems only use mobile as a capture and display device. They usually send the query image in a compressed format like JPEG to the server, and apply all other steps, like local feature extraction and searching, on the server side. As the computation capacity of smartphones becomes more and more powerful, extracting local features on the mobile client can be done very fast now, almost in real time. So recent mobile visual systems tend to extract local features on the mobile side. However, such local features need to be compressed before transmission; otherwise, sending the raw local features may cost more than sending the image. As shown in [7, 3], if we compress each local feature to tens of bits and send the compressed bits, the transmission cost/time will be reduced by many times, compared to sending the JPEG images. So in this paper, we will only focus on the paradigm that transmits compressed local features instead of JPEG images.

One straightforward approach for compressing local features is to quantize each local feature to a visual word on the mobile side, and then send the visual words to the server. However, most quantization methods with large vocabulary

[1]Not only because the bandwidth and speed of networks is still limited, but also because sending more data will cost users more money and consume more power.

(which is important for good search results) such as vocabulary tree [12], are not applicable on current mobile devices, due to the limited memory and computation capacity. Similarly, some promising fast search methods like [11] are not suitable for mobile visual search either, because they usually need large memory and/or heavy computation on the mobile side.

The most popular way for mobile visual search nowadays is to compress the local features on the mobile side by some coding method, for instance CHoG [4], in which the raw features is encoded by using entropy based coding method, and can be decoded to approximately recover features at the server. The server will then quantize the recovered features to visual codewords and following the standard model of "bag of words" (BoW), which represents one image as collections of visual words contained in the image.

1.2. Motivations and Contributions

In this paper, we present a new mobile visual search system based on Bag of Hash Bits (BoHB) instead of conventional Bag of Words. More specifically, in the proposed BoHB approach, each local feature is encoded to tens of hash bits using similarity preserving hashing functions, and each image is then represented as a bag of hash bits instead of bag of words.

First, the proposed BoHB method meets the unique requirements of mobile visual search: for instance, the mobile side only needs very little memory (i.e., storing tens of vectors with the same dimension of the local feature) and cheap computation (i.e., tens of inner products for each local feature). And moreover, the data sent from mobile to server is also very compact, about the same as the state-of-the-art mobile visual search method like CHoG [7, 4], much smaller than sending JPEG images.

Moreover, in terms of efficient searching, roughly speaking, the main difference between bag of words representation and bag of hash bits representation is how to search the database and find matched local features for each local feature contained in the query. In the bag of words model, this step is done by quantizing each local feature to a visual word, and then all local features with the same word index are considered as "matched" ones. To some extend, the hash bits of each local feature can also be viewed as the visual word index, however, the advantages of using hash bits are: 1) In "bag of words" representation, the distance between "word index" of local features is meaningless. For example, word index 4 is not "meaningfully" closer to word index 5 than word index 200, since word index are just clustering labels; however, the hamming distance between the hash bits is actually meaningful when we use similarity preserving hash functions, like PCA hashing, SPICA hashing [9] or LSH [5]. The hamming distance among hash bits is often designed to approximate the original feature distance, and hence is helpful for matching local features much more accurately. 2) the hash bits allow us to create the indexing structure in a very flexible manner, eg., by building multiple hash tables. Hence, searching matched local features can be done by checking multiple buckets in multiple hash tables. These advantages are important in developing successful search systems that maintain high retrieval accuracy while reducing the number of the candidate results and hence the search time.

Some previous works [3, 7] have reported that hashing may not be a good choice to compress local features, and hence not suitable for mobile visual search. We believe such preliminary conclusions are drawn based on implementations that did not fully explore the potentials of the hash techniques. In this paper, we will present a different finding and develop a hash based system for mobile visual search that significantly outperforms prior solutions. The key ideas underlying our approaches are:

1. Use compact hashing (e.g., PCA hashing (PCH) or SPICA hashing [9]) instead of random hash functions like Locality Sensitive Hashing (LSH) [5].

2. Build multiple hash tables and probe multiple buckets in each hash table within certain Hamming distance thresholds when searching for the nearest neighbor ("matched") local features, instead of just using the linear scan over the hash bits .

3. Generate multiple hash tables through hash bit reuse, which further helps reduce the number of transmitted data to tens of bits per local feature.

The other focus of the paper is to develop effective and efficient features suitable for mobile product search. Boundary features are especially suitable for this purpose, since the object boundary can be represented in a very compact way, without further compression. Moreover, boundary feature is complementary with local features that have been used in typical systems. Local features can capture unique content details of the objects very well. However, it lacks adequate descriptions about the object shape, which can actually be provided by the boundary information. There are some works on boundary feature [14]. However, the combination of boundary features with the local features has not been explored for mobile visual search. To the best of our knowledge, our work is the first one to fuse local feature and boundary feature together for mobile product search. One of the main difficulties to use boundary features is to automatically segment the objects in the database and obtain the boundaries. However, for product image databases, this is usually not a major concern because of the relatively clean background in the images crawled from the sites like Amazon and Zappos. Even for the images captured and submitted by users, usually the product

object is located in the center of the picture with a high contrast to the background. By applying automatic saliency cut techniques like those proposed in[6], we will demonstrate the abilities (see Figure 2) to automatically extract boundaries for product images. Finally for images with complicated backgrounds, we also provide interactive segmentation tools like Grabcut [13] on the mobile side to further improve the boundary accuracy extracted from the query images.

The main contributions of this work include:

1. We have provided a mobile visual scheme based on Bag of Hash Bits instead of conventional Bag of Words. Our "Bag of Hash Bits" approach significantly outperforms the-state-of-art visual search methods, including not only mobile ones[7, 4] but also conventional (desktop) search sytems[11, 12].

2. We have developed new techniques to use hash bits, make hashing an very effective way for compressing local features and matching local features. We have also incorporated boundary reranking to improve the accuracy of mobile product search, especially at the category level.

3. We have collected a large scale challenging product search data sets with diverse categories. These product data sets will be released to facilitate further research in this exciting research area. Moreover, we have implemented a fully functional mobile product search system including all the functions and the large product data set described in the paper.

2. An Overview for the Proposed Approach

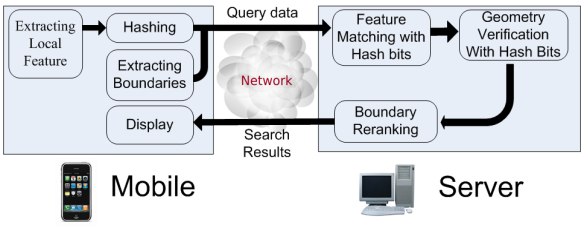

Figure 1. Architecture of the proposed mobile product search approach based on bag of hash bits and boundary reranking.

Figure 1 shows the overall workflow of the proposed system. In our work, we choose SURF as the local feature, because of its proven performance in accuracy and speed in several existing systems. For database indexing, each local feature in the database is encoded into M bits (M is tens in our case) by using similarity preserving hashing functions. Multiple hash tables are built, each of which uses a subset of the k bits.

For online searching, first, on the client (mobile) side, we compress each local feature in the query image to M bits by the same similarity preserving hashing function. We also encode the (x,y) coordinates of each local feature, using less than 10 extra bits in a way similar to [7], in order to use the spatial layout information for reranking. We then send the hash bits (M bits) and the coordinate bits (less than 10 bits) for all the local features, together with the boundary curve of the whole object to the server. Note that only one boundary curve is needed and it is usually very compact, for example, less than 200 bytes, and hence will increase the transmission cost very little.

For a local feature in the query image and a local feature in the database, if they fall into two buckets within a Hamming distance r in any hash table, the database local feature is considered a "matched" feature for the query local feature. The search process starts with finding all matched local features to each query local feature by probing all buckets within hamming distance r in all hash tables. Once the matched features are found, we collect candidate images whose "matched" local features exceed a certain threshold. We then apply an image-level geometry verification to compute the geometry similarity scores between the query image and candidate images. The geometry verification utilizes both the hash bits and the locations of local features in the image. Finally, we integrates object boundary features into our reranking process. By fusing the geometry similarity score and boundary similarity score, we rerank the candidate images and return the final top K retrieved images.

Our bag of hash bit approach requires similar bit budgets (e.g., 60-100 bits per local feature) as the state-of-art mobile visual search works like CHoG [4, 7], but a much higher search accuracy and faster search speed as shown in the experiments. For example, as shown in the experiments over a large dataset of product images, compared to CHoG, our approach can obtain about the same accuracy (in terms of recall for top K results) but tens of times speedup in the search step, or perform significantly better in both search accuracy (30% improvement) and search time (several times speedup). The BoHB method also (significantly) outperforms popular conventional visual search systems, such as bag of words via vocabulary tree[12], or product quantization[11]. Moreover, hashing based approach is very fast to compute (only requiring tens of inner products), and very easy to implement on mobile devices, and applicable for different types of local features, hashing algorithms and image databases.

Moreover, from the boundary curve, we extract a boundary feature called "central distance", which is translation, scale, and rotation invariant to boundary changes. By incorporating the boundary feature into the reranking step, the overall search performance is further improved, especially

in retrieving products of the same categories (shoe, furniture, etc).

3. Mobile Visual Search with Bag of Hash Bits

3.1. Hash Local Features into Bits

We will apply linear hashing methods for encoding each local features to hash bits. More specifically, for any local feature x, where x is a 128 dimension vector when using SURF in our case, we can get one hash bit for x by

$$b = sign(v^T x - t) \quad (1)$$

Here v is a projection vector in the feature space, and t is a threshold scalar. Though v can be a projection vector from any linear similarity preserving hashing method, we have found randomly generated v like in LSH[5] performs quite poorly. On the other hand, projections from compact hashing like PCA hashing or SPICA hashing [9] will provide much better search performance. So in the rest of this paper, we assume v comes from PCA projections or SPICA [9]. Moreover, as well known in hashing research area, balancing hash bit will usually improve the search performance, so we choose t as the median value such that half of each bit are +1, and the other are -1.

Following the considerations in [7], constrained by transmission speed, we limit the number of hash bits for each local feature to less than 100.

3.2. Matching Local Features with Hash Bits

For one local feature in the query image, we need to find out its "matched" ones, i.e., nearest neighbor local features in the database. Our current product data sets consist of $300k - 400k$ images, and each image will have about $100 - 200$ local features on average, so in total there are more than 30M local features in our database. To find matched local features for all local features in the query image, we need to search the 30M local features for hundreds of times. If we apply linear scan to search for matched local features, the computation will be prohibitively expensive, even with tens of bits for each local feature.

Matching Local Features with Multiple Hash Tables One popular technique to achieve fast sublinear search rather than linear scan, is to utilize multiple hash tables.

For one hash bit/function, denote p_{NN} as the probability of one query point (local feature in our case) and one of its neighbor points to have the same bit value, and p_{any} as one query point and a random database point to have the same bit value. When using similarity persevering hash functions, p_{NN} will be larger than p_{any}. Suppose we use k bits in one table. For the simplicity of our discussion, we assume p_{NN} is the same for every bit, and so is and p_{any}. And moreover, bits are independent. (Violation of these assumptions will make the discussion much more complex but lead to similar conclusions.) Denote $P_{NN}(k, r)$ as the probability of one query point and one of its nearest neighbors to fall into two buckets whose hamming distance is no larger than r, where $r \ll k$. We have:

$$P_{NN}(k, r) = \sum_{i=0,...r} C_k^i (p_{NN})^{k-i} (1 - p_{NN})^i$$

Similarly, denote $P_{any}(k, r)$ as the probability of one query point and a random database point to fall into two buckets whose hamming distance is no larger to r. Similarly,

$$P_{any}(k, r) = \sum_{i=0,...r} C_k^i (p_{any})^{k-i} (1 - p_{any})^i$$

Note that $p_{any} < p_{NN}$ and $r \ll k$, so $P_{any}(k, r)$ will decrease much faster than $P_{NN}(k, r)$ when k increases. Note that the expected number of returned nearest neighbors is $N_{NN} P_{NN}(k, r)$ where N_{NN} is the number of total nearest neighbor points for the query point. Moreover, the expected number of all returned points is $N P_{any}(k, r)$, where N is the number of points in the database. So the precision of nearest neighbors in returned points is

$$\frac{N_{NN} P_{NN}(k, r)}{N P_{any}(k, r)},$$

If k becomes larger, $\frac{N_{NN} * P_{NN}(k,r)}{N * P_{any}(k,r)}$ will becomes larger too, and hence the precision of finding matched local features will be high. However, when k is large, $N_{NN} * P_{NN}(k, r)$ itself may be too small, and hence we cannot obtain enough nearest neighbors. So we can increase the number of hash tables L to improve the chance of obtaining nearest neighbors, while still keep the high precision.

If we only check one bucket (i.e., $r = 0$) in each hash table, L often has to be very large like hundreds to get a reasonable recall for finding nearest neighbors. One popular solution to reduce the number of tables is to probe multiple buckets in one hash table. For example, if we set r as 2 or 3, and check all buckets within hamming distance r in each hash table, the number of needed tables can then be reduced significantly, to about ten for example, because $P_{NN}(k, r)$ will increase when r becomes larger, for a fixed k.

In practice, hamming distance r is usually a small number, e.g., ≤ 3, the number of bits k in each hash table is about $20 - 40$, and L is $5 - 20$.

Multiple Hash Tables Proliferation We adopt the idea of using multi hashing table to find nearest neighbor local features for the query ones. However, the number of bits to build L hash tables is Lk, usually hundreds or thousands bits. For mobile visual search, we only have a budget of tens of bits per local feature. So instead of sending hundreds/thousands of hash bits for each local feature over the mobile network, we only send tens of bits per local feature, but generate multiple hash tables by

reusing the bits. More specifically, suppose we have M bits for each local feature, we will build each hash table by randomly choosing a subset of k bits from M bits. We have observed if M is more than 2 or 3 times larger than k, constructing tables by reusing bits does not largely affect the search result, especially when the number of hash tables is not large, e.g., about ten in our case. We can thus construct multiple tables without increasing the total amount of bits needed for each local feature.

3.3. Geometry Verification with Hash Bits

Suppose one database image contains several matched local features, one would like to check if these matches are geometrically consistent, i.e., whether a valid geometric transformation can be established between the feature positions in the query image and the positions in the database image. The existence of a consistent geometric transformation between two images strongly indicates that the image indeed contains similar object(s). Considering the popular geometric verification method, RANSAC, is too slow, we apply a fast geometry verification method based on length ratio, inspired by [7]. The method is about tens/hundreds of times faster than the RANSAC algorithm and was shown to perform well on real-world databases. Intuitively, it estimates the portion of matched features between query and reference images that share a consistent scale change. The higher value this is, the higher score the candidate reference image receives.

Length Ratio Similarity with Hash Bits

More specifically, for a the query image I_q and a database image I_{db}, suppose they have $m \geq 2$ matched local features. For two features p and q in I_q, suppose their matched features are p' and q' in I_{db} respectively. Denote x_p and y_p as the (x, y) coordinate of local feature p in the image. The length ratio is defined as the ratio of distance between p and q and distance between p' and q': $\frac{||(x_{p'}-x_{q'})^2+(y_{p'}-y_{q'})^2||^{\frac{1}{2}}}{||(x_p-x_q)^2+(y_p-y_q)^2||^{\frac{1}{2}}}$. There are C_m^2 ratio values between I_q and I_{db}, since there are m matched local features, and C_m^2 matched feature pairs. Each ratio value i will be quantized to some bin a in the ratio value histogram. Suppose, ration value i is computed with local feature p, p' and q, q' , then i's vote to bin j is computed as

$$v_{i,j} = \alpha^{(d_{pp'}+d_{qq'})}, \text{if } j == a,$$

$$v_{i,j} = 0, \text{ otherwise.}$$

Here α is a constant which is smaller than but close to 1, and $d_{pp'}$ and $d_{qq'}$ are the hamming distances between p, p' and q, q' respectively. Then the maximum value in the histogram is taken as the geometry similarity score between two images, or more specifically,

$$S_g(I_q, I_{db}) = \max_j \sum_{i=1,\ldots,C_m^2} v_{i,j}. \quad (2)$$

However, one observation is: the similarity score as above results in an approximately quadratic growth versus the number of matched points. In our experiment, we have found this put too much weight on the number of matches while ignoring the quality of matches themselves. One distracter image with complex background can, e.g., have higher score than the correct one just because it has more matches to the background clutter of query image. We thus divide the maximum value in the histogram by the total number of matches to further improve the robustness to noisy matches.

4. Boundary Reranking

To obtain boundary features, we need to extract the boundaries first. Since product objects are usually the most salient regions in images, we applied the SaliencyCut algorithm to extract the boundaries automatically [6].

We also implement the interactive segmentation Grabcut [13] on the mobile device, to further improve the segmentation accuracy for the query.

Some example results of the automatic SaliencyCut are shown in figure 2.

Figure 2. Examples of automatic SaliencyCut results. The first 4 are segmented correctly, while the last two do not find perfect cut due to shadow, lighting, and distracting background. Pictures are best viewed in color.

There are different boundary features proposed [14], however, in our system, we utilize a very straightforward boundary feature, central distance, because of its simplicity and robustness.

Before feature extraction, we first smooth the boundary by using a moving average filter to eliminate noises on the boundary. Then the feature is expressed by the distances between sampled points $\mathbf{p}(n)$ along the shape boundary and the shape center $\mathbf{c} = (c_x, c_y)$:

$$\mathbf{D}(n) = \frac{\|\mathbf{p}(n) - \mathbf{c}\|_2}{\max_n \|\mathbf{p}(n) - \mathbf{c}\|_2}, n = 1, 2, \ldots, N$$

The points $\mathbf{p}(n)$ are sampled with a fixed length along the boundary. For a desired feature length N, the sampling step can be set to L/N, where L is the boundary length (the total number of pixels in the boundary curve). It is easy

to see the central distance feature is invariant to translation. In addition, the feature $\mathbf{D}$ is normalized by its maximum element, and hence will be scale invariannt.

Moreover, to make it rotation invariant and start point independent, the discrete Fourier transform (DFT) is applied:

$$\mathbf{F}(n) = |f[\mathbf{D}(n)]|, n = 1, 2, \ldots, N \quad (3)$$

where $f[\cdot]$ is the discrete Fourier transform and can be compute efficiently by fast Fourier transform (FFT) in linear time. Any circular shift of the feature vector $\mathbf{D}(n)$ only affects the phases of its frequency signal, while $\mathbf{F}(n)$, the magnitude of frequency signal, keeps unchanged. In sum, $\mathbf{F}(n)$ will be translation, scale, and rotation invariant to boundary changes.

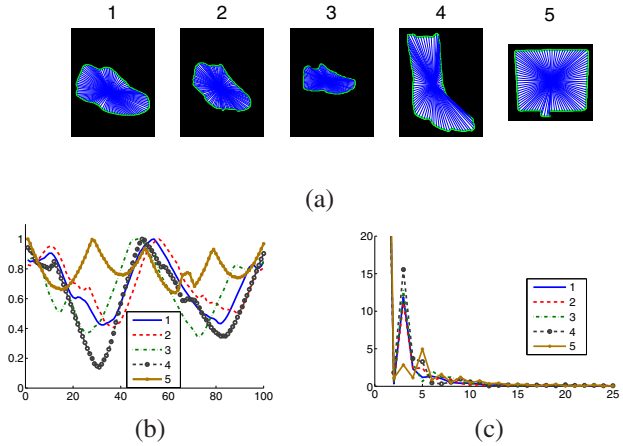

Figure 3. (a) 5 example object boundaries. (b) The central distance features for 5 samples. (c) The central distance features in frequency domain. The boundaries are extracted by the automatic SaliencyCut method. As shown in (b) and (c), the similarity among the central distance features or their frequencies capture the similarity among boundaries quite well. Graphs are best viewed in color.

For one query image I_q and one database image I_{db}, their boundary similarity $S_b(I_q, I_{db})$ is defined as $S_b(I_q, I_{db}) = e^{-||F_q - F_{db}||}$, where F_q and F_{db} are the frequency magnitude for I_q and I_{db}, as defined in equation (3).

We fuse the two similarity, i.e., geometry similarity S_g as computed in (2) and boundary similarity S_b, with a lineaer combination: $S(I_q, I_{db}) = S_g(I_q, I_{db}) + \lambda S_b(I_q, I_{db})$. The combine similarity s are used to rerank the top results.

5. Experiments

We have provided some video demos at `http://www.ee.columbia.edu/~jh2700`, to demonstrate the end-to-end mobile product search system that has been operational over actual mobile networks. Our system will have the same speed in most of the steps as other mobile visual search system, e.g, local feature extraction on mobile (< 1s), transmitting query data through network (usually $<$ 1s). The step of compressing each local features to hash bits is very fast, and can almost be ignored, compared to other steps.

The main difference between our approach and other mobile product search system is the searching step: searching with bag of hash bits v.s. searching with bag of words. To further justify our approach, we conduct experiments to provide quantize analysis on this searching step. Since the whole searching step only involves computation on the server and does not involve mobile, our experiments are conducted on a workstation. We will report both search accuracy and search time in our experiments. Note that the time of other steps (e.g., feature extraction, transmission) are independent of the database size. The database size will only affect this searching step. So the searching time represents the scalability of the system.

5.1. Data Sets

The existing product (or object) image sets, e.g., "Stanford Mobile Visual Data Set" [2], or "UKBench" object data set [12], usually have a small scale like thousands of images. or contain mainly 2D objects like CD/DVD/Book covers.

For mobile product search applications, the most meaningful data set may actually come from online shopping companies like Amazon, Ebay, etc. So we have collected two large scale product sets from Amazon, Zappos, and Ebay, with $300k - 400k$ images of product objects, from diverse categories like shoes, clothes, groceries, electrical devices, etc. These two product image sets are the largest and most challenging benchmark product datasets to date.

Product Data Set 1
The first data set includes 360K product images crawled from Amazon.com. It contains about 15 categories of products. For this data set we have created a query set of 135 product images. Each query image will have one groundtruth image in the database, which contains exactly the same product as the query, but will differ in object sizes, orientations, backgrounds, lightings, cameras, etc.

Product Data Set 2
The second data set contains 400K product images collected from multiple sources, such as Ebay.com, Zappos.com, Amazon.com, etc, and with hundreds of categories. For this data set we have collected a query set of 205 product images.

The image sizes for both sets are usually 200-400 by 200-400. And each image usually contains about 50-500 SURF features. No subwindow is provided for each query image. And moreover, the boundaries for product objects in both database and queries are extracted by automatic SaliencyCut.

Some examples of query images and their groundtruths from our data sets are shown in Figure 4.

Figure 4. Some example queries and their groundtruths in Product Dataset 2. The first row are queries, and the second row are corresponding groundtruths.

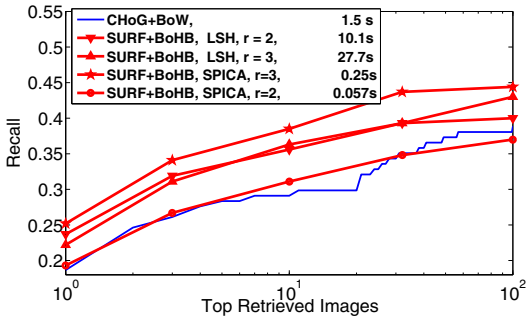

Figure 5. Performance comparison between CHoG based approach and our BoHB approach on Product DataSet 1. For BoHB, we have tried bits from two different hash methods, SPICA hashing [9] and LSH. Search time of different approaches are included in the legends. Graph is best viewed in color.

5.2. Performance of Bag of Hash Bits

First, we compare our "bag of hash bits" approach to CHoG [4, 7] approach, which is the state-of-the-art in mobile visual search area, on Product Data Set 1.

In CHoG implementation, each CHoG feature is about 80 bits. The CHoG features will be quantized with vocabulary tree [12], which is the state-of-the-art method to quantize local features. Since the quantization step for CHoG approach is done on the server side, using large scale codebook is possible. In our implementation, we use a codebook with 1M codewords.

For our "Bag of Hash Bits", we use 80 bits for each local feature by using SPICA hashing[9] or LSH[5]. And then we build 12 hash tables, each of which is constructed by randomly choosing 32 bits from the 80 bits. We will check buckets within hamming distance r in each hash table. r is set to 1-3 in our experiments.

As shown in figure 5, with bits generated from SPICA hashing and hamming distance $r = 2$, our approach of "Bag of Hash Bits" on surf features can obtain about the same recall as BoW on "CHoG" features, but the search speed is improved by orders of magnitude. And if we set $r = 3$, we can improve the accuracy significantly, e.g., about 30% improvement for the recall at top 10 retrieved results, while be several times faster.

However, if we use bits from LSH, the search time of our BoHB approach will be increased by tens/hundreds of times. The main reason is: LSH bits are randomly generated and hence are quite redundant. That actually explains why previous hashing based systems (usually utilizing LSH bits) perform quite poorly.

We also compare our "bag of hash bits" approach to other popular visual search systems, such as BoW via vocabulary tree[12] or product quantization [11], with SURF features, even though they may not be applicable to mobile visual search scenarios.

For vocabulary tree implementation, we follow the paradigm in vocabulary tree in [12]. The codebook size on SURF features is up to 1M, which is the largest codebook in the current literatures. As shown in figure 6, with bits generated from PCA hashing (PCH), and hamming distance $r = 1$ or 2, the proposed BoHB approach can achieve 2-3 fold increase in terms of recall. For product quantization approach, we utilize the product quantization method to match top K nearest neighbor local features for each query local feature, and then rerank the candidate database images. In our current implementation, $K = 1000$, and reranking is based on the counts of matched local features in each candidate images. [2] We can see that product quantization method achieves relatively good accuracy (slightly lower than our top results), but (much) slower search speed. However, if LSH or ITQ[8] hash bits are utilized, we will see the search time of the BoHB approach will be quite long. This confirms the merit to combine compact hashing of low redundance (like PCA hashing or SPICA hashing) with the proposed Bags of Hash Bits idea.

5.3. Performance of Boundary Reranking

If we repeat our BoHB approach with SPICA or PCH bits and r = 3 as in figure 5, but include boundary reranking, the recall of top 100 results will have about relatively 8%-10% improvement. The improvement caused by boundary reranking seems good, but not exciting. However, that is mainly due to the strict definition of our groundtruth. For each query, we only define one groundtruth, which is exactly the same product as the query. Some examples of search results without/with boundary reranking (BR) are shown in figure 7. As shown, our boundary reranking is very helpful to filter out noisy results, and improve the search quality, especially increase the number of relevant products (in the sense of the same category). But this is not represented in recall, under the strict definition of our groundtruth. However, the advantage of boundary reranking will be very helpful in practice, especially when the

[2] We have tried other K and different reranking methods e.g., build BoW histograms with matched local features and compute cosine similarity between histograms, or sum the distances between query local features and matched local features in each candidate image. We choose the current K and ranking method, because they provides the best accuracy.

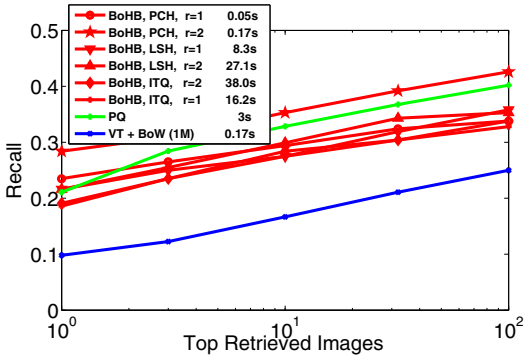

Figure 6. Performance comparison between the proposed BoHB approach and other visual search methods including BoW (with Vocabulary Tree) approach and product quantization approach, on Product DataSet 2. For BoHB, we have tried bits from different hash methods, such ass PCA hashing (PCH), ITQ hashing[8] and LSH. Search time of different approaches are included in the legends. Graph is best viewed in color.

query product does not exist in the database, and hence relevant/similar products will be the best we can return.

Figure 7. Example queries and their top 5 search results, without or with boundary reranking (BR). Pictures are best viewed in color.

6. Discussions and Conclusions

In this paper, we have proposed a new paradigm for mobile product search. Our approach utilizes a new idea of searching with bag of hash bits, and reranking with boundary features. The proposed approach significantly outperforms state-of-the-art mobile visual search systems and several popular conventional visual search methods.

However there are still several main reasons to cause failures of our systems: drastic change in camera perspective and/or lighting, too small image/object size, non-rigid objects, insufficient(or non-discriminative) local features, etc.

Some of our future works include: exploring better features and hash bits to further improve search performance, indexing images directly instead of local features to reduce the server memory, and extending the BoHB method to other multimedia search tasks like audio or video search, etc.

7. Acknowledgement

We thank Wei Li, Yunting Xiao, and Ellen Yi-Lun Wu for their help in this project.

References

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *ECCV*, 2006.

[2] V. Chandrasekhar, D. Chen, S. Tsai, N. Cheung, H. Chen, G. Takacs, Y. Reznik, R. Vedantham, R. Grzeszczuk, J. Bach, et al. The stanford mobile visual search data set. In *ACM conference on Multimedia systems*, 2011.

[3] V. Chandrasekhar, M. Makar, G. Takacs, D. Chen, S. Tsai, N. Cheung, R. Grzeszczuk, Y. Reznik, and B. Girod. Survey of sift compression schemes. In *Proceedings of International Mobile Multimedia Workshop (IMMW)*, 2010.

[4] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod. Chog: Compressed histogram of gradients a low bit-rate feature descriptor. In *CVPR*, 2009.

[5] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

[6] M. Cheng, G. Zhang, N. Mitra, X. Huang, and S. Hu. Global contrast based salient region detection. In *CVPR*, 2011.

[7] B. Girod, V. Chandrasekhar, D. Chen, N. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S. Tsai, and R. Vedantham. Mobile visual search. *Signal Processing Magazine, IEEE*, 2011.

[8] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[9] J. He, S. Chang, R. Radhakrishnan, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, 2011.

[10] J. He, T. Lin, J. Feng, and S. Chang. Mobile product search with bag of hash bits. In *Demo session of ACM MM*, 2011.

[11] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI, IEEE Transactions on*, 2011.

[12] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[13] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics*, 2004.

[14] M. Yang, K. Kpalma, J. Ronsin, et al. A survey of shape feature extraction techniques. *Pattern Recognition*, 2008.

[15] F. Yu, R. Ji, and S. Chang. Active query sensing for mobile location search. In *ACM MM*, 2011.