Energy Conservation by Adaptive Feature Loading for Mobile Content-Based Image Retrieval

Karthik Kumar, Yamini Nimmagadda, Yu-Ju Hong, and Yung-Hsiang Lu
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907
{kumar25,ynimmaga,hong41,yunglu}@purdue.edu

ABSTRACT

We present an adaptive loading scheme to save energy for content based image retrieval (CBIR) in a mobile system. In CBIR, images are represented and compared by high-dimensional vectors called features. Loading these features into memory and comparing them consumes a significant amount of energy. Our method adaptively reduces the features to be loaded into memory for each query image. The reduction is achieved by estimating the difficulty of the query and by reusing cached features in memory for subsequent queries. We implement our method on a PDA and obtain overall energy reduction of 61.3% compared with an existing CBIR implementation.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; C.4 [**Performance of Systems**]: Design studies

General Terms

Algorithms, Design, Measurement, Performance

Keywords

mobile content-based image retrieval, energy saving, adaptive feature reduction, similarity index

1. INTRODUCTION

Most mobile systems, such as cellular phones and PDAs, have built-in cameras. With the improvement in storage technology, it is possible to store several gigabytes of images on these systems. This results in a need for better organization and retrieval in these image collections. Retrieval by image names becomes difficult as the number of images increases, and often the names may not describe the contents of the images. These systems usually have miniature keyboards and keyword-based search is inconvenient. Content-Based Image Retrieval (CBIR) provides a different, possibly more convenient approach to find images. CBIR is performed by analyzing images and extracting their features. These features are represented by high-dimensional numerical "feature vectors", and comparisons between images are performed by matching their feature vectors.

Existing CBIR assumes that the mobile systems are "thin clients"— they transmit query images to servers. Feature extraction and matching are performed at the servers. However, the mobile systems' large collections of images may be unavailable at the servers. Transmitting several gigabytes of images to servers through wireless networks for searching one image is unrealistic. Moreover, the mobile systems may not have access to wireless networks (for example, inside a national park). Hence, it is desired to be able to perform CBIR on mobile systems themselves.

Existing CBIR implementations load the entire image collections into memory (RAM) before comparing the features. The mobile systems have limited resources: energy, processor speed, and memory. Even though the capacities of flash memory have been steadily rising in the past few years, most mobile systems have no more than 64 MB memory. This is because energy is limited, and the amount of power needed to hold data in RAM increases with the size of RAM. Flash memory requires almost the same amount of power regardless of sizes. Large image collections can be stored in the flash memory, but current technology does not permit these images to be accommodated into RAM. Even if a mobile system has sufficient RAM, loading the entire image collection into RAM takes time and energy. Thus, it is necessary to develop the technology for CBIR on mobile systems with large flash memory and much smaller RAM. This paper uses "memory" and "RAM" interchangeably; we use "flash" for non-volatile solid-state storage.

This paper presents a method to save energy for mobile CBIR by adaptively selecting the features to be loaded into memory. The features are selected based on (1) the estimated difficulty of the query and (2) the features of previous queries. A search is "difficult" if the query image is similar to many images in the collection; more features are needed to distinguish the images. This consumes time and energy. If a search is easier, fewer features are loaded into memory for comparison; this saves time and energy. Furthermore, we cache features in memory for subsequent similar queries to save energy.

We implement our algorithm on an HP iPAQ 6945 PDA with a 416 MHz Intel XScale processor and 64 MB of mem-

ISLPED'08, August 11–13, 2008, Bangalore, India.

ory, using a 2 GB mini SD flash card for storage. We perform CBIR with different sizes of image collections, different query images and different numbers of image queries. Our system saves up to 61.3% energy for a single image query, while maintaining an accuracy of 80% for a collection of 5000 images.

2. RELATED WORK

CBIR has been widely studied [2]. A typical CBIR algorithm extracts features from all the images offline and stores these features in index files on disks or in flash. We choose ImgSeek, an open source CBIR algorithm based on [6] as the basis for our work. ImgSeek uses a feature vector of 60 Haar wavelet coefficients for painted images and 40 coefficients for scanned images. Most CBIR programs load the entire index files into memory before comparing the query image's features. CBIR requires large amounts of memory and computation in order to return accurate results.

It becomes important to analyze the memory performance of CBIR because of the latency involved in accessing secondary storage such as flash memory. Forster in [3] suggests using "filtered retrieval" to reduce time and RAM requirements. Robles et al. in [8] present a parallel implementation of CBIR designed for a shared memory system. Since CBIR involves finding similar images, it is important to analyze and quantify accuracy in CBIR. Rudinac et al. in [9] analyze the effects of the lengths of feature vectors and observe that shorter vectors do not necessarily degrade accuracy. Gunther et al. in [5] suggest a methodology for developing test images in CBIR. They systematically modify test images which differ in terms of shape, color, texture etc. French et al. in [4] give a scoring scheme for query results in CBIR.

All existing mobile CBIR systems examine the use of client-server architecture. The image is captured on the mobile device (client) and transmitted to a server. The server then performs CBIR and returns the results to the client. Ahmad et al. [1] present a Java-based framework that uses a client to capture the image and a server to perform matching. Yeh et al. [10] propose a system that first queries an image database (on a server) by using a photo from a mobile device. After a match is found, the system then performs a Google image search using the keywords obtained from the initial query. Jia et al. [7] propose an architecture to query the web directly using images taken from mobile devices.

In this paper: (1) We present a system in which image retrieval is performed *entirely* on the mobile device. The images are captured, and the feature extraction and search are performed on the device itself. (2) Energy is limited in mobile devices. Our system is the first to characterize the relation between accuracy and energy for CBIR on the mobile system. We adaptively reduce the computation required for CBIR to save energy. (3) We analyze and quantify the features loaded into memory for CBIR. In mobile devices, the features are stored in flash and the energy required to read from flash into memory becomes significant. We adaptively reduce the features that are loaded in memory to save energy. (4) Our system is the first to analyze the energy consumption over multiple queries. We show an energy-efficient CBIR implementation by reusing features in memory.

3. ENERGY CONSERVATION IN MOBILE CBIR

This section first characterizes the energy consumption of ImgSeek. ImgSeek loads all features into memory before search. Our method saves energy by making the following contributions: (1) The algorithm does not load the whole index file at once. Instead, the indexes are loaded as needed. (2) The lengths of features used for comparison are adaptively determined based on the similarity between the query image and the image collection. (3) For multiple queries, some indexes may have already been loaded and additional indexes are loaded only when necessary.

3.1 Modeling ImgSeek's Energy Consumption

We use ImgSeek as an example for our analysis, but our work is general and can be applied to other CBIR algorithms. For CBIR, a feature vector containing m elements [f_1, f_2, ..., f_m] is used to represent an image in each color channel (40 is suggested for m in [6]). Each of these elements α ($1 \leq \alpha \leq m$) can assume a value in the range [l_α, h_α]. ImgSeek uses [l_α, h_α] = [1, 16384] for all α. Let i be an image in the collection I, $i \in I$, with feature vector $f(i)$. Similarity between images is given by the number of elements in the feature vector that are common between them. Inverted indexes are used to indicate the set of images with a particular value in the features. Let $R(v)$ be the set of images whose features have an element of value v:

$$R(v) = \{i \in I : \exists s, 1 \leq s \leq m, f_s = v, f_s \in f(i)\}. \quad (1)$$

Thus each feature v has a set of images which contain v as part of their feature vectors. The inverted indexes are calculated *offline* for the entire image collection and stored in flash memory. When the CBIR application starts, all the features are loaded into memory from flash and consume energy $E_d(|I|)$, where $|I|$ is the size of the image collection. When an image is queried, its top m features are extracted online and consume energy E_q. The inverted indexes are used to assign scores to all the images that share each of the m features with the query image; this consumes energy $E_s(|I|)$. The value of m is fixed and denoted by m_o (= 40). The energy consumed by CBIR can be modeled as $E_d(|I|) + E_q + E_s(|I|)$. For n queries the energy consumed by CBIR is

$$E_{cbir} = E_d(|I|) + n \times (E_q + E_s(|I|)). \quad (2)$$

The energy to load the indexes is consumed only once. ImgSeek loads *all* features to memory *before* search and reduces the time a user has to wait for the results after a query. This is showed as t_{cbir} in Figure 1(a). Two problems arise when the entire indexes are loaded into memory. First, the available memory in a mobile system is limited. Second, even if a mobile system has a sufficient amount of memory, loading the indexes takes time and consumes energy.

3.2 Adaptive Index Loading

We propose to load only the features that are relevant to the query image *after* the query image is selected and its features are extracted. In our method, loading indexes and performing comparison are interleaved, as shown in Figure 1(b). The energy consumption is expressed as E_{int} and it

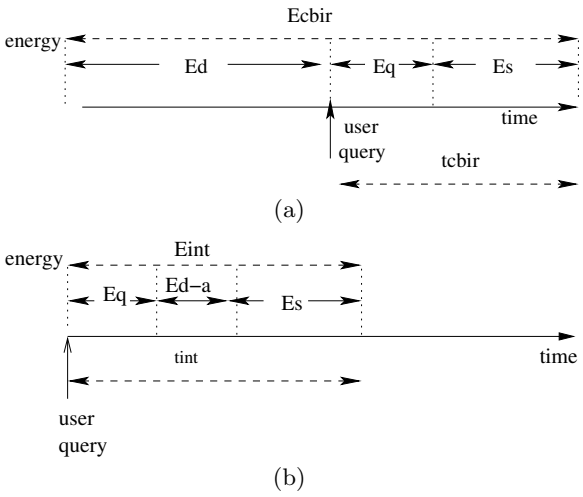

Figure 1: Energy and time in CBIR- (a) existing implementation is compared with (b) our interleaving scheme.

includes three terms: (1) E_q to extract the query image's features, (2) $E_{d-a}(m, |I|)$ to load the features needed to compare the images from flash to memory, and (3) $E_s(m, |I|)$ to perform comparison. Here, m is the number of the features for each image and $|I|$ is the size of the image collection. For comparing n images, E_{int} is

$$E_{int} = n \times (E_q + E_{d-a}(m, |I|) + E_s(m, |I|)). \quad (3)$$

The additional delay of this method is t_{int} - t_{cbir}; our measurements show that the delay is less than 2 seconds for 14,000 images and acceptable for most users. The energy consumption compared with the original implementation is the ratio of (3) and (2):

$$\frac{E_{int}}{E_{cbir}} = \frac{n \times (E_q + E_{d-a}(m, |I|) + E_s(m, |I|))}{E_d(|I|) + n \times (E_q + E_s(|I|))}. \quad (4)$$

A smaller ratio means less energy consumption by interleaving and thus more energy saving.

3.2.1 Selective Loading

In the first improvement, the feature length m is fixed equal to m_o. Equation (4) is simplified as

$$\frac{E_{int}}{E_{cbir}} = \frac{k + n \times E_{d-a}(|I|)}{k + E_d(|I|)}, \quad (5)$$

here $k = n \times (E_q + E_s(|I|))$. The value n means the number of queries and it signifies the energy reduction by loading only the features that are relevant to each query image. It indicates the number of *additional* searches to perform until this interleaving scheme consumes the same amount of energy as loading all the features at once before search.

3.2.2 Adaptive Feature Length

The next improvement is achieved by reducing the number of features for comparison. We can estimate the "difficulty"

Figure 2: Test images used (a) identical, (b) blocks of 4 pixels averaged, (c) gray scale, (d) inverted colors, (e) abstract mimic response, (f) blocks of 4 pixels shuffled.

of a query based on the similarity between the query image and the images in the collection. A search is more difficult if many images are similar to the query image and more features are needed to discern the closest matches. If the query image is dissimilar to most images, the closest matches may be found using fewer features. We develop "similarity index" ρ. A smaller ρ means the search is easier. Let f denote the complete m_o features of the query image i, $f_j \in f(i)$. The similarity index is defined as

$$\rho = \frac{\sum_{1 \le j \le m_o} |R(f_j)|}{m_o |I|}, \quad (6)$$

here $|R()|$ is the number of images that have at least one identical feature as the query image, as defined in (1). To reduce the runtime overhead in computing ρ, $\frac{R(v)}{m_o|I|}$ can be calculated in advance for each v. Even if the image collection spans all possible features, storing $\frac{R(v)}{m_o|I|}$ will require only $16384 \times 2 = 32$ kB (16384 features, 2 bytes per feature, for up to $2^{16} - 1 = 16535$ images).

We experimentally determine the relation between the similarity index ρ and the minimum feature length required to give an accuracy of 80%. Accuracy is defined as follows: We use p to quantify the quality of the search results and p is the sum of $p1$ and $p2$. For a query image, A is a set of similar images. The top k ($k > |A|$) images that are returned by the CBIR algorithm are considered for calculating the accuracy. Images that are ranked $k+1$ or worse are ignored. The top k ranked images form a sequence [u_1, u_2, ..., u_k]. If an image in A is ranked γ ($1 \le \gamma \le k$), then γ is added to $p1$. The $\delta(b)$ function is one if b is true and zero if b is false.

$$p1 = \sum_{i \in A} \gamma \times \delta(i = u_\gamma). \quad (7)$$

If an image in A is not among the top k images, a penalty τ is added to $p2$. This penalty τ has to be larger than k in order to reflect missing an expected image.

$$p2 = \sum_{i \in A} \tau \times \delta(\neg(\exists \gamma, 1 \le \gamma \le k, i = u_\gamma)). \quad (8)$$

$$p = p1 + p2. \quad (9)$$

A smaller p means better accuracy. The accuracy ratio due to using m features instead of m_o is given by the ratio of their p values ($= \frac{p_{m_o}}{p_m}$).

In our experiments, we use 6 images for A based on the suggestion in [4]. These images are created by processing the query image with the following methods: (1) identical,

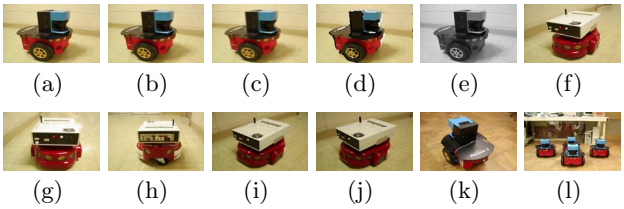

Figure 3: Accuracy criterion for sample results using feature length m_o. The desired results are (a), (b), (c), (d), and (e) (from the set of test images introduced in the image collection). The other images are considered unwanted.

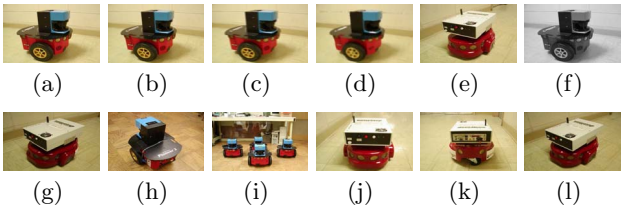

Figure 4: Accuracy criterion for sample results using feature length $m = 20$. The images (a), (b), (c), (d), and (f) are the desired results. The orders of the unwanted images have no effect in calculating the accuracy.

(2) blocks of 4 pixels averaged, (3) gray scale, (4) inverted colors, (5) abstract mimic response, and (6) blocks of 4 pixels shuffled. One set of test images used is shown in Figure 2. Only the top 12 results ($k = 12$) are considered because they can fit in the screen of the PDA. The penalty τ we use is 13. Based on this accuracy metric, we use the p with 40 features as the basis.

An example of our accuracy calculation is shown in Figures 3 and 4. Figure 3 shows the results using feature length m_o; (a), (b), (c), (d) and (e) are the desired results from the set A. The value of p1 is the sum of their ranks $1+2+3+4+5 = 15$ and p2 is 13. Thus $p_{m_o} = 15+13 = 28$. Similarly, from Figure 4 we obtain p1 to be $1+2+3+4+6 = 16$ and p2 to be 13, and $p_m = 16 + 13 = 29$. The accuracy ratio $= 28/29 = 96\%$.

We perform experiments by using different query images in various image collections containing 1000 to 10000 images. From our experiments, we find that for a given value of ρ, the number of features needed can be calculated using this formula:

$$m = 2400\rho^3 - 1800\rho^2 + 460\rho - 14 \qquad (10)$$

to achieve at least 80% accuracy. As can be seen in Figure 5, for a database with similar images ($\rho = 0.4$), 30 more features are needed to achieve the accuracy when compared to a database with dissimilar images ($\rho = 0.03$).

The observations from the previous paragraphs can be summarized as follows: (1) The similarity index provides a quantitative measure between the query image and the images in the database. (2) Fewer features are needed to achieve the desired accuracy if the similarity index is low. Our system utilizes these observations to save energy. The features of the images in the database have been constructed before a query. When a query image is received, our system first calculates the complete m_o features of the image and the similarity index ρ. The actual length m of the features is obtained by equation (10). The user has an option to add the query image into the database so that the new image is also compared in the next query. No additional computation is needed for the query image since its full-length features have already been computed.

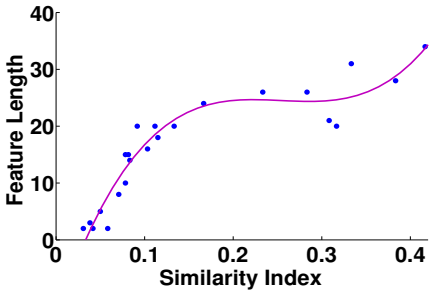

Figure 5: The minimum lengths of features to achieve 80% accuracy for different similarity indexes

The complete features of our image database are organized into 6 segments, each with 10 elements. Our system performs CBIR using the least number of segments needed, i.e. $10 \times \lceil \frac{m}{10} \rceil$. The energy improvement due to selective loading can be obtained from equation (4). If the user is not satisfied with the search results, the user can raise the accuracy requirement to improve the search. An additional segment of features is used to improve the previous search results.

3.2.3 *Handling Multiple Queries*

Consecutive query images are likely to be similar to each other, and this similarity is called temporal locality. Images that are taken around the same location may exhibit spatial locality for the same reason. In the third improvement, we exploit both temporal and spatial locality for multiple queries to save energy. Figure 6 shows that a user walks towards a float in a parade and sends three query images. These images share some features (since the background and the float are the same); this scenario is similar to keyword-based search when a user refines the search. For a query image, if some of the required features are already loaded into memory, these features will not be reloaded from flash again. Thus, the features residing in memory gradually increase with the number of queries. Eventually, the memory is full and some features must be evicted. We suggest using the LRU replacement policy but this situation does not occur in our experiments. We explore up to 5 successive queries in our experiments.

4. EXPERIMENTS

4.1 Measurement Setup

We port ImgSeek to an HP iPAQ 6945 PDA as the baseline for comparison. Some of the sample results returned

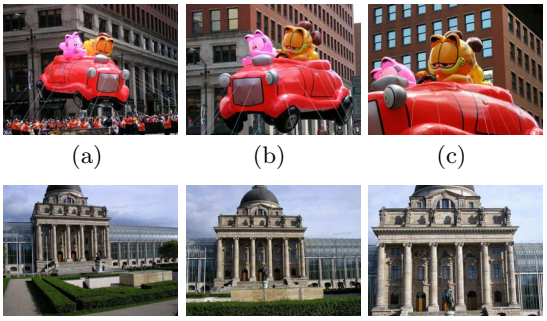

Figure 6: Multiple query images taken as a user approaches (1) a float in (a)-(c) and (2) a building in (d)-(f). In both cases, for the third query image, 35% of the required indexes are already loaded for the previous queries.

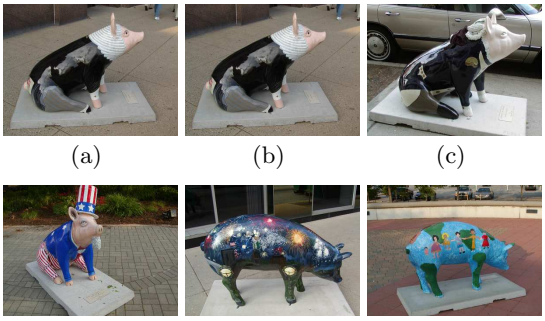

Figure 7: Results returned by CBIR. (a) is the query image and (b)-(f) are the top results returned.

by ImgSeek are shown in Figure 7. We use the experimental setup shown in Figure 8. The current drawn from the the battery is obtained by measuring the voltage across a 0.25 Ω resistor. The voltage is sampled at 100 kHz using a National Instruments data acquisition card installed on a separate computer. The PDA has a 2 GB flash memory card to store 14000 images downloaded by crawling the Internet. The average size of these images is 28 kB and they occupy around 400 MB in flash memory. The PDA has a built-in 1.3-Megapixels camera and the average size of an image taken on the PDA is 150 kB. If all images are taken by the PDA's camera, only 14000 images can be stored on the flash. We use different sizes of image collections from 1000 images to 14000 images in our experiments. The features are extracted from these collections offline and stored in flash memory. The features for a collection of 14000 images occupy 28 MB. This is because the features are arranged using inverse indexes, and $R(v)$ contains a list of image IDs for each feature v in each color plane. Each image ID occurs 40 times per color plane, with 3 color planes, and with an average of 8 characters (and 2 bytes per character) for each image ID (including the path), each image occupies a total of $40 \times 3 \times 8 \times 2 = 2$ kB, and 14000 images occupy around 28 MB.

4.2 Energy Savings

We measure the energy for the following schemes: s1- existing implementation of ImgSeek, s2- selective loading, s3- selective adaptive loading, and s4- selective adaptive loading with caching. It is noted that adaptive loading results in some loss of accuracy. In our experiments we observe that the accuracy remains above 80% for adaptive loading using the accuracy criterion defined in Section 3.2.2. Our measurements are performed for various query images. The query images are selected at random from the image collection stored on the flash. Six images are obtained from each query image by the modifications suggested in Section 3.2.2. We perform our experiments on image collections of different sizes from 1000 to 14000 images. The additional delay $(t_{int} - t_{cbir})$ of s2, s3, and s4 is around 0.8 to 1.2 seconds.

Figure 8: (a) CBIR implemented on HP iPAQ 6945 (b) Energy measurement.

4.2.1 *Energy for a Single Query*

Figure 9 shows the total energy required to perform CBIR using s1, s2, and s3. Note that s4 saves energy only when handling multiple queries, so we show the results in the next subsection. For the existing implementation of ImgSeek, the PDA does not have sufficient memory to perform CBIR as the size of the image collection approaches 10000 images. The size of the feature index for 10000 images is 20 MB and we find that it does not fit in the program memory available in the HP iPAQ. With selective loading, the other schemes can be used for larger image collections. Both s2 and s3 demonstrate significant amounts of energy savings compared to s1. The schemes save 53 - 61.3% energy for the collection of 5000 images. This is the total system energy measured from the battery.

4.2.2 *Energy for Multiple Queries*

Figure 10 shows the energy consumption ratio $\frac{E_{int}}{E_{cbir}}$ defined in equation (4) for s2, s3, and s4 over s1 for a collection of 5000 images. When fewer features are in memory because of adaptive loading, energy savings span from 53% for a single query to 9% for five queries for s2. For s3, on average only 24 features are loaded into memory. The energy required to search also reduces because fewer features need to be compared. Additional energy is saved by s4 because even fewer features are loaded into memory as a result of caching. We observe 25% similarity on average between features of consecutive searches. For 24 features, this results in 18 features being loaded from flash, and 6 features reused

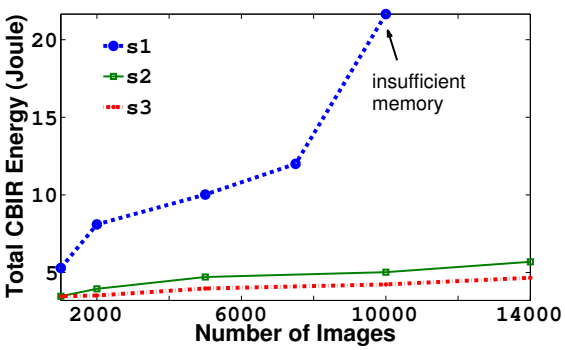

Figure 9: Energy consumption of s1, s2 and s3 for mobile CBIR for a single query image for different sizes of image collections.

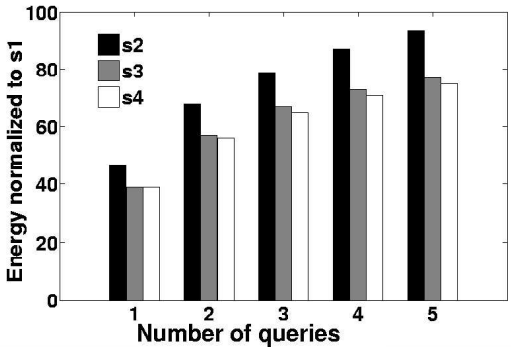

Figure 10: Energy consumption for mobile CBIR for different number of queries. Shorter bars mean more energy savings and are desirable.

in memory. The additional energy savings offered by s4 is small (0.08 J per query). This is because s3 already uses a small feature length. For s4, the features to be loaded into memory are further reduced due to caching. The overhead in accessing flash dominates and hence there is less room for additional savings. Moreover, we use random images for successive queries (both similar and dissimilar). Adaptive loading in s3 results in lower accuracy. If degraded accuracy is undesirable, s2 may be extended to include caching with the same number (m_o) of features.

As the number of queries increases, the energy consumption by s2 - s4 increases. For example the energy consumption by s3 (relative to s1) increases from 38.7% for a single query to 78% for five queries. In s1, all the features are loaded in memory at the beginning and no features need to be loaded between queries. In s2 - s4, a small subset of relevant features are loaded for each query. For a large number of queries, more energy is consumed to load the features for each query and the energy ratios increase. It may become more energy efficient to load all features into memory at once, but this is possible only if all the features can fit in memory. We extrapolate our measurements and find that s1 becomes more energy efficient than s2 after eight queries; and more efficient than s3 and s4 after more than 30 queries.

5. CONCLUSION

We present an adaptive loading method to save energy for performing CBIR on a PDA. The method selects the features to be loaded into memory, thus reducing the energy consumption while maintaining 80% accuracy. We implement our method on an HP iPAQ 6945 PDA and show that we save 61.3% energy for a collection of 5000 images.

6. ACKNOWLEDGMENTS

This project is supported in part by NSF CCF-0541267. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the sponsor.

7. REFERENCES

[1] I. Ahmad and M. Gabbouj. Compression and Network Effect on Content-Based Image Retrieval on Java Enabled Mobile Devices. In *Finnish Signal Processing Symposium*, pages 35–38, 2005.

[2] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Computing Surveys*, 40:5:1–5:60, April 2008.

[3] J. Forster. Reducing Time and RAM Requirements in Content-Based Image Retrieval using Retrieval Filtering. In *Informatiktage*, pages 143–146, 2007.

[4] J. C. French, W. N. Martin, and J. V. S. Watson. A Qualitative Examination of Content-Based Image Retrieval Behavior using Systematically Modified Test Images. In *Midwest Symposium on Circuits and Systems*, pages 655–658, 2002.

[5] N. J. Gunther and G. Beretta. A Benchmark for Image Retrieval using Distributed Systems over the Internet:BIRDS-I. In *SPIE - Internet Imaging II*, pages 252–267, 2001.

[6] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast Multiresolution Image Querying. In *International Conference on Computer Graphics and Interactive Techniques*, pages 277–286, 1995.

[7] M. Jia, X. Fan, X. Xie, M. Li, and W.-Y. Ma. Photo-to-Search: Using Camera Phones to Inquire of the Surrounding World. In *International Conference on Mobile Data Management*, pages 46–46, 2006.

[8] O. Robles, J. Bosque, L. Pastor, and A. Rodriguez. Performance analysis of a cbir system on shared-memory systems and heterogeneous clusters. In *Computer Architecture for Machine Perception*, pages 309–314, 2005.

[9] S. Rudinac, G. Zajic, M. Uscumlic, M. Rudinac, and B. Reljin. Comparison of cbir systems with different number of feature vector components. *Semantic Media Adaptation and Personalization*, pages 199–204, 2007.

[10] T. Yeh, K. Tollmar, and T. Darrell. Searching the Web with Mobile Images for Location Recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 76–81, 2004.