SnapFind: Brute Force Interactive Image Retrieval

Larry Huston
Intel Research Pittsburgh
larry.huston@intel.com

Rahul Sukthankar
Intel Research Pittsburgh
rahul.sukthankar@intel.com

Derek Hoiem
SCS, Carnegie Mellon
dhoiem@cs.cmu.edu

Jiaying Zhang
EECS, University of Michigan
jiayingz@engin.umich.edu

Abstract

SnapFind is an image retrieval system that enables efficient interactive search of large data sets by exploiting active disk technology. In contrast to earlier approaches, where data is typically pre-indexed for efficient retrieval according to a fixed scheme, SnapFind provides users with the flexibility to search non-indexed data in a brute force manner. The query is translated into a customized searchlet that is executed in parallel by processors near the storage devices. This enables the majority of irrelevant images to be discarded where they are stored. Partial results are displayed during search execution allowing users to interactively refine the query without waiting for search termination. This paper argues that algorithms with user-adjustable parameters are preferable to black-box image retrieval techniques.

1. Introduction

Content-based image retrieval (CBIR) research has made significant progress in the last decade [13]. Early work (*e.g.*, Virage [2], QBIC [6]) focused largely on whole-image searches, where data was typically processed offline and compactly represented as a multi-dimensional vector. Alternately, images were indexed offline into several semantic categories [17]. Those systems enabled interactive queries in a computationally-efficient manner on computers with limited processing power; however, they could not support queries about local regions *within* an image since indexing sub-regions within an image would be prohibitively expensive. Although whole-image searches are well-suited to queries corresponding to general image content (*e.g.*, "sunsets"), they are poor at recognizing objects that only occupy a portion of the image (*e.g.*, "people wearing jeans").

As computers become more powerful, region-based image retrieval [3, 14, 16] is becoming more feasible. In this approach, portions of the image are characterized by features such as color, texture, shape and position. Region-based image retrieval addresses many of the problems with whole-image retrieval, but the search results can be biased by the method used to partition the image into regions – and this decomposition is often static for performance reasons.

Object-based image retrieval is an alternate approach to extracting semantic content within images. Motivated by research in face and object detection [15], this approach performs a windowed scan over the entire image and identifies regions that match are flagged by a pre-trained classifier. Such an approach works best when the classifier has been trained with a large amount of training data, but recent results indicate that it may also work with smaller numbers of training examples, when augmented by user feedback [8].

Despite these technological improvements, the semantic gap [5] between the user's needs and the capability of CBIR algorithms remains significant. Relevance feedback [11] and active learning [4] are two attempts at extracting some of this implicit semantic content from the user. In the former, the retrieval system changes its behavior based on positive and negative feedback provided by the user on partial query results. In the latter, the system identifies data that could particularly improve accuracy and asks the user to label this additional data. Both of these approaches allow the user to influence the search in an implicit rather than an explicit manner.

This paper proposes a brute-force approach to interactive search that exploits active disk technology [1, 12] with flexible algorithms that provide explicit user control. Our application, SnapFind, enables users to examine partial results and iteratively refine their searches.

2. Interactive Brute-Force Search

For the foreseeable future, we believe that the user will remain a key component of any image retrieval system. However, image processing can eliminate a large fraction of irrelevant data, utilizing the user's limited attention more effectively.

In current systems, the user is typically limited to two options. The first is to manually search through all of the images. This ensures a high recall rate, but is extremely time-consuming. The other approach, advocated by many content-based image retrieval systems, is to index the data on pre-computed search criteria. This allows fast retrieval, but if the pre-computed criteria do not fit the user's needs, the user can do little to find the desired images. Our goal is to build an interactive search system that explores the space between these two extremes. Such a search system should leverage the user's understanding of the semantic content to drive the search. Specifically, we believe that interactive search should address the following requirements.

- Support flexible setting of algorithm parameters to enable queries that are specialized both to the search criteria and to the data being searched.
- Provide tools and visual feedback to help the user choose the best algorithms and parameter settings for the particular search.
- Show partial results to the user as they become available so the user can refine unsatisfactory queries before all the data is processed.
- The user should be able to tune the precision/recall trade-offs to match the current search requirements. For instance, a security analyst may accept a high false positive rate to minimize the risk of missing an important image.
- The system must not assume specific indices or data representations so that new algorithms can be easily deployed.

Interactive brute-force search has received little attention because of its perceived impracticality for performance reasons. We have built a system, named Diamond, that addresses many performance challenges associated with brute-force search. This paper focuses on SnapFind, an image search application running over Diamond, that meets the requirements outlined above.

3. Diamond

SnapFind uses the Diamond system [9] to efficiently perform brute-force search. Figure 1 illustrates the Diamond architecture. Diamond separates the *front end*, which encapsulates domain-specific application code on the host computer, from the *back end*, which consists

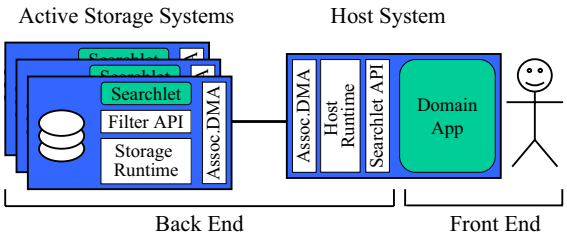

Figure 1. Diamond Architecture

of a domain-independent infrastructure that is common across a wide range of search applications.

A Diamond application runs on the host computer and interacts with the user to formulate a query. Once a query has been formulated, the application translates the query into a set of machine executable tasks (termed a *searchlet*) that the storage devices use to filter out data that does not match the search criteria. The searchlet contains all of the domain-specific knowledge and acts as a proxy for the application (and the user).

The searchlet consists of a set of *filters* and some configuration state (*e.g.*, filter parameters). For example, a searchlet that retrieves images of people in dark business suits could contain two filters: a color histogram filter to find dark regions and a human face detector. Each filter can independently discard an object. Objects that pass all filters are presented to the domain application through the searchlet API.

The domain application may perform additional processing on matching objects such as cross-object correlations or consulting auxiliary databases. Once a domain application determines that a particular object matches the user's criteria, the object is shown to the user.

To increase interactive performance, Diamond caches the results of each filter to speed up subsequent queries. This is implemented by uniquely identifying each filter and its dependencies (attributes and arguments) without explicit support from the application or the application developer. Diamond's cache improves interactive search time by quickly eliminating objects that failed to pass a previously-executed filter and by reusing previously-computed results (*e.g.* output of feature extraction). This form of caching is particularly useful for searches with iterative query refinement.

4. SnapFind

SnapFind (see Figure 2) is a flexible, extensible image retrieval application that supports a variety of algorithms. Its key features are described below.

4.1. Image Retrieval Algorithms

SnapFind is designed to support a wide range of region- and object-based image searches. Adding a new algorithm requires the developer to provide functions for configuring the algorithm and for evaluating each image. SnapFind currently supports the following algorithms.

- **Color Histograms** SnapFind builds region-based histograms for color matching from example image patches provided by the user at query-time. A large set of parameters can be tuned (*e.g.*, number of bins, color space, stride, similarity metrics, etc).
- **Texture filters** SnapFind provides region-based texture filters built from user-provided examples. As with color histograms, users can interactively adjust a wide range of search parameters.
- **Offline-trained Object Detection** SnapFind also supports offline-trained detectors, such as face detection [10, 15]. These detectors are well-suited for identifying semantic content that is not user-specific, but since the precision/recall trade-offs are made at training time, the user has limited ability to adjust their performance. SnapFind can partially mitigate this problem by providing multiple classifiers, each trained with different parameters.

4.2. Search Specification

When starting a new search, the user must create a set of predicates that capture the desired semantic request. A predicate is created by choosing an algorithm and adjusting the appropriate parameters. For example-based algorithms (color or texture), the user provides examples patches by highlighting regions in sample images (from the local disk, the Web, or previous search results).

After the predicates are defined, the user constructs a search by combining these predicates using boolean operators. When the user starts a search, SnapFind generates a searchlet using the search parameters and passes the searchlet to Diamond. Diamond evaluates this searchlet on each object in the data set (typically in parallel, on the active storage system), and returns matching objects to SnapFind where they are displayed to the user.

4.3. Search Refinement

An important feature of SnapFind is its ability to interactively refine a search based on partial results. This is critical because of the difference between a user's needs and the limitations of current technology to automatically identify the desired semantic content. To address this semantic gap, the user is given immediate feedback on the progress of the current query (in terms of matching images). Based on this feedback, the user can adjust the search parameters without waiting for the initial query to terminate. Similar ideas have previously been shown to be successful in the context of interactive data analysis of large databases [7].

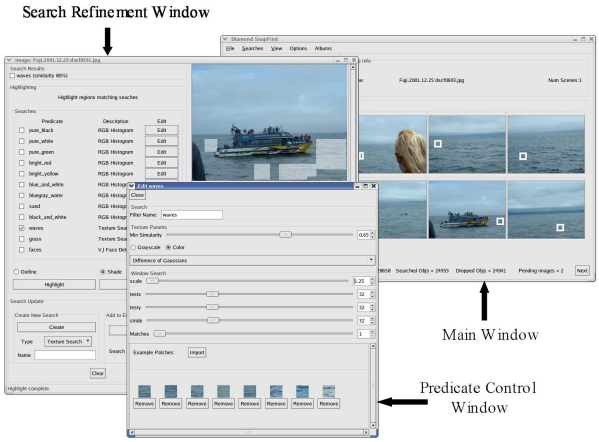

Figure 2. SnapFind screen shot

During the progress of a search, SnapFind presents matching images to the user as they become available. If the search is correct, then the user can scan the results until the desired images are located.

In the more likely scenario, the results will not precisely match what the user had in mind. In this case the user may refine the search by adding predicates or adjusting parameters. A key to successful refinement is providing the user with visual feedback on how any proposed changes could affect the results.

When results are displayed to the user, SnapFind indicates the reason for selecting an image by highlighting those regions that match the current query. Additionally, SnapFind allows the user to evaluate different predicates on sample images (returned results or from other sources) and to visually highlight those image regions that match the current predicate settings. The user can then refine the predicate parameters so that the search can include or omit specific images.

5. Evaluation

This section examines some of SnapFind's retrieval characteristics. First we explore how modifying search parameters affects search results. Next we examine how some basic SnapFind searches compare to Blobworld [3]. We then report experiences on searching a reasonably-large collection of digital images.

Texture	Similarity	Scale Step	Hits
Wave	0.30	fixed-scale	7,731
Wave	0.30	1.2	11,826
Wave	0.60	fixed-scale	1,890
Wave	0.60	1.2	4,209
Wave	0.90	fixed-scale	53
Wave	0.90	1.2	121
Grass	0.30	fixed-scale	2,352
Grass	0.30	1.2	4,707
Grass	0.60	fixed-scale	301
Grass	0.60	1.2	724
Grass	0.90	fixed-scale	4
Grass	0.90	1.2	8

Table 1. Two different texture searches where the normalized similarity threshold and the search scale were varied.

Color	Stride	Scale Step	Hits
Blue-gray	8	fixed-scale	3,953
Blue-gray	8	1.2	4,665
Blue-gray	32	fixed-scale	1,482
Blue-gray	32	1.2	1,819
Striped Shirt	8	fixed-scale	1,658
Striped Shirt	8	1.2	3,267
Striped Shirt	32	fixed-scale	194
Striped Shirt	32	1.2	514

Table 2. Two color searches, where the stride and the scale were varied.

5.1. Effect of Search Parameters

These experiments test the hypothesis (see Section 2) that interactive search systems should provide users with the ability to explicitly control algorithm parameters. Our dataset consisted of 112,404 images gathered from personal photos, commercial image datasets, and images from the web. These images were distributed over 12 storage nodes (1.2 GHz Intel® Pentium® III processors, 512 MB RAM and 73 GB SCSI disks), connected via a 1 Gbps Ethernet switch. The host system contained a 3.06 GHz Intel® Pentium® Xeon™processor, 2 GB RAM, and a 120 GB IDE disk. The storage nodes were connected to the host via 1 Gbps Ethernet.

The first set of experiments examined two texture queries: ocean waves and grass. Texture was modeled using a difference-of-Gaussian filter trained using a handful of example image patches. Table 1 shows the effects of varying two parameters: the normalized similarity threshold, and whether to search at multiple scales.

These results confirm that adjusting the parameters has a significant impact on the number of objects that are returned. The appropriate setting of such parameters are dictated by the needs of the application domain and the user's personal preferences. For instance, someone who quickly wants to find a few images that match a particular criteria would prefer to set the parameters for high precision, while a user who is penalized harshly for false negatives (*e.g.*, homeland security) may set the parameters for high recall to return a large number of images and tolerating many false positives. In addition to this well-understood precision/recall trade-off, SnapFind users are faced with the trade-off between retrieval accuracy and the time spent manually filtering search results.

The second set of experiments examined two color histogram queries: blue-gray ocean water, and a child's striped shirt. Table 2 shows the effects of varying two parameters: the stride, and the scale.

Because the first query has a uni-modal color distribution, a small region sample is likely to match many patches in the entire region at any scale. Since the second query has a multi-modal color distribution, searching at different scales greatly increases the number of matching images, decreasing the false negative rate. These results confirm that query settings are both application- and data- dependent.

5.2. Effectiveness of simple queries

These experiments compare the effectiveness of SnapFind with simple predicates against Blobworld using similar local features. Blobworld employs segmentation to decompose images into several regions, and describes each region with a summary of its color and texture statistics. We also examine the impact of allowing the SnapFind user to iteratively refine the search.

The test set was 800 images from ten categories of data from Corel (Arabian horses, auto racing, elephants, helicopters, lions, owls, polar bears, windsurfing, whitetail deer, and wolves). Within each category, we assigned the first three images to be query images. The complete list of the images in the dataset, and the blobs used for querying the image set, are available at `http://www.cs.cmu.edu/~dhoiem/obir`. Table 3 reports the results of these experiments.

For the Blobworld tests we performed three queries on each image category and computed the average precision for the top 10 and 50 images. The reported numbers were averaged over the three queries. We report two sets of numbers for Blobworld: the first is the results when using a fixed weighting of 0.5 for color and 1.0 for texture (labeled BW1); the second set varied the

Test	Metric	Avg	Arab. Horses	Auto Race	Eleph-ants	Helicop-ters	Lions	Owls	Polar Bears	Wind surf	W.Tail Deer	Wolves
BW	P(10)	44.3%	76.7%	73.3%	33.3%	20.0%	26.7%	86.7%	30.0%	6.7%	36.7%	50.0%
1	P(30)	39.4%	81.1%	57.8%	30.0%	8.9%	18.9%	80.0%	33.3%	11.1%	34.4%	38.9%
BW	P(10)	46.0%	86.7%	73.3%	46.7%	13.3%	30.0%	83.3%	30.0%	6.7%	40.0%	50.0%
2	P(30)	41.3%	83.3%	57.8%	35.6%	11.1%	18.9%	83.3%	35.6%	14.4%	34.4%	38.9%
SF	P(10)	42.7%	33.3%	70.0%	50.0%	13.3%	50.0%	30.0%	76.7%	23.3%	36.6%	43.3%
1	P(30)	31.7%	24.4%	60.0%	33.3%	10.0%	45.6%	21.1%	55.6%	17.8%	21.1%	27.8%
SF	P(10)	58.0%	80.0%	80.0%	40.0%	30.0%	70.0%	70.0%	70.0%	40.0%	50.0%	50.0%
2	P(30)	46.0%	50.0%	70.0%	43.3%	20.0%	56.7%	56.7%	53.3%	30.0%	43.3%	36.6%

Table 3. A comparison of SnapFind (SF) to Blobworld (BW) on several categories from Corel. BW1 uses a fixed weighting for the computed values while BW2 uses the best weighting for each object class. SF1 is without interactive refinement while SF2 provides the user with a 3-minute interactive refinement period. Three precision metrics are reported for each class.

weightings of the object features and reported the best results for each category (labeled BW2).

Using SnapFind we ran two different experiments. The first allowed the user to create predicates for color and texture using samples from one of the query images. In the second, the user interactively refines the initial query using the partial results returned during the search. For each query, the user was given three minutes to refine the query. Normally, SnapFind presents results to the user as they become available. To enable a direct comparison with Blobworld, we modified SnapFind to complete the search, and to rank its results by simply averaging the confidence of each of the predicates. More sophisticated methods of combining predicates would improve SnapFind's reported results for this task.

These results show that SnapFind with refinement achieves better average performance than Blobworld, validating our belief that user refinement should improve search results. Within each of the object classes, the results are more varied. The helicopter class is difficult, and neither algorithm performed well on that test; SnapFind's refinement step allowed the user to add more color distributions to match the different types of helicopters (military, coast guard, etc). On the lion class, SnapFind does better than Blobworld and we see that additional refinement further improves the search. On the owl class, Blobworld does significantly better than SnapFind. However, SnapFind results do improve with interactive query refinement. When SnapFind outperforms Blobworld, we believe it is because the user can focus on the distinctive features of the object (*e.g.* the texture of a lion's mane) instead of the overall characteristics of the object. For the cases where Blobworld does better than SnapFind, the images tend to be similar on global features that are better represented by Blobworld's more sophisticated color/texture model.

5.3. SnapFind Search Experience

These experiments explore SnapFind's effectiveness for searching large image collections. Here, one of the authors searched for pictures of his child in a red Halloween costume. There were 30 such images in the data set, and most of them were taken at night. These experiments used the same data set and hardware configurations as the first experiment.

The queries used three predicates: one to find the red costume, one to locate dark patches corresponding to night, and a face detector to find the child. The author performed several searches using different combinations of these predicates. Each query was executed twice, once with caching disabled, and once with a warm cache to illustrate the two extremes of Diamond performance. In real searches, where the user iteratively refines the search, caching should provide significant benefits. Table 4 shows results of these experiments.

The first row of the table estimates the time required to manually search the image collection and serves as a baseline measure as this is the only way to guarantee 100% recall. We obtained this estimate by measuring the number of images a user could classify in a 5 minute period, and extrapolating to the size of our data set. This estimate is optimistic because it assumes the user can maintain accuracy and a high rate of processing for an extended period.

The rest of the table shows results for different combinations of filters. We make several observations. First, since SnapFind displays partial results as soon as they are available, the user can examine displayed images in parallel with the Diamond search. Second, in the uncached cases, the total time is the same as the system time, indicating that the user is able to process images at the rate that they are delivered. In the cached cases,

	Red Similarity	Black Similarity	Face Detector Used?	Desired Images (Recall %)	Images Viewed by User	Uncached		Cached	
						Total Time (s)	System Time (s)	Total Time (s)	System Time (s)
Manual	–	–	N	30 (100%)	112,404	14,290	–	–	–
Red-only 1	0.70	–	N	28 (93.3%)	1,903	326	326	320	38
Red-only 2	0.90	–	N	19 (63.3%)	87	325	325	37	2
Red-Black 1	0.70	0.70	N	20 (66.7%)	721	327	327	106	15
Red-Black 2	0.90	0.70	N	13 (43.3%)	33	325	325	20	1
All	0.90	0.70	Y	11 (36.7%)	16	319	319	13	1

Table 4. Trade-off between recall and user's time. Searches used combinations of three predicates, several thresholds and with query caching enabled/disabled.

SnapFind displays images faster than the user can consume them. Third, to achieve a high recall the user was forced to manually classify 1,903 images. In this case, the user processing time was so high that caching made little difference.

In real-world searches (*e.g.*, surveillance, medical images), a user will spend more time processing retrieved results than in this simple test. We also believe that caching will provide some benefit, giving lower system times than in the uncached case. These observations lead us to conclude that interactive brute-force search is practical because Diamond can deliver search results faster than a user can consume them.

6. Conclusions

We believe that interactive brute-force search is most useful when the user places a high value on the retrieval results and is willing to invest a little time to improve search quality. This type of search is also appropriate when the image collection is being frequently modified, such as for surveillance applications, since precomputing image features may not be feasible.

Acknowledgments

We would like to thank M. Satyanarayanan and R. Wickremesinghe for their valuable contributions.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *Proc. of ASPLOS*, 1998.

[2] J. Bach et al. The Virage image search engine: an open framework for image management. In *Proc. SPIE*, 1996.

[3] C. Carson et al. Blobworld: Image segmentation using expectation-maximization and its application to image querying. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(8), 2002.

[4] D. Cohn, Z. Ghahramani, and M. Jordan. Active learning with statistical models. In *Proc. of Neural Information Processing Systems*, 1996.

[5] I. Cox et al. The Bayesian image retrieval system, PicHunter: Theory, implementation and psychophysical experiments. *IEEE Trans. on Image Processing*, 9(1), 2000.

[6] M. Flickner et al. Query by image and video content: the QBIC system. *IEEE Computer*, 28, 1995.

[7] J. Hellerstein et al. Interactive data analysis: The CONTROL project. *IEEE Computer*, August 1999.

[8] D. Hoiem et al. Object-based image retrieval using the statistics of images. In *Proc. of IEEE Computer Vision and Pattern Recognition*, 2004.

[9] L. Huston et al. Diamond: A storage architecture for early discard in interactive search. In *Proc. USENIX Conference on File and Storage Technologies*, 2004.

[10] R. Lienhart, A. Kuranov, and V. Pisarevsky. Analysis of detection cascades of boosted classifiers for rapid object detection. In *Proc. of DAGM Pattern Recognition Symposium*, 2003.

[11] T. Minka and R. Picard. Interactive learning using a society of models. *Pattern Recognition*, 30, 1997.

[12] E. Riedel, G. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *Proc. of VLDB*, August 1998.

[13] A. Smeulders and M. Worring. Content-based image retrieval at the end of the early years. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(12), 2000.

[14] J. Smith and S.-F. Chang. VisualSEEk: A fully automated content-based image query system. In *Proc. of ACM Multimedia*, 1996.

[15] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. of IEEE Computer Vision and Pattern Recognition*, 2001.

[16] J. Wang. *Integrated Region-Based Image Retrieval*. Kluwer Academic Publishers, 2001.

[17] J. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics-sensitive integrated matching for picture LIbraries. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(9), 2001.