ADAPTIVE FILTERING AND INDEXING FOR IMAGE DATABASES [1]

A. D. Alexandrov† **W. Y. Ma**‡ **A. El Abbadi**† **B. S. Manjunath**‡
†Department of Computer Science
‡Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106

ABSTRACT

In this paper we combine image feature extraction with indexing techniques for efficient retrieval in large texture images databases. A 2D image signal is processed using a set of Gabor filters to derive a 120 component feature vector representing the image. The feature components are ordered based on the relative importance in characterizing a given texture pattern, and this facilitates the development of efficient indexing mechanisms. We propose three different sets of indexing features based on the "best feature", the average feature and a combination of both. We investigate the tradeoff between accuracy and discriminating power using these different indexing approaches, and conclude that the combination of "best feature" and the average feature gives the best results.

Keywords: Texture Images, Gabor Filters, Multi-dimensional Indexing, Indexing Structures.

1 INTRODUCTION

Searching a large database of image data poses many challenging problems. In particular, issues related to image feature extraction, defining similarity measures, and developing suitable search mechanisms need to be addressed. While there are several papers in the literature that address some of these issues, they tend to be one of the following two categories: (a) research which address primarily the image feature detection and pattern recognition with little regard to efficient search (most of these papers assume sequential search), or (b) database related papers which address the problem of multi-dimensional indexing, often making unrealistic assumptions about the nature of the data/features. This paper describes results of our preliminary work on bridging this gap between image processing and database as we systematically explore the potential of several indexing mechanisms in developing efficient architectures for image retrieval.

The main distinction between pattern recognition and searching in image databases is that while exact pattern matching is the main criterion in pattern recognition, in database retrieval the objective is to help the user navigate through a large number of images. In fact, the user may not even be interested in the best match! S/he might use the retrieved patterns to further modify the search specifications. Thus, while the constraints on pattern recognition is somewhat relaxed in a database environment, efficiency (in terms of reducing the retrieval time) and robustness (in terms of not missing target database patterns) is of much concern. We believe that by combining image processing with database search in an intelligent way one can achieve these objectives.

Recent research on image data retrieval can be broadly classified into three categories based on the level of image feature analysis. At the low level, which requires no domain specific or context information, features could include color, histogram, and texture information. At the intermediate level shape primitives are often used. Preprocessing here requires robust segmentation to detect region boundaries. At the highest level, domain specific information could be used in applications such as face recognition and satellite image data retrieval. An intelligent visual database system will have all three components of visual processing in addition to textual annotations and non-image specific information integrated into the query system.[1,10,12]

[1] This work was partially supported by the NSF under grant number IRI-9411330, and by the NASA under grant numbers NAGW-3888 and NAGW-3951

In order to focus the issues, we consider a texture image database. Texture information is one of the basic cues on which patterns can be retrieved. In the image analysis literature, texture classification and segmentation continues to be an active research area. In texture image analysis, previous approaches to classification include those based on Markov random field models, cooccurrence matrices, the Wold model, and the wavelet based methods. In general, algorithms based on multichannel analysis have been quite promising and have the advantage that the associated feature information can be systematically organized. However, the main problem with such multiresolution analysis is the large dimensionality of the feature vectors.

In this paper, we start by proposing a three layered framework for indexing image databases in Section 2. Most previous approaches to indexing image databases are based on extracting a **single** set of features from the images. This set of features is relatively small and is used for both defining the similarity measure and for indexing. In our case, the similarity measure is based on 120 features and hence it is not feasible to index directly on this set of features. For this reason we introduce a new indexing specific layer that is based on a smaller set of *indexing features* extracted from the set of 120 features. In the same section we study the performance of a representative indexing data structure, the R^*-tree.[2] In Section 3, we describe the feature extraction mechanism, the similarity measure and an adaptive filter ordering to order the feature components in the feature space. Section 4 presents our main results. We describe three different indexing approaches based on the "best filter" as defined by the feature ordering algorithm, on the average of the feature values and on a combination of the two. Section 5 concludes the paper.

2 MULTI-DIMENSIONAL INDEXING

2.1 The Indexing Framework

The main goal of indexing is to reduce search time compared to sequential searching. Until recently image databases were relatively small and therefore sequential search was acceptable. However recent applications e.g. digital libraries of images, require more sophisticated index structures and search strategies if they are going to be used in scientific and commercial applications. Here we describe a general framework for indexing in image databases. The framework is hierarchical and includes three components: *Image data*, *Feature level representation*, and *Indexing structure* (Figure 1). Although we utilize all three layers, existing approaches tends to combine the top two layers.[4,16]

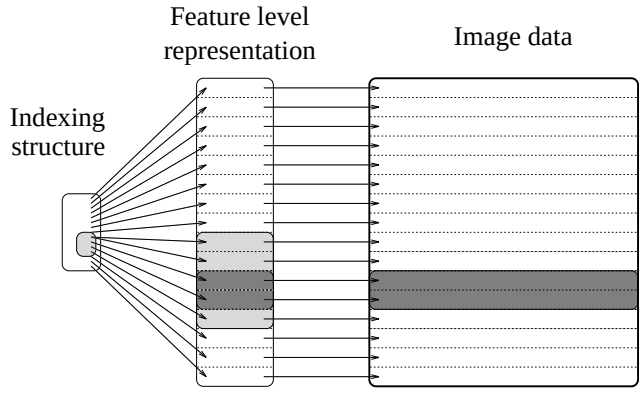

Figure 1: Indexing in an image database. The lighter shaded areas is the reduced search space, the darker shaded area is the final set of images.

The lowest level is the image data. Here each image is stored in its complete form e.g. as a matrix of pixel intensities. The data in this layer does not participate directly in the search process. It is only used for extraction of the *features* for the second layer and for user interface. The second layer is the feature representation. For

each image a feature vector is extracted and stored here. The feature vector is much smaller in size than the complete representation of the image but, as in our case, can still be of substantial size. The main purpose of this layer is for defining and computing the *similarity measure* between images. If no indexing is done, only the bottom two layers are used. In this case searching is done by sequentially going through each feature vector and computing its distance to the search image's feature vector. The result of the search is the k closest feature vectors. The actual k images corresponding to these feature vectors are returned to the user. Such a feature level representation facilitates the development of comparison measures to distinguish image patterns. Furthermore, compared to measuring image similarity directly on the raw image data, computational costs are reduced since features are significantly smaller than the original image.

The purpose of the indexing layer is to reduce the number of feature vectors that the second layer examines. In this layer, each image is represented very concisely using a small number of *indexing features*, which are used during the search process to eliminate unlikely matches. It passes to the second layer the set of (the identifiers of) the images that are likely matches. We refer to this set as the reduced search space and denote it with R_S, where S stands for the set of all images. Now the second layer has to examine and compute the similarity distance only to the feature vectors in R_S. This is done in $\frac{|R_S|}{|S|}\%$ of the time needed for sequential search. One can easily see that the smaller the size of R_S, the faster the search is. We define the ratio $\frac{|R_S|}{|S|}$ as the *discriminating power*. A discriminating power of 20%, for example, means that the similarity computation in the second layer will be done five times faster than without indexing. The smaller the percentage, the greater the discriminating power. The discriminating power alone, however, is not enough to determine the speed-up provided by the indexing method. The reason is that the indexing structure itself introduces some overhead. Therefore the second important characteristic of an indexing structure is its efficiency. We are interested in how fast the indexing method can produce the reduced search set, R_S, for a fixed value of the discriminating power $\frac{|R_S|}{|S|}$. The most naive indexing structure would be to use no indexing at all, i.e., perform a sequential search. We estimate efficiency of an indexing method by comparing its speed to this naive method.

In this paper we are mainly concerned with producing indexing methods with high discriminating power. In the beginning, though, we examine the efficiency of multidimensional indexing structures in order to determine what kind of indexing methods will really lead to fast searching. We would like to avoid situations where the indexing has high discriminating power but its efficiency is so poor that the indexing part takes more time than the search would take without using indexing. As we will show in section 2.3, the quest for higher discriminating power can potentially lead to a such a situation.

2.2 The indexing structure

An important observation is that even if sequential search is used for the indexing "structure" there still can be a significant speed-up of the whole image retrieval process. This happens when the indexing data structure can be made much smaller (by an order of an magnitude in our case) than the feature level representation. Thus the search time in the indexing structure is only a small portion of the whole search process, with the feature level sequential search in the reduced search space taking most of the it. On the other hand if the indexing structure is relatively big (when many indexing features are used to obtain high discriminating power) the efficiency of the indexing structure becomes an issue.

Our goal in this paper is not to develop a new indexing structure but rather to efficiently use the existing ones. If our indexing method is based on a single indexing feature only, then we can use one of the many one-dimensional data structures, like the B-trees,[5] which are studied extensively and are known to provide excellent efficiency. Any one-dimensional indexing method should easily provide the speed-up suggested by its discrimination power. In image databases due to the nature of the data most often more than one indexing features are used. There are several data structures that are suitable for indexing multi-dimensional data. Examples of these are the variants of the R-tree,[2,7,8] the k-d-B tree,[13] the Grid file,[11] etc. All these data structures have reasonable performance for a small number of dimensions, but for a big number of dimensions their performance deteriorates, a phenomenon known as the "dimensionality curse".

We next analyze the performance of the R^*-tree as a representative of spatial data structures. In particular, we are interested in how its efficiency deteriorates with increases in the number of dimensions.

2.3 Evaluating the R^*-tree

The R^*-tree is a relatively new member of the R-tree family. The R-trees are a class of multidimensional data structures. They are a hierarchical collection of d-dimensional rectangles. Similarly to B-trees, the leaf nodes (buckets) appear at the same level. The leaves contain the actual data. Each data object is stored together with its minimum enclosing rectangle. Each bucket is associated with a bounding rectangle that encloses the smaller rectangles of its children. The grouping into bigger rectangles continues up the tree until at the top a single all encompassing bounding rectangle, the root, is left. A range search in an R-tree is executed by recursively traversing all the buckets whose bounding rectangles intersect with the search region (usually a d-dimensional rectangle).

In the first experiment we compare the performance of the R^*-tree for various dimensions to the performance of the sequential search. Since the number of disk accesses determines the search time, our measure for performance was the number of disk reads. We used a set of 1000 uniformly distributed d-dimensional points and varied the number of dimensions. For each number of dimensions we performed 2000 random range searches with a fixed volume and averaged the number of disk reads performed. In Figure 2(a) we show three curves corresponding to search volumes of 10%, 1% and 0.1% of the total space. The performance of the R^*-tree is shown relative to the performance of the sequential search algorithm. The number disk reads of the latter is independent of the search volume and is denoted with 100%.

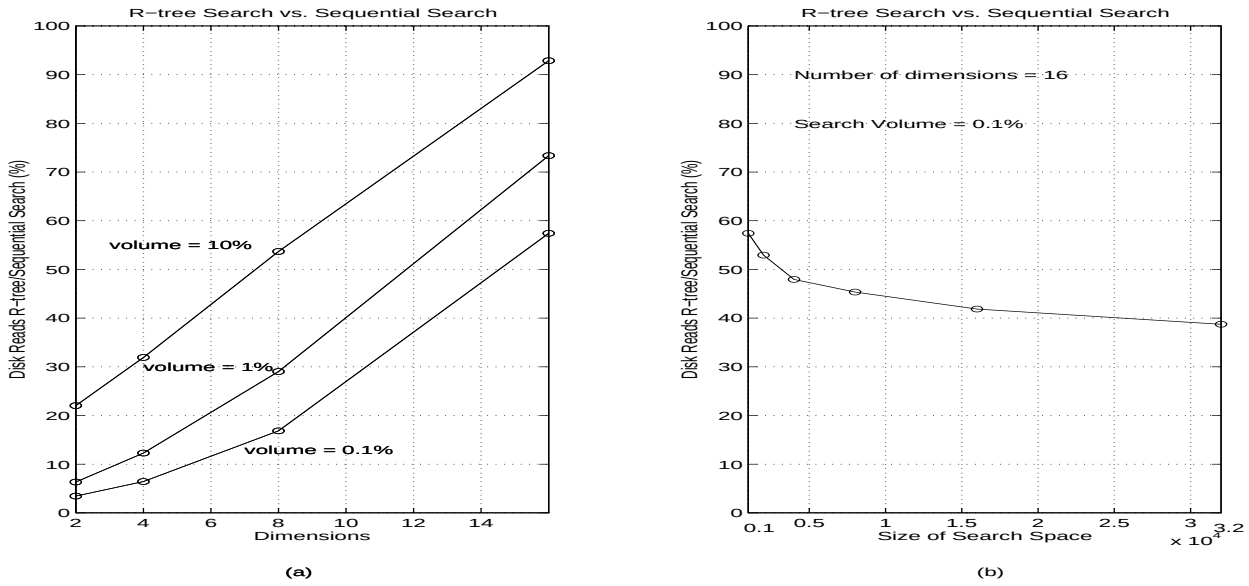

Figure 2: Performance of the R^*-tree: (a) different number of dimensions; (b) for different data sizes.

We now make two observations: First, for the given search volumes, the R^*-tree performs better than the sequential search. Second, the performance of the R^*-tree degenerates quickly with the increase of the number of dimensions. In particular in 16 dimensions a search volume of 0.1% translates into a number of disk reads that is 58% of the disk reads needed for sequential search. Ideally an indexing method would translate a range search with search volume of x% into x% of the disk reads needed for sequential search. This, of course, is unattainable due to the fact that indexing introduces some overhead in terms of additional disk memory. Observe that for 16 dimensions and a search volume of 0.1% the R^*-tree is 580 times worse than the ideal. In contrast with 2 dimensions and a 10% search volume, the tree is only about 2 times less efficient than the ideal.

Since the relative performance of indexing structures generally increases with the size of the data set, we performed a second experiment to evaluate the rate of the increase of performance of the R^*-tree. We repeated the previous tests, but fixed the number of dimensions to 16, and the search volume to 0.1%. The results for sizes of the data set between 1,000 and 32,000 points are shown in Figure 2(b). Indeed the performance improves as the number of points in the tree increases but the improvement is very slow. Even for 32 thousand points the R^*-tree performs 380 times worse than the ideal.

We already noted that in order to achieve higher search speed, the indexing structure should provide high discriminating power. The results for the R^*-tree, though, indicate that increasing the number of dimensions by using more indexing features may be counterproductive. While for a small number of dimensions the indexing structure is efficient and relatively small in size, as the number of dimensions increases the indexing structure grows in size and becomes inefficient. Both of these eventually lead to unacceptably slow indexing. The overhead for indexing may become too large compared to the similarity search in the second layer and thus eliminate the advantages of the smaller search space. The only way to ensure efficient indexing for a large number of dimensions is to use indexing features with extremely high discriminating power. As we can see for 16 dimensions even 0.1% percent is not small enough. In our experience with indexing image in image databases, independent of the number of indexing features used, even a discriminating power of 1% is very difficult to obtain.

From this analysis of the R^*-tree, we make the conclusion that although high discriminating power is very important for fast searching, increasing the discriminating power should be done by increasing indexing dimensionality only as a last resort. Therefore in the remaining part of this paper we will concentrate on obtaining high discriminating power for a small number of dimensions - one or two.

3 ADAPTIVE TEXTURE FEATURE EXTRACTION

3.1 Texture Features

A typical query in our case is a region of interest provided by the user (for example, outlining a region from an image such as a vegetation patch in a satellite photograph, or a tumor in a brain MR scan). The information is thus implicit in the intensity pattern within this region (which is usually a rectangular patch). Image intensities usually range between 0-255, and even for modest image dimensions (e.g., 128 x 128 pixels) this forms a very large dimensional image space to search for. In transforming this intensity pattern, one is therefore interested in transformations which reduce the dimensionality of the space while preserving the similarity between patterns in the feature space. Our approach to this problem is based on image filtering using Gabor filters. In the following, we summarize the basic steps and we refer to[9] for complete details.

Gabor filters are often used in modeling orientation selective and spatial frequency selective receptive field properties of cortical cells. They have been shown to be optimal in the sense of minimizing the joint two-dimensional uncertainty in space and frequency.[6] This localization property has been found to be very useful in many image analysis problems such as texture classification and segmentation, and human face recognition.

Gabor functions are Gaussians modulated by complex sinusoids. The particular form of the filters that are used in our experiments is given by:

$$G_{m\theta}(x,y) = a^{-m} g(x',y') \exp\left[2\pi j W x'\right] \tag{1}$$

where $a = 1.18$, $m = \{0,1,2,\ldots,9\}$, $W = 0.45$, and

$$g(x,y) = \left(\frac{\lambda}{2\pi\sigma^2}\right) \exp(-\frac{\lambda^2 x^2 + y^2}{2\sigma^2}) \tag{2}$$

$$x' = a^{-m}(x\cos\theta + y\sin\theta), y' = a^{-m}(-x\sin\theta + y\cos\theta)$$

where $\lambda = 2$ is the spatial aspect ratio, $\sigma = 2.5$ is the standard deviation of the Gaussian, and θ is the rotation angle of the major axis of Gaussian. We use twelve orientations, i.e., $\theta = \{0, 15°, 30°, \ldots, 165°\}$. Our past

experience in using these filters suggests that the number of scales and orientations used (a total of 120 different filters) is adequate to capture the relevant texture feature information.

Processing an image pattern $I(x, y)$ with a filter $G_{m\theta}(x, y)$ results in the output

$$W_{m\theta}(x, y) = \int I(x_1, y_1) G^*_{m\theta}(x - x_1, y - y_1) dx_1 dy_1 \tag{3}$$

where * indicates complex conjugate. Note that we are using ten different scales and twelve orientations at each scale, resulting in a total of 120 filters. In our representation, we are interested only in the associated energy of the filtered outputs. The image pattern is then characterized by the mean $f_{mean}(m, \theta)$ and standard deviation $f_{std}(m, \theta)$ of the energy, averaged over all the pixel locations in $W_{m\theta}(x, y)$. Thus, processing a texture pattern with the whole set of filters results in a 120 x 2 dimensional feature vector. As we will see below, only the mean feature vector is used in the indexing structure and the complete feature vector is used only during the final stages to order the retrieved database patterns based on their similarity which is as defined below.

Similarity Measure: Let us consider two image patterns i and j, and let $f^{(i)}(m, \theta)$ represent the corresponding feature for the i^{th} pattern. Then we define a normalized distance between the two patterns in the feature space (m, θ) as:

$$d_{i,j}(m, \theta) = \left| \frac{f^{(i)}_{mean}(m, \theta) - f^{(j)}_{mean}(m, \theta)}{\sigma_{mean}(m, \theta)} \frac{f^{(i)}_{std}(m, \theta) - f^{(j)}_{std}(m, \theta)}{\sigma_{std}(m, \theta)} \right| \tag{4}$$

where $\sigma_{mean}(m, \theta)$ and $\sigma_{std}(m, \theta)$ are the standard deviations of the distribution of features $f_{mean}(m, \theta)$ and $f_{std}(m, \theta)$, respectively, in the image database. Note that the smaller the value $d_{i,j}(m, \theta)$ is, the more similar are the two feature components. It reaches the minimum value 0 when $f^{(i)}_{mean}(m, \theta) = f^{(j)}_{mean}(m, \theta)$ and $f^{(i)}_{std}(m, \theta) = f^{(j)}_{std}(m, \theta)$.

While the above feature measures provide sufficient discrimination to achieve close to 100% classification on a large texture database, they are not convenient to develop an efficient indexing mechanism. In order to facilitate this indexing, we suggest in the next section a rank-ordering of the filters based on their importance in discriminating individual patterns in the database.

3.2 Adaptive Filter Selection

We have developed a scheme[9] for determining the relative importance of the Gabor filters according to their potential discriminating power for a given search texture pattern. Although originally it was developed to minimize the image processing computations, it was realized soon that it plays an important role in indexing efficiently by (adaptively) defining a natural order on the feature space. The filter ordering is pattern dependent and is based on the average spectral information of all the images in the database. The difference between the spectrum of the input image and the average spectrum, appropriately normalized, provides information about the salient spectral characteristics of the given image. This is used to identify the *best*, the *second best*, and so on, discriminating filters from the given dictionary of (120 in our case) Gabor filters. The filter ordering will be used in our indexing methods to achieve higher discriminating power with better accuracy. This is done by selecting the best filter to index on, depending on the search image pattern.

4 INDEXING PERFORMANCE EVALUATION

In the following sections we examine several indexing approaches and evaluate the relationship between increasing speed and decreasing accuracy. In particular, we explore how much loss of accuracy can be tolerated while obtaining a certain speed-up in the search process.

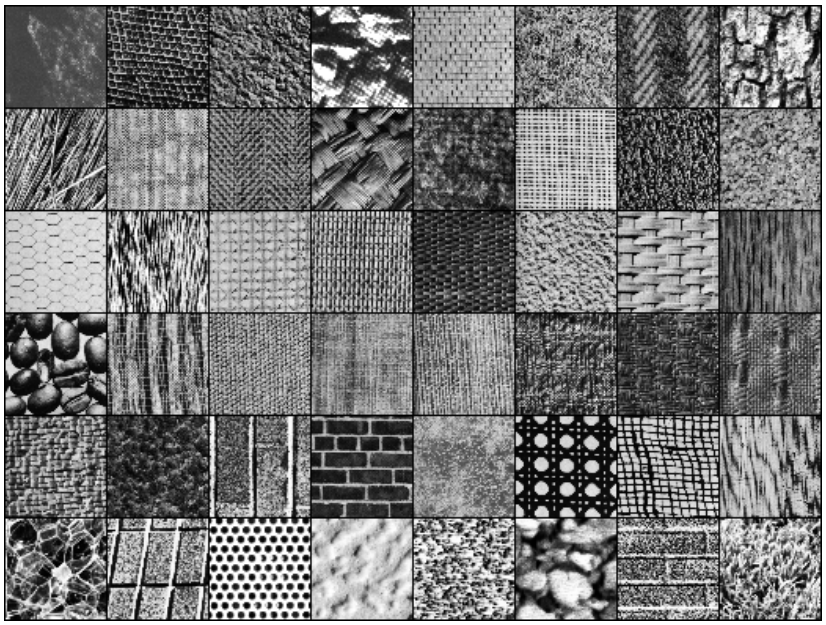

Figure 3: 48 patterns from the Brodatz texture image database.

4.1 The image database testbed

For the tests that follow we use 56 images of different textures, 48 of which are taken from the Brodatz album[3] (Figure 3). Each of the images is divided into 16 rectangular regions, providing 16 sample. This gives us a total of 896 different samples that form our image database. We denote the individual samples with s_i and the set of all samples with S. We determined that the mean part of the feature vectors, f_{mean}, is by far more important for indexing than the standard deviation, f_{std}. Therefore we base our indexing methods only on the f_{mean} portion of the feature vectors. In the different experiments conducted, we estimate the discriminating power of the features and the accuracy corresponding to different speed-ups. These statistics are gathered by executing one search for each of the 896 samples in the database and then averaging the results. Each search consists of using the indexing method to reduce the search space, S, by eliminating unlikely close matches. The analysis is based on the size and contents of the reduced search space, R_S. The whole search process consists of two parts. First using the indexing method, the search space is reduced from S to R_S. After this, the elements of R_S are examined sequentially and the distance to the search sample is computed using the similarity measure. The closest k images from R_S are returned by the search. We only examine the indexing part of the search, that is we stop before the similarity matching step. The discriminating power of the indexing method, $\frac{|R_S|}{|S|}$, is our measure for speed-up. To measure accuracy we count the number of correct hits. Correct hits are the samples in the reduced search space that come from the same image as the search sample.

4.2 Using one filter for indexing

In this section we examine the indexing quality of individual filters. The idea is to reduce the search space based on only one of the 120 fields of the feature vector. This is done by retaining only a certain percent of the closest samples, with respect to the indexing filter, and eliminating the rest.

First we focus on the fact that for different images different filters are appropriate for indexing. In the following experiment, for each sample in the database, s_i, an indexing filter $F(s_i)$ is determined which is used to reduce the search space by eliminating the entries in the database that differ too much from s_i in their value for $F(s_i)$.

In choosing $F(s_i)$ for each sample s_i, we will rely on the importance ordering (Section 3.2) of the filters for this specific sample.

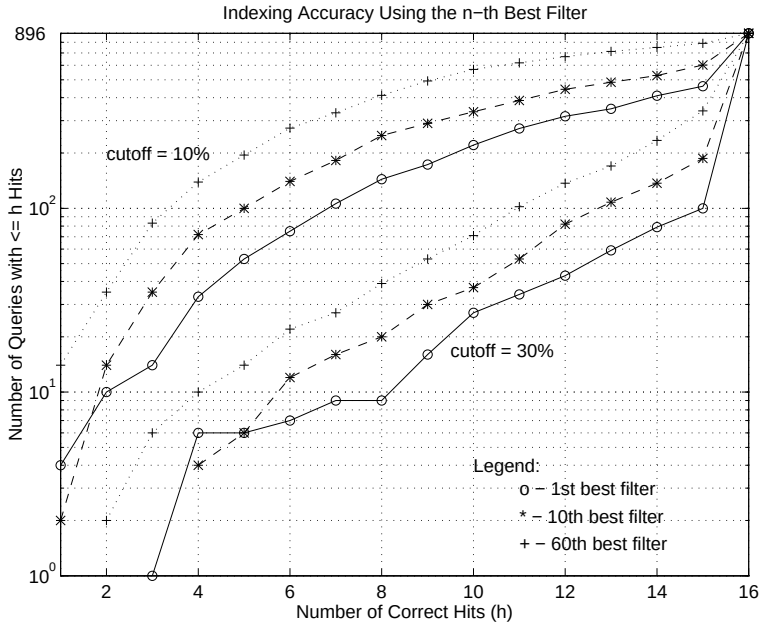

Figure 4: Quality of filters with different importance. The y-axis is in log_{10} scale.

In three different experiments $F(s_i)$ is chosen to be the n-th most important filter for s_i, for $n = 1, 10, 60$. Note that different samples will have different indexing filters since $F(s_i)$ depends on the sample. Given the search sample, s_i, we examine the value of the $F(s_i)$ filter for each of the 896 sample in the database and retain only the *cutoff%* samples, closest to s_i (with respect to $F(s_i)$). Denote the set of the retained samples (the reduced search space) with $R_S(s_i)$. Note that this guarantees that the size of $R_S(s_i)$ is *cutoff%* of the size of S. Hence, the cutoff percent exactly determines the discriminating power of the indexing feature. To measure the accuracy of the method, we use the number of correct hits (the number of samples in $R_S(s_i)$ that come from the same image as s_i). Ideally all 16 samples are in $R_S(s_i)$. In the worst case only one sample, s_i itself, is in $R_S(s_i)$. The results are shown in Figure 4. Each curve corresponds to a fixed cutoff percent (10% or 30% corresponding to sample sizes of 90 and 267) and a fixed n (1, 10 or 60). The interpretation of the plot is as follows: consider point (2, 10) from the curve with *cutoff* = 10% and $n = 1$. The point indicates that out of the 896 searches there were 10 cases in which 2 or less correct hits occurred. In general a lower curve corresponds to higher accuracy (smaller number of "bad" hits). All the curves meet at (16, 896) since the number of samples from one image is 16 and therefore no search can return more than 16 hits.

The graph confirms that ordering of the filters is good, meaning that the more important filters produce higher accuracy when used for indexing. In particular, any filter other than the best for the particular image under consideration will provide worse performance. Hence, no single filter is appropriate for all images, rather, for each image a particular feature is best for indexing purposes.

4.3 Using the best filter for indexing

In the previous section we established that choosing the most important filter provides the highest level of accuracy in discrimination. In this section we examine more closely the performance of the best filter for indexing for a wider range of cutoff values. We repeated the experiment from the previous section using the most important

filter ($n = 1$). In Figure 5, the different curves correspond to different cutoff percentages (5% through 50%). Two of the curves representing 10% and 30% cutoff are the same as in Figure 4.

It is clear that we can arbitrarily increase the speed of search by decreasing the cutoff percentage (i.e. increasing discriminating power). The purpose of this graph is to determine the maximum discriminating power that can be achieved while maintaining a desired level of accuracy. If we know the amount of accuracy necessary, the graph can be used as follows: assume that our accuracy specification requires that there be at least 7 samples from the same image (correct hits) in $R_S(s_i)$ in 99% of the searches, then from the six curves only three satisfy our requirements. These are the ones for 30, 40 and 50 percent cutoff. The reasoning for the 30% curve, for example, is that the number of searches which return six or less good hits is seven (point (6, 7) of the curve). This is 0.78% of all cases. Therefore in the remaining 99.22% of the cases the searches return more than 6 sample from the same image. The percentage for the 40% curve is 99.44% (5 bad cases out of 896). The 20% curve does not qualify with 98.44% of searches satisfying the requirement (14 bad cases out of 896). On the other hand if the requirement was to have at least 5 samples from the same pattern in 95% of the cases even the 10% curve qualifies with 96.34% (33 bad search cases). The 5% curve is still not accurate enough with 85.83% (127 bad searches).

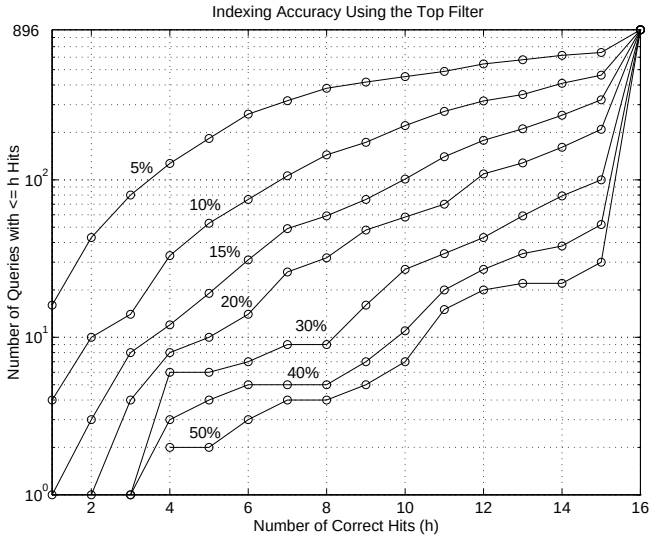

Figure 5: Indexing quality of the best filter for the search image. The y-axis is in log scale.

The important question as to the acceptable accuracy is application dependent and is difficult to answer. First of all, the level of accuracy achieved for a cutoff of 50% is excellent and should be acceptable in all cases. In our experiment only 2 searches (0.2%) retrieved less than 4 samples from the same image and all but 7 searches (0.8%) preserved more than 10 samples from the same image in the reduced search space. ¿From Figure 5, we observe that the best tradeoff between speedup and accuracy is obtained at approximately the 15% cutoff. In particular, note that the transition from 10% to 15% is quite significant in that the number of queries with fewer hits is considerably reduced while not substantially increasing the search space.

4.4 Using the average feature for indexing

From the previous section we can deduce that indexing using the best filter on a per image basis will give good results. Since each of the 120 filters could be the best one for some search image, this approach requires the use of 120 different index structures, one per filter. In this section we explore an alternative approach that uses a single indexing feature and therefore requires only one indexing structure. This feature, denoted AVG, is the average value of the 120 components of feature vector. It is computed for each sample in the database and is used to build the index structure.

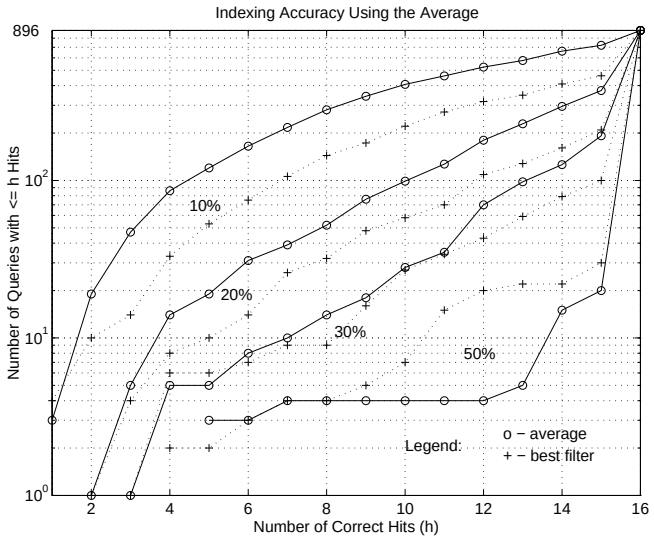

Figure 6: Indexing quality of the average of all filter values. The y-axis is in log scale

The experiment we carried out is the same as in the previous section except that here we use AVG instead of the best filter to reduce the image space by eliminating samples whose AVG value differs too much from the search sample's value. The results are shown in Figure 6. For comparison the curves for the best filter are added. It is interesting to observe that AVG provides a better indexing feature for big cutoff values, 40% and more, but the most important filter is more accurate for smaller cutoff values. This indicates that when extremely high accuracy and small speed-up is needed, the average should be preferred for indexing, for high speed-up and smaller accuracy the best filter is more appropriate.

Although using AVG for indexing is important on its own, we obtain our most important result by combining it with the best filter.

4.5 Using "best feature" with the average feature

In this section we analyze how the combination of two indexing features affects the accuracy and discriminating power of indexing. The general framework of this approach is to reduce the search space by elimination entries that differ too much in either of the two indexing features. The conclusion from the previous sections was that reducing the search space by narrowing the cutoff window decreases accuracy significantly. By adding a second indexing feature we can reduce the search space while increasing accuracy. The two indexing features that we use are the best filter and the average, AVG.

We present the results of an experiment that is similar to those in the previous three sub-sections. The difference is that here we use two cutoff values $cutoff_F$ and $cutoff_{AVG}$ for the best filter and for the average respectively. Let $R_S^F(s_i)$ and $R_S^{AVG}(s_i)$ be the reduced search space when using the best filter only and the average only respectively. The combined reduced search space is $R_S^{both}(s_i) = R_S^F(s_i) \cap R_S^{AVG}(s_i)$. In practice this indexing cutoff method is equivalent to retaining all samples that fall within a rectangle in 2-D space with the two axes being the best filter and the AVG value for s_i. Note that when using two indexing features we cannot calculate *a priori* the size of $R_S^{both}(s_i)$ although we know that the sizes of $R_S^F(s_i)$ and $R_S^{AVG}(s_i)$ are $cutoff_F\%$ and $cutoff_{AVG}\%$ of the size of the database (896 in our case). Instead we have to measure it experimentally. We do this by counting the number of samples in each $R_S^{both}(s_i)$ for all 896 samples s_i and than calculating their average. In Figure 7, the average size of the reduced search space,$R_S^{both}(s_i)$, is given for each of the different

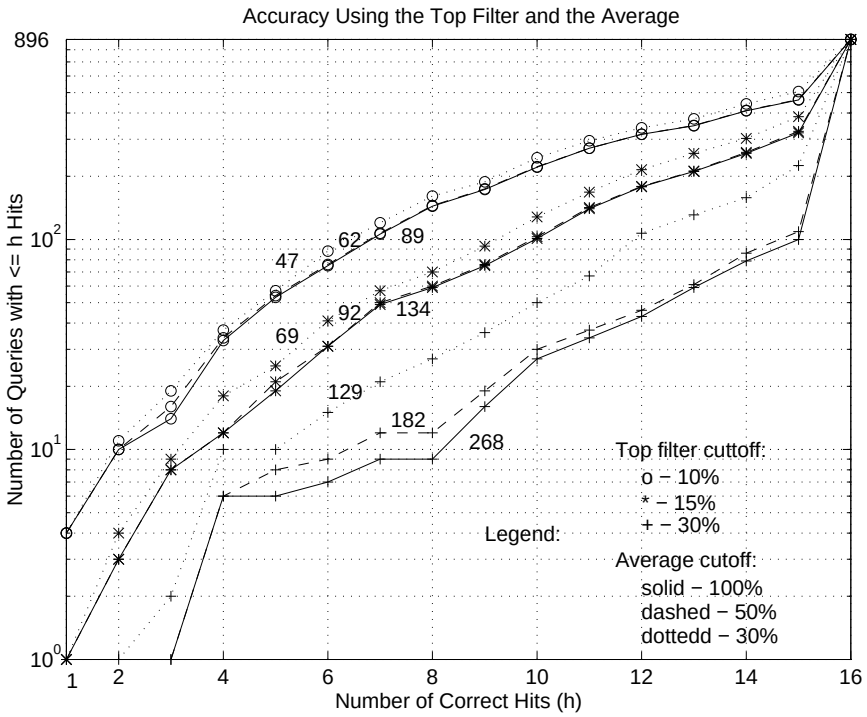

Figure 7: Indexing quality of the best filter combined with average of all filter values for the search image.

curves. The nine curves shown correspond to the combinations of 10%, 15% and 30% for $cutoff_F$ and 30%, 50%, 100% for $cutoff_{AVG}$. For $cutoff_{AVG} = 100\%$ the method reduces to indexing with the best filter alone (the three solid lines). Consider the three curves for $cutoff_F = 30\%$. As the cutoff for the average decreases from 100% to 50% and 30% the size of the search space decreases from 268 (exactly 30% of all the samples) to 182 (about 20%) and to 129 (or 14.4%). Naturally, the accuracy decreases at the same time.

The important result, though, is seen when the last curve (30%, 30%) is compared to the curve for (15%, 100%). The latter in fact corresponds to indexing using the best filter alone with a cutoff of 15%. The plot shows that by using the two indexing features we were able to achieve better accuracy by a factor of about 2, while maintaining the size of the reduced space (in fact reducing it slightly from 134 to 129). The benefit of adding a second indexing feature is even bigger for the the cases where $cutoff_F$ is 15% or less. Compared to using the best filter alone, the use of $cutoff_{AVG} = 30\%$ reduces the search space by a factor of almost two while maintaining the accuracy virtually unchanged. For example, the search was cut from 134 to 69 for a $cutoff_F = 15\%$, while maintaining accuracy almost unchanged.

Similarly to using the best filter alone, the two-index features approach requires the use of 120 index structures, each of which is two-dimensional and is based on one of the 120 filters and the average value. As we discussed in Section 2 the 2-D structure will be less efficient than an one-dimensional index structure. Fortunately the size of the individual index structures will be small enough (we estimate it at less than 5% of the disk space for the feature representation of the image database) so that the speed-up effect obtained by reducing the search space almost in half is not lost.

5 CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a framework for indexing texture image databases. A large dimensional feature vector is extracted from the raw image data, and the feature components are ordered for every image in terms of their relative discriminating power. This ordering, in combination with the average feature value is used to index the database. Our experiments show that this approach results in high discrimination while maintaining reasonable accuracy. We are currently implementing the actual index structures on the texture image database.

6 REFERENCES

[1] J. R. Bach, S. Paul, and R. Jain, "A visual information management system for the interactive retrieval of faces," IEEE Trans. Knowledge and Data Engineering, vol. 5, No. 4, pp. 619-628, August 1993.

[2] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger, "The R^*-tree: An Efficient Robust Access Method for Points and Rectangles," 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, 23-25 May 1990.

[3] P. Brodatz, "Textures: A Photographic Album for Artists & Designers," New York: Dover, New York, 1966.

[4] T.-C. Chieuh, "Content-Based Image Indexing," Proc. Intl. Conf. on Very Large Data Bases,September 1994.

[5] D. Comer,"The Ubiquitous B-tree," ACM Computing Surveys 11,2(June 1979), 121-137.

[6] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," in J. Opt. Soc. Am. A, vol. 2, pp. 1160-1169, 1985.

[7] D. Greene. "An Implementation and Performance Analysis of Spatial Data Access Methods," Proceedings of the 5th International Conference on Data Engineering, 606-615, 1989.

[8] A. Guttman, "R-trees: a Dynamic Index Structure for Spatial Searching," in Proceedings of the SIGMOD Conference, Boston, June 1984, 47-57.

[9] W. Y. Ma, B. S. Manjunath, "Pictorial Queries: Combining Feature Extraction with Database Search," in Technical Report CIPR 94-18, Univ. of California at Santa Barbara, Nov. 1994.

[10] W. Niblack et al, "The QBIC project: Querying Images by Content Using Color, Texture, and Shape," in SPIE 1908, Storage and Retrieval for Image and Video Database, Feb. 1993.

[11] J. Nievergelt, H. Hinterberger, and K. C. Sevcik,"The Grid File: An Adaptable Symmetric Multikey File Structure," in ACM Transactions on Database Systems 9, 1, (March 1984), 38-71.

[12] A. Pentland, R. W. Picard, and S. Sclaroff, "Photobook: tools for content based manipulation of image databases," in proc. SPIE conference on storage and retrieval of image and video databases -II, Vol. 2185, pp. 34-47, San Jose, February 1994.

[13] J. T. Robinson, "The k-d-B-tree: a Search Structure for Large Multidimensional Dynamic Indexes," in Proceedings of the SIGMOD Conference, Ann Arbor, MI, Apr 1981, 10-18.

[14] T. Sellis, N. Roussopoulos, and C. Faloutsos. "The R^+-tree: a Dynamic Index for Multi-Dimensional Objects," Computer Science TR-1795, University of Maryland, College Park, MD, February 1987.

[15] M. Stonebraker, T. Sellis, and E. Hanson, "An Analysis of Rule Indexing Implementations in Data Base Systems," in Proceedings of the First International Conference on Expert Database Systems, Charleston, SC, April 1986, 353-364.

[16] N. Yazdani, M. Ozsoyoglu, G. Ozsoyoglu, "A framework for feature-based indexing for spatial databases," Proceedings. Seventh International Working Conference on Scientific and Statistical Database Management, p. 259-69, Sept. 1994.